

Metodología de la programación orientada a objetos **2da. Edición**

LEOBARDO LÓPEZ ROMÁN



libroweb

 Alfaomega



Metodología de la programación orientada a objetos

Segunda Edición

Leobardo López Román

 Alfaomega

López Román, Leobardo
Metodología de la programación Orientada a Objetos. - 2da ed.
564 p. ; 21x24 cm.

ISBN 978-607-707-589-9

1. Informática. I. Título
CDD 005.3

Metodología de la programación Orientada a Objetos - 2da. Ed.

Leobardo López Román

Derechos reservados © Alfaomega Grupo Editor, S.A. de C. V., México
2da. Edición: Alfaomega Grupo Editor, México, 2013

© 2013 Alfaomega Grupo Editor, S.A. de C.V.

Pitágoras 1139, Col. Del Valle, 03100, México D.F.

Miembro de la Cámara Nacional de la Industria Editorial Mexicana

Registro N° 2317

Página Web: <http://www.alfaomega.com.mx>

E-mail: atencionalcliente@alfaomega.com.mx

ISBN: 978-607-707-589-9

Derechos reservados:

Esta obra es propiedad intelectual de su autor y los derechos de publicación en lengua española han sido legalmente transferidos al editor. Prohibida su reproducción parcial o total por cualquier medio sin permiso por escrito del propietario de los derechos del copyright.

Esta edición puede venderse en todos los países del mundo.

Impreso en México. Printed in Mexico.

Empresas del grupo:

Argentina: Alfaomega Grupo Editor Argentino, S.A.

Paraguay 1307 P.B. "11", Buenos Aires, Argentina, C.P. 1057

Tel.: (54-11) 4811-7183 / 0887 - E-mail: ventas@alfaomegaeditor.com.ar

México: Alfaomega Grupo Editor, S.A. de C.V.

Pitágoras 1139, Col. Del Valle, México, D.F., México, C.P. 03100

Tel.: (52-55) 5575-5022 - Fax: (52-55) 5575-2420 / 2490. Sin costo: 01-800-020-4396

E-mail: atencionalcliente@alfaomega.com.mx

Colombia: Alfaomega Colombiana S.A.

Carrera 15 No. 64 A 29, Bogotá, Colombia

PBX (57-1) 2100122 - Fax: (57-1) 6068648 - E-mail: cliente@alfaomega.com.co

Chile: Alfaomega Grupo Editor, S.A.

Doctor La Sierra 1437 - Providencia, Santiago, Chile

Tel.: (56-2) 235-4248 - Fax: (56-2) 235-5786 - E-mail: agechile@alfaomega.cl

Dedico este libro

A mi madre Socorro Román Maldonado

A mi esposa Maricela Villa Acosta

A mis hijos Leobardo y Eleazar

Agradecimientos

A Dios, que es mi guía e inspiración, le doy las gracias por su bondad y generosidad al darme salud, fuerza de voluntad y capacidad para desarrollar mi obra editorial.

Mi gratitud para esa gran y noble institución que es la Universidad de Sonora, que en su seno he encontrado la oportunidad de desarrollar mi obra editorial, la cual, ya es parte de su esencia misma.

También quiero agradecer a mis compañeros maestros y alumnos del Departamento de Ingeniería Industrial y de Sistemas de la Universidad de Sonora, y del Instituto Tecnológico de Hermosillo, quienes de una u otra forma han contribuido para que mi obra editorial sea una realidad.

Asimismo agradezco a mis maestros y amigos Manuel Sáiz y Maria Luisa López López (Malichi) sus consejos y su guía; a los maestros Jesús Manuel Solís Santoscoy y Jesús Horacio Pacheco Ramírez sus comentarios y sugerencias.

Un agradecimiento muy especial para mi esposa Maricela Villa Acosta, y para mis hijos Leobardo y Eleazar, a quienes les he quitado mucho tiempo y paciencia para desarrollar esta obra.

Y especialmente a usted, amigo lector, que me ha dado la oportunidad, quiero serle útil a través de mi obra editorial, en primera instancia a través de mis libros anteriores, ahora a través de este libro, y en un futuro espero poner a su disposición un libro que implemente esta metodología en lenguaje Java u otro similar.

Indudablemente habrán quedado errores u omisiones, las cuales espero subsanar en una futura edición, así que, amable lector, si tiene alguna sugerencia para el mejoramiento de esta obra, mucho le agradeceré me la haga llegar. Gracias.

“Escribir un libro, es el más alto honor de un maestro”



Leobardo López Román

Universidad de Sonora. Hermosillo, Sonora, México
llopez@industrial.uson.mx

Mensaje del Editor

Los conocimientos son esenciales en el desempeño profesional. Sin ellos es imposible lograr las habilidades para competir laboralmente. La universidad o las instituciones de formación para el trabajo ofrecen la oportunidad de adquirir conocimientos que serán aprovechados más adelante en beneficio propio y de la sociedad. El avance de la ciencia y de la técnica hace necesario actualizar continuamente esos conocimientos. Cuando se toma la decisión de embarcarse en una vida profesional, se adquiere un compromiso de por vida: mantenerse al día en los conocimientos del área u oficio que se ha decidido desempeñar.

Alfaomega tiene por misión ofrecerles a estudiantes y profesionales conocimientos actualizados dentro de lineamientos pedagógicos que faciliten su utilización y permitan desarrollar las competencias requeridas por una profesión determinada. Alfaomega espera ser su compañera profesional en este viaje de por vida por el mundo del conocimiento.

Alfaomega hace uso de los medios impresos tradicionales en combinación con las tecnologías de la información y las comunicaciones (IT) para facilitar el aprendizaje. Libros como este tienen su complemento en una página Web, en donde el alumno y su profesor encontrarán materiales adicionales, información actualizada, pruebas (test) de autoevaluación, diapositivas y vínculos con otros sitios Web relacionados.

Esta obra contiene numerosos gráficos, cuadros y otros recursos para despertar el interés del estudiante, y facilitarle la comprensión y apropiación del conocimiento.

Cada capítulo se desarrolla con argumentos presentados en forma sencilla y estructurada claramente hacia los objetivos y metas propuestas. Cada capítulo concluye con diversas actividades pedagógicas para asegurar la asimilación del conocimiento y su extensión y actualización futuras.

Los libros de Alfaomega están diseñados para ser utilizados dentro de los procesos de enseñanza-aprendizaje, y pueden ser usados como textos guía en diversos cursos o como apoyo para reforzar el desarrollo profesional.

Alfaomega espera contribuir así a la formación y el desarrollo de profesionales exitosos para beneficio de la sociedad.

Leobardo López Román

Nació en Guamúchil, Sinaloa, México.

Es Licenciado en Informática con especialidad en Sistemas de Información, egresado del Instituto Tecnológico de Culiacán en 1981. En Culiacán, Sinaloa, México.

En 1982 obtuvo el Diplomado en Ciencias de la Computación; y en 1990 la Maestría en Ciencias de la Computación en la Fundación Arturo Rosenblueth, A.C. En México, D.F.

Profesionalmente se ha desempeñado como programador, programador-analista y analista de sistemas en empresas públicas, privadas, de investigación y educativas.

Desde 1982 ha sido profesor de programación de computadoras en diversas instituciones de educación superior, como lo son:

- Instituto Tecnológico de Ciudad Guzmán
- Instituto Tecnológico de Culiacán
- Instituto Tecnológico de Hermosillo
- Actualmente es Maestro de Tiempo Completo en la Universidad de Sonora

Ha recibido el Premio Anual de Profesor Distinguido de la División de Ingeniería de la Universidad de Sonora en 1995, 1998, 2003, 2005, 2006 y 2011.

También recibió el Premio PEDRO INFANTE (Ámbito científico), otorgado por la Secretaría de Educación Pública y Cultura del Estado de Sinaloa, y los H. Ayuntamientos de Salvador Alvarado, Mocorito y Angostura.

Asimismo ha desempeñado otras actividades académicas, como diseño, creación y revisión curricular de nivel licenciatura y postgrado en Computación, Informática y Sistemas de Información, presidente de academia de Computación y Tecnología de la Información, Coordinador de Licenciatura en Informática, ha dirigido más de 40 trabajos profesionales para titulación. Ha sido conferencista en congresos, jornadas de educación y simposios realizados en diversos países.

Además del presente, es autor de los libros:

- **"Programación Estructurada y Orientada a Objetos un enfoque algorítmico 3ª Edición"**, Alfaomega, México, 2011.
- **"Lógica para Computación"**, Alfaomega, México, 2011. Teniendo como coautor a Felipe Ramírez.
- **"Metodología de la Programación Orientada a Objetos"**, Alfaomega, México, 2006.
- **"Programación Estructurada en Lenguaje C"**, Alfaomega, México, 2005.
- **"Programación Estructurada un enfoque algorítmico 2ª Edición"**, Alfaomega, México, 2003.
- **"Programación Estructurada en Turbo Pascal 7"**, Alfaomega, México, 1998.
- **"Programación Estructurada un enfoque algorítmico"**, Alfaomega, México, 1994.

Contenido

1. Introducción a la programación	1	2.3 Estructuras de control.....	33
1.1 Conceptos generales	3	2.4 Resumen de conceptos que debe dominar	33
La computadora.....	3	2.5 Contenido de la página Web de apoyo	34
El programa.....	6	2.5.1 Resumen gráfico del capítulo	34
Estructuras de datos	6	2.5.2 Autoevaluación	34
Operaciones primitivas elementales	7	2.5.3 Power Point para el profesor (*).....	34
Estructuras de control	7	3. La secuenciación	35
El lenguaje de programación	7	3.1 Nuestro primer problema.....	37
Características de los lenguajes de programación	7	3.2 Estructura y diseño de un algoritmo.....	37
La programación	8	3.3 Nuestro primer algoritmo.....	39
Características de un buen programa	8	3.4 Funciones matemáticas.....	41
1.2 Evolución de los paradigmas de programación	9	3.5 Ejercicios resueltos	47
Características de los paradigmas de programación	9	3.6 Ejercicios propuestos.....	51
Programación tradicional.....	9	3.7 Resumen de conceptos que debe dominar	53
Programación estructurada	10	3.8 Contenido de la página Web de apoyo	53
Programación modular	11	3.8.1 Resumen gráfico del capítulo	53
Programación con abstracción de datos	11	3.8.2 Autoevaluación	53
Programación orientada a objetos.....	11	3.8.3 Programas en Java	53
1.3 El proceso de programación	12	3.8.4 Ejercicios resueltos	53
1. Definición del problema.....	12	3.8.5 Power Point para el profesor (*).....	53
2. Análisis del problema	12	4. La selección	55
3. Diseño del programa.....	13	4.1 La selección doble (if-then-else)	57
4. Codificación del programa.....	14	4.1.1 Sangrado (indentación) y etiquetas.....	60
5. Implantación del programa.....	14	4.1.2 Expresiones lógicas	61
6. Mantenimiento del programa	14	4.1.3 if's anidados.....	65
1.4 El algoritmo.....	15	4.1.4 Ejercicios resueltos para la selección doble (if-then-else)	68
Ejercicios	15	4.2 La selección simple (if-then).....	72
1.5 Ejercicios propuestos.....	18	4.2.1 Ejercicios resueltos para la selección simple (if-then).....	74
1.6 Resumen de conceptos que debe dominar	19	4.3 La selección múltiple (switch).....	77
1.7 Contenido de la página Web de apoyo	19	4.3.1 Ejercicio resuelto para la selección múltiple (switch)81	
1.7.1 Resumen gráfico del capítulo	19	4.4 Ejercicios propuestos.....	83
1.7.2 Autoevaluación	19	4.5 Resumen de conceptos que debe dominar	85
1.7.3 Power Point para el profesor (*).....	19	4.6 Contenido de la página Web de apoyo	86
2. Elementos para solucionar problemas enseudocódigo	21	4.6.1 Resumen gráfico del capítulo	86
2.1 Estructuras de datos.....	23	4.6.2 Autoevaluación	86
2.1.1 Tipos de datos.....	23	4.6.3 Programas en Java	86
2.1.2 Variables	24	4.6.4 Ejercicios resueltos	86
2.1.3 Constantes.....	26	4.6.5 Power Point para el profesor (*).....	86
2.2 Operaciones primitivas elementales	26		
2.2.1 Declarar	26		
2.2.2 Lectura de datos (Entrada)	28		
2.2.3 Operaciones aritméticas fundamentales	29		
2.2.4 Escritura de datos (Salida)	32		

5. La repetición	87	7. Métodos	183
5.1 La repetición do...while	89	7.1 Métodos que no regresan valor	184
5.1.1 Contadores y acumuladores.....	91	7.2 Formato general de una clase con métodos.....	191
5.1.2 Ejercicios resueltos para la repetición do...while	95	7.3 Variables globales, locales y parámetros	192
5.1.3 Ejercicios propuestos para la repetición do...while. 100		7.3.1 Variables globales	192
5.2 La repetición for	108	7.3.2 Variables locales	193
5.2.1 for anidados.....	112	7.3.3 Parámetros.....	194
5.2.2 Ejercicios resueltos para la repetición for.....	112	7.4 Funciones estándar.....	198
5.2.3 Simulación del for con do...while.....	120	7.4.1 Funciones cadena de caracteres	198
5.2.4 Ejercicios propuestos para la repetición for.....	121	7.4.2 Validación de la entrada de datos.....	204
5.3 La repetición while	129	7.4.3 Funciones especiales.....	205
5.3.1 Simulación del do...while con while	131	7.5 Métodos que regresan valor.....	207
5.3.2 Simulación del for con while	132	Referencia de métodos que devuelven valor.....	208
5.3.3 Ejercicios resueltos para la repetición while.....	135	7.6 Ejercicios resueltos	210
5.3.4 Ejercicios propuestos para la repetición while.....	140	7.7 Ejercicios propuestos.....	217
5.4 Resumen de conceptos que debe dominar.....	142	7.8 Resumen de conceptos que debe dominar.....	220
5.5 Contenido de la página Web de apoyo	143	7.9 Contenido de la página Web de apoyo	220
5.5.1 Resumen gráfico del capítulo	143	7.9.1 Resumen gráfico del capítulo	220
5.5.2 Autoevaluación	143	7.9.2 Autoevaluación	220
5.5.3 Programas en Java	143	7.9.3 Programas en Java	220
5.5.4 Ejercicios resueltos	143	7.9.4 Ejercicios resueltos	220
5.5.5 Power Point para el profesor (*).....	143	7.9.5 Power Point para el profesor (*).....	220
6. Arreglos	145	8. Programación orientada a objetos usando el diagrama de clases	221
6.1 Arreglos unidimensionales.....	147	8.1 Objetos.....	222
Manejo de los elementos del arreglo unidimensional	148	8.1.1 Qué son los objetos.....	223
Definición del arreglo unidimensional.....	148	8.1.2 Cómo están formados los objetos.....	223
6.1.1 Ejercicios resueltos para unidimensionales.....	150	8.1.3 Cuando y cómo identificar los objetos	224
6.2 Arreglos bidimensionales.....	155	8.2 Clases y su relación con los objetos	225
Definición del arreglo bidimensional.....	155	8.2.1 Determinar las clases.....	225
Manejo de los elementos del arreglo bidimensional	156	8.2.2 Representación de la clase y sus instancias.....	226
6.2.1 Ejercicios resueltos para bidimensionales.....	158	8.3 Métodos y encapsulación.....	227
6.3 Arreglos tridimensionales	165	8.3.1 Métodos	227
Definición del arreglo tridimensional.....	166	8.3.2 Encapsulación	227
Manejo de los elementos del arreglo tridimensional	167	8.4 Diseño del diagrama de clases	228
6.3.1 Ejercicios resueltos para tridimensionales.....	168	8.4.1 Modificadores de acceso (visibilidad)	229
6.4 Arreglos tetradimensionales	168	8.5 Generar instancias de una clase.....	232
Definición del arreglo tetradimensional.....	171	8.6 Arquitectura modelo-vista-controlador	233
Manejo de los elementos del arreglo tetradimensional... 171		8.7 Resumen de conceptos que debe dominar.....	235
6.4.1 Ejercicios resueltos para tetradimensionales	173	8.8 Contenido de la página Web de apoyo	235
6.5 Ejercicios propuestos	173	8.8.1 Resumen gráfico del capítulo	235
6.6 Resumen de conceptos que debe dominar.....	180	8.8.2 Autoevaluación	235
6.7 Contenido de la página Web de apoyo	181	8.8.3 Power Point para el profesor (*).....	235
6.7.1 Resumen gráfico del capítulo	181		
6.7.2 Autoevaluación	181		
6.7.3 Programas en Java	181		
6.7.4 Ejercicios resueltos	181		
6.7.5 Power Point para el profesor (*).....	181		

9. Programación orientada a objetos aplicando la estructura de secuenciación	237
9.1 Nuestro primer problema.....	238
9.2 Diseño de algoritmos OO usando la secuenciación en pseudocódigo.....	240
9.3 Constructores y destructores.....	251
9.4 Ejercicios resueltos.....	252
9.5 Ejercicios propuestos.....	261
9.6 Resumen de conceptos que debe dominar.....	261
9.7 Contenido de la página Web de apoyo.....	262
9.7.1 Resumen gráfico del capítulo.....	262
9.7.2 Autoevaluación.....	262
9.7.3 Programas en Java.....	262
9.7.4 Ejercicios resueltos.....	262
9.7.5 Power Point para el profesor (*).....	262
10. Programación orientada a objetos aplicando las estructuras de selección	263
10.1 Diseño de algoritmos OO usando la selección doble (if-then-else).....	264
10.2 Diseño de algoritmos OO usando la selección simple (if-then).....	268
10.3 Diseño de algoritmos OO usando la selección múltiple (switch).....	270
10.4 Ejercicios resueltos.....	273
10.5 Ejercicios propuestos.....	281
10.6 Resumen de conceptos que debe dominar.....	282
10.7 Contenido de la página Web de apoyo.....	282
10.7.1 Resumen gráfico del capítulo.....	282
10.7.2 Autoevaluación.....	282
10.7.3 Programas en Java.....	282
10.7.4 Ejercicios resueltos.....	282
10.7.5 Power Point para el profesor (*).....	282
11. Programación orientada a objetos aplicando las estructuras de repetición	283
11.1 Diseño de algoritmos OO usando la repetición do...while.....	284
11.1.1 Contadores y acumuladores.....	288
11.1.2 Ejercicios resueltos para do...while.....	291
11.1.3 Ejercicios propuestos para do...while.....	302
11.2 Diseño de algoritmos OO usando la repetición for.....	302
11.2.1 Ejercicios resueltos para for.....	304
11.2.2 Ejercicios propuestos para for.....	309
11.3 Diseño de algoritmos OO usando la repetición while.....	309
11.3.1 Ejercicios resueltos para while.....	313
11.3.2 Ejercicios propuestos para while.....	323
11.4 Resumen de conceptos que debe dominar.....	323
11.5 Contenido de la página Web de apoyo.....	324
11.5.1 Resumen gráfico del capítulo.....	324
11.5.2 Autoevaluación.....	324
11.5.3 Programas en Java.....	324
11.5.4 Ejercicios resueltos.....	324
11.5.5 Power Point para el profesor (*).....	324
12. Programación orientada a objetos aplicando arreglos	325
12.1 Diseño de algoritmos OO usando arreglos unidimensionales.....	326
12.1.1 Ejercicios propuestos para arreglos unidimensionales.....	334
12.2 Diseño de algoritmos OO usando arreglos bidimensionales.....	334
12.2.1 Ejercicios propuestos para arreglos bidimensionales.....	344
12.3 Diseño de algoritmos OO usando arreglos tridimensionales.....	345
12.3.1 Ejercicios propuestos para arreglos tridimensionales.....	349
12.4 Diseño de algoritmos OO usando arreglos tetradimensionales.....	349
12.4.1 Ejercicios propuestos para arreglos tetradimensionales.....	355
12.5 Resumen de conceptos que debe dominar.....	355
12.6 Contenido de la página Web de apoyo.....	356
12.6.1 Resumen gráfico del capítulo.....	356
12.6.2 Autoevaluación.....	356
12.6.3 Programas en Java.....	356
12.6.4 Ejercicios resueltos.....	356
12.6.5 Power Point para el profesor (*).....	356
13. Programación orientada a objetos usando herencia	357
13.1 Herencia.....	358
13.2 Diseño del diagrama de clases con herencia.....	360
13.2.1 Superclases y subclases.....	362
13.3 Diseño de algoritmos OO usando herencia.....	363
13.4 Ejercicios resueltos.....	371
13.5 Ejercicios propuestos.....	381
13.6 Resumen de conceptos que debe dominar.....	384
13.7 Contenido de la página Web de apoyo.....	384
13.7.1 Resumen gráfico del capítulo.....	384
13.7.2 Autoevaluación.....	384
13.7.3 Programas en Java.....	384
13.7.4 Ejercicios resueltos.....	384
13.7.5 Power Point para el profesor (*).....	384

14. Programación orientada a objetos usando polimorfismo	385	Expansión.....	447
14.1 Polimorfismo.....	386	Altas.....	447
14.2 Diseño del diagrama de clases con polimorfismo.....	387	Bajas.....	448
14.2.1 Clases abstractas.....	388	Cambios.....	448
14.3 Diseño de algoritmos OO usando polimorfismo.....	389	Consultas.....	448
14.4 Ejercicios resueltos.....	395	Obtención de reportes.....	448
14.5 Ejercicios propuestos.....	405	15.6 Ejercicios resueltos	467
14.6 Resumen de conceptos que debe dominar.....	408	15.7 Ejercicios propuestos	467
14.7 Contenido de la página Web de apoyo.....	409	15.8 Resumen de conceptos que debe dominar	481
14.7.1 Resumen gráfico del capítulo.....	409	15.9 Contenido de la página Web de apoyo	482
14.7.2 Autoevaluación.....	409	15.9.1 Resumen gráfico del capítulo.....	482
14.7.3 Programas en Java.....	409	15.9.2 Autoevaluación.....	482
14.7.4 Ejercicios resueltos.....	409	15.9.3 Programas en Java.....	482
14.7.5 Power Point para el profesor (*).....	409	15.9.4 Ejercicios resueltos.....	482
15. Registros y archivos	411	15.9.5 Power Point para el profesor (*).....	482
Características de los archivos.....	414	16. Otros temas	483
Tipos de archivos.....	415	16.1 Clasificación (ordenación) de datos	484
Operaciones sobre archivos.....	415	Métodos de ordenamiento o clasificación.....	484
Operaciones con registros.....	416	Clasificación (ordenación) de un archivo.....	491
15.1 Organización de archivos	416	Clasificación de un archivo por dos datos concatenados.....	496
Tipos de organización de archivos.....	416	16.2 Uso del tipo static	500
15.2 Manejo de registros en seudocódigo	418	16.3 Uso del this	512
Definición de registros.....	418	16.4 Recursividad	518
Proceso de un registro.....	419	16.5 Graficación	519
15.3 Operaciones para el manejo de archivos en seudocódigo	424	Arco.....	520
Creación de archivo secuencial.....	424	Barra.....	520
Creación de archivo directo.....	425	Barra3D.....	520
Abrir archivo secuencial.....	426	Circulo.....	520
Abrir archivo directo.....	427	CierraGraph.....	520
Escritura de objetos (registros).....	427	DetectaGraph.....	520
Lectura de objetos (registros).....	428	DibujaPoli.....	520
Localización de componente (encontrar).....	428	Elipse.....	520
Cerrar archivos.....	429	IniciaGraph.....	520
Funciones para el proceso de archivos.....	429	Linea.....	521
Funciones de directorio.....	430	RebPastel.....	521
15.4 Proceso de un archivo secuencial	431	Rectangulo.....	521
Creación.....	431	16.6 Resumen de conceptos que debe dominar	521
Expansión.....	431	16.7 Contenido de la página Web de apoyo	521
Actualización con altas, bajas y cambios.....	431	16.7.1 Resumen gráfico del capítulo.....	521
Obtención de reportes.....	431	16.7.2 Autoevaluación.....	521
Emisión de reportes con cortes de control.....	441	16.7.3 Programas en Java.....	521
15.5 Proceso de un archivo directo	447	16.7.4 Power Point para el profesor (*).....	521
Creación.....	447		

A	Apéndice A.....	523
	Conclusiones y recomendaciones	524
B	Apéndice B	525
	Cómo evolucionar de la programación estructurada a la programación orientada a objetos.....	526
	Diferencia entre la programación estructurada y la orientada a objetos	528
C	Apéndice C	533
	Algoritmos sin usar etiquetas.....	534
	Bibliografía.....	539

Información del contenido de la página Web

El material marcado con asterisco (*) solo está disponible para docentes.

Capítulo 1

Introducción a la programación

- Mapa conceptual del capítulo
- Autoevaluación para el alumno
- **Presentaciones en Power Point para el profesor

Capítulo 2

Elementos para solucionar problemas enseudocódigo

- Mapa conceptual del capítulo
- Autoevaluación para el alumno
- **Presentaciones en Power Point para el profesor

Capítulo 3

La secuenciación

- Mapa conceptual del capítulo
- Autoevaluación para el alumno
- **Presentaciones en Power Point para el profesor

Capítulo 4

La selección

- Mapa conceptual del capítulo
- Autoevaluación para el alumno
- **Presentaciones en Power Point para el profesor

Capítulo 5

La repetición

- Mapa conceptual del capítulo
- Autoevaluación para el alumno
- **Presentaciones en Power Point para el profesor

Capítulo 6

Arreglos

- Mapa conceptual del capítulo
- Autoevaluación para el alumno
- **Presentaciones en Power Point para el profesor

Capítulo 7

Métodos

- Mapa conceptual del capítulo
- Autoevaluación para el alumno
- **Presentaciones en Power Point para el profesor

Capítulo 8

Programación orientada a objetos usando el diagrama de clases

- Mapa conceptual del capítulo
- Autoevaluación para el alumno
- **Presentaciones en Power Point para el profesor

Capítulo 9

Programación orientada a objetos aplicando la estructura de secuenciación

- Mapa conceptual del capítulo
- Autoevaluación para el alumno
- **Presentaciones en Power Point para el profesor

Capítulo 10

Programación orientada a objetos aplicando las estructuras de selección

- Mapa conceptual del capítulo
- Autoevaluación para el alumno
- **Presentaciones en Power Point para el profesor

Capítulo 11

Programación orientada a objetos aplicando las estructuras de repetición

- Mapa conceptual del capítulo
- Autoevaluación para el alumno
- **Presentaciones en Power Point para el profesor

Capítulo 12

Programación orientada a objetos aplicando arreglos

- Mapa conceptual del capítulo
- Autoevaluación para el alumno
- **Presentaciones en Power Point para el profesor

Capítulo 13

Programación orientada a objetos usando herencia

- Mapa conceptual del capítulo
- Autoevaluación para el alumno
- **Presentaciones en Power Point para el profesor

Capítulo 14

Programación orientada a objetos usando polimorfismo

- Mapa conceptual del capítulo
- Autoevaluación para el alumno
- **Presentaciones en Power Point para el profesor

Registro en la Web de apoyo

Para tener acceso al material de la página Web de apoyo del libro:

1. Ir a la página <http://libroweb.alfaomega.com.mx>
2. Registrarse como usuario del sitio y propietario del libro.
3. Ingresar al apartado de inscripción de libros y registrar la siguiente clave de acceso

4. Para navegar en la plataforma virtual de recursos del libro, usar los nombres de Usuario y Contraseña definidos en el punto número dos. El acceso a estos recursos es limitado. Si quiere un número extra de accesos, escriba a webmaster@alfaomega.com.mx

Estimado profesor: Si desea acceder a los contenidos exclusivos para docentes, por favor contacte al representante de la editorial que lo suele visitar o escribanos a: webmaster@alfaomega.com.mx

	<p>Conceptos para recordar: bajo este icono se encuentran definiciones importantes que refuerzan lo explicado en la página.</p>
	<p>Comentarios o información extra: este icono ayuda a comprender mejor o ampliar el texto principal.</p>
	<p>Contenidos interactivos: indica la presencia de contenidos extra en la Web.</p>

Prólogo

En la actualidad muchos estudiantes y profesionales de la programación de computadoras están aprendiendo Java, que es un lenguaje orientado a objetos. Sin embargo, muchos de ellos no están aprendiendo a programar orientado a objetos, porque se les está enseñando prácticamente en forma directa con el lenguaje Java, y no se les está enseñando a “pensar”; es decir, no están desarrollando la lógica de la programación orientada a objetos.

Mi idea es que lo fundamental al aprender a programar computadoras es desarrollar la lógica necesaria para solucionar problemas en forma algorítmica, independientemente de algún lenguaje de programación; esto es, aprender a diseñar algoritmos usando un pseudolenguaje, y no hacerlo directamente con un lenguaje.

Metodología de la Programación Orientada a Objetos 2ª Edición es un libro que viene a coadyuvar en la solución de una necesidad largamente experimentada por la comunidad académica de la programación de computadoras: contar con una metodología que permita conducir la enseñanza-aprendizaje de la programación, mediante el uso de un pseudolenguaje de diseño de programas o algoritmos orientados a objetos, y dirigido a personas que están iniciando sus estudios de programación de computadoras.

En este libro se presenta una metodología de la programación de computadoras que contiene en forma natural los conceptos, estructuras y filosofía que se han generado hasta estos tiempos en que la programación orientada a objetos y el lenguaje Java marcan la pauta de la programación de computadoras.

Esta metodología es el resultado de la integración y adaptación de varias técnicas, como son los conceptos y estructuras de la programación orientada a objetos: objetos, clases, encapsulación, herencia, polimorfismo; el diagrama de clases de UML (Unified Modeling Language, desarrollado por G. Booch, I. Jacobson y J. Rumbaugh); la arquitectura modelo-vista-controlador; algunos conceptos introducidos por el lenguaje Java; y los conceptos y bases lógicas de la programación estructurada en pseudocódigo. Dicha metodología permite diseñar programas o algoritmos orientados a objetos, bien estructurados, bien documentados, eficaces, eficientes y fáciles de darles mantenimiento.

El método ha sido plasmado de manera tal que conduce el proceso enseñanza-aprendizaje de la programación de

computadoras en forma didáctica, simple, completa, consistente, y práctica con una gran cantidad y variedad de ejercicios cuidadosamente seleccionados y probados en clase por el autor, y que va desde un nivel de principiante, pasando por intermedio, y establece las bases para el nivel avanzado.

Lo relevante de este método es que enseña a programar computadoras utilizando un pseudolenguaje, es decir, sin utilizar la computadora directamente. Esto permite desarrollar las capacidades mentales que una persona debe tener para programar computadoras y sienta las bases de disciplina y buena estructura. Este enfoque se le dificulta a mucha gente; sin embargo, hay que enfrentarlo, porque siendo la programación una actividad intelectual que requiere mucha creatividad, capacidad de abstracción, de análisis y de síntesis, éstas no se pueden desarrollar operando un lenguaje en la computadora, sino ejercitando la mente en forma apropiada.

Problemática de la enseñanza de la programación orientada a objetos

En la actualidad, algunos maestros de programación pensamos que, para aprender a programar computadoras, el estudiante debe seguir un proceso: empezando con el estudio de la lógica básica de la programación usando un pseudolenguaje como pseudocódigo, enseguida debe aprender a implementar esa lógica en lenguaje C, luego debe aprender a diseñar programas o algoritmos orientados a objetos, para después aprender a implementar esa lógica orientada a objetos en lenguaje Java. Sin embargo, hace unos cuantos años, en ese proceso involucrábamos al lenguaje Pascal. Con el paso del tiempo, seguramente involucraremos a otros lenguajes. El hecho es que la aparición de nuevos lenguajes que se convierten en moda, mete en grandes problemas a la comunidad académica de la programación de computadoras.

Concretamente, en los últimos años se ha insistido y ejercido una gran presión para que Java sea el primer y único lenguaje que se enseñe; además, que se use desde la fase introductoria a la programación, eliminando un curso previo de lógica. También se dice que al estudiar el lenguaje Java va implícita la lógica; que la programación es mucho más fácil, rápida, agradable y avanzada en Java que lo que anteriormente era la programación, etcétera.

Alguna gente, incluso autores de libros y artículos, dice que cualquier persona que no sepa nada de programación puede entender fácilmente los conceptos de la programación orientada a objetos; y estoy de acuerdo: en un nivel abstracto cualquiera puede comprenderlos, pero en el momento en que se debe implementar los objetos en instrucciones en un lenguaje de programación es donde se dan cuenta de que “algo” falta. Esto se debe a que un programa orientado a objetos se compone por un conjunto de objetos y métodos que implementan las funciones de los objetos, y a esos métodos hay que enviarles mensajes a través de parámetros, para que hagan cosas. De manera que ese “algo” que falta es la lógica básica de la programación, esto es: tipos de datos (entero, real, cadena, carácter, arreglos, registros y archivos); estructuras de control; secuenciación, if-then, if-then-else, switch, do... while, for, while; métodos (módulos y funciones definidas por el usuario); parámetros por valor y por referencia. Es por ello que digo que esas estructuras son la base de la programación orientada a objetos y que una persona que no domine esos conceptos jamás podrá comprender cómo implementar los objetos.

Si revisamos la historia de la enseñanza-aprendizaje de la programación de computadoras, un fenómeno similar se dio con el lenguaje BASIC, que vino a ganarle lugar a la programación con diagramas de flujo, lenguaje COBOL y FORTRAN, supuestamente haciendo más fácil aprender a programar usando directamente el lenguaje. Esto causó mucha confusión y problemas en la enseñanza-aprendizaje y práctica profesional de la programación, pero después vino el lenguaje Pascal y la programación estructurada a componer la situación. Tomó muchos años para que en algunas (no todas) instituciones de educación se convencieran de que lo adecuado es enseñar primero la lógica básica de la programación usando un pseudolenguaje como pseudocódigo, y después enseñar cómo implementar esa lógica en un lenguaje de programación como lenguaje C u otro.

El problema actual es que muchas instituciones de educación han sucumbido ante la presión y están enseñando el lenguaje Java desde el primer semestre. Esto ha causado mucha confusión, porque la programación orientada a objetos no está sustentada metodológicamente para niveles básicos de aprendizaje, y están cayendo en la situación de “enseñar a usar” el lenguaje Java, que es un lenguaje orientado a objetos, pero no están enseñando a programar orientado a objetos usando el lenguaje Java, que sería lo correcto.

Propuesta sugerida en este libro

Este autor está de acuerdo en que Java o cualquier otro lenguaje orientado a objetos que venga a sustituirlo sea utilizado como primer lenguaje en la enseñanza-aprendizaje de la programación, pero antes debe enseñarse un curso de lógica de la programación orientada a objetos, donde los estudiantes aprendan los fundamentos de la programación orientada a objetos usando un pseudolenguaje como pseudocódigo, aplicándolo en el diseño de programas o algoritmos orientados a objetos, es decir, en forma independiente de un lenguaje de programación; esto es lo que le presento en este libro. Y en una siguiente instancia, el estudiante deberá aprender a implementar esos fundamentos y algoritmos en lenguaje Java (u otro similar); esto le presentaré próximamente en otro libro.

Esto es para no “casar” la formación lógica de los estudiantes con el lenguaje que esté de moda, porque al cambiar el lenguaje que está de moda, la formación que se les dio con el anterior lenguaje se convierte en “deformación”; la programación es lógica y debemos desarrollarla en los estudiantes independientemente de algún lenguaje de programación, porque los lenguajes van y vienen, y la lógica ahí debe estar, para ser implementada en el lenguaje en turno.

La metodología propuesta

La metodología que se presenta en este libro se divide en dos partes; en la primera, que abarca del capítulo uno al siete, se estudia la técnica pseudocódigo y su uso en el diseño de algoritmos pequeños que tienen una sola tarea o función; por tanto, se establece el uso de una clase y dentro de la clase el método principal, donde se plasma la lógica que soluciona el problema. En esta primera parte se da énfasis al desarrollo de la lógica básica de la programación usando un pseudolenguaje. Se estudian los tipos de datos, identificadores, operaciones de entrada, cálculo y salida, usando las estructuras de control: la secuenciación; la selección simple (if-then), doble (if-then-else) y múltiple (switch); la repetición do...while, la repetición for y la repetición while; los arreglos unidimensionales, bidimensionales, tridimensionales y tetradimensionales; y por último, en esta primera parte, se estudia cómo usar más de un método en la clase en problemas que involucran a más de una tarea o función, métodos que no regresan valor (equivalente a módulos en la programación estructurada), métodos que regresan valor (equivalentes a funciones

definidas por el usuario en la programación estructurada), parámetros por valor y por referencia.

Este autor tiene la convicción de que el estudiante debe desarrollar las bases lógicas de la programación. Es por ello que esta primera parte es lo que se estudia en un primer curso de lógica de programación con técnicas estructuradas, pero enfocando la estructura del algoritmo en forma apropiada a la programación orientada a objetos, usando una clase y dentro de la clase el método principal, dejando bien cimentadas las bases lógicas de la programación de computadoras.

En la segunda parte, que abarca del capítulo ocho al dieciséis, es donde se estudian de lleno los conceptos de la programación orientada a objetos, integrándolos con el concepto de diagrama de clases de UML (Unified Modeling Language), con la arquitectura modelo-vista-controlador, con las estructuras estudiadas en los primeros siete capítulos y la incorporación de los conceptos de la programación orientada a objetos en la técnica pseudocódigo, logrando una metodología de la programación que permite diseñar algoritmos orientados a objetos.

Mejoras de esta segunda edición

Las mejoras realizadas en esta segunda edición se dividen en cinco partes:

Primera:

Considerando que el lenguaje C es la base de C++, C#, Java, etcétera, a la metodología se le han hecho algunos ajustes para asemejarla un poco más a las estructuras básicas de dichos lenguajes, como es el manejo de las palabras IF, THEN, ELSE, SWITCH, DO, WHILE y FOR en minúsculas. En las expresiones lógicas el uso de == (doble signo de igual) en lugar de = (un signo igual) y != en lugar de <> para no igual. Y el uso de apóstrofes en lugar de comillas en las constantes de tipo carácter.

Segunda:

Los capítulos 5. La repetición do...while, 6. La repetición for y 7. La repetición while, se han juntado en el capítulo 5. La repetición. Recorriéndose el capítulo 8 al 6, el 9 al 7, el 10 al 8, el 11 al 9, el 12 al 10, el 13 al 11, el 14 al 12, el 15 al 13 y el 16 al 14.

Tercera:

Algunos ejercicios han sido ampliados y/o modificados para darle una mayor profundidad y una mejor aplicación

a la metodología de la programación orientada a objetos, lo que ayudará a que el aprendizaje de la técnica sea mejor.

Cuarta:

Se han agregado dos nuevos capítulos: el capítulo 15, en el que se presenta el tema de registros y archivos, el cual significa una ampliación muy importante, y el capítulo 16, en el que se presentan otros temas: la clasificación (ordenación) de datos, el uso del tipo static (estático), el uso del this, el manejo de la recursividad y graficación.

Quinta:

Aunque el propósito de este libro no es estudiar ningún lenguaje de programación, en la dirección <http://virtual.alfaomega.com.mx>, que es una web de ayuda que la editorial Alfaomega ha puesto a su disposición, podrá encontrar los programas correspondientes de todos los algoritmos del libro, codificados en lenguaje Java, en una versión más actual que incluye el uso de la clase Scanner. Además, en la web del libro, se encuentran ejercicios resueltos de los capítulos 3 al 7 y del 9 al 15. Asimismo, en la web se encuentran otros recursos de apoyo, que en cada capítulo se indican.

Organización del libro

El material de la presente obra está dividido en dieciséis capítulos y tres apéndices. En el capítulo uno se presenta una introducción a la programación. En el capítulo dos se describen los elementos para solucionar problemas en pseudocódigo, como son tipos de datos, identificadores, variables, operaciones aritméticas, lectura y escritura de datos. En el capítulo tres se estudia la estructura de control secuenciación, su definición, estructura de un algoritmo y su aplicación en algunos ejemplos. El capítulo cuatro trata la selección simple (if-then), doble (if-then-else) y múltiple (switch), su definición, formato en pseudocódigo y utilización con ejercicios resueltos.

En el capítulo cinco se explica la repetición do...while, la repetición for, y la repetición while, su definición, formato y aplicación en ejercicios resueltos. En el capítulo seis se presentan los arreglos unidimensionales, bidimensionales, tridimensionales y tetradimensionales; su definición, formato, manipulación y uso en ejercicios resueltos.

En el capítulo siete se tratan los métodos; métodos que no regresan valor, equivalente a módulos en la programación estructurada. Se aprenderá el manejo de variables

globales, locales y el uso de parámetros por valor y por referencia. Asimismo, se estudian los métodos que regresan valor, equivalente a funciones definidas por el usuario en la programación estructurada. También se estudian algunas funciones estándar como son: funciones para la manipulación de cadenas de caracteres y algunas funciones especiales.

En el capítulo ocho se estudian los conceptos de objetos, clases, métodos y encapsulación, y se explica la forma de cómo involucrarlos para diseñar el diagrama de clases. Asimismo, se incluye el uso de la arquitectura modelo-vista-controlador.

En el capítulo nueve se estudia cómo diseñar la lógica de cada una de las clases usando pseudocódigo, incorporando en esta técnica los conceptos de clases, objetos y encapsulación. Se presenta el diseño de algoritmos orientados a objetos aplicando la estructura de secuenciación con ejemplos que se estudiaron en el capítulo tres.

En el capítulo diez se presenta el diseño de algoritmos orientados a objetos aplicando las estructuras de selección if-then-else, if-then y switch, con ejemplos que se estudiaron en el capítulo cuatro.

En el capítulo once se presenta el diseño de algoritmos orientados a objetos aplicando las estructuras de repetición do...while, for y while, con ejemplos que se estudiaron en el capítulo cinco.

En el capítulo doce se presenta el diseño de algoritmos orientados a objetos usando arreglos unidimensionales, bidimensionales, tridimensionales y tetradimensionales, con ejemplos que se estudiaron en el capítulo ocho.

En el capítulo trece se estudia el concepto de herencia, se explica cómo diseñar el diagrama de clases y el diseño de las clases usando pseudocódigo, involucrando la herencia.

En el capítulo catorce se introduce el concepto de polimorfismo y se estudia cómo diseñar algoritmos involucrándolo.

El capítulo quince trata lo referente a registros y archivos, donde se presentan definiciones, organización de archivos, cómo manejarlos en pseudocódigo, el proceso de un archivo secuencial, el proceso de un archivo directo, obtención de reportes, ejercicios resueltos y propuestos.

El capítulo dieciséis, como si fuera un apéndice, presenta una serie de temas sueltos: clasificación (ordenación) de datos, el uso del tipo static (estático), el uso del this, el

manejo de la recursividad y graficación, donde cada tema puede estudiarse en el momento en que se considere necesario y permiten que la metodología de la programación orientada a objetos se utilice en temas más avanzados de programación.

En el apéndice A se presentan algunas conclusiones y recomendaciones. Aunque es importante que todos los lectores lo revisen; es particularmente relevante para aquellos lectores que piensen que a esta metodología le falta algo. En el apéndice B se explica cómo evolucionar de la programación estructurada al uso de la programación orientada a objetos. En el apéndice C se presentan algunos algoritmos sin usar etiquetas. Por último se tiene la bibliografía.

Cómo usar la metodología

Este trabajo requiere que el lector tenga conocimientos previos sobre conceptos generales relacionados con las computadoras, en caso de no ser así, se recomienda la lectura de algún libro que sirva como introducción a las computadoras, a las tecnologías de la información o a la informática.

El libro puede ser utilizado como texto o consulta en materias de fundamentos de programación, metodología de la programación, programación de computadoras, programación para ingenieros y similares, que se cursan en los primeros dos o tres semestres de carreras como Informática, Computación Sistemas Computacionales, Sistemas de Información, Ingeniería Industrial y de Sistemas, Ingeniería Mecatrónica, Ingeniería Electrónica, Ingeniería Eléctrica, entre otras, como lo son preparatorias, bachilleratos y carreras de nivel técnico.

Para estudiantes que empiezan la programación desde cero se deben estudiar todos los capítulos en el orden en que están: los capítulos del 1 al 7 en alrededor de 40 horas, y del 8 al 16 en aproximadamente 40 horas.

Para estudiantes que ya llevaron un curso de metodología de la programación estructurada y/o un lenguaje como Pascal o C, deben hacer un repaso rápido de los capítulos del 2 al 7, y luego estudiar detenidamente los capítulos del 8 al 16.

Advertencia didáctica

Se ha tomado el ejemplo de calcular sueldo(s) de empleado(s) -pago de sueldo o nómina- como un problema pivote para facilitar la explicación de las estructuras que integran la metodología. Esto quiere decir que, al estudiar cada elemento o estructura, el primer problema que se plantea es el de pago de sueldo, primero de una manera muy simple, y luego con una variante apropiada para la aplicación de lo que se está explicando en cada momento. Esto es con fines didácticos, ya que es una situación fácilmente entendible, que permite al estudiante (lector) familiarizarse con ella porque se trata desde el inicio y durante todo el libro, y así podemos dedicar todo nuestro esfuerzo mental al aspecto lógico de la metodología. Es decir, que los problemas no deben ser muy complicados, para dirigir todo nuestro esfuerzo mental a comprender la lógica de la metodología y no a entender problemas innecesariamente complejos.

En consecuencia, es probable que el lector perciba que éste es un libro con una orientación administrativa; sin embargo, después de explicarle cada estructura con el ejemplo antes referido, en los ejercicios resueltos, se presenta la aplicación de la estructura con otros tipos de problemas, dándole a la metodología un enfoque de aplicación general, es decir, tanto para el área administrativa, como para la ingeniería.

Resumiendo

Aprender a programar no es fácil ni rápido, es un proceso que debe iniciar con el desarrollo de la lógica usando un pseudolenguaje de diseño de programas o algoritmos. Si el estudiante ya domina la lógica hasta lo que es la programación estructurada, debe aprender la lógica de la programación orientada a objetos. Y después, en una siguiente instancia, debe aprender cómo implementar la lógica usando un lenguaje de programación como Java u otro similar.

Con el estudio de la metodología y fundamentos de programación que le presento en este libro, el estudiante aprenderá la lógica de la programación orientada a objetos sin estar “casado” con ningún lenguaje, y de aquí en adelante podrá aprender y comprender cualquier lenguaje orientado a objetos como Java, C#, UML, etcétera. Sin ser esto cierto, se podría decir que esta metodología es algo así como un UML para principiantes.

Asimismo, creo que en esta obra es donde los conceptos de la programación orientada a objetos están más accesibles, claros y aterrizados aplicándolos en problemas cotidianos de proceso de datos.

Introducción a la programación

Contenido

- 1.1 Conceptos generales
- 1.2 Evolución de los paradigmas de programación
- 1.3 El proceso de programación
- 1.4 El algoritmo
- 1.5 Ejercicios propuestos
- 1.6 Resumen de conceptos que debe dominar
- 1.7 Contenido de la página Web de apoyo
 - El material marcado con asterisco (*) sólo está disponible para docentes.*
 - 1.7.1 Resumen gráfico del capítulo
 - 1.7.2 Autoevaluación
 - 1.7.3 Power Point para el profesor (*)

Objetivos del capítulo

- Repasar conceptos generales de computación.
- Comprender los fundamentos del funcionamiento de la computadora y sus componentes.
- Entender el concepto de programa.
- Comprender las características principales de un lenguaje de programación.
- Entender cuáles son las características de un buen programa.
- Conocer los diversos paradigmas de programación.
- Entender cuáles son los pasos que integran el proceso de programación.
- Aprender qué es el algoritmo, cuáles son las características que tiene y aplicar los conceptos aprendidos en situaciones de la vida cotidiana.

Competencias

- Competencia general del capítulo
 - *Describir los conceptos básicos de la programación.*
- Competencias específicas del capítulo
 - Describe el concepto de computadora como herramienta de proceso de datos (entrada-proceso-salida).
 - Define los conceptos programa de computadora, lenguaje de programación y qué es la programación.
 - Describe la evolución de los paradigmas de programación.
 - Enuncia los pasos del proceso de programación.
 - Describe los pasos de la metodología para resolver problemas.
 - Define el concepto de algoritmo.
 - Enuncia las características de los algoritmos.
 - Compilar y ejecutar programas.

Introducción

Antes de iniciar el tema objeto de esta obra es conveniente, sobre todo para aquellos que comienzan su instrucción informática, hacer un breve repaso de conceptos fundamentales que se han de tomar en cuenta cuando se desarrollan programas para computadoras.

En este primer capítulo el lector efectuará un repaso de los principales conceptos acerca de las computadoras. Se presenta la computadora como una herramienta que se utiliza para resolver problemas en el accionar cotidiano de las empresas, organizaciones o instituciones. Esto se hace mediante el esquema del procesamiento electrónico de datos que es E-P-S (Entrada-Proceso-Salida), es decir, entran datos como materia prima y luego se procesan para transformarlos en información que se emite como salida.

Se estudian también los elementos funcionales básicos que componen una computadora, que son: la unidad central de proceso, la unidad de memoria, la unidad de entrada y la unidad de salida. Cabe aclarar que es deseable que estos conceptos ya los domine el estudiante, o bien que los estudie en algún otro libro de introducción a la computación, a la informática o a tecnologías de la información.

Se expone el concepto de programa, que es un conjunto de instrucciones que guían a la computadora para realizar alguna actividad o resolver algún problema. También se detalla que el programa se compone de estructuras de datos, operaciones primitivas elementales y estructuras de control.

Se explica que un lenguaje de programación es el medio a través del cual le comunicamos a la computadora la secuencia de instrucciones que debe ejecutar para llevar a cabo actividades y tareas o para solucionar problemas. Además, se expone que todo lenguaje está compuesto por un alfabeto, un vocabulario y una gramática, luego se revisa el concepto de lo que es la programación y se enumeran las características de un buen programa.

Se presenta la evolución de los paradigmas de programación, que inició con las primeras estructuras que se inventaron cuando apareció la programación de computadoras. Luego, sobre esas bases se gestó la programación estructurada para dar lugar a la programación modular. Enseguida se agregó la programación con abstracción de datos, para llegar al desarrollo de la programación orientada a objetos.

Se detallan los pasos que integran el proceso de programación: definición del problema, análisis del problema, diseño del programa, codificación del programa, implantación del programa y mantenimiento del programa.

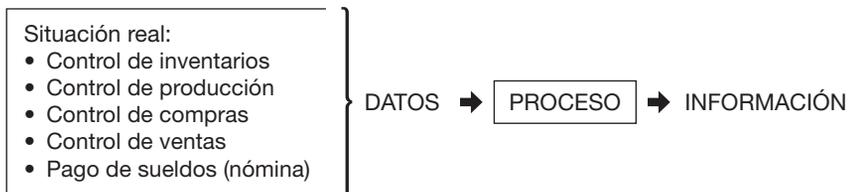
Se define que el algoritmo es una secuencia de pasos que llevan a la solución de un problema o a la ejecución de una tarea o actividad. Se plantea que los pasos del algoritmo deben tener las siguientes características: ser simples, claros, precisos, exactos; tener un orden lógico; tener un principio y un fin.

En el siguiente capítulo se estudian los elementos para solucionar problemas enseudocódigo.

1.1 Conceptos generales

La computadora

La computadora es una herramienta que se utiliza para representar cualquier situación de la realidad en forma de datos, los cuales se procesan después para generar información. Esquemáticamente:



Esto quiere decir que toda situación que pueda ser abstraída y representada en forma de datos puede ser manejada mediante la computadora porque el esquema del proceso de datos es E-P-S (Entrada-Proceso-Salida), es decir, los datos entran como materia prima y se procesan para transformarlos en la información que se da como salida. Por ejemplo, en una situación de pago de sueldos (nómina) un trabajador puede ser representado mediante los datos: Nombre del empleado, Número de horas trabajadas y Cuota por hora. El sueldo se obtiene multiplicando Número de horas trabajadas por Cuota por hora y se da como salida el nombre y el sueldo. Tanto los datos como el procedimiento necesario para generar la información se suministran a la computadora en forma de un programa constituido por instrucciones. La computadora interpreta y ejecuta las instrucciones del programa de acuerdo con ciertas reglas de sintaxis que conforman el lenguaje de programación, mediante el cual podemos comunicarle lo que debe hacer.

Los elementos básicos que componen una computadora son la unidad central de proceso, la unidad de memoria, la unidad de entrada y la unidad de salida.

La *unidad central de proceso* es el “cerebro” que controla el funcionamiento de los componentes y ejecuta las operaciones aritméticas y lógicas. Las operaciones del procesador central son muy simples, pero ejecutadas a una velocidad muy alta —del orden de millones por segundo— permiten la ejecución de tareas simples o complejas.

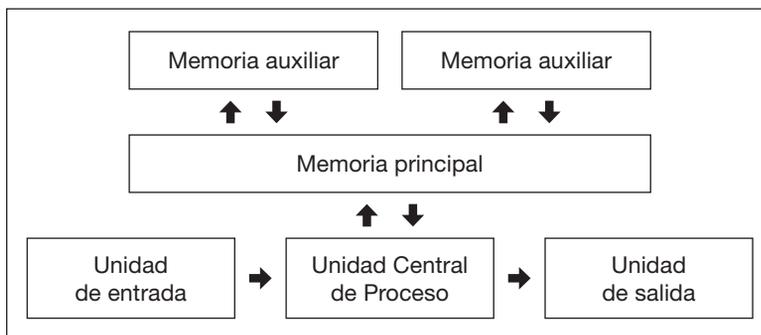


Fig. 1.1 Diagrama que describe la organización funcional de una computadora.

La *memoria* se utiliza para almacenar los datos, y a éstos se les aplican las operaciones del procesador. Existen dos tipos de memoria: la principal y la auxiliar. La memoria principal permite al procesador extraer y almacenar datos a una velocidad comparable a la propia. Cada operación propicia por lo menos un acceso a la memoria. Para que el procesador pueda avanzar de una operación a la siguiente sin retraso, el programa de instrucciones se almacena en esta memoria; en otras palabras, la memoria principal guarda tanto las instrucciones como los datos sobre los que actúa el procesador central. La memoria principal está limitada por su alto costo; debido a esto no es posible conservar en ella grandes cantidades de datos e instrucciones y, en consecuencia, sólo se usa para guardar lo que el procesador esté utilizando por el momento. Además, tiene la característica de que no permite almacenar datos permanentemente, pues si se apaga la computadora se pierde lo que haya en memoria. Por tales razones, las computadoras están equipadas con memorias auxiliares para almacenamiento masivo y permanente de datos, tales como discos magnéticos fijos, disquetes (discos flexibles) magnéticos removibles, discos compactos, cintas magnéticas, entre otros. Estos dispositivos tienen más capacidad que la memoria principal, pero son más lentos. Los datos pueden almacenarse en ellos de manera permanente; es decir, pueden guardarse para usos posteriores.



Fig. 1.2 Computadora personal.

La *unidad de entrada* se utiliza para introducir datos del exterior en la memoria de la computadora a través de dispositivos periféricos de entrada como teclados de terminales, ratón (mouse), discos, módem, lector de código de barras, escáners, etcétera. Esta unidad realiza automáticamente la traducción de símbolos inteligibles para la gente en símbolos que la máquina pueda manejar.



Fig. 1.3 Pantalla o monitor, dispositivo periférico de salida estándar.

La *unidad de salida* permite transferir datos de la memoria al exterior a través de dispositivos periféricos tales como impresoras, pantallas de video, módems, etcétera. Esta unidad realiza automáticamente la traducción de símbolos que puede manejar la máquina en símbolos inteligibles para la gente.



Fig. 1.4 Teclado, dispositivo periférico de entrada estándar.



Fig. 1.5 Impresora, dispositivo periférico de salida impresa.



Fig. 1.6 Mouse, dispositivo periférico de entrada.



Fig. 1.7 Unidad de DVD, dispositivo periférico de entrada/salida.

El programa

Un programa es un conjunto de instrucciones que guían a la computadora para resolver algún problema o realizar alguna actividad.

Un programa se compone por tres elementos: estructuras de datos, operaciones primitivas elementales y estructuras de control, como se muestra a continuación:

programa = estructuras de datos
+ **operaciones primitivas elementales**
+ **estructuras de control**

Estructuras de datos

Son las formas de representación interna de la computadora. Los hechos reales, representados en forma de datos, pueden estar organizados de diferentes maneras (estructuras de datos). Por ejemplo, el nombre del empleado, el número de horas trabajadas y la cuota por hora son los datos mediante los cuales se representa un empleado en una situación de pago de sueldos o nómina.

Operaciones primitivas elementales

Son las acciones básicas que la computadora “sabe” hacer y que se ejecutan sobre los datos para darles entrada, procesarlos y emitirlos como salida, convertidos en información. Por ejemplo, el sueldo de un empleado se calcula multiplicando el número de horas trabajadas por la cuota que se le paga por cada hora de trabajo.

Estructuras de control

Son las formas lógicas de funcionamiento de la computadora mediante las que se dirige el orden en que deben ejecutarse las instrucciones del programa. Las estructuras de control son: la *secuenciación*, que es la capacidad de ejecutar instrucciones en secuencia, una tras otra; la *selección*, que es la capacidad de escoger o seleccionar si algo se ejecuta o no y optar por una de dos o más alternativas; y la *repetición*, que es la capacidad de realizar en más de una ocasión (es decir, repetir cierta cantidad de veces) una instrucción o conjunto de instrucciones: por ejemplo, calcular el sueldo a un empleado, pero repitiendo el cálculo n veces para n empleados.

El lenguaje de programación

El lenguaje de programación es el medio a través del cual le comunicamos a la computadora el programa o el conjunto de instrucciones que debe ejecutar para llevar a cabo actividades o solucionar problemas. Ejemplos de lenguajes de programación son: Java, C#, C++, C, Pascal, Visual Basic, FORTRAN, COBOL, etcétera. Todo lenguaje permite el manejo de los tres elementos que componen un programa, a saber: estructuras de datos, operaciones primitivas elementales y estructuras de control.

Recordemos que mediante un programa podemos representar en forma de datos cualquier situación de nuestra realidad. A los datos se les da entrada a la computadora mediante dispositivos de entrada como teclado, lector óptico de caracteres, ratón, escáner, etcétera; una vez que los datos están en la computadora, se procesan para convertirlos en información, la cual será emitida hacia el exterior de la computadora mediante dispositivos de salida como son la pantalla, la impresora, etcétera.

Características de los lenguajes de programación

Todo lenguaje está compuesto por un alfabeto, un vocabulario y una gramática. A continuación se describen estos componentes.

1. Alfabeto o conjunto de caracteres

Es el conjunto de elementos estructurales del lenguaje:

- a. Caracteres alfabéticos (letras minúsculas y mayúsculas).
- b. Caracteres numéricos (dígitos del 0 al 9).
- c. Caracteres especiales (símbolos especiales tales como [.), [), [:], [;], [\$], [#], [/] y muchos otros).

2. Vocabulario o léxico

Es el conjunto de palabras válidas o reservadas del lenguaje. Por ejemplo, las palabras `do`, `for`, `while`, `if`, `else`, `switch`, `int`, `float`, `double`, `char`, tienen un significado predeterminado en los lenguajes Java, C++ y C; es decir, son palabras reservadas de esos lenguajes. Así, cada lenguaje tiene sus propias palabras reservadas.

3. Gramática

Es el conjunto de reglas sintácticas que se deben seguir para construir frases, oraciones o instrucciones. Siguiendo los lineamientos de la gramática o sintaxis, se construyen las instrucciones mediante las cuales logramos transmitirle a la computadora lo que deseamos. Por ejemplo, para leer datos debemos seguir ciertas reglas, lo propio para imprimir, etcétera.

La programación

Generalmente, se consideran sinónimos los conceptos *programación* y *codificación*, lo cual constituye un error. Debemos tener presente que la finalidad de un programa es realizar algún proceso sobre ciertos datos para obtener ciertos resultados. La preparación de un programa implica aspectos tales como: ¿para qué sirve el proceso que se desea representar?, ¿qué datos usará, qué resultados producirá y cómo se realizará el proceso sobre los datos para obtener los resultados esperados? Una vez identificado lo anterior, se procede a diseñar la manera como la computadora deberá hacerlo, tomando en cuenta su estructura interna y su funcionamiento. Hasta ese momento se tiene representada la solución de una manera convencional (algoritmo), pero enseguida se procede a codificar el programa que solucionará el problema, utilizando un lenguaje de programación.

Características de un buen programa

Un programa bien escrito debe tener ciertas características básicas que le permitan operar correctamente; las principales serían las siguientes:

Operatividad: Lo mínimo que debe hacer un programa es funcionar; es decir, producir los resultados esperados independientemente de cualquier otra característica.

Legibilidad: Un programa puede hacerse más legible dándole cierto formato al código, utilizando el sangrado (indentación) para reflejar las estructuras de control del programa e insertando espacios o tabuladores. Es conveniente diseñar reglas propias para darle uniformidad a todos los programas.

Transportabilidad: Un programa transportable es el que puede ejecutarse en otro entorno sin hacerle modificaciones importantes. Mientras menos modificaciones se hagan será más transportable, así que es conveniente no utilizar características especiales del hardware ni “facilidades” especiales del software.

Claridad: Esta característica se refiere a la facilidad con que el texto del programa comunica las ideas subyacentes. El programa debe indicar claramente lo que el programador desea. Una buena programación es similar a la elaboración de un documento legal; por ejemplo, conviene utilizar nombres adecuados para los identificadores, hacer comentarios correctos, claros y concisos, etcétera.

Modularidad: Dividir el programa en un número de módulos pequeños y fáciles de comprender puede ser la contribución más importante a su calidad. Cada módulo debe realizar sólo una tarea específica, y no más. Los módulos tienen la virtud

de minimizar la cantidad de código que el programador debe comprender a la vez, además de que permiten la reutilización de código.

1.2 Evolución de los paradigmas de programación

Desde que la programación de computadoras apareció como tal, la forma, el paradigma o modelo que se usa ha evolucionado constantemente. Sin embargo, las bases de la programación no han cambiado; simplemente se han ido añadiendo nuevos conceptos y nuevas estructuras. Todo inició con las primeras estructuras que se inventaron cuando apareció la programación de computadoras como tal, que en este libro se está conceptualizando como programación tradicional. Luego, sobre esas bases se gestó la programación estructurada, para dar lugar a la programación modular. Enseguida se agregó la programación con abstracción de datos para llegar al desarrollo de la programación orientada a objetos. En la siguiente figura se esquematiza la evolución de los paradigmas de programación.

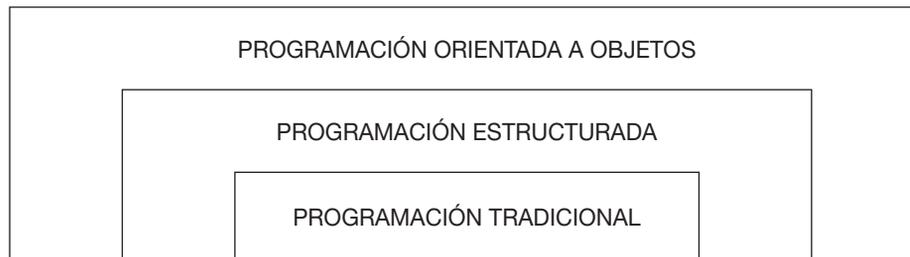


Fig. 1.8 Evolución de los paradigmas de programación

Características de los paradigmas de programación

La evolución de los paradigmas de programación ha tenido tres grandes pasos. El primer gran paso se dio cuando la programación de computadoras se inventó como tal (es lo que se esquematiza en la parte de debajo de la figura como programación tradicional). Luego, como un segundo gran paso, surgió la programación estructurada. Después se experimentó un pequeño paso que dio lugar a la programación modular. Enseguida vino otro pequeño paso que permitió el surgimiento de la programación con abstracción de datos. Luego se dio el tercer gran paso que es la aparición de la programación orientada a objetos.

Programación tradicional

La programación tradicional, que se denominaba programación de computadoras, tuvo sus inicios a principios de la década de 1950. Los lenguajes de programación que se utilizaban eran los predecesores de FORTRAN y las primeras versiones de éste. Las estructuras lógicas de control que se utilizaban eran la secuenciación, IF-THEN, IF-THEN-ELSE y DO (en la actualidad conocido como FOR). La técnica de diseño de programas utilizada era la de los diagramas de flujo.

La estructura general o arquitectura de un programa consistía de un solo módulo, como se muestra a continuación:

Programa

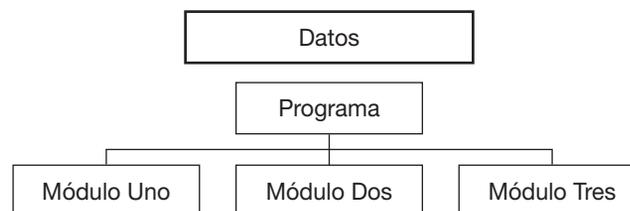
Y ese programa o módulo estaba formado por un conjunto de instrucciones.

Instrucción 1
Instrucción 2
Instrucción 3
Instrucción 4
Instrucción 5
Instrucción 6
Instrucción 7
- - - - -
- - - - -
- - - - -
- - - - -
- - - - -
Instrucción N

Programación estructurada

La programación estructurada tuvo sus inicios a mediados de la década de 1960. Los lenguajes de programación que se utilizaban eran PASCAL, COBOL estructurado, BASIC estructurado, FORTRAN con estilo estructurado, FORTRAN 90, Lenguaje C. Las estructuras de control utilizadas eran la secuenciación, IF-THEN, IF-THEN-ELSE, CASE, FOR, DO-UNTIL y DOWHILE. Otras características eran que se podía dividir un programa en módulos y funciones y estilo de programación. Las técnicas de diseño de programas que se utilizaban eran diagramas Warnier, diagramas estructurados, diagramas Chapin, pseudocódigo y Top Down Design, entre otras.

La estructura general o arquitectura de un programa consistía de datos y de un conjunto de módulos jerarquizados, como se muestra a continuación:



Y cada módulo estaba formado por un conjunto de instrucciones.

Módulo Uno	Módulo Dos	Módulo Tres
Instrucción 1	Instrucción 1	Instrucción 1
Instrucción 2	Instrucción 2	Instrucción 2
Instrucción 3	Instrucción 3	Instrucción 3
- - - - -	- - - - -	- - - - -
- - - - -	- - - - -	- - - - -
- - - - -	- - - - -	- - - - -
- - - - -	- - - - -	- - - - -
Instrucción N	Instrucción N	Instrucción N

Al diseñar la solución en módulos, la programación estructurada permitía solucionar problemas más grandes y más complejos de una mejor forma que la que antes se usaba.

Programación modular

La programación modular tuvo sus inicios a fines de la década de 1970 y principios de la de 1980. El lenguaje de programación que se utilizó fue MODULA 2 y emergió el concepto de encapsulación (en un módulo o paquete se encapsulaban los datos y las funciones que los manipulaban).

Programación con abstracción de datos

La programación con abstracción de datos se generó en la década de 1980. El lenguaje de programación que se utilizó fue ADA. Con éste emergió el concepto de Tipos Abstractos de Datos (TAD).

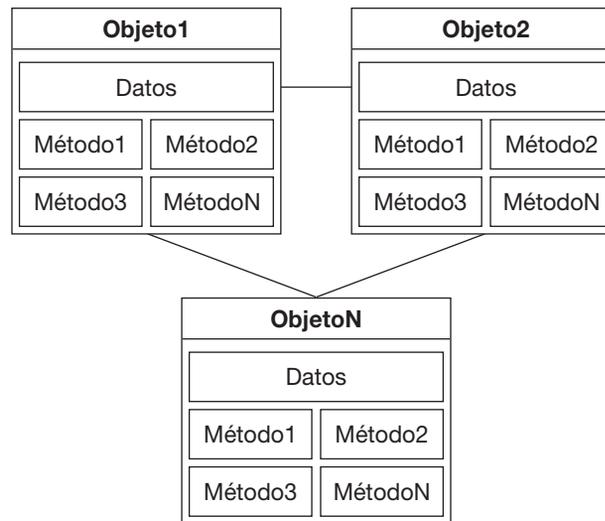
Programación orientada a objetos

El concepto de la programación orientada a objetos sólo se puso en boga a finales de la década de 1980 y principios de la de 1990, a pesar de que ya se había generado muchos años antes. A este tipo de programación la caracterizan los conceptos Objetos, Clases, Encapsulación, Herencia y Polimorfismo. Los principales lenguajes de programación que se utilizan son: C++, Java y C# y las técnicas de diseño que se utilizan son Booch, Rumbaugh, Jacobson, Yourdon, UML (Unified Modeling Language), entre otras.

La estructura general o arquitectura de un programa consiste en un conjunto de objetos, y cada objeto se compone por datos y un conjunto de métodos, donde cada método (que es equivalente al concepto de módulo en la programación estructurada) está formado por un conjunto de instrucciones, como se muestra a continuación:



Al diseñar la solución en módulos, la programación estructurada permitía solucionar problemas más grandes y más complejos de una mejor forma que la que antes se usaba.



La programación orientada a objetos permite manejar mejor la complejidad de los programas y la reutilización de código porque permite una mayor pulverización o segmentación de los programas a través de los objetos de una forma más eficiente que como anteriormente se hacía con la programación estructurada.

La programación orientada a objetos permite manejar mejor la complejidad de los programas y la reutilización de código porque permite una mayor pulverización o segmentación de los programas a través de los objetos de una forma más eficiente que como anteriormente se hacía con la programación estructurada.

1.3 El proceso de programación

Para desarrollar un programa de computadora se requiere seguir el proceso de programación, el cual consiste de una serie de pasos que comienza con la definición del problema y conduce a la elaboración e implantación del programa que lo soluciona. A continuación se describen los pasos que se deben seguir.

1. Definición del problema

Este proceso inicia cuando surge la necesidad de resolver algún problema mediante la computadora. Para empezar, se debe de identificar el problema y comprender la utilidad de la solución que se alcance. Se debe tener una visión general del problema estableciendo las condiciones iniciales (los puntos de partida) y, además, los límites del problema, es decir, dónde empieza y dónde termina. Por ejemplo, si tenemos que calcular el sueldo de un empleado, la solución que se logre permitirá obtener la cantidad que debe pagársele y servirá precisamente para pagarle. La situación anterior es una actividad de pago de sueldos y parte de que el empleado tiene algunos atributos como su nombre, el tiempo trabajando y el sueldo que percibe por unidad de tiempo dedicada a su labor.

2. Análisis del problema

A continuación es necesario entender en detalle el problema en cuestión para obtener una radiografía en términos de los DATOS disponibles como materia prima y definir el PROCESO necesario para convertir los datos en la INFORMACIÓN requerida.

La *primera etapa* consiste en definir los resultados esperados, es decir, la INFORMACIÓN que deberá producirse como salida. Respecto al problema de pago de sueldos, se requiere que emita como salida un cheque que contenga dos datos:

Nombre del empleado, que lo identifica.

Sueldo que debe pagársele por su trabajo.

La *segunda etapa* consiste en identificar los DATOS que se tienen como materia prima y que constituirán la entrada del programa. En este ejemplo tenemos los datos:

Nombre del empleado

Número de horas trabajadas

Cuota por hora

La *tercera etapa* tiene como finalidad determinar el PROCESO necesario para convertir los datos de entrada en la información que se tendrá como salida. Volviendo al ejemplo, puesto que se requieren dos datos de salida, determinemos el proceso de la siguiente manera:

a. ¿Cómo se obtiene el nombre del empleado?

No implica ningún cálculo, pues es un dato que se obtiene como entrada y que no se modifica.

b. ¿Cómo se obtiene el sueldo?

El sueldo es un dato que no existe como entrada, pero se calcula con la fórmula: $\text{sueldo} = \text{número horas trabajadas} \times \text{cuota por hora}$.

En este momento ya se tiene una comprensión clara del problema y podemos avanzar hacia el siguiente paso.

3. Diseño del programa

Durante este paso se procede a diseñar la lógica para la solución al problema a través de dos procesos:

Elaboración del algoritmo: Se diseña el algoritmo de la solución al problema; es decir, se estructura la secuencia lógica de pasos que la computadora deberá seguir para solucionar el problema utilizando alguna técnica de diseño de algoritmos como pseudocódigo, diagramas de flujo (en desuso con la programación estructurada), diagramas Warnier, diagramas Chapin, etcétera. Equiparando esta actividad con la construcción de una casa, equivale a diseñar el plano arquitectónico.

Prueba de escritorio: Se simula el funcionamiento del algoritmo con datos propios respecto al problema y se comprueban a mano los resultados a fin de validar la correcta operación del algoritmo. Si quedamos satisfechos con los resultados de la prueba, habremos agotado este punto, pero en caso contrario se deberá modificar el algoritmo y posteriormente volverlo a probar hasta que esté correcto. Es posible que se deba retroceder a cualquier paso precedente.

En este momento se tiene ya diseñada la solución al problema y estamos listos para pasar al siguiente punto.

4. Codificación del programa

En este paso se procede a codificar el programa en el lenguaje de programación que vayamos a utilizar. Este proceso es sumamente sencillo; dado que ya tenemos diseñado el programa, sólo nos concretamos a convertir las acciones del algoritmo en instrucciones de computadora. El programa codificado debe editarse, compilarse, probarse y depurarse; es decir, se ejecuta para verificar su buen funcionamiento y se hacen las correcciones o los ajustes pertinentes hasta que quede correcto. Si aparecen errores difíciles de corregir en este paso, quiere decir que debemos retroceder al paso 3 o al paso 2.

Para que un programa pueda ser entendido y ejecutado por la computadora, debe estar en lenguaje máquina o código objeto. El programa que nosotros hacemos en papel a lápiz o pluma debe ser traducido por un **compilador** a código asequible para la máquina mediante la **compilación**. El proceso de compilación es el siguiente: una vez que tenemos codificado el programa en papel, debe ser introducido mediante el proceso de **edición**, para lo cual se utiliza un **editor** que nos permite crear un archivo en el cual introducimos el **programa fuente** con las instrucciones que nosotros elaboramos en el lenguaje que estemos utilizando en este momento. El programa fuente es sometido al proceso de **compilación**, en el que mediante un compilador (traductor del lenguaje) se traduce instrucción por instrucción a código objeto, creándose un archivo con el **programa objeto**, entendible directamente por la máquina. Si en el proceso de traducción se encuentra algún error, se suspende el proceso; el programador debe corregir el error en el programa fuente y luego someterlo de nuevo al proceso de compilación.

Una vez que el proceso de compilación ha terminado con éxito, se tiene el programa objeto, el cual puede ser ejecutado por la computadora, que seguirá las instrucciones paso a paso, llevando a cabo las acciones que se le indican y emitiendo los resultados correspondientes. Si éstos no son satisfactorios o existen errores, el proceso de programación debe ser repetido desde alguno de los pasos precedentes.

5. Implantación del programa

Una vez que el programa está correcto, se instala y se pone a funcionar, entrando en operación normalmente dentro de la situación específica para la que se desarrolló. Debe ser supervisado continuamente para detectar posibles cambios o ajustes que sea necesario realizar.

6. Mantenimiento del programa

Un programa que está en operación, por un lado, podría presentar errores, los cuales deben corregirse. Por otro lado, podría requerir cambios o ajustes en sus datos, proceso o información; esto implica que eventualmente necesitará mantenimiento para adecuarlo a los cambios que le imponga la dinámica cambiante de las empresas o de los problemas.

Lo anterior nos sitúa en una dinámica infinita, ya que si surge la necesidad de darle mantenimiento tendremos que regresar a algún paso precedente: al 4, al 3, al 2 o al 1, para definir de nuevo el problema.

1.4 El algoritmo

En el proceso de programación hay un paso que es crucial a la hora de desarrollar un programa: el diseño del programa; en otras palabras, diseñar o elaborar el algoritmo de la solución.

El *algoritmo* es una secuencia ordenada y cronológica de pasos que llevan a la solución de un problema o a la ejecución de una tarea o actividad.

Los pasos del algoritmo deben tener las siguientes características:

- Ser simples, claros, precisos, exactos
- Tener un orden lógico
- Tener un principio y un fin

Aplicando el concepto de algoritmo a situaciones de nuestra vida cotidiana, tenemos como ejemplos de algoritmos las señas para encontrar una dirección, las recetas de cocina, los planos de construcción, las instrucciones para armar o utilizar un juguete, etcétera.

Cuando diseñemos algoritmos deberemos considerar que los pasos cumplan con las características antes mencionadas.

A continuación aplicaremos estos conceptos en una situación de nuestra vida cotidiana, a efectos de resaltar las características de los pasos de todo algoritmo. Posteriormente, en los capítulos subsecuentes estaremos aplicándolos al desarrollo de algoritmos que deberá ejecutar la computadora.

Ejercicios

Elaborar un algoritmo para que guíe a una persona normal (estudiante de secundaria, preparatoria o profesional) a *cambiar un foco fundido*, considerando que algún foco de nuestra casa (sala, comedor, baño, recámara, etc.) está fundido.

Piense en los pasos que deberá seguir...

Algoritmo *Cambiar foco fundido*:

1. Quitar el foco fundido
2. Colocar el foco nuevo
3. Fin

Si usted pensó en pasos que no coinciden exactamente con éstos pero sí conducen de manera efectiva a ejecutar el cambio del foco, entonces estará correcto, como es el caso de este algoritmo. Si bien es cierto que son pocos pasos, si usted los entiende y en consecuencia los puede ejecutar para lograr cambiar el foco, entonces está correcto.

Ahora bien, supóngase que estamos tratando de entrenar a un robot para que haga la tarea. En tal caso no funcionará el algoritmo; tendremos que ser más específicos y claros tomando en cuenta las capacidades elementales del robot.

Preguntémonos qué sabe hacer el robot y con base en ello elaboremos el algoritmo.

Capacidades del robot:

- Colocar la escalera
Suponemos que se tiene una escalera especial para que la maniobre el robot. Sabe traer la escalera desde su lugar y colocarla debajo del foco fundido.
- Subir a la escalera
Sabe subir por los peldaños de la escalera hasta alcanzar el foco fundido o el lugar del foco.
- Quitar el foco fundido
Sabe girar el foco a la izquierda hasta que salga.
- Obtener foco de repuesto
Suponemos que se tiene una caja con focos de repuesto de todas las medidas y que el robot sabe dónde se encuentra la caja y puede obtener un foco nuevo y en buenas condiciones igual al que traiga en su brazo.
- Colocar el foco de repuesto
Sabe colocar y girar el foco de repuesto a la derecha hasta que esté apretado.
- Bajar de la escalera
Sabe bajarse de la escalera hasta estar en el piso.
- Guardar la escalera
Sabe guardar la escalera en el lugar correspondiente.
- Fin
Sabe que ha terminado de ejecutar la tarea.

A continuación tenemos el algoritmo:

Algoritmo *Cambiar foco fundido*:

1. Colocar la escalera
2. Subir a la escalera
3. Quitar el foco fundido
4. Bajarse de la escalera
5. Obtener foco de repuesto
6. Subirse a la escalera
7. Colocar el foco de repuesto
8. Bajar de la escalera
9. Guardar la escalera
10. Fin

Ahora pensemos en otro robot que tiene las siguientes capacidades:

- Colocar la escalera
Sabe traer la escalera desde su lugar y colocarla debajo del foco fundido.
- Subir un peldaño
Sabe subir un peldaño de la escalera y sabe detectar cuándo alcanza el foco o el lugar del foco.
- Dar vuelta a la izquierda
Sabe girar el foco una vuelta a la izquierda y detectar cuándo sale.

- Obtener foco de repuesto.
Suponemos que se tiene una caja con focos de repuesto de todas las medidas, y que el robot sabe dónde se encuentra la caja y puede obtener un foco nuevo y en buenas condiciones igual al que traiga en su brazo.
- Dar vuelta a la derecha
Sabe girar el foco una vuelta a la derecha y detectar cuándo está apretado.
- Bajar un peldaño
Sabe bajar un peldaño de la escalera y sabe detectar cuándo está en el piso.
- Guardar la escalera.
Sabe guardar la escalera en el lugar correspondiente.
- Fin.
Sabe que ha terminado de ejecutar la tarea.

Es obvio que el algoritmo que hicimos anteriormente no serviría para este nuevo robot. Es necesario hacer otro algoritmo considerando las capacidades de este robot.

A continuación tenemos el algoritmo:

Algoritmo *Cambiar foco fundido*:

1. Colocar la escalera
2. Repetir
 - Subir un peldaño
 - Hasta alcanzar el foco
3. Repetir
 - Dar vuelta a la izquierda
 - Hasta que el foco salga
4. Repetir
 - Bajar un peldaño
 - Hasta estar en el piso
5. Obtener foco de repuesto
6. Repetir
 - Subir un peldaño
 - Hasta alcanzar el lugar del foco
7. Repetir
 - Dar vuelta a la derecha
 - Hasta que el foco este apretado
8. Repetir
 - Bajar un peldaño
 - Hasta estar en el piso
9. Guardar la escalera
10. Fin

Todas las actividades que llevamos a cabo los seres humanos son algoritmos que hemos aprendido a seguir. Caminar y lavarse los dientes, por ejemplo, son secuencias lógicas de pasos. La civilización está basada en el orden de las cosas y de acciones; éstas se organizan conforme a secuencias lógicas, y a esto se le llama *programación*.



Cuando se diseña un algoritmo se anotan paso a paso, en secuencia, las acciones que se ejecutarán. En ocasiones hay que repetir uno o varios pasos cierto número de veces (14 por ejemplo); en tal caso tenemos que controlar el primer paso, el segundo, el tercero, y así sucesivamente hasta el paso catorce para que pueda terminar el proceso. Esto se conoce como ciclo repetitivo. En otras ocasiones tenemos que llegar a un resultado partiendo de dos o más situaciones. En este caso debemos tomar en cuenta, por un lado, cómo se llega desde una parte y, por el otro, cómo se llegaría desde la otra parte, la alternativa; en estas circunstancias se utiliza la selección.

Durante el desarrollo de algoritmos para computadora es necesario idear los pasos que la máquina deberá seguir para realizar alguna tarea o actividad o para resolver algún problema de nuestra vida cotidiana. Cuando se presenta el problema de realizar cierta tarea, debemos efectuar diversas actividades subyacentes tales como hacer preguntas, buscar objetos conocidos o dividir el problema en partes. A menudo hacemos todo eso inconscientemente, pero al elaborar algoritmos para solucionar problemas con la computadora debemos detallar esas actividades de manera explícita.

Si se nos solicita una tarea verbalmente, por lo general hacemos preguntas hasta que quede claro lo que debemos hacer. Por ejemplo, preguntamos cuándo, por qué, dónde y otros aspectos, hasta que la tarea queda completamente especificada. Si las instrucciones están por escrito podemos anotar preguntas en los márgenes, subrayar palabras, resaltar frases u oraciones, o indicar de alguna otra forma que la idea no está clara y necesita precisión.

Si la tarea nos la imponemos nosotros mismos, puede ser que la pregunta se plantee a nivel subconsciente. Algunas preguntas comunes en el contexto de la programación son:

- ¿Qué datos tenemos para trabajar y cuál es su apariencia?
- ¿Cuántos datos hay?
- ¿Cómo sabremos que ya están procesados todos los datos?
- ¿Cuál debe ser el formato de la salida?
- ¿Cuántas veces debe repetirse el proceso?
- ¿Qué condiciones especiales de error pueden presentarse?

Nota: Cuando se diseña un algoritmo se anotan *paso a paso*, en secuencia, las acciones que se ejecutarán. En ocasiones hay que repetir uno o varios pasos cierto número de veces (14 por ejemplo); en tal caso tenemos que controlar el primer paso, el segundo, el tercero, y así sucesivamente hasta el paso catorce para que pueda terminar el proceso. Esto se conoce como *ciclo repetitivo*.

En otras ocasiones tenemos que llegar a un resultado partiendo de dos o más situaciones. En este caso debemos tomar en cuenta, por un lado, cómo se llega desde una parte y, por el otro, cómo se llegaría desde la otra parte, la alternativa; en estas circunstancias se utiliza la *selección*.

1.5 Ejercicios propuestos

1. Elaborar un algoritmo para hacer palomitas de maíz en una cacerola puesta al fuego usando sal y maíz.
2. Elaborar un algoritmo que permita cambiar un vidrio roto de una ventana.
3. Elaborar un algoritmo para cambiar un neumático pinchado.
4. Elaborar un algoritmo para hacer una llamada telefónica.
5. Definir “su” robot, es decir, lo que sabe hacer tomando como referencia lo que usted hace, y elaborar un algoritmo que lo guíe desde que se despierta por la mañana hasta que llega a la escuela, al trabajo o a algún otro lugar.

1.6 Resumen de conceptos que debe dominar

- Qué es una computadora y sus componentes principales
- Qué es un programa y cuáles son las características de un buen programa
- Cuáles son las características de los lenguajes de programación
- Cuáles son los paradigmas de programación
- Cuáles son los pasos del proceso de programación
- Qué es el algoritmo y cuáles son sus características

1.7 Contenido de la página Web de apoyo

El material marcado con asterisco () sólo está disponible para docentes.*

1.7.1 Resumen gráfico del capítulo

1.7.2 Autoevaluación

1.7.3 Power Point para el profesor (*)

Elementos para solucionar problemas en seudocódigo

Contenido

- 2.1 Estructuras de datos
 - 2.1.1 Tipos de datos
 - 2.1.2 Variables
 - 2.1.3 Constantes
- 2.2 Operaciones primitivas elementales
 - 2.2.1 Declarar
 - 2.2.2 Lectura de datos (Entrada)
 - 2.2.3 Operaciones aritméticas fundamentales
 - 2.2.4 Escritura de datos (Salida)
- 2.3 Estructuras de control
- 2.4 Resumen de conceptos que debe dominar
- 2.5 Contenido de la página Web de apoyo
 - El material marcado con asterisco (*) sólo está disponible para docentes.*
 - 2.5.1 Resumen gráfico del capítulo
 - 2.5.2 Autoevaluación
 - 2.5.3 Power Point para el profesor (*)

Objetivos del capítulo

- Estudiar los elementos para diseñar algoritmos en seudocódigo: las estructuras de datos, datos numéricos enteros y reales, datos carácter, cadena de caracteres, etcétera.
- Aprender las operaciones primitivas elementales: declaraciones de variables, de constantes y de tipos, lectura de datos (entrada), escritura de datos (salida).
- Describir el uso de las operaciones aritméticas fundamentales (sumar, restar, multiplicar y dividir).
- Conocer las estructuras de control: la secuenciación, la selección (if-then, if-then-else, switch) y la repetición (do... while, for, while).

Competencias

- Competencia general del capítulo
 - *Describir los elementos para solucionar problemas en seudocódigo.*
- Competencias específicas del capítulo
 - Describe los elementos para solucionar problemas en seudocódigo; estructuras de datos: variables, constantes, tipos de datos básicos entero, real, cadena y carácter.
 - Enuncia los elementos para solucionar problemas en seudocódigo; operaciones primitivas elementales: Declarar variables, solicitar datos, leer datos, hacer cálculos e imprimir información.
 - Enuncia las estructuras de control: secuenciación, selección y repetición.

Introducción

Con el estudio del capítulo 1, usted ya conoce los principales conceptos acerca de las computadoras: la computadora y sus componentes, el programa, el lenguaje de programación y la programación; las características de un buen programa; la evolución de los paradigmas de programación; los pasos que integran el proceso de programación; lo que es un algoritmo y sus características.

En este capítulo el lector aprenderá los elementos para diseñar algoritmos enseudocódigo. Se define queseudocódigo es una técnica para diseñar programas o elaborar algoritmos. Se explica que las estructuras de datos son las formas como la computadora representa los datos internamente y qué son los tipos de datos simples: datos numéricos enteros y reales, datos carácter, datos cadena de caracteres y booleanos.

Se enseña qué son, cómo se representan y cómo se usan enseudocódigo las operaciones primitivas elementales, que son: declaraciones de variables, de constantes y de tipos, lectura de datos (entrada), escritura de datos (salida).

También se especifica qué son, cómo se representan y cómo se usan las operaciones aritméticas fundamentales (sumar, restar, multiplicar y dividir), mediante las cuales se hacen cálculos.

Asimismo, se expone que las estructuras de control son las formas lógicas con las que funciona la computadora y mediante las cuales se dirige su funcionamiento, y se les da orden lógico a las operaciones primitivas elementales que actúan sobre los datos.

Se explica que las estructuras de control son la *secuenciación*, la *selección* —que a su vez tiene tres formas: simple (if-then), doble (if-then-else) y múltiple (switch)— y la *repetición*, que también tiene tres formas: do...while, for y while.

En el siguiente capítulo se estudia la estructura de control: la *secuenciación*, qué es, cómo se representa y cómo se usa enseudocódigo.

Ya sabemos lo que es un algoritmo y conocemos la manera de diseñar soluciones para problemas de la vida cotidiana. Ahora aplicaremos estos conceptos a la elaboración de algoritmos para programar computadoras, utilizando elseudocódigo.

Elseudocódigo es una técnica para diseño de programas (algoritmos) que permite definir las estructuras de datos, las operaciones que se aplicarán a los datos y la lógica que tendrá el programa de computadora para solucionar un determinado problema. Utiliza unseudolenguaje muy parecido a nuestro idioma, pero que respeta las directrices y los elementos de los lenguajes de programación.

En este capítulo analizaremos las capacidades básicas o elementales de una computadora, ya que es indispensable conocerlas para poder desarrollar algoritmos.

Los elementos básicos para solucionar problemas son las estructuras de datos, las operaciones primitivas elementales y las estructuras de control. A continuación explicaremos cada una de ellas y su forma de representación enseudocódigo.

2.1 Estructuras de datos

Las estructuras de datos son las formas de representación interna de datos de la computadora mediante las que se representa cualquier situación en ésta; es decir, son los tipos de datos que maneja la máquina. Por ejemplo, podemos representar a un trabajador mediante los datos nombre del empleado, número de horas trabajadas, cuota por hora, etcétera.

2.1.1 Tipos de datos

Los tipos de datos más comunes son los numéricos (entero y real), cadena de caracteres (alfabéticos o alfanuméricos), carácter y boolean, cada uno de los cuales puede manejarse como una constante o como una variable. A continuación se describe cada tipo.

Datos numéricos

Estos datos se dividen a su vez en ENTEROS Y REALES. Los ENTEROS son los números que no contienen componentes fraccionarios y, por tanto, no incluyen el punto decimal; pueden ser positivos o negativos, como por ejemplo 450, -325, 4, ó -4. Se representan como Entero. Los REALES son los números que contienen una parte fraccionaria y, por tanto, incluyen el punto decimal; pueden ser positivos o negativos, como por ejemplo 465.0, 42.325, 800.02, -24.5, ó -42.3. Se representan como Real.

Nota 1: Estos conceptos de Entero y Real son acuñados y válidos en el ámbito computacional y se diferencian en que el tipo Real contiene punto y fracción decimal, mientras que el tipo Entero no contiene punto ni parte fraccionaria.

Nota 2: Los lenguajes de programación como C, C++ y otros proporcionan más de un tipo de dato de tipo entero y real. Por ejemplo, Java tiene los tipos de datos: byte, short, int y long, y todos son de tipo entero; asimismo, proporciona los tipos: float y double, y ambos son de tipo real. Sin embargo, aquí en pseudocódigo sólo nos interesa manejar Entero y Real; en el momento en el que nos corresponda traducir los algoritmos a instrucciones en un lenguaje sí usaremos la variante más apropiada al problema que estemos resolviendo, de acuerdo a los rangos de valores de los datos.

Datos cadena de caracteres

Los datos cadena de caracteres están compuestos por una hilera (cadena) de caracteres alfabéticos, numéricos y especiales, que sirven para representar y manejar datos tales como nombres de personas o de empresas, descripciones de artículos o productos de inventarios, direcciones de personas o empresas, entre otros. Se representa como Cadena.

Este tipo de datos también se conoce como alfabético o alfanumérico, ya que su contenido —sea cual fuere— siempre se considera como una serie de caracteres; los valores (constantes) de este tipo se encierran entre comillas (“ ”). Por ejemplo:

```
"Universidad de Sonora"
"Socorro Román Maldonado"
"Dr. de la Torre y Vicente Guerrero # 75"
"Guamúchil, Sinaloa, México"
"Tornillo X25 ¾"
"Bicicleta Cross Z-47"
"Cualquier otra cosa que sea una constante literal"
```



Estos conceptos de Entero y Real son acuñados y válidos en el ámbito computacional y se diferencian en que el tipo Real contiene punto y fracción decimal, mientras que el tipo Entero no contiene punto ni parte fraccionaria. Los lenguajes de programación como C, C++ y otros proporcionan más de un tipo de dato de tipo entero y real. Por ejemplo, Java tiene los tipos de datos: byte, short, int y long, y todos son de tipo entero; asimismo, proporciona los tipos: float y double, y ambos son de tipo real. Sin embargo, aquí en pseudocódigo sólo nos interesa manejar Entero y Real; en el momento en el que nos corresponda traducir los algoritmos a instrucciones en un lenguaje sí usaremos la variante más apropiada al problema que estemos resolviendo, de acuerdo a los rangos de valores de los datos.

Datos carácter

El tipo de dato carácter utiliza un byte y puede almacenar un carácter: cualquier carácter válido para la computadora según el código ASCII. Se representa como Carácter y los valores (constantes) de este tipo se encierran entre apóstrofes (' '). Por ejemplo: 's', 'n', 'S', 'N'.

Datos boolean

Este tipo de datos está compuesto por los valores True (verdadero) y False (falso); no puede ser leído o escrito, sólo asignado, y es útil para plantear cierto tipo de condiciones en el manejo de las estructuras lógicas de control. Se representa como Boolean.

2.1.2 Variables

Las variables sirven para representar y manejar datos. Todo dato que vaya a ser introducido a la computadora y todo dato que vaya a ser generado o calculado a partir de otros datos para obtener algún resultado tienen que identificarse y manejarse en forma de variable. Una variable tiene las siguientes características:

a) Nombre. Es el identificador de la variable que servirá para referenciarla.

Las reglas para asignar nombres de variables y otros identificadores como métodos, funciones, clases, objetos, etcétera, son:

1. Se pueden utilizar combinaciones de letras mayúsculas y minúsculas (A...Z, a...z); dígitos (0...9) y el símbolo de subrayado (o guión bajo _ aunque por cuestiones de estilo no lo usaremos en este libro).
2. El nombre debe iniciar con una letra.
3. Es conveniente que la longitud no pase de 20 caracteres.
4. No debe ser palabra reservada (como if, then, else, for, do, while, etc.).

Nota: Los nombres de las variables y otros identificadores deben definirse de tal manera que indiquen lo que están representando, ya sea datos, funciones, métodos, clases u objetos; es decir, deben ser un mnemónico de lo que representan. Por ejemplo, si se va a manejar el dato del nombre del empleado, se podría usar Nombre o NombreEmpleado. El dato de horas trabajadas se podrá nombrar HorasTrabajadas. El dato de cuota por hora se podría manejar como CuotaHora. Usted, como programador, puede adoptar su propio estilo para dar nombres a variables. Sin embargo, considere lo siguiente:

- Hay autores de lenguajes de programación que usan sólo letras mayúsculas. Ejemplos: NOMBRE, HORASTRABAJADAS, CUOTAHORA, X, Y, Z.
 - Otros autores usan sólo letras minúsculas. Ejemplos: nombre, horastrabajadas, cuotahora, x, y, z.
 - Otros utilizan tanto mayúsculas como minúsculas. Ejemplo: CuotaHora.
5. Estilo de programación. En este libro se utilizará el estilo que maneja el lenguaje Java y los demás lenguajes modernos. Dicho estilo es el siguiente:
 - a) En nombres de clases:
 - La primera letra en mayúscula y las demás en minúsculas. Ejemplos: Empleado, Alumno, Cliente.



Los nombres de las variables y otros identificadores deben definirse de tal manera que indiquen lo que están representando, ya sea datos, funciones, métodos, clases u objetos; es decir, deben ser un mnemónico de lo que representan. Por ejemplo, si se va a manejar el dato del nombre del empleado, se podría usar Nombre o NombreEmpleado. El dato de horas trabajadas se podrá nombrar HorasTrabajadas. El dato de cuota por hora se podría manejar como CuotaHora. Usted, como programador, puede adoptar su propio estilo para dar nombres a variables.

- Si se juntan varias palabras, se aplica el punto anterior a cada una de las palabras. Ejemplos: EmpleadoConfianza, EmpleadoSindicalizado, EmpleadoPorHoras.
- b) En nombres de variables, funciones, objetos, métodos y otros:
 - Si el elemento se compone por una sola palabra, se usan puras minúsculas. Ejemplos: nombre, sueldo, x, y, z, x1, y2, z1, x25, y11, z12.
- Si se juntan varias palabras, de la segunda palabra en adelante, la inicial es mayúscula y las demás, minúsculas. Ejemplos: nombreEmpleado, cuotaPorHora, cuotaHora, sueldoEmpleado, objetoEmpleado, objetoAlumno, calcularSueldo.

Nota: Algunos lenguajes de programación no admiten letras acentuadas; otros sí las permiten. Para mantener esta metodología lo más general posible, aquí no usaremos letras acentuadas; por lo tanto, no emplearemos nombres de variables como: opción, producción, tamaño, año, número. Los nombres que manejaremos en su lugar serán: opcion, produccion, tamaño, anio, numero. Observe que en lugar de la ñ se coloca *ni* para evitar nombres de variables altisonantes. Sin embargo, si el lenguaje que usted va a usar para codificar los algoritmos que esté diseñando lo permite, entonces puede usar letras acentuadas.

Nota: Cuando se va a dar nombre a un dato como el del número de horas trabajadas, el nombre más entendible es numeroDeHorasTrabajadas. Sin embargo, es demasiado grande, entonces lo que debemos hacer es “abreviar” algunas de las palabras. Podría quedar: numHrsTrab (éste es un buen nombre, porque se entiende lo que representa y no es muy largo). Con nombres como éste debe tenerse cuidado, porque un error muy común de quienes recién se inician en este tema es ponerle puntos (como cuando se abrevia aplicando las reglas gramaticales del español). Para esta situación, podrían definir num.Hrs.Trab., lo que es erróneo porque en nombres de identificadores no se admite el punto. Otro problema común en los principiantes es poner espacios, por ejemplo: num Hrs Trab, lo que constituye también un error.

b) Contenido. Se puede considerar que toda variable tiene una “casilla” donde se almacena el valor que toma en cada ocasión. Esto es similar a tener un casillero donde las casillas sirven para guardar objetos y tienen su respectiva identificación que permite referenciarlas. Por ejemplo:

CASILLA	
nombreEmp	
cuotaHora	
horasTrab	

c) Tipo de datos. Toda variable debe estar asociada a un tipo de datos Entero, Real, Cadena, etcétera y, de acuerdo con esto, sólo podrá tomar valores válidos para el tipo de dato definido.



Algunos lenguajes de programación no admiten letras acentuadas; otros sí las permiten. Para mantener esta metodología lo más general posible, aquí no usaremos letras acentuadas; por lo tanto, no emplearemos nombres de variables como: opción, producción, tamaño, año, número. Los nombres que manejaremos en su lugar serán: opcion, produccion, tamaño, anio, numero. Observe que en lugar de la ñ se coloca *ni* para evitar nombres de variables altisonantes. Sin embargo, si el lenguaje que usted va a usar para codificar los algoritmos que esté diseñando lo permite, entonces puede usar letras acentuadas.



Cuando se va a dar nombre a un dato como el del número de horas trabajadas, el nombre más entendible es numeroDeHorasTrabajadas. Sin embargo, es demasiado grande, entonces lo que debemos hacer es “abreviar” algunas de las palabras. Podría quedar: numHrsTrab (éste es un buen nombre, porque se entiende lo que representa y no es muy largo). Con nombres como éste debe tenerse cuidado, porque un error muy común de quienes recién se inician en este tema es ponerle puntos (como cuando se abrevia aplicando las reglas gramaticales del español). Para esta situación, podrían definir num.Hrs.Trab., lo que es erróneo porque en nombres de identificadores no se admite el punto. Otro problema común en los principiantes es poner espacios, por ejemplo: num Hrs Trab, lo que constituye también un error.

2.1.3 Constantes

Las constantes son valores específicos, y en consecuencia invariables. Por ejemplo, constantes de tipo entero son: 5, 10, -56 y 20; constantes de tipo real son: 3.1416, 40.5, -1.5 y 2.718; constantes de tipo cadena de caracteres son: “Universidad de Sonora”, “Tornillo X25 ¾”, “Rosales # 245 Sur”; constantes de tipo carácter son: ‘s’, ‘n’, ‘S’, ‘N’; y constantes de tipo boolean son True y False.

2.2 Operaciones primitivas elementales

Las operaciones primitivas elementales son las acciones básicas que la computadora puede ejecutar. A continuación estudiaremos las características de cada una.

2.2.1 Declarar

Es una acción que se considera como no ejecutable, ya que mediante ésta se declara o define el entorno en el que se ejecutará el algoritmo (programa). En esta parte se declaran las variables, constantes, tipos, objetos, etcétera. A continuación se explican.

a) Constantes. Como ya se explicó, las constantes son valores específicos. Sin embargo, se pueden declarar constantes simbólicas, es decir, mediante un identificador, con el siguiente formato:

```
Constantes
  NOMCONSTANTE = Valor
```

En donde:

Constantes	Identifica la acción que se va a realizar, la cual es declaración de constantes simbólicas.
NOMCONSTANTE	Es el identificador de la constante.
Valor	Es el valor que se asigna como constante.

Ejemplos:

```
Declarar
Constantes
  PI = 3.14159265
  CIEN = 100
  COMENTARIO1 = "Aprobado"
  COMENTARIO2 = "Reprobado"
```



Los identificadores o nombres de constantes se asignan usando letras mayúsculas y números.

Nota: Los identificadores o nombres de constantes se asignan usando letras mayúsculas y números.

b) Tipos. Se declaran nuevos tipos de datos. El formato es:

```
Tipos
  NombreTipo = (Valor1, Valor2, ..., ValorN)
```

En donde:

NombreTipo Es el nombre del nuevo tipo de dato.

Valor1, Valor2, ..., ValorN
 Son los valores que integran el nuevo tipo de dato.

Ejemplo:

```
Declarar
Tipos
  Colores = (Negro, Blanco, Azul, Rojo, Amarillo)
Variables
  colFondo: Colores
```

Se ha declarado el tipo de dato `Colores`, y luego la variable `colFondo` del tipo `Colores`; es decir, la variable `colFondo` podrá tomar los valores definidos en el tipo `Colores`. Este tipo será utilizado más ampliamente con arreglos, registros y archivos en los capítulos correspondientes.

c) Variables. Esta acción permite representar en un programa las estructuras de datos necesarias para abstraer y representar en la computadora la situación real del problema o de la tarea que se manejará; en otras palabras, se declaran las variables necesarias mediante el siguiente formato:

```
Variables
  nomVariable1: Tipo de dato
  nomVariable2: Tipo de dato
  .
  .
  nomVariableN: Tipo de dato
```

En donde:

Variables Identifica la acción que se va a realizar, la cual es declaración de variables.

nomVariable1, nomVariable2, nomVariableN
 Son los nombres de las variables, de acuerdo a los lineamientos antes expuestos.

Tipo de dato Indica el tipo de dato que tendrá la variable; a saber: Entero, Real, Cadena, etcétera.

Ejemplo:

Para representar la situación de pago de sueldos que ya se ha mencionado, tendremos que hacer lo siguiente:

```
Declarar
Variables
    nombreEmp: Cadena
    horasTrab: Entero
    cuotaHora: Real
    sueldo: Real
```



En el tipo de dato Cadena puede indicarse la longitud en número de caracteres que le caben a la variable. En el ejemplo podría quedar así:
`nombreEmp: Cadena[30]`
 Es decir, a la variable `nombreEmp` le caben 30 caracteres.
 En este libro vamos a utilizar Cadena.

Nota: En el tipo de dato Cadena puede indicarse la longitud en número de caracteres que le caben a la variable. En el ejemplo podría quedar así:

```
nombreEmp: Cadena[30]
```

Es decir, a la variable `nombreEmp` le caben 30 caracteres.

En este libro vamos a utilizar Cadena.

d) Objetos. Aunque aquí también se pueden declarar objetos, en este libro lo haremos como se explica en el capítulo 9, ya que es a partir de ahí que los usaremos.

2.2.2 Lectura de datos (Entrada)

Esta operación nos permite introducir los datos a la computadora, es decir, introducir la materia prima para el proceso. La introducción de datos puede hacerse desde un teclado, lector de código de barras, un archivo en un disco, un ratón (mouse) o desde cualquier otro dispositivo de entrada. El dispositivo estándar de entrada es el teclado. El formato para leer datos es:

```
Leer nomVariable1, nomVariable2, nomVariableN
```

En donde:

Leer Indica la acción que se va a realizar, la cual es lectura de datos.

`nomVariable1, nomVariable2, nomVariableN`
 Son los nombres de las variables donde se leerán los datos.

Ejemplo:

Para la situación de pago de sueldos que venimos siguiendo se deben leer el nombre del empleado, el número de horas trabajadas y la cuota por hora:

```
Leer nombreEmp, horasTrab, cuotaHora
```

Esta acción espera a que se tecleen los datos correspondientes, los cuales se almacenarán en las variables especificadas. En virtud de que los nombres de las variables son mnemónicos con respecto a los datos que representan, se facilita la

comprensión de la acción. Con esta acción entendemos que se leerán el nombre, las horas trabajadas y la cuota por hora, y además que la lectura implica un proceso interactivo en el que se solicitan los datos en la pantalla. Ejemplo:

```
Teclee nombre del empleado:_
Teclee número de horas trabajadas:_
Teclee cuota por hora:_
```

Esta acción de lectura de datos puede hacerse de una forma más detallada acompañándola con una solicitud de los datos para enfatizar la interactividad entre la persona que introducirá los datos y la computadora. En tal caso la lectura queda así:

```
Solicitar nombre del empleado, número de horas trabajadas,
      cuota por hora
Leer nombreEmp, horasTrab, cuotaHora
```

También se podría leer así:

```
Solicitar nombre del empleado
Leer nombreEmp
Solicitar número de horas trabajadas
Leer horasTrab
Solicitar cuota por hora
Leer cuotaHora
```

Nota: Solicitar los datos es equivalente a imprimir en la pantalla haciendo la petición de que se introduzcan los datos. Se podría colocar así:

```
Imprimir "Teclee nombre del empleado"
Imprimir "Teclee número de horas trabajadas"
Imprimir "Teclee cuota por hora"
```

Nota: Cuando en una acción de solicitar vaya(n) alguna(s) palabra(s) con letra(s) acentuada(s), la(s) vamos a poner con la acentuación, como en `Solicitar número de horas trabajadas`, en la que número se puso con acento. Sin embargo, cuando el algoritmo se codifique en un lenguaje, si éste lo permite, se pone acentuada; si no, se pone sin acento. Hay lenguajes que en una instrucción como `print` admiten algo como "número" pero, cuando el programa funciona, en lugar de la `ú` se imprime un carácter raro; entonces se imprimirá `nmero` en lugar de número.

2.2.3 Operaciones aritméticas fundamentales

Estas operaciones permiten modificar la apariencia de los datos para generar información; en otras palabras, procesar los datos que entran como materia prima para convertirlos en información que saldrá de la computadora.



Quando en una acción de solicitar vaya(n) alguna(s) palabra(s) con letra(s) acentuada(s), la(s) vamos a poner con la acentuación, como en `Solicitar número de horas trabajadas`, en la que número se puso con acento. Sin embargo, cuando el algoritmo se codifique en un lenguaje, si éste lo permite, se pone acentuada; si no, se pone sin acento. Hay lenguajes que en una instrucción como `print` admiten algo como "número" pero, cuando el programa funciona, en lugar de la `ú` se imprime un carácter raro; entonces se imprimirá `nmero` en lugar de número.

Las operaciones aritméticas son:

+	Suma
-	Resta
*	Multiplicación
/	División real
\	División entera
Mod	Residuo de una división entera
=	Asignación

Mediante tales operaciones se forman expresiones aritméticas (fórmulas) para realizar los cálculos.

Formato:

```
Calcular variable = expresión
```

En donde:

variable	Es el nombre de la variable en la que se asignará el resultado de expresión.
expresión	Es un valor constante, una variable, una expresión algebraica o fórmula (para calcular un valor), el cual se le asigna a la variable que está a la izquierda del símbolo =.
=	Es el símbolo que indica la asignación del valor de expresión a la variable.

Ejemplo:

```
Calcular sueldo = horasTrab * cuotaHora
```

Explicación:

Se calcula `horasTrab * cuotaHora` y el resultado se coloca en `sueldo`.

Nota: La palabra `Calcular` puede omitirse. La expresión queda así:

```
sueldo = horasTrab * cuotaHora
```

Otros ejemplos:

<code>a = 1</code>	a toma el valor 1
<code>b = a + 1</code>	b toma el resultado de 2
<code>b = b - 1</code>	b toma el resultado de b - 1
<code>y = 5 / 2</code>	y toma el valor 2.5. División real (hasta decimales)
<code>z = 5 \ 2</code>	z toma el valor 2. División entera (no decimales)
<code>r = 5 Mod 2</code>	r toma el valor 1, es decir, el residuo.
<code>observacion = "Aprobado"</code>	observacion toma el valor "Aprobado"

La palabra `Calcular` puede omitirse. La expresión queda así:
`sueldo = horasTrab * cuotaHora`

Ejemplo:

Las expresiones aritméticas deben escribirse en una línea para que puedan introducirse a la computadora. Por ejemplo, la expresión:

$$N = \frac{X + Y}{Y - 1}$$

nosotros la entendemos perfectamente y la podríamos evaluar así:

$$\begin{aligned} \text{Si } X &= 5 \\ Y &= 3 \end{aligned}$$

luego haríamos la sustitución de valores y la resolveríamos:

$$N = \frac{5 + 3}{3 - 1} = \frac{8}{2} = 4$$

Sin embargo, la computadora no entiende este formato, por lo cual debemos escribir la expresión original en una línea para que la reconozca. Después de hacerlo nos queda de la siguiente forma:

$$n = (x + y) / (y - 1)$$

La computadora examina toda la expresión y va evaluando cada componente de acuerdo a cierto orden de precedencia que tienen las operaciones aritméticas, que es el siguiente:

1. Operaciones entre paréntesis.

Primero busca lo que está entre paréntesis y lo evalúa. En caso de haber paréntesis anidados, evalúa primero los más internos y luego prosigue con los externos.

2. Operaciones unarias.

A continuación evalúa las operaciones unarias, es decir, de cambio de signo. Ejemplo: a -4 le cambia el signo a 4.

3. Multiplicación y división.

Posteriormente evalúa las operaciones de multiplicación y división, las cuales tienen el mismo nivel de precedencia.

4. Suma y resta.

Estas operaciones se evalúan al final, ya que tienen el más bajo nivel de precedencia.

Nota 1: Cuando existen varias operaciones de un mismo nivel, se evalúan en orden de izquierda a derecha.

Nota 2: En el ejemplo se encerraron entre paréntesis las operaciones $(x + y)$ y $(y - 1)$ para cerciorarse de que se hiciera primero la suma, luego la resta y por último la división, conforme al planteamiento del problema original. Si la expresión se hubiera escrito $n = x + y / y - 1$ estaría incorrecta, porque la computadora primero haría la división de y / y , luego la suma y después la resta, lo que arrojaría un resultado incorrecto.



Cuando existen varias operaciones de un mismo nivel, se evalúan en orden de izquierda a derecha.

En el ejemplo se encerraron entre paréntesis las operaciones $(x + y)$ y $(y - 1)$ para cerciorarse de que se hiciera primero la suma, luego la resta y por último la división, conforme al planteamiento del problema original. Si la expresión se hubiera escrito $n = x + y / y - 1$ estaría incorrecta, porque la computadora primero haría la división de y / y , luego la suma y después la resta, lo que arrojaría un resultado incorrecto.

Otros ejemplos de conversión en una línea:

$$x = A+B - \frac{Y}{z} \quad x = a+b - y/z \quad \text{o} \quad x = a+b - (y/z)$$

$$W = \frac{A \cdot C}{D} + \frac{B}{C} \quad w = (a * c/d) + b/c$$

$$Z = \frac{T(Y-2) + R}{S} \quad z = (t * (y-2) + r)/s$$

$$X = \frac{CE - BF}{AE - BD} \quad x = ((c*e) - (b*f)) / ((a*e) - (b*d))$$

$$F = \frac{9}{5} C + 32 \quad f = 9/5 * c + 32$$

2.2.4 Escritura de datos (Salida)

Mediante la escritura damos salida a los datos de la computadora hacia un medio periférico como por ejemplo la pantalla de video, la impresora, disco u otro. La salida estándar es el video.

Formato:

```
Imprimir nomVariable1, nomVariable2, nomVariableN
```

En donde:

Imprimir

Identifica la acción de escritura.

nomVariable1, nomVariable2, nomVariableN

Son los nombres de las variables que contienen los datos que serán impresos.

Ejemplo:

En el ejemplo que venimos siguiendo se debe imprimir como resultado el nombre del empleado y su sueldo. Con la acción o instrucción:

```
Imprimir nombreEmp, sueldo
```

se indica que se imprimen el nombre y el sueldo, que es la información requerida en este problema. Por el momento no se hace énfasis en darle algún formato a la salida; sólo se indican los datos que salen en términos de las variables que los contienen,

aprovechando que estamos utilizando nombres mnemónicos. Cuando se analiza el problema es donde se define el formato de la salida, por ejemplo si será un cheque, un recibo o algún otro documento, y será en la codificación del programa cuando se le dé a la salida algún formato específico.

Nota: En caso de que se desee acompañar los datos con letreros, se hace poniéndolos entre comillas. Por ejemplo:

```
Imprimir "Nombre = ", nombreEmp, "Sueldo = ", sueldo
```

Explicación:

Se imprime el letrero Nombre = , luego se imprime el contenido de la variable nombreEmp, luego se imprime el letrero Sueldo = y por último se imprime el contenido de la variable sueldo.

Nota: Si se quiere indicar que la salida es hacia la impresora se puede colocar la palabra Impresora. Por ejemplo:

```
Imprimir Impresora, nombreEmp, sueldo
```



En caso de que se desee acompañar los datos con letreros, se hace poniéndolos entre comillas. Por ejemplo:
Imprimir "Nombre = ", nombreEmp, "Sueldo = ", sueldo



Si se quiere indicar que la salida es hacia la impresora se puede colocar la palabra Impresora. Por ejemplo:
Imprimir Impresora, nombreEmp, sueldo

2.3 Estructuras de control

Son las formas lógicas de cómo trabaja internamente la computadora, y es mediante éstas que se dirige su funcionamiento, es decir, se le da orden lógico a las operaciones primitivas elementales que actúan sobre los datos. Las estructuras de control son la secuenciación, la selección —que a su vez tiene tres formas: simple (if-then), doble (if-then-else) y múltiple (switch)— y la repetición, que también tiene tres formas: do...while, for, while. En los capítulos subsecuentes se estudiarán detalladamente cada una de estas estructuras.

2.4 Resumen de conceptos que debe dominar

- Estructuras de datos.
 - Tipos de datos en pseudocódigo: datos numéricos enteros y reales; datos carácter y cadena de caracteres. Los conceptos de constantes, variables y sus características.
- Operaciones primitivas elementales
 - Declarar variables, constantes y tipos
 - Lectura de datos (Entrada)
 - Operaciones aritméticas fundamentales (suma, resta, multiplicación y división)
 - Escritura de datos (Salida)
- Cuáles son las estructuras de control
 - Secuenciación
 - Selección (if-then, if-then-else, switch)
 - Repetición (do...while, for, while)

2.5 Contenido de la página Web de apoyo

El material marcado con asterisco () sólo está disponible para docentes.*

2.5.1 Resumen gráfico del capítulo

2.5.2 Autoevaluación

2.5.3 Power Point para el profesor (*)

Contenido

- 3.1 Nuestro primer problema
- 3.2 Estructura y diseño de un algoritmo
- 3.3 Nuestro primer algoritmo
- 3.4 Funciones matemáticas
- 3.5 Ejercicios resueltos
- 3.6 Ejercicios propuestos
- 3.7 Resumen de conceptos que debe dominar
- 3.8 Contenido de la página Web de apoyo
 - El material marcado con asterisco (*) sólo está disponible para docentes.*
 - 3.8.1 Resumen gráfico del capítulo
 - 3.8.2 Autoevaluación
 - 3.8.3 Programas en Java
 - 3.8.4 Ejercicios resueltos
 - 3.8.5 Power Point para el profesor (*)

Objetivos del capítulo

- Estudiar la estructura de control secuenciación.
- Aprender a utilizar las funciones matemáticas (seno, coseno, arco tangente, logaritmo natural, etc.).
- Aplicar lo aprendido en la solución de ejercicios resueltos y propuestos.

Competencias

- Competencia general del capítulo
 - *Analizar problemas y diseñar algoritmos que los solucionen aplicando la secuenciación.*
- Competencias específicas del capítulo
 - Explica la estructura de control secuenciación.
 - Diseña algoritmos para solucionar problemas del ámbito escolar y cotidiano de nivel básico, aplicando los conceptos básicos y la estructura de control secuenciación.
 - Elabora algoritmos aplicando funciones matemáticas.

Introducción

Después de haber estudiado el capítulo 2, usted ya domina los elementos para diseñar algoritmos en pseudocódigo: los tipos de datos, cómo declarar variables y constantes, cómo solicitar y leer datos, cómo hacer cálculos, cómo imprimir datos y, además, cuáles son las estructuras de control.

En este capítulo se inicia el estudio de las estructuras de control. Primero se aborda la estructura de control más elemental que es la *secuenciación*. Ésta es la capacidad lógica que tiene la computadora de ejecutar un conjunto de instrucciones secuencialmente, una a una, desde la primera hasta la última.

Se explica que los lenguajes de programación proporcionan funciones estándar, como son las funciones matemáticas seno, coseno, arco tangente, logaritmo natural, etcétera; se especifica cómo representarlas y usarlas en pseudocódigo.

Algo esencial en el aprendizaje de la programación de computadoras es la aplicación de los conceptos estudiados en la elaboración de algoritmos para solucionar problemas. Si el estudiante no hace algoritmos, no aprende; es por ello que es fundamental que se ejercite estudiando los problemas planteados en la sección de ejercicios resueltos y propuestos.

Cuando estudie los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la propuesta en el libro; luego, verifique sus resultados con los del texto, analice las diferencias y vea si tiene errores. Al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no es igual que el del libro, no necesariamente está mal. Usted debe ir aprendiendo que hay ciertas diferencias que puede haber, sobre todo en lo que respecta al nombre de las variables.

En el siguiente capítulo se estudia la estructura de control selección, qué es, cómo se representa y cómo se usa en pseudocódigo.

La secuenciación es una estructura que permite controlar la ejecución de un conjunto de acciones en orden secuencial, esto es, ejecuta la primera acción, luego la que sigue y así sucesivamente hasta la última, como se muestra a continuación:

Algoritmo IDENTIFICACIÓN

1. Acción uno
2. Acción dos
3. Acción tres
4. Acción cuatro
5. Fin

El algoritmo consiste en un conjunto de pasos secuenciales. La secuenciación permite ejecutar primero la acción uno, después la dos, luego la tres, enseguida la cuatro y por último el fin. Dichas acciones pueden consistir en operaciones primitivas elementales como declarar variables, leer datos, calcular alguna expresión o fórmula, o imprimir datos, de acuerdo con los lineamientos descritos en el capítulo anterior. Como puede notarse, es conveniente etiquetar cada acción con números desde el 1 en forma ascendente de uno en uno, para denotar el orden secuencial.



En el capítulo anterior dijimos que un programa de computadora está formado por un conjunto de instrucciones codificadas usando un lenguaje de programación como Java. Sin embargo aquí, al diseñar algoritmos, a los pasos que los forman les estaremos llamando acciones. Es decir, las acciones de los algoritmos se convertirán en instrucciones cuando se haga la codificación del programa.

Nota: En el capítulo anterior dijimos que un programa de computadora está formado por un conjunto de instrucciones codificadas usando un lenguaje de programación como Java. Sin embargo aquí, al diseñar algoritmos, a los pasos que los forman les estaremos llamando acciones. Es decir, las acciones de los algoritmos se convertirán en instrucciones cuando se haga la codificación del programa.

3.1 Nuestro primer problema

A continuación se presenta un ejemplo para aplicar los conceptos antes descritos y además para explicar la forma como se arma un algoritmo.

Ejercicio:

Elaborar un algoritmo para calcular e imprimir el sueldo de un empleado.

Siguiendo el proceso de programación se hace lo siguiente:

1. *Definir el problema.*

Calcular el sueldo de un empleado.

2. *Analizar el problema.*

Información por producir: Nombre, Sueldo.

Datos disponibles: Nombre, Número de horas trabajadas y Cuota por hora.

Proceso por seguir: $\text{Sueldo} = \text{Número de horas trabajadas} \times \text{Cuota por hora}$.

3. *Diseñar el programa.*

Se diseña la estructura de la solución, elaborando el algoritmo de acuerdo con los lineamientos que se explican en la sección siguiente.

3.2 Estructura y diseño de un algoritmo

En esta sección se explica el uso de los elementos que integran la estructura de un algoritmo, aplicándolos al problema definido en la sección anterior.

1. Encabezado

Todo algoritmo debe tener un encabezado como identificación, el cual debe empezar con la palabra *Algoritmo*, seguida por una breve descripción de lo que hace.

Para el problema que nos ocupa puede ser:

```
Algoritmo CALCULA SUELDO DE UN EMPLEADO
```

2. Clases

Un algoritmo está formado por un conjunto de una o más clases, y por lo menos debe tener una clase. Cada clase está formada por un conjunto de uno o más métodos y en todo el algoritmo debe haber un método principal, que es donde inicia su funcionamiento. Con los métodos se implementan las funciones que hace el algoritmo; así, en un algoritmo pequeño, como son los que haremos en los primeros capítulos, habrá una sola clase y esa clase tendrá un solo método, que será el principal, y en ese método van las acciones del algoritmo. A continuación se muestra el formato que se va a utilizar:

```

Algoritmo IDENTIFICACIÓN
Clase NomClase
  1. Método principal()
    a. Acción 1
    b. Acción 2
    c. Acción 3
    d. Acción 4
    e. Acción 5
    f. Fin Método principal
Fin Clase NomClase
Fin

```

Explicación:

Se tiene el encabezado del algoritmo con su identificación de lo que hace.

El algoritmo tiene una clase que inicia con `Clase NomClase` y finaliza en `Fin Clase NomClase`.

La clase tiene un método, que es el método principal, el cual contiene las acciones que resuelven el problema y termina con `Fin Método principal`.

Nota 1: En los algoritmos que se estudiarán desde el capítulo 3 hasta el 6 se utilizará sólo una clase y dentro de esa clase se tendrá sólo el método principal, en el cual desarrollaremos la lógica que resuelve los problemas que plantearemos. Sin embargo, la clase puede tener más de un método: algo que se estudiará en el capítulo 7. Además, en un algoritmo puede haber más de una clase, lo que se estudiará del capítulo 8 en adelante.

Nota 2: Desde el capítulo 3 hasta el 7 estaremos utilizando un concepto de clase limitado, que es el siguiente: es una estructura de un algoritmo (programa) en la que se representa la solución de un problema en torno a un objeto (empleado, alumno, obrero, etc.) o a un conjunto de objetos iguales (empleados, alumnos, obreros, etc.). Del capítulo 8 en adelante se estará utilizando el concepto de clase completo, como lo define la programación orientada a objetos.

3. Declarar

El primer paso en el diseño de un algoritmo consiste en declarar los elementos que se necesiten como variables, constantes, tipos de datos, etcétera. En el caso que nos ocupa, se requieren las variables siguientes:

nombreEmp	tipo Cadena de caracteres para manejar el dato nombre.
horasTrab	tipo Entero para manejar el número de horas trabajadas.
cuotaHora	tipo Real para manejar la cuota que se le paga por hora.
sueldo	tipo Real para manejar el sueldo que se va a pagar al empleado.



En los algoritmos que se estudiarán desde el capítulo 3 hasta el 6 se utilizará sólo una clase y dentro de esa clase se tendrá sólo el método principal, en el cual desarrollaremos la lógica que resuelve los problemas que plantearemos. Sin embargo, la clase puede tener más de un método: algo que se estudiará en el capítulo 7. Además, en un algoritmo puede haber más de una clase, lo que se estudiará del capítulo 8 en adelante.

Desde el capítulo 3 hasta el 7 estaremos utilizando un concepto de clase limitado, que es el siguiente: es una estructura de un algoritmo (programa) en la que se representa la solución de un problema en torno a un objeto (empleado, alumno, obrero, etc.) o a un conjunto de objetos iguales (empleados, alumnos, obreros, etc.). Del capítulo 8 en adelante se estará utilizando el concepto de clase completo, como lo define la programación orientada a objetos

Esto se representa en pseudocódigo de la manera siguiente:

```
Declarar variables
  nombreEmp: Cadena
  horasTrab: Entero
  cuotaHora: Real
  sueldo: Real
```

4. Leer, calcular e imprimir

El segundo paso y los que le siguen pueden consistir en acciones tales como leer datos, calcular alguna expresión aritmética e imprimir datos tantas veces como se requieran y en el orden apropiado para resolver el problema en cuestión.

Lectura de datos. En este punto se empiezan a introducir los datos disponibles como materia prima mediante una operación de lectura precedida por una solicitud de los datos. En nuestro problema esto quedaría así:

```
Solicitar nombre del empleado, número de horas trabajadas,
  cuota por hora
Leer nombreEmp, horasTrab, cuotaHora
```

Hacer cálculos. El siguiente punto es procesar la entrada para producir la salida mediante la ejecución de cálculos basados en expresiones aritméticas. En el ejemplo esta acción se expresa así:

```
Calcular sueldo = horasTrab * cuotaHora
```

Impresión de datos. El último punto estriba en dar salida a la información requerida, imprimiendo las variables que la contienen. En el ejemplo se expresa así:

```
Imprimir nombreEmp, sueldo
```

5. Fin del método principal, de la clase y del algoritmo

Por último se tiene el fin del método principal, el fin de la clase y el fin del algoritmo.

3.3 Nuestro primer algoritmo

A continuación armaremos nuestro primer algoritmo en pseudocódigo. El algoritmo servirá para calcular el sueldo de un empleado utilizando la estructura de control SECUENCIACION y todos los demás conceptos explicados hasta el momento.

```

Algoritmo CALCULA SUELDO DE UN EMPLEADO
Clase Empleado1
1. Método principal()
  a. Declarar variables
     nombreEmp: Cadena
     horasTrab: Entero
     cuotaHora, sueldo: Real
  b. Solicitar nombre del empleado, número de horas trabajadas,
     cuota por hora
  c. Leer nombreEmp, horasTrab, cuotaHora
  d. Calcular sueldo = horasTrab * cuotaHora
  e. Imprimir nombreEmp, sueldo
  f. Fin Método principal
Fin Clase Empleado1
Fin

```



En la zona de descarga de la Web del libro está disponible:

Programa en Java: Empleado1.java

Nota: El propósito de este libro es enseñar a desarrollar algoritmos en pseudocódigo. Por tanto, aquí no se explica nada del lenguaje Java; sin embargo, en la dirección <http://virtual.alfaomega.com.mx>, que es una web de ayuda que la editorial Alfaomega ha puesto a su disposición, podrá encontrar los programas correspondientes de los algoritmos del libro, codificados en Java.

Explicación:

Este algoritmo —como todos— comienza con el encabezado, en el cual se establece una identificación de lo que hace: CALCULA SUELDO DE UN EMPLEADO.

El algoritmo tiene una clase: la Clase `Empleado1`, que termina con `Fin Clase Empleado1`. Observe que en el nombre de la clase se le ha puesto `Empleado1`; el 1 significa que posteriormente habrá otra clase relacionada con un empleado y se le pondrá `Empleado2`, y así sucesivamente. Esto también se aplicará para alumnos, obreros, etcétera.

La Clase `Empleado1` tiene un solo método: el Método principal, el cual tiene las siguientes acciones:

1. Se hacen las declaraciones; se declaran las variables:

<code>nombreEmp</code>	tipo Cadena para manejar el dato nombre.
<code>horasTrab</code>	tipo Entero para manejar el número de horas trabajadas.
<code>cuotaHora</code>	tipo Real para manejar la cuota que se le paga por hora.
<code>sueldo</code>	tipo Real para manejar el sueldo a pagar al empleado.

Nota: Cuando hay más de una variable de un mismo tipo de dato, se pueden definir juntas separándolas por coma, como `cuotaHora, sueldo: Real`



El propósito de este libro es enseñar a desarrollar algoritmos en pseudocódigo. Por tanto, aquí no se explica nada del lenguaje Java; sin embargo, en la dirección <http://virtual.alfaomega.com.mx>, que es una web de ayuda que la editorial Alfaomega ha puesto a su disposición, podrá encontrar los programas correspondientes de los algoritmos del libro, codificados en Java.



Cuando hay más de una variable de un mismo tipo de dato, se pueden definir juntas separándolas por coma, como `cuotaHora, sueldo: Real`

Esquemáticamente, en la memoria las variables quedan definidas así:

nombreEmp	<input type="text"/>
cuotaHora	<input type="text"/>
horasTrab	<input type="text"/>
sueldo	<input type="text"/>

- Se solicitan los datos nombre del empleado, número de horas trabajadas y cuota por hora. Esto significa que en la pantalla se imprime lo siguiente:

```
Teclee nombre del empleado:_
Teclee número de horas trabajadas:_
Teclee cuota por hora:_
```

- Se leen el nombre del empleado, número de horas trabajadas y cuota por hora en las variables `nombreEmp`, `horasTrab` y `cuotaHora`. Esta acción espera a que se le teclee el dato correspondiente a cada petición y los datos tecleados se guardan en las variables. Esquemáticamente queda así:

nombreEmp	<input type="text" value="Socorro Román Maldonado"/>
cuotaHora	<input type="text" value="250.00"/>
horasTrab	<input type="text" value="40"/>
sueldo	<input type="text"/>

- Se calcula el $\text{sueldo} = \text{horasTrab} * \text{cuotaHora}$. Se multiplican los contenidos de las variables `horasTrab` por `cuotaHora`; es decir, 250.00 por 40, y el resultado se coloca en `sueldo`. Esquemáticamente queda así:

sueldo	<input type="text" value="10000.00"/>
--------	---------------------------------------

- Se imprimen los datos `nombreEmp` y `sueldo`. Es decir, se imprime lo siguiente:
El contenido de la variable `nombreEmp`, que es:
Socorro Román Maldonado
y el contenido de la variable `sueldo`, que es:
10000.00
- Fin del método principal. Se termina el método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Práctica: En este momento se recomienda ir al punto de ejercicios resueltos, estudiar algunos ejemplos y resolver otros de los ejercicios propuestos.

3.4 Funciones matemáticas

Todo lenguaje de programación proporciona una gran variedad de funciones estándar, es decir, que ya están definidas por el lenguaje: por ejemplo, las funciones matemáticas, las cuales son funciones de carácter general que nos facilitan la ejecución de ciertos cálculos de índole técnica o científica; ejemplos de estas funciones son: seno, coseno, raíz cuadrada, etcétera. A continuación se explican en detalle:

Función: Seno.

Efecto: Calcula el seno.

Formato:

Seno (x)

En donde:

Seno

Es la palabra que identifica a la función.

x

Es el parámetro que indica el valor al que se le obtiene el seno. Debe darse en radianes y debe ser de tipo real. El tipo de dato del resultado que devuelve es real.

Ejemplo:

 $y = \text{Seno}(x)$

Se calcula el seno de x, y se le asigna a y.

Función: Coseno.

Efecto: Calcula el coseno.

Formato:

Coseno (x)

En donde:

Coseno

Es la palabra que identifica a la función.

x

Es el parámetro que indica el valor al que se le obtiene el coseno. Debe estar dado en radianes y debe ser de tipo real. El tipo de dato del resultado que devuelve es real.

Ejemplo:

 $y = \text{Coseno}(x)$

Se calcula el coseno de x, y se le asigna a y.

Función: Arco tangente.

Efecto: Calcula el arco tangente.

Formato:

ArcoTan (x)

En donde:

ArcoTan

x

Es la palabra que identifica a la función.

Es el parámetro que indica el valor al que se le obtiene el arco tangente. Debe ser de tipo real. El tipo de dato del resultado es real.

Ejemplo:

$$y = \text{ArcoTan}(x)$$

Se calcula el arco tangente de x, y se le asigna a y.

Función: Logaritmo natural.

Efecto: Calcula el logaritmo natural.

Formato:

Ln (X)

En donde:

Ln

x

Es la palabra que identifica a la función.

Es el parámetro que indica el valor al que se le obtiene el logaritmo natural (x debe ser mayor que 0). Debe ser de tipo real. El tipo de dato del resultado es real.

Ejemplo:

$$y = \text{Ln}(x)$$

Se calcula el logaritmo natural de x, y se le asigna a y.

Función: Exponencial.

Efecto: Calcula e elevada a la x potencia, donde e es la base del sistema logarítmico natural (e=2.7182818...).

Formato:

Exp (x)

Exp

x

Es la palabra que identifica a la función.

Es el parámetro que indica la potencia a la que se elevará la constante e. Puede ser de tipo entero o real. El tipo de dato del resultado es real.

Ejemplo:

$$y = \text{Exp}(4)$$

Se calcula e elevada a la potencia 4 y se le asigna a y.

Función: Absoluto.

Efecto: Calcula el valor absoluto.

Formato:

```
Absoluto(x)
```

En donde:

Absoluto

Es la palabra que identifica a la función.

x

Es el parámetro que indica el valor al que se le obtiene el valor absoluto. Puede ser de tipo entero o real.

El tipo de dato del resultado es igual que el tipo del parámetro, es decir, si el parámetro es entero el resultado será entero; si el parámetro es real el resultado será real.

Ejemplo:

```
y = Absoluto(-24.56)
```

Se obtiene el valor absoluto de -24.56 y se le asigna a y.

Función: Raíz cuadrada.

Efecto: Calcula la raíz cuadrada.

Formato:

```
RaizCuad(x)
```

En donde:

RaizCuad

Es la palabra que identifica a la función.

x

Es el parámetro que indica el valor al que se le calcula la raíz cuadrada. Puede ser de tipo entero o real.

El tipo de dato del resultado es real.

Ejemplo:

```
y = RaizCuad(16)
```

Se calcula la raíz cuadrada de 16 y se le asigna a y.

Función: Potencia.

Efecto: Calcula la potencia de una base elevado a un exponente.

Formato:

```
Potencia(base, exponente)
```

En donde:

Potencia	Es la palabra que identifica a la función.
base	Es un parámetro que indica la base.
exponente	Es un parámetro que indica la potencia a la que se elevará la base. El tipo de dato puede ser entero o real.

Ejemplo:

```
y = Potencia(x, 3)
```

Se calcula la potencia de x elevada a la 3, y se le asigna a y.

Nota: En los lenguajes de programación, cada función tiene su nombre en inglés, o con base en siglas en inglés o abreviaturas de palabras en inglés como sin, sqrt, pow. Aquí, en este pseudocódigo, las estaremos manejando en forma castellanizada, como se mostró anteriormente.

Ejercicio 3.4.1

Elaborar un algoritmo que permita leer el tamaño de un ángulo en radianes, luego que calcule e imprima el seno y coseno.

(Primero hágalo usted; después compare la solución)

```
Algoritmo CALCULOS LOGARITMICOS DE ANGULO
Clase Angulo1
  1. Método principal()
    a. Declarar variables
       tamAngulo, senAng, cosAng: Real
    b. Solicitar tamaño del ángulo en radianes
    c. Leer tamAngulo
    d. Calcular senAng = Seno(tamAngulo)
       cosAng = Coseno(tamAngulo)
    e. Imprimir senAng, cosAng
    f. Fin Método principal
Fin Clase Angulo1
Fin
```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Angulo1.java



Explicación:

El algoritmo tiene una clase: la Clase Angulo1, que termina con Fin Clase Angulo1.

La Clase Angulo1 tiene un solo método: el Método principal, el cual tiene las siguientes acciones:



En los lenguajes de programación, cada función tiene su nombre en inglés, o con base en siglas en inglés o abreviaturas de palabras en inglés como sin, sqrt, pow. Aquí, en este pseudocódigo, las estaremos manejando en forma castellanizada, como se mostró anteriormente.

1. Se declaran las variables:
 tamAngulo para leer el tamaño del ángulo.
 senAng para calcular el seno.
 cosAng para calcular el coseno.
2. Se solicita el tamaño del ángulo en radianes.
3. Se lee el dato en tamAngulo.
4. Se calcula el seno, coseno.
5. Se imprimen los datos de salida.
6. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 3.4.2

En los lenguajes de programación, las funciones seno, coseno y tangente requieren que el argumento o parámetro que se les envíe esté dado en radianes. Si el dato disponible está dado en grados, se puede hacer la conversión a radianes con la siguiente equivalencia:

$$1^\circ = \frac{\pi}{180} = 0.0174775 \text{ radianes}$$

Para convertir de radianes a grados:

$$1 \text{ radián} = \frac{180}{\pi} = 57.21 \text{ grados}$$

Elaborar un algoritmo que permita leer un número en radianes e imprima su equivalencia en grados; asimismo, que permita leer un número en grados e imprima su equivalencia en radianes.

(Primero hágalo usted; después compare la solución)

```

Algoritmo CONVIERTE RADIANES A GRADOS Y GRADOS A RADIANES
Clase RadianGrado1
1. Método principal()
  a. Declarar
    Constantes
      PI = 3.14159265
    Variables
      radianes, grados, numRadianes, numGrados: Real
  b. Solicitar número de radianes
  c. Leer radianes
  d. Solicitar número de grados
  e. Leer grados
  f. Calcular numGrados = radianes *(180/PI)
      numRadianes = grados *(PI/180)
  g. Imprimir numGrados, numRadianes
  h. Fin Método principal
Fin Clase RadianGrado1
Fin

```



En la zona de descarga de la Web del libro está disponible:
 Programa en Java: RadianGrado1.java

Explicación:

El algoritmo tiene una clase: la Clase `RadianGrad01`, que termina con `Fin Clase RadianGrad01`.

La Clase `RadianGrad01` tiene un solo método: el Método `principal`, el cual tiene las siguientes acciones:

1. Se hacen las declaraciones. Se define la constante $PI = 3.14159265$
Se definen las variables:
`radianes` para leer el número de radianes.
`grados` para leer el número de grados.
`numRadianes` para calcular el número de radianes.
`numGrados` para calcular el número de grados.
2. Se solicita el número de radianes.
3. Se lee el dato en la variable `radianes`.
4. Se solicita el número de grados.
5. Se lee el dato en la variable `grados`.
6. Se calcula el número de grados y el número de radianes.
7. Se imprimen los datos de salida.
8. Fin del método `principal`. Luego se tiene el fin de la clase y el fin del algoritmo.

Práctica: En este momento se recomienda ir al punto de ejercicios resueltos y estudiar los ejemplos. Si considera que requiere más práctica, puede ir a resolver algunos de los ejercicios propuestos.

3.5 Ejercicios resueltos

A continuación se presentan ejercicios resueltos. Se le recomienda que primero haga usted el algoritmo y después compare su solución con la del libro.

Ejercicio 3.5.1

Elaborar un algoritmo para calcular el área de un triángulo. Se requiere imprimir como salida el área del triángulo. Los datos disponibles para leer como entrada son la base y la altura del triángulo.

El área se calcula:
$$\text{Área} = \frac{\text{Base} \times \text{Altura}}{2}$$

Diseño del algoritmo

(Primero hágalo usted; después compare la solución)

```
Algoritmo AREA TRIANGULO
Clase AreaTriangulo1
1. Método principal()
  a. Declarar variables
    area, base, altura: Real
  b. Solicitar base, altura
  c. Leer base, altura
  d. Calcular area = (base * altura) / 2
  e. Imprimir area
  f. Fin Método principal
Fin Clase AreaTriangulo1
Fin
```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: AreaTriangulo1.java

Explicación:

El algoritmo tiene una clase: la Clase AreaTriangulo1, que termina con Fin Clase AreaTriangulo1.

La Clase AreaTriangulo1 tiene un solo método: el Método principal, el cual tiene las siguientes acciones:

1. Se declaran las variables:
base para leer y manejar la base del triángulo.
altura para leer y manejar la altura del triángulo.
area para calcular el área.
2. Se solicitan los datos de base y de altura.
3. Se leen los datos en base y altura.
4. Se calcula el área con la expresión: $(base * altura) / 2$.
Nota: En este caso sólo se necesita un cálculo
5. Se imprime el dato obtenido en area.
6. Fin del método. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 3.5.2

Elaborar un algoritmo para calcular el promedio de calificaciones de un estudiante. Los datos disponibles para lectura son el nombre, calificación 1, calificación 2, calificación 3 y calificación 4 de cada uno de los cuatro exámenes presentados. La información que se debe imprimir es el nombre y el promedio de las calificaciones. El promedio se obtiene sumando las cuatro calificaciones y dividiendo la suma entre 4.

(Primero hágalo usted; después compare la solución)

```
Algoritmo CALCULA PROMEDIO DE UN ALUMNO
Clase Alumno1
1. Método principal()
  a. Declarar variables
     nombreAlum: Cadena
     calif1, calif2, calif3, calif4, promedio: Real
  b. Solicitar nombre del alumno, calificación 1,
     calificación 2, calificación 3, calificación 4
  c. Leer nombreAlum, calif1, calif2, calif3, calif4
  d. Calcular promedio = (calif1+calif2+calif3+calif4)/4
  e. Imprimir nombreAlum, promedio
  f. Fin Método principal
Fin Clase Alumno1
Fin
```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Alumno1.java



Explicación:

El algoritmo tiene una clase: la Clase Alumno1, que termina con Fin Clase Alumno1.

La Clase Alumno1 tiene un solo método: el Método principal, el cual tiene las siguientes acciones:

1. Se declaran las variables:
nombreAlum para leer y manejar el nombre del alumno.
calif1, calif2, calif3, calif4 para leer las cuatro calificaciones.
promedio para calcular el promedio.
2. Se solicitan los datos del nombre del alumno y las cuatro calificaciones.
3. Se leen el nombre del alumno y las cuatro calificaciones.
4. Se calcula el promedio final.
5. Se imprimen los datos de nombreAlum y promedio.
6. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 3.5.3

Elaborar un algoritmo que calcule e imprima el precio de venta de un artículo. Se tienen los datos de descripción del artículo y del costo de producción. El precio de venta se calcula añadiéndole al costo el 120 % como utilidad y el 15 % de impuesto.

(Primero hágalo usted; después compare la solución)

```

Algoritmo PRECIO DE VENTA
Clase PrecioArticulo1
1. Método principal()
  a. Declarar variables
    descripcion: Cadena
    costo, precioVta: Real
  b. Solicitar descripción del artículo, costo
  c. Leer descripcion, costo
  d. Calcular precioVta = costo + (costo*1.2) +
                        ((costo+(costo*1.2))*0.15)
  e. Imprimir descripcion, precioVta
  f. Fin Método principal
Fin Clase PrecioArticulo1
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: PrecioArticulo1.java

Explicación:

El algoritmo tiene una clase: la Clase PrecioArticulo1, que termina con Fin Clase PrecioArticulo1.

La Clase PrecioArticulo1 tiene un solo método: el Método principal, el cual tiene las siguientes acciones:

1. Se declaran las variables:
 descripcion para leer y manejar la descripción.
 costo para leer el costo del artículo.
 precioVta para calcular el precio de venta.
2. Se solicitan los datos descripción del artículo y costo.
3. Se leen los datos.
4. Se calcula el precio de venta.
5. Se imprimen los datos de descripcion y de precioVta.
6. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Tabla 3.1: ejercicios resueltos disponibles en la zona de descarga del capítulo 3 de la Web del libro.

Ejercicio	Descripción
Ejercicio 3.5.4	Calcula la hipotenusa de un triángulo rectángulo
Ejercicio 3.5.5	Convierte temperaturas Fahrenheit
Ejercicio 3.5.6	Calcula el volumen de un cilindro
Ejercicio 3.5.7	Calcula el tamaño de un ángulo
Ejercicio 3.5.8	Calcula el cateto B de un triángulo rectángulo
Ejercicio 3.5.9	Realiza cálculos logarítmicos
Ejercicio 3.5.10	Realiza cálculos logarítmicos

3.6 Ejercicios propuestos

1. Elaborar un algoritmo que calcule e imprima el costo de producción de un artículo, teniendo como datos la descripción y el número de unidades producidas. El costo se calcula multiplicando el número de unidades producidas por un factor de costo de materiales de 3.5 y sumándole al producto un costo fijo de 10 700.
2. Elaborar un algoritmo que calcule e imprima el costo de un terreno cuadrado o rectangular, teniendo como datos la anchura y la longitud en metros y el costo del metro cuadrado.
3. Elaborar un algoritmo que lea una cantidad de horas e imprima su equivalente en minutos, segundos y días.
4. Similar al del alumno (Ejercicio 3.5.2), con la diferencia de que en lugar del promedio se obtiene una calificación final multiplicando las calificaciones 1,2,3 y 4 por los porcentajes 30, 20,10 y 40 %, respectivamente, y sumando los productos.
5. La velocidad de la luz es de 300 000 kilómetros por segundo. Elaborar un algoritmo que lea un tiempo en segundos e imprima la distancia que recorre la luz en dicho tiempo.
6. Hacer un algoritmo que obtenga e imprima el valor de Y a partir de la ecuación $Y = 3X^2 + 7X - 15$ solicitando como dato de entrada el valor de X.
7. Una temperatura en grados Celsius (C) se puede convertir a su equivalente Fahrenheit (F) con la fórmula:

$$F = \frac{9}{5} C + 32$$

De Fahrenheit a Celsius con la fórmula:

$$C = F - 32 \frac{5}{9}$$

Elaborar un algoritmo que lea una temperatura en grados Celsius y obtenga e imprima la temperatura Fahrenheit equivalente.

8. Elabore un algoritmo que lea un número de pies y calcule e imprima su equivalente en yardas, pulgadas, centímetros y metros, de acuerdo con las siguientes equivalencias: 1 pie = 12 pulgadas, 1 yarda = 3 pies, 1 pulgada = 2.54 cm, 1 metro = 100 cm.
9. Elaborar un algoritmo que lea el artículo y su costo. La utilidad es el 150% y el impuesto es el 15%. Calcular e imprimir artículo, utilidad, impuesto y precio de venta.
10. Elaborar un algoritmo que lea el radio de un círculo e imprima el área.
 $\text{ÁREA} = \pi r^2$
11. Elaborar un algoritmo que lea la cantidad de dólares que se va a comprar y el tipo de cambio en pesos (costo de un dólar en pesos). Calcular e imprimir la cantidad que se debe pagar en pesos por la cantidad de dólares indicada.

12. Elaborar un algoritmo que permita leer valores para X, Y, Z y W e imprima el valor de F.

$$F = \frac{(4x^2y^2 - 2zw)^2}{4x^{1/2} b^{3/4}}$$

13. Elaborar un algoritmo que lea el radio(r) de una esfera, calcule e imprima el volumen y el área.

$$\text{VOLUMEN} = \frac{4\pi r^3}{3} \quad \text{ÁREA} = \pi r^2$$

14. Elaborar un algoritmo que lea el valor de W e imprima el valor de Z.

$$Z = \frac{1}{\sqrt{2\pi}} e^{-\frac{w^2}{2}}$$

15. Elaborar un algoritmo que lea la cantidad de dólares que se va a comprar y el tipo de cambio (costo de un dólar) en: yenes, pesetas, libras esterlinas y marcos. Calcular e imprimir la cantidad que se debe pagar en yenes, pesetas, libras esterlinas y marcos.

16. Elaborar un algoritmo que permita leer un valor e imprima el logaritmo natural, el exponencial, el valor absoluto y la raíz cuadrada.

17. Elaborar un algoritmo que permita leer el tamaño de un ángulo en radianes e imprima la tangente, cotangente, secante y cosecante.

$$\text{tangente} = \frac{\text{seno}}{\text{coseno}} \quad \text{secante} = \frac{1}{\text{coseno}}$$

$$\text{cotangente} = \frac{\text{coseno}}{\text{seno}} \quad \text{cosecante} = \frac{1}{\text{seno}}$$

18. Elaborar un algoritmo similar al anterior, sólo que el dato que se lee estará dado en grados. Debe convertirse los grados leídos a radianes antes de hacer los cálculos.

19. Elaborar un algoritmo que permita leer el tamaño de un ángulo en grados e imprima el seno y coseno. Debe convertirse los grados leídos a radianes antes de hacer los cálculos.

20. Elaborar un algoritmo que permita leer valores para A y B e imprima Y, Z y W.

$$Y = 3a^2b^2\sqrt{2a} \quad W = 4\sqrt{2^a a} (3a^2b^2 - \sqrt{2a})$$

$$Z = \frac{12a^{1/2}}{b^{3/4}}$$

3.7 Resumen de conceptos que debe dominar

- La secuenciación
- Estructura y diseño de un algoritmo en pseudocódigo
- Funciones matemáticas (seno, coseno, arco tangente, logaritmo natural, exponencial, elevar a potencias, raíz cuadrada, etc.).

3.8 Contenido de la página Web de apoyo

El material marcado con asterisco () sólo está disponible para docentes.*

3.8.1 Resumen gráfico del capítulo

3.8.2 Autoevaluación

3.8.3 Programas en Java

3.8.4 Ejercicios resueltos

3.8.5 Power Point para el profesor (*)

4

La selección

Contenido

- 4.1 La selección doble (if-then-else)
 - 4.1.1 Sangrado (indentación) y etiquetas
 - 4.1.2 Expresiones lógicas
 - 4.1.3 if's anidados
 - 4.1.4 Ejercicios resueltos para la selección doble (if-then-else)
- 4.2 La selección simple (if-then)
 - 4.2.1 Ejercicios resueltos para la selección simple (if-then)
- 4.3 La selección múltiple (switch)
 - 4.3.1 Ejercicio resuelto para la selección múltiple (switch)
- 4.4 Ejercicios propuestos
- 4.5 Resumen de conceptos que debe dominar
- 4.6 Contenido de la página Web de apoyo

El material marcado con asterisco () sólo está disponible para docentes.*

 - 4.6.1 Resumen gráfico del capítulo
 - 4.6.2 Autoevaluación
 - 4.6.3 Programas en Java
 - 4.6.4 Ejercicios resueltos
 - 4.6.5 Power Point para el profesor (*)

Objetivos del capítulo

- Aprender a elaborar algoritmos utilizando la estructura de control selección doble (if-then-else).
- Estudiar el uso de las expresiones lógicas simples (>, <, >=, <=, !=, ==) y complejas (AND, OR, XOR, NOT).
- Aprender a elaborar algoritmos utilizando la anidación de if's, la selección simple (if-then) y la selección múltiple (switch).
- Aplicar lo aprendido en la solución de ejercicios resueltos y propuestos.

Competencias

- Competencia general del capítulo
 - Analizar problemas y diseñar algoritmos que los solucionen aplicando la selección if-then-else, if-then y switch.
- Competencias específicas del capítulo
 - Diseña algoritmos aplicando la selección doble (if-then-else).
 - Elabora algoritmos aplicando la selección simple (if-then).
 - Desarrolla algoritmos aplicando la selección múltiple (switch).

Introducción

Con el estudio del capítulo anterior, usted ya domina la secuenciación y cómo diseñar algoritmos usando esa estructura de control.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos utilizando la estructura de control: la selección.

Se explica que la selección es una estructura que permite tomar decisiones y que tiene tres formas: la selección doble (if-then-else), la selección simple (if-then) y la selección múltiple (switch).

Se enseña que la selección doble (if-then-else) sirve para plantear situaciones en las que se tienen dos alternativas de acción, de las cuales se debe escoger una. Se describe cómo plantear expresiones lógicas simples usando los operadores relacionales (>, <, >=, <=, !=, ==) y expresiones lógicas complejas usando los operadores lógicos (AND, OR, XOR, NOT).

Se detalla que la selección simple (if-then) sirve para plantear situaciones en las que se tiene una sola alternativa de acción.

Se explica que la selección múltiple (switch) sirve para plantear situaciones en las que se tienen múltiples alternativas de acción, de las cuales se debe escoger una.

Asimismo, se plantea la anidación de if's; es decir, que un if tenga adentro otro if, y así sucesivamente.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejercite estudiando los problemas planteados en los ejercicios resueltos y propuestos. Al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro; luego verifique sus resultados con los del libro, analice las diferencias y vea si tiene errores. Al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no está igual que el del libro, no necesariamente está mal. Usted debe ir aprendiendo a analizar las diferencias y a comprender que a veces, aunque haya diferencias, las dos soluciones están correctas.

En el siguiente capítulo se estudia la estructura de control de repetición, qué es, cómo se representa y cómo se usa en pseudocódigo.

La selección es una estructura que permite controlar la ejecución de acciones que requieren de ciertas condiciones para su realización. De acuerdo con dichas condiciones se “selecciona” si las acciones se ejecutan o no. En ocasiones se tienen operaciones que son excluyentes, es decir, que sólo tiene que ejecutarse una o la otra, pero no ambas de manera simultánea; también puede presentarse el caso que se tengan varias opciones de acción. En estos casos es necesario utilizar la estructura de control de selección. Para una mejor comprensión del concepto, veamos el siguiente ejemplo:

Calcular el sueldo de un empleado.

Información por producir:

Nombre y sueldo

Datos disponibles:

Nombre, número de horas trabajadas y cuota por hora

Proceso:

El sueldo se calcula de la forma siguiente:

Si el número de horas trabajadas es mayor que 40, el excedente de 40 se paga al doble de la cuota por hora. En caso de no ser mayor que 40, se paga a la cuota por hora normal.

En esta situación, el sueldo se calcula de dos formas distintas:

1. Si es menor o igual que 40:
Sueldo = Número de horas trabajadas X cuota por hora
2. Si es mayor que 40:
Sueldo = (40 X cuota por hora) +
((número de horas trabajadas - 40) X (cuota por hora X 2))

Por lo tanto, es necesario utilizar la estructura de selección para calcular el sueldo de la primera forma si el número de horas trabajadas es menor o igual que 40, o de la segunda en el otro caso.

La estructura de selección tiene tres formas: **simple**, **doble** y **múltiple**, de acuerdo con el número de alternativas de acción, es decir, si hay una, dos o más de dos, respectivamente. Por razones didácticas, primero se estudiará la **doble**, después la **simple** y por último la **múltiple**.

4.1 La selección doble (if-then-else)

Esta estructura de selección permite controlar la ejecución de acciones cuando se tienen dos opciones alternativas de acción y, por la naturaleza de éstas, se debe ejecutar una o la otra, pero no ambas a la vez, es decir, son mutuamente excluyentes.

Formato:

```
if condición then
    Acción(es)
else
    Acción(es)
endif
```

En donde:

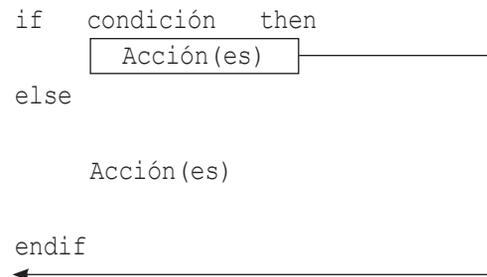
if (Si)	Identifica la estructura de selección.
condición	Es una expresión lógica que denota la situación específica mediante la comparación de dos operandos para dar un resultado booleano (falso, verdadero); es decir, si se cumple o no se cumple. En el apartado 4.1.2 Expresiones lógicas se explica en detalle.

then (Entonces)	Indica el curso de acción si se cumple la condición.
Acción(es)	Es la acción o conjunto de acciones en pseudocódigo que se ejecutarán en el bloque correspondiente.
else (si no; caso contrario)	Indica el curso de acción cuando no se cumple la condición.
endif	Indica el fin de la estructura de selección (del if).

Funcionamiento:

Al llegar al if se evalúa la condición:

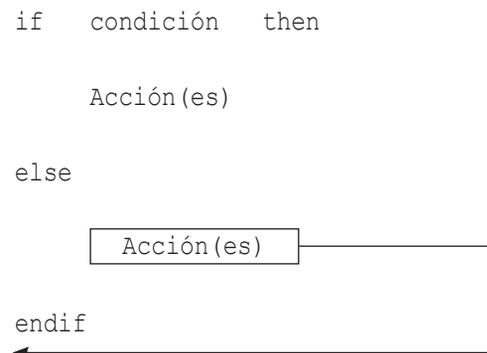
- a) Opción verdadera (then).
Si se cumple, se ejecuta(n) la(s) acción(es) del then y luego salta hasta la siguiente después del endif (fin del if).



Nota: En este caso ejecuta la opción del then e ignora la del else.

- b) Opción falsa (else).

De lo contrario, salta hacia el else, ejecuta la(s) acción(es), y después salta a la siguiente acción después del endif (fin del if).



Nota: En este caso ejecuta la opción del else e ignora la del then.

Sugerencia: En este momento se sugiere ir al apartado 4.1.2 y estudiar la parte correspondiente a Expresiones lógicas simples y regresar a este punto.

Aplicando lo anterior al ejemplo de cálculo de sueldo, queda:

```

if horasTrab <= 40 then
    sueldo = horasTrab * cuotaHora
else
    sueldo = (40*cuotaHora) + ((horasTrab-40) * (cuotaHora*2))
endif

```



La condición se plantea comparando `horasTrab` (horas trabajadas) con 40, mediante el operador relacional `<=` “Menor o igual que” (esto se explica con todo detalle dos puntos más adelante -4.1.2 Expresiones lógicas-). En caso de cumplirse la condición, se ejecutará la acción o acciones de la opción `then` y, de no cumplirse, se ejecutará la acción o acciones de la opción `else`.

A continuación se presenta el algoritmo completo para calcular el sueldo del empleado, considerando el doble para el excedente de 40 horas trabajadas.

```

Algoritmo CALCULO SUELDO DOBLE
Clase Empleado2
  1. Método principal()
    a. Declarar variables
      nombreEmp: Cadena
      horasTrab: Entero
      cuotaHora, sueldo: Real
    b. Solicitar nombre del empleado, número de horas
      trabajadas, cuota por hora
    c. Leer nombreEmp, horasTrab, cuotaHora
    d. if horasTrab <= 40 then
      1. sueldo = horasTrab * cuotaHora
    e. else
      1. sueldo = (40*cuotaHora)+
        ((horasTrab-40)*(cuotaHora*2))
    f. endif
    g. Imprimir nombreEmp, sueldo
    h. Fin Método principal
  Fin Clase Empleado2
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Empleado2.java



Explicación:

El algoritmo tiene una clase: la Clase `Empleado2`, que termina con `Fin Clase Empleado2`.

La Clase `Empleado2` tiene un solo método: el `Método principal`, el cual tiene las siguientes acciones:

- a. Se declaran las variables que ya conocemos.
- b. Se solicita el nombre del empleado, número de horas trabajadas y cuota por hora.
- c. Se leen los datos en `nombreEmp`, `horasTrab` y `cuotaHora`.
- d. Se compara si `horasTrab <= 40`. Si se cumple, entonces:
 1. Se calcula el sueldo de la forma simple.
- e. Si no se cumple (en caso contrario):
 1. Se calcula el sueldo considerando horas dobles.



Observe que tanto en el then como en el else la acción de calcular el sueldo está enumerada con el inciso 1; en el siguiente apartado se explica por qué.

- f. Finaliza el if (endif).
- g. Se imprime el nombre del empleado y el sueldo.
- h. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Nota: Observe que tanto en el then como en el else la acción de calcular el sueldo está enumerada con el inciso 1; en el siguiente apartado se explica por qué.

4.1.1 Sangrado (indentación) y etiquetas

Las instrucciones o acciones de los algoritmos pueden etiquetarse con números y letras, alternativamente, para identificar más fácilmente su orden en los diversos niveles de subordinación.

Si vemos el ejemplo que se da más adelante, el primer nivel de acciones son los métodos que se etiquetan con números; el primer método que es el principal se enumera con el 1, el segundo con el 2, y así sucesivamente. En el siguiente nivel de subordinación, adentro de los métodos, las acciones se etiquetan con letras. Además, en el ejemplo, en el método principal, las acciones a, c y d tienen subacciones, las cuales se jerarquizan dejando una sangría y se etiquetan con números. Y así, al haber otro nivel de subordinación, las acciones se vuelven a etiquetar con letras para usar alternativamente números, letras, números, letras y así sucesivamente. Con base en lo anterior, la estructura que se tendría sería la siguiente:

Algoritmo EJEMPLO SANGRADO(INDENTACION) Y ETIQUETAS

```

Clase NomClase
  1. Método principal()
    a. Acción A
      1. Acción A1
      2. Acción A2
    b. Acción B
    c. Acción C
      1. Acción C1
    d. Acción D
      1. Acción D1
        a. Acción D1A
        b. Acción D1B
      2. Acción D2
        a. Acción D2A
      3. Acción D3
    e. Acción E
    f. Fin Método principal

  2. Método dos()
    a. Acción A
    b. Acción B
    c. Fin Método dos
  
```

```

3. Método tres()
  a. Acción A
  b. Acción B
  c. Acción C
  d. Fin Método tres
Fin NomClase
Fin

```

Como podemos ver, la jerarquía y subordinación se denota dejando sangrados y etiquetados con números y letras, alternativamente, en los diferentes niveles.

Nota: A algunos maestros no les gusta usar etiquetas para enumerar los pasos de los algoritmos. No obstante, en este caso consideramos que es muy bueno para personas que recién se están formando porque da una mayor disciplina a la persona y un orden más entendible a los algoritmos. Una vez que la persona sabe programar, podrá o no usar etiquetas. En el presente libro se recomienda el uso de las etiquetas, pero si desea, puede omitirlas. En el Apéndice C (Algoritmos sin usar etiquetas), se presentan algunos algoritmos sin usar etiquetas.

Sin embargo, estimado maestro, si usted está utilizando este libro como texto, es conveniente que utilice los ejemplos como están planteados en el libro, es decir, con etiquetas, porque si usted explica los algoritmos sin etiquetas luego, cuando los alumnos estudien el libro y vean los algoritmos con etiquetas, seguramente les causará confusión y, en consecuencia, la utilidad del libro podría ser limitada.

4.1.2 Expresiones lógicas

En los dos apartados precedentes ya se ha hecho la sugerencia de remitirse a este punto, es decir, se le va a enviar a estudiar este punto en dos oportunidades para que estudie una parte en una ocasión y la otra en la siguiente; la idea es que después de estudiar la parte correspondiente debe regresar.

Las expresiones lógicas sirven para plantear condiciones mediante la comparación de dos o más operandos que dan como resultado un valor booleano verdadero o falso, es decir, se cumple o no se cumple la condición. Se pueden clasificar en simples y complejas.

Expresiones lógicas simples

Se forman relacionando operandos, variables y/o constantes mediante operadores relacionales, de la forma siguiente:

Expresión lógica simple =

Operando1 Operador relacional Operando2



A algunos maestros no les gusta usar etiquetas para enumerar los pasos de los algoritmos. No obstante, en este caso consideramos que es muy bueno para personas que recién se están formando porque da una mayor disciplina a la persona y un orden más entendible a los algoritmos. Una vez que la persona sabe programar, podrá o no usar etiquetas. En el presente libro se recomienda el uso de las etiquetas, pero si desea, puede omitirlas. En el Apéndice C (Algoritmos sin usar etiquetas), se presentan algunos algoritmos sin usar etiquetas.

Sin embargo, estimado maestro, si usted está utilizando este libro como texto, es conveniente que utilice los ejemplos como están planteados en el libro, es decir, con etiquetas, porque si usted explica los algoritmos sin etiquetas luego, cuando los alumnos estudien el libro y vean los algoritmos con etiquetas, seguramente les causará confusión y, en consecuencia, la utilidad del libro podría ser limitada.

En donde:

Operando1, Operando2 Son variables o valores constantes de algún tipo de datos: Entero, Real, Cadena, etcétera. En una expresión, ambos operandos deben ser del mismo tipo de dato.

Operador relacional Cualquiera de los siguientes:

Operadores relacionales	Significado
<	Menor que
>	Mayor que
<=	Menor o igual que
>=	Mayor o igual que
==	Igual a o igual que
!=	Diferente de o no igual a

Ejemplos:

```
x == 1      significa ¿x es igual a 1?
n != z      significa ¿n es no igual a z?
n >= 5     significa ¿n es mayor o igual a 5?
z <= 5     significa ¿z es menor o igual a 5?
y == z      significa ¿y es igual a z?
nombre == "Socorro Román Maldonado" significa ¿nombre
es igual a Socorro Román Maldonado?
```



Observe que el doble signo de igual (==) se utiliza para plantear comparaciones, es decir, expresiones lógicas, y un signo de igual (=) se utiliza para plantear cálculos a través de expresiones aritméticas o fórmulas.

Como ya mencionamos, los posibles resultados para cada una de las expresiones lógicas son verdadero (V —True—) o falso (F —False—); es decir, Sí se cumple o NO se cumple.

Nota: Observe que el doble signo de igual (==) se utiliza para plantear comparaciones, es decir, expresiones lógicas, y un signo de igual (=) se utiliza para plantear cálculos a través de expresiones aritméticas o fórmulas.

Expresiones lógicas complejas

Se forman relacionando operandos booleanos mediante operadores lógicos, como sigue:

Expresión lógica compleja =

Operando booleano1 operador lógico Operando booleano2

En donde:

Operando booleano1, Son expresiones lógicas que

Operando booleano2 proporcionan un valor verdadero (V) o falso (F).

operador lógico Cualquiera de los siguientes: AND, OR, XOR, NOT.

AND

Relaciona dos operandos booleanos. Proporciona un valor verdadero (V) si los dos son verdaderos (V); en caso contrario da un resultado falso (F).

Nota: Para un mejor tratamiento del tema, se ha decidido incluir en las tablas de valor V para los valores verdaderos. Sin embargo, los lenguajes de programación no emiten este valor como tal, sino una T, que es la sigla de la palabra inglesa *True*, que en español significa verdad.

Si se hace la suposición de que EXP1 y EXP2 son expresiones booleanas y se plantea la expresión:

(EXP1) AND (EXP2) Proporciona las combinaciones de resultados mostrados en la siguiente tabla:

EXP1	EXP2	(EXP1) AND (EXP2)
V	V	V
V	F	F
F	V	F
F	F	F

Ejemplo:

Una escuela aplica dos exámenes a sus aspirantes, por lo que cada uno de ellos obtiene dos calificaciones, denotadas como c1 y c2. El aspirante que obtenga calificaciones de 80 o más en ambos exámenes es aceptado; en caso contrario es rechazado.

```
if (c1 >= 80)AND(c2 >= 80) then
    Imprimir "Aceptado"
else
    Imprimir "Rechazado"
endif
```

OR

Relaciona dos operandos booleanos. Proporciona un valor verdadero (V) si uno de los dos es verdadero (V); en caso contrario da un resultado falso (F). En la siguiente tabla se presentan las posibles combinaciones:

EXP1	EXP2	(EXP1) OR (EXP2)
V	V	V
V	F	V
F	V	V
F	F	F

Ejemplo:

Una escuela aplica dos exámenes a sus aspirantes, por lo que cada uno de ellos obtiene dos calificaciones, denotadas como c1 y c2. El aspirante que obtenga una calificación mayor que 90 en cualquiera de los exámenes es aceptado; en caso contrario es rechazado.



Para un mejor tratamiento del tema, se ha decidido incluir en las tablas de valor V para los valores verdaderos. Sin embargo, los lenguajes de programación no emiten este valor como tal, sino una T, que es la sigla de la palabra inglesa *True*, que en español significa verdad.

```

if (c1 > 90)OR(c2 > 90) then
  Imprimir "Aceptado"
else
  Imprimir "Rechazado"
endif

```

XOR

Relaciona dos operandos booleanos. Proporciona un resultado verdadero (V) si uno de los dos es verdadero (V), pero no ambos; en caso contrario da un valor falso (F). En la siguiente tabla se presentan las posibles combinaciones.

EXP1	EXP2	(EXP1) XOR (EXP2)
V	V	F
V	F	V
F	V	V
F	F	F

Ejemplo:

Una empresa aplica dos exámenes a sus aspirantes a ingresar como trabajadores, por lo que cada uno de ellos obtiene dos calificaciones denotadas como c1 y c2. El aspirante que obtenga 100 en cualquiera de los exámenes, pero no en ambos, es aceptado; en caso contrario es rechazado.

```

if (c1 == 100)XOR(c2 == 100) then
  Imprimir "Aceptado"
else
  Imprimir "Rechazado"
endif

```



Aunque suena ilógico rechazar a quien haya obtenido 100 en ambos exámenes, supongamos que si obtiene 100 en ambos el aspirante le queda grande al puesto; esto provocaría que no desempeñe su trabajo con entusiasmo.

Nota: Aunque suena ilógico rechazar a quien haya obtenido 100 en ambos exámenes, supongamos que si obtiene 100 en ambos el aspirante le queda grande al puesto; esto provocaría que no desempeñe su trabajo con entusiasmo.

NOT

Este operador relaciona sólo un operando booleano y da como resultado un valor opuesto al que tenga el operando. En la siguiente tabla se presentan las posibles combinaciones.

EXP	NOT (EXP)
V	F
F	V

Ejemplo:

Un alumno tiene una calificación final (`calFin`) y se desea imprimir el resultado de aprobado si la calificación es igual o mayor a 70; o bien, reprobado en caso contrario.

Solución sin usar NOT:

```
if calFin >= 70 then
    Imprimir "Aprobado"
else
    Imprimir "Reprobado"
endif
```

Solución usando NOT:

```
if NOT(calFin >= 70) then
    Imprimir "Reprobado"
else
    Imprimir "Aprobado"
endif
```

Nota: El operador lógico NOT sirve para cambiar el resultado de una expresión lógica.

Presentamos el orden de precedencia de los operadores relacionales y lógicos.

1. Paréntesis ()
2. NOT
3. AND
4. OR, XOR
5. <, >, ==, <=, >=, !=

4.1.3 if's anidados

Una estructura de selección (if) puede tener anidada a otra y ésta a otra y así sucesivamente. Por ejemplo:

```
if condición then
    if condición then
        Acción(es)
    else
        Acción(es)
    endif
else
    if condición then
        Acción(es)
    else
        Acción(es)
    endif
endif
```



El operador lógico NOT sirve para cambiar el resultado de una expresión lógica.

Se tiene un `if` principal, el cual tiene anidado en el `then` un `if`, que tiene su propio `then`, `else` y `endif`. Por el `else` también hay un `if` anidado que contiene su `then`, `else` y `endif`.

Otro ejemplo de anidación sería el caso de tener una instrucción simple por el `then` y un `if` por el `else`. La estructura quedaría:

```
if condición then
    Acción(es)
else
    if condición then
        Acción(es)
    else
        Acción(es)
    endif
endif
```

Vemos que se tiene el `then` donde no lleva `if` anidado, mientras que por el `else` se tiene el `if` anidado.

Otra alternativa de anidación es el caso que se tenga un `if` anidado por el `then`, mientras que por el `else` no se tiene `if` anidado. El esqueleto de la estructura es:

```
if condición then
    if condición then
        Acción(es)
    else
        Acción(es)
    endif
else
    Acción(es)
endif
```

Ejercicio:

Elaborar un algoritmo similar al anterior de **CALCULO SUELDO DOBLE**, pero ahora tomando en cuenta que se tiene otra alternativa: las horas que exceden de 50 se pagan al triple de la cuota por hora.

A continuación se presenta el algoritmo:

(Primero hágalo usted; después compare la solución)

```
Algoritmo CALCULO SUELDO TRIPLE
Clase Empleado3
1. Método principal()
    a. Declarar variables
        nombreEmp: Cadena
        horasTrab: Entero
        cuotaHora, sueldo: Real
    b. Solicitar nombre del empleado, número de horas trabajadas,
        cuota por hora
```

```

c. Leer nombreEmp, horasTrab, cuotaHora
d. if horasTrab <= 40 then
    1. sueldo = horasTrab * cuotaHora
e. else
    1. if horasTrab <= 50 then
        a. sueldo=(40*cuotaHora)+
            ((horasTrab-40)*(cuotaHora*2))
    2. else
        a. sueldo=(40*cuotaHora)+(10*cuotaHora*2)+
            ((horasTrab-50)*(cuotaHora*3))
    3. endif
f. endif
g. Imprimir nombreEmp, sueldo
h. Fin Método principal
Fin Clase Empleado3
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Empleado3.java



Explicación:

En el Método principal de la Clase Empleado3 se tienen las acciones (los pasos a, b y c se plantean igual al algoritmo para el cálculo del sueldo doble; lo diferente son los siguientes pasos):

- d. Se compara si `horasTrab <= 40`. Si es así, entonces:
 1. Se calcula el sueldo de la forma simple.
- e. Si no, quiere decir que `horasTrab` es mayor que 40.
 1. Se compara si `horasTrab <= 50`. Si es así, entonces:
 - a. Se calcula el sueldo: 40 horas a la cuota por hora normal más el excedente de 40 a la cuota por hora al doble (por 2).
 2. Si no, quiere decir que `horasTrab` es mayor que 50.
 - a. Se calcula de la forma triple: 40 horas a la cuota por hora normal, más 10 horas a la cuota por hora al doble (por 2), más el excedente de 50 a la cuota por hora al triple (por 3).
 3. Fin del if interno.
- f. Fin del if principal.
- g. Se imprimen los datos de salida.
- h. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Práctica: En este momento se recomienda ir al siguiente punto de ejercicios resueltos para la selección doble (if-then-else) y estudiar algunos ejemplos. Además, si quiere más práctica, puede resolver algunos de los ejercicios propuestos al final del capítulo para aplicar el if-then-else.



Práctica: En este momento se recomienda ir al siguiente punto de ejercicios resueltos para la selección doble (if-then-else) y estudiar algunos ejemplos. Además, si quiere más práctica, puede resolver algunos de los ejercicios propuestos al final del capítulo para aplicar el if-then-else.

4.1.4 Ejercicios resueltos para la selección doble (if-then-else)

A continuación se presentan ejercicios resueltos. Se le recomienda que primero haga usted el algoritmo y después compare su solución con la del libro.

Ejercicio 4.1.4.1

Elaborar un algoritmo para calcular el promedio de calificaciones de un estudiante. Los datos disponibles para lectura son el nombre, calificación 1, calificación 2, calificación 3 y calificación 4 de cada uno de los cuatro exámenes presentados. La información que se debe imprimir es el nombre, el promedio de las calificaciones y un comentario de “Aprobado” si obtiene 60 o más o “Reprobado” en caso contrario. El promedio se obtiene sumando las cuatro calificaciones y dividiendo la suma entre 4. Este problema lo vamos a solucionar de tres formas distintas.

(Primero hágalo usted; después compare la solución)

Primer método de solución: Usando la variable observación.

```

Algoritmo CALCULA PROMEDIO DE UN ALUMNO
Clase Alumno2
  1. Método principal()
    a. Declarar variables
      nombreAlum: Cadena
      calif1, calif2, calif3, calif4, promedio: Real
      observacion: Cadena
    b. Solicitar nombre del alumno, calificación 1,
      calificación 2, calificación 3, calificación 4
    c. Leer nombreAlum, calif1, calif2, calif3, calif4
    d. Calcular promedio = (calif1+calif2+calif3+calif4)/4
    e. if promedio >= 60 then
      1. observacion = "Aprobado"
    f. else
      1. observacion = "Reprobado"
    g. endif
    h. Imprimir nombreAlum, promedio, observacion
    i. Fin Método principal
  Fin Clase Alumno2
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Alumno2.java

Explicación:

En el Método principal de la Clase Alumno2 se tienen las acciones:

- Se declaran las variables.
- Se solicitan el nombre y las cuatro calificaciones.
- Se leen los datos en las variables correspondientes.

- d. Se calcula el promedio.
- e. Si compara si `promedio >= 60`. Si se cumple, entonces:
 1. En la variable `observacion` se coloca el valor "Aprobado".
- f. Si no se cumple:
 1. En la variable `observacion` se coloca el valor "Reprobado".
- g. Fin del if.
- h. Se imprimen los datos `nombreEmp`, `promedio` y `observación`.
- i. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

En la zona de descarga del capítulo 4 de la Web del libro está disponible el ejercicio resuelto.



Ejercicio	Descripción
Ejercicio 4.1.4.1	Dos métodos alternativos de solución al problema anterior

Ejercicio 4.1.4.2

Elaborar un algoritmo que lea dos números y que imprima el mayor. Se supone que son números diferentes; por lo tanto, no se debe averiguar si son iguales o si son diferentes.

(Primero hágalo usted; después compare la solución)

```

Algoritmo MAYOR 2 NUMEROS
Clase Mayor2Numeros
  1. Método principal()
    a. Declarar variables
      a, b: Entero
    b. Solicitar número 1, número 2
    c. Leer a, b
    d. if a > b then
      1. Imprimir a
    e. else
      1. Imprimir b
    f. endif
    g. Fin Método principal
Fin Clase Mayor2Numeros
Fin
  
```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: `Mayor2Numeros.java`



Explicación:

En el Método principal de la Clase `Mayor2Numeros` se tienen las acciones:

- a. Se declaran las variables.
- b. Se solicitan el número 1 y el número 2.
- c. Se leen en las variables a y b.

- d. Se compara si $a > b$. Si es así, entonces:
 1. Se imprime a como el mayor.
- e. Si no (en caso contrario):
 1. Se imprime b como el mayor.
- f. Fin del if.
- g. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 4.1.4.3

Elaborar un algoritmo que lea tres números y que imprima el mayor. Se supone que son números diferentes; por lo tanto, no se debe averiguar si son iguales o si son diferentes. Este problema lo vamos a solucionar de tres formas distintas.

Primer método de solución:

Utilizando if-then-else y expresiones lógicas simples, es decir, sin usar AND.

(Primero hágalo usted; después compare la solución)

```
Algoritmo MAYOR 3 NUMEROS
Clase Mayor3Numeros1
1. Método principal()
  a. Declarar variables
    a, b, c: Entero
  b. Solicitar número 1, número 2, número 3
  c. Leer a, b, c
  d. if a > b then
    1. if a > c then
      a. Imprimir a
    2. else
      a. Imprimir c
    3. endif
  e. else
    1. if b > c then
      a. Imprimir b
    2. else
      a. Imprimir c
    3. endif
  f. endif
  g. Fin Método principal
Fin Clase Mayor3Numeros1
Fin
```



En la zona de descarga de la Web del libro está disponible:

Programa en Java: Mayor3Numeros1.java

Explicación:

En el Método principal de la Clase Mayor3Numeros1 se tienen las acciones:

- a. Se declaran las variables
- b. Se solicitan el número 1, el número 2 y el número 3.

- c. Se leen en a, b y c.
- d. Se compara si $a > b$. Si es así, entonces:
 - 1. Se compara si $a > c$. Si es así, entonces:
 - a. Se imprime a como el mayor.
 - 2. Si no:
 - a. Se imprime c como el mayor.
 - 3. Fin del if.
- e. Si no:
 - 1. Se compara si $b > c$. Si es así, entonces:
 - a. Se imprime b como el mayor.
 - 2. Si no:
 - a. Se imprime c como el mayor.
 - 3. Fin del if.
- f. Fin del if.
- g. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

En la zona de descarga del capítulo 4 de la Web del libro está disponible el ejercicio resuelto.



Ejercicio	Descripción
Ejercicio 4.1.4.4	Lee cuatro números e imprime el mayor

Sugerencia: En este momento se sugiere ir al apartado 4.1.2 y estudiar la parte correspondiente a Expresiones lógicas complejas (si no lo ha hecho) y regresar a este punto.

Ejercicio 4.1.4.5

Elaborar un algoritmo que lea tres números y que imprima el mayor. Se supone que son números diferentes. Es la segunda vez que vamos a solucionar este problema.

Segundo método de solución: Utilizando if-then-else y AND.

(Primero hágalo usted; después compare la solución)

```

Algoritmo MAYOR 3 NUMEROS
Clase Mayor3Numeros2
1. Método principal()
  a. Declarar variables
    a, b, c: Entero
  b. Solicitar número 1, número 2, número 3
  c. Leer a, b, c
  d. if (a > b)AND(a > c) then
    1. Imprimir a
  e. else
    1. if b > c then
      a. Imprimir b
  
```



Sugerencia: En este momento se sugiere ir al apartado 4.1.2 y estudiar la parte correspondiente a Expresiones lógicas complejas (si no lo ha hecho) y regresar a este punto.

```

        2. else
            a. Imprimir c
        3. endif
    f. endif
    g. Fin Método principal
Fin Clase Mayor3Numeros2
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Mayor3Numeros2.java

Explicación:

En el Método principal de la Clase Mayor3Numeros2 se tienen las acciones:

- a. Se declaran las variables.
- b. Se solicitan el número 1, el número 2 y el número 3.
- c. Se leen en a, b y c.
- d. Se compara si $(a > b)$ y $(a > c)$. Si se cumple, entonces:
 1. Se imprime a como el mayor.
- e. Si no:
 1. Se compara si $b > c$. Si se cumple, entonces:
 - a. Se imprime b como el mayor.
 2. Si no:
 - a. Se imprime c como el mayor.
 3. Fin del if.
- f. Fin del if.
- g. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Tabla 4.1: ejercicios resueltos disponibles en la zona de descarga del capítulo 4 de la Web del libro.

Ejercicio	Descripción
Ejercicio 4.1.4.6	Lee cuatro números e imprime el mayor
Ejercicio 4.1.4.7	Lee tres lados y dice el tipo de triángulo que es
Ejercicio 4.1.4.8	Realiza cálculos con la ecuación cuadrática

4.2 La selección simple (if-then)

Esta estructura de selección permite controlar la ejecución de acciones cuando existe una sola alternativa de acción. Se utiliza cuando alguna acción o conjunto de acciones está condicionada para que se lleve a cabo su ejecución, pero no se tiene una opción alterna.

Formato:

```
if condición then
    Acción(es)
endif
```

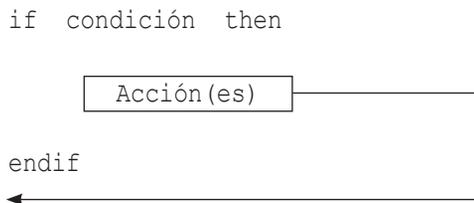
En donde:

if (Si)	Identifica la estructura de control de selección.
then (Entonces)	Indica el curso de acción que se debe seguir si se cumple la condición.
endif	Indica el fin de la estructura de selección (del if).

Funcionamiento:

Al llegar al if se evalúa la condición:

a) Si se cumple, se ejecuta(n) la(s) acción(es) del then y luego salta hasta la siguiente después del endif (fin del if).



b) Si no se cumple, salta hasta después del endif, es decir, no hace nada.

Ejercicio:

Siguiendo con el mismo problema de cálculo del sueldo, ahora se otorga un incentivo del 5% del sueldo si el empleado trabajó más de 40 horas; es decir, al sueldo se le agrega el 5% del mismo sueldo.

A continuación se tiene el algoritmo:

Algoritmo CALCULO SUELDO CON INCENTIVO

Clase Empleado4

1. Método principal()

a. Declarar variables

nombreEmp: Cadena

horasTrab: Entero

cuotaHora, sueldo: Real

b. Solicitar nombre del empleado, número de horas trabajadas, cuota por hora

c. Leer nombreEmp, horasTrab, cuotaHora

d. sueldo = horasTrab * cuotaHora

e. if horasTrab > 40 then

1. sueldo = sueldo + (sueldo * 0.05)

```

f. endif
g. Imprimir nombreEmp, sueldo
h. Fin Método principal
Fin Clase Empleado4
Fin

```



Práctica: En este momento se recomienda ir al siguiente punto de ejercicios resueltos para la selección simple (if-then) y estudiar algunos ejemplos. Además, si quiere más práctica, puede resolver algunos de los ejercicios propuestos al final del capítulo para aplicar el if-then.

En la zona de descarga de la Web del libro está disponible:
Programa en Java: Empleado4.java

Como se puede ver, una vez calculado el sueldo se verifica si trabajó más de 40 horas. Si es así, se le adiciona el 5% del mismo sueldo. (Nótese que aquí no aparece otra alternativa).

Práctica: En este momento se recomienda ir al siguiente punto de ejercicios resueltos para la selección simple (if-then) y estudiar algunos ejemplos. Además, si quiere más práctica, puede resolver algunos de los ejercicios propuestos al final del capítulo para aplicar el if-then.

4.2.1 Ejercicios resueltos para la selección simple (if-then)

A continuación se presentan ejercicios resueltos; se le recomienda que primero haga usted el algoritmo y después compare su solución con la del libro.

Ejercicio 4.2.1.1

Elaborar un algoritmo que lea tres números y que imprima el mayor. Se supone que son números diferentes. Es la tercera ocasión que solucionamos este problema.

Tercer método de solución: Utilizando if-then y AND.

(Primero hágalo usted; después compare la solución)

```

Algoritmo MAYOR 3 NUMEROS
Clase Mayor3Numeros3
1. Método principal()
a. Declarar variables
   a, b, c: Entero
b. Solicitar número 1, número 2, número 3
c. Leer a, b, c
d. if (a > b)AND(a > c) then
   1. Imprimir a
e. endif
f. if (b > a)AND(b > c) then
   1. Imprimir b
g. endif
h. if (c > a)AND(c > b) then
   1. Imprimir c

```

```

        i. endif
        j. Fin Método principal
    Fin Clase Mayor3Numeros3
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Mayor3Numeros3.java



Explicación:

En el Método principal de la Clase Mayor3Numeros3, se tienen las acciones:

- a. Se declaran las variables.
- b. Se solicitan el número 1, el número 2 y el número 3.
- c. Se leen los datos en a, b y c.
- d. Se compara si $(a > b)$ y $(a > c)$. Si se cumple, entonces:
 1. Se imprime a como el mayor.
- e. Fin del if.
- f. Se compara si $(b > a)$ y $(b > c)$. Si se cumple, entonces:
 1. Se imprime b como el mayor.
- g. Fin del if.
- h. Se compara si $(c > a)$ y $(c > b)$. Si se cumple, entonces:
 1. Se imprime c como el mayor.
- i. Fin del if.
- j. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 4.2.1.2

Elaborar un algoritmo que lea cinco números y que imprima el mayor. Se supone que son números diferentes. Restricciones: usar if-then, no usar else ni AND.

(Primero hágalo usted; después compare la solución)

```

Algoritmo MAYOR 5 NUMEROS
Clase Mayor5Numeros1
1. Método principal()
    a. Declarar variables
        a, b, c, d, e, mayor: Entero
    b. Solicitar número 1, número 2, número 3,
        número 4, número 5
    c. Leer a, b, c, d, e
    d. mayor = a
    e. if b > mayor then
        1. mayor = b
    f. endif
    g. if c > mayor then
        1. mayor = c
    h. endif
    i. if d > mayor then
        1. mayor = d

```

```

j. endif
k. if e > mayor then
    1. mayor = e
l. endif
m. Imprimir mayor
n. Fin Método principal
Fin Clase Mayor5Numeros1
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Mayor5Numeros1.java

Explicación:

En el Método principal de la Clase Mayor5Numeros1 se tienen las acciones:

- a. Se declaran las variables.
- b. Se solicitan el número 1, el número 2, el número 3, el número 4 y el número 5.
- c. Se leen en a, b, c, d y e.
- d. Se coloca en mayor el valor de a.
- e. Se compara si $b > mayor$. Si se cumple, entonces:
 1. Se coloca en mayor el valor de b.
- f. Fin del if.
- g. Se compara si $c > mayor$. Si se cumple, entonces:
 1. Se coloca en mayor el valor de c.
- h. Fin del if.
- i. Se compara si $d > mayor$. Si se cumple, entonces:
 1. Se coloca en mayor el valor de d.
- j. Fin del if.
- k. Se compara si $e > mayor$. Si se cumple, entonces:
 1. Se coloca en mayor el valor de e.
- l. Fin del if.
- m. Se imprime mayor, que contiene el número mayor.
- n. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Tabla 4.2: ejercicios resueltos disponibles en la zona de descarga del capítulo 4 de la Web del libro.

Ejercicio	Descripción
Ejercicio 4.2.1.3	Lee el tamaño de un ángulo e imprime el tipo que es
Ejercicio 4.2.1.4	Lee dos ángulos e imprime si son iguales o diferentes
Ejercicio 4.2.1.5	Realiza cálculos logarítmicos de un ángulo
Ejercicio 4.2.1.6	Calcula equivalencias de pies a yardas y otras
Ejercicio 4.2.1.7	Realiza cálculos con la segunda ley de Newton

4.3 La selección múltiple (switch)

Esta estructura de selección permite controlar la ejecución de acciones cuando se tienen más de dos opciones alternativas de acción.

Formato:

```
switch selector
  1: Acción(es)
  2: Acción(es)
  3: Acción(es)
  4: Acción(es)
default
  Acción(es)
endswitch
```

En donde:

switch	Identifica la estructura de selección múltiple.
selector	Es una variable de tipo Entero, Carácter o algún tipo de dato ordinal (que esté constituido por un conjunto ordenado y finito de valores), que traerá un valor que indicará el caso por ejecutar.
1, 2, 3, 4	Son las etiquetas que identifican cada caso de acuerdo a los valores que puede tomar <code>selector</code> .
Acción(es)	Es una acción o conjunto de acciones en pseudocódigo.
default	Si <code>selector</code> no toma ninguno de los valores colocados, se va por esta opción de <code>default</code> .
endswitch	Indica el fin de la estructura <code>switch</code> .

Funcionamiento:

Si `selector` es 1:

Se ejecuta(n) la(s) acción(es) del caso uno y luego se va hasta después del `endswitch`.

Si `selector` es 2:

Se ejecuta(n) la(s) acción(es) del caso dos y luego se va hasta después del `endswitch`.

Si `selector` es 3:

Se ejecuta(n) la(s) acción(es) del caso tres y luego se va hasta después del `endswitch`.

Si `selector` es 4:

Se ejecuta(n) la(s) acción(es) del caso cuatro y luego se va hasta después del `endswitch`.

Si no se cumple ninguno de los casos anteriores, se ejecuta(n) la(s) acción(es) de la opción `default` y luego se va después del `endswitch`.

Nota: El `default` es opcional; en caso de no estar presente la estructura quedará así:



El `default` es opcional; en caso de no estar presente la estructura quedará así:

```
switch selector
  1: Acción(es)
  2: Acción(es)
  3: Acción(es)
  4: Acción(es)
endswitch
```

Y si `selector` no toma ninguno de los posibles valores, simplemente se salta hasta después de `endswitch`, es decir, no hace nada.

```

switch selector
  1: Acción(es)
  2: Acción(es)
  3: Acción(es)
  4: Acción(es)
endswitch

```

Y si `selector` no toma ninguno de los posibles valores, simplemente se salta hasta después de `endswitch`, es decir, no hace nada.

Ejercicio:

Elaborar un algoritmo que lea el número de día (un valor entre 1 y 7) e imprima domingo si es 1, lunes si es 2, ..., sábado si es 7.

```

Algoritmo DICE DIA
Clase DiceDia1
  1. Método principal()
    a. Declarar variables
       numDia: Entero
    b. Solicitar número de día
    c. Leer numDia
    d. switch numDia
       1: Imprimir "Domingo"
       2: Imprimir "Lunes"
       3: Imprimir "Martes"
       4: Imprimir "Miércoles"
       5: Imprimir "Jueves"
       6: Imprimir "Viernes"
       7: Imprimir "Sábado"
    e. default
       1. Imprimir "No está en el rango de 1 a 7"
    f. endswitch
    g. Fin Método principal
  Fin Clase DiceDia1
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: DiceDia1.java

Explicación:

En el Método principal de la Clase DiceDia1 se tienen las acciones:

- Se declara la variable para leer el número de día.
- Se solicita el número de día.
- Se lee en `numDia`.
- Se plantea la estructura de selección switch:
Si `numDia` es 1 se imprime "Domingo".

- Si numDia es 2 se imprime "Lunes".
- Si numDia es 3 se imprime "Martes".
- Si numDia es 4 se imprime "Miércoles".
- Si numDia es 5 se imprime "Jueves".
- Si numDia es 6 se imprime "Viernes".
- Si numDia es 7 se imprime "Sábado".
- e. Si no toma (default) ningún valor de los anteriores:
Imprime "No está en el rango de 1 a 7".
- f. Fin del switch.
- g. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Otros aspectos que la estructura switch permite:

- a. Colocar como etiquetas de casos a más de un valor. Podremos hacerlo como sigue:

```

Algoritmo DICE PAR O IMPAR
Clase DiceParImpar
1. Método principal()
  a. Declarar variables
    num: Entero
  b. Solicitar número entero (0-9)
  c. Leer num
  d. switch num
    0,2,4,6,8: Imprimir num, "es par"
    1,3,5,7,9: Imprimir num, "es impar"
  e. default
    1. Imprimir num, "No está en el rango 0-9"
  f. endswitch
  g. Fin Método principal
Fin Clase DiceParImpar
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: DiceParImpar.java



Explicación:

En el Método principal de la Clase DiceParImpar se tienen las acciones:

- a. Se declara la variable para leer el número.
- b. Se solicita el número.
- c. Se lee el número.
- d. Se plantea la selección switch:
 - Si num es 0, 2, 4, 6 ó 8 imprime "es par".
 - Si num es 1, 3, 5, 7 ó 9 imprime "es impar".

- e. Si no (default) toma ningún valor de los anteriores:
Imprime “No está en el rango 0-9”.
 - f. Fin del switch.
 - g. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.
- b. O bien, se pueden utilizar tipos de datos carácter. Podremos hacerlo como sigue:

```

Algoritmo DICE SI ES VOCAL O NO
Clase DiceVocal
1. Método principal()
  a. Declarar variables
    car: Carácter
  b. Solicitar carácter
  c. Leer car
  d. switch car
    'a','e','i','o','u':
      Imprimir car, "Es una vocal minúscula"
    'A','E','I','O','U':
      Imprimir car, "Es una vocal mayúscula"
  e. default
    1. Imprimir car, "No es vocal"
  f. endswitch
  g. Fin Método principal
Fin Clase DiceVocal
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: DiceVocal.java

Explicación:

En el Método principal de la Clase DiceVocal se tienen las acciones:

- a. Se declara la variable para leer el carácter.
- b. Se solicita el carácter.
- c. Se lee en car.
- d. Se plantea la selección switch:
Si car está entre 'a','e','i','o','u':
Imprime “Es una vocal minúscula”.
Si car está entre 'A','E','I','O','U':
Imprime “Es una vocal mayúscula”.
- e. Si no (default):
1. Imprime “No es vocal”.
- f. Fin del switch.
- g. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.



Práctica: En este momento se recomienda ir al siguiente punto de ejercicio resuelto para la selección múltiple (switch) y estudiarlo. Además, si quiere más práctica, puede resolver algunos de los ejercicios propuestos al final del capítulo para aplicar el switch.

Práctica: En este momento se recomienda ir al siguiente punto de ejercicio resuelto para la selección múltiple (switch) y estudiarlo. Además, si quiere más práctica, puede resolver algunos de los ejercicios propuestos al final del capítulo para aplicar el switch.

4.3.1 Ejercicio resuelto para la selección múltiple (switch)

Ejercicio 4.3.1.1

Una empresa vende hojas de hielo seco, con las condiciones siguientes:

- Si el cliente es tipo 1 se le descuenta el 5 %
- Si el cliente es tipo 2 se le descuenta el 8 %
- Si el cliente es tipo 3 se le descuenta el 12 %
- Si el cliente es tipo 4 se le descuenta el 15 %

Cuando el cliente realiza una compra se generan los datos siguientes:

Nombre del cliente

Tipo de cliente (1, 2, 3, 4)

Cantidad de hojas compradas

Precio por hoja

Elabore un algoritmo que lea estos datos, haga cálculos e imprima:

Nombre del cliente

Subtotal por pagar (Cantidad de hojas X Precio por hoja)

Descuento (el porcentaje correspondiente del Subtotal por pagar)

Neto por pagar (Subtotal – Descuento)

Utilizar switch.

(Primero hágalo usted; después compare la solución)

```
Algoritmo CLIENTE HOJAS HIELO SECO
Clase Cliente1
1. Método principal()
  a. Declarar variables
     nombreClie: Cadena
     tipoClie, cantidad: Entero
     precioUni, subTotal, descuento, netoPagar: Real
  b. Solicitar nombre, tipo cliente, cantidad,
     precio unitario
  c. Leer nombreClie, tipoClie, cantidad, precioUni
  d. subTotal = cantidad * precioUni
  e. switch tipoClie
```

```

        1: descuento = subTotal * 0.05
        2: descuento = subTotal * 0.08
        3: descuento = subTotal * 0.12
        4: descuento = subTotal * 0.15
    f. endswitch
    g. netoPagar = subTotal - descuento
    h. Imprimir nombreClie, subTotal, descuento, netoPagar
    i. Fin Método principal
    Fin Clase Clientel
    Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Cliente1.java

Explicación:

En el Método principal de la Clase Clientel se tienen las acciones:

- a. Se declaran las variables.
- b. Se solicitan los datos.
- c. Se leen en nombreClie, tipoClie, cantidad, precioUni.
- d. Se calcula el subtotal.
- e. Se plantea el switch con el selector tipoClie:
 - Si tipoClie es 1 calcula descuento con el 5%.
 - Si tipoClie es 2 calcula descuento con el 8%.
 - Si tipoClie es 3 calcula descuento con el 12%.
 - Si tipoClie es 4 calcula descuento con el 15%.
- f. Fin del switch
- g. Se calcula netoPagar.
- h. Imprime nombreClie, subTotal, descuento y netoPagar.
- i. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Nota: En caso que se permita que el cliente pueda ser de tipo diferente de 1 a 4, y sólo tengan descuento estos tipos, se agregaría el default colocándole cero a descuento; esa parte quedaría:

En caso que se permita que el cliente pueda ser de tipo diferente de 1 a 4, y sólo tengan descuento estos tipos, se agregaría el default colocándole cero a descuento; esa parte quedaría:

```

e. switch tipoClie
    1: descuento = subTotal * 0.05
    2: descuento = subTotal * 0.08
    3: descuento = subTotal * 0.12
    4: descuento = subTotal * 0.15
f. default
    1. descuento = 0
g. endswitch

```

```

e. switch tipoClie
    1: descuento = subTotal * 0.05
    2: descuento = subTotal * 0.08
    3: descuento = subTotal * 0.12
    4: descuento = subTotal * 0.15
f. default
    1. descuento = 0
g. endswitch

```

4.4 Ejercicios propuestos

1. Elaborar un algoritmo para calcular e imprimir el precio de un terreno del cual se tienen los siguientes datos: largo, ancho y precio por metro cuadrado. Si el terreno tiene más de 400 metros cuadrados se hace un descuento de 10 %.
2. Igual al ejercicio anterior, sólo que si el terreno tiene más de 500 metros cuadrados el descuento es de 17 % y si tiene más de 1000 el descuento es de 25 %.
3. Elabore un algoritmo para calcular e imprimir los valores de X y Y, teniendo como entrada el valor de X y calculando el de Y de acuerdo con las siguientes condiciones:
Si $X < 0$ entonces $Y = 3X + 6$
Si $X \geq 0$ entonces $Y = X^2 + 6$
4. Elaborar un algoritmo que imprima el costo de un pedido de un artículo del cual se tiene la descripción, la cantidad pedida y el precio unitario. Si la cantidad pedida excede de 50 unidades, se hace un descuento de 15%.
5. Un cliente ordena cierta cantidad de hojas de hielo seco, viguetas y armazones; las hojas de hielo seco tienen 20% de descuento y las viguetas 15%. Los datos que se tienen por cada tipo de artículo son: la cantidad pedida y el precio unitario. Además, si se paga de contado todo tiene un descuento del 7%.
Elaborar un algoritmo que calcule e imprima el costo total de la orden, tanto para el pago de contado como para el caso de pago de crédito.
6. Igual que el ejercicio 4.3.1.1, pero además: si la cantidad de hojas de hielo seco solicitada es mayor que 50, se hace un descuento adicional de 5%; en caso de que sea mayor que 100 el descuento adicional es de 10 %.
7. Elaborar un algoritmo que lea los datos de un estudiante (nombre y tres calificaciones parciales) e imprima el nombre y la calificación final de acuerdo a lo siguiente: para aprobar el curso debe tener 70 o más en cada una de las tres calificaciones, la calificación final será el promedio. En caso de haber reprobado uno o más exámenes ordinarios, la calificación final será NA (No Acreditado).
8. De acuerdo con la clase de sus ángulos, los triángulos se clasifican en:
Rectángulo tiene un ángulo recto (igual a 90°)
Obtusángulo tiene un ángulo obtuso (mayor que 90° pero menor 180°)
Acutángulo los tres ángulos son agudos (menor que 90°)
Elaborar un algoritmo que permita leer el tamaño de los tres ángulos (A,B,C) de un triángulo e imprima qué tipo es.
9. En un almacén de venta de trajes, si se compra uno se hace el 50% de descuento, si se compran 2, el 55%, si se compran 3, el 60%, y si compra más de 3, el 65%. Elaborar un algoritmo que lea la cantidad de trajes y el precio unitario (todos tienen el mismo precio) e imprima el subtotal por pagar, el descuento y el total por pagar.
10. Dos triángulos son congruentes si tienen la misma forma y tamaño, es decir, sus ángulos y lados correspondientes son iguales. Elaborar un algoritmo que lea los tres ángulos y lados de dos triángulos e imprima si son congruentes.
11. Un trapecio es isósceles si sus dos ángulos de la base son iguales. Elaborar un algoritmo que lea los ángulos A y B de la base y que imprima si el trapecio es isósceles o no.

12. Elaborar un algoritmo que permita leer los datos de un empleado (nombre, tipo de empleado, número de horas trabajadas y cuota que se le paga por hora) y calcule e imprima el sueldo por pagar. Si el empleado es tipo 1 se le pagan las horas extras (más de 40 horas) a 1.5 de la cuota por hora, si es tipo 2, a 2, si es tipo 3, a 2.5, y si es tipo 4, a 3 veces la cuota por hora.
13. Se tiene un terreno A cuadrado que mide LADO metros por lado a un precio COSTOA por metro cuadrado y se tiene un terreno B rectangular que mide BASE metros de base y ALTURA metros de altura a un COSTOB por metro cuadrado. Elaborar un algoritmo que lea los datos de los dos terrenos e imprima cuál es el más barato o si cuestan igual.
14. Elabore un algoritmo que lea el número de mes entre 1 y 12 y que imprima el nombre del mes correspondiente: si es 1 “Enero”, si es 2 “Febrero”,..., etcétera.
15. En el hotel Guamúchil se hace un descuento del 10% si el cliente se hospeda más de 5 días, del 15% si se hospeda más de 10 días y del 20% si se hospeda más de 15 días. Elaborar un algoritmo que lea el número de días y el precio diario de la habitación e imprima el subtotal por pagar, el descuento y el total por pagar.
16. Elaborar un algoritmo que permita hacer conversiones de temperaturas entre grados Fahrenheit, Celsius, Kelvin y Rankine. Primero debe preguntar qué tipo de grados quiere convertir. Por ejemplo: si se le indica que se desea convertir una temperatura en grados Fahrenheit, debe leer la cantidad de grados y luego calcular e imprimir su equivalente en grados Celsius, Kelvin y Rankine, y así debe hacer lo mismo para cada uno de los otros tipos. Para convertir a Celsius a la temperatura Fahrenheit se le resta 32 y se multiplica por 5/9. Para convertir a Kelvin, se le suma 273 a los grados Celsius. Para convertir a Rankine a los grados Fahrenheit se le suma 460.
17. Elaborar un algoritmo que permita leer el tamaño de un ángulo en radianes o en grados (debe preguntar en qué lo va a leer) e imprima el seno hiperbólico, coseno hiperbólico y tangente hiperbólica. Si lo lee en grados, debe hacer la conversión a radianes. En el ejercicio 3.5.9 se indica cómo hacer los cálculos.
18. Elaborar un algoritmo que permita leer los datos de un aspirante a ingresar a la carrera de Ingeniería Industrial y de Sistemas de la Universidad de Sonora: nombre del aspirante, promedio en bachillerato y tipo de bachillerato (1-Físico matemático, 2-etc.,..., 5) y que imprima que es aceptado si tiene un promedio mayor a 90, o bien, si tiene un promedio entre 80 y 90 y trae bachillerato Físico matemático. En caso de no ser así imprimir rechazado.
19. Elaborar un algoritmo que permita leer los datos de un automóvil (marca, origen y costo) imprima el impuesto por pagar y el precio de venta (incluido el impuesto). Si el origen es Alemania el impuesto es 20%, si es de Japón el impuesto es 30%, si es de Italia, 15%, y si es de USA, 8%.

20. Un sistema de ecuaciones lineales

$$ax + by = c$$

$$dx + ey = f$$

se puede resolver con las fórmulas

$$X = \frac{ce - bf}{ae - bd} \quad Y = \frac{af - cd}{ae - bd} \quad \text{si } (ae - bd) \neq 0.$$

Elaborar un algoritmo que lea los coeficientes a,b,c,d,e y f, y que calcule e imprima los valores de X y Y. Si $(ae - bd) <> 0$, debe calcular e imprimir los valores de X y Y; en caso contrario debe imprimir un mensaje que indique que no tiene solución.

21. Elaborar un algoritmo que permita leer los datos de un empleado (nombre, tipo de empleado y sueldo), imprima el incremento de sueldo y su nuevo sueldo de acuerdo a lo siguiente: si es tipo de empleado 1 se le aumentará el 5%, si es tipo 2 se le aumentará el 7%, si es 3, el 9%, si es 4, el 12% y si es 5, el 15%.
22. Elaborar un algoritmo que permita leer una letra e imprima si es vocal o si es consonante.
23. Elaborar un algoritmo que permita leer el tamaño de un ángulo en radianes o en grados y que imprima la tangente, secante, cotangente y cosecante. Debe preguntar en qué tiene el tamaño del ángulo y, dependiendo de si es en grados o radianes, los cálculos deben hacerse según corresponda.
24. Elaborar un algoritmo que permita leer 4 números e imprima el mayor. Debe validar que sean diferentes, es decir, si hay números iguales debe enviar un mensaje de error.
25. Elaborar un algoritmo que permita hacer conversiones entre pesos, yenes, pesetas y marcos. Debe preguntar qué moneda desea convertir; por ejemplo, si indica que yenes, debe leer cuántos yenes comprará y cuánto cuesta un yen en pesos, pesetas y marcos, luego debe imprimir cuánto es en cada una de las monedas. Así lo hará también para cada una de las otras monedas.
26. Una temperatura en grados Celsius (C) se puede convertir a su equivalente Fahrenheit (F) con la fórmula:

$$F = \frac{9}{5} C + 32$$

De Fahrenheit a Centígrados con la fórmula: $C = (F-32) \frac{5}{9}$

Elaborar un algoritmo que pregunte qué quiere convertir. Si quiere convertir Celsius, que lea la temperatura en grados Celsius y calcule e imprima la temperatura Fahrenheit equivalente. Si quiere convertir Fahrenheit debe hacer lo propio.

4.5 Resumen de conceptos que debe dominar

- La selección doble (if-then-else)
- Expresiones lógicas
 - Simples (>, <, >=, <=, !=, ==)
 - Complejas (AND, OR, XOR, NOT)
- if's anidados
- La selección simple (if-then)
- La selección múltiple (switch)

4.6 Contenido de la página Web de apoyo

El material marcado con asterisco () sólo está disponible para docentes.*

4.6.1 Resumen gráfico del capítulo

4.6.2 Autoevaluación

4.6.3 Programas en Java

4.6.4 Ejercicios resueltos

4.6.5 Power Point para el profesor (*)

5

La repetición

Contenido

- 5.1 La repetición do...while
 - 5.1.1 Contadores y acumuladores
 - 5.1.2 Ejercicios resueltos para la repetición do...while
 - 5.1.3 Ejercicios propuestos para la repetición do...while
- 5.2 La repetición for
 - 5.2.1 For anidados
 - 5.2.2 Ejercicios resueltos para la repetición for
 - 5.2.3 Simulación del for con do...while
 - 5.2.4 Ejercicios propuestos para la repetición for
- 5.3 La repetición while
 - 5.3.1 Simulación del do...while con while
 - 5.3.2 Simulación del for con while
 - 5.3.3 Ejercicios resueltos para la repetición while
 - 5.3.4 Ejercicios propuestos para la repetición while
- 5.4 Resumen de conceptos que debe dominar
- 5.5 Contenido de la página Web de apoyo
 - El material marcado con asterisco (*) sólo está disponible para docentes.*
 - 5.5.1 Resumen gráfico del capítulo
 - 5.5.2 Autoevaluación
 - 5.5.3 Programas en Java
 - 5.5.4 Ejercicios resueltos
 - 5.5.5 Power Point para el profesor (*)

Objetivos del capítulo

- Estudiar las estructuras de control de repetición do...while, for y while.
- Asimilar la forma de emitir la información en forma de reporte y cómo utilizar contadores y acumuladores.
- Aprender a calcular promedios o medias aritméticas.
- Aprender a obtener el mayor y el menor de un conjunto de datos y a utilizar un ciclo repetitivo anidado dentro de otro, es decir, un do...while dentro de otro do...while utilizando todas las estructuras aprendidas hasta este momento, además de los conceptos y estructuras nuevas que se presentan en este capítulo.
- Estudiar cómo plantear un for dentro de otro for, un do...while dentro de un for y un for dentro de un do...while.
- Aprender a simular la estructura for con do... while.
- Aprender a anidar un while dentro de otro while (se combinará con el uso de las estructuras estudiadas hasta ahora, por ejemplo while con for y do...while).
- Ilustrar la forma de plantear ciclos for y do...while con while.
- Aplicar lo aprendido en la solución de ejercicios resueltos y propuestos.

Competencias

- Competencia general del capítulo
 - *Analizar problemas y diseñar algoritmos que los solucionen aplicando la repetición do...while, for y while.*
- Competencias específicas del capítulo
 - Diseña algoritmos aplicando la repetición do...while.
 - Elabora algoritmos aplicando la repetición for.
 - Desarrolla algoritmos aplicando la repetición while.

Introducción

Con el estudio del capítulo anterior usted ya domina la selección y cómo diseñar algoritmos usando esa estructura de control.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos utilizando la estructura de control: la repetición.

Se explica que la repetición es una estructura que permite controlar la ejecución de acciones que se deben realizar en más de una ocasión, es decir, se deben repetir, que con esta estructura se forman ciclos repetitivos y que tiene tres formas: la repetición `do...while`, la repetición `for` y la repetición `while`.

Se enseña que la repetición `do...while` sirve para plantear situaciones en las que una acción o conjunto de acciones se repetirán mientras se cumpla una condición en un rango de 1 a n veces.

Se expone que la repetición `for` sirve para plantear situaciones en las que una acción o conjunto de acciones se repetirán un número de veces conocido de antemano: n veces.

Se explica que la repetición `while` sirve para plantear situaciones en las que una acción o conjunto de acciones se repetirán mientras se cumpla una condición en un rango de cero a n veces.

También se trata la forma de emitir la información en forma de reporte y se detalla la manera de cómo utilizar contadores y acumuladores, cómo obtener promedios o medias aritméticas al procesar varios elementos, cómo obtener el mayor y el menor de un conjunto de datos y cómo utilizar un ciclo repetitivo anidado dentro de otro, es decir, un `do...while` dentro de otro `do...while`, lógicamente, utilizando todas las estructuras aprendidas hasta este momento, además de los conceptos y estructuras nuevas que se presentan en este capítulo.

Se enseña a plantear un `for` dentro de otro `for`, un `do...while` dentro de un `for` y un `for` dentro de un `do...while`. Asimismo, cómo simular la estructura `for` con `do...while`.

También se describe cómo anidar un `while` dentro de otro `while` y se combina con el uso de las estructuras estudiadas hasta ahora. Por ejemplo, cómo anidar `while` con `for` y `do...while`. Además, se trata la forma de plantear ciclos `for` y `do...while` con `while`.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejercite estudiando los problemas planteados en los ejercicios resueltos y propuestos. Al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro, luego verifique sus resultados con los del libro, analice las diferencias y vea si tiene errores. Al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no está igual que el del libro, no necesariamente está mal. Usted debe ir aprendiendo a analizar las diferencias y a comprender que a veces, aunque haya diferencias, las dos soluciones están correctas.

En el siguiente capítulo se estudia la estructura de datos denominada arreglos, qué es, cómo se representa y su uso en pseudocódigo.

5.1 La repetición do...while

La repetición do...while permite controlar la ejecución de acciones en forma repetitiva, mientras la condición de control del ciclo repetitivo sea verdadera.

Formato:

```
do
    Acción(es)
while condición
```

En donde:

do	Identifica la estructura como un ciclo repetitivo e indica el inicio de éste.
Acción(es)	Son las acciones que se ejecutan dentro del ciclo.
while	Indica el fin del ciclo y significa que “mientras” se cumpla la condición, vuelve al inicio del ciclo do; en caso contrario, se sale del ciclo do...while.
condición	Es una expresión lógica que controla la repetición del ciclo.

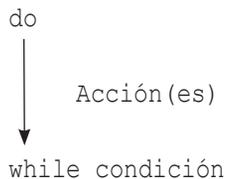


Este tipo de repetición se ejecuta de 1 a N veces ya que las acciones del ciclo se ejecutan por lo menos una vez y se pueden ejecutar cualquier cantidad (N) de veces.

Nota: Este tipo de repetición se ejecuta de 1 a N veces ya que las acciones del ciclo se ejecutan por lo menos una vez y se pueden ejecutar cualquier cantidad (N) de veces.

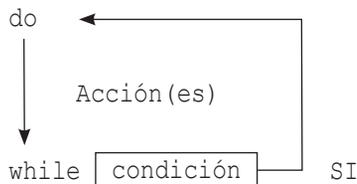
Funcionamiento:

1. Llega al do, entra al ciclo y ejecuta la(s) acción(es).



2. Llega al while y se evalúa la condición.

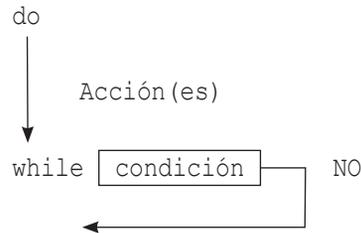
- a) Si se cumple, va hacia el inicio de la estructura do, lo que significa volver a ejecutar lo que esta dentro del do.





Cabe aclarar que esta estructura originalmente se inventó como do-until: hacer lo que está dentro del ciclo hasta que se cumpla la condición. Sin embargo, el inventor del lenguaje C la implementó como do...while, es decir, hacer lo que está dentro del ciclo mientras se cumpla la condición. Y en virtud de que el lenguaje C es la base de C++, Java y seguramente de los lenguajes que se diseñen en el futuro, es que esta estructura se ha convertido en el estándar; por ello, en este libro la usaremos de esta forma.

b) Si no se cumple, va a la siguiente acción después del while; es decir, sale del ciclo repetitivo.



Nota: Cabe aclarar que esta estructura originalmente se inventó como do-until: hacer lo que está dentro del ciclo hasta que se cumpla la condición. Sin embargo, el inventor del lenguaje C la implementó como do...while, es decir, hacer lo que está dentro del ciclo mientras se cumpla la condición. Y en virtud de que el lenguaje C es la base de C++, Java y seguramente de los lenguajes que se diseñen en el futuro, es que esta estructura se ha convertido en el estándar; por ello, en este libro la usaremos de esta forma.

Ejemplo:

Elaborar un algoritmo que calcule e imprima el sueldo de varios empleados. Cada empleado se tratará en forma similar al primer problema que planteamos en el capítulo 3 cuando estudiamos la secuenciación.

A continuación se tiene el algoritmo de la solución:

```

Algoritmo CALCULA SUELDOS DE EMPLEADOS
Clase Empleados1
1. Método principal()
  a. Declarar variables
     nombreEmp: Cadena
     horasTrab: Entero
     cuotaHora, sueldo: Real
     desea: Carácter
  b. do
     1. Solicitar nombre, número de horas trabajadas,
        cuota por hora
     2. Leer nombreEmp, horasTrab, cuotaHora
     3. Calcular sueldo = horasTrab * cuotaHora
     4. Imprimir nombreEmp, sueldo
     5. Preguntar "¿Desea procesar otro empleado (S/N)?"
     6. Leer desea
  c. while desea == 'S'
  d. Fin Método principal
Fin Clase Empleados1
Fin
  
```



En la zona de descarga de la Web del libro está disponible:

Programa en Java: Empleados1.java

Explicación:

En el Método principal de la Clase Empleados1 se tienen las acciones:

- a. Se declaran las variables que ya conocemos: nombreEmp, horasTrab, cuotaHora y sueldo. Además, desea es una variable de tipo carácter que servirá para controlar al ciclo repetitivo.
- b. Se inicia el planteamiento del ciclo repetitivo do:
 1. Se solicitan el nombre, número de horas trabajadas y cuota por hora.
 2. Se leen en nombreEmp, horasTrab y cuotaHora.
 3. Se calcula el sueldo.
 4. Imprime nombreEmp y sueldo.
 5. Se pregunta si “¿Desea procesar otro empleado (S/N)?”, pregunta a la cual se debe contestar S para SÍ o N para NO.
 6. Se lee en desea la respuesta que se dé a la pregunta anterior.
- c. Delimita el fin del ciclo repetitivo. Si se cumple la condición `desea == 'S'`, el control se transfiere hacia el do y entra de nuevo al ciclo repetitivo. En caso de no cumplirse, el control se transfiere hacia la siguiente acción después del while, es decir, se sale del ciclo do. 'S' se pone entre apóstrofes porque es un valor (constante) de tipo carácter.
- d. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Nota 1: La palabra varios significa que se tiene más de un empleado y no se sabe cuántos son. Es por ello que en el paso 5 se pregunta si desea procesar otro empleado, en el paso 6 se lee la respuesta en la variable `desea` y en el paso c, que cierra el ciclo, se plantea la condición `desea == 'S'`. De esta forma se controla que se puedan procesar varios empleados (o lo que corresponda), aun cuando no se conozca cuántos son.

En este capítulo estaremos planteando problemas que tienen esta naturaleza de repetición. Posteriormente plantearemos problemas en los que se conoce de antemano cuántos elementos se procesarán.

Nota 2: Observe que las variables que se utilizan para manejar los datos del primer empleado son las mismas variables que se utilizan para manejar los datos del segundo empleado, y así para todos los empleados que se procesen.

5.1.1 Contadores y acumuladores

En ocasiones, en los ciclos repetitivos se necesita contar el número de veces que se procesa algún elemento, o bien, hacer acumulaciones de cantidades para obtener totalizadores. Veamos el siguiente ejemplo:

En el problema anterior de sueldos de empleados es necesario que los datos se impriman en forma de un reporte que tenga el siguiente formato:



La palabra varios significa que se tiene más de un empleado y no se sabe cuántos son. Es por ello que en el paso 5 se pregunta si desea procesar otro empleado, en el paso 6 se lee la respuesta en la variable `desea` y en el paso c, que cierra el ciclo, se plantea la condición `desea == 'S'`. De esta forma se controla que se puedan procesar varios empleados (o lo que corresponda), aun cuando no se conozca cuántos son.

En este capítulo estaremos planteando problemas que tienen esta naturaleza de repetición. Posteriormente plantearemos problemas en los que se conoce de antemano cuántos elementos se procesarán.

Observe que las variables que se utilizan para manejar los datos del primer empleado son las mismas variables que se utilizan para manejar los datos del segundo empleado, y así para todos los empleados que se procesen.

REPORTE DE EMPLEADOS		
NOMBRE	SUELDO	
XXXXXXXXXXXXXXXXXXXXXXXXXX	99,999.99	} Encabezado
XXXXXXXXXXXXXXXXXXXXXXXXXX	99,999.99	
-	-	} Detalle
-	-	
XXXXXXXXXXXXXXXXXXXXXXXXXX	99,999.99	
TOTAL 999 EMPLEADOS	999,999.99	} Total o pie

Como puede observar, el reporte tiene 3 partes:

- La primera parte es el *encabezado*, que deberá aparecer una sola vez al principio. La primera línea del encabezado es el nombre del reporte, que en este caso es un REPORTE DE EMPLEADOS, la segunda línea son letreros que identifican los datos que aparecerán en el reporte, que en este caso son el NOMBRE y el SUELDO de los empleados.
- La segunda parte es el *detalle* de cada empleado que aparecerá tantas veces como empleados haya. El conjunto de XXXXXXXXXXXXXXXXXXXXXXXX indica que va un dato cadena de caracteres, que es el nombre del empleado; por otro lado, 99,999.99 indica que va un dato de tipo real, el sueldo del empleado.
- La tercera parte es una línea de *total o pie del reporte* que aparecerá una sola vez al final y tiene como propósito mostrar una estadística o resumen de los datos que aparecen en el detalle del reporte:

```
TOTAL 999 EMPLEADOS          999,999.99
```

Los 999 indican que va un dato de tipo entero -el total de empleados-, es decir, el número de empleados que están en el reporte y los 999,999.99 indican que va un dato de tipo real: es un totalizador o acumulador con la suma de los sueldos de todos los empleados. Para el primero se requiere el uso de un contador y para el segundo un acumulador.

¿Que es un contador? Una variable de tipo entero que en nuestro ejemplo podría llamarse `totEmpleados` que tiene como función contener el número de empleados que se procesan. El contador funciona de la forma siguiente: al principio se inicia con cero y dentro del ciclo se incrementa en 1, para así contar a cada empleado procesado. Al final, después del fin del ciclo, se podrá imprimir el contenido del contador, que será el total de empleados procesados. A continuación se muestra el funcionamiento del contador:

```
totEmpleados = 0
do
  Procesar empleado
  --
  totEmpleados = totEmpleados + 1
  --
while desea == 'S'
Imprimir totEmpleados
```

¿Qué es un acumulador? Es una variable de tipo numérico, que en nuestro ejemplo puede ser `totSueldos`, cuya función es contener la suma de un determinado conjunto de datos, que en el ejemplo está representado por los sueldos. La forma de operar del acumulador es la siguiente: al principio se inicia con cero, dentro del ciclo se incrementa con lo que tenga la variable que contiene el dato por acumular, que en este caso es `sueldo`; al final, después del fin del ciclo, se podrá imprimir el contenido del acumulador que es el total de la suma, en este caso de sueldos. A continuación se muestra el funcionamiento del acumulador:

```
totSueldos = 0
do
  Procesar empleado
  --
  totSueldos = totSueldos + sueldo
  --
  --
  --
while desea == 'S'
Imprimir totSueldos
```

A continuación se presenta el algoritmo completo:

Algoritmo CALCULA SUELDOS DE EMPLEADOS

Clase Empleados2

1. Método principal()

a. Declarar variables

nombreEmp: Cadena

horasTrab, totEmpleados: Entero

cuotaHora, sueldo, totSueldos: Real

desea: Carácter

b. Imprimir encabezado

c. `totEmpleados = 0`

d. `totSueldos = 0`

e. do

1. Solicitar nombre, número de horas trabajadas, cuota por hora

2. Leer `nombreEmp`, `horasTrab`, `cuotaHora`

3. Calcular `sueldo = horasTrab * cuotaHora`

4. Imprimir `nombreEmp`, `sueldo`

5. `totEmpleados = totEmpleados + 1`

6. `totSueldos = totSueldos + sueldo`

7. Preguntar "¿Desea procesar otro empleado (S/N)?"

8. Leer `desea`

f. while `desea == 'S'`

g. Imprimir `totEmpleados`, `totSueldos`

h. Fin Método principal

Fin Clase Empleados2

Fin

En la zona de descarga de la Web del libro está disponible:

Programa en Java: `Empleados2.java`



Explicación:

En el Método principal de la Clase Empleados2 se tienen las acciones:

- a. Se declaran las variables que ya conocemos: nombreEmp, horasTrab, cuotaHora, sueldo y desea; además, se declaran totEmpleados, que servirá para contar el total de empleados procesados, y totSueldos, que se utiliza para acumular la suma de los sueldos de todos los empleados.
- b. Se indica que se imprime el encabezado del reporte de acuerdo con el formato. Otra forma de imprimirlo sería así:
 Imprimir “ REPORTE DE EMPLEADOS”
 Imprimir “NOMBRE SUELDO”
 Imprimir “_____”
- c. Se inicia en cero el contador de empleados.
- d. Se inicia en cero el acumulador de sueldos.
- e. Se inicia el planteamiento del ciclo repetitivo do:
 1. Se solicitan el nombre, número de horas trabajadas y cuota por hora.
 2. Se leen en nombreEmp, horasTrab y cuotaHora.
 3. Se calcula el sueldo.
 4. Imprime nombreEmp y sueldo.
 5. Se incrementa el contador de empleados en 1.
 6. Se incrementa el acumulador de sueldos con sueldo.
 7. Se pregunta si “¿Desea procesar otro empleado (S/N)?”, pregunta a la cual se debe contestar S para SÍ o N para NO.
 8. Se lee en desea la respuesta que se dé a la pregunta anterior.
- f. Delimita el fin del ciclo repetitivo. Si se cumple la condición desea == 'S', el control se transfiere hacia el do y entra de nuevo al ciclo repetitivo. En caso de no cumplirse, el control se transfiere hacia la siguiente acción despues del while, es decir, se sale del ciclo.
- g. Se imprimen el total de empleados y el total de sueldos.
- h. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Funciones Break y Continue

La mayoría de los lenguajes proporcionan las funciones Break y Continue.

Break

La función Break interrumpe (termina) la ejecución de un ciclo repetitivo y transfiere el control a la siguiente acción después del ciclo.

Formato:

Break

Ejemplo:

```
do
    .
    Break
    .
while condición
```

Continue

La función `Continue` permite enviar el control al inicio del ciclo repetitivo, es decir, transfiere el control para ejecutar la siguiente repetición del ciclo.

Formato:

```
Continue
```



`Break` y `Continue` son aplicables en ciclos `do...while`, `for` y `while`. Si no están dentro de un ciclo habrá error.

Ejemplo:

```
do
    .
    Continue
    .
while condición
```

Nota: `Break` y `Continue` son aplicables en ciclos `do...while`, `for` y `while`. Si no están dentro de un ciclo habrá error.

En este libro no se le dará aplicación a esas funciones porque, si un programa está bien estructurado, no son necesarias, y este libro se enfoca precisamente en el diseño de programas bien estructurados.

5.1.2 Ejercicios resueltos para la repetición do...while

A continuación se presentan ejercicios resueltos. Se le recomienda que primero haga usted el algoritmo y después compare su solución con la del libro.

Cabe aclarar que por la naturaleza de la repetición `do...while`, los problemas que se plantean en este capítulo tienen una orientación más administrativa porque esta estructura es más natural para resolver problemas de esta índole. Sin embargo, en el punto siguiente (`for`), donde los problemas tienen una orientación más hacia ingeniería, utilizaremos el `do...while` para resolverlos.

Ejercicio 5.1.2.1

Una empresa vende hojas de hielo seco con las condiciones siguientes:

- Si el cliente es tipo 1 se le descuenta el 5 %.
- Si el cliente es tipo 2 se le descuenta el 8 %.
- Si el cliente es tipo 3 se le descuenta el 12 %.
- Si el cliente es tipo 4 se le descuenta el 15 %.

Cuando un cliente realiza una compra se generan los datos siguientes:

- Nombre del cliente
- Tipo de cliente (1,2,3,4)
- Cantidad de hojas compradas
- Precio por hoja

Elaborar un algoritmo que permita procesar varios clientes e imprima el reporte:

REPORT DE CLIENTES			
NOMBRE	SUBTOTAL	DESCUENTO	NETO A PAGAR
XXXXXXXXXXXXXXXXXXXX	99,999.99	99,999.99	99,999.99
XXXXXXXXXXXXXXXXXXXX	99,999.99	99,999.99	99,999.99
-	-	-	-
XXXXXXXXXXXXXXXXXXXX	99,999.99	99,999.99	99,999.99
TOTAL 999 CLIENTES	99,999.99	99,999.99	99,999.99

Cálculos:

- Subtotal = Cantidad de hojas X Precio por hoja
- Descuento es el porcentaje correspondiente del subtotal por pagar
- Neto por pagar = Subtotal – Descuento
- Y la sumatoria de cada uno de estos datos para imprimirlos como totales.

(Primero hágalo usted; después compare la solución)

```

Algoritmo CLIENTES HOJAS HIELO SECO
Clase Clientes1
1. Método principal()
  a. Declarar variables
     nombreClie: Cadena
     tipoClie, cantidad, totClientes: Entero
     precioUni, subTotal, descuento, netoPagar,
     totSubTot, totDescuento, totNeto: Real
     desea: Carácter
  b. Imprimir encabezado
  c. totClientes = 0
     totSubTot = 0
     totDescuento = 0
     totNeto = 0
  d. do
     1. Solicitar nombre, tipo cliente, cantidad,
        precio unitario
     2. Leer nombreClie, tipoClie, cantidad, precioUni
     3. subTotal = cantidad * precioUni
     4. switch tipoClie
        1: descuento = subTotal * 0.05
        2: descuento = subTotal * 0.08
        3: descuento = subTotal * 0.12
        4: descuento = subTotal * 0.15

```

```

5. endswitch
6. netoPagar = subTotal - descuento
7. Imprimir nombreClie, subTotal, descuento, netoPagar
8. totClientes = totClientes + 1
   totSubTot = totSubTot + subTotal
   totDescuento = totDescuento + descuento
   totNeto = totNeto + netoPagar
9. Preguntar "¿Desea procesar otro cliente (S/N)?"
10. Leer desea
e. while desea == 'S'
f. Imprimir totClientes, totSubTot, totDescuento, totNeto
g. Fin Método principal
Fin Clase Clientes1
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Clientes1.java



Explicación:

En el Método principal de la Clase Clientes1 se tienen las acciones:

- a. Se declaran las variables.
- b. Se imprime el encabezado.
- c. Se inician en cero los totales generales.
- d. Se inicia el planteamiento del ciclo repetitivo do:
 1. Se solicitan los datos.
 2. Se leen en nombreClie, tipoClie, cantidad, precioUni.
 3. Se calcula el subtotal.
 4. Se plantea el switch con el selector tipoClie.
 - Si tipoClie es 1 calcula descuento con el 5%.
 - Si tipoClie es 2 calcula descuento con el 8%.
 - Si tipoClie es 3 calcula descuento con el 12%.
 - Si tipoClie es 4 calcula descuento con el 15%.
 5. Fin del switch.
 6. Se calcula el total por pagar.
 7. Imprime nombreClie, subTotal, descuento y netoPagar.
 8. Se incrementan los totales.
 9. Se pregunta "¿Desea procesar otro cliente (S/N)?".
 10. Se lee la respuesta en desea.
- e. Delimita el fin del ciclo repetitivo. Si se cumple la condición desea == 'S', el control se transfiere hacia el do y entra de nuevo al ciclo repetitivo. En caso de no cumplirse, el control se transfiere hacia la siguiente acción después del while, es decir, se sale del ciclo.
- f. Imprime los totales totClientes, totSubTot, totDescuento, totNeto.
- g. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 5.1.2.2

Elaborar un algoritmo que proporcione el siguiente reporte:

ANÁLISIS DE CALIFICACIONES					
NOMBRE	CAL.1	CAL.2	CAL.3	CAL.4	PROMEDIO
XXXXXXXXXXXXXXXXXXXX	99.99	99.99	99.99	99.99	99.99
XXXXXXXXXXXXXXXXXXXX	99.99	99.99	99.99	99.99	99.99
-	-	-	-	-	-
XXXXXXXXXXXXXXXXXXXX	99.99	99.99	99.99	99.99	99.99
PROMEDIOS GENERALES	99.99	99.99	99.99	99.99	99.99

A partir de que se tiene el NOMBRE y las calificaciones 1, 2, 3 y 4 de varios alumnos, el promedio se obtiene sumando las cuatro calificaciones y dividiendo el resultado entre cuatro.

El promedio general de cada calificación se calcula sumando las calificaciones de todos los alumnos y dividiendo el resultado entre el número de alumnos.

(Primero hágalo usted; después compare la solución)

Algoritmo CALIFICACIONES DE ALUMNOS

Clase Alumnos1

1. Método principal()

a. Declarar variables

nombreAlum: Cadena

totAlumnos: Entero

calif1, calif2, calif3, calif4, promedio, promCal1,

promCal2, promCal3, promCal4, promProm, totCal1,

totCal2, totCal3, totCal4, totProm: Real

desea: Carácter

b. Imprimir encabezado

c. totAlumnos = 0

totCal1 = 0; totCal2 = 0; totCal3 = 0;

totCal4 = 0; totProm = 0;

d. do

1. Solicitar nombre, calificación 1, calificación 2, calificación 3, calificación 4

2. Leer nombreAlum, calif1, calif2, calif3, calif4

3. promedio = (calif1 + calif2 + calif3 + calif4)/4

4. Imprimir nombreAlum, calif1, calif2, calif3,

calif4, promedio

5. totAlumnos = totAlumnos + 1

6. totCal1 = totCal1 + calif1

totCal2 = totCal2 + calif2

totCal3 = totCal3 + calif3

totCal4 = totCal4 + calif4

totProm = totProm + promedio

7. Preguntar "¿Desea procesar otro alumno (S/N)?"

8. Leer desea

```

e. while desea == 'S'
f. promCal1 = totCal1 / totAlumnos
   promCal2 = totCal2 / totAlumnos
   promCal3 = totCal3 / totAlumnos
   promCal4 = totCal4 / totAlumnos
   promProm = totProm / totAlumnos
g. Imprimir promCal1, promCal2, promCal3, promCal4, promProm
h. Fin Método principal
Fin Clase Alumnos1
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Alumnos1.java



Explicación:

En el Método principal de la Clase Alumnos1 se tienen las acciones:

- a. Se declaran las variables.
 - b. Imprime encabezado.
 - c. Inicia en cero los acumuladores de calificaciones y el contador de alumnos.
 - d. Inicia el ciclo do:
 1. Se solicitan nombre y las calificaciones 1, 2, 3 y 4.
 2. Se leen en nombreAlum, calif1, calif2, calif3, calif4.
 3. Se calcula el promedio.
 4. Se imprimen nombreAlum, calif1, calif2, calif3, calif4 y promedio.
 5. Se incrementa el totAlumnos en 1.
 6. Se incrementa totCal1 con calif1.
totCal2 con calif2.
totCal3 con calif3.
totCal4 con calif4.
totProm con promedio.
 7. Se pregunta “¿Desea procesar otro alumno (S/N)?”.
 8. Se lee la respuesta en desea.
 - e. Fin del ciclo while desea == 'S'. Si se cumple, vuelve al do; si no, sale del ciclo.
 - f. Se calcula el promCal1.
Se calcula el promCal2.
Se calcula el promCal3.
Se calcula el promCal4.
Se calcula el promProm.
- Nota:** Observe que los promedios se calculan afuera del ciclo.
- g. Imprime promCal1, promCal2, promCal3, promCal4, promProm.
 - h. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Tabla 5.1: ejercicios resueltos disponibles en la zona de descarga del Capítulo 5 de la Web del libro.

Ejercicio	Descripción
Ejercicio 5.1.2.3	Procesa las calificaciones de varios alumnos
Ejercicio 5.1.2.4	Procesa artículos con inflación
Ejercicio 5.1.2.5	Procesa la producción de varios obreros con dos ciclos
Ejercicio 5.1.2.6	Procesa la producción de varios obreros ampliado
Ejercicio 5.1.2.7	Procesa las compras de varios clientes
Ejercicio 5.1.2.8	Procesa las ventas de varios vendedores y también se plantea para cuando se tienen 15 vendedores

Resumiendo

La repetición `do...while` es útil para resolver problemas del tipo:

1) Algo se repite varias veces, es decir, se debe ejecutar en más de una ocasión, pero no se sabe cuántas veces se repetirá. Se controla haciendo la pregunta y leyendo en una variable la respuesta (desea, otro), para plantear la condición de control del ciclo. Es el caso de todos los problemas planteados en este punto, con excepción del último.

2) Algo se repite un número conocido de veces, es decir, se sabe cuántas veces se repetirá. Se controla utilizando un contador que va contando la cantidad de veces que entra al ciclo, el cual se utiliza para plantear la condición de control del ciclo. Es el caso del último algoritmo de este punto. Este tipo de problemas son los que se plantean en el siguiente apartado ya que son naturales para la repetición `for`, no obstante que también pueden resolverse utilizando `do...while`.

5.1.3 Ejercicios propuestos para la repetición `do...while`

1. Elaborar un algoritmo que imprima el siguiente reporte:

ARTÍCULO	COSTOS DE PRODUCCIÓN			
	UNIDADES PRODUCIDAS	FACTOR COSTO	COSTO FIJO	COSTO DE PRODUCCIÓN
XXXXXXXXXXXXXXXXXX	9999	999.99	99,999.99	999,999.99
XXXXXXXXXXXXXXXXXX	9999	999.99	99,999.99	999,999.99
-	-	-	-	-
-	-	-	-	-
-	-	-	-	-
-	-	-	-	-
XXXXXXXXXXXXXXXXXX	9999	999.99	99,999.99	999,999.99
TOTAL 999 ARTÍCULOS				99,999,999.99

Datos disponibles por cada artículo:

- Descripción
- Cantidad de unidades producidas
- Factor de costo de materiales
- Costo fijo

Cálculo del costo de producción =

Número de unidades producidas X Factor de costo de materiales

+ Costo fijo

Al final imprimir el total de artículos procesados y el total general del costo de producción.

2. Elaborar un algoritmo que imprima el siguiente reporte:

ARTÍCULO	PRECIOS DE VENTA			PRECIO DE VENTA
	COSTO DE PRODUCCIÓN	UTILIDAD	IMPUESTO	
XXXXXXXXXX	99,999.99	99,999.99	99,999.99	999,999.99
XXXXXXXXXX	99,999.99	99,999.99	99,999.99	999,999.99
-	-	-	-	-
XXXXXXXXXX	99,999.99	99,999.99	99,999.99	999,999.99
XXXXXXXXXX	99,999.99	99,999.99	99,999.99	999,999.99
TOTAL 999	999,999.99	999,999.99	999,999.99	9,999,999.99

Datos disponibles por cada artículo:

- Descripción
- Costo de producción

Cálculos:

- Utilidad = 120 % del costo de producción
- Impuesto = 15 % (costo de producción + utilidad)
- Precio de venta = costo de producción + utilidad + impuesto

3. Elaborar un algoritmo para imprimir una factura que contenga los datos de los artículos vendidos a un cliente y que tenga el siguiente formato:

FACTURA			
NOMBRE DEL CLIENTE : XXXXXXXXXXXXXXXXXXXXXXXXXXXX			
ARTÍCULO	CANTIDAD	PRECIO UNITARIO	PRECIO TOTAL
XXXXXXXXXXXXXXXXXX	999	99,999.99	999,999.99
XXXXXXXXXXXXXXXXXX	999	99,999.99	999,999.99
-	-	-	-
XXXXXXXXXXXXXXXXXX	999	99,999.99	999,999.99
		SUBTOTAL	9,999,999.99
		IMPUESTO 15%	9,999,999.99
		TOTAL	9,999,999.99

Datos disponibles por cada artículo:

- Descripción
- Cantidad de artículos
- Precio unitario de venta
- Además se tiene el nombre del cliente

Cálculos:

- Precio total = cantidad de artículos X precio unitario
- Subtotal = la suma de los precios totales
- Impuesto = 15 % del subtotal
- Total = subtotal + impuesto

4. Igual al caso anterior, pero que se puedan procesar tantas facturas como clientes se hayan atendido.
5. Elaborar un algoritmo que imprima el siguiente reporte:

NÓMINA QUINCENAL			
NOMBRE	SDO. BRUTO	IMPUESTO	SDO. NETO
XXXXXXXXXXXXXXXXXX	99,999.99	99,999.99	99,999.99
XXXXXXXXXXXXXXXXXX	99,999.99	99,999.99	99,999.99
-	-	-	-
-	-	-	-
XXXXXXXXXXXXXXXXXX	99,999.99	99,999.99	99,999.99
TOTAL 999	999,999.99	999,999.99	999,999.99

Datos disponibles por cada empleado:

- Nombre
- Sueldo mensual
- Antigüedad

Cálculos:

- Sueldo bruto = (sueldo mensual / 2) + prima de antigüedad.

La prima de antigüedad se otorga a partir del tercer año de labores y es de 2% anual. El impuesto se calcula usando la tabla siguiente:

Lím. inferior	Lím. superior	Cuota fija	Porcentaje
1	300	30	3%
301	700	50	8%
701	1100	100	11%
1101	1700	150	16%
1701	adelante	220	20%

Cuando el sueldo bruto excede el salario mínimo, se calcula el excedente y se busca éste en la tabla a fin de determinar en cuál rango se encuentra.

El impuesto será la cuota fija más el porcentaje indicado de la diferencia del excedente sobre el límite inferior.

6. Elaborar un algoritmo que contabilice una cuenta de cheques. Al inicio se le introduce el nombre del cuentahabiente y el saldo inicial. A continuación se pueden hacer depósitos y retiros. Cuando sea depósito se incrementa al saldo y cuando sea retiro se resta. Este programa terminará cuando ya no se desee hacer movimientos. Se requiere la impresión del siguiente reporte:

ESTADO DE CUENTA

CUENTAHABIENTE: XXXXXXXXXXXXXXXXXXXX

SALDO INICIAL: 99,999,999.99

MOVIMIENTO	DEPÓSITO	RETIRO	SALDO
1	999,999.99		99,999,999.99
2		999,999.99	99,999,999.99
3	999,999.99		99,999,999.99
4		999,999.99	99,999,999.99
N			
TOTALES	9,999,999.99	9,999,999.99	999,999,999.99

7. Igual al Ejercicio 5.1.2.1, excepto en lo siguiente:

Si el pedido es por más de 50 hojas, se hace un 5% de descuento adicional.

Si el pedido es por más de 100 hojas, se hace un 10% de descuento adicional.

8. Elaborar un algoritmo que lea los datos de varios estudiantes (nombre y tres calificaciones parciales) e imprima el siguiente reporte:

REPORTE DE CALIFICACIONES

NOMBRE	CALIF. FINAL
XXXXXXXXXXXXXXXXXXXXXXXXXX	999.99
XXXXXXXXXXXXXXXXXXXXXXXXXX	999.99
	.
XXXXXXXXXXXXXXXXXXXXXXXXXX	999.99
TOTAL 999 ALUMNOS	

Para aprobar el curso, debe tener 70 o más en cada una de las tres calificaciones; la calificación final será el promedio. En caso de haber reprobado uno o más exámenes ordinarios, la calificación final será NA (No Acreditado).

9. Se tienen los datos del transporte del elevador de un edificio. Por cada viaje hecho durante el día, se tienen los siguientes datos:

Viaje:

Cantidad de personas: --

Peso del viaje: --

Viaje:

Cantidad de personas: --

Peso del viaje: --

Viaje:

Cantidad de personas: --

Peso del viaje: --

Elaborar un algoritmo que lea los datos de los viajes del día y que al final imprima:

```

ESTADÍSTICA DEL DÍA
CANTIDAD DE VIAJES:                999
CANTIDAD DE PERSONAS TRANSPORTADAS: 999
PESO TRANSPORTADO (KILOS):         9999.99
PROMEDIO DE PERSONAS POR VIAJE:    99.99
PROMEDIO DE PESO POR VIAJE:        9999.99

```

10. El equipo de fútbol CACHORROS requiere llevar un conteo de las acciones que realiza durante un partido. Las acciones están catalogadas como sigue:

1 tiro a gol	7 amonestación	13 pase equivocado
2 tiro desviado	8 gol anotado	0 fin de juego
3 falta recibida	9 gol recibido	
4 falta cometida	10 balón perdido	
5 fuera de juego	11 balón recuperado	
6 tiro de esquina	12 pase correcto	

Elaborar un algoritmo que permita capturar las acciones que se den en el partido, Por acción se introduce el número de acción y al terminar el partido, con la acción 0, se debe imprimir la siguiente estadística:

ESTADÍSTICA DEL JUEGO		
ACCIONES	CACHORROS	OTRO EQUIPO
tiros a gol:	---	---
tiros desviados:	---	---
faltas recibidas:	---	---
faltas cometidas:	---	---
fuera de juego:	---	---
tiros de esquina:	---	---
amonestaciones:	---	---
goles anotados:	---	---
goles recibidos:	---	---
balones perdidos:	---	---
balones recuperados:	---	---
pases correctos:	---	---
pases equivocados:	---	---

EQUIPO GANADOR: XXXXXXXXXXXXXXXXXXXX o bien, si no hubo ganador, imprimir
EL JUEGO FINALIZÓ EMPATADO

11. Se tienen los datos de varios autos importados. Elaborar un algoritmo que permita leer los datos de cada automóvil (marca, origen y costo) e imprima el siguiente reporte:

REPORTE DE AUTOMÓVILES IMPORTADOS				
MARCA	ORIGEN	COSTO	IMPUESTO	PRECIO VTA.
XXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXX	99,999.99	99,999.99	99,999.99
XXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXX	99,999.99	99,999.99	99,999.99
-	-	-	-	-
XXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXX	99,999.99	99,999.99	99,999.99
TOTAL 999 AUTOMÓVILES		99,999.99	99,999.99	99,999.99
ALEMANIA: --				
JAPÓN: --				
ITALIA: --				
EUA: --				
PAÍS DEL QUE SE IMPORTARON MÁS AUTOMÓVILES: X-----X				

Cálculos:

- IMPUESTO si el origen es Alemania el impuesto es 20%, si es de Japón el impuesto es 30%, si es de Italia el 15% y si es de USA el 8%.
- PRECIO VTA. se suma el costo más el impuesto.
- TOTALES se pide el total de autos importados, así como totales del costo, impuesto y precio de venta. Por último, el total de autos importados de cada país.

12. En el hotel Guamúchil se tienen los datos de los huéspedes. Por cada huésped se tiene: el nombre, el tipo de habitación (1, 2, 3, 4, 5) y la cantidad de días que la ocupó; se hace un descuento del 10% si el cliente se hospeda más de 5 días, del 15% si se hospeda más de 10 días y del 20% si se hospeda más de 15 días; de acuerdo al tipo de habitación se tienen las tarifas:

Tipo habitación	Tarifa diaria
1	120.00
2	155.00
3	210.00
4	285.00
5	400.00

Elaborar un algoritmo que lea los datos de los huéspedes e imprima el siguiente reporte:

NOMBRE DEL HUÉSPED	REPORTE DE HUÉSPEDES				
	DÍAS	TARIFA	SUBTOTAL	DESCTO.	TOTAL
XXXXXXXXXXXXXXXX	999	9,999.99	9,999.99	9,999.99	9,999.99
XXXXXXXXXXXXXXXX	999	9,999.99	9,999.99	9,999.99	9,999.99
-	-	-	-	-	-
XXXXXXXXXXXXXXXX	999	9,999.99	9,999.99	9,999.99	9,999.99
TOTAL 999 HUÉSPEDES			9,999.99	9,999.99	9,999.99
TOTAL DE DÍAS DE OCUPACIÓN:					
TIPO 1: --	TIPO 2: --	TIPO 3: --	TIPO 4: --	TIPO 5: --	

TARIFA es el precio de la habitación de acuerdo al tipo.
 SUBTOTAL se multiplica el número de días por la tarifa.
 DESCTO es el descuento que se le hace de acuerdo a la cantidad de días.
 TOTAL se calcula SUBTOTAL menos DESCTO.
 TOTAL DE DÍAS DE OCUPACIÓN es la sumatoria de las cantidades de días.
 TIPO1 es el total de días de ocupación de habitación tipo 1, y así para todos.

13. En la carrera Ingeniería Industrial y de Sistemas de la Universidad de Sonora se tienen varios alumnos, y por cada alumno los datos:

Nombre del alumno: X-----X

Cada alumno puede haber cursado varias materias. Por cada materia que cursó, se tienen los datos:

Materia: X-----X

Calificación 1: ---

Calificación 2: ---

Calificación 3: ---

Nombre del alumno: X-----X

Materia: X-----X

Calificación 1: ---

Calificación 2: ---

Calificación 3: ---

Elaborar un algoritmo que permita leer los datos de cada uno de los alumnos e imprima el siguiente reporte:

REPORTE DE MATERIAS CURSADAS			
NOMBRE	MATERIA	CALIF. FINAL	OBSERVACIÓN
XXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXX	999.99	APROBADO
	XXXXXXXXXXXXXXXXX	999.99	REPROBADO
	-	-	-
	XXXXXXXXXXXXXXXXX	999.99	APROBADO
TOTAL ALUMNO	999 MATERIAS	PROMEDIO: 999.99	
XXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXXXXX	999.99	APROBADO
	XXXXXXXXXXXXXXXXX	999.99	REPROBADO
	-	-	-
	XXXXXXXXXXXXXXXXX	999.99	APROBADO
TOTAL ALUMNO	999 MATERIAS	PROMEDIO: 999.99	

TOTAL ALUMNO	999 MATERIAS	PROMEDIO: 999.99	

TOTAL GENERAL	999 ALUMNOS	PROMEDIO GENERAL: 999.99	

Cálculos:

- CALIF. FINAL es el promedio de las tres calificaciones de la materia.
- OBSERVACIÓN es el comentario APROBADO o REPROBADO según sea la calificación final.
- TOTAL ALUMNO se imprime el número de materias que cursó y el promedio de las calificaciones finales de todas las materias que cursó.
- TOTAL GENERAL se imprime la cantidad de alumnos procesados y el promedio general de todos los alumnos.

14. Similar al Ejercicio 5.1.2.8, sólo que ahora se debe imprimir el siguiente reporte:

REPORTE DE INCENTIVOS		
NOMBRE	TOTAL VENDIDO	INCENTIVO
XXXXXXXXXXXXXXXXXXXXXXX	99,999.99	99,999.99
XXXXXXXXXXXXXXXXXXXXXXX	99,999.99	99,999.99
-	-	-
XXXXXXXXXXXXXXXXXXXXXXX	99,999.99	99,999.99
TOTAL 999 VENDEDORES	99,999.99	99,999.99
EL MEJOR VENDEDOR ES: XXXXXXXXXXXXXXXXXXXXXXX		
CON EL TOTAL VENDIDO: 99,999.99		

Es decir, que es el mismo reporte; sólo se añaden dos datos más al final: el nombre del mejor vendedor (el que haya vendido más) y cuánto fue el total vendido por éste. Vamos a suponer que no va a suceder que dos vendedores hayan vendido la misma cantidad.

15. En una cooperativa pesquera de Guaymas, Sonora, México, se tienen varios pescadores de camarón y cada pescador hace varios viajes de pesca. Se tienen los datos: nombre del pescador y la cantidad de kilos de camarón que entregó por cada uno de los viajes que realizó. Elaborar un algoritmo que lea esos datos y que imprima el siguiente reporte:

REPORTE DE PESCA DE CAMARÓN		
NOMBRE DEL PESCADOR	TOTAL PESCA (kg)	SUELDO
XXXXXXXXXXXXXXXXXXXXXXX	999	99,999.99
XXXXXXXXXXXXXXXXXXXXXXX	999	99,999.99
-	-	-
XXXXXXXXXXXXXXXXXXXXXXX	999	99,999.99
TOTAL 999 PESCADORES	9999	99,999.99
NOMBRE PESCADOR CON MAYOR PESCA: XXXXXXXXXXXXXXXXXXXXXXX		
PESCA QUE REALIZÓ: 999		
NOMBRE PESCADOR CON MENOR PESCA: XXXXXXXXXXXXXXXXXXXXXXX		
PESCA QUE REALIZÓ: 999		

Cálculos:

- Se lee el nombre de un pescador, luego cada una de las cantidades de kilogramos pescados por viaje de pesca, se suman estas cantidades para calcular el TOTAL PESCA (Kg); en otras palabras, es la sumatoria de las cantidades pescadas de todos los días que fue de pesca.

- El sueldo se calcula a 30.00 por kilo.
- Al final se pide el TOTAL de pescadores, el TOTAL del TOTAL PESCA (Kg) y el total de los sueldos de todos los pescadores. Además, el nombre del pescador que realizó la mayor pesca y cuánto fue ésta, así como el nombre del pescador que realizó la menor pesca y cuánto fue ésta. Se supone que el TOTAL PESCA (Kg) de los pescadores son diferentes.

16. En un equipo de baloncesto se tienen varios jugadores y cada jugador participó en varios juegos. Se tienen los datos: nombre del jugador y la cantidad de puntos que anotó por cada uno de los juegos en que participó. Elaborar un algoritmo que lea esos datos y que imprima el siguiente reporte:

REPORTE DE ANOTACIONES		
NOMBRE DEL JUGADOR	TOTAL PUNTOS	NIVEL DE ANOTACIÓN
XXXXXXXXXXXXXXXXXXXXXXXXXX	999	BAJO
XXXXXXXXXXXXXXXXXXXXXXXXXX	999	MEDIO
-	-	-
XXXXXXXXXXXXXXXXXXXXXXXXXX	999	ALTO
TOTAL 999 JUGADORES		9999
NOMBRE JUGADOR MAYOR TOTAL PUNTOS: XXXXXXXXXXXXXXXXXXXXXXXX		
TOTAL PUNTOS QUE ANOTÓ: 999		
NOMBRE JUGADOR MENOR TOTAL PUNTOS: XXXXXXXXXXXXXXXXXXXXXXXX		
TOTAL PUNTOS QUE ANOTÓ: 999		

Cálculos:

- Se lee el nombre de un jugador, luego cada una de las cantidades de puntos anotados por juego en que participó, se suman estas cantidades para calcular el TOTAL PUNTOS; en otras palabras, es la sumatoria de las cantidades anotadas de todos los juegos en que participó.
- El nivel de anotación se determina así: imprime BAJO si anotó 100 puntos o menos; imprime MEDIO si anotó más de 100 y hasta 200 puntos; e imprime ALTO si anotó más de 200 puntos.
- Al final se pide el TOTAL de jugadores y el TOTAL del TOTAL PUNTOS. Además, el nombre del jugador que anotó el mayor TOTAL PUNTOS y cuánto fue éste, así como el nombre del jugador que anotó el menor TOTAL PUNTOS y cuánto fue éste. Se supone que el TOTAL PUNTOS de los jugadores son diferentes.

Nota: Todos los ejercicios resueltos y propuestos en el siguiente punto se pueden solucionar con lo estudiado hasta este apartado.

5.2 La repetición for

La repetición for es una estructura que permite controlar la ejecución de acciones que se repetirán un número de veces conocido de antemano. Este tipo de repetición es controlada por un contador que empieza en un valor inicial y va hasta un valor final, incrementándose o decrementándose de acuerdo a un valor, para contar la cantidad de veces que entrará al ciclo. Se dice que el for se repite N veces.



Todos los ejercicios resueltos y propuestos en el siguiente punto se pueden solucionar con lo estudiado hasta este apartado.

Formato:

```
for contador=valorInicial; condición; incremento
    Acción(es)
endfor
```

En donde:

for	Es la palabra reservada que identifica la estructura de repetición.
contador	Es una variable que puede ser de tipo entero, real o carácter; la cual se utilizará como índice o contador que controlará la repetición del ciclo. El <code>contador</code> tomará el <code>valorInicial</code> , evalúa la condición y, si es verdadera, entra al ciclo <code>for</code> a ejecutar las acciones que están dentro del ciclo; si no es verdadera se sale del ciclo. Al llegar al <code>endfor</code> , éste lo regresa al <code>for</code> incrementando el <code>contador</code> de acuerdo con el <code>incremento</code> .
valorInicial	Es el valor inicial que tomará el contador. Puede ser una constante, variable o expresión de acuerdo al tipo de dato de la variable de control del ciclo. Ejemplo: <code>i=1</code> .
condición	Es una expresión lógica mediante la que se establece la condición de ejecución del ciclo, es decir, si se cumple entra al ciclo; si no se cumple se sale del ciclo. Ejemplo: <code>i<=10</code> .
Acción(es)	Es una acción o grupo de acciones en pseudocódigo que se ejecutarán dentro del ciclo.
incremento	Es una expresión aritmética mediante la cual se lleva a cabo el incremento del <code>contador</code> del ciclo. Ejemplos:

```
i=i+1   o   i++
m=m-1   o   m--
x=x+2
z=z+2.5
```

Nota: En los lenguajes C, C++, Java y derivados existe el operador aritmético `++`, que significa incremento de 1, y el operador `--`, que significa decremento de 1.

`endfor` Delimita el fin del ciclo.

Funcionamiento

1. Se inicia el contador con el `valorInicial` y se evalúa la condición; si se cumple, entra al ciclo y ejecuta la(s) acción(es).

```
for [contador=valorInicial; condición; incremento]
    ↓ Acción(es)
endfor
```



En los lenguajes C, C++, Java y derivados existe el operador aritmético `++`, que significa incremento de 1, y el operador `--`, que significa decremento de 1.

2. Al llegar al `endfor`, remite el control al inicio del ciclo, actualizando el valor del contador de acuerdo al incremento (o decremento).

```

for contador=valorInicial; condición; incremento
    Acción(es)
endfor

```

3. Al volver el control al inicio del ciclo, se evalúa la condición:

a) Si se cumple, entra al ciclo a ejecutar la(s) acción(es).

```

for contador=valorInicial; condición; incremento
    Acción(es)
endfor

```

Después de lo anterior, llega al `endfor`, el cual remite el control al inicio del `for`, actualizando el valor del contador de acuerdo con el incremento (o decremento).

b) Si no se cumple, se sale del ciclo, dirigiéndose a la siguiente acción después del `endfor`, es decir, se sale del ciclo.

```

for contador=valorInicial; condición; incremento
    Acción(es)
endfor

```

Ejemplo:

```

for i=1; i<=10; i++
    Imprimir i
endfor

```

Se trata de un ciclo repetitivo en el que la acción (`Imprimir i`) se ejecutará diez veces, ya que el contador `i` tomará el valor inicial de 1, luego de 2, y así sucesivamente hasta llegar a 10, con incrementos de uno, donde `i` es una variable de tipo entero que debe ser declarada antes de iniciar el `for`, en declaraciones de variables. Al iniciar `i` toma el valor de 1. Se evalúa la condición: ¿ `i<=10` ? Si se cumple, ejecuta lo que está dentro del ciclo (`Imprimir i`). La primera vez `i` tendrá valor de 1; por lo tanto, la condición se cumple y se imprime `i`.

Después de lo anterior llega al `endfor`, el cual remite el control hacia el encabezado del `for`; en este momento se aumenta el contador `i` en 1. Al llegar otra vez al `for`, evalúa de nuevo la condición; si se cumple, como es nuestro caso, entrará de nuevo al ciclo y así sucesivamente. En el ejemplo:

`i` toma el valor de 1, entra al ciclo e imprimirá: 1
`i` toma el valor de 2, entra al ciclo e imprimirá: 2
`i` toma el valor de 3, entra al ciclo e imprimirá: 3
`i` toma el valor de 4, entra al ciclo e imprimirá: 4

i toma el valor de 5, entra al ciclo e imprimirá: 5
i toma el valor de 6, entra al ciclo e imprimirá: 6
i toma el valor de 7, entra al ciclo e imprimirá: 7
i toma el valor de 8, entra al ciclo e imprimirá: 8
i toma el valor de 9, entra al ciclo e imprimirá: 9
i toma el valor de 10, entra al ciclo e imprimirá: 10

Nota: La condición $i \leq 10$ quiere decir que cuando el valor de *i* sea hasta 10, va a entrar al ciclo; cuando *i* tenga el valor 11, no se cumplirá la condición y saldrá del ciclo.

Si deseamos que entre al ciclo 30 veces, la condición será $i \leq 30$ y saldrá del ciclo cuando *i* tenga 31. Si deseamos que entre al ciclo 50 veces, la condición será $i \leq 50$ y saldrá del ciclo cuando *i* tenga 51.

El ciclo puede plantearse:

```
for i=10; i>=1; i--
    Imprimir i
endfor
```

En este caso *i* tomará primero el valor de 10, luego el 9, y así sucesivamente hasta llegar a 1. Debido a que se está planteando un ciclo con decremento, el valor inicial debe ser mayor que el valor de la condición ($i \geq 1$); en este caso entrará al ciclo hasta que *i* tome el valor de 1 y saldrá del ciclo cuando *i* tenga 0 (cero).

En este ejemplo se imprimirá: 10 9 8 7 6 5 4 3 2 1

Ejemplo:

Elaborar un algoritmo que calcule e imprima la suma de los números del 1 hasta el 100.

A continuación se tiene el algoritmo de la solución:

```
Algoritmo SUMA NUMEROS 1-100
Clase SumaNumeros1
1. Método principal()
    a. Declarar variables
        indice, sumatoria: Entero
    b. sumatoria = 0
    c. for indice=1; indice<=100; indice++
        1. sumatoria = sumatoria + indice
    d. endfor
    e. Imprimir sumatoria
    f. Fin Método principal
Fin Clase SumaNumeros1
Fin
```



La condición $i \leq 10$ quiere decir que cuando el valor de *i* sea hasta 10, va a entrar al ciclo; cuando *i* tenga el valor 11, no se cumplirá la condición y saldrá del ciclo.

Si deseamos que entre al ciclo 30 veces, la condición será $i \leq 30$ y saldrá del ciclo cuando *i* tenga 31. Si deseamos que entre al ciclo 50 veces, la condición será $i \leq 50$ y saldrá del ciclo cuando *i* tenga 51.

En la zona de descarga de la Web del libro está disponible:

Programa en Java: SumaNumeros1.java



Explicación:

En el Método principal de la Clase SumaNumeros1 se tienen las acciones:

- a. Se declaran las variables:
 - `indice` para manejar el contador del ciclo.
 - `sumatoria` para calcular la sumatoria de los números del 1 al 100.
- b. Se inicia el acumulador en cero.
- c. Ciclo for desde `indice = 1` hasta 100 con incrementos de 1.
 1. Se incrementa el acumulador `sumatoria` con `indice`.
- d. Fin del ciclo for.
- e. Se imprime la sumatoria.
- f. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

5.2.1 for anidados

Al igual que todas las estructuras de control, es posible que un ciclo for contenga anidado otro ciclo y éste a otro; veamos, por ejemplo, el siguiente ciclo:

```
for i=1; i<=10; i++
  Imprimir i
  for j=1; j<=10; j++
    Imprimir j
  endfor
endfor
```

Se trata de un ciclo controlado por `i`, dentro del cual se imprime el valor de `i`; además, contiene anidado un ciclo `for` controlado por `j`, donde se imprime el valor de `j`. Por cada una de las veces que entre en el primer ciclo `for` (el más externo), entrará diez veces al ciclo más interno; esto significa que por las diez veces que entrará en `i`, lo hará 100 veces en `j`.

5.2.2 Ejercicios resueltos para la repetición for**Ejercicio 5.2.2.1**

Elaborar un algoritmo que calcule e imprima la suma de los números pares del 2 hasta el 160.

(Primero hágalo usted; después compare la solución)

```
Algoritmo SUMA NUMEROS 2-160
Clase SumaNumeros2
1. Método principal()
  a. Declarar variables
    i, suma: Entero
  b. suma = 0
  c. for i=2; i<=160; i=i+2
    1. suma = suma + i
```

```

    d. endfor
    e. Imprimir suma
    f. Fin Método principal
Fin Clase SumaNumeros2
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: SumaNumeros2.java



Explicación:

En el Método principal de la Clase SumaNumeros2 se tienen las acciones:

- a. Se declaran las variables:
 - `i` para manejar el contador del ciclo.
 - `suma` para calcular la sumatoria de los números del 2 al 160.
- b. Se inicia el acumulador en cero.
- c. Ciclo for desde `i=2` hasta 160 con incrementos de 2.
 - Se incrementa el acumulador `suma` con `i`.
- d. Fin del ciclo for.
- e. Se imprime la suma.
- f. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 5.2.2.2

Elaborar un algoritmo que calcule e imprima la suma $1 + 1/2 + 1/3 + 1/4 \dots + 1/50$.

(Primero hágalo usted; después compare la solución)

```

Algoritmo SUMATORIA
Clase SumaNumeros3
1. Método principal()
  a. Declarar variables
    i: Entero
    valor, suma: Real
  b. suma = 0
  c. for i=1; i<=50; i++
    1. valor = 1/i
    2. suma = suma + valor
  d. endfor
  e. Imprimir suma
  f. Fin Método principal
Fin Clase SumaNumeros3
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: SumaNumeros3.java



Explicación:

En el Método principal de la Clase SumaNumeros3 se tienen las acciones:

- a. Se declaran las variables:
 - `i` para controlar el ciclo.
 - `valor` para calcular cada uno de los valores por sumar.
 - `suma` para calcular la sumatoria.
- b. Se inicia el acumulador `suma` en cero.
- c. Ciclo for desde `i = 1` hasta 50 con incrementos de 1.
 1. Se calcula el `valor = 1 / i`.
 2. Se incrementa `suma` con `valor`.
- d. Fin del for.
- e. Se imprime la suma.
- f. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 5.2.2.3

Elaborar un algoritmo que lea 20 números y que calcule e imprima el promedio de dichos números.

(Primero hágalo usted; después compare la solución)

```

Algoritmo PROMEDIO DE 20 NUMEROS
Clase PromedioNumeros1
1. Método principal()
  a. Declarar variables
    i, numero, sumatoria: Entero
    promedio: Real
  b. sumatoria = 0
  c. for i=1; i<=20; i++
    1. Solicitar número
    2. Leer numero
    3. sumatoria = sumatoria + numero
  d. endfor
  e. promedio = sumatoria / 20
  f. Imprimir promedio
  g. Fin Método principal
Fin Clase PromedioNumeros1
Fin
  
```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: PromedioNumeros1.java

Explicación:

En el Método principal de la Clase PromedioNumeros1 se tienen las acciones:

- a. Se declaran las variables.
- b. Se inicia `sumatoria` en cero.
- c. Ciclo for desde `i = 1` hasta 20 con incrementos de 1.

1. Se solicita el número.
 2. Se lee en numero.
 3. Se incrementa sumatoria con numero.
- d. Fin del for.
- e. Se calcula el promedio dividiendo sumatoria entre 20.
- f. Se imprime el promedio.
- g. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 5.2.2.4

Elaborar un algoritmo que solicite la cantidad de números que se van a procesar y lea la respuesta en N; luego que lea los N números y calcule e imprima el promedio de dichos números.

(Primero hágalo usted; después compare la solución)

```

Algoritmo PROMEDIO DE N NUMEROS
Clase PromedioNumeros2
1. Método principal()
  a. Declarar variables
    i, n, numero, sumatoria: Entero
    promedio: Real
  b. Solicitar cantidad de números a procesar
  c. Leer n
  d. sumatoria = 0
  e. for i=1; i<=n; i++
    1. Solicitar número
    2. Leer numero
    3. sumatoria = sumatoria + numero
  f. endfor
  g. promedio = sumatoria / n
  h. Imprimir promedio
  i. Fin Método principal
Fin Clase PromedioNumeros2
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: PromedioNumeros2.java



Explicación:

En el Método principal de la Clase PromedioNumeros2 se tienen las acciones:

- a. Se declaran las variables.
- b. Se solicita la cantidad de números.
- c. Se lee en n.
- d. Se inicia sumatoria en cero.
- e. Ciclo for desde i = 1 hasta n.
 1. Se solicita el número.
 2. Se lee en numero.
 3. Se incrementa sumatoria con numero.

- f. Fin del for.
- g. Se calcula el promedio dividiendo `sumatoria` entre `n`.
- h. Se imprime el promedio.
- i. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 5.2.2.5

Elaborar un algoritmo que imprima el seno, coseno y arco tangente de X para valores de X desde -1 hasta 1 con intervalos de 0.2 . Debe imprimir una tabla:

X	Seno X	Coseno X	Arco tangente X
-1.0	99.99	99.99	99.99
-0.8	99.99	99.99	99.99
-	-	-	-
1.0	99.99	99.99	99.99

(Primero hágalo usted; después compare la solución)

```

Algoritmo SENO COSENO ARCO TANGENTE DE -1 HASTA 1
Clase Logaritmos1
1. Método principal()
  a. Declarar variables
    x, senoX, cosenoX, arcoTanX: Real
  b. Imprimir encabezado
  c. for x=-1; x<=1; x=x+0.2
    1. senoX = Seno(X)
    2. cosenoX = Coseno(X)
    3. arcoTanX = ArcTan(X)
    4. Imprimir x, senoX, cosenoX, arcoTanX
  d. endfor
  e. Fin Método principal
Fin Clase Logaritmos1
Fin
  
```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Logaritmos1.java

Explicación:

En el Método principal de la Clase Logaritmos1 se tienen las acciones:

- a. Se declaran las variables.
- b. Se imprime el encabezado.
- c. Inicia ciclo for desde $x = -1$ hasta 1 con incrementos de 0.2 .
 - 1. Se calcula `senoX`.
 - 2. Se calcula `cosenoX`.
 - 3. Se calcula `arcoTanX`.
 - 4. Se imprime `x`, `senoX`, `cosenoX`, `arcoTanX`.
- d. Fin del ciclo for.
- e. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 5.2.2.6

Una temperatura en grados Fahrenheit (F) se convierte a grados Celsius (C) con la fórmula: $C = 5/9 * (F-32)$. Elaborar un algoritmo que imprima una tabla desde 1 hasta 65 (con intervalos de 1) grados Fahrenheit con sus equivalencias en grados Celsius:

Fahrenheit	Celsius
1	99.99
2	99.99
-	
-	
65	99.99

(Primero hágalo usted; después compare la solución)

```

Algoritmo EQUIVALENCIAS FAHRENHEIT CELSIUS
Clase EquivalenciasFahr1
1. Método principal()
  a. Declarar variables
    fahrenheit, celsius: Real
  b. Imprimir encabezado
  c. for fahrenheit=1; fahrenheit<=65; fahrenheit++
    1. celsius = 5/9 * (fahrenheit-32)
    2. Imprimir fahrenheit, celsius
  d. endfor
  e. Fin Método principal
Fin Clase EquivalenciasFahr1
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: EquivalenciasFahr1.java

**Explicación:**

En el Método principal de la Clase EquivalenciasFahr1 se tienen las acciones:

- Se declaran las variables.
- Imprime el encabezado.
- Inicia ciclo for desde `fahrenheit = 1` hasta 65 con incrementos de 1.
 - Calcula `celsius`.
 - Imprime `fahrenheit, celsius`.
- Fin del for.
- Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 5.2.2.7

La escuela FORD No. 8 de Guamúchil, Sinaloa, tiene actualmente 750 alumnos. Se espera tener un crecimiento anual del 12%. Elaborar un algoritmo que calcule e imprima la población estudiantil que se espera tener en el año 2035.

(Primero hágalo usted; después compare la solución)

```

Algoritmo ESTIMAR POBLACION
Clase PoblacionEstudiantil
1. Método principal()
  a. Declarar variables
    n: Entero
    pobFinal: Real
  b. pobFinal = 750
  c. for n=2010; n<=2035; n++
    1. pobFinal = pobFinal + (pobFinal * 0.12)
  d. endfor
  e. Imprimir pobFinal
  f. Fin Método principal
Fin Clase PoblacionEstudiantil
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: PoblacionEstudiantil.java

Explicación:

En el Método principal de la Clase PoblacionEstudiantil se tienen las acciones:

- a. Se declaran las variables.
- b. Se inicia pobFinal en 750.
- c. Ciclo for desde n = 2010 hasta 2035 con incrementos de 1.
 1. Se calcula pobFinal añadiéndole el 12%.
- d. Fin ciclo for.
- e. Se imprime pobFinal.
- f. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 5.2.2.8

Elaborar un algoritmo que lea un valor N, entero y positivo, y que le calcule e imprima su factorial. Por ejemplo, si se lee el 5, su factorial es el producto de $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$. El factorial de 0 es 1.

(Primero hágalo usted; después compare la solución)

```

Algoritmo FACTORIAL
Clase Factorial1
1. Método principal()
  a. Declarar variables
    num, i, fact: Entero
  b. Solicitar número
  c. Leer num
  d. if num == 0 then
    1. fact = 1

```

```

e. else
    1. fact = 1
    2. for i=num; i>=1; i--
        a. fact = fact * i
    3. endfor
f. endif
g. Imprimir fact
h. Fin Método principal
Fin Clase Factorial1
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Factorial1.java



Explicación:

En el Método principal de la Clase Factorial1 se tienen las acciones:

- a. Se declaran las variables.
- b. Se solicita el número.
- c. Se lee en num.
- d. Se compara si `num == 0`. Si se cumple, entonces:
 1. Se calcula `fact = 1`.
- e. Si no (else):
 1. Se inicia `fact = 1`.
 2. Ciclo for desde `i = num` hasta 1 con decrementos de -1.
 - a. Se calcula `fact = fact * i`.
 3. Fin del for.
- f. Fin del if.
- g. Se imprime el factorial `fact`.
- h. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Tabla 5.2: ejercicios resueltos disponibles en la zona de descarga del capítulo 5 de la Web del libro.

Ejercicio	Descripción
Ejercicio 5.2.2.9	Calcula el factorial a N números
Ejercicio 5.2.2.10	Calcula números aplicando la secuencia Fibonacci
Ejercicio 5.2.2.11	Calcula las potencias de los números del 1 al 8
Ejercicio 5.2.2.12	Realiza cálculos con la ecuación cuadrática
Ejercicio 5.2.2.13	Calcula intereses de una inversión fija
Ejercicio 5.2.2.14	Calcula intereses de una inversión leyendo datos
Ejercicio 5.2.2.15	Procesa la producción de 15 trabajadores con dos ciclos
Ejercicio 5.2.2.16	Procesa las ventas de varios vendedores
Ejercicio 5.2.2.17	Procesa la producción de 15 obreros
Ejercicio 5.2.2.18	Procesa la producción de 15 obreros (otra forma)
Ejercicio 5.2.2.19	Procesa la producción de varios obreros

5.2.3 Simulación del for con do...while

Como ya se explicó anteriormente, el for es controlado por un contador. También se ha descrito la forma de cómo manejar contadores con do...while. El ejemplo:

```
for i=1; i<=10; i++
    Imprimir i
endfor
```

lo podemos implementar con do...while de la siguiente forma:

```
i=0
do
    i=i+1
    Imprimir i
while i < 10
```



Es decir, con el do...while se pueden implementar soluciones a problemas que son naturales para el for.

Explicación:

- Antes de entrar al ciclo se inicia el contador *i* en cero.
- Dentro del ciclo se incrementa *i* en 1 y se imprime.
- La condición se plantea mientras el contador *i* sea menor que 10, vuelve al do.

Es decir, con el do...while se pueden implementar soluciones a problemas que son naturales para el for.

A continuación se plantearán algunos algoritmos que se resolvieron durante este capítulo con for, pero ahora utilizando do...while. Y, como podrá observar, la diferencia será la forma de manejar el contador, la cual se describió líneas antes.

Ejercicio 5.2.3.1

Elaborar un algoritmo similar al Ejercicio 5.2.2.5 utilizando do...while en lugar de for.

```
Algoritmo SENO COSENO ARCO TANGENTE DE -1 HASTA 1
Clase Logaritmos2
1. Método principal()
    a. Declarar variables
        x, senoX, cosenoX, arcoTanX: Real
    b. Imprimir encabezado
    c. x = -1.2
    d. do
        1. x = x + 0.2
        2. senoX = Seno(x)
        3. cosenoX = Coseno(x)
        4. arcoTanX = ArcTan(x)
        5. Imprimir x, senoX, cosenoX, arcoTanX
    e. while x < 1
    f. Fin Método principal
Fin Clase Logaritmos2
Fin
```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Logaritmos2.java



Tabla 5.3: ejercicios resueltos disponibles en la zona de descarga del capítulo 5 de la Web del libro.

Ejercicio	Descripción
Ejercicio 5.2.3.2	Calcula equivalencias Fahrenheit Celsius
Ejercicio 5.2.3.3	Calcula factoriales a N números
Ejercicio 5.2.3.4	Realiza cálculos con la ecuación cuadrática

5.2.4 Ejercicios propuestos para la repetición for

1. Elaborar un algoritmo que lea N números diferentes y calcule e imprima el mayor y el menor.
2. Elaborar un algoritmo que lea un valor N y que imprima un triángulo de asteriscos, como se muestra a continuación. Si el valor leído es 5 imprimir:

```
*
* *
* * *
* * * *
* * * * *
```

3. Elaborar un algoritmo que permita leer un número N par y calcule e imprima la suma de los números pares del 2 hasta el número leído. Si el número leído es menor a 2 debe imprimir un mensaje de error.
4. Elaborar un algoritmo que lea un valor N, luego que lea N números de entrada e imprima el total, el promedio, el mayor y el menor.
5. Elaborar un algoritmo que permita leer un número e imprima una tabla con las potencias de los números desde 1 hasta el número leído. La potencia de 1 es 1 elevado a la potencia 1. La potencia de 2 es 2 elevado a la potencia 2, y así sucesivamente, hasta la potencia del número leído.

Número	Potencia
1	9999
2	9999
-	
-	
8	9999

6. Elaborar un algoritmo para calcular e imprimir el cuadrado de los números impares del 1 al 15.
7. Elaborar un algoritmo que permita leer el valor inicial y el valor final en grados Fahrenheit e imprima una tabla con equivalencias en grados Celsius, desde el valor inicial hasta el valor final de 1 en 1.

Fahrenheit	Celsius
Valor inicial	99.99
-	-
-	-
-	-
Valor final	99.99

8. Elaborar un algoritmo similar al anterior, sólo que permita leer el valor inicial y el valor final en grados Celsius e imprima la tabla con equivalencias en grados Fahrenheit, desde el valor inicial hasta el valor final de 1 en 1.
9. Elaborar un algoritmo que imprima la secante, cosecante y tangente de X para valores de X desde -1 hasta 1 con intervalos de 0.1. Debe imprimir una tabla:

X	Secante X	Cosecante X	Tangente X
- 1.0	99.99	99.99	99.99
- 0.8	99.99	99.99	99.99
-			
-			
1.0	99.99	99.99	99.99

10. Elaborar un algoritmo que permita que los valores de los coeficientes a, b y c se comporten así: B debe ir de 1 hasta 7, C debe ir de 7 a 1, A debe tomar cada vez la diferencia de B - C. También, debe imprimir para cada juego de valores de a, b y c si tiene raíz única, raíces complejas o raíces reales, similar al Ejercicio 6.2.12.
11. Cada equipo de béisbol de la Liga Mexicana del Pacífico tiene un cuadro de 30 jugadores. Supóngase que cada equipo de la liga prepara un listado donde por cada jugador aparecen los datos siguientes: nombre del jugador, peso, edad. Elaborar un algoritmo para leer estos datos y que emita el siguiente reporte:

ESTADÍSTICA DE JUGADORES		
EQUIPO	PROMEDIO PESO	PROMEDIO EDAD
1	999.99	99.9
2	999.99	99.9
-		
10	999.99	99.9
PROMEDIO GENERAL PESO: 999.99		
PROMEDIO GENERAL EDAD: 99.9		

12. Elaborar un algoritmo para determinar e imprimir una tabla de amortización de un préstamo. Para ello se tienen como datos el saldo por amortizar, la tasa de interés anual y el número de meses que se tienen de plazo. Imprimir el reporte siguiente:

TABLA DE AMORTIZACIÓN

Saldo: 99,999,999.99

Interés anual: 999.99

Número de meses: 99

MES	SALDO INSOLUTO	CUOTA FIJA	INTERÉS	MENSUALIDAD
1	999,999.99	99,999.99	9,999.99	999,999.99
2	999,999.99	99,999.99	9,999.99	999,999.99
-	-	-	-	-
N	999,999.99	99,999.99	9,999.99	999,999.99
TOTALES	9,999,999.99	999,999.99	99,999.99	9,999,999.99

Datos disponibles:

- Saldo
- Interés anual
- Número de meses (plazo)

El proceso que se debe seguir para obtener la información es el siguiente:

- Como datos de entrada se tienen el saldo, interés anual y número de meses.
- El mes es un dato derivable, de 1 a n meses, según sea el plazo.
- El saldo insoluto es el capital que se debe en el mes correspondiente.
- La cuota fija se determina dividiendo el saldo entre el número de meses.
- El interés se determina mediante la aplicación del interés mensual sobre el saldo insoluto.
- La mensualidad se establece sumando la cuota fija más el interés.
- Los totales son el producto de la acumulación de la cuota fija, el interés y la mensualidad.

13. Elaborar un algoritmo para calcular la cantidad que se tendría ahorrada después de 10 años si se depositan mil pesos mensualmente a una tasa de interés mensual de 3%, capitalizable cada mes, es decir, que al capital se le agregan los intereses.
14. Elaborar un algoritmo para calcular la cantidad que se tendría ahorrada después de 15 años si se depositan quince mil pesos a una tasa de interés de 3.7% mensual, capitalizable cada mes.
15. Una empresa de teléfonos ha decidido incrementar la tarifa de la renta mensual por uso del teléfono en un 4% mensual. La tarifa en enero de 2010 es de 57.00. Elaborar un algoritmo que imprima el monto de la renta mensual en enero de 2011, 2012, 2013, ..., 2035.
16. Elaborar un algoritmo para estimar la población estudiantil de una escuela que se espera tener en un determinado año. Los datos que se tienen son: la población actual de la escuela (número de estudiantes), el porcentaje de crecimiento anual que se espera tener, el año actual y el año al que se desea estimar el crecimiento. Todos estos datos deben ser leídos para imprimir al final lo estimado.
17. Los números Fibonacci constituyen una secuencia que empieza con 0 y 1; el número que sigue a éstos se calcula sumando los dos anteriores y así sucesivamente. Elaborar un algoritmo que lea un número N e imprima los N primeros números de la secuencia. Si N no es mayor que cero, debe imprimir un mensaje de error.

18. Elaborar un algoritmo que permita leer una medida (N) en número de metros y que imprima una tabla de equivalencias a yardas, pulgadas y pies, desde 1 metro hasta N metros de uno en uno. Equivalencias: 1 pie = 12 pulgadas, 1 yarda = 3 pies, 1 pulgada = 2.54 cm, 1 metro = 100 cm. Se debe imprimir la tabla siguiente:

CONVERSIONES			
METROS	YARDAS	PULGADAS	PIES
1	9999.99	9999.99	9999.99
2	9999.99	9999.99	9999.99
-	-	-	-
N	9999.99	9999.99	9999.99

19. Similar al anterior, sólo que se lee un valor inicial y un valor final. Por ejemplo, si los valores leídos son 100 y 200, las conversiones se harán desde 100 hasta 200 de uno en uno.
20. Similar al anterior, sólo que el dato que se debe convertir es pies, es decir, es el que se leerá. También hacerlo para pulgadas y yardas.
21. Se tienen 12 escuelas y por cada escuela los datos: nombre de la escuela, la población actual (número de estudiantes), el porcentaje de crecimiento anual que se espera tener. Estamos en el año actual. Elaborar un algoritmo que permita leer los datos de cada una de las escuelas e imprima el siguiente reporte:

REPORTE DE ESCUELAS		
NOMBRE DE ESCUELA	POBL. EST. ACTUAL	POBL. EST. AÑO 2035
XXXXXXXXXXXXXXXXXXXXXX	9999.9	9999.9
XXXXXXXXXXXXXXXXXXXXXX	9999.9	9999.9
-	-	-
XXXXXXXXXXXXXXXXXXXXXX	9999.9	9999.9
TOTAL 999 ESCUELAS	9999.9	9999.9
LA ESCUELA MAYOR SERÁ: XXXXXXXXXXXXXXXXXXXXXXXX		
LA ESCUELA MENOR SERÁ: XXXXXXXXXXXXXXXXXXXXXXXX		

22. Elaborar un algoritmo similar al anterior, con la diferencia de que se tienen varias escuelas, es decir, no se sabe cuántas escuelas son; todo lo demás es igual al problema anterior.
23. Una compañía manufacturera fabrica un solo producto. Se tienen los siguientes datos de la producción de cada uno de los siete días de la semana: cantidad de unidades producidas, costo de operación por la producción del día y costo de los materiales utilizados.

Elaborar un algoritmo que lea dichos datos e imprima el siguiente reporte:

COSTOS DE PRODUCCIÓN				
DIA	UNIDADES PRODUCIDAS	COSTO PRODUCCIÓN	COSTO MATERIALES	COSTO UNITARIO
99	999	99,999.99	99,999.99	99,999.99
99	999	99,999.99	99,999.99	99,999.99
-	-	-	-	-
99	999	99,999.99	99,999.99	99,999.99
TOTAL	999	99,999.99	99,999.99	99,999.99

COSTO UNITARIO PROMEDIO: 99,999.99

- COSTO UNITARIO se calcula sumando el costo de producción más el costo de materiales y dividiéndolo entre las unidades producidas.
- TOTAL es la sumatoria de unidades producidas de todos los días, de los costos de producción y de los costos de materiales.
- COSTO UNITARIO PROMEDIO es la sumatoria de los costos unitarios diarios entre la cantidad de días.

24. Similar al caso anterior, excepto en lo siguiente: se tienen ocho plantas. Emitir el reporte:

COSTOS DE PRODUCCIÓN				
DIA	UNIDADES PRODUCIDAS	COSTO PRODUCCIÓN	COSTO MATERIALES	COSTO UNITARIO
99	999	99,999.99	99,999.99	99,999.99
99	999	99,999.99	99,999.99	99,999.99
-	-	-	-	-
99	999	99,999.99	99,999.99	99,999.99
TOTAL	999	99,999.99	99,999.99	99,999.99

COSTO UNITARIO PROMEDIO: 99,999.99

Ahora las unidades producidas, costo de producción y costo de materiales son la sumatoria de todos los días de producción de cada planta; el costo unitario promedio es por planta y al final el costo unitario promedio general.

25. Una compañía manufacturera de bombillas tiene 5 plantas: Hermosillo, Guamúchil, Tijuana, Culiacán y México. Se tienen los datos de la producción de los días de la semana, de la siguiente forma:

Planta 1: Hermosillo
 Día 1: Cantidad producida
 Cantidad de defectuosas
 Día 2: Cantidad producida
 Cantidad de defectuosas

 Día 7: Cantidad producida
 Cantidad de defectuosas

Planta 2: Guamúchil
 Día 1: Cantidad producida
 Cantidad de defectuosas
 Día 2: Cantidad producida
 Cantidad de defectuosas

 Día 7: Cantidad producida
 Cantidad de defectuosas

Elaborar un algoritmo que lea dichos datos e imprima el siguiente reporte:

REPORTE DE CONTROL DE CALIDAD			
PLANTA	UNIDADES PRODUCIDAS	UNIDADES DEFECTUOSAS	% DEFECTUOSAS
HERMOSILLO	999	999	999.99
GUAMÚCHIL	999	999	999.99
-	-	-	-
MÉXICO	999	999	999.99
TOTAL	999	999	
PORCENTAJE TOTAL DE DEFECTUOSAS: 999.99			

- % DEFECTUOSAS se calcula dividiendo unidades defectuosas entre unidades producidas.
- TOTAL es la sumatoria de unidades producidas y unidades defectuosas.
- PORCENTAJE TOTAL DE DEFECTUOSAS es la sumatoria del %DEFECTUOSAS de cada planta entre el número de plantas.

26. Se tienen 5 inversiones, y por cada una los datos: capital, tasa de interés anual y el plazo en número de meses. Elaborar un algoritmo que lea los datos de cada inversión e imprima el siguiente reporte:

INVERSIÓN 1			
CAPITAL: 99,999.99			
INTERÉS ANUAL: 999.99			
PLAZO EN MESES: 999			
MES	CAPITAL	INTERÉS	SALDO
1	99,999.99	99,999.99	99,999.99
2	99,999.99	99,999.99	99,999.99
3	-	-	-
-	-	-	-
N	99,999.99	99,999.99	99,999.99
TOTAL INTERÉS GANADO: -----			

INVERSIÓN 2

CAPITAL: -----
 INTERÉS ANUAL: -----
 PLAZO EN MESES: -----

MES	CAPITAL	INTERÉS	SALDO
1	99,999.99	99,999.99	99,999.99
2	99,999.99	99,999.99	99,999.99
3	-	-	-
-	-	-	-
-	-	-	-
N	99,999.99	99,999.99	99,999.99

TOTAL INTERÉS GANADO: 99,999.99

TOTAL GENERAL INVERTIDO: 999,999.99
 TOTAL GENERAL INTERÉS GANADO: 999,999.99

27. Similar al Ejercicio 5.2.2.15, pero con las diferencias siguientes:

- El número de días que laboró cada trabajador puede variar respecto de los demás trabajadores, es decir, un trabajador pudo haber trabajado 5 días, otro 4, otro 6, otro 4, etcétera.
- Imprimir al final del reporte:

TRABAJADOR MÁS PRODUCTIVO: XXXXXXXXXXXXXXXXXXXXXXXXXX

TOTAL DE UNIDADES FABRICADAS: 9999

TRABAJADOR MENOS PRODUCTIVO: XXXXXXXXXXXXXXXXXXXXXXXXXX

TOTAL DE UNIDADES FABRICADAS: 9999

Suponiendo que no habrá cantidades producidas iguales entre los trabajadores, el trabajador más productivo es el que tiene el mayor número de unidades producidas, mientras que el menos productivo es el que tiene el menor número de unidades producidas.

28. En una empresa se tienen 15 vendedores. Por cada vendedor se tiene su nombre y la cantidad vendida en pesos por cada uno de los varios días en que laboró. Elaborar un algoritmo que lea esos datos e imprima el siguiente reporte:

REPORTE DE SUELDOS		
NOMBRE	TOTAL VENTA	SUELDO
XXXXXXXXXXXXXXXXXXXXXXX	99,999.99	99,999.99
XXXXXXXXXXXXXXXXXXXXXXX	99,999.99	99,999.99
-	-	-
XXXXXXXXXXXXXXXXXXXXXXX	99,999.99	99,999.99
TOTAL 999 VENDEDORES	99,999.99	99,999.99
NOMBRE DEL MEJOR VENDEDOR:	XXXXXXXXXXXXXXXXXXXXXXX	
NOMBRE DEL PEOR VENDEDOR:	XXXXXXXXXXXXXXXXXXXXXXX	

Cálculos:

- TOTAL VENTA es la sumatoria de la venta de todos los días de trabajo.
- SUELDO es el 4% del TOTAL VENTA del vendedor +
 - 300.00 si el TOTAL VENTA está entre 15 001.00 y 25 000.00.
 - 600.00 si el TOTAL VENTA está entre 25 001.00 y 35 000.00.
 - 900.00 si el TOTAL VENTA es mayor a 35 000.00.
- NOMBRE DEL MEJOR VENDEDOR es el vendedor que tenga el TOTAL VENTA mayor.
- NOMBRE DEL PEOR VENDEDOR es el vendedor que tenga el TOTAL VENTA menor.

29. Similar al caso anterior, excepto en lo siguiente: se tienen varios vendedores y cada vendedor laboró 6 días.

30. En un equipo de baloncesto se tienen varios jugadores. Por cada jugador se tiene su nombre y la cantidad de puntos que anotó en cada uno de los 10 partidos en que jugó. Elaborar un algoritmo que lea esos datos e imprima el siguiente reporte:

ESTADÍSTICA DE JUGADORES

NOMBRE	TOTAL PUNTOS	NIVEL DE ANOTACIÓN
XXXXXXXXXXXXXXXXXXXXXX	999	DEFICIENTE
XXXXXXXXXXXXXXXXXXXXXX	999	BUENO
-	-	-
XXXXXXXXXXXXXXXXXXXXXX	999	EXCELENTE
TOTAL 99 JUGADORES	999	
NOMBRE DEL MEJOR ANOTADOR:	XXXXXXXXXXXXXXXXXXXXXX	
NOMBRE DEL PEOR ANOTADOR:	XXXXXXXXXXXXXXXXXXXXXX	

Cálculos:

- TOTAL PUNTOS es la sumatoria de los puntos que anotó en todos los juegos.
- NIVEL DE ANOTACIÓN es un comentario que indica:
 - DEFICIENTE si TOTAL PUNTOS es menor de 40.
 - BUENO si TOTAL PUNTOS está entre 40 y 90.
 - EXCELENTE si TOTAL PUNTOS es mayor de 90.
- TOTAL por un lado es el total de jugadores y, por el otro, es el total de puntos anotados por todos los jugadores.
- NOMBRE DEL MEJOR ANOTADOR es el jugador que tiene el TOTAL PUNTOS mayor.
- NOMBRE DEL PEOR ANOTADOR es el jugador que tiene el TOTAL PUNTOS menor.

31. Similar al caso anterior, excepto en lo siguiente: se tienen 12 jugadores y cada jugador participó en varios partidos.

32. Similar al ejercicio propuesto 15 del apartado anterior, excepto en lo siguiente: se tienen 150 pescadores y cada pescador realizó 20 viajes.

33. Similar al ejercicio propuesto 15 del apartado anterior, excepto en lo siguiente: se tienen 150 pescadores y cada pescador realizó varios viajes.

34. Similar al ejercicio propuesto 15 del apartado anterior, excepto en lo siguiente: se tienen varios pescadores y cada pescador realizó 20 viajes.
35. Similar al ejercicio propuesto 16 del apartado anterior, excepto en lo siguiente: se tienen 12 jugadores y cada jugador participó en 50 juegos.
36. Similar al ejercicio propuesto 16 del apartado anterior, excepto en lo siguiente: se tienen 12 jugadores y cada jugador participó en varios juegos.
37. Similar al ejercicio propuesto 16 del apartado anterior, excepto en lo siguiente: se tienen varios jugadores y cada jugador participó en 50 juegos.

Nota: Estos ejercicios también pueden resolverse utilizando las repeticiones do...while y while.

5.3 La repetición while

La repetición while es una estructura que permite controlar la ejecución de acciones que se repetirán en un rango de 0 (cero) a N veces; esto se debe a que la condición de control del ciclo se coloca al principio de la estructura y entra al ciclo mientras la condición sea verdadera. En caso de que no se cumpla la condición, se termina el ciclo.

Formato:

```
while condición
    Acción(es)
endwhile
```

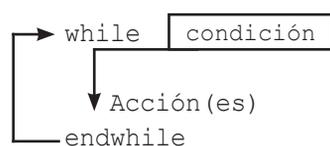
En donde:

while	Identifica la estructura y su inicio como un ciclo repetitivo.
condición	Es una expresión lógica que controla la ejecución del ciclo.
Acción(es)	Es la acción o acciones que se ejecutarán dentro del ciclo.
endwhile	Delimita el fin del ciclo repetitivo; envía el control al inicio de la estructura.

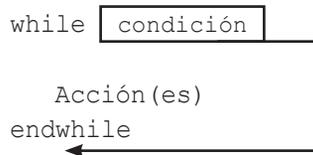
Funcionamiento:

Se evalúa la condición:

- a) Si se cumple, entra al ciclo, se ejecuta(n) la(s) acción(es), y al llegar al endwhile envía el control al while, lo cual implica volver a evaluar la condición.



b) Si no se cumple la condición, entonces se va a la siguiente acción después del `endwhile`, es decir, se sale del ciclo.



Al llegar al `while`, lo primero que se hace es evaluar la condición que controla el ciclo; si se cumple, entra al ciclo y se ejecutan las acciones especificadas dentro de éste. Al llegar al `endwhile` (fin del ciclo), el control se remite al inicio de la estructura, donde se evaluará de nuevo. En caso de no cumplirse la condición, el control se traslada a la primera acción después del `endwhile` (fin), lo cual implica salirse del ciclo.

Por lo general la condición se establece mediante el uso de una variable que se compara con cierto valor u otra variable. Debido a que la condición se encuentra al inicio del ciclo, debe hacerse una lectura adelantada o iniciar dicha variable para que tenga un valor la primera ocasión en que llega al `while`. Además, dentro del ciclo, debe actualizarse el valor de esa variable, por lo que es necesario incluir una lectura o asignarle un nuevo valor.

Nota: Cabe aclarar que esta estructura originalmente se inventó como `DOWHILE`: hacer mientras se cumpla la condición. Sin embargo, el inventor del lenguaje C la implementó como `while`, es decir, le eliminó la palabra `do`. Y en virtud de que el lenguaje C es la base de C++, Java y seguramente de los lenguajes que se diseñen en el futuro, en este libro la usaremos de esta forma.

Por ejemplo, si tenemos que procesar varios empleados y no sabemos cuántos son, la estructura general quedaría:

```

Preguntar "¿Desea procesar empleado (S/N)?"
Leer desea
while desea == 'S'
  Procesar empleado
  --
  --
  Preguntar "¿Desea procesar empleado (S/N)?"
  Leer desea
endwhile
  
```

Cabe aclarar que esta estructura originalmente se inventó como `DOWHILE`: hacer mientras se cumpla la condición. Sin embargo, el inventor del lenguaje C la implementó como `while`, es decir, le eliminó la palabra `do`. Y en virtud de que el lenguaje C es la base de C++, Java y seguramente de los lenguajes que se diseñen en el futuro, en este libro la usaremos de esta forma.

Lo primero que hace es evaluar si la condición se cumple, y por el hecho de que desde la primera vez pudiera no cumplirse es que el `while` permite plantear ciclos que se repiten en un intervalo de 0 (cero) a N veces.

En la estructura anterior se utiliza la variable `desea`, que es de tipo carácter. Para controlar el ciclo, antes del inicio del ciclo se hace la pregunta una sola vez y la lectura adelantada, a fin de que dicha variable tome un valor inicial. Después de lo anterior llega al `while`. Una vez procesado el empleado, para actualizar el valor de la variable, dentro del ciclo se incluyen la pregunta y lectura. Como puede observarse, estas dos acciones se colocan dos veces.

Nota: Lo primero que hace es evaluar si la condición se cumple, y por el hecho de que desde la primera vez pudiera no cumplirse es que el `while` permite plantear ciclos que se repiten en un intervalo de 0 (cero) a N veces.

5.3.1 Simulación del do...while con while

En virtud de que con la estructura while se plantean ciclos que van en un rango de 0 hasta N veces, con while es posible solucionar problemas de tipo do...while. A continuación se presenta un ejemplo que es natural para el do...while, pero resuelto con while.

Ejemplo:

Elaborar un algoritmo que permita procesar varios empleados, igual al primer ejemplo del capítulo 5 (do...while). Por cada empleado se leen los datos (nombre del empleado, número de horas trabajadas y cuota por hora) y se imprime el nombre y sueldo.

A continuación se tiene el algoritmo de la solución:

```

Algoritmo CALCULA SUELDOS DE EMPLEADOS
Clase Empleados3
  1. Método principal()
    a. Declarar variables
       nombreEmp: Cadena
       horasTrab: Entero
       cuotaHora, sueldo: Real
       desea: Carácter
    b. Preguntar "¿Desea procesar empleado (S/N)?"
    c. Leer desea
    d. while desea == 'S'
       1. Solicitar nombre, número de horas trabajadas,
          cuota por hora
       2. Leer nombreEmp, horasTrab, cuotaHora
       3. Calcular sueldo = horasTrab * cuotaHora
       4. Imprimir nombreEmp, sueldo
       5. Preguntar "¿Desea procesar empleado (S/N)?"
       6. Leer desea
    e. endwhile
    f. Fin Método principal
  Fin Clase Empleados3
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Empleados3.java



Nota: Este es un problema en el que se procesan varios empleados y no se sabe cuántos. Como usted podrá recordar, ya lo resolvimos con el do...while en el capítulo 5, es decir, es un problema natural para el do...while. Sin embargo, aquí lo estamos resolviendo con while para que usted vea que puede ser resuelto con ambas estructuras y observe la diferencia al utilizar while: antes del ciclo se pregunta "¿Desea procesar empleado (S/N)?" y luego se lee en desea la respuesta. Esta lectura adelantada se hace para que la primera vez que llegue al while, la variable contenga un valor 'S' o 'N'.



Este es un problema en el que se procesan varios empleados y no se sabe cuántos. Como usted podrá recordar, ya lo resolvimos con el do...while en el capítulo 5, es decir, es un problema natural para el do...while. Sin embargo, aquí lo estamos resolviendo con while para que usted vea que puede ser resuelto con ambas estructuras y observe la diferencia al utilizar while: antes del ciclo se pregunta "¿Desea procesar empleado (S/N)?" y luego se lee en desea la respuesta. Esta lectura adelantada se hace para que la primera vez que llegue al while, la variable contenga un valor 'S' o 'N'.

Explicación:

En el Método principal de la Clase Empleados3 se tienen las acciones:

- a. Se declaran las variables que ya conocemos: nombreEmp, horasTrab, cuotaHora y sueldo. Además, desea es una variable carácter que servirá para controlar al ciclo repetitivo.
- b. Pregunta “¿Desea procesar empleado (S/N)?”.
- c. Lee la respuesta en desea.
- d. Inicia ciclo while. Si desea== 'S' entra al ciclo:
 1. Se solicitan el nombre, número de horas trabajadas y cuota por hora.
 2. Se leen en nombreEmp, horasTrab, cuotaHora.
 3. Se calcula el sueldo.
 4. Imprime nombreEmp, sueldo.
 5. Se pregunta si “¿Desea procesar empleado (S/N)?”, pregunta a la cual se debe contestar S para SÍ o N para NO.
 6. Se lee en desea la respuesta que se dé a la pregunta anterior.
- e. endwhile delimita el fin del ciclo while y envía el control hacia el inicio del ciclo.
- f. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

5.3.2 Simulación del for con while

En virtud de que con la estructura while se plantean ciclos que van en un rango de 0 hasta N veces, con while es posible solucionar problemas de tipo for. A continuación se presenta un ejemplo que es natural para el for, pero resuelto con while y con do... while.

Ejemplo:

Elaborar un algoritmo que imprima los números del 1 al 10.

Este es un problema natural para el for porque se conoce cuántas veces se repetirá.

Solución usando for:

```

Algoritmo IMPRIME 1-10
Clase ImprimeNumeros1
  1. Método principal()
    a. Declarar variables
       i: Entero
    b. for i=1; i<=10; i++
       1. Imprimir i
    c. endfor
    d. Fin Método principal
  Fin Clase ImprimeNumeros1
Fin
  
```



En la zona de descarga de la Web del libro está disponible:

Programa en Java: ImprimeNumeros1.java

Explicación:

En el Método principal de la Clase `ImprimeNumeros1` se tienen las acciones:

- a. Se declara la variable `i`.
- b. Se plantea el ciclo for desde `i = 1` hasta 10 con incrementos de 1; cada vez que entra al ciclo imprime el valor de `i`.
- c. Fin del ciclo for:
Imprimirá: 1 2 3 4 5 6 7 8 9 10.
- d. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Solución usando while:

```

Algoritmo IMPRIME 1-10
Clase ImprimeNumeros2
  1. Método principal()
    a. Declarar variables
      i: Entero
    b. i = 0
    c. while i < 10
      1. i = i + 1
      2. Imprimir i
    d. endwhile
    e. Fin Método principal
Fin Clase ImprimeNumeros2
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: `ImprimeNumeros2.java`

**Explicación:**

En el Método principal de la Clase `ImprimeNumeros2` se tienen las acciones:

- a. Se declara la variable `i`.
- b. Se inicia el contador `i` en 0.
- c. Se plantea el ciclo while. Se pregunta si `i` es menor a 10 ($i < 10$). Si se cumple, entra al ciclo, donde:
Incrementa `i` en 1.
Imprime el valor de `i`.
- d. Fin del ciclo while, que lo envía al inicio del ciclo.
Imprimirá: 1 2 3 4 5 6 7 8 9 10.
- e. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.



En otras palabras, lo que estamos haciendo es manejar un contador, el cual se inicia en cero antes del ciclo y dentro del ciclo se incrementa en 1 y se imprime. En la condición de control del ciclo se pregunta si es menor a 10.

En otras palabras, lo que estamos haciendo es manejar un contador, el cual se inicia en cero antes del ciclo y dentro del ciclo se incrementa en 1 y se imprime. En la condición de control del ciclo se pregunta si es menor a 10.

Solución usando do...while:

```
Algoritmo IMPRIME 1-10
Clase ImprimeNumeros3
  1. Método principal()
    a. Declarar variables
       i: Entero
    b. i = 0
    c. do
       1. i = i + 1
       2. Imprimir i
    d. while i < 10
    e. Fin Método principal
  Fin Clase ImprimeNumeros3
Fin
```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: ImprimeNumeros3.java

Explicación:

En el Método principal de la Clase ImprimeNumeros3 se tienen las acciones:

- a. Se declara la variable *i*.
- b. Se inicia el contador *i* en 0.
- c. Inicia el ciclo do...while; entra al ciclo, donde:
 1. Incrementa *i* en 1.
 2. Imprime el valor de *i*.
- d. Cierra el ciclo con while, pregunta si $i < 10$.
Si es así se regresa al do a repetir el ciclo; si no, se sale del ciclo.
Imprimirá: 1 2 3 4 5 6 7 8 9 10.
- e. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

En otras palabras, lo que estamos haciendo es manejar un contador, el cual se inicia en cero antes del ciclo, dentro del ciclo se incrementa en 1 y se imprime. En la condición de control del ciclo se pregunta si es menor a 10.

Diferencia entre los tipos de repetición

Los tipos de repetición do...while, for y while se diferencian entre sí de acuerdo al rango de repeticiones que permiten:

El do...while permite un rango de repeticiones que va desde 1 hasta N veces, es decir, lo que está dentro del ciclo se deberá ejecutar al menos una vez y, mientras se cumpla la condición de ejecución del ciclo, cualquier cantidad de veces.

El for es útil para controlar ciclos en los que se conoce de antemano el número de veces que se deberán ejecutar las acciones que están dentro del ciclo. Esto es porque se controla con un contador, que toma desde un valor inicial hasta un valor final con un incremento.

El while permite un rango de repeticiones que va desde 0 (cero) hasta N veces, porque lo primero que se hace es evaluar la condición que controla el ciclo; si ésta se cumple entra al mismo, pero si no se cumple se va a la siguiente acción después del ciclo; esto permite que, al llegar la primera vez al ciclo, si no se cumple la condición, no entre ninguna vez al ciclo y, en caso de cumplirse, pueda entrar una y otra vez, es decir, hasta N veces.

Así, cuando se tiene un problema que contiene repeticiones, debemos analizar el tipo de repetición que es:

- Si se conoce exactamente cuántas veces se va a repetir, es tipo for.
- Si se sabe que algo se va a repetir, no se sabe cuántas veces, y se tiene la certeza de que sí va a haber al menos una ejecución, es tipo do...while.
- Si se sabe que algo se va a repetir, no se sabe cuántas veces y que puede repetirse desde 0 (cero) hasta N veces, es tipo while.

5.3.3 Ejercicios resueltos para la repetición while

Ejercicio 5.3.3.1

El sueldo que perciben los vendedores de una empresa automotriz está integrado de la manera siguiente: el salario mínimo, más \$ 100.00 por cada auto vendido, más el 2 % del valor de los autos vendidos.

Datos que se tienen por cada vendedor:

Nombre del vendedor: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

precio auto : 999,999.99

precio auto : 999,999.99

precio auto : 999,999.99

Nombre del vendedor : XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

precio auto : 999,999.99

precio auto : 999,999.99

precio auto : 999,999.99

.....

Nombre del vendedor : XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

precio auto : 999,999.99

precio auto : 999,999.99

Como se puede apreciar, se tienen varios vendedores; por cada vendedor, se tiene el nombre y el precio de cada auto que vendió en la quincena. Es posible que algunos vendedores no hayan realizado venta alguna; en tal caso, sólo se tendrá el nombre.

Elaborar un algoritmo que permita leer los datos e imprima el siguiente reporte:

NOMINA QUINCENAL	
NOMBRE	SUELDO
XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99,999.99
XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99,999.99
- - -	- - -
XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99,999.99
TOTALES 999	999,999.99

(Primero hágalo usted; después compare la solución)

```

Algoritmo VENDEDORES DE AUTOS
Clase VendedoresAutos
1. Método principal()
  a. Declarar variables
     nombreVend: Cadena
     desea, otro: Carácter
     totAutos, totVend: Entero
     precioAuto, salMin, sueldo,
     totSueldos, totVendido: Real
  b. Solicitar el salario mínimo
  c. Leer salMin
  d. Imprimir Encabezado
  e. totSueldos = 0
     totVend = 0
  f. do
     1. Solicitar nombre del vendedor
     2. Leer nombreVend
     3. totAutos = 0
        totVendido = 0
     4. Preguntar "¿Hay auto vendido (S/N)?"
     5. Leer otro
     6. while otro == 'S'
        a. Solicitar precio del auto
        b. Leer precioAuto
        c. totAutos = totAutos + 1
           totVendido = totVendido + precioAuto
        d. Preguntar "¿Hay otro auto vendido (S/N)?"
        e. Leer otro
     7. endwhile
     8. sueldo = salMin+(totAutos*100)+(totVendido*0.02)
     9. Imprimir nombreVend, sueldo
    10. totVend = totVend + 1
        totSueldos = totSueldos + sueldo
    11. Preguntar "¿Hay otro vendedor (S/N)?"
    12. Leer desea
  g. while desea == 'S'
  h. Imprimir totVend, totSueldos
  i. Fin Método principal
Fin Clase VendedoresAutos
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: VendedoresAutos.java

Explicación:

En el Método principal de la Clase VendedoresAutos se tienen las acciones:

- a. Se declaran las variables.
- b. Se solicita el salario mínimo.

- c. Se lee en `salMin`.
- d. Imprime encabezado.
- e. Inicia `totSueldos` en 0.
Inicia `totVend` en 0.
- f. Inicia ciclo `do` para procesar varios vendedores:
 1. Solicita el nombre del vendedor.
 2. Se lee en `nombreVend`.
 3. Inicia `totAutos` en 0.
Inicia `totVendido` en 0.
 4. Pregunta “¿Hay auto vendido (S/N)?”.
 5. Lee la respuesta en `otro`.
 6. Inicia ciclo `while` para procesar los autos vendidos por el vendedor.
Compara si `otro` es igual a ‘S’; si es así, entra al ciclo:
 - a. Solicita el precio del auto.
 - b. Se lee en `precioAuto`.
 - c. Incrementa `totAutos` en 1.
Incrementa `totVendido` con `precioAuto`.
 - d. Pregunta “¿Hay otro auto vendido (S/N)?”.
 - e. Lee la respuesta en `otro`.
 7. Fin del ciclo `while`.
 8. Calcula `sueldo=salMin+(totAutos*100)+(totVendido*0.02)`.
 9. Imprime `nombreVend`, `sueldo`.
 10. Incrementa `totVend` en 1.
Incrementa `totSueldos` con `sueldo`.
 11. Pregunta “¿Hay otro vendedor (S/N)?”.
 12. Lee la respuesta en `desea`.
- g. Fin ciclo (`do...while`) mientras `desea` sea igual a ‘S’ vuelve al `do`; si no, sale del ciclo.
- h. Imprime `totVend`, `totSueldos`.
- i. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Nota: El primer ciclo es del tipo `do...while` porque son varios vendedores. Sin embargo, también podría haberse resuelto con `while`.

El segundo ciclo, que procesa los autos vendidos, sólo puede resolverse con `while` porque la repetición que se presenta puede ir desde 0 (cero) hasta cualquier cantidad de autos vendidos.

Ejercicio 5.3.3.2

La Comisión Nacional del Agua, Delegación Sonora, lleva un registro de las lluvias que se presentan en todas las poblaciones del estado, de manera que se tienen los siguientes datos:

```
Población 1: X-----X
Lluvia: ----
Lluvia: ----
-----
Lluvia: ----
```



El primer ciclo es del tipo `do...while` porque son varios vendedores. Sin embargo, también podría haberse resuelto con `while`.

El segundo ciclo, que procesa los autos vendidos, sólo puede resolverse con `while` porque la repetición que se presenta puede ir desde 0 (cero) hasta cualquier cantidad de autos vendidos.

```

Población 2: X-----X
  Lluvia: ----
  Lluvia: ----
  ----
  Lluvia: ----
Población N: X-----X
  Lluvia: ----
  Lluvia: ----
  ----
  Lluvia: ----

```

Por cada población aparece el dato lluvia tantas veces como lluvias haya habido (este dato está en milímetros cúbicos). Podría darse el caso de que en alguna población no traiga ningún dato de lluvia: esto quiere decir que no llovió. Elaborar un algoritmo que lea estos datos e imprima el reporte:

POBLACIÓN	REPORTE DE LLUVIAS	TOTAL LLUVIA
XXXXXXXXXXXXXXXXXXXXX		999.99
XXXXXXXXXXXXXXXXXXXXX		999.99
---		---
XXXXXXXXXXXXXXXXXXXXX		999.99
TOTAL 999 POBLACIONES		999.99
TOTAL POBLACIONES DONDE NO LLOVIÓ: 999		

Cálculos:

- TOTAL LLUVIA es el total de lluvia que se presentó en una población. Se calcula sumando la cantidad de milímetros cúbicos de todas las lluvias.
- Se piden dos totales TOTAL de poblaciones procesadas y el total de lluvia en todas las poblaciones. También se pide la cantidad de poblaciones donde no se presentó lluvia alguna.

(Primero hágalo usted; después compare la solución)

```

Algoritmo LLUVIAS
Clase Lluvias
1. Método principal()
  a. Declarar variables
     poblacion: Cadena
     otro, hay: Carácter
     totPobla, totPobNoLluvia: Entero
     lluvia, totLluvia, toTotLluvia: Real
  b. Imprimir encabezado
  c. totPobla = 0
     totPobNoLluvia = 0
     toTotLluvia = 0
  d. Preguntar "¿Hay poblacion (S/N)?"
  e. Leer hay

```

```

f. while hay == 'S'
    1. Solicitar población
    2. Leer poblacion
    3. totLluvia = 0
    4. Preguntar "¿Hay lluvia (S/N)?"
    5. Leer otro
    6. while otro == 'S'
        a. Solicitar lluvia en milímetros cúbicos
        b. Leer lluvia
        c. totLluvia = totLluvia + lluvia
        d. Preguntar "¿Hay otra lluvia (S/N)?"
        e. Leer otro
    7. endwhile
    8. Imprimir poblacion, totLluvia
    9. totPobla = totPobla + 1
       toTotLluvia = toTotLluvia + totLluvia
    10. if totLluvia == 0 then
        a. totPobNoLluvia = totPobNoLluvia + 1
    11. endif
    12. Preguntar "¿Hay otra poblacion (S/N)?"
    13. Leer hay
g. endwhile
h. Imprimir totPobla, toTotLluvia, totPobNoLluvia
i. Fin Método principal
Fin Clase Lluvias
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Lluvias.java



Explicación:

En el Método principal de la Clase Lluvias se tienen las acciones:

- a. Se declaran las variables.
- b. Imprime encabezado.
- c. Inicia totPobla en 0, totPobNoLluvia en 0 y toTotLluvia en 0.
- d. Preguntar "¿Hay poblacion (S/N)?".
- e. Lee la respuesta en hay.
- f. Inicia ciclo while mientras hay es igual a 'S'; entonces entra al ciclo:
 1. Solicita población.
 2. Se lee en poblacion.
 3. Inicia totLluvia en 0.
 4. Preguntar "¿Hay lluvia (S/N)?".
 5. Lee la respuesta en otro.
 6. Inicia ciclo while mientras otro es igual a 'S'; entonces entra al ciclo:
 - a. Solicita lluvia en milímetros cúbicos.
 - b. Se lee en lluvia.
 - c. Incrementa totLluvia con lluvia.



El primer ciclo se ha resuelto con `while`. Sin embargo, también podría resolverse con `do...while`; inclusive, sería más natural.

El segundo ciclo, que procesa las lluvias, sólo puede resolverse con `while` porque la repetición que se presenta puede ir desde 0 (cero) hasta cualquier cantidad de lluvias.

- d. Pregunta “¿Hay otra lluvia (S/N)?”.
- e. Lee la respuesta en `otro`.
- 7. Fin del ciclo `while`.
- 8. Imprime `poblacion`, `totLluvia`.
- 9. Incrementa `totPobla` en 1.
Incrementa `toTotLluvia` con `totLluvia`.
- 10. Si `totLluvia == 0` entonces:
 - a. Incrementa `totPobNoLluvia` en 1.
- 11. Fin del `if`.
- 12. Pregunta “¿Hay otra poblacion (S/N)?”.
- 13. Lee la respuesta en `hay`.
- g. Fin del ciclo `while`.
- h. Imprime `totPobla`, `toTotLluvia`, `totPobNoLluvia`.
- i. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Nota: El primer ciclo se ha resuelto con `while`. Sin embargo, también podría resolverse con `do...while`; inclusive, sería más natural.

El segundo ciclo, que procesa las lluvias, sólo puede resolverse con `while` porque la repetición que se presenta puede ir desde 0 (cero) hasta cualquier cantidad de lluvias.

Tabla 5.4: ejercicios resueltos disponibles en la zona de descarga del capítulo 5 de la Web del libro.

Ejercicio	Descripción
Ejercicio 5.3.3.3	Procesa materiales requeridos para fabricar productos
Ejercicio 5.3.3.4	Procesa mantenimientos de máquinas
Ejercicio 5.3.3.5	Procesa la producción de 10 estaciones de trabajo
Ejercicio 5.3.3.6	Procesa la producción de varias estaciones de trabajo
Ejercicio 5.3.3.7	Calcula valores para XYX
Ejercicio 5.3.3.8	Obtiene pares entre 0 y 20
Ejercicio 5.3.3.9	Determina si un número es par o impar

5.3.4 Ejercicios propuestos para la repetición `while`

1. Similar al Ejercicio 5.3.3.1, sólo que ahora se tienen 20 vendedores y al final del reporte debe imprimirse el nombre del vendedor que tuvo el mayor total de venta y el del que tuvo el menor total de venta, suponiendo que no habrá cantidades iguales de total de venta.
2. Similar al Ejercicio 5.3.3.2, sólo que ahora se tienen 45 poblaciones y al final del reporte debe imprimirse el nombre de la población que tuvo el mayor total de lluvia y el de la que tuvo el menor total de lluvia (pero habiendo llovido), suponiendo que no habrá cantidades iguales de total de lluvia.
3. Elaborar un algoritmo similar al del Ejercicio 5.1.2.6, con la diferencia de que ahora cada obrero trabajó varios días y puede darse el caso de no haber laborado ningún día.

4. Elaborar un algoritmo similar al del Ejercicio 5.1.2.6, con la diferencia de que ahora se tienen 15 obreros, cada obrero trabajó varios días y puede darse el caso de no haber laborado ningún día.
5. Elaborar un algoritmo similar al ejercicio propuesto 30 de la repetición for, con la diferencia de que ahora, por cada jugador, se tiene su nombre y la cantidad de puntos que anotó en cada uno de los partidos en que lo hizo y puede darse el caso de que no haya anotado puntos en ningún juego.
6. Elaborar un algoritmo similar al ejercicio propuesto 30 de la repetición for, con la diferencia de que ahora se tienen 12 jugadores y por cada jugador se tiene su nombre y la cantidad de puntos que anotó en cada uno de los partidos en que lo hizo y puede darse el caso de que no haya anotado puntos en ningún juego.
7. Una compañía manufacturera fabrica el producto A. Para fabricar una unidad de dicho producto se requieren los siguientes materiales:
 Material 1: 3 unidades
 Material 2: 4 unidades
 Material 3: 1 unidades
 Material 4: 2 unidades
 Material 5: 3 unidades
 Material 6: 2 unidades

Se tiene como dato el costo de una unidad de cada uno de los seis materiales. Elaborar un algoritmo que lea los costos de los materiales, que luego lea pedidos del producto A (en cada pedido se tiene el dato cantidad de unidades del producto A) y que cuando termine de leer los pedidos imprima:

LISTADO DE MATERIALES REQUERIDOS

MATERIAL	TOTAL DE UNIDADES	COSTO ESTIMADO
1	999	99,999.99
2	999	99,999.99
3	999	99,999.99
4	---	---
5	---	---
6	999	99,999.99
COSTO TOTAL		99,999.99

8. Similar al ejercicio anterior, sólo que ahora se fabrican el producto A, el producto B y el producto C. Para fabricar el producto B se requieren los materiales:
 Material 1: 2 unidades
 Material 2: 5 unidades
 Material 3: 2 unidades
 Material 4: 1 unidades
 Material 5: 2 unidades
 Material 6: 4 unidades

y para fabricar el producto C se requieren los materiales:

Material 1: 7 unidades
 Material 2: 1 unidades
 Material 3: 5 unidades

Material 4: 4 unidades

Material 5: 2 unidades

Material 6: 3 unidades

9. Similar al Ejercicio 5.3.3.4, sólo que al final se indique la máquina que tuvo la mayor cantidad de mantenimientos y, además, se imprima cuál fue el tipo de mantenimiento que más se aplicó, suponiendo que no habrá iguales.
10. Similar al Ejercicio 5.3.3.5, sólo que al final imprima el porcentaje de estaciones de trabajo que estuvieron deficientes, el porcentaje de las que estuvieron bien y el porcentaje de las que estuvieron excelentes.
11. En la línea de producción de una compañía manufacturera se pueden presentar nueve tipos de fallas (1, 2, 3, 4, 5, 6, 7, 8, 9). Se tienen los datos de las fallas ocurridas en cada uno de los 6 días laborables de la semana: por cada día se tiene el tipo de falla, tantas veces como fallas se hayan presentado; en un día pudo no haber falla alguna. Elaborar un algoritmo que lea estos datos e imprima el siguiente reporte:

REPORTE SEMANAL DE FALLAS

TOTAL DE FALLAS OCURRIDAS: 999

TOTAL FALLAS TIPO 1: 999

TOTAL FALLAS TIPO 2: 999

TOTAL FALLAS TIPO 3: 999

TOTAL FALLAS TIPO 4: 999

TOTAL FALLAS TIPO 5: 999

TOTAL FALLAS TIPO 6: 999

TOTAL FALLAS TIPO 7: 999

TOTAL FALLAS TIPO 8: 999

TOTAL FALLAS TIPO 9: 999

TIPO DE FALLA QUE MÁS OCURRIÓ: 999

Nota: Como práctica, también se pueden realizar los ejercicios resueltos y los ejercicios propuestos para las repeticiones `do...while` y `for`.

5.4 Resumen de conceptos que debe dominar

- La estructura de repetición tipo `do...while`.
Emitir la información en forma de reporte.
Utilizar contadores y acumuladores, obtener promedios o medias aritméticas al procesar varios elementos y obtener el mayor y el menor de un conjunto de datos. Utilizar un ciclo repetitivo anidado dentro de otro.
- La estructura de repetición tipo `for`.
Utilizar ciclos anidados, es decir, un `for` dentro de otro `for`, un `do...while` dentro de un `for` y un `for` dentro de un `do...while`.
Cómo simular el `for` con `do...while`.
- La estructura de repetición tipo `while`.
Anidar ciclos repetitivos `while` dentro de otro `while`.
Cómo anidar `while` con `for` y `do...while`.
Plantear (simular) ciclos tipo `for` y `do...while` con `while`.

5.5 Contenido de la página Web de apoyo

El material marcado con asterisco () sólo está disponible para docentes.*

5.5.1 Resumen gráfico del capítulo

5.5.2 Autoevaluación

5.5.3 Programas en Java

5.5.4 Ejercicios resueltos

5.5.5 Power Point para el profesor (*)

6

Arreglos

Contenido

- 6.1 Arreglos unidimensionales
 - 6.1.1 Ejercicios resueltos para unidimensionales
- 6.2 Arreglos bidimensionales
 - 6.2.1 Ejercicios resueltos para bidimensionales
- 6.3 Arreglos tridimensionales
 - 6.3.1 Ejercicios resueltos para tridimensionales
- 6.4 Arreglos tetradimensionales
 - 6.4.1 Ejercicios resueltos para tetradimensionales
- 6.5 Ejercicios propuestos
- 6.6 Resumen de conceptos que debe dominar
- 6.7 Contenido de la página Web de apoyo
 - El material marcado con asterisco (*) sólo está disponible para docentes.*
 - 6.7.1 Resumen gráfico del capítulo
 - 6.7.2 Autoevaluación
 - 6.7.3 Programas en Java
 - 6.7.4 Ejercicios resueltos
 - 6.7.5 Power Point para el profesor (*)

Objetivos del capítulo

- Estudiar la estructura de datos denominada arreglos unidimensionales (de una dimensión), bidimensionales (de dos dimensiones), tridimensionales (tres dimensiones) y tetradimensionales (cuatro dimensiones).
- Aprender la definición y manipulación de los elementos de los diferentes tipos de arreglos.
- Aplicar lo aprendido en la solución de ejercicios resueltos y propuestos.

Competencias

- Competencia general del capítulo
 - Analizar problemas y diseñar algoritmos que los solucionen aplicando arreglos.
- Competencias específicas del capítulo
 - Diseña algoritmos aplicando arreglos unidimensionales.
 - Elabora algoritmos aplicando arreglos bidimensionales.
 - Desarrolla algoritmos aplicando arreglos tridimensionales.
 - Diseña algoritmos aplicando arreglos tetradimensionales.

Introducción

Con el estudio del capítulo anterior, usted ya domina la repetición y cómo diseñar algoritmos usando esa estructura de control, incluyendo el uso de todos los conceptos y estructuras de los capítulos anteriores.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos utilizando la estructura de datos denominada arreglos.

Se explica que el arreglo es un tipo de dato estructurado, formado por un conjunto de elementos de un mismo tipo de datos y que puede almacenar más de un valor a la vez, con la condición de que todos los elementos deben ser del mismo tipo de dato, es decir, que se puede tener un arreglo de datos enteros, reales, cadenas, etcétera.

Se expone que los arreglos se clasifican de acuerdo con el número de dimensiones que tienen. Así, se tienen los unidimensionales, los bidimensionales y los multidimensionales (de más de dos dimensiones); dentro de estos están los tridimensionales, tetradimensionales, etcétera. En este libro se tratan los unidimensionales, bidimensionales, tridimensionales y tetradimensionales; se abordan aspectos tales como la definición, manipulación de los elementos de los diferentes tipos de arreglos y su aplicación en pseudocódigo.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejercite estudiando los problemas planteados en los ejercicios resueltos y propuestos. Al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro, luego verifique sus resultados con los del libro, analice las diferencias y vea sus errores. Al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no está igual que el del libro, no necesariamente está mal. Usted debe ir aprendiendo a analizar las diferencias y a comprender que a veces, aunque haya diferencias, las dos soluciones están correctas.

En el siguiente capítulo se estudian los métodos.

El arreglo es un tipo de dato estructurado formado por un conjunto de elementos de un mismo tipo de datos.

En los capítulos anteriores hemos utilizado los tipos de datos Entero, Real, Cadena, Carácter y Boolean, los cuales son considerados como datos de tipo simple, puesto que una variable que se define con alguno de estos tipos sólo puede almacenar un valor a la vez, es decir, existe una relación de uno a uno entre la variable y el número de elementos (valores) que es capaz de almacenar.

En cambio, un dato de tipo estructurado, como el arreglo, puede almacenar más de un elemento (valor) a la vez, con la condición de que todos los elementos deben ser del mismo tipo de dato, es decir, que se puede tener un arreglo de datos enteros, reales, etcétera.

Los arreglos se clasifican de acuerdo con el número de dimensiones que tienen. Así, se tienen los unidimensionales (de una dimensión), los bidimensionales (dos dimensiones) y los multidimensionales (de más de dos dimensiones); dentro de éstos están los tridimensionales (tres dimensiones), tetradimensionales (cuatro dimen-

siones), etcétera. En este libro se tratarán los unidimensionales, bidimensionales, tridimensionales y tetradimensionales.

6.1 Arreglos unidimensionales

El arreglo unidimensional o de una dimensión está formado por un conjunto de elementos de un mismo tipo de datos que se almacenan bajo un mismo nombre y se diferencian por la posición que tiene cada elemento dentro del arreglo de datos.

Veamos el siguiente ejemplo:

Se tiene el número de unidades producidas por un obrero en cada uno de los 30 días del mes. Elaborar un algoritmo que permita leer la producción de cada uno de los 30 días, sin que se pierda la producción de ninguno de los días; esto es, se lee la producción del primer día, se lee la producción del segundo día, sin que se pierda la del primero, y así sucesivamente, de modo que al leer la producción del día 30 no se pierda la de ninguno de los 29 días anteriores.

a. Una opción sería usar 30 variables, una para cada día, de la siguiente manera: `produccion1, produccion2, produccion3, --, produccion30`

b. Otra opción es usar un arreglo con una dimensión de 30 elementos, como se muestra en la siguiente figura:

<code>produccion</code>	0	
	1	
	2	
	.	
	.	
	.	
	29	

Explicación:

En la figura tenemos un arreglo llamado `produccion` con 30 elementos, el primero de ellos se identifica con la posición 0, el segundo tiene la posición 1, el tercero la posición 2 y así sucesivamente hasta el elemento treinta, que tiene la posición número 29. Así, la producción del día 1 se almacena en el elemento número 0, la producción del día 2 se almacena en el elemento 1, y así sucesivamente hasta la producción del día 30, que se almacena en el elemento 29.

Nota: Hay metodologías de la programación y lenguajes en los que los elementos de un arreglo inician con el número 1 y van hasta N, donde N es el número de elementos del arreglo. Por ejemplo, un arreglo de 50 elementos tendrá desde el elemento 1 hasta el elemento 50 y un arreglo de 100 elementos tendrá desde el elemento 1 hasta el elemento 100.

Sin embargo, los lenguajes C, C++, Java y derivados tienen la peculiaridad de que el primer elemento de un arreglo es el número 0 (cero), el segundo es el número 1, el tercero el 2, y así sucesivamente hasta el elemento N-1, donde N es el número



Hay metodologías de la programación y lenguajes en los que los elementos de un arreglo inician con el número 1 y van hasta N, donde N es el número de elementos del arreglo. Por ejemplo, un arreglo de 50 elementos tendrá desde el elemento 1 hasta el elemento 50 y un arreglo de 100 elementos tendrá desde el elemento 1 hasta el elemento 100.

Sin embargo, los lenguajes C, C++, Java y derivados tienen la peculiaridad de que el primer elemento de un arreglo es el número 0 (cero), el segundo es el número 1, el tercero el 2, y así sucesivamente hasta el elemento N-1, donde N es el número de elementos del arreglo. Por ejemplo, un arreglo de 50 elementos tendrá desde el elemento 0 hasta el elemento 49 y un arreglo de 100 elementos tendrá desde el elemento 0 hasta el elemento 99. En la metodología que se está presentando en este libro se utilizará este concepto.

de elementos del arreglo. Por ejemplo, un arreglo de 50 elementos tendrá desde el elemento 0 hasta el elemento 49 y un arreglo de 100 elementos tendrá desde el elemento 0 hasta el elemento 99. En la metodología que se está presentando en este libro se utilizará este concepto.

Definición del arreglo unidimensional

Cuando se define un arreglo, es necesario hacerlo como una variable. En la parte de declaraciones de variables se utiliza el siguiente formato:

```
nombreVariable: Arreglo[Tamaño] Tipo de dato
```

En donde:

nombreVariable	Es el nombre de identificación de la variable.
Arreglo	Es la palabra reservada que indica que la variable es un arreglo.
Tamaño	Es un número entero que indica la cantidad de elementos que tendrá el arreglo; por ejemplo, 10, 20, 50, 100, 500, 1000, etcétera.
Tipo de dato	Es el tipo de dato que tendrá el conjunto de elementos del arreglo que se está definiendo. Puede ser Entero, Real, Cadena, etcétera.

Si aplicamos los conceptos anteriores para definir un arreglo que nos sirva para almacenar la producción de los 30 días del mes, tenemos:

```
Declarar variables
  produccion: Arreglo[30] Entero
```

Explicación:

- `produccion` es el nombre de la variable que se está declarando.
- Es un arreglo que contiene 30 elementos (del 0 al 29).
- Cada elemento del arreglo es un dato de tipo entero.

Manejo de los elementos del arreglo unidimensional

Cada elemento individual de un arreglo se relaciona con el nombre de la variable y un número que indica la posición que ocupa el elemento dentro del arreglo. Dicho número se pone entre `[]` y se le llama subíndice, índice o suscrito. De acuerdo con lo anterior, en nuestro ejemplo tenemos que:

El elemento 1 se relaciona con `produccion[0]`

El elemento 2 se relaciona con `produccion[1]`

· · · · ·
· · · · ·

El elemento 30 se relaciona con `produccion[29]`

El subíndice puede ser un valor constante de tipo entero como: 0, 1, 2, ..., 29.

También puede ser una variable de tipo entero, como:

```
produccion[i]
```

O bien, puede ser una expresión algebraica que dé un resultado de tipo entero, como:

```
produccion[i+3]
produccion[(i*4)-j]
```

Como toda variable, una de tipo arreglo puede usarse para leer datos, asignarle valores mediante expresiones aritméticas, imprimir su contenido, formar parte de expresiones lógicas, etcétera. Por ejemplo:

```
produccion[0] = 20
Leer produccion[i]
Leer produccion[10]
produccion[20] = produccion[0] + produccion[5]
Imprimir produccion[20]
```

Ejemplo:

Elaborar un algoritmo que lea la producción de un obrero en cada uno de los 30 días del mes y que lo imprima.

La lectura se podría hacer de la siguiente manera:

```
Solicitar producción del día
Leer produccion[0]
Solicitar producción del día
Leer produccion[1]
.
Solicitar producción del día
Leer produccion[29]
```

O bien, planteando un ciclo repetitivo, se hace la lectura más eficiente:

```
for i=0; i<=29; i++
    Solicitar produccion[i]
    Leer produccion[i]
endfor
```

Explicación:

Se plantea el ciclo desde que *i* tome el valor de 0 hasta 29. Por cada valor de *i* entra al ciclo, donde se solicita y lee la producción del día número *i*, es decir, la primera vez en el elemento 0 del arreglo *produccion*, la segunda en el elemento 1, y así hasta llegar a leer la producción del día 30 en el elemento 29.

Nota: En el ciclo anterior se podría solicitar así: `Solicitar producción[i+1]`. Esto es porque se tienen los días de producción 1, 2, 3, hasta 30, y los elementos del arreglo van de 0, 1, 2, hasta 29; se le suma 1 a *i* (*i*+1) para que el mensaje que aparece en la pantalla sea 1 en lugar de 0 la primera vez; la segunda será 2 en lugar de 1, y así hasta 30 en lugar de 29 la trigésima vez.



En el ciclo anterior se podría solicitar así: `Solicitar producción[i+1]`. Esto es porque se tienen los días de producción 1, 2, 3, hasta 30, y los elementos del arreglo van de 0, 1, 2, hasta 29; se le suma 1 a *i* (*i*+1) para que el mensaje que aparece en la pantalla sea 1 en lugar de 0 la primera vez; la segunda será 2 en lugar de 1, y así hasta 30 en lugar de 29 la trigésima vez.

El algoritmo completo quedaría de la siguiente manera:

```

Algoritmo PRODUCCION 30 DIAS
Clase Produccion
1. Método principal()
  a. Declarar variables
    produccion: Arreglo[30] Entero
    i: Entero
  b. for i=0; i<=29; i++
    1. Solicitar produccion[i]
    2. Leer produccion[i]
  c. endfor
  d. for i=0; i<=29; i++
    1. Imprimir produccion[i]
  e. endfor
  f. Fin Método principal
Fin Clase Produccion
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Produccion.java

Explicación:

En el Método principal de la Clase Produccion se tienen las acciones:

- a. Se declara la variable `produccion` como un arreglo de 30 elementos de datos de tipo entero para almacenar, en cada elemento, cada uno de los 30 días de producción.
- b. Inicia ciclo for que va desde 0 hasta 29 con incrementos de 1.
 1. Solicita la producción del día número `i`.
 2. Lee la producción en el elemento número `i`.
- c. Fin del ciclo for.
- d. Inicia ciclo for que va desde 0 hasta 29 con incrementos de 1.
 1. Imprime la producción del día número `i`, que está en el elemento número `i`.
- e. Fin del ciclo for.
- f. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

6.1.1 Ejercicios resueltos para unidimensionales

Ejercicio 6.1.1.1

Elaborar un algoritmo que lea el nombre de un vendedor y las ventas realizadas en cada uno de los 30 días del mes, que las almacene en un arreglo y que imprima el reporte siguiente:

Nombre del vendedor: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Venta del día 1: 999,999.99

Venta del día 2: 999,999.99

.

Venta del día 30: 999,999.99

Venta total del mes: 9,999,999.99

Donde la venta total del mes se calcula mediante la suma de las ventas realizadas en cada uno de los 30 días.

(Primero hágalo usted; después compare la solución)

```
Algoritmo VENTAS MES
Clase Venta
1. Método principal()
  a. Declarar variables
     nombreVend: Cadena
     ventas: Arreglo[30] Real
     i: Entero
     totVenta: Real
  b. Solicitar nombre del vendedor
  c. Leer nombreVend
  d. for i=0; i<=29; i++
     1. Solicitar ventas[i]
     2. Leer ventas[i]
  e. endfor
  f. totVenta = 0
  g. Imprimir nombreVend
  h. for i=0; i<=29; i++
     1. Imprimir ventas[i]
     2. totVenta = totVenta + ventas[i]
  i. endfor
  j. Imprimir totVenta
  k. Fin Método principal
Fin Clase Venta
Fin
```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Venta.java



Explicación:

En el Método principal de la Clase Venta se tienen las acciones:

- Se declaran las variables:
ventas como un arreglo de 30 elementos para almacenar las ventas de cada uno de los 30 días del mes; totVenta para calcular la sumatoria de las ventas de los 30 días.
- Se solicita el nombre del vendedor.
- Se lee en nombreVend.
- Inicia ciclo for desde i=0 hasta 29 con incrementos de 1.
 - Solicita la venta del día número i.
 - Se lee en ventas[i].
- Fin del for.
- Inicia totVenta en 0.
- Imprime nombreVend.

- h. Inicia ciclo for desde $i=0$ hasta 29 con incrementos de 1.
 - 1. Imprime `ventas[i]`.
 - 2. Incrementa `totVenta` con `ventas[i]`.
- i. Fin del for.
- j. Imprime `totVenta`.
- k. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 6.1.1.2

Elaborar un algoritmo que lea los elementos de dos arreglos, cada uno con 10 números enteros. Calcular los elementos de un tercer arreglo, sumando los elementos correspondientes de los dos primeros de la siguiente manera: que se sume el elemento 1 del primer arreglo y el 1 del segundo, que el resultado se almacene en el 1 del tercero, y así sucesivamente. Además, se requiere que al final imprima los tres arreglos de la siguiente forma:

Arreglo 1	+	Arreglo 2	=	Arreglo 3
99		99		999
99		99		999
-		-		-
99		99		999

(Primero hágalo usted; después compare la solución)

```

Algoritmo SUMA ARREGLOS
Clase SumaArreglos1
1. Método principal()
  a. Declarar variables
    a, b, s: Arreglo[10] Entero
    i: Entero
  b. for i=0; i<=9; i++
    1. Solicitar elemento i del arreglo a
    2. Leer a[i]
    3. Solicitar elemento i del arreglo b
    4. Leer b[i]
    5. Calcular s[i] = a[i] + b[i]
  c. endfor
  d. Imprimir encabezado
  e. for i=0; i<=9; i++
    1. Imprimir a[i], b[i], s[i]
  f. endfor
  g. Fin Método principal
Fin Clase SumaArreglos1
Fin
  
```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: SumaArreglos1.java

Explicación:

En el Método principal de la Clase SumaArreglos1 se tienen las acciones:

- a. Se declaran las variables:
a, b, s como arreglos de 10 elementos cada uno.
i: Entero.
- b. Inicia ciclo for desde i=0 hasta 9 con incrementos de 1.
 1. Solicita elemento i del arreglo a.
 2. Se lee en a[i].
 3. Solicita elemento i del arreglo b.
 4. Se lee en b[i].
 5. Calcula $s[i]=a[i]+b[i]$.
- c. Fin del for.
- d. Imprime encabezado.
- e. Inicia ciclo for desde i=0 hasta 9 con incrementos de 1.
 1. Imprime a[i], b[i], s[i].
- f. Fin del for.
- g. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 6.1.1.3

Elaborar un algoritmo que permita leer un vector de 10 números en un arreglo A de 10 elementos, lo mismo para un arreglo B, y que calcule e imprima el producto de A x B. Para obtener el producto de dos vectores se multiplica el elemento 1 del vector A por el elemento 1 del vector B, el 2 de A por el 2 de B, y así sucesivamente, obteniéndose la sumatoria de los productos; el resultado no es un vector, sino un valor simple.

(Primero hágalo usted; después compare la solución)

```

Algoritmo PRODUCTO DE VECTORES
Clase ProductoVectores
1. Método principal()
  a. Declarar variables
    vectorA, vectorB: Arreglo[10] Entero
    r, producto: Entero
  b. for r=0; r<=9; r++
    1. Solicitar vectorA[r]
    2. Leer vectorA[r]
  c. endfor
  d. for r=0; r<=9; r++
    1. Solicitar vectorB[r]
    2. Leer vectorB[r]
  e. endfor
  f. Imprimir encabezado
  g. producto = 0
  h. for r=0; r<=9; r++
    1. Imprimir vectorA[r], vectorB[r]
    2. producto = producto + (vectorA[r] * vectorB[r])

```

```

        i. endfor
        j. Imprimir producto
        k. Fin Método principal
    Fin Clase ProductoVectores
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: ProductoVectores.java

Explicación:

En el Método principal de la Clase ProductoVectores se tienen las acciones:

- a. Se declaran las variables.
- b. Inicia ciclo for desde $r=0$ hasta 9.
 1. Solicita `vectorA[r]`.
 2. Se lee en `vectorA[r]`.
- c. Fin del for.
- d. Inicia ciclo for desde $r=0$ hasta 9.
 1. Solicita elemento r del vector B.
 2. Se lee en `vectorB[r]`.
- e. Fin del for.
- f. Imprime encabezado.
- g. Inicia producto en 0.
- h. Inicia ciclo for desde $r=0$ hasta 9.
 1. Imprime `vectorA[r]`, `vectorB[r]`.
 2. Incrementa producto con $(\text{vectorA}[r] * \text{vectorB}[r])$.
- i. Fin del for.
- j. Imprime producto.
- k. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Tabla 6.1: ejercicios resueltos disponibles en la zona de descarga del capítulo 6 de la Web del libro.

Ejercicio	Descripción
Ejercicio 6.1.1.4	Calcula la desviación de la media de los elementos de un arreglo
Ejercicio 6.1.1.5	Realiza cálculos con la producción de un obrero
Ejercicio 6.1.1.6	Maneja una fila de números
Ejercicio 6.1.1.7	Maneja 20 números en un arreglo y obtiene el mayor
Ejercicio 6.1.1.8	Maneja nombres, pesos y estaturas de personas en dos arreglos
Ejercicio 6.1.1.9	Maneja nombres, edades y sueldos de personas en dos arreglos

6.2 Arreglos bidimensionales

El arreglo bidimensional o de dos dimensiones está formado por un conjunto de elementos de un mismo tipo de dato que se almacenan bajo un mismo nombre y que, al igual que en el unidimensional, se diferencian por la posición que tiene cada elemento dentro del arreglo de datos, con la aclaración de que la disposición de los elementos es en forma rectangular o cuadrada, donde la primera dimensión está dada por los renglones y la segunda por las columnas. Un arreglo de este tipo, también conocido como matriz, es de orden $M \times N$, donde M es el número de renglones y N el número de columnas, es decir, en forma de tabla.

Ejemplo:

Un arreglo de orden 4×5 tiene 4 renglones y 5 columnas, es decir, cada renglón se divide en 5 columnas, como se muestra a continuación:

Columna	0	1	2	3	4
Renglón 0					
Renglón 1					
Renglón 2					
Renglón 3					

Para esta matriz tenemos:

Los elementos:	0,0	0,1	0,2	0,3	0,4
	1,0	1,1	1,2	1,3	1,4
	2,0	2,1	2,2	2,3	2,4
	3,0	3,1	3,2	3,3	3,4

Es decir, el elemento renglón 0, columna 0; el elemento renglón 0, columna 1; y así hasta el elemento renglón 3, columna 4.

Definición del arreglo bidimensional

Al definir un arreglo es necesario hacerlo como una variable, por lo cual en la parte de declaraciones de variables se utiliza el siguiente formato:

```
nombreArreglo: Arreglo[tamRenglones][tamColumnas] Tipo de dato
```

En donde:

nombreArreglo	Es el nombre de identificación de la variable.
Arreglo	Es la palabra reservada que indica que la variable es un arreglo.
tamRenglones	Indica el número de renglones que tendrá el arreglo.
tamColumnas	Indica el número de columnas que tendrá el arreglo.
Tipo de dato	Es el tipo de dato que tiene el conjunto de elementos del arreglo que se está definiendo; pueden ser Entero, Real, Cadena, etcétera.

Si aplicamos los conceptos del formato anterior para definir la matriz de orden 4 x 5 de números enteros, tenemos:

```
Declarar variables
matriz: Arreglo[4][5] Entero
```

En donde:

- `matriz` es el nombre de la variable.
- Es un arreglo que contiene 4 renglones y 5 columnas (20 elementos).
- Cada elemento del arreglo es un dato de tipo entero.

Manejo de los elementos del arreglo bidimensional

Para relacionar cada elemento individual de una matriz se usan dos subíndices; el primero indica el renglón y el segundo la columna, como sigue:

```
matriz[renglon][columna]
```

En donde:

`renglon` indica el número de renglón y `columna` indica el número de columna que ocupa el elemento relacionado.

Los subíndices pueden ser constantes, variables o expresiones de tipo entero, como se explicó para el caso de los unidimensionales.

Como toda variable, una de tipo matriz puede usarse para leer datos, asignarle valores mediante expresiones aritméticas, imprimir su contenido, formar parte de expresiones lógicas, etcétera.

Ejemplos:

```
matriz[1][1] = 20
Leer matriz[r][c]
Leer matriz[3][4]
matriz[1][2] = matriz[1][2] + matriz[2][3]
Imprimir matriz[1][2]
```

Ejemplo:

Elaborar un algoritmo que lea números de tipo entero para una matriz de 4 renglones por 5 columnas y además que los imprima.

Para leer los elementos de una matriz es necesario utilizar dos ciclos repetitivos anidados: el primero, y más externo, para procesar los renglones, y el segundo, que estará anidado dentro del primero, para procesar las columnas.

Si definimos:

```
Declarar variables
numeros: Arreglo[4][5] Entero
```

la lectura se hace así:



Los subíndices pueden ser constantes, variables o expresiones de tipo entero, como se explicó para el caso de los unidimensionales.

```

for ren=0; ren<=3; ren++
  for col=0; col<=4; col++
    Solicitar numeros[ren][col]
    Leer numeros[ren][col]
  endfor
endfor

```

Explicación:

Se utiliza un ciclo tipo for desde 0 hasta 3 renglones y, dentro del proceso de cada renglón, un ciclo de 0 a 4 para procesar las columnas. De lo anterior se tiene que:

cuando `ren` tome el valor de 0, `col` tomará los valores desde 0 hasta 4, es decir, 5 veces; cuando `ren` tome el valor de 1, `col` tomará otra vez los valores desde 0 hasta 4, y así sucesivamente hasta que `ren` tome el valor de 3, también `col` tomará desde 0 hasta 4, terminando el proceso de lectura de los 20 elementos de la matriz de 4 x 5. Puede observarse que dentro del ciclo más interno se solicita el elemento `ren, col` y luego se lee. Para imprimir los elementos de la matriz se hace un proceso similar, sólo que en lugar de leer se imprime.

A continuación tenemos el algoritmo completo:

```

Algoritmo MATRIZ NUMEROS
Clase MatrizNumeros
  1. Método principal()
    a. Declarar variables
      numeros: Arreglo[4][5] Entero
      ren, col: Entero
    b. for ren=0; ren<=3; ren++
      1. for col=0; col<=4; col++
        a. Solicitar numeros[ren][col]
        b. Leer numeros[ren][col]
      2. endfor
    c. endfor
    d. for ren=0; ren<=3; ren++
      1. for col=0; col<=4; col++
        a. Imprimir numeros[ren][col]
      2. endfor
    e. endfor
    f. Fin Método principal
  Fin Clase MatrizNumeros
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: MatrizNumeros.java



Explicación:

En el Método principal de la Clase MatrizNumeros se tienen las acciones:

- a. Se declaran las variables:
 numeros como un arreglo con 4 renglones por 5 columnas de tipo Entero.
 ren, col para controlar los ciclos.
- b. Inicia ciclo for desde ren=0 hasta 3.
 1. Inicia ciclo for desde col=0 hasta 4.
 - a. Solicita elemento ren, col de numeros.
 - b. Se lee en numeros[ren][col].
 2. Fin del for.
- c. Fin del for.
- d. Inicia ciclo for desde ren=0 hasta 3.
 1. Inicia ciclo for desde col=0 hasta 4.
 - a. Imprime numeros[ren][col].
 2. Fin del for.
- e. Fin del for.
- f. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

6.2.1 Ejercicios resueltos para bidimensionales

Ejercicio 6.2.1.1

Elaborar un algoritmo que lea números enteros para una matriz de 5 x 7, que imprima los elementos de la matriz y que al final de cada renglón imprima la suma de todos sus elementos, y también que imprima total por cada columna.

(Primero hágalo usted; después compare la solución)

```

Algoritmo SUMA POR RENGLONES Y POR COLUMNAS
Clase SumaRenglonesColumnas
1. Método principal()
  a. Declarar variables
    matriz: Arreglo[5][7] Entero
    ren, col, sumaRen, sumaCol: Entero
  b. for ren=0; ren<=4; ren++
    1. for col=0; col<=6; col++
      a. Solicitar matriz[ren][col]
      b. Leer matriz[ren][col]
    2. endfor
  c. endfor
  d. for ren=0; ren<=4; ren++
    1. sumaRen = 0
    2. for col=0; col<=6; col++
      a. Imprimir matriz[ren][col]
      b. sumaRen = sumaRen + matriz[ren][col]
    3. endfor
    4. Imprimir sumaRen
  e. endfor

```

```

f. for c=0; c<=6; c++
    1. sumaCol = 0
    2. for r=0; r<=4; r++
        a. sumaCol = sumaCol + matriz[r][c]
    3. endfor
    4. Imprimir sumaCol
g. endfor
h. Fin Método principal
Fin Clase SumaReglonesColumnas
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: SumaReglonesColumnas.java



Explicación:

En el Método principal de la Clase SumaReglonesColumnas se tienen las acciones:

- a. Se declaran las variables:
 - matriz como arreglo de 5 renglones por 7 columnas.
 - sumaRen para calcular la sumatoria por renglón.
 - sumaCol para calcular la sumatoria por columna.
- b. Inicia ciclo for desde ren=0 hasta 4.
 1. Inicia ciclo for desde col=0 hasta 6.
 - a. Solicita matriz[ren][col].
 - b. Se lee en matriz[ren][col].
 2. Fin del for.
- c. Fin del for.
- d. Inicia ciclo for desde ren=0 hasta 4.
 1. Inicia sumaRen en 0.
 2. Inicia ciclo for desde col=0 hasta 6.
 - a. Imprime matriz[ren][col].
 - b. Incrementa sumaRen con matriz[ren][col].
 3. Fin del for.
 4. Imprime sumaRen.
- e. Fin del for.
- f. Inicia ciclo for desde c=0 hasta 6.
 1. Inicia sumaCol en 0.
 2. Inicia ciclo for desde r=0 hasta 4.
 - a. Incrementa sumaCol con matriz[r][c].
 3. Fin del for.
 4. Imprime sumaCol.
- g. Fin del for.
- h. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 6.2.1.2

Elaborar un algoritmo que lea números enteros para los elementos de dos matrices de 5 x 5 y que calcule cada elemento de una tercera matriz sumando los elementos correspondientes de las dos anteriores. Al final imprimir las tres matrices.

(Primero hágalo usted; después compare la solución)

```

Algoritmo SUMA MATRICES
Clase SumaMatrices
1. Método principal()
  a. Declarar variables
    matriz1, matriz2, matriz3: Arreglo[5][5] Entero
    ren, col: Entero
  b. for ren=0; ren<=4; ren++
    1. for col=0; col<=4; col++
      a. Solicitar matriz1[ren][col]
      b. Leer matriz1[ren][col]
    2. endfor
  c. endfor
  d. for ren=0; ren<=4; ren++
    1. for col=0; col<=4; col++
      a. Solicitar matriz2[ren][col]
      b. Leer matriz2[ren][col]
    2. endfor
  e. endfor
  f. for ren=0; ren<=4; ren++
    1. for col=0; col<=4; col++
      a. matriz3[ren][col]= matriz1[ren][col]+
        matriz2[ren][col]
    2. endfor
  g. endfor
  h. for ren=0; ren<=4; ren++
    1. for col=0; col<=4; col++
      a. Imprimir matriz1[ren][col]
    2. endfor
  i. endfor
  j. for ren=0; ren<=4; ren++
    1. for col=0; col<=4; col++
      a. Imprimir matriz2[ren][col]
    2. endfor
  k. endfor
  l. for ren=0; ren<=4; ren++
    1. for col=0; col<=4; col++
      a. Imprimir matriz3[ren][col]
    2. endfor
  m. endfor
  n. Fin Método principal
Fin Clase SumaMatrices
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: SumaMatrices.java

Explicación:

En el Método principal de la Clase SumaMatrices se tienen las acciones:

- a. Se declaran las variables:
matriz1, matriz2, matriz3 como arreglos de 5 por 5.
- b. Inicia ciclo for desde ren=0 hasta 4.
 1. Inicia ciclo for desde col=0 hasta 4.
 - a. Solicita matriz1[ren][col].
 - b. Se lee en matriz1[ren][col].
 2. Fin del for.
- c. Fin del for.
- d. Inicia ciclo for desde ren=0 hasta 4.
 1. Inicia ciclo for desde col=0 hasta 4.
 - a. Solicita matriz2[ren][col].
 - b. Se lee en matriz2[ren][col].
 2. Fin del for.
- e. Fin del for.
- f. Inicia ciclo for desde ren=0 hasta 4.
 1. Inicia ciclo for desde col=0 hasta 4.
 - a. Calcula $matriz3[ren][col]=matriz1[ren][col]+matriz2[ren][col]$.
 2. Fin del for.
- g. Fin del for.
- h. Inicia ciclo for desde ren=0 hasta 4.
 1. Inicia ciclo for desde col=0 hasta 4.
 - a. Imprime matriz1[ren][col].
 2. Fin del for.
- i. Fin del for.
- j. Inicia ciclo for desde ren=0 hasta 4.
 1. Inicia ciclo for desde col=0 hasta 4.
 - a. Imprime matriz2[ren][col].
 2. Fin del for.
- k. Fin del for.
- l. Inicia ciclo for desde ren=0 hasta 4.
 1. Inicia ciclo for desde col=0 hasta 4.
 - a. Imprime matriz3[ren][col].
 2. Fin del for.
- m. Fin del for.
- n. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 6.2.1.3

Se tienen los siguientes datos:

Nombre obrero 1: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Producción mes 1: 999

Producción mes 2: 999

Producción mes 6: 999

Nombre obrero 2: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Producción mes 1: 999

Producción mes 2: 999

Producción mes 6: 999

Nombre obrero 20: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Producción mes 1: 999

Producción mes 2: 999

Producción mes 6: 999

Elaborar un algoritmo que lea estos datos en dos arreglos: uno unidimensional para los nombres de los 20 obreros y otro bidimensional en el que se tendrán 20 renglones (uno para cada obrero) por 6 columnas (una para la producción de cada mes):

	nombres	produccion					
		0	1	2	3	4	5
0							
1							
2							
-							
-							
19							

Además, se requiere que imprima el reporte siguiente:

NOMBRE DEL OBRERO	REPORTE SEMESTRAL DE PRODUCCIÓN					MES6	TOT. PROD.
	MES1	MES2	MES3	MES4	MES5		
XXXXXXXXXXXXXXXXXXXX	999	999	999	999	999	999	999
XXXXXXXXXXXXXXXXXXXX	999	999	999	999	999	999	999
-							
XXXXXXXXXXXXXXXXXXXX	999	999	999	999	999	999	999
TOTAL							999

(Primero hágalo usted; después compare la solución)

```

Algoritmo PRODUCCION 20 OBREROS
Clase ObrerosMatrices
1. Método principal()
  a. Declarar variables
     nombres: Arreglo[20] Cadena
     produccion: Arreglo[20][6] Entero
     ren, col, totProd, totProdMes, toTotProd: Entero
  b. for ren=0; ren<=19; ren++
     1. Solicitar nombres[ren]
     2. Leer nombres[ren]
     3. for col=0; col<=5; col++
        a. Solicitar produccion[ren][col]
        b. Leer produccion[ren][col]
     4. endfor
  c. endfor
  d. Imprimir encabezado
  e. toTotProd = 0
  f. for ren=0; ren<=19; ren++
     1. Imprimir nombres[ren]
     2. totProd = 0
     3. for col=0; col<=5; col++
        a. Imprimir produccion[ren][col]
        b. totProd = totProd + produccion[ren][col]
     4. endfor
     5. Imprimir totProd
     6. toTotProd = toTotProd + totProd
  g. endfor
  h. for col=0; col<=5; col++
     1. totProdMes = 0
     2. for ren=0; ren<=19; ren++
        a. totProdMes = totProdMes +
           produccion[ren][col]
     3. endfor
  i. endfor
  j. Imprimir toTotProd
  k. Fin Método principal
Fin Clase ObrerosMatrices
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: ObrerosMatrices.java



Explicación:

En el Método principal de la Clase ObrerosMatrices se tienen las acciones:

- a. Se declaran las variables.
- b. Inicia ciclo for desde ren=0 hasta 19.
 1. Solicita nombre del obrero.
 2. Se lee en nombres[ren].
 3. Inicia ciclo for desde col=0 hasta 5.

- a. Solicita producción.
 - b. Se lee en `produccion[ren][col]`.
4. Fin del for.
- c. Fin del for.
- d. Imprimir encabezado.
- e. Inicia `toTotProd` en 0.
- f. Inicia ciclo for desde `ren=0` hasta 19.
 1. Imprime `nombres[ren]`.
 2. Inicia `totProd` en 0.
 3. Inicia ciclo for desde `col=0` hasta 5.
 - a. Incrementa `totProd` con `produccion[ren][col]`.
 4. Fin del for.
 5. Imprime `totProd`.
 6. Incrementa `toTotProd` con `totProd`.
- g. Fin del for.
- h. Inicia ciclo for desde `col=0` hasta 5.
 1. Inicia `totProdMes` en 0.
 2. Inicia ciclo for desde `ren=0` hasta 19.
 - a. Incrementa `totProdMes` con `produccion[ren][col]`.
 3. Fin del for.
 4. Imprime `totProdMes`.
- i. Fin del for.
- j. Imprime `toTotProd`.
- k. Fin del método principal. Luego se tiene el fin de la clase y el fin del algoritmo.

Tabla 6.2: ejercicios resueltos disponibles en la zona de descarga del capítulo 6 de la Web del libro.

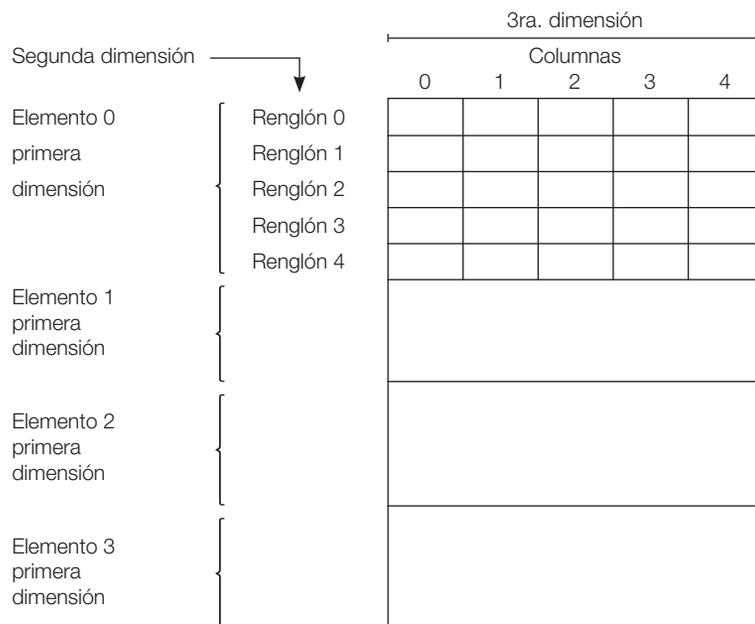
Ejercicio	Descripción
Ejercicio 6.2.1.4	Obtiene la transpuesta de una matriz de números
Ejercicio 6.2.1.5	Indica si dos matrices de números son iguales
Ejercicio 6.2.1.6	Multiplica una matriz por vector columna
Ejercicio 6.2.1.7	Suma por renglones y columnas una matriz de números
Ejercicio 6.2.1.8	Maneja la producción de 10 artículos en 3 turnos
Ejercicio 6.2.1.9	Maneja los estados y sus capitales
Ejercicio 6.2.1.10	Maneja una distribución de frecuencias en una matriz
Ejercicio 6.2.1.11	Maneja un cuadrado mágico en una matriz
Ejercicio 6.2.1.12	Multiplica dos matrices

6.3 Arreglos tridimensionales

El arreglo tridimensional o de tres dimensiones está formado por un conjunto de elementos de un mismo tipo de datos que se almacenan bajo un mismo nombre y

que, al igual que en los unidimensionales y bidimensionales, se diferencian por la posición que tiene cada elemento dentro del arreglo de datos, con la aclaración de que la disposición de los elementos es una combinación del arreglo unidimensional y bidimensional. La primera dimensión se podría esquematizar como el arreglo unidimensional, un conjunto de elementos; la segunda y tercera dimensión son un arreglo de dos dimensiones que constituye a cada elemento de la primera dimensión.

Un arreglo de tres dimensiones se podría leer como un arreglo de matrices, es decir, un arreglo compuesto por X elementos, donde cada elemento es un arreglo de $M \times N$ de dos dimensiones, esquemáticamente:



Un arreglo de tres dimensiones se podría leer como un arreglo de matrices, es decir, un arreglo compuesto por X elementos, donde cada elemento es un arreglo de $M \times N$ de dos dimensiones.

El arreglo tridimensional esquematizado tiene 4 elementos en la primera dimensión; cada elemento de ésta es un arreglo de 5×5 , es decir, un arreglo de 2 dimensiones. Se lee como un arreglo de matrices o un prisma rectangular.

Este arreglo está compuesto por los elementos:

Elemento	0,0,0	0,1,0	0,2,0	0,3,0	0,4,0
	0,0,1	0,1,1	0,2,1	0,3,1	0,4,1
	0,0,2	0,1,2	0,2,2	0,3,2	0,4,2
	0,0,3	0,1,3	0,2,3	0,3,3	0,4,3
	0,0,4	0,1,4	0,2,4	0,3,4	0,4,4

```

Elemento 1,0,0  1,1,0  1,2,0  1,3,0  1,4,0
          1,0,1  1,1,1  1,2,1  1,3,1  1,4,1
          1,0,2  1,1,2  1,2,2  1,3,2  1,4,2
          1,0,3  1,1,3  1,2,3  1,3,3  1,4,3
          1,0,4  1,1,4  1,2,4  1,3,4  1,4,4
Elemento 2,0,0  2,1,0  2,2,0  2,3,0  2,4,0
          2,0,1  2,1,1  2,2,1  2,3,1  2,4,1
          2,0,2  2,1,2  2,2,2  2,3,2  2,4,2
          2,0,3  2,1,3  2,2,3  2,3,3  2,4,3
          2,0,4  2,1,4  2,2,4  2,3,4  2,4,4
Elemento 3,0,0  3,1,0  3,2,0  3,3,0  3,4,0
          3,0,1  3,1,1  3,2,1  3,3,1  3,4,1
          3,0,2  3,1,2  3,2,2  3,3,2  3,4,2
          3,0,3  3,1,3  3,2,3  3,3,3  3,4,3
          3,0,4  3,1,4  3,2,4  3,3,4  3,4,4

```

Es un arreglo de 4 x 5 x 5, el cual contiene 100 elementos.

Definición del arreglo tridimensional

Como ya lo hemos mencionado, al definir un arreglo es necesario hacerlo como una variable, por lo cual, en la parte de declaración de variables, se utiliza el siguiente formato:

```
nomVar: Arreglo[primeraDim][segundaDim][terceraDim] Tipo de dato
```

En donde:

nomVar	Es el nombre de la variable.
Arreglo	Indica que es un arreglo.
primeraDim	Indica la cantidad de elementos de la primera dimensión.
segundaDim	Indica la cantidad de elementos de la segunda dimensión del arreglo.
terceraDim	Indica la cantidad de elementos de la tercera dimensión del arreglo.
Tipo de dato	Es el tipo de dato de los elementos del arreglo.

Ejemplo:

```
numeros: Arreglo[4][5][5] Entero
```

numeros	Es un arreglo de tres dimensiones: 4 elementos de 5 x 5, es decir, un arreglo de 4 elementos, cada uno de los cuales es una matriz de 5 x 5.
---------	--

Manejo de los elementos del arreglo tridimensional

Para relacionar cada elemento individual de un arreglo de tres dimensiones se usan tres subíndices; el primero indica la primera dimensión del elemento, el segundo la segunda dimensión y el tercero la tercera dimensión, como sigue:

```
nomVar[primera][segunda][tercera]
```

En donde:

primera	Indica el número de elemento en la primera dimensión.
segunda	Indica el número de elemento en la segunda dimensión.
tercera	Indica el número de elemento en la tercera dimensión.



Los subíndices pueden ser constantes, variables o expresiones de tipo entero.

Nota: Los subíndices pueden ser constantes, variables o expresiones de tipo entero.

Al igual que toda variable, una de tipo arreglo tridimensional puede usarse para leer datos, asignarle valores mediante expresiones aritméticas, imprimir su contenido, etcétera.

Ejemplos:

```
nomVar[2][3][4] = 50
Leer nomVar[2][2][3]
Leer nomVar[1][1][3]
nomVar[1][1][4] = nomVar[2][2][3] + nomVar[1][1][3] +
    nomVar[2][3][4]
Imprimir nomVar[1][1][4]
```

Ejemplo:

Elaborar un algoritmo que lea números de tipo entero para un arreglo como el esquematizado líneas antes, es decir, un arreglo de 4 x 5 x 5, y los imprima.

A continuación se presenta el algoritmo de la solución:

```
Algoritmo ARREGLO TRIDIMENSIONAL
Clase ArregloTresDim1
1. Método principal()
  a. Declarar variables
    numeros: Arreglo[4][5][5] Entero
    pri, seg, ter: Entero
  b. for pri=0; pri<=3; pri++
    1. for seg=0; seg<4; seg++
      a. for ter=0; ter<=4; ter++
        1. Solicitar elemento pri,seg,ter
        2. Leer numeros[pri][seg][ter]
      b. endfor
    2. endfor
```

```

c. endfor
d. for pri=0; pri<=3; pri++
    1. for seg=0; seg<4; seg++
        a. for ter=0; ter<=4; ter++
            1. Imprimir numeros[pri][seg][ter]
        b. endfor
    2. endfor
e. endfor
f. Fin Método principal
Fin Clase ArregloTresDim1
Fin

```



En la zona de descarga de la Web del libro está disponible:

Programa en Java: ArregloTresDim1.java

Explicación:

Se utilizan tres ciclos for: el primero, y más externo, para procesar los elementos de la primera dimensión; el segundo, que está anidado dentro del primero, para procesar los renglones de la segunda dimensión; y el tercer ciclo, que está anidado dentro del segundo, para procesar las columnas. En nuestro ejemplo necesitamos un ciclo for desde 0 hasta 3 para procesar cada uno de los cuatro elementos de la primera dimensión, dentro de éste un ciclo de 0 a 4 para procesar los renglones de la segunda dimensión y, dentro del proceso de cada renglón, un ciclo de 0 a 4 para procesar las columnas (la tercera dimensión).

6.3.1 Ejercicios resueltos para tridimensionales

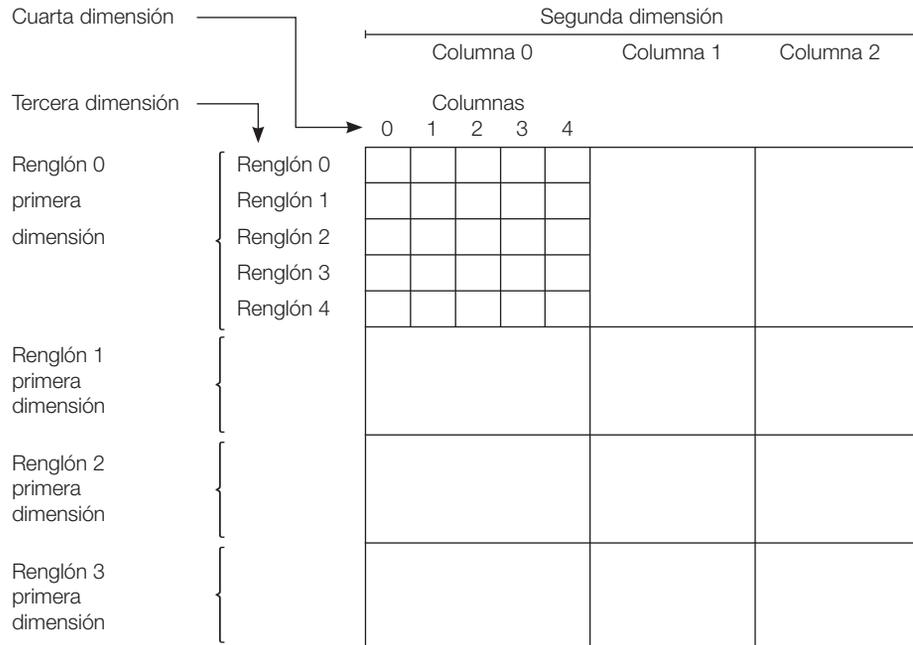
Tabla 6.3: ejercicios resueltos disponibles en la zona de descarga del capítulo 6 de la Web del libro.

Ejercicio	Descripción
Ejercicio 6.3.1.1	Maneja la producción de una empresa en un arreglo de tres dimensiones
Ejercicio 6.3.1.2	Similar al anterior, pero con más cálculos
Ejercicio 6.3.1.3	Similar al anterior, pero con más cálculos

6.4 Arreglos tetradimensionales

El arreglo tetradimensional o de cuatro dimensiones está formado por un conjunto de elementos de un mismo tipo de datos que se almacenan bajo un mismo nombre y que, al igual que en los unidimensionales, bidimensionales y tridimensionales, se diferencian por la posición que tiene cada elemento dentro del arreglo de datos, con la aclaración de que la disposición de los elementos es una doble combinación del arreglo bidimensional. Las primeras dos dimensiones se podrían esquematizar como un arreglo bidimensional y la tercera y cuarta dimensión esquematizan otro arreglo bidimensional dentro del primero.

Un arreglo de cuatro dimensiones se podría leer como una matriz de matrices, es decir, un arreglo de dos dimensiones $M \times N$ compuesto por elementos, donde cada elemento es un arreglo de dos dimensiones $P \times Q$. Esquemáticamente:



Un arreglo de cuatro dimensiones se podría leer como una matriz de matrices, es decir, un arreglo de dos dimensiones $M \times N$ compuesto por elementos, donde cada elemento es un arreglo de dos dimensiones $P \times Q$.

El arreglo tetradimensional esquematizado tiene 4 elementos en la primera dimensión y 3 elementos en la segunda dimensión para formar un arreglo de dos dimensiones, donde cada elemento de ésta es un arreglo de 5 x 5, es decir, un arreglo de 2 dimensiones (la tercera y cuarta), se lee como una matriz de matrices.

Este arreglo está compuesto por los elementos:

Elemento	0,0,0,0	0,0,1,0	0,0,2,0	0,0,3,0	0,0,4,0
	0,0,0,1	0,0,1,1	0,0,2,1	0,0,3,1	0,0,4,1
	0,0,0,2	0,0,1,2	0,0,2,2	0,0,3,2	0,0,4,2
	0,0,0,3	0,0,1,3	0,0,2,3	0,0,3,3	0,0,4,3
	0,0,0,4	0,0,1,4	0,0,2,4	0,0,3,4	0,0,4,4
Elemento	0,1,0,0	0,1,1,0	0,1,2,0	0,1,3,0	0,1,4,0
	0,1,0,1	0,1,1,1	0,1,2,1	0,1,3,1	0,1,4,1
	0,1,0,2	0,1,1,2	0,1,2,2	0,1,3,2	0,1,4,2
	0,1,0,3	0,1,1,3	0,1,2,3	0,1,3,3	0,1,4,3
	0,1,0,4	0,1,1,4	0,1,2,4	0,1,3,4	0,1,4,4

Elemento	0,2,0,0	0,2,1,0	0,2,2,0	0,2,3,0	0,2,4,0
	0,2,0,1	0,2,1,1	0,2,2,1	0,2,3,1	0,2,4,1
	0,2,0,2	0,2,1,2	0,2,2,2	0,2,3,2	0,2,4,2
	0,2,0,3	0,2,1,3	0,2,2,3	0,2,3,3	0,2,4,3
	0,2,0,4	0,2,1,4	0,2,2,4	0,2,3,4	0,2,4,4
Elemento	1,0,0,0	1,0,1,0	1,0,2,0	1,0,3,0	1,0,4,0
	1,0,0,1	1,0,1,1	1,0,2,1	1,0,3,1	1,0,4,1
	1,0,0,2	1,0,1,2	1,0,2,2	1,0,3,2	1,0,4,2
	1,0,0,3	1,0,1,3	1,0,2,3	1,0,3,3	1,0,4,3
	1,0,0,4	1,0,1,4	1,0,2,4	1,0,3,4	1,0,4,4
Elemento	1,1,0,0	1,1,1,0	1,1,2,0	1,1,3,0	1,1,4,0
	1,1,0,1	1,1,1,1	1,1,2,1	1,1,3,1	1,1,4,1
	1,1,0,2	1,1,1,2	1,1,2,2	1,1,3,2	1,1,4,2
	1,1,0,3	1,1,1,3	1,1,2,3	1,1,3,3	1,1,4,3
	1,1,0,4	1,1,1,4	0,1,2,4	1,1,3,4	1,1,4,4
Elemento	1,2,0,0	1,2,1,0	1,2,2,0	1,2,3,0	1,2,4,0
	1,2,0,1	1,2,1,1	1,2,2,1	1,2,3,1	1,2,4,1
	1,2,0,2	1,2,1,2	1,2,2,2	1,2,3,2	1,2,4,2
	1,2,0,3	1,2,1,3	1,2,2,3	1,2,3,3	1,2,4,3
	1,2,0,4	1,2,1,4	1,2,2,4	1,2,3,4	1,2,4,4
Elemento	2,0,0,0	2,0,1,0	2,0,2,0	2,0,3,0	2,0,4,0
	2,0,0,1	2,0,1,1	2,0,2,1	2,0,3,1	2,0,4,1
	2,0,0,2	2,0,1,2	2,0,2,2	2,0,3,2	2,0,4,2
	2,0,0,3	2,0,1,3	2,0,2,3	2,0,3,3	2,0,4,3
	2,0,0,4	2,0,1,4	2,0,2,4	2,0,3,4	2,0,4,4
Elemento	2,1,0,0	2,1,1,0	2,1,2,0	2,1,3,0	2,1,4,0
	2,1,0,1	2,1,1,1	2,1,2,1	2,1,3,1	2,1,4,1
	2,1,0,2	2,1,1,2	2,1,2,2	2,1,3,2	2,1,4,2
	2,1,0,3	2,1,1,3	2,1,2,3	2,1,3,3	2,1,4,3
	2,1,0,4	2,1,1,4	2,1,2,4	2,1,3,4	2,1,4,4
Elemento	2,2,0,0	2,2,1,0	2,2,2,0	2,2,3,0	2,2,4,0
	2,2,0,1	2,2,1,1	2,2,2,1	2,2,3,1	2,2,4,1
	2,2,0,2	2,2,1,2	2,2,2,2	2,2,3,2	2,2,4,2
	2,2,0,3	2,2,1,3	2,2,2,3	2,2,3,3	2,2,4,3
	2,2,0,4	2,2,1,4	2,2,2,4	2,2,3,4	2,2,4,4
Elemento	3,0,0,0	3,0,1,0	3,0,2,0	3,0,3,0	3,0,4,0
	3,0,0,1	3,0,1,1	3,0,2,1	3,0,3,1	3,0,4,1
	3,0,0,2	3,0,1,2	3,0,2,2	3,0,3,2	3,0,4,2
	3,0,0,3	3,0,1,3	3,0,2,3	3,0,3,3	3,0,4,3
	3,0,0,4	3,0,1,4	3,0,2,4	3,0,3,4	3,0,4,4
Elemento	3,1,0,0	3,1,1,0	3,1,2,0	3,1,3,0	3,1,4,0
	3,1,0,1	3,1,1,1	3,1,2,1	3,1,3,1	3,1,4,1
	3,1,0,2	3,1,1,2	3,1,2,2	3,1,3,2	3,1,4,2
	3,1,0,3	3,1,1,3	3,1,2,3	3,1,3,3	3,1,4,3
	3,1,0,4	3,1,1,4	3,1,2,4	3,1,3,4	3,1,4,4
Elemento	3,2,0,0	3,2,1,0	3,2,2,0	3,2,3,0	3,2,4,0
	3,2,0,1	3,2,1,1	3,2,2,1	3,2,3,1	3,2,4,1
	3,2,0,2	3,2,1,2	3,2,2,2	3,2,3,2	3,2,4,2
	3,2,0,3	3,2,1,3	3,2,2,3	3,2,3,3	3,2,4,3
	3,2,0,4	3,2,1,4	3,2,2,4	3,2,3,4	3,2,4,4

Es un arreglo de $4 \times 3 \times 5 \times 5$, el cual contiene 300 elementos.

Definición del arreglo tetradimensional

Como ya lo hemos mencionado, al definir un arreglo es necesario hacerlo como una variable, por lo cual, en la parte de declaración de variables, se utiliza el siguiente formato:

```
nomVar: Arreglo[priDim][segDim][terDim][cuarDim] Tipo de dato
```

En donde:

nomVar	Es el nombre de la variable.
Arreglo	Indica que es un arreglo.
priDim	Indica la cantidad de elementos de la primera dimensión del arreglo (10 por ejemplo).
segDim	Indica la cantidad de elementos de la segunda dimensión del arreglo.
terDim	Indica la cantidad de elementos de la tercera dimensión del arreglo.
cuarDim	Indica la cantidad de elementos de la cuarta dimensión del arreglo.
Tipo de dato	Es el tipo de dato de los elementos del arreglo.

Ejemplo:

```
numeros: Arreglo[4][3][5][5] Entero
```

numeros	Es un arreglo de cuatro dimensiones: 4 renglones por 3 columnas, donde cada elemento es un arreglo de 5 x 5; es decir, una matriz de 4 x 3, donde cada uno de los elementos es una matriz de 5 x 5.
---------	---

Manejo de los elementos del arreglo tetradimensional

Para relacionar cada elemento individual de un arreglo de cuatro dimensiones se usan cuatro subíndices; el primero indica la primera dimensión (renglón de la primera matriz) del elemento, el segundo la segunda dimensión (columna de la primera matriz), el tercero la tercera dimensión (renglón de la matriz dentro de cada elemento de la primer matriz) y el cuarto la cuarta dimensión (columna de la matriz dentro de cada elemento de la primer matriz), como sigue:

```
tetra[ren1][col1][ren2][col2]
```

En donde:

ren1	Indica el número de elemento en la primera dimensión.
col1	Indica el número de elemento en la segunda dimensión.
ren2	Indica el número de elemento en la tercera dimensión.
col2	Indica el número de elemento en la cuarta dimensión.

Nota: Los subíndices pueden ser constantes, variables o expresiones de tipo entero.



Los subíndices pueden ser constantes, variables o expresiones de tipo entero.

Al igual que toda variable, una de tipo arreglo tetradimensional puede usarse para leer datos, asignarle valores mediante expresiones aritméticas, imprimir su contenido, etcétera.

Ejemplos:

```
tetra[3][2][3][4] = 50
Leer tetra[3][2][2][3]
Leer tetra[3][1][1][3]
tetra[3][1][1][4] = tetra[3][2][2][3]+
                    tetra[3][1][1][3]+tetra[3][2][3][4]
Imprimir tetra[3][1][1][4]
```

Ejemplo:

Elaborar un algoritmo que lea números de tipo entero para un arreglo como el esquematizado líneas antes, es decir, un arreglo de 4 x 3 x 5 x 5, y los imprima.

```
Algoritmo ARREGLO TETRADIMENSIONAL
Clase ArregloCuatroDim1
1. Método principal()
  a. Declarar variables
    numeros: Arreglo[4][3][5][5] Entero
    ren1, col1, ren2 , col2: Entero
  b. for ren1=0; ren1<=3; ren1++
    1. for col1=0; col1<=2; col1++
      a. for ren2=0; ren2<=4; ren2++
        1. for col2=0; col2<=4; col2++
          a. Solicitar
            numeros[ren1][col1][ren2][col2]
          b. Leer numeros[ren1][col1][ren2][col2]
        2. endfor
      b. endfor
    2. endfor
  c. endfor
  d. for ren1=0; ren1<=3; ren1++
    1. for col1=0; col1<=2; col1++
      a. for ren2=0; ren2<=4; ren2++
        1. for col2=0; col2<=4; col2++
          a. Imprimir
            numeros[ren1][col1][ren2][col2]
        2. endfor
      b. endfor
    2. endfor
  e. endfor
  f. Fin Método principal
Fin Clase ArregloCuatroDim1
Fin
```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: ArregloCuatroDim1.java

Explicación:

Se utilizan cuatro ciclos for: el primero, y más externo, para procesar los elementos de la primera dimensión (renglones de la matriz más externa); el segundo, que está anidado dentro del primero, para procesar la segunda dimensión (columnas de la matriz más externa); el tercer for para procesar la tercera dimensión (renglones de la matriz más interna); y el cuarto ciclo, que está anidado dentro del tercero, para procesar la cuarta dimensión (las columnas de la matriz más interna). En nuestro ejemplo necesitamos un ciclo for desde 0 hasta 3 para procesar cada uno de los cuatro elementos de la primera dimensión, dentro de éste un ciclo de 0 a 2 para procesar las columnas de la matriz más externa, dentro de éste un ciclo de 0 a 4 para procesar los renglones de la matriz más interna y, dentro del proceso de cada renglón de esta matriz, un ciclo de 0 a 4 para procesar las columnas (la cuarta dimensión).

6.4.1 Ejercicios resueltos para tetradimensionales

Tabla 6.4: ejercicios resueltos disponibles en la zona de descarga del capítulo 6 de la Web del libro.

Ejercicio	Descripción
Ejercicio 6.4.1.1	Maneja la producción de una empresa en un arreglo de cuatro dimensiones
Ejercicio 6.4.1.2	Similar al anterior, pero con más cálculos
Ejercicio 6.4.1.3	Similar al anterior, pero con más cálculos

6.5 Ejercicios propuestos

1. Elaborar un algoritmo que permita leer 20 nombres de personas en un arreglo y consultarlos de acuerdo con el número de posición que ocupan dentro del arreglo.
2. Elaborar un algoritmo para leer 30 días de ventas realizadas por un empleado y que imprima el día en que tuvo la mayor y la menor venta, así como las cantidades correspondientes.
3. Elaborar un algoritmo que permita leer 30 números en un arreglo y que imprima el número mayor, el menor y la cantidad de veces que se repiten ambos.
4. Elaborar un algoritmo similar al del Ejercicio 6.1.1.4, con la excepción de que al final debe imprimir:

TOTAL DE ELEMENTOS ARRIBA DE LA MEDIA:	999
TOTAL DE ELEMENTOS ABAJO DE LA MEDIA:	999
TOTAL DE ELEMENTOS IGUAL A LA MEDIA:	999
5. Se tienen varios obreros, y por cada obrero se tienen los siguientes datos: nombre y la producción de los 30 días del mes. Elaborar un algoritmo que los lea y genere el siguiente reporte:

NOMBRE	REPORTE MENSUAL DE PRODUCCIÓN		DÍAS ARRIBA DEL PROM.
	PROD. MES	PROM. DIARIO	
XXXXXXXXXXXXXXXXXXXXXX	999	999	999
XXXXXXXXXXXXXXXXXXXXXX	999	999	999
.			
XXXXXXXXXXXXXXXXXXXXXX	999	999	999
TOTAL 999	9999	9999	9999

6. Una empresa tiene varios vendedores, y por cada vendedor se tiene el nombre y la venta que realizó. Elaborar un algoritmo permita leer dichos datos y que proporcione un reporte de comisiones de ventas en el cual aparezcan todos los vendedores que tengan ventas mayores que el nivel de comisión, el cual se calcula así:

Nivel de comisión = $3/4$ * (promedio de ventas)

Comisión = 5 % sobre el excedente de lo que vendió por arriba del nivel de comisión.

NOMBRE DEL VENDEDOR	COMISIONES DE VENDEDORES	
	VENTAS	COMISIÓN
XXXXXXXXXXXXXXXXXXXXXX	999,999.99	99,999.99
XXXXXXXXXXXXXXXXXXXXXX	999,999.99	99,999.99

XXXXXXXXXXXXXXXXXXXXXX	999,999.99	99,999.99
TOTAL 999 VENDEDORES	9,999,999.99	999,999.99

7. Elaborar un algoritmo que permita leer 15 números en un arreglo y pregunte si se desea introducir un nuevo número en lugar de cualquiera de los que están en el arreglo; entonces deberá leer el número que se introdujo y el lugar del elemento por el que se cambiará para hacer el cambio e imprimir el arreglo antes y después del cambio.
8. Elaborar un algoritmo que permita leer 25 números en un arreglo e imprima el arreglo y la cantidad de números que son cero, la cantidad de números que están abajo de cero y la cantidad de números arriba de cero.
9. Elaborar un algoritmo que permita leer 50 números enteros en un arreglo e imprimirlos de mayor a menor.
10. Elaborar un algoritmo que permita leer 15 nombres de personas, los clasifique en orden descendente y los imprima.
11. Elaborar un algoritmo que lea números en una matriz de 6 x 6, la imprima y coloque al final la suma por columnas.
12. Se tienen 15 estaciones de trabajo, cada una de las cuales tiene un encargado, del cual se conocen su nombre y la producción que tuvo por cada uno de los meses del año. Elaborar un algoritmo que lea los 15 nombres y los guarde en un arreglo, y que haga lo mismo con los 12 meses de producción de cada una de las estaciones y los almacene en una matriz de 15 x 12. Se requiere que imprima el siguiente reporte:

ANÁLISIS DE PRODUCCIÓN	
ESTACIÓN	TOTAL PRODUCCIÓN
99	999999
99	999999

99	999999
TOTAL	9999999

ESTACIÓN MÁS PRODUCTIVA: 99
 ENCARGADO DE LA ESTACIÓN: XXXXXXXXXXXXXXXXXXXXXXXX
 CANTIDAD PRODUCIDA:999999

13. Elaborar un algoritmo que lea el nombre de 20 trabajadores y su producción mensual por cada uno de los 12 meses del año en dos arreglos: uno para nombres y otro para producciones, en los cuales el elemento N corresponde al trabajador N. Se requiere que imprima el siguiente reporte:

ANÁLISIS DE PRODUCCIÓN	
NOMBRE	TOTAL PRODUCCIÓN
XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99999
XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99999

XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99999
PROMEDIO DE PRODUCCIÓN:	99999

Nota: Debe imprimir sólo los que tengan producción por arriba del promedio.

14. Se tiene la producción de los 7 días de la semana de 20 plantas. Elaborar un algoritmo que lea estos datos y los almacene en un arreglo de 20 renglones para las plantas, en 7 columnas para cada día de producción, y que imprima el número de planta que tuvo la mayor producción semanal.
15. Igual al ejercicio anterior, sólo que ahora deberá imprimir el número de planta con mayor producción en un día, el día en que esto sucedió y con cuánta producción.
16. Elaborar un algoritmo que genere una matriz de 10 x 10, en la cual asigne ceros a todos los elementos, excepto a los de la diagonal principal, donde asignará unos. Imprimir dicha matriz.
17. Elaborar un algoritmo que genere una matriz de 10 x 10, en la cual asigne ceros a todos los elementos desde la diagonal principal hacia abajo y a los demás los coloque unos. Imprimir dicha matriz.
18. Generar una matriz de 10 x 10, con ceros en los elementos desde la diagonal principal hacia arriba; a los demás deberá colocarle unos. Imprimir dicha matriz.
19. Elaborar un algoritmo que permita leer números en una matriz de 6 x 7 y haga lo propio para un vector de 6 elementos. Calcular el producto de matriz por vector fila. (El producto de la matriz por el vector fila es un vector donde el elemento 1 está dado por la sumatoria de los productos del elemento 1 del vector por cada uno de los elementos de la columna 1 de la matriz, y así para el 2, 3, etc.). Imprimir la matriz, el vector y el vector resultante.

20. Elaborar un algoritmo que permita leer números en una matriz de 5 x 6. Debe imprimir dicha matriz y la sumatoria por renglones y por columnas utilizando un arreglo unidimensional para obtener la sumatoria de todos los renglones y otro arreglo de una dimensión para calcular la sumatoria de todas las columnas.
21. Una matriz es nula si todos sus elementos son iguales a cero. Elaborar un algoritmo que lea números en una matriz de 5 x 4 e indique si la matriz es nula o no.
22. Elaborar un algoritmo que permita leer números para una matriz A de 6 x 4, haga lo propio para una matriz B, las imprima e indique si la matriz A es mayor que la matriz B. (Para que A sea mayor que B, cada elemento A_{ij} debe ser por lo menos igual que cada elemento B_{ij} correspondiente y debe existir al menos un A_{ij} mayor que su correspondiente B_{ij} ; en caso contrario, A no es mayor que B).
23. Elaborar un algoritmo que lea un número, así como los números para una matriz de 5 x 6. Calcular la matriz producto por un número, esto es, multiplicar cada elemento de la matriz por el número y colocarlo en el elemento correspondiente de la matriz resultante. Imprimir las dos matrices.
24. Una compañía manufacturera tiene 12 plantas. Elaborar un algoritmo que permita leer el nombre de cada planta y la producción que se hizo en cada uno de los siete días de la semana. Utilizar un arreglo de una dimensión para leer los nombres de las plantas y un arreglo de dos dimensiones (12 x 7) para leer la producción de las doce plantas (uno en cada renglón) en los siete días, una columna para cada día. La idea es leer el nombre de la primera planta y luego la producción hecha en cada uno de los siete días, luego procesar la planta 2, posteriormente la 3 y así sucesivamente. Imprimir el siguiente reporte:

PLANTA	REPORTE SEMANAL DE PRODUCCIÓN							PROD. SEMANAL
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	DÍA 6	DÍA 7	
XXXXXXX	999	999	999	999	999	999	999	999
XXXXXXX	999	999	999	999	999	999	999	999

XXXXXXX	999	999	999	999	999	999	999	999
TOTALES	999	999	999	999	999	999	999	999

PLANTA MÁS PRODUCTIVA: XXXXXXXXXXXXXXXXXXXXXXXXXX
 PRODUCCIÓN DE LA PLANTA MÁS PRODUCTIVA: 999
 DÍA CON MAYOR PRODUCCIÓN: 999
 MAYOR PRODUCCIÓN EN UN DÍA: 999

25. Elaborar un algoritmo similar al anterior, excepto que ahora en el reporte debe agregarse un dato más por cada planta: el día más productivo. Es decir, de los siete días, imprimir el número de día en el que tuvo la mayor producción.
26. Elaborar un algoritmo que permita leer números enteros para cada uno de los elementos de una matriz A de 3 x 4, lo mismo para una matriz B de 4 x 5 y calcular una matriz P de 3 x 5, multiplicando la matriz A por la matriz B.
27. Una compañía departamental tiene 8 sucursales, en cada sucursal tiene 6 departamentos de ventas y, por cada departamento, se tiene la venta que se hizo en cada uno de los 7 días de la semana. Elaborar un algoritmo que lea los datos de las ventas en un arreglo de tres dimensiones: la primera dimensión la conforman las sucursales, la segunda dimensión los departamentos, y la tercera, los 7 días de la semana. Esquemáticamente:

		Día 1	Día 2	Día 3	Día 4	Día 5	Día 6	Día 7
Sucursal 1	Depto. 1							
	Depto. 2							
	Depto. 3							
	Depto. 4							
	Depto. 5							
	Depto. 6							
Sucursal 2								
Sucursal 8								

Una vez leídos los datos, que imprima el siguiente reporte:

REPORTE SEMANAL DE VENTAS

SUCURSAL 1							
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	DÍA 6	DÍA 7
DEPARTAMENTO-1	---	---	---	---	---	---	---
DEPARTAMENTO-2	---	---	---	---	---	---	---
DEPARTAMENTO-3	---	---	---	---	---	---	---
DEPARTAMENTO-4	---	---	---	---	---	---	---
DEPARTAMENTO-5	---	---	---	---	---	---	---
DEPARTAMENTO-6	---	---	---	---	---	---	---
SUCURSAL 2							

SUCURSAL 8							
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	DÍA 6	DÍA 7
DEPARTAMENTO-1	---	---	---	---	---	---	---
DEPARTAMENTO-2	---	---	---	---	---	---	---
DEPARTAMENTO-3	---	---	---	---	---	---	---
DEPARTAMENTO-4	---	---	---	---	---	---	---
DEPARTAMENTO-5	---	---	---	---	---	---	---
DEPARTAMENTO-6	---	---	---	---	---	---	---

28. Elaborar un algoritmo similar al anterior, pero ahora debe imprimir el total de ventas por departamento y por cada día para cada sucursal y el total general de ventas. Esquemáticamente:

REPORTE SEMANAL DE VENTAS

SUCURSAL 1								
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	DÍA 6	DÍA 7	TOTAL
DEPARTAMENTO-1	---	---	---	---	---	---	---	---
DEPARTAMENTO-2	---	---	---	---	---	---	---	---
DEPARTAMENTO-3	---	---	---	---	---	---	---	---
DEPARTAMENTO-4	---	---	---	---	---	---	---	---
DEPARTAMENTO-5	---	---	---	---	---	---	---	---
DEPARTAMENTO-6	---	---	---	---	---	---	---	---
TOTALES	---	---	---	---	---	---	---	---
SUCURSAL 2								

SUCURSAL 8								
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	DÍA 6	DÍA 7	TOTAL
DEPARTAMENTO-1	---	---	---	---	---	---	---	---
DEPARTAMENTO-2	---	---	---	---	---	---	---	---
DEPARTAMENTO-3	---	---	---	---	---	---	---	---
DEPARTAMENTO-4	---	---	---	---	---	---	---	---
DEPARTAMENTO-5	---	---	---	---	---	---	---	---
DEPARTAMENTO-6	---	---	---	---	---	---	---	---
TOTALES	---	---	---	---	---	---	---	---
TOTAL GENERAL DE VENTAS								---

29. Elaborar un algoritmo similar al anterior, pero además debe imprimir el total de ventas general por cada sucursal, el departamento que tuvo la mayor venta de cada sucursal y cuánto fue la venta, así como el total de ventas general, la sucursal que tuvo la mayor venta y cuánto fue ésta.
30. Una compañía departamental tiene 8 sucursales, en cada sucursal tiene 6 departamentos de ventas, por cada departamento se tienen 4 vendedores y por cada vendedor se tiene la venta que se hizo en cada uno de los 7 días de la semana. Elaborar un algoritmo que lea los datos de las ventas en un arreglo de cuatro dimensiones: la primera dimensión la conforman las sucursales, la segunda dimensión, los departamentos, la tercera, los vendedores y la cuarta, los 7 días de la semana. Esquemáticamente:

		Departamento 1							Departamento 6		
		Día -----Día									
		1	2	3	4	5	6	7			
Sucursal 1	Vendedor 1										
	Vendedor 2										
	Vendedor 3										
	Vendedor 4										
Sucursal 2											
Sucursal 8											

Una vez leídos los datos, que imprima el siguiente reporte:

REPORTE SEMANAL DE VENTAS

SUCURSAL 1							
DEPARTAMENTO 1							
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	DÍA 6	DÍA 7
VENDEDOR-1	---	---	---	---	---	---	---
VENDEDOR-2	---	---	---	---	---	---	---
VENDEDOR-3	---	---	---	---	---	---	---
VENDEDOR-4	---	---	---	---	---	---	---
DEPARTAMENTO 2							

DEPARTAMENTO 6							
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	DÍA 6	DÍA 7
VENDEDOR-1	---	---	---	---	---	---	---
VENDEDOR-2	---	---	---	---	---	---	---
VENDEDOR-3	---	---	---	---	---	---	---
VENDEDOR-4	---	---	---	---	---	---	---
SUCURSAL 8							
DEPARTAMENTO 1							

DEPARTAMENTO 6							

31. Elaborar un algoritmo similar al anterior, pero que ahora imprima el siguiente reporte:

REPORTE SEMANAL DE VENTAS								
SUCURSAL 1								
DEPARTAMENTO 1								
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	DÍA 6	DÍA 7	TOTAL
VENDEDOR-1	---	---	---	---	---	---	---	---
VENDEDOR-2	---	---	---	---	---	---	---	---
VENDEDOR-3	---	---	---	---	---	---	---	---
VENDEDOR-4	---	---	---	---	---	---	---	---
TOTALES	---	---	---	---	---	---	---	---

DEPARTAMENTO 8								
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	DÍA 6	DÍA 7	TOTAL
VENDEDOR-1	---	---	---	---	---	---	---	---
VENDEDOR-2	---	---	---	---	---	---	---	---
VENDEDOR-3	---	---	---	---	---	---	---	---
VENDEDOR-4	---	---	---	---	---	---	---	---
TOTALES	---	---	---	---	---	---	---	---

SUCURSAL 2								
DEPARTAMENTO 1								

DEPARTAMENTO 6								

SUCURSAL 8								
DEPARTAMENTO 1								

DEPARTAMENTO 6								

TOTAL GENERAL DE PRODUCCIÓN								

32. Elaborar un algoritmo similar al anterior pero, además, al final de cada sucursal debe imprimir el departamento que tuvo la mayor venta y cuánto fue ésta. Al final del reporte imprimir la venta total de toda la compañía, la sucursal que tuvo la mayor venta y cuánto fue ésta.

6.6 Resumen de conceptos que debe dominar

- Definición, manipulación y utilización de arreglos:
 - unidimensionales (una dimensión)
 - bidimensionales (dos dimensiones)
 - tridimensionales (tres dimensiones)
 - tetradimensionales (cuatro dimensiones)

6.7 Contenido de la página Web de apoyo.....

El material marcado con asterisco () sólo está disponible para docentes.*

6.7.1 Resumen gráfico del capítulo

6.7.2 Autoevaluación

6.7.3 Programas en Java

6.7.4 Ejercicios resueltos

6.7.5 Power Point para el profesor (*)

7

Métodos

Contenido

- 7.1 Métodos que no regresan valor
- 7.2 Formato general de una clase con métodos
- 7.3 Variables globales, locales y parámetros
 - 7.3.1 Variables globales
 - 7.3.2 Variables locales
 - 7.3.3 Parámetros
- 7.4 Funciones estándar
 - 7.4.1 Funciones cadena de caracteres
 - 7.4.2 Validación de la entrada de datos
 - 7.4.3 Funciones especiales
- 7.5 Métodos que regresan valor
- 7.6 Ejercicios resueltos
- 7.7 Ejercicios propuestos
- 7.8 Resumen de conceptos que debe dominar
- 7.9 Contenido de la página Web de apoyo
 - El material marcado con asterisco (*) sólo está disponible para docentes.*
 - 7.9.1 Resumen gráfico del capítulo
 - 7.9.2 Autoevaluación
 - 7.9.3 Programas en Java
 - 7.9.4 Ejercicios resueltos
 - 7.9.5 Power Point para el profesor (*)

Objetivos del capítulo

- Ilustrar cómo utilizar más de un método dentro de una clase.
- Estudiar el uso de métodos que no regresan valor.
- Aprender el manejo de variables de clase (globales), variables locales y el uso de parámetros por valor y por referencia.
- Estudiar el uso de métodos que regresan valor.
- Ilustrar el uso de algunas funciones estándar como son: funciones para la manipulación de cadenas de caracteres y algunas funciones especiales.
- Aplicar lo aprendido en la solución de ejercicios resueltos y propuestos.

Competencias

- Competencia general del capítulo
 - Analizar problemas y diseñar algoritmos que los solucionen aplicando métodos.
- Competencias específicas del capítulo
 - Diseña algoritmos aplicando métodos que no regresan valor.
 - Elabora algoritmos aplicando variables de clase (globales), variables locales y parámetros por valor y por referencia.
 - Desarrolla algoritmos aplicando métodos que regresan valor.
 - Diseña algoritmos aplicando funciones para la manipulación de cadenas de caracteres y algunas funciones especiales.

Introducción

Con el estudio del capítulo anterior, usted ya domina la estructura de datos denominada arreglos y cómo diseñar algoritmos usando esa estructura.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos utilizando métodos; es decir, en una clase, además del método principal, utilizar otros métodos.

Se expone el uso de métodos que no regresan valor (void) y métodos que regresan valor (definidas por el usuario). También se detalla el manejo de variables de clase (globales), locales y el uso de parámetros para conectarlas. Asimismo, se estudian algunas funciones estándar, como son: funciones para la manipulación de cadenas de caracteres, validación de la entrada de datos y algunas funciones especiales.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejercite estudiando los problemas planteados en los ejercicios resueltos y propuestos. Al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro, luego verifique sus resultados con los del libro, analice las diferencias y vea sus errores. Al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no está igual que el del libro, no necesariamente está mal. Usted debe ir aprendiendo a analizar las diferencias y a comprender que a veces, aunque haya diferencias, las dos soluciones están correctas.

En el siguiente capítulo se estudian las estructuras de datos: registros y archivos.

7.1 Métodos que no regresan valor

Un método es una función, módulo, segmento, rutina, subrutina o subalgoritmo que puede ser definido dentro de una clase con el propósito de ejecutar una función, comportamiento o tarea específica, pudiendo ser llamado o invocado desde el método principal u otro método cuando se requiera. Existen dos tipos de métodos: métodos que no regresan valor y métodos que regresan valor. En este punto se estudian los métodos que no regresan valor y más adelante, en otro apartado, se tratarán los métodos que regresan valor.

En los capítulos precedentes se han planteado problemas pequeños, es decir, que hacen una sola cosa o tienen una sola función, y en consecuencia se han diseñado algoritmos que tienen una sola clase, dentro de la cual se tiene sólo el método principal, donde se implementa la lógica que resuelve el problema.

Sin embargo, en ocasiones se tienen problemas más grandes y complejos que involucran más de una tarea o función, complicando la solución en un solo método. Ahora estudiaremos cómo utilizar más de un método dentro de la clase. La idea es que se diseñe un método por separado para implementar cada función y, en su momento, desde el método principal se llama o invoca a cada método para que haga su tarea o función; el método principal, que es donde inicia el funcionamiento del

algoritmo, dirigirá la lógica general de la solución y se apoyará llamando a los otros métodos para que realicen sus funciones específicas en el momento que se requieran. Así, el formato de la clase será el siguiente:

```
Algoritmo ALGO
Clase NomClase
  1. Método principal()
    a. Acción a
    b. Acción b
    c. Acción c
    d. Fin Método principal

  2. Método funcionUno()
    a. Acción a
    b. Acción b
    c. Acción c
    d. Fin Método funcionUno

  3. Método funcionDos()
    a. Acción a
    b. Acción b
    c. Acción c
    d. Fin Método funcionDos

  4. Método funcionTres()
    a. Acción a
    b. Acción b
    c. Acción c
    d. Fin Método funcionTres
Fin Clase NomClase
Fin
```

En donde:

Se tiene el esquema de un algoritmo que tiene una clase y dentro de la clase se tienen cuatro métodos: el Método principal, el Método funcionUno, el Método funcionDos y el Método funcionTres.

Cada uno de los métodos funcionUno, funcionDos y funcionTres tiene la lógica necesaria para implementar una función específica para resolver el problema. Sin embargo, en el Método principal es donde inicia el funcionamiento del algoritmo, el que tiene la lógica general que resuelve el problema y en su momento deberá llamar a cada uno de los otros métodos de la clase, es decir, al Método funcionUno, al Método funcionDos y al Método funcionTres.

Ejemplo:

Elaborar un algoritmo que ayude a un niño a revisar sus tareas referentes a las operaciones aritméticas fundamentales: sumar, restar, multiplicar y dividir.

El proceso es el siguiente:

Se ofrecerá un menú de opciones para escoger lo que desee hacer de acuerdo con el siguiente formato:

TE PUEDO AYUDAR A:
1. SUMAR 2. RESTAR 3. MULTIPLICAR 4. DIVIDIR 5. FIN
ESCOGER OPCIÓN

En caso de que el niño escoja la opción 1, está indicando que desea revisar operaciones de sumar. Enseguida se debe establecer un proceso interactivo para que el niño introduzca los dos números que se van a sumar y su resultado, luego para que la computadora le indique si la suma está correcta o incorrecta y después preguntar si desea revisar otra suma. Si es así, deberá repetir todo el proceso para revisar una nueva operación de sumar; algo parecido a lo siguiente:

Ayudando a revisar operaciones de sumar

Teclee primer número: 45

Teclee segundo número: 13 +

Teclee el resultado: 58

La suma está correcta

¿Desea revisar otra suma (S/N)?

Para el caso de la resta, multiplicación y división se seguirá un proceso similar, pero con las diferencias correspondientes.

¿Qué se requiere para solucionar este problema?

El problema tiene cuatro funciones o tareas específicas:

Sumar	Es la parte que permite ayudar a revisar operaciones de sumar.
Restar	Es la parte que permite ayudar a revisar operaciones de restar.
Multiplicar	Es la parte que permite ayudar a revisar operaciones de multiplicar.
Dividir	Es la parte que permite ayudar a revisar operaciones de dividir.

Por tanto, se requiere un método para implementar la solución de cada una de esas funciones, es decir, un método para sumar, otro método para restar, otro método para multiplicar y otro método para dividir, además del método principal que dirigirá el funcionamiento general del algoritmo, que llamará a los métodos sumar, restar, multiplicar y dividir cuando requiera que cada uno haga su tarea o función.

A continuación se presenta el algoritmo de la solución:

Algoritmo AYUDA

Clase Ayuda

1. Declaraciones de clase

Variables

num1, num2, resuNi, resuMaq: Entero

desea: Carácter

opcion: Entero

2. Método principal()

a. do

1. Imprimir el menú de opciones

TE PUEDO AYUDAR A:
1. SUMAR
2. RESTAR
3. MULTIPLICAR
4. DIVIDIR
5. FIN
ESCOGER OPCIÓN

2. Leer opcion

3. switch opcion

1: ayudaSumar()

2: ayudaRestar()

3: ayudaMultiplicar()

4: ayudaDividir()

4. endswitch

b. while opcion != 5

c. Fin Método principal

3. Método ayudaSumar()

a. do

1. Solicitar número uno, número dos, resultado

2. Leer num1, num2, resuNi

3. Calcular resuMaq = num1 + num2

4. if resuMaq == resuNi then

a. Imprimir "La suma esta correcta"

5. else

a. Imprimir "La suma esta incorrecta"

6. endif

7. Preguntar "¿Desea revisar otra suma (S/N)?"

8. Leer desea

```
    b. while desea == 'S'
    c. Fin Método ayudaSumar

4. Método ayudaRestar()
  a. do
    1. Solicitar número uno, número dos, resultado
    2. Leer num1, num2, resuNi
    3. Calcular resuMaq = num1 - num2
    4. if resuMaq == resuNi then
      a. Imprimir "La resta esta correcta"
    5. else
      a. Imprimir "La resta esta incorrecta"
    6. endif
    7. Preguntar "¿Desea revisar otra resta (S/N)?"
    8. Leer desea
  b. while desea == 'S'
  c. Fin Método ayudaRestar

5. Método ayudaMultiplicar()
  a. do
    1. Solicitar número uno, número dos, resultado
    2. Leer num1, num2, resuNi
    3. Calcular resuMaq = num1 * num2
    4. if resuMaq == resuNi then
      a. Imprimir "La multiplicación esta correcta"
    5. else
      a. Imprimir "La multiplicación esta incorrecta"
    6. endif
    7. Preguntar "¿Desea revisar otra multiplicación (S/N)?"
    8. Leer desea
  b. while desea == 'S'
  c. Fin Método ayudaMultiplicar

6. Método ayudaDividir()
  a. do
    1. Solicitar número uno, número dos, resultado
    2. Leer num1, num2, resuNi
    3. Calcular resuMaq = num1 / num2
    4. if resuMaq == resuNi then
      a. Imprimir "La división esta correcta"
    5. else
      a. Imprimir "La división esta incorrecta"
    6. endif
    7. Preguntar "¿Desea revisar otra división (S/N)?"
    8. Leer desea
  b. while desea == 'S'
  c. Fin Método ayudaDividir
Fin Clase Ayuda
Fin
```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Ayuda.java

Explicación:

Algoritmo AYUDA

Es el encabezado del algoritmo

1. Declaraciones de clase (globales a la clase)

Se declaran las variables

num1, num2, resuNi, resuMaq: de tipo Entero.

desea: de tipo Carácter.

opcion: de tipo Entero.

En esta parte se hacen las declaraciones de clase. Se pueden declarar tipos, constantes y variables, y podrán ser utilizadas en cualquier método de la clase.

2. Método principal()

a. Inicia ciclo do.

1. Imprimir el menú de opciones donde se solicita la opción.

2. Se lee la respuesta en opcion.

3. Inicia switch opcion:

Si opcion es 1, entonces: Llama al método ayudaSumar().

Si opcion es 2, entonces: Llama al método ayudaRestar().

Si opcion es 3, entonces: Llama al método ayudaMultiplicar().

Si opcion es 4, entonces: Llama al método ayudaDividir().

4. Fin del switch.

b. Fin ciclo while opcion != 5. Va al do.

c. Fin Método principal.

Todo algoritmo diseñado con métodos inicia su funcionamiento en el método principal, que será el que dirigirá la operación del resto de los métodos, es decir, los llamará para que realicen la tarea que les corresponde.

3. Método ayudaSumar()

a. Inicia ciclo do.

1. Solicita número uno, número dos y resultado del niño.

2. Lee en num1, num2, resuNi.

3. Calcula resultado de la máquina.

4. Si resuMaq == resuNi entonces:

a. Imprime "La suma esta correcta".

5. Si no:

a. Imprime "La suma esta incorrecta".

6. Fin if.

7. Pregunta "¿Desea revisar otra suma (S/N)?".

8. Lee la respuesta en desea.

b. Fin ciclo while desea == 'S'. Regresa al do.

c. Fin Método ayudaSumar.

En este método se ayuda a revisar operaciones de sumar.

4. Método `ayudaRestar()`
 - a. Inicia ciclo `do`.
 1. Solicita número uno, número dos y resultado del niño.
 2. Lee en `num1`, `num2`, `resuNi`.
 3. Calcula resultado de la máquina.
 4. Si `resuMaq == resuNi` entonces:
 - a. Imprime “La resta esta correcta”.
 5. Si no:
 - a. Imprime “La resta esta incorrecta”.
 6. Fin del `if`.
 7. Pregunta “¿Desea revisar otra resta (S/N)?”.
 8. Lee la respuesta en `desea`.
 - b. Fin ciclo `while desea == 'S'`. Regresa al `do`.
 - c. Fin Método `ayudaRestar`.

En este método se ayuda a revisar operaciones de restar.

5. Método `ayudaMultiplicar()`
 - a. Inicia ciclo `do`.
 1. Solicita número uno, número dos y resultado del niño.
 2. Lee en `num1`, `num2`, `resuNi`.
 3. Calcula resultado de la máquina.
 4. Si `resuMaq == resuNi` entonces:
 - a. Imprime “La multiplicación esta correcta”.
 5. Si no:
 - a. Imprime “La multiplicación esta incorrecta”.
 6. Fin del `if`.
 7. Pregunta “¿Desea revisar otra multiplicación (S/N)?”.
 8. Lee la respuesta en `desea`.
 - b. Fin ciclo `while desea == 'S'`. Regresa al `do`.
 - c. Fin Método `ayudaMultiplicar`.

En este método se ayuda a revisar operaciones de multiplicar.

6. Método `ayudaDividir()`.
 - a. Inicia ciclo `do`.
 1. Solicita número uno, número dos y resultado del niño.
 2. Lee en `num1`, `num2`, `resuNi`.
 3. Calcula resultado de la máquina.
 4. Si `resuMaq == resuNi` entonces:
 - a. Imprime “La división esta correcta”.

5. Si no:
 - a. Imprime “La división esta incorrecta”.
6. Fin del if.
7. Pregunta “¿Desea revisar otra división (S/N)?”.
8. Lee la respuesta en `desea`.
- b. Fin ciclo while `desea == 'S'`. Regresa al do.
- c. Fin Método `ayudaDividir`.

En este método se ayuda a revisar operaciones de dividir.

Luego se tiene el fin de la clase y el fin del algoritmo.

7.2 Formato general de una clase con métodos

Una clase puede tener más de un método. A continuación se explica el formato general de un algoritmo para cuando se presenta esta situación. Como ya sabemos, la clase empieza con el encabezado de la clase:

```
Clase NomClase
```

Parte 1. Se tiene una parte de declaraciones de clase:

1. Declaraciones de clase
 - Constantes
 - Tipos
 - Variables

Éstas son declaraciones globales en la clase, es decir, se pueden utilizar en cualquier método de la clase. Esta parte es opcional; es decir, puede estar presente o no.

Parte 2. Se tiene el método principal. Es el paso 2 dentro de la clase, si hay declaraciones de clase; pero si no hay declaraciones de clase, éste es el paso 1. Como ya se indicó anteriormente, éste es el método obligatorio y es donde inicia el funcionamiento de la clase:

2. Método `principal()`
 - a. Declarar
 - Constantes
 - Tipos
 - Variables
 - b. Acción1
 - c. Acción2
 - d. AcciónN
 - e. Fin Método principal

Parte 3. Se tienen uno o más métodos subordinados, que son opcionales:

```
3. Método funcionUno()
  a. Declarar
     Constantes
     Tipos
     Variables
  b. Acción1
  c. Acción2
  d. AcciónN
  e. Fin Método funcionUno

4. Método funcionDos()
  a. Declarar
     Constantes
     Tipos
     Variables
  b. Acción1
  c. Acción2
  d. AcciónN
  e. Fin Método funcionDos

5. Método funcionTres()
  a. Declarar
     Constantes
     Tipos
     Variables
  b. Acción1
  c. Acción2
  d. AcciónN
  e. Fin Método funcionTres
```

```
Fin Clase NomClase
```

Por último se tiene el fin de la clase.

Como se observa, cada método de la clase puede tener su parte de declaraciones: tipos, constantes y variables, la cual es opcional. Lo que ahí se declare sólo puede ser utilizado en el método correspondiente; es decir, se les conoce como locales al método donde se definen.

7.3 Variables globales, locales y parámetros

7.3.1 Variables globales

Cuando un algoritmo utiliza métodos se pueden declarar variables tanto en el contexto global de la clase como de manera local en cada método. A las variables definidas en el contexto global se les llama variables globales o variables de clase, las cuales pueden utilizarse en cualquier método de la clase. A continuación se presenta un algoritmo que muestra cómo usar una variable global o de clase:

```

Algoritmo VARIABLE GLOBAL
Clase VarGlobal
  1. Declaraciones de clase
    Variables
    x: Entero

  2. Método principal()
    a. x = 0
    b. cambiar()
    c. Imprimir x
    d. Fin Método principal

  3. Método cambiar()
    a. x = 1
    b. Fin Método cambiar
Fin Clase VarGlobal
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: VarGlobal.java



Explicación:

Este algoritmo VARIABLE GLOBAL tiene el método principal y un método subordinado llamado `cambiar()`. Se utiliza una variable global o de clase (global a la clase), la cual es definida en la parte de declaraciones de la clase. A continuación, en el Método principal se le asigna el valor cero a dicha variable ($x=0$); luego se llama o invoca al Método `cambiar()`, el cual también utiliza la variable `x`, asignándole el valor 1. Al regresar al Método principal, se imprime la variable `x`, cuyo valor es 1.

Esto significa que se utiliza una sola variable global o de clase (global a la clase), a la cual se le asigna el valor cero en el Método principal y después se le asigna 1 en el Método `cambiar()`. Al regresar al Método principal se imprime `x`, con el valor actual 1.

Nota: En el algoritmo de ayuda del punto anterior se utilizan este tipo de variables.



En el algoritmo de ayuda del punto anterior se utilizan este tipo de variables.

7.3.2 Variables locales

Las variables locales son las que se definen en cada método, las cuales sólo pueden utilizarse en el contexto del método en que fueron definidas. A continuación se presenta un algoritmo que muestra cómo usar una variable local:

```

Algoritmo VARIABLE LOCAL
Clase VarLocal
  1. Declaraciones de clase
    Variables
    x: Entero

```

```

2. Método principal()
  a. x = 0
  b. cambiar()
  c. Imprimir x
  d. Fin Método principal

3. Método cambiar()
  a. Declarar variables
     x: Entero
  b. x = 1
  c. Fin Método cambiar
Fin Clase VarLocal
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: VarLocal.java

Explicación:

Este algoritmo VARIABLE LOCAL tiene el método principal y un método subordinado llamado `cambiar()`. Se utiliza una variable global o de clase (global a la clase) `x`, definida en la parte de declaraciones de la clase (globales de la clase); a continuación (en el método principal) se le asigna el valor cero, luego se invoca al Método `cambiar()`, en donde se define una variable local con el nombre de `x`, a la cual se le asigna el valor 1. Al regresar al Método `principal`, se imprime la variable `x`, cuyo valor es cero. Revise el algoritmo y vea si esto es cierto... ¿Ya lo hizo? Si es cierto, ¿por qué?

Respuesta: Aunque las dos variables tienen el mismo nombre, son diferentes, ya que una se definió como global a la clase y la otra como local. Aun cuando la local definida en el Método `cambiar()` se llame igual que la global, para este método tiene mayor jerarquía la local, es decir, oculta a la global (de la clase) porque tiene el mismo nombre. En otras palabras, en el algoritmo se define una variable global o de clase `x` a la que se le asigna el valor cero (en el método principal), luego se llama al Método `cambiar()`, donde se define una variable local `x` y se le asigna el valor 1. Al regresar al método principal se imprime `x`; ¿cuál `x`? Obviamente la global, porque es la única de las dos `x` que se puede usar en el Método `principal`, que tiene el valor de cero.

Práctica: En este momento se recomienda ir al punto de ejercicios resueltos, estudiar algunos ejemplos y resolver otros de los ejercicios propuestos para aplicar métodos que no regresan valor, variables globales y locales.

7.3.3 Parámetros

Si analizamos los conceptos de variables de clase (globales) y locales, inferimos que cuando se usan variables locales, éstas son independientes de las globales y de las de otros métodos. En ocasiones puede ser necesario conectar una variable de clase (global) con una local para transmitir datos entre ambas, o bien conectar variables



Práctica: En este momento se recomienda ir al punto de ejercicios resueltos, estudiar algunos ejemplos y resolver otros de los ejercicios propuestos para aplicar métodos que no regresan valor, variables globales y locales.

locales de un método con variables locales de otro(s) método(s); esto es posible mediante el uso de parámetros, donde las variables fungen como tales. Existen parámetros por referencia y parámetros por valor.

7.3.3.1 Parámetro por referencia

El **parámetro por referencia** (o parámetro variable, en algunos lenguajes) es una variable local de un método que se define como parámetro en el encabezado de éste y sirve para conectarse con otra variable de otro método mediante el envío de su dirección, es decir, se conecta con la otra variable a través de su contenido; al llamarse o invocarse el método se establece la conexión, convirtiéndose en sinónimos. Esto significa que lo que le suceda a la variable local del método llamado le sucederá a la variable del método con la que fue conectada al hacer la llamada, porque utilizan la misma posición (dirección) de memoria. A continuación se presenta un algoritmo que muestra el uso de un parámetro por referencia:

```

Algoritmo PARAMETRO POR REFERENCIA
Clase ParametroPorReferencia
  1. Método principal()
    a. Declarar variables
       x: Entero
    b. x = 0
    c. Imprimir x
    d. cambiar(x)
    e. Imprimir x
    f. Fin Método principal

  2. Método cambiar(Ref y: Entero)
    a. y = 1
    b. Fin Método cambiar
Fin Clase ParametroPorReferencia
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: ParametroPorReferencia.java



Explicación:

En el `Método principal` se define la variable `x`, a la cual se le asigna el valor cero y se imprime; luego se llama al `Método cambiar()` y, dado que deseamos conectar esta variable con una variable local del método, entonces indicamos entre paréntesis dicha variable (`cambiar(x)`). En el encabezado del `Método cambiar` y especificado entre paréntesis se define la variable que fungirá como enlace con la variable de otro método; la palabra `Ref` indica que se está definiendo un parámetro por referencia (llamado tipo variable en algunos lenguajes como Turbo Pascal), y es el nombre de la variable que funge como parámetro (aquí se le llama parámetro formal); `Entero` es el tipo de dato de la variable parámetro. Nótese que `y` queda definida como una variable local, independientemente de que se definan otras en la parte de declaración de variables del método, es decir, que en un método es posible declarar variables en dos partes: una es la normal (la parte de declaración de



Se le llama parámetro formal al definido en el método que será llamado y parámetro actual al parámetro utilizado al hacer la llamada al método. Este concepto se aplica a todos los tipos de parámetros.

variables) y la otra es el encabezado del método, donde se definen las variables que fungirán como parámetros. Cuando se llama al método, las dos variables se conectan convirtiéndose en sinónimos, es decir que x e y son lo mismo y en la memoria las dos hacen referencia al mismo contenido; esto significa que lo que le suceda a y en el método le sucederá a x , de ahí que también se le conoce como parámetro tipo variable, ya que su valor puede cambiar.

Una vez que el control esté dentro del método, a y se le asigna el valor de 1, lo cual le sucede también a x . Al regresar al Método principal se imprime x , que tendrá el valor de 1.

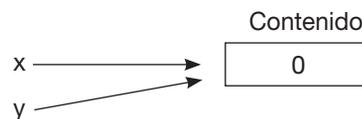
Nota: Se le llama parámetro formal al definido en el método que será llamado y parámetro actual al parámetro utilizado al hacer la llamada al método. Este concepto se aplica a todos los tipos de parámetros.

Esquemáticamente:

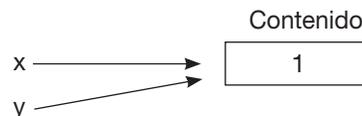
Se define la variable x en el Método principal, la cual tiene su contenido en la memoria, y se le coloca 0. Gráficamente:



Cuando se llama al Método `cambiar()`, se conectan x e y vía parámetro por referencia, lo que hace que sean sinónimos, es decir, que y utilice la misma dirección de memoria que x . Gráficamente:



Cuando a y se le coloca 1, ¿qué le pasa a x ?



A x le pasa lo mismo que le sucede a y ; es decir, a y se le coloca 1 e indirectamente le sucede lo mismo a x , porque manejan el mismo contenido en la memoria.

7.3.3.2 Parámetro por valor

El **parámetro por valor** es una variable local de un método que se define como parámetro en el encabezado de éste y sirve para conectarse con otra variable de otro método mediante el envío de su valor, en el momento de hacer la llamada del método, después de lo cual ya no hay relación. De ahí en adelante lo que le sucede a la variable parámetro no afectará a la variable del otro método, que sólo le envía su valor. A continuación se presenta un algoritmo que muestra cómo utilizar un parámetro por valor.

```

Algoritmo PARAMETRO POR VALOR
Clase ParametroPorValor
  1. Método principal()
    a. Declarar variables
       x: Entero
    b. x = 0
    c. Imprimir x
    d. cambiar(x)
    e. Imprimir x
    f. Fin Método principal

  2. Método cambiar(Val y: Entero)
    a. y = 1
    b. Fin Método cambiar
Fin Clase ParametroPorValor
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: ParametroPorValor.java



Explicación:

Si analizamos este algoritmo, podemos observar que la única diferencia con el anterior es que al definir la variable que funge como parámetro en el método se indica la palabra `Val`, lo que significa que es un parámetro por valor.

La diferencia con respecto al parámetro por referencia es que, al llamar al método, el valor del parámetro actual `x` se le asigna al parámetro `y` (del método) nada más, es decir, que de ahí en adelante lo que le suceda a `y` es independiente de `x`. En otras palabras, si `x` vale 0, al llamar al Método `cambiar` a `y` se le asigna 0 (el mismo valor que `x`); en la memoria `x` tiene su propio contenido y `y` tiene el suyo.

En el ejemplo anterior, a la variable `x` se le asigna 0; luego se llama `cambiar(x)`. En ese momento a `y` se le asigna el valor de `x`, es decir, 0. Ya en el método, a `y` se le asigna 1 y se regresa al Método `principal` a imprimir `x`, la cual tiene el valor de 0. Es decir, al ir al Método `cambiar` no le afectó lo que se le asignó a `y`. Por ello, este tipo de parámetro funge sólo como un valor que se envía al método.

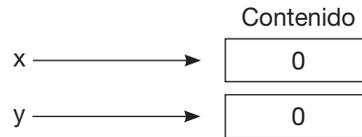
Cabe hacer la aclaración de que se ha usado un solo parámetro, pero pueden usarse más. Los parámetros indicados tanto al definir el método como al llamarlo desde el método principal deben coincidir en número y tipo.

Esquemáticamente:

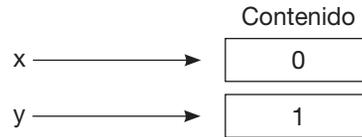
Se define la variable `x` en el Método `principal`, la cual tiene su contenido en la memoria, y se le coloca 0. Gráficamente:



Cuando se llama al Método `cambiar`, se conectan `x` e `y` vía parámetro por valor, lo que hace que el valor de `x` se le asigne a `y`. De ahí en adelante `x` e `y` no tienen ninguna relación. Gráficamente:



Es decir, `x` tiene su propio contenido en la memoria y la variable `y` tiene el propio. Cuando a `y` se le coloca 1, ¿qué le pasa a `x`?



A `x` no le pasa nada; es decir, a `y` se le coloca 1 en su contenido y `x` ni cuenta se da, porque tiene su propio contenido independiente de `y`.

Práctica: En este momento se recomienda ir al punto de ejercicios resueltos, estudiar algunos ejemplos y resolver otros de los ejercicios propuestos para aplicar parámetros por valor y por referencia.



Práctica: En este momento se recomienda ir al punto de ejercicios resueltos, estudiar algunos ejemplos y resolver otros de los ejercicios propuestos para aplicar parámetros por valor y por referencia.

7.4 Funciones estándar

Las funciones estándar son funciones proporcionadas por cualquier lenguaje de programación de alto nivel. Dichas funciones se agrupan como aritméticas, cadena de caracteres o alfabéticas, archivos, graficación, objetos, especiales, etcétera. En los capítulos anteriores hemos estudiado algunas funciones estándar, mientras que en éste y en otros capítulos se estudiarán otras funciones: por ejemplo, las funciones para archivos se presentan en el capítulo de registros y archivos.

7.4.1 Funciones cadena de caracteres

Carácter

La función **Carácter** proporciona el carácter representado por un valor entero en sistema numérico decimal, de acuerdo al código ASCII.

Formato:

`Carácter(x)`

En donde:

`Carácter`

Es la palabra que identifica a la función.

`x`

Es el parámetro que indica el valor entero en sistema numérico decimal, que se se tomará como base para determinar el carácter correspondiente. Debe ser de tipo entero. El tipo de dato del resultado es `Carácter`.

Ejemplo:

```
car = Carácter(65)
```

`car` toma el valor `A`; es decir, la letra `A` corresponde al entero decimal 65 en el código ASCII.

Ordinal

La función **Ordinal** proporciona el valor entero decimal que se utiliza para representar un carácter, de acuerdo al código ASCII.

Formato:

```
Ordinal(x)
```

En donde:

Ordinal

Es la palabra que identifica a la función.

x

Es el parámetro que indica el carácter al cual se determinará el valor entero decimal que se utiliza para representarlo. Debe ser de tipo `Carácter`. El tipo de dato del resultado es entero.

Ejemplo:

```
num = Ordinal('A')
```

`num` toma el valor 65; es decir, la letra `A` corresponde al entero decimal 65 en el código ASCII.

Concatenar

La función **Concatenar** permite concatenar o “unir” cadenas de caracteres.

Ejemplo:

Si tenemos las variables:

```
nombrePila = "María"  
apellidoPaterno = "López"  
apellidoMaterno = "Acosta"
```

Podemos concatenar de la forma:

```
nombrePersona = Concatenar(nombrePila +  
    apellidoPaterno + apellidoMaterno)
```

o bien, de la forma siguiente:

```
nombrePersona = nombrePila + apellidoPaterno + apellidoMaterno
```

En donde:

Concatenar

Es una función que realiza la concatenación. Entonces `NombrePersona` tendrá el contenido “María López Acosta”

Subcadena

La función **Subcadena** permite obtener una copia de una subcadena (una parte de una cadena).

Formato:`Subcadena(cadenaFuente, inicio, tamaño)`*En donde:*

Subcadena

Identifica la función.

cadenaFuente

Es la variable tipo cadena de caracteres de la cual se obtendrá la subcadena.

inicio

Es un número, variable o expresión de tipo entero que indica la posición desde la cual se empezará a obtener la subcadena.

tamaño

Es un número entero, variable o expresión de tipo entero que indica la cantidad de caracteres que serán copiados.

Ejemplo:

Si tenemos la cadena `nombrePersona` con el valor “María Lopez Acosta” y se aplica lo siguiente:

`a = Subcadena(nombrePersona, 7, 5)`

`a` tomará el valor “Lopez”. Porque se copian 5 caracteres desde el 7.

Borrar

La función **Borrar** borra o quita una subcadena de una cadena.

Formato:`Borrar(cadenaFuente, inicio, tamaño)`*En donde:*

Borrar

Identifica la función.

cadenaFuente

Es la variable tipo cadena de caracteres de la cual se borrará.

inicio

Es un número, variable o expresión de tipo entero que indica la posición desde donde se empezará a borrar.

tamaño Es un número, variable o expresión de tipo entero que indica la cantidad de caracteres que serán borrados.

Ejemplo:

Si tenemos:

```
comentario = "EL FUTBOL SOCCER ES POCO INTERESANTE"
```

y le aplicamos lo siguiente:

```
Borrar(comentario, 21, 4)
```

tendremos :

```
comentario = "EL FUTBOL SOCCER ES INTERESANTE"
porque se borra a partir del carácter 21 (desde la P) y se borran cuatro caracteres (POCO).
```

Insertar

La función **Insertar** inserta caracteres en una cadena.

Formato:

```
Insertar(caracteres, cadenaObjeto, inicio)
```

En donde:

Insertar	Identifica la función.
caracteres	Es una cadena de caracteres, ya sea constante o variable.
cadenaObjeto	Es la variable tipo cadena de caracteres en la cual se insertarán los caracteres.
inicio	Es un número, variable o expresión de tipo entero que indica la posición desde la cual empezará la inserción de los caracteres.

Ejemplo:

Si tenemos:

```
comentario = "EL FUTBOL SOCCER ES INTERESANTE"
```

y le aplicamos:

```
Insertar("POCO", comentario, 21)
```

Quedará:

```
comentario ="EL FUTBOL SOCCER ES POCO INTERESANTE"
```

Longitud

La función **Longitud** proporciona la longitud de una cadena.

Formato:

Longitud(cadenaFuente)

En donde:

Longitud	Identifica la función.
cadenaFuente	Es la variable tipo cadena de caracteres de la cual se proporcionará su longitud en número de caracteres.

Ejemplo:

Si tenemos:

```
comentario = "EL FUTBOL SOCCER ES INTERESANTE"
```

y le aplicamos:

```
n = Longitud(comentario)
```

n tomará el valor 31 porque comentario tiene 31 caracteres.

Posición

La función **Posición** busca una cadena dentro de otra y proporciona la posición donde inicia.

Formato:

Posición(cadenaFuente, cadenaBuscada)

En donde:

Posición	Identifica la función.
cadenaFuente	Es la variable tipo cadena de caracteres en la cual se buscará la otra cadena.
cadenaBuscada	Es la cadena de caracteres que será buscada.

Ejemplo:

Si tenemos:

```
comentario = "EL FUTBOL SOCCER ES POCO INTERESANTE"
```

y le aplicamos:

```
a = Posición(comentario, "POCO")
```

a tomará el valor de 21, donde empieza la cadena buscada.

Cadena

La función **Cadena** convierte un valor numérico real o entero en cadena de caracteres.

Formato:

Cadena (numero, cadenaObjeto)

En donde:

Cadena	Identifica la función.
numero	Es una variable o número entero o real.
cadenaObjeto	Es una variable tipo cadena de caracteres en la cual se colocará el valor convertido.

Ejemplo:

Si tenemos n con el valor 278, y le aplicamos:

Cadena (n, n2)

n2 toma el valor “278”, es decir, transforma el número entero en cadena de caracteres.

Nota: La variable numérica tiene la opción de formatearse para tipos reales. Por ejemplo, si n = 278.55 se convierte de la siguiente forma:

Cadena (n:10:2, n2)

Donde se está formateando el contenido de la variable n, a 10 posiciones de salida con 2 decimales. Entonces el valor de n2 será “278.55”.

Valor

La función **Valor** convierte una cadena de caracteres (string) a número entero o real.

Formato:

Valor (cadenaFuente, numero)

En donde:

Valor	Identifica la función.
cadenaFuente	Es la cadena de caracteres que se convertirá a numérico.
numero	Es una variable de tipo entero o real, en la cual se asignará el valor de la cadena de caracteres pero en formato numérico.

Ejemplo:

Si tenemos n2 con el valor “278.55”, y la aplicamos:

Valor (n2, n)

n toma el valor 278.55.

Práctica: En este momento se recomienda ir al punto de ejercicios resueltos, estudiar algunos ejemplos y resolver otros de los ejercicios propuestos para aplicar las funciones explicadas en este punto.

7.4.2 Validación de la entrada de datos

Las funciones del punto anterior son útiles para algunas operaciones, tales como puede ser la validación de datos. Por ejemplo, si debemos leer un número entero podríamos hacerlo de la siguiente manera:

A continuación se presenta el algoritmo de la solución:

```

Algoritmo LEER IMPRIMIR N
Clase LeeImprimeN
  1. Método principal()
    a. Declarar variables
       n: Entero
    b. Solicitar número entero
    c. Leer n
    d. Imprimir n
    e. Fin Método principal
  Fin Clase LeeImprimeN
Fin

```



En la zona de descarga de la Web del libro está disponible:

Programa en Java: LeelImprimeN.java

Explicación:

Se define la variable *n*, se solicita un número entero, se lee en *n*, se imprime el número leído, fin.

O bien, podríamos leer ese número validando la lectura (entrada de datos). ¿Qué significa esto? Que cuando se teclea el dato, puede tratarse de una letra o símbolo especial, lo cual sería un error. La idea es que el algoritmo valide la entrada, es decir, detecte si hay error en la entrada del dato y que permita recuperarse del error, o sea, no quedarse tirado con el error. A continuación se presenta un algoritmo que permite leer el número entero y, si hay error, volver a leer hasta que se teclee sin error:

```

Algoritmo LEER IMPRIMIR N VALIDANDO
Clase LeeImprimeValidando
  1. Método principal()
    a. Declarar variables
       n, i, bandera: Entero
       numero: Cadena
       num: Carácter
    b. do
      1. do
        a. bandera = 1
        b. Solicitar número
        c. Leer numero
        d. for i=0; i<=Longitud(numero)-1; i++
          1. num = SubCadena(numero,i,1)

```

```

                2. if NOT(((num>='0')AND(num<='9'))OR
                    (num=='-')) then
                    a. bandera = 0
                3. endif
                4. if (num=='-')AND(i>0) then
                    a. bandera = 0
                5. endif
            e. endfor
        2. while bandera != 1
        3. Valor(numero,n)
        4. if (n < -32768)OR(n > 32767) then
            a. bandera = 0
        5. endif
    c. while bandera != 1
    d. Imprimir n
    e. Fin Método principal
Fin Clase LeeImprimeValidando
Fin

```

En la zona de descarga de la Web del libro está disponible:
Programa en Java: LeelprimeValidando.java



Explicación:

Entra en un ciclo repetitivo, luego en otro ciclo donde la bandera se “enciende” colocándole 1. Enseguida se solicita y lee el número en una variable tipo Cadena, luego entra en un ciclo que revisa cada uno de los caracteres leídos; si alguno(s) no es un dígito entre 0 y 9 o no es el signo de menos, entonces se “apaga” la bandera, colocándole 0. Al llegar al while del ciclo, se evalúa si la bandera no está “apagada” (diferente de 1); si es así, se devuelve a entrar al ciclo para volver a leer el número; esto se repetirá mientras que la bandera no se “apague”, es decir, hasta que se hayan leído sólo números. Al salir del while lo convierte a numérico para evaluar si está en el rango del tipo Entero (entre -32768 y 32767); si no, regresa al primer do para volver entrar al segundo do y leer de nuevo el número, hasta que lea un número correcto, se sale del ciclo y lo imprime.

Observación: La variable bandera pudo haberse definido de tipo Boolean y usar los valores True (Verdadero) y False (Falso), en lugar de 1 y 0.

Práctica: En este momento se recomienda ir al punto de ejercicios resueltos, estudiar algunos ejemplos y resolver otros de los ejercicios propuestos para aplicar validación de la entrada de datos.

7.4.3 Funciones especiales

En este punto se estudiarán algunas otras funciones especiales de índole general.

Random

La función **Random** proporciona un número aleatorio, es decir, un número generado al azar; realmente es un número pseudoaleatorio porque es generado por una función matemática, por lo cual no es totalmente aleatorio.



Observación: La variable bandera pudo haberse definido de tipo Boolean y usar los valores True (Verdadero) y False (Falso), en lugar de 1 y 0.

Práctica: En este momento se recomienda ir al punto de ejercicios resueltos, estudiar algunos ejemplos y resolver otros de los ejercicios propuestos para aplicar validación de la entrada de datos.

Formato:

```
Random[ (n) ]
```

En donde:

Random

Identifica la función.

n

Es opcional:

Cuando no está presente se genera un valor entre 0 y 1
 $0 \leq \text{valor} < 1$. El valor es tipo Real.

Cuando está presente se genera un valor entre 0 y N
 $0 \leq \text{valor} < n$. El valor es tipo Entero.

Ejemplos:

```
num = Random(20)
```

num tomará un valor entre 0 y 20, es decir, cualquiera desde 0 al 19.

```
num = Random
```

num tomará un valor entre 0 y 1, es decir, cualquiera entre 0 y 0.9999999....

IniciaRandom

La función **IniciaRandom** permite iniciar la semilla del generador de números aleatorios. Los números aleatorios (pseudo) se generan a partir del valor que tiene la semilla, el cual se almacena en la variable Randseed. Cada vez que se ejecuta esta función Randseed toma un nuevo valor generado al azar. Formato:

```
IniciaRandom
```

Sonido

La función **Sonido** genera un sonido en el altavoz de la computadora de acuerdo a la frecuencia que se le indique hasta que se encuentre con la función NoSonido.

Formato:

```
Sonido(frecuencia)
```

En donde:

Sonido

Es el nombre de la función.

frecuencia

Es un número entero que indica la frecuencia del sonido.

NoSonido

La función **NoSonido** detiene la generación de sonido en el altavoz.

Formato:

```
NoSonido
```

Detener

La función **Detener** detiene el funcionamiento de la computadora la cantidad de milisegundos que se le indique.

Formato:

Detener(tiempo)

En donde:

tiempo es un número entero que indica la cantidad de milisegundos que se detendrá el funcionamiento de la computadora.

Ejemplo:

```
Sonido(250) /* Se activa el sonido con frecuencia 250 */
Detener(500) /* Se detiene el funcionamiento 500 milisegundos,
              antes de ir a la sig. Instrucción */
NoSonido /* Se detiene el sonido */
```

7.5 Métodos que regresan valor

Son métodos equivalentes a las funciones definidas por el usuario en el contexto de la programación estructurada. Este tipo de métodos los puede definir el programador con el propósito de ejecutar alguna función específica y por lo general se usan cuando se trata de hacer algún cálculo que será requerido en varias ocasiones en el método principal o en algún otro método del algoritmo. El método que devuelve valor es casi igual en todos los aspectos al método (void) que no regresa valor, excepto que el método que regresa valor devuelve un valor de un tipo de datos simple a su punto de referencia y es utilizado como si fuera una variable de un tipo simple de datos.

El nombre del método puede estar seguido por uno o más parámetros encerrados entre paréntesis. Por lo general transfieren datos a parámetros por valor.

Definición de métodos que devuelven valor

Para definir métodos que devuelven valor se utiliza el siguiente formato:

```
2. Método nomFuncion(parametro): Tipo de dato
  a. Declarar
      Constantes
      Tipos
      Variables
  b. Acción1
  c. Acción2
  d. AcciónN
  e. return valor
  f. Fin Método nomFuncion
```

En donde:

Método Es la palabra que indica que se está definiendo un método.
 nomFuncion Es el nombre del método, el cual fungirá como cualquier variable de tipo simple.

parametro	Es la variable que fungirá como parámetro. Es la que recibe el valor que servirá como base para hacer los cálculos en la función (puede haber más de uno).
Tipo de dato	Es el tipo de dato del resultado que regresará el método.
return	Indica que el método está regresando el valor obtenido o calculado.
valor	Es el valor que regresa el método.

Referencia de métodos que devuelven valor

Para llamar o invocar a un método que regresa valor, se indica el nombre de éste poniendo entre paréntesis el parámetro que se envía, utilizándose como cualquier otra variable de tipo simple. Por ejemplo:

```
x = 5
n = nomFuncion(x)
```

x toma el valor 5; luego se llama al método `nomFuncion`, enviándole como parámetro x. El método utiliza el valor de x y hace algún cálculo devolviendo un valor, el cual es colocado en n.

Ejemplo:

Elaborar un algoritmo que evalúe la función:

$$f = \frac{X!}{Y! (X - Y)!}$$

de la cual se tienen los datos X e Y, mismos que deberán solicitarse y leerse. Factorial de X se divide entre el Factorial de Y multiplicado por el factorial de X-Y.

A continuación se presenta el algoritmo de la solución:

```
Algoritmo EVALUA EL VALOR DE F
Clase EvaluaF
1. Método principal()
  a. Declarar variables
    x, y: Entero
    f: Real
  b. Solicitar X, Y
  c. Leer x, y
  d. f = factorial(x) / (factorial(y) * factorial(x-y))
  e. Imprimir f
  f. Fin Método principal

2. Método factorial(Val n: Entero): Entero
  a. Declarar variables
    i, fact: Entero
```

```

b. if n == 0 then
    1. fact = 1
c. else
    1. fact = 1
    2. for i=n; i>=1; i--
        a. fact = fact * i
    3. endfor
d. endif
e. return fact
f. Fin Método factorial
Fin Clase EvaluaF
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: EvaluaF.java



Explicación:

Dentro de la clase se definen dos métodos: uno es el método principal; el otro método calcula el factorial de un número, el cual es recibido en n como parámetro por valor:

Se tiene el Método principal:

1. Método principal()
 - a. Se declaran las variables:
 x, y : Entero.
 f : Real.
 - b. Se solicitan X, Y .
 - c. Se leen en x, y .
 - d. Calcula $f = \text{factorial}(x) / (\text{factorial}(y) * \text{factorial}(x-y))$.
 Llama al Método factorial tres veces:
 $\text{factorial}(x)$ Enviando x como parámetro.
 $\text{factorial}(y)$ Enviando y como parámetro.
 $\text{factorial}(x-y)$ Enviando el resultado de $x-y$ como parámetro.
 Cada vez factorial devuelve el valor correspondiente al factorial calculado tomando como base el valor del parámetro.
 - e. Imprime f .
 - f. Fin del Método principal.

2. Método factorial(Val n : Entero): Entero.

Define n como parámetro donde recibirá el valor que se le envíe como parámetro cada vez que sea llamado desde el método principal. Devuelve un valor de tipo entero.

- a. Se declaran las variables:
 i, fact : Entero.
- b. Si $n == 0$ entonces:
 1. Coloca 1 en fact ; es decir, 1 es el factorial de 0.

- c. Si no:
 1. Inicia `fact` en 1.
 2. Inicia ciclo for desde `i = n` hasta 1 con decrementos de -1.
 - A. Calcula `fact = fact * i`.
 3. Fin del for.
- d. Fin del if.
- e. Regresa el valor que calculó en `fact`.
- f. Fin del Método `factorial`. Luego se tiene el fin de la clase y el fin del algoritmo.

Práctica: En este momento se recomienda ir al punto de ejercicios resueltos, estudiar algunos ejemplos y resolver otros de los ejercicios propuestos para aplicar métodos que regresan valor.

7.6 Ejercicios resueltos

Ejercicio 7.6.1

Elaborar un algoritmo que ofrezca un menú de opciones mediante el cual se pueda escoger calcular el área de alguna de las figuras geométricas: triángulo, cuadrado, rectángulo y círculo. Una vez seleccionada la opción, que llame a un método que permita solicitar los datos necesarios, leerlos, hacer el cálculo correspondiente e imprimirlo.

$$\text{Área de triángulo} = \frac{\text{BASE X ALTURA}}{2}$$

$$\text{Área de cuadrado} = \text{LADO}^2$$

$$\text{Área de círculo} = \pi r^2$$

$$\text{Área de rectángulo} = \text{BASE X ALTURA}$$

Debe ofrecer el siguiente menú de opciones, donde está solicitando la opción deseada:

AREAS FIGURAS GEOMETRICAS:
1. TRIANGULO
2. CUADRADO
3. RECTANGULO
4. CIRCULO
5. FIN
ESCOGER OPCIÓN

Usar variables locales.

A continuación se tiene el algoritmo de la solución:

(Primero hágalo usted; después compare la solución)



Práctica: En este momento se recomienda ir al punto de ejercicios resueltos, estudiar algunos ejemplos y resolver otros de los ejercicios propuestos para aplicar métodos que regresan valor.

Algoritmo AREAS

Clase AreasFigGeometricas

1. Método principal()

- a. Declarar variables
opcion: Entero
- b. do
 1. Imprimir MENU

AREAS FIGURAS GEOMETRICAS
1. TRIANGULO
2. CUADRADO
3. RECTANGULO
4. CIRCULO
5. FIN
ESCOGER OPCIÓN

2. Leer opcion
3. switch opcion
 - 1: calcularAreaTriangulo()
 - 2: calcularAreaCuadrado()
 - 3: calcularAreaRectangulo()
 - 4: calcularAreaCirculo()
4. endswitch

- c. while opcion != 5
- d. Fin Método principal

2. Método calcularAreaTriangulo()

- a. Declarar variables
base, altura, areaTria: Real
- b. Solicitar Base, Altura
- c. Leer base, altura
- d. $areaTria = (base * altura) / 2$
- e. Imprimir areaTria
- f. Fin Método calcularAreaTriangulo

3. Método calcularAreaCuadrado()

- a. Declarar variables
lado, areaCuad: Real
- b. Solicitar Lado
- c. Leer lado
- d. $areaCuad = Potencia(lado, 2)$
- e. Imprimir areaCuad
- f. Fin Método calcularAreaCuadrado

```

4. Método calcularAreaRectangulo()
  a. Declarar variables
     areaRec, base, altura: Real
  b. Solicitar Base, Altura
  c. Leer base, altura
  d. areaRec = base * altura
  e. Imprimir areaRec
  f. Fin Método calcularAreaRectangulo

5. Método calcularAreaCirculo()
  a. Declarar
     Constantes
     PI = 3.145926536
     Variables
     areaCirc, radio: Real
  b. Solicitar Radio
  c. Leer radio
  d. areaCirc = PI * Potencia(radio,2)
  e. Imprimir areaCirc
  f. Fin Método calcularAreaCirculo
Fin Clase AreasFigGeometricas
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: AreasFigGeometricas.java

Explicación:

Dentro de la clase se definen cinco métodos: uno es el método principal, y los otros métodos calculan el área de un triángulo, cuadrado, rectángulo y círculo, respectivamente:

Se tiene el Método principal:

1. Método principal()
 - a. Se declara la variable:
 - opcion para leer la opción que escoga.
 - b. Inicia ciclo do.
 1. Imprimir MENU y solicita la opción.
 2. Lee en opcion.
 3. switch opcion:
 - Si opcion es 1: Llama al Método calcularAreaTriangulo().
 - Si opcion es 2: Llama al Método calcularAreaCuadrado().
 - Si opcion es 3: Llama al Método calcularAreaRectangulo().
 - Si opcion es 4: Llama al Método calcularAreaCirculo().
 4. Fin del switch.
 - c. Fin ciclo while opcion != 5. Va al do.
 - d. Fin Método principal.

2. Método `calcularAreaTriangulo()`
 - a. Se declaran las variables.
 - b. Solicita Base y Altura.
 - c. Lee base, altura.
 - d. Calcula $\text{areaTria} = (\text{base} * \text{altura}) / 2$.
 - e. Imprime `areaTria`.
 - f. Fin Método `calcularAreaTriangulo`.
3. Método `calcularAreaCuadrado()`
 - a. Se declaran las variables.
 - b. Solicita Lado.
 - c. Lee lado.
 - d. Calcula $\text{areaCuad} = \text{Potencia}(\text{lado}, 2)$.
 - e. Imprime `areaCuad`.
 - f. Fin Método `calcularAreaCuadrado`.
4. Método `calcularAreaRectangulo()`
 - a. Se declaran las variables.
 - b. Solicita Base y Altura.
 - c. Lee base, altura.
 - d. Calcula $\text{areaRec} = \text{base} * \text{altura}$.
 - e. Imprime `areaRec`.
 - f. Fin Método `calcularAreaRectangulo`.
5. Método `calcularAreaCirculo()`
 - a. Se declaran la constante y las variables.
 - b. Solicita Radio.
 - c. Lee radio.
 - d. Calcula $\text{areaCirc} = \text{PI} * \text{Potencia}(\text{radio}, 2)$.
 - e. Imprime `areaCirc`.
 - f. Fin Método `calcularAreaCirculo`. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 7.6.2

Elaborar un algoritmo que permita leer dos números de tipo real en el método principal, que en un método los intercambie vía parámetros por referencia y los imprima en el método principal. Usar parámetros por referencia.

A continuación se tiene el algoritmo de la solución:

(Primero hágalo usted; después compare la solución)

```
Algoritmo INTERCAMBIA NUMEROS
Clase IntercambiaNumeros
1. Método principal()
  a. Declarar variables
    a, b: Real
  b. Solicitar número 1, número 2
```

```

    c. Leer a, b
    d. Imprimir a, b
    e. intercambiar(a,b)
    f. Imprimir a, b
    g. Fin Método principal

2. Método intercambiar(Ref a1, b1: Real)
  a. Declarar variables
     auxiliar: Real
  b. auxiliar = a1
  c. a1 = b1
  d. b1 = auxiliar
  e. Fin Método intercambiar
Fin Clase IntercambiaNumeros
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: IntercambiaNumeros.java

Explicación:

Dentro de la clase IntercambiaNumeros se definen dos métodos:

1. Método principal()
 - a. Se declaran las variables.
 - b. Solicita número 1 y número 2.
 - c. Lee a, b.
 - d. Imprime a, b.
 - e. Llama intercambiar(a,b) enviando a y b como parámetros, que se conectarán con a1 y b1.
 - f. Imprime a, b.
 - g. Fin Método principal.
2. Método intercambiar(Ref a1, b1: Real)

a1 y b1 se definen como parámetros por referencia, que se conectarán con a y b, respectivamente, al ser llamado este método.

 - a. Se declara la variable auxiliar.
 - b. Se coloca a1 en auxiliar.
 - c. Se coloca b1 en a1.
 - d. Se coloca auxiliar en b1.
 - e. Fin Método intercambiar. Luego se tiene el fin de la clase y el fin del algoritmo.

Ejercicio 7.6.3

Elaborar un algoritmo que permita leer tres números de tipo entero e imprima el mayor, utilizando un método para leer los números, otro método para obtener y devolver el mayor y un método para imprimir el mayor. Usar parámetros por valor y por referencia.

A continuación se tiene el algoritmo de la solución:
(Primero hágalo usted; después compare la solución)

```

Algoritmo MAYOR 3 NUMEROS CON METODOS
Clase Mayor3Numeros4
1. Método principal()
  a. Declarar variables
    a, b, c, may: Entero
  b. leerNumeros(a, b, c)
  c. may = obtenerMayor(a, b, c)
  d. imprimirMayor(may)
  e. Fin Método principal

2. Método leerNumeros(Ref n1,n2,n3: Entero)
  a. Solicitar número 1, número 2, número 3
  b. Leer n1, n2, n3
  c. Fin Método leerNumeros

3. Método obtenerMayor(Val num1,num2,num3: Entero): Entero
  a. Declarar variables
    mayor: Entero
  b. mayor = num1
  c. if num2 > mayor then
    1. mayor = num2
  d. endif
  e. if num3 > mayor then
    1. mayor = num3
  f. endif
  g. return mayor
  h. Fin Método obtenerMayor

4. Módulo imprimirMayor(Val m: Entero)
  a. Imprimir m, "ES EL MAYOR"
  b. Fin Método imprimirMayor
Fin Clase Mayor3Numeros4
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Mayor3Numeros4.java



Explicación:

Dentro de la clase Mayor3Numeros4 se definen los métodos:

1. Método principal()
 - a. Se declaran las variables.
 - b. Llama leerNumeros(a, b, c) enviando a, b y c como parámetros.
 - c. Coloca en may lo que regrese obtenerMayor(a, b, c) enviando a, b y c como parámetros.
 - d. Llama imprimirMayor(may) enviando may como parámetro.
 - e. Fin Método principal.

2. Método leerNumeros (Ref n1, n2, n3: Entero).
n1, n2 y n3 se definen como parámetros por referencia; se conectarán con a, b y c.
 - a. Solicita número 1, número 2 y número 3.
 - b. Lee n1, n2, n3.
 - c. Fin Método leerNumeros.
3. Método obtenerMayor (Val num1, num2, num3: Entero): Entero.
num1, num2 y num3 se declaran como parámetros por valor, que se conectarán con a, b y c.
 - a. Se declara la variable mayor.
 - b. Se coloca num1 en mayor.
 - c. Si num2 > mayor entonces:
 1. Se coloca num2 en mayor.
 - d. Fin del if.
 - e. Si num3 > mayor entonces:
 1. Se coloca num3 en mayor.
 - f. Fin del if.
 - g. return mayor.
 - h. Fin Método obtenerMayor.
4. Método imprimirMayor (Val m: Entero).
m se declara como parámetro por valor, que se conecta con may.
 - a. Imprime m, "ES EL MAYOR".
 - b. Fin Método imprimirMayor. Luego se tiene el fin de la clase y el fin del algoritmo.

Tabla 7.1: ejercicios resueltos disponibles en la zona de descarga del capítulo 7 de la Web del libro.

Ejercicio	Descripción
Ejercicio 7.6.4	Realiza cálculos con medidas estadísticas
Ejercicio 7.6.5	Eleva a una potencia en un método
Ejercicio 7.6.6	Calcula la suma de raíces cuadradas en un método
Ejercicio 7.6.7	Calcula la suma de cuadrados en un método
Ejercicio 7.6.8	Calcula la media de un arreglo con métodos
Ejercicio 7.6.9	Eleva a potencias
Ejercicio 7.6.10	Obtiene el mayor y menor de una matriz de números
Ejercicio 7.6.11	Determina si una matriz es simétrica o no
Ejercicio 7.6.12	Obtiene el dígito verificador de un número
Ejercicio 7.6.13	Indica si una frase es un palíndromo
Ejercicio 7.6.14	Dice fecha con letras
Ejercicio 7.6.15	Realiza validación en entrada de datos
Ejercicio 7.6.16	Indica cuántas veces se usó cada vocal
Ejercicio 7.6.17	Separa una frase en palabras
Ejercicio 7.6.18	Separa una frase en palabras e indica la mayor
Ejercicio 7.6.19	Genera números aleatorios

En estos últimos algoritmos se han elaborado diversos métodos que permiten hacer validaciones. Es conveniente que el programador las utilice cotidianamente en los algoritmos que desarrolle, porque son indispensables para que los algoritmos tengan una mejor calidad.



En estos últimos algoritmos se han elaborado diversos métodos que permiten hacer validaciones. Es conveniente que el programador las utilice cotidianamente en los algoritmos que desarrolle, porque son indispensables para que los algoritmos tengan una mejor calidad.

7.7 Ejercicios propuestos

1. Elaborar un algoritmo que proporcione un menú que permita escoger una de las funciones siguientes: tangente, cotangente, secante y cosecante. El cálculo de cada una es de la forma siguiente:

$$\text{Tangente}(X) = \frac{\text{Seno}(X)}{\text{Coseno}(X)}$$

$$\text{Cotangente}(X) = \frac{\text{Coseno}(X)}{\text{Seno}(X)}$$

$$\text{Secante}(X) = \frac{1}{\text{Coseno}(X)}$$

$$\text{Cosecante}(X) = \frac{1}{\text{Seno}(X)}$$

2. Elaborar un algoritmo que proporcione un menú donde ofrezca hacer conversiones entre metros, yardas, pulgadas y pies. Si escoge metros, en un método debe leer la cantidad N de metros e imprimir una tabla de equivalencias a yardas, pulgadas y pies, desde 1 metro hasta N metros de uno en uno. Equivalencias: 1 pie = 12 pulgadas, 1 yarda = 3 pies, 1 pulgada = 2.54 cm, 1 metro = 100 cm. Se debe imprimir la tabla siguiente:

METROS	CONVERSIONES		
	YARDAS	PULGADAS	PIES
1	9999.99	9999.99	9999.99
2	9999.99	9999.99	9999.99
-	-	-	-
N	9999.99	9999.99	9999.99

Y así si escoge pies, pulgadas o yardas se hace lo propio para cada una de ellas.

3. Similar al anterior, sólo que se lee un valor inicial y un valor final. Por ejemplo, si los valores leídos son 100 y 200, las conversiones se harán desde 100 hasta 200 de uno en uno.
4. Elaborar un algoritmo que ofrezca un menú que permita escoger la impresión de las tablas de multiplicar, dividir, sumar y restar, y dentro de cada opción que permita escoger las del 1, 2, 3, 4, 5, 6, 7, 8, 9 y 10.

5. Elaborar un algoritmo que defina tres matrices A, B y S de 7×7 , que lea números enteros para A y B, obtenga S multiplicando los elementos $A(1,1)$ y $B(1,1)$ y lo coloque en el $S(1,1)$, y así sucesivamente. Al final debe imprimir las tres matrices. Utilizar un método para leer y otro para imprimir.
6. El sistema monetario mexicano incluye, entre billetes y monedas, las denominaciones siguientes: 500, 200, 100, 50, 20, 10, 5, 2, 1 pesos y 50, 20, 10, 5 centavos. Hacer un algoritmo que lea una cantidad y que imprima cuántos billetes y monedas y de qué denominaciones se necesitan para cubrir dicha cantidad. Por ejemplo, la cantidad 577.80 generaría: 5 billetes de 100, 1 de 50, 1 de 20, 1 de 5, 1 de 2 pesos, una moneda de 50 centavos, 1 de 20, 1 de 10. Nota: en lugar de 5 de 100, podrían ser dos de 200 y uno de 100, o uno de 500.
7. Igual al problema 5.1.2.6 del capítulo 5, que calcula el sueldo a empleados, y al anterior, pero que al final presente una distribución de efectivo donde indique cuántos billetes y monedas y de qué denominaciones se requieren para cubrir el total por pagar.
8. Según el teorema de Pitágoras el cuadrado de la hipotenusa es igual a la suma del cuadrado de los catetos: $(C^2 = A^2 + B^2)$.
Utilizando este concepto es posible conocer de qué tipo es un triángulo:
Si $C^2 = A^2 + B^2$ es un triángulo rectángulo.
Si $C^2 < A^2 + B^2$ es un triángulo acutángulo.
Si $C^2 > A^2 + B^2$ es un triángulo obtusángulo.
Elaborar un algoritmo que permita leer los dos catetos y la hipotenusa e imprima el tipo de triángulo que es. Usar un método que reciba los catetos y devuelva el tipo de triángulo que es.
9. Elaborar un algoritmo que imprima las potencias de 2 que no excedan 8500, utilizando un método que calcule la potencia de cada número.
10. Elaborar un algoritmo que lea 20 números enteros y positivos y que para cada uno imprima si es par o impar. Utilizar un método que determine si es par o impar.
11. Elaborar un algoritmo que lea un valor X (entero y positivo); y que imprima el valor de Y.

$$Y = \frac{X^1}{1!} + \frac{X^2}{2!} + \frac{X^3}{3!} + \dots + \frac{X^x}{X!}$$

Usar un método que eleve X a la potencia y otro método que calcule el factorial en cada ocasión.

12. Elaborar un algoritmo que lea números enteros en un arreglo de 5×7 , que lo imprima e imprima la sumatoria por renglones y por columnas utilizando un arreglo para el cálculo de las sumatorias de los 5 renglones y lo propio para las 7 columnas. Utilice un método para hacer la lectura, otro para los cálculos y otro para imprimir éstos, dirigidos por un método de control que es el principal.
13. Elaborar un algoritmo que permita leer diez valores para A, B y C, calcular los valores de X1 y X2 para cada tripleta de valores e imprimir A, B, C, X1 y X2. Utilizar un método para hacer los cálculos. Utilizar la ecuación cuadrática:

$$X = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

14. Elaborar un algoritmo que lea dos números y llame un método que calcule el máximo común divisor (mcd) y otra que calcule el mínimo común múltiplo (mcm) y los imprima al regresar al método principal.
15. Similar al anterior, sólo que ahora se deberán leer ocho veces los dos números.
16. Elaborar un algoritmo que permita leer el tamaño de un ángulo y calcule e imprima el seno, coseno y tangente. Utilizar métodos.
17. Un número entero es primo si es divisible sólo por la unidad y por sí mismo. Elaborar un algoritmo que lea un valor N y calcule e imprima los números primos entre 1 y N.
18. Elaborar un algoritmo que lea el dividendo y divisor de una división entera, que llame un método al cual envíe dichos valores mediante parámetros, calcule el cociente y el residuo de la división sin utilizar los operadores `\`, `Mod`, y devuelva los resultados mediante parámetros.
19. Elaborar un algoritmo que permita leer un número entero e indique si es capicúa (es capicúa si se lee igual de izquierda a derecha que en sentido contrario). Utilizar un método que reciba como parámetro el número y que proporcione el valor `True` si es capicúa o `False` en caso de no ser capicúa. Ejemplo: 1991.
20. Elaborar un algoritmo similar al que calcula la media, varianza y desviación estándar, pero ahora usar variables locales en los métodos. Además, utilice un método para leer los números y métodos que regresan valor para calcular el valor de cada medida.
21. Elaborar un algoritmo que permita leer una frase u oración y que imprima la palabra más larga, la más corta y la posición donde inicia cada una de ellas. Utilizar un método que obtenga cada palabra.
22. Elaborar un algoritmo que permita leer una expresión aritmética, la cual estará formada por operadores aritméticos, operandos y paréntesis. Separar y colocar los operandos en un arreglo y los operadores en otro; por último, que imprima la lectura de la expresión y los dos arreglos. Los paréntesis no se colocarán en ninguno de los dos arreglos.
23. Elaborar un algoritmo que permita leer las longitudes de los tres lados de un triángulo y calcule e imprima cada uno de los tres ángulos, utilizando un método para el cálculo de cada uno de los ángulos.
24. Elaborar un algoritmo que permita leer números enteros en una matriz de 4 x 5 e imprima los elementos que al mismo tiempo sean el mayor de su renglón y el mayor de su columna.
25. Elaborar un algoritmo que permita leer una frase u oración e imprima cuántas veces se utilizó cada letra del alfabeto. Utilizar un método que reciba como parámetros la frase y la posición en la que se debe obtener el carácter y que obtenga un carácter en cada llamada.
26. Elaborar un algoritmo que permita leer una frase u oración y que imprima la palabra más larga, contemplando la posibilidad de que haya más de una palabra con la longitud más larga; en tal caso, se puede utilizar un arreglo para guardarlas y al final imprimir todas las palabras que tuvieron la máxima longitud.
27. Elaborar un algoritmo que permita leer 10 números enteros en un arreglo e imprima cuántas veces se utilizó cada uno de los dígitos del 0 al 9.

28. En un triángulo rectángulo es posible que se desee calcular el cateto A, el cateto B o la hipotenusa C de acuerdo a los datos que se tengan disponibles. Elaborar un algoritmo que permita seleccionar el cálculo que se va realizar, leer los datos correspondientes e imprimir el resultado. Por ejemplo, si se escoge calcular la hipotenusa, se deben leer el cateto A y el cateto B; si se selecciona calcular el cateto A, se leen el cateto B y la hipotenusa C; si se escoge calcular el cateto B, se deben leer el cateto A y la hipotenusa. Utilizar métodos para hacer cada uno de los cálculos.
29. Elaborar un algoritmo que permita leer una cantidad de tipo Real, que le inserte comas cada tres cifras y la imprima.

7.8 Resumen de conceptos que debe dominar

- Métodos que no regresan valor
- Variables de clase (globales) y locales; parámetros por referencia y por valor
- Métodos que regresan valor
- Funciones cadenas de caracteres
- Validación de la entrada de datos
- Funciones especiales

7.9 Contenido de la página Web de apoyo

El material marcado con asterisco (*) sólo está disponible para docentes.

7.9.1 Resumen gráfico del capítulo

7.9.2 Autoevaluación

7.9.3 Programas en Java

7.9.4 Ejercicios resueltos

7.9.5 Power Point para el profesor (*)

Programación orientada a objetos usando el diagrama de clases

Contenido

- 8.1 Objetos
 - 8.1.1 Qué son los objetos
 - 8.1.2 Cómo están formados los objetos
 - 8.1.3 Cuándo y cómo identificar los objetos
- 8.2 Clases y su relación con los objetos
 - 8.2.1 Determinar las clases
 - 8.2.2 Representación de la clase y sus instancias
- 8.3 Métodos y encapsulación
 - 8.3.1 Métodos
 - 8.3.2 Encapsulación
- 8.4 Diseño del diagrama de clases
 - 8.4.1 Modificadores de acceso (visibilidad)
- 8.5 Generar instancias de una clase
- 8.6 Arquitectura modelo-vista-controlador
- 8.7 Resumen de conceptos que debe dominar
- 8.8 Contenido de la página Web de apoyo
 - El material marcado con asterisco (*) sólo está disponible para docentes.*
 - 8.8.1 Resumen gráfico del capítulo
 - 8.8.2 Autoevaluación
 - 8.8.3 Power Point para el profesor (*)

Objetivos del capítulo

- Aprender los conceptos básicos de la programación orientada a objetos: objetos, clases, métodos y encapsulación.
- Aplicar lo aprendido en el diseño de programas o algoritmos orientados a objetos, a través del uso del diagrama de clases (adaptación hecha del UML —Unified Modeling Language—, desarrollado por G. Booch, I. Jacobson y J. Rumbaugh), y la arquitectura modelo-vista-controlador.

Competencias

- Competencia general del capítulo
 - Analizar problemas y diseñar el diagrama de clases que los solucionen aplicando los conceptos y la arquitectura orientada a objetos.
- Competencias específicas del capítulo
 - Describe los conceptos que caracterizan a la programación orientada a objetos: objetos, clases, métodos y encapsulación.
 - Diseña diagramas de clase aplicando los conceptos de objetos, clases, métodos, encapsulación y la arquitectura modelo-vista-controlador.

Introducción

Con el estudio de los capítulos anteriores, usted ya domina las bases lógicas de la programación orientada a objetos en pseudocódigo.

En este capítulo se exponen los conceptos básicos de la programación orientada a objetos (objetos, clases, métodos y encapsulación) y su aplicación en el diseño de programas o algoritmos orientados a objetos, a través del uso del diagrama de clases (adaptación hecha del UML —Unified Modeling Language—, desarrollado por G. Booch, I. Jacobson y J. Rumbaugh) y la arquitectura modelo-vista-controlador.

Se expone que un programa orientado a objetos está formado por un conjunto de objetos, donde cada objeto está formado por datos y un conjunto de métodos. Se explica qué son los objetos, cómo están formados, cuándo y cómo identificarlos, qué son las clases y su relación con los objetos, cómo generar instancias de una clase y cómo diseñar el diagrama de clases.

Cabe aclarar que en este capítulo no se presentan ejercicios porque este tema se aplica en todos los ejercicios de los capítulos restantes.

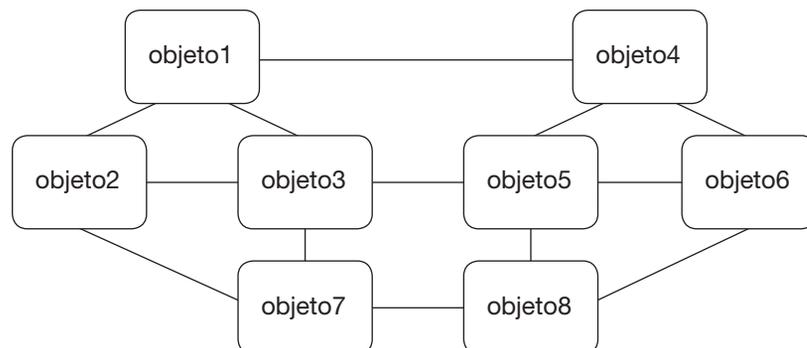
En el siguiente capítulo se estudia la programación orientada a objetos aplicando la estructura de secuenciación.

8.1 Objetos

La Programación Orientada a Objetos (POO, también identificada como OOP —por sus siglas en inglés Object-Oriented Programming) es un método de programación muy parecido a la forma en que nosotros hacemos cosas. Es una evolución natural de las innovaciones más recientes en el diseño de los lenguajes de programación. La POO es más estructurada que las tentativas previas de la programación estructurada y es más modular y abstracta que los intentos previos en abstracción de datos y ocultamiento de detalles. La programación orientada a objetos está caracterizada por las siguientes propiedades: Objetos, Clases, Encapsulación, Herencia y Polimorfismo.

Un programa orientado a objetos está formado por una colección de objetos que interactúan conjuntamente para representar y solucionar algún problema.

Gráficamente, podríamos observar un programa orientado a objetos como se muestra en la siguiente figura:



Explicación:

El programa está constituido por un conjunto de objetos relacionados entre sí, los cuales permiten hacer la entrada de datos, los cálculos necesarios para resolver el problema y dar la salida de datos convertidos en información.

8.1.1 Qué son los objetos

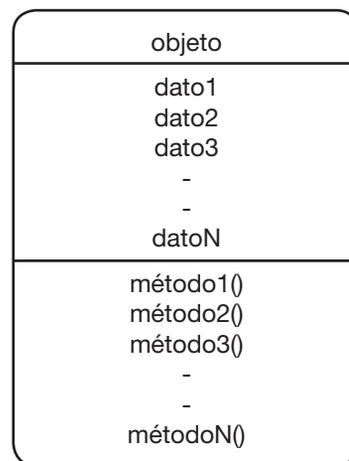
Los objetos son entes, entidades, sujetos o cosas que encontramos en el dominio del problema de nuestra realidad, entiéndase: en situaciones o problemas de nuestro mundo cotidiano empresarial, organizacional o institucional, que se requiere manejar o solucionar mediante la computadora.

8.1.2 Cómo están formados los objetos

Los objetos están formados por dos elementos:

1. **Datos** que representan su estructura. En otras palabras, atributos que describen su forma o apariencia.
2. **Métodos** que implementan las funciones propias de su comportamiento, es decir, métodos que manipulan los datos para hacer la entrada, proceso y salida de los mismos. En el contexto no orientado a objetos, a los métodos se les conoce como procedimientos, módulos, funciones, subrutinas, etcétera.

Esquemáticamente, podríamos observar un objeto como se muestra en la siguiente figura:

*Explicación:*

El objeto se representa con un rectángulo redondeado y está formado por dos partes: una parte son los datos que representan la estructura del objeto y la otra parte son los métodos que implementan las operaciones que se deben realizar sobre los datos.

8.1.3 Cuándo y cómo identificar los objetos

En el entorno del proceso de programación, en el paso de análisis del problema, el elemento central es el objeto; es decir, se estudia el problema y se identifican los objetos involucrados y sus relaciones.

A continuación, planteamos un problema para ejemplificar cómo identificar los objetos.

Definición del problema (es el paso 1 del proceso de programación):

Elaborar un programa que permita calcular el sueldo de los empleados de una empresa.

Análisis del problema (es el paso 2 del proceso de programación):

1) Planteamiento del problema

En la empresa X se tienen varios empleados; por cada empleado se tienen los datos: Nombre, Número de horas trabajadas y Cuota que se le paga por hora trabajada. Cada quincena se requiere emitir un cheque con los datos: Nombre del empleado y Sueldo que se le debe pagar. El sueldo se calcula multiplicando las horas trabajadas por cuota por hora.

Resumiendo, desde la perspectiva de entrada-proceso-salida:

Información que debe imprimir como salida: Nombre y Sueldo.

Datos que se deben leer: Nombre, Horas trabajadas y Cuota por hora.

Proceso por seguir: $\text{Sueldo} = \text{Horas trabajadas} \times \text{Cuota por hora}$.

2) Identificar objetos

Para identificar los objetos presentes en el dominio del problema, se debe estudiar el caso expuesto con el propósito de distinguir y determinar el objeto o colección de objetos que el programa debe representar y manejar para solucionar la situación planteada.

¿Cómo se identifican los objetos? Se buscan los sustantivos presentes en la especificación; los sustantivos son los objetos. En nuestro problema se menciona un sustantivo: los empleados.

Aplicando esto, en el problema formulado líneas antes tenemos que nuestro programa debe representar una colección de empleados como la que se muestra en la siguiente figura:

objetoEmpleado	objetoEmpleado	---	objetoEmpleado
Nombre Juan	Nombre Pedro	---	Nombre Mateo
Horas trabajadas 40	Horas trabajadas 45	---	Horas trabajadas 55
Cuota por hora 50.00	Cuota por hora 100.00	---	Cuota por hora 80.00
Sueldo 2000.00	Sueldo 4500.00	---	Sueldo 4400.00

Explicación:

Se muestra la colección de empleados que debe representar el programa, es decir, a un objetoEmpleado que es el empleado Juan, a otro objetoEmpleado que es el empleado Pedro, y a otro empleado, y a otro, hasta el último objetoEmpleado presente, que es el empleado Mateo.

Así, tenemos una colección de objetos iguales; es decir, se tiene un conjunto de empleados, donde cada empleado es un objeto que tiene la misma estructura de datos y tiene el mismo comportamiento que todos los demás objetos (empleados).

Nota 1: Observe que se está utilizando un rectángulo redondeado para representar a cada objeto.

Nota 2: En este momento se está esquematizando cada objeto desde el punto de vista de su estructura, es decir, de los datos que lo representan; en puntos subsiguientes se explicará cómo involucrar a los métodos.



Observe que se está utilizando un rectángulo redondeado para representar a cada objeto.

En este momento se está esquematizando cada objeto desde el punto de vista de su estructura, es decir, de los datos que lo representan; en puntos subsiguientes se explicará cómo involucrar a los métodos.

8.2 Clases y su relación con los objetos

La clase es una representación abstracta que describe a un conjunto de objetos; en otras palabras, es un tipo de dato abstracto que representa a un conjunto de objetos que tienen en común una misma estructura (de datos) y un mismo comportamiento (las mismas funciones), es decir, que son la misma cosa y hacen lo mismo.

En la fase de diseño del programa, el elemento central es la clase porque un conjunto de objetos iguales debe ser representado en forma abstracta por una clase, que fungirá como una plantilla o molde para crear todos y cada uno de los objetos necesarios.

En el entorno del proceso de programación sigue el paso:

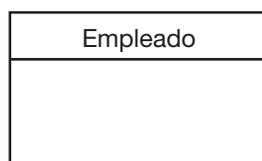
3. Diseño del programa

Para diseñar el programa, se elabora o diseña el algoritmo que soluciona el problema en dos pasos: primero, se diseña el diagrama de clases; segundo, se diseña la lógica de las clases en seudocódigo; después se hace la prueba de escritorio. En este capítulo se explica cómo diseñar el diagrama de clases.

8.2.1 Determinar las clases

Una vez identificados los objetos del problema, se procede a diseñar el diagrama de clases, para lo cual debemos primero determinar la(s) clase(s) que necesitaremos y luego dibujar el diagrama con la representación de la(s) clase(s) y sus instancias.

Los objetos identificados en el punto anterior se convierten en clases que, como ya se mencionó, son la representación abstracta de un conjunto de objetos; aplicando este concepto al problema que nos ocupa, el cual tiene un conjunto de objetos de empleados, tenemos que dicha colección se representa como clase, mediante el símbolo que se indica en la siguiente figura:

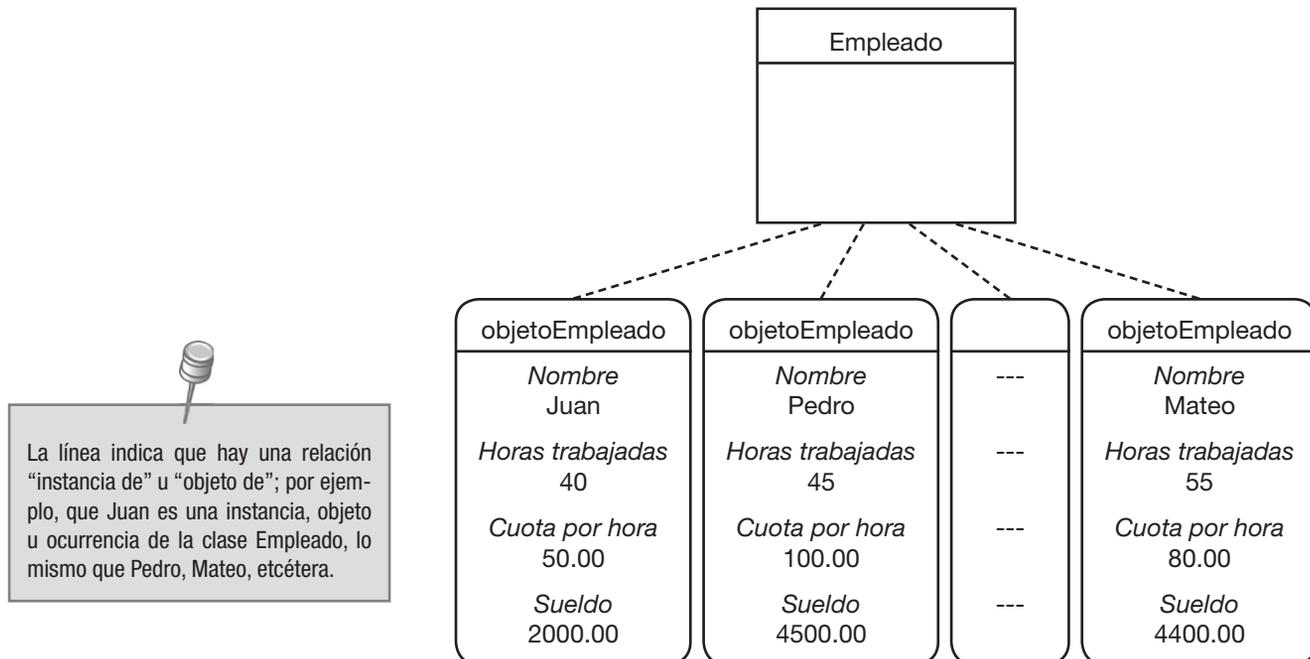


Explicación:

Una clase se representa con un rectángulo. En nuestro ejemplo, estamos representando la clase Empleado.

8.2.2 Representación de la clase y sus instancias

En el programa que habremos de elaborar, debemos definir una clase como un nuevo tipo de dato abstracto y, con base en la referida clase, vamos a generar instancias u ocurrencias específicas de dicha clase. Cada una de estas instancias son los objetos que representarán a cada uno de los empleados presentes en nuestra aplicación; la relación entre una clase y sus instancias se representa como se muestra en la siguiente figura:



Explicación:

Se tiene la clase Empleado, de la cual se genera una instancia, que es el objeto que representa al empleado Juan. Asimismo, se genera otra instancia, que es el objeto que representa al empleado Pedro, y así sucesivamente hasta generar la instancia que es el objeto que representa al empleado Mateo.

Nota: La línea indica que hay una relación “instancia de” u “objeto de”; por ejemplo, que Juan es una instancia, objeto u ocurrencia de la clase Empleado, lo mismo que Pedro, Mateo, etcétera.

8.3 Métodos y encapsulación

8.3.1 Métodos

Como ya lo hemos mencionado, un objeto es representado por medio de datos: datos que deben entrar como materia prima y datos que deben calcularse a partir de otros datos. Los métodos son las acciones que implementan el comportamiento o las funciones de un objeto: por ejemplo, leer o dar entrada a los datos, hacer cálculos, imprimir o dar salida de datos.

8.3.2 Encapsulación

Como ya se explicó anteriormente, un conjunto de objetos es representado en forma abstracta por una clase, la cual es un tipo de dato que está compuesto por dos elementos: datos y métodos. La encapsulación significa colocar juntos los datos y los métodos dentro de un objeto. Uno de los principios más importantes de la programación orientada a objetos es que el programador debe pensar en el código y los datos juntos durante el diseño del programa. Ninguno, ni el código ni los datos, existen en un vacío. Los datos dirigen el flujo del código y el código manipula la forma y valores de los datos.

Cuando el código y los datos son entidades separadas, siempre existe el peligro de llamar al método correcto con los datos erróneos o viceversa. Juntar los dos es trabajo del programador. El objeto mantiene en sincronía los datos y los métodos empacados juntos en el mismo.

Al diseñar la estructura de una clase, se deben hacer dos cosas:

1. Definir los datos que representarán a los objetos de la clase.

Los datos que representan a un objeto son de dos tipos: un tipo son los datos que entrarán como materia prima, es decir, que se deberán leer a través de dispositivos de entrada. El otro tipo son datos que se deben generar mediante cálculos.

2. Definir los métodos que implementarán las acciones de los objetos.

Los métodos son de dos tipos: un tipo son los métodos que colocan o establecen (setters) valores a los datos del objeto, ya sean datos que se leen o datos que se generan mediante cálculos. El otro tipo son los métodos que acceden u obtienen (getters) los valores de los datos para utilizarlos, ya sea para hacer cálculos, para darles salida o hacer otra cosa con ellos.

¿Cómo identificar los métodos? Se buscan los verbos presentes en el planteamiento del problema. En nuestro caso se debe:

Imprimir como salida: Nombre y Sueldo.

Leer los datos: Nombre, Horas trabajadas y Cuota por hora.

Calcular el sueldo: $\text{Sueldo} = \text{Horas trabajadas} \times \text{Cuota por hora}$.

Típicamente, para establecer los valores para cada uno de los datos de un objeto se requiere definir y diseñar un método perteneciente al objeto que permita colocar (setter) el valor a cada dato, ya sea que lo lea o que lo calcule. Asimismo, para dar

salida al valor de un dato se requiere definir y diseñar un método que permita acceder (getter) al valor del dato para darlo como salida, esto es, un método para obtener cada dato.

Como muchos aspectos de la programación orientada a objetos, la encapsulación de datos es una disciplina que se debe respetar siempre. Para establecer u obtener los datos de un objeto, se debe hacer mediante métodos del mismo objeto y no leerlos o accederlos directamente.

Ejemplo:

En el problema de calcular el sueldo de varios empleados, tenemos que cada objeto de empleado se representa mediante los datos:

nombreEmp	Nombre del empleado
horasTrab	Número de horas trabajadas
cuotaHora	Cuota por hora
sueldo	Sueldo del empleado



Observe que los métodos setter inician su nombre con “establecer” o “calcular” y los métodos getter inician con “obtener”. Asimismo, observe que no se define un método getter para todos los datos; en este caso sólo se definieron para los datos que se van a imprimir como salida: nombreEmp y sueldo.

Para cada dato, se debe definir un método para establecer o colocar (setter) el valor que tomará —ya sea que lo tome a través de la lectura del dato, de la realización de cálculos, o de otra forma— y un método obtener (getter) para cada dato a cuyo valor se requiera acceder para utilizarlo, ya sea para hacer cálculos, para imprimirlo o para hacer alguna otra cosa con él.

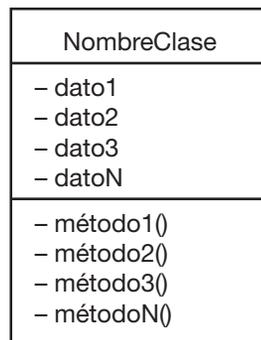
En nuestro problema, se requieren los siguientes métodos:

establecerNombreEmp(). Para establecer el valor del dato nombreEmp.
 establecerHorasTrab(). Para establecer el valor del dato horasTrab.
 establecerCuotaHora(). Para establecer el valor del dato cuotaHora.
 calcularSueldo(). Para calcular y establecer el valor del dato sueldo.
 obtenerNombreEmp(). Para acceder e imprimir el valor del dato nombreEmp.
 obtenerSueldo(). Para acceder e imprimir el valor del dato sueldo.

Nota: Observe que los métodos setter inician su nombre con “establecer” o “calcular” y los métodos getter inician con “obtener”. Asimismo, observe que no se define un método getter para todos los datos; en este caso sólo se definieron para los datos que se van a imprimir como salida: nombreEmp y sueldo.

8.4 Diseño del diagrama de clases

Una vez que se tienen definidos los datos y los métodos necesarios, se procede a diseñar la estructura de la clase, que al mismo tiempo se convierte en el diagrama de clases. Se hace con el siguiente formato:



En donde:

NombreClase	Indica el nombre de la clase.
dato1, dato2, dato3, datoN	Son los datos que representarán a cada uno de los objetos de la clase.
método1(), método2(), método3(), métodoN()	Son los métodos que establecerán y obtendrán los valores de los datos de cada uno de los objetos de esta clase.
()	Indica que es un método.
-	Modificador de acceso (visibilidad).

8.4.1 Modificadores de acceso (visibilidad)

Con los símbolos - + # = * se indica la visibilidad que tendrá cada dato o método. A éstos se les conoce como modificadores de acceso: Privado (-), Protegido (#), Público (+), Estático (=) y Abstracto(*). Esto indica desde qué partes son visibles para ser utilizados.

8.4.1.1 Visibilidad de los datos

- Privado (Private)

Los datos que se declaran como privados sólo pueden ser vistos y utilizados por métodos de la misma clase. Por defecto (default), los datos son privados.

Protegido (Protected)

Los datos que se declaran como protegidos pueden ser vistos y utilizados por métodos de la misma clase y por métodos de subclases derivadas de la clase donde están declarados.

+ Público (Public)

Los datos que se declaran como públicos pueden ser vistos y utilizados tanto por métodos de la misma clase como por métodos de otras clases.

= Estático (Static)

Los datos que se declaran como estáticos son únicos para toda la clase; es decir, no pertenecen a ninguna instancia (objeto) de la clase, pero pueden ser vistos y utilizados por todas las instancias de la clase.

8.4.1.2 Visibilidad de los métodos

- Privado (Private)

Los métodos que se declaran como privados sólo pueden ser vistos y utilizados por métodos de la misma clase.

Protegido (Protected)

Los métodos que se declaran como protegidos pueden ser vistos y utilizados por métodos de la misma clase y por métodos de subclases derivadas de la clase donde están declarados.

+ Público (Public)

Los métodos que se declaran como públicos pueden ser vistos y utilizados tanto por métodos de la misma clase como por métodos de otras clases. Por defecto (default), los métodos son públicos.

= Estático (Static)

Los métodos que se declaran como estáticos son únicos para toda la clase, es decir, no pertenecen a ninguna instancia (objeto) de la clase, pero pueden ser vistos y utilizados por métodos de todas las instancias de la clase.

* Abstracto (Abstract)

Los métodos que se declaran como abstractos no tienen implementación; por tanto, deben ser implementados en subclases.

Ejemplo:

ClaseEjemplo
- dato1 + dato2 # dato3 = dato4
- método1() + método2() # método3() = método4() * método5()

En donde:

Se tienen los datos:

- dato1 Privado (Private)
 + dato2 Público (Public)
 # dato3 Protegido (Protected)
 = dato4 Estático (Static)

Y los métodos:

- método1 Privado (Private)

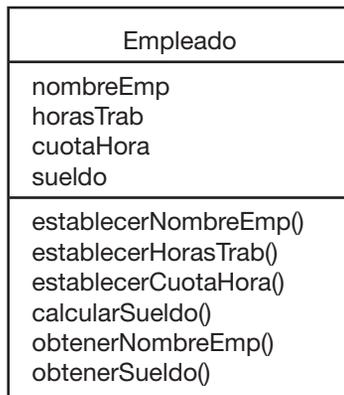
+ método2 Público (Public)
 # método3 Protegido (Protected)
 = método4 Estático (Static)
 * método5 Abstracto (Abstract)

Nota: Por defecto (default), los datos son privados; es decir, si no se les define su visibilidad mediante alguno de los modificadores de acceso (- + # =), entonces los datos son privados.

Por defecto (default), los métodos son públicos; es decir, si no se les define su visibilidad mediante alguno de los modificadores de acceso (- + # = *), entonces los métodos son públicos.

En la metodología que se está presentando en este libro, en los capítulos 9, 10, 11, 12 y 15 estaremos utilizando datos privados y métodos públicos. Es por ello que no se estará indicando el modificador de acceso. Por otro lado, en los capítulos 13, 14 y 16 usaremos otros modificadores de acceso, los cuales se indicarán en su momento.

Aplicando el formato en nuestro ejemplo, nos queda el siguiente diagrama de clases:



Explicación:

Se tiene el diagrama de clases, el cual consta de una clase.

Nombre de la clase: Empleado

Datos que tiene la clase:

nombreEmp Nombre del empleado
 horasTrab Número de horas trabajadas
 cuotaHora Cuota por hora
 sueldo Sueldo del empleado

Métodos que tiene la clase:

establecerNombreEmp(). Para establecer el valor del dato nombreEmp.
 establecerHorasTrab(). Para establecer el valor del dato horasTrab.
 establecerCuotaHora(). Para establecer el valor del dato cuotaHora.



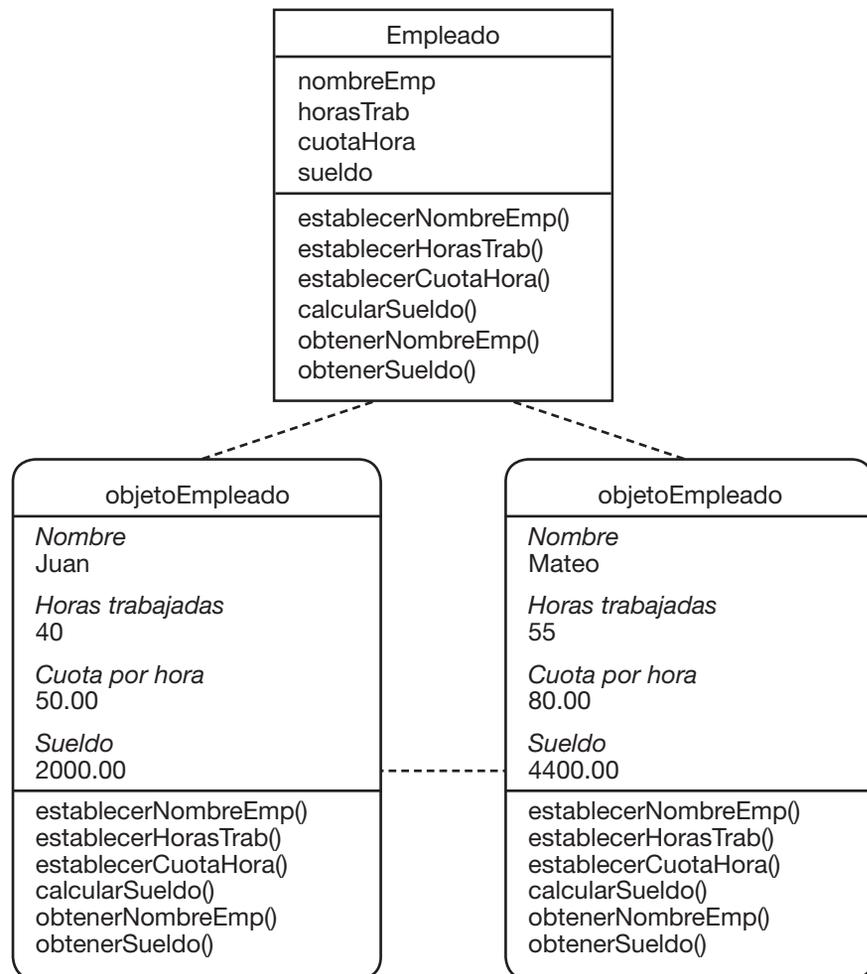
Por defecto (default), los datos son privados; es decir, si no se les define su visibilidad mediante alguno de los modificadores de acceso (- + # =), entonces los datos son privados.

Por defecto (default), los métodos son públicos; es decir, si no se les define su visibilidad mediante alguno de los modificadores de acceso (- + # = *), entonces los métodos son públicos.

calcularSueldo(). Para calcular y establecer el valor del dato sueldo.
 obtenerNombreEmp(). Para acceder e imprimir el valor del dato nombreEmp.
 obtenerSueldo(). Para acceder e imprimir el valor del dato sueldo.

8.5 Generar instancias de una clase

Una vez que tenemos definida una clase como Empleado, ésta se utiliza para generar instancias u ocurrencias. Estas instancias son los objetos que darán vida al funcionamiento de nuestro programa. Esto quiere decir que la clase es como un molde, una plantilla o tipo de dato que se utiliza para generar objetos específicos. En la siguiente figura se esquematiza:



Explicación:

Se tiene la clase Empleado, de la cual se genera una instancia, que es el objetoEmpleado que representa al empleado Juan, y así sucesivamente, hasta generar la instancia que es el objetoEmpleado que representa al empleado Mateo.

Observe que cada objetoEmpleado que se crea de la clase Empleado tendrá los datos:

nombreEmp
horasTrab
cuotaHora
sueldo

Y también tendrá los métodos:

establecerNombreEmp()
establecerHorasTrab()
establecerCuotaHora()
calcularSueldo()
obtenerNombreEmp()
obtenerSueldo().

Nota: La línea indica que hay una relación “instancia de” u “objeto de”: por ejemplo, que Juan es una instancia, objeto u ocurrencia de la clase Empleado; lo mismo Mateo, etcétera.



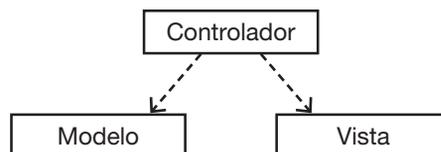
La línea indica que hay una relación “instancia de” u “objeto de”: por ejemplo, que Juan es una instancia, objeto u ocurrencia de la clase Empleado; lo mismo Mateo, etcétera.

8.6 Arquitectura modelo-vista-controlador

El diagrama de clases que se ha diseñado es lo que constituye el modelo de nuestro programa. Sin embargo, este modelo necesita ser utilizado en la solución del problema y para hacerlo se requiere aplicar la arquitectura modelo-vista-controlador, la cual establece que un programa orientado a objetos está formado por tres partes:

1. El modelo. Es la clase o conjunto de clases identificadas en el dominio del problema, mismas que representan al objeto u objetos presentes en el problema.
2. La vista. Es la interfaz de usuario, es decir, lo que el usuario observará en la pantalla, impresora u otro dispositivo de salida de la computadora al operar el programa.
3. El controlador. Es la parte que permite que el usuario interactúe con la vista para utilizar el modelo que representa y soluciona el problema.

En programas complejos la arquitectura modelo-vista-controlador se ve así:





La flecha punteada representa una relación de dependencia en el sentido de que la clase Controlador utiliza a la clase Modelo y a la clase Vista.

Donde la parte Controlador es la que lleva el control del funcionamiento del programa, auxiliándose de la parte Vista, que seguramente es una sofisticada interfaz gráfica de usuario, y de la parte Modelo, que es la que representa y soluciona el problema.

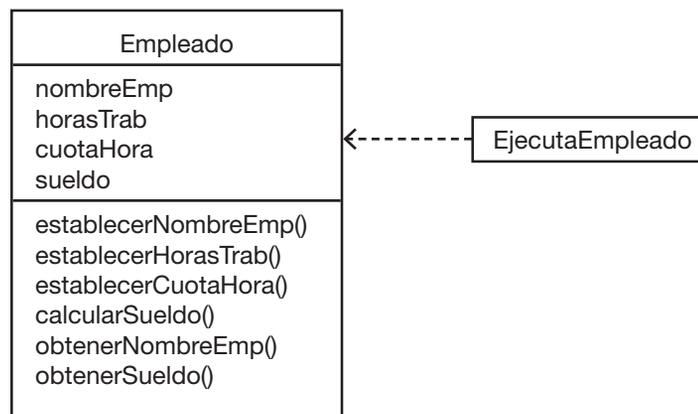
Nota: La flecha punteada representa una relación de dependencia en el sentido de que la clase Controlador utiliza a la clase Modelo y a la clase Vista.

Sin embargo, en programas no muy complejos o en los que la vista es sencilla, la parte Controlador incluye a la parte Vista, y la arquitectura del programa se simplifica a dos partes, la cual se ve así:



Esta arquitectura es la que se utiliza en los problemas que se plantean en este libro debido a que está dirigido a un nivel donde todavía no se enfrentan problemas muy complejos.

Aplicando esta arquitectura al problema que hemos venido diseñando, el diagrama de clases nos queda de la forma siguiente:



A diferencia de todos los demás capítulos de este libro, en éste no se presentan problemas resueltos ni problemas por resolver debido a que la aplicación del diseño de diagramas de clases se hará conjuntamente con los temas que siguen, para así complementar la metodología de diseño de algoritmos orientados a objetos.

Los problemas que se plantean en los capítulos 9, 10 y 11 serán solucionados con un diagrama de clases similar al presentado en este punto. Sin embargo, en los capítulos 12 y 13, donde se tratan los temas de herencia y polimorfismo, será donde se estudie cómo diseñar el diagrama de clases involucrando más clases.

Explicación:

Este diagrama consta de dos clases: una es Empleado que es el modelo que representa y soluciona el problema planteado. La otra es EjecutaEmpleado, que es la clase Controlador, que utiliza el modelo, que es la clase Empleado, para controlar la interacción con el usuario y permitir que el programa sea operado por el usuario para resolver su problema.

Nota 1: A diferencia de todos los demás capítulos de este libro, en éste no se presentan problemas resueltos ni problemas por resolver debido a que la aplicación del diseño de diagramas de clases se hará conjuntamente con los temas que siguen, para así complementar la metodología de diseño de algoritmos orientados a objetos.

Nota 2: Los problemas que se plantean en los capítulos 9, 10 y 11 serán solucionados con un diagrama de clases similar al presentado en este punto. Sin embargo, en los capítulos 12 y 13, donde se tratan los temas de herencia y polimorfismo, será donde se estudie cómo diseñar el diagrama de clases involucrando más clases.

¿Qué hemos hecho hasta ahora?

En este capítulo se han explicado los conceptos básicos de la programación orientada a objetos y se ha planteado la forma de involucrarlos para diseñar el diagrama de clases, el cual contiene la estructura general del programa (algoritmo) que estamos diseñando, esto es, las clases necesarias para resolver el problema.

¿Qué sigue?

Para diseñar programas (algoritmos) aplicando la metodología de la programación orientada a objetos que se está presentando, se deben diseñar dos elementos: uno es el diagrama de clases, y el otro es diseñar la lógica de cada una de las clases que integran el diagrama de clases utilizando la técnica pseudocódigo, para así tener completa la metodología de diseño de programas (algoritmos) orientados a objetos. En el siguiente capítulo se explica cómo hacerlo.

8.7 Resumen de conceptos que debe dominar

- Objetos
- Clases
- Métodos
- Encapsulación
- Métodos
- Diagrama de clases
- Modificadores de acceso
- Arquitectura modelo-vista-controlador

8.8 Contenido de la página Web de apoyo

El material marcado con asterisco () sólo está disponible para docentes*

8.8.1 Resumen gráfico del capítulo

8.8.2 Autoevaluación

8.8.3 Power Point para el profesor (*)

Programación orientada a objetos aplicando la estructura de secuenciación

Contenido

- 9.1 Nuestro primer problema
- 9.2 Diseño de algoritmos OO usando la secuenciación en pseudocódigo
- 9.3 Constructores y destructores
- 9.4 Ejercicios resueltos
- 9.5 Ejercicios propuestos
- 9.6 Resumen de conceptos que debe dominar
- 9.7 Contenido de la página Web de apoyo
El material marcado con asterisco () sólo está disponible para docentes.*
 - 9.7.1 Resumen gráfico del capítulo
 - 9.7.2 Autoevaluación
 - 9.7.3 Programas en Java
 - 9.7.4 Ejercicios resueltos
 - 9.7.5 Power Point para el profesor (*)

Objetivos del capítulo

- Aprender cómo diseñar el algoritmo usando la técnica pseudocódigo y aplicando la estructura de control de secuenciación en el diseño de algoritmos orientados a objetos. Esto es, cómo diseñar la lógica de cada una de las clases definidas en el diagrama de clases, integrando los conceptos de objetos, clases, métodos y encapsulación con los conceptos estudiados en los primeros capítulos en la técnica pseudocódigo.
- Aplicar lo aprendido en la solución de ejercicios resueltos y propuestos.

Competencias

- Competencia general del capítulo
 - *Analizar problemas y diseñar algoritmos que los solucionen aplicando la arquitectura orientada a objetos y la secuenciación.*
- Competencias específicas del capítulo
 - Describe como diseñar algoritmos orientados a objetos usando la secuenciación en pseudocódigo.
 - Define los conceptos constructores y destructores.
 - Diseña algoritmos orientados a objetos aplicando el diagrama de clases y los conceptos de objetos, clases, métodos y encapsulación, integrados con la estructura de control secuenciación en la técnica pseudocódigo.

Introducción

Con el estudio del capítulo anterior, usted ya domina los conceptos básicos de la programación orientada a objetos y sabe cómo diseñar el diagrama de clases.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos mediante la programación orientada a objetos aplicando la estructura de secuenciación.

Se explica cómo diseñar el algoritmo usando la técnica pseudocódigo, tomando como base el diagrama de clases, diseñado con los conceptos aprendidos en el capítulo anterior, aplicando la estructura de control de secuenciación en el diseño de algoritmos orientados a objetos. Esto es, cómo diseñar la lógica de cada una de las clases definidas en el diagrama de clases, integrando las bases lógicas de la programación, estudiadas en los primeros capítulos, con los conceptos de objetos, clases, métodos y encapsulación de la programación orientada a objetos, en la técnica pseudocódigo.

Se expone que los lenguajes orientados a objetos como Java proporcionan por defecto (default) un método constructor y un proceso destructor para cada clase definida.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejercite estudiando los problemas planteados en los ejercicios resueltos y propuestos. Al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro; luego verifique sus resultados con los del libro, analice las diferencias y vea sus errores. Al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no está igual que el del libro, no necesariamente está mal. Usted debe ir aprendiendo a analizar las diferencias y a comprender que a veces, aunque haya diferencias, las dos soluciones están correctas.

En el siguiente capítulo se estudia la programación orientada a objetos aplicando las estructuras de selección.

9.1 Nuestro primer problema

Para diseñar un programa o algoritmo orientado a objetos se hacen dos cosas: primero se diseña el diagrama de clases siguiendo los lineamientos explicados en el capítulo anterior y, segundo, se diseña la lógica de cada una de las clases usando la técnica pseudocódigo. En este punto aplicaremos la estructura de control de secuenciación en pseudocódigo, que estudiamos en el capítulo 3, pero aplicada conjuntamente con el diagrama de clases en el entorno de la programación orientada a objetos.

A continuación se presenta un ejemplo para aplicar los conceptos antes descritos y además para explicar la forma como se diseña un algoritmo orientado a objetos.

Ejemplo:

Elaborar un algoritmo para calcular e imprimir el sueldo de un empleado.

Siguiendo el proceso de programación se hace lo siguiente:

1. Definir el problema.

Elaborar un algoritmo que permita calcular el sueldo de un empleado.

2. Analizar el problema.

Se tienen los datos del empleado: Nombre, Número de horas trabajadas y la Cuota que se le paga por hora trabajada. Se requiere imprimir los datos: Nombre del empleado y Sueldo que se le debe pagar. El Sueldo se calcula multiplicando las horas trabajadas por cuota por hora.

Resumiendo, desde la perspectiva de entrada-proceso-salida:

Información que debe imprimir como salida: Nombre y Sueldo.

Datos que debe leer: Nombre, Número de horas trabajadas y Cuota por hora.

Calcular sueldo: $\text{Sueldo} = \text{Horas trabajadas} \times \text{Cuota por hora}$.

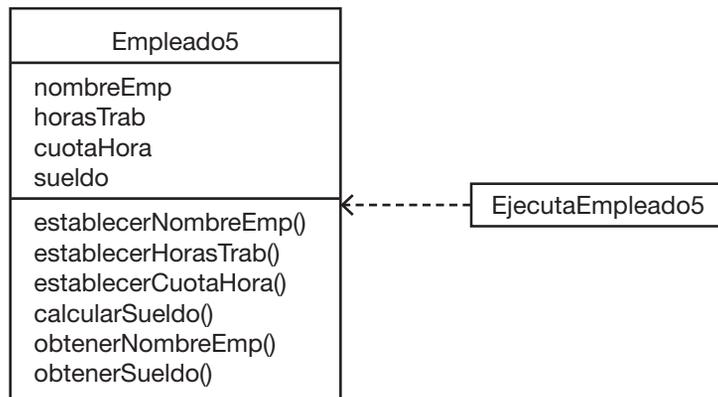
3. Diseñar el programa.

Para diseñar el programa, se elabora el algoritmo que soluciona el problema en dos pasos:

Primero:

Se diseña el diagrama de clases, aplicando los lineamientos que se explicaron en el capítulo anterior (Capítulo 8). Nos queda el siguiente diagrama:

Diagrama de clases



Explicación:

Este diagrama consta de dos clases: una es la clase `Empleado5`, que es el modelo que representa y soluciona el problema planteado, la cual está formada por:

Los datos:

<code>nombreEmp</code>	Nombre del empleado
<code>horasTrab</code>	Número de horas trabajadas
<code>cuotaHora</code>	Cuota por hora
<code>sueldo</code>	Sueldo del empleado



Aquí se utiliza el nombre de clase desde Empleado5, porque Empleado1, Empleado2, Empleado3 y Empleado 4 se usaron en algunas de las clases de capítulos anteriores. Lo mismo sucederá con Alumno, Angulo, Obrero, etcétera.

Y los métodos:

establecerNombreEmp(). Para establecer el valor del dato nombreEmp.
 establecerHorasTrab(). Para establecer el valor del dato horasTrab.
 establecerCuotaHora(). Para establecer el valor del dato cuotaHora.
 calcularSueldo(). Para calcular y establecer el valor del dato sueldo.
 obtenerNombreEmp(). Para acceder e imprimir el valor del dato nombreEmp.
 obtenerSueldo(). Para acceder e imprimir el valor del dato sueldo.

La otra es la clase EjecutaEmpleado5, que es la clase controlador, la cual utiliza el modelo, que es la clase Empleado5, para controlar la interacción con el usuario representando y resolviendo su problema.

Nota: Aquí se utiliza el nombre de clase desde Empleado5, porque Empleado1, Empleado2, Empleado3 y Empleado 4 se usaron en algunas de las clases de capítulos anteriores. Lo mismo sucederá con Alumno, Ángulo, Obrero, etcétera.

Segundo:

Se diseña la lógica de cada una de las clases usando pseudocódigo para armar el algoritmo que resuelve el problema. A continuación se explica cómo elaborar el algoritmo de la solución.

9.2 Diseño de algoritmos OO usando la secuenciación en pseudocódigo

En este punto se explica cómo diseñar el algoritmo orientado a objetos usando pseudocódigo. Cuando se tiene más de una clase el formato del algoritmo es el siguiente:

```

Algoritmo ALGO
  Clase NomClase1
    1. Declarar
      Datos
      Objetos
      Variables

    2. Método funcionUno()
      a. Acción a
      b. Acción b
      c. Fin Método funcionUno

    3. Método funcionN()
      a. Acción a
      b. Acción b
      c. Fin Método funcionN
  Fin Clase NomClase1
  
```

```
Clase NomClase2
  1. Declarar

  2. Método funcionUno()
    a. Acción a
    b. Fin Método funcionUno

  3. Método funcionN()
    a. Acción a
    b. Fin Método funcionN
Fin Clase NomClase2

Clase NomClaseN
  1. Declarar

  2. Método principal()
    a. Acción a
    b. Fin Método principal

  3. Método funcionUno()
    a. Acción a
    b. Fin Método funcionUno

  4. Método funcionN()
    a. Acción a
    b. Fin Método funcionN
Fin Clase NomClaseN
Fin
```

Explicación:

Se tiene el esquema de un algoritmo que tiene tres clases (puede haber más clases). Cada clase tiene su parte de declaraciones, donde se declaran los datos, y enseña los métodos que necesite. Cabe aclarar que, en un algoritmo que tiene varias clases, sólo en una clase se puede tener método principal, ya que es ahí donde el programa inicia su funcionamiento y en el momento en que lo requiera utilizará las otras clases y métodos.

No obstante que un algoritmo puede estar formado por varias o muchas clases, en esta parte de la metodología (capítulos 9, 10, 11 y 12) estaremos utilizando dos clases, esto es, la clase modelo y la clase controlador, donde la clase controlador incluye a la parte vista.

Aplicando este formato en nuestro ejemplo:

Tomando como base el diagrama de clases diseñado en el punto anterior, donde se tienen dos clases: la Clase `Empleado5` y la Clase `EjecutaEmpleado5`, ahora se procederá a diseñar la lógica de cada una de las clases usando pseudocódigo. Queda el siguiente algoritmo:

```

Algoritmo CALCULAR SUELDO DE UN EMPLEADO
Clase Empleado5
1. Declarar datos
    nombreEmp: Cadena
    horasTrab: Entero
    cuotaHora: Real
    sueldo: Real
2. Método establecerNombreEmp(nom: Cadena)
    a. nombreEmp = nom
    b. Fin Método establecerNombreEmp
3. Método establecerHorasTrab(horasTr: Entero)
    a. horasTrab = horasTr
    b. Fin Método establecerHorasTrab
4. Método establecerCuotaHora(cuotaHr: Real)
    a. cuotaHora = cuotaHr
    b. Fin Método establecerCuotaHora
5. Método calcularSueldo()
    a. sueldo = horasTrab * cuotaHora
    b. Fin Método calcularSueldo
6. Método obtenerNombreEmp(): Cadena
    a. return nombreEmp
    b. Fin Método obtenerNombreEmp
7. Método obtenerSueldo(): Real
    a. return sueldo
    b. Fin Método obtenerSueldo
Fin Clase Empleado5
Clase EjecutaEmpleado5
1. Método principal()
    a. Declarar variables
        nomEmp: Cadena
        hrsTra: Entero
        cuoHr: Real
    b. Declarar, crear e iniciar objeto
        Empleado5 objEmpleado = new Empleado5()
    c. Solicitar nombre, número de horas trabajadas,
        cuota por hora
    d. Leer nomEmp, hrsTra, cuoHr
    e. Establecer objEmpleado.establecerNombreEmp(nomEmp)
        objEmpleado.establecerHorasTrab(hrsTra)
        objEmpleado.establecerCuotaHora(cuoHr)
    f. Calcular objEmpleado.calcularSueldo()
    g. Imprimir objEmpleado.obtenerNombreEmp()
        objEmpleado.obtenerSueldo()
    h. Fin Método principal
Fin Clase EjecutaEmpleado5
Fin

```



En Java se genera un archivo .java por cada clase.



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Empleado5.java y EjecutaEmpleado5.java

Nota: En Java se genera un archivo .java por cada clase.

Explicación:

Se tiene el encabezado del algoritmo:

```
Algoritmo CALCULAR SUELDO DE UN EMPLEADO
```

El cual tiene dos clases: `Empleado5` y `EjecutaEmpleado5`.

Clase `Empleado5`

Inicia `Empleado5`, que es la clase modelo que representa al objeto empleado que está presente en este problema, la cual está formada por la parte de declaraciones y seis métodos:

```
1. Declarar datos
   nombreEmp: Cadena
   horasTrab: Entero
   cuotaHora: Real
   sueldo: Real
```

Este paso permite declarar las variables que representan los datos o atributos que caracterizan al objeto empleado, el cual se creará a partir de esta clase; es decir, los datos: nombre del empleado, número de horas trabajadas, cuota que se le paga por hora trabajada y sueldo. Observe que, aunque se están declarando variables que representan datos, preferimos usar “declarar datos” porque esta clase es la clase modelo que se utilizará en la clase ejecuta como molde para crear objetos, y el concepto más preciso en este punto es el de datos; sin embargo, podría ser equivalente ponerle “declarar atributos” o “declarar variables”.

```
2. Método establecerNombreEmp(nom: Cadena)
   a. nombreEmp = nom
   b. Fin Método establecerNombreEmp
```

Este paso es un método (setter) que permite establecer o colocar el valor correspondiente al dato `nombreEmp` (nombre del empleado). Observe que `nom` es un parámetro que recibe el valor que se le envía desde donde se llama a este método y el valor que recibe, lo coloca en el dato `nombreEmp`.

```
3. Método establecerHorasTrab(horasTr: Entero)
   a. horasTrab = horasTr
   b. Fin Método establecerHorasTrab
```

Este paso es un método (setter) que permite establecer o colocar el valor correspondiente al dato `horasTrab` (número de horas trabajadas). Observe que `horasTr` es un parámetro que recibe el valor que se le envía desde donde se llama a este método y el valor que recibe, lo coloca en el dato `horasTrab`.

```
4. Método establecerCuotaHora(cuotaHr: Real)
   a. cuotaHora = cuotaHr
   b. Fin Método establecerCuotaHora
```

Este paso es un método (setter) que permite establecer o colocar el valor correspondiente al dato `cuotaHora` (cuota por hora). Observe que `cuotaHr` es un parámetro que recibe el valor que se le envía desde donde se llama a este método y el valor que recibe, lo coloca en el dato `cuotaHora`.

```
5. Método calcularSueldo()
  a. sueldo = horasTrab * cuotaHora
  b. Fin Método calcularSueldo
```

Este paso es un método (setter) que permite establecer o colocar el valor correspondiente al dato `sueldo`. Observe que se multiplica el valor del dato `horasTrab` por el valor del dato `cuotaHora` y el valor resultante se coloca en el dato `sueldo`.

```
6. Método obtenerNombreEmp(): Cadena
  a. return nombreEmp
  b. Fin Método obtenerNombreEmp
```

Este paso es un método (getter) que permite acceder al valor del dato `nombreEmp`. Observe que lo único que hace es retornar (return) el valor del dato `nombreEmp` al punto desde donde se llama a este método. Cadena es el tipo de dato del valor que regresa.

```
7. Método obtenerSueldo(): Real
  a. return sueldo
  b. Fin Método obtenerSueldo
```

Este paso es un método (getter) que permite acceder al valor del dato `sueldo`. Observe que lo único que hace es retornar (return) el valor del dato `sueldo` al punto desde donde se llama a este método. Real es el tipo de dato del valor que regresa.

Fin Clase Empleado5

Indica el fin de la clase Empleado5.

Clase EjecutaEmpleado5

Inicia EjecutaEmpleado5, que es la clase controlador que utiliza a la clase Empleado5 para controlar la interacción con el usuario y permite que el programa sea operado por éste para resolver su problema. Aunque puede tener más métodos, en problemas con la magnitud que enfrentaremos en este libro sólo tiene el método principal, que a continuación se explica:

```
1. Método principal()
```

Es el encabezado del método principal e indica su inicio.

```
  a. Declarar variables
```

Indica que se hacen declaraciones de variables.

```

nomEmp: Cadena
hrsTra: Entero
cuoHr: Real

```

Se declaran las variables necesarias para implementar la lógica del algoritmo. En este caso sólo se declaran variables para leer cada uno de los datos a los que hay que dar entrada. Éstas son variables estáticas que se almacenan en el segmento de datos de la memoria; son como las variables normales que se usan en la programación tradicional o estructurada. Observe que en esta clase ejecuta se indica declarar variables y no datos, como en la clase modelo, porque esta clase es el controlador y no se utiliza para crear objetos.

nomEmp	Para leer el nombre del empleado.
hrsTra	Para leer el número de horas trabajadas.
cuoHr	Para leer la cuota por hora.

b. Declarar, crear e iniciar objeto

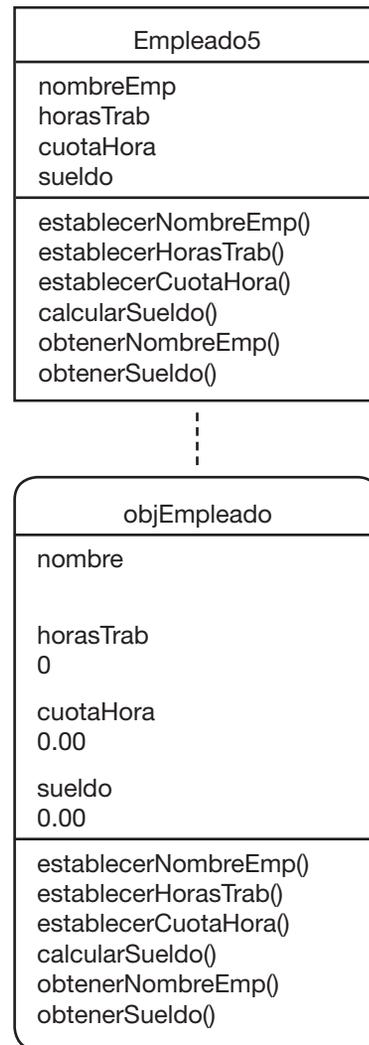
Indica que se declara(n), se genera(n) e inicia(n) objeto(s); en este caso se crea un solo objeto:

```
Empleado5 objEmpleado = new Empleado5()
```

En donde:

Empleado5	Es la clase modelo que se usa como base para declarar y crear el objeto.
objEmpleado	Es el identificador del objeto.
new	Es una función que crea un nuevo objeto.
Empleado5()	Es el método constructor que <code>new</code> utiliza para crear el objeto <code>objEmpleado</code> tomando como base la clase <code>Empleado5</code> y establece sus valores iniciales en la memoria. Para más detalle, ver el apartado 9.3 Constructores y destructores.

Gráficamente, sucede lo siguiente:



Explicación:

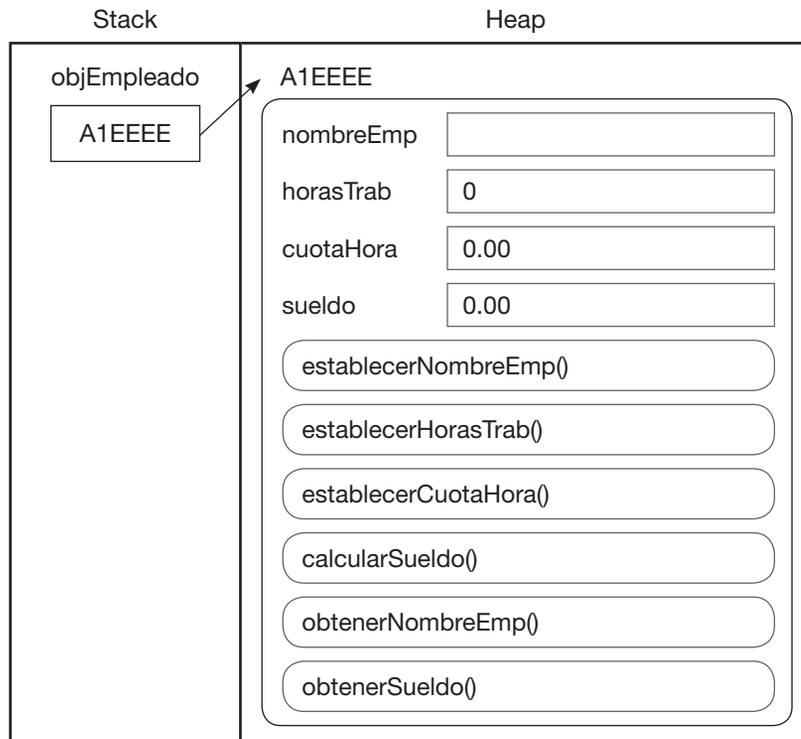
Se genera el objeto objEmpleado tomando como base la clase Empleado5. El objeto creado en la memoria dinámica (Heap) tiene dos elementos. Un elemento son los datos que lo representan: nombreEmp, que se inicia en nulo; horasTrab, que se inicia en cero; cuotaHora, que se inicia en cero; y sueldo, que se inicia en cero. El otro elemento son los métodos: establecerNombreEmp(), establecerHorasTrab(), establecerCuotaHora(), calcularSueldo(), obtenerNombreEmp() y obtenerSueldo(), los cuales permiten acceder a los datos del objeto, ya sea para establecer o para obtener valores.

En la memoria sucede lo siguiente:

Los lenguajes de programación, dividen la memoria de la computadora en: segmento de código (Code), segmento de datos (Data), segmento de pila (Stack) y segmento de variables dinámicas (Heap).

En el segmento de código se almacenan las instrucciones del programa; en el segmento de datos se almacenan las variables estáticas, es decir, las variables normales que se usan en la programación tradicional o estructurada; en el segmento de pila (Stack) se almacenan los datos temporales; y en el segmento de variables dinámicas (Heap) es donde se almacenan los objetos.

En la memoria, nuestro ejemplo se representa así:



Explicación:

En el Stack se crea una referencia (`objEmpleado`) a una dirección (`A1EEEE`) en el Heap que es donde se asigna espacio para alojar el objeto creado, el cual tiene los datos que lo representan y los métodos que permiten acceder a los datos para establecerlos u obtenerlos.

- c. Solicitar nombre, número de horas trabajadas, cuota por hora
- d. Leer `nomEmp`, `hrsTra`, `cuoHr`

Se solicita el nombre del empleado y se lee el valor tecleado en `nomEmp`.
Se solicita el número de horas trabajadas y se lee el valor tecleado en `hrsTra`.
Se solicita la cuota por hora y se lee el valor tecleado en `cuoHr`.

Vamos a suponer que se le dio entrada a los siguientes valores:

<code>nomEmp</code>	<input type="text" value="Socorro Román Maldonado"/>
<code>hrsTra</code>	<input type="text" value="40"/>
<code>cuoHr</code>	<input type="text" value="100.00"/>

```
e. Establecer objEmpleado.establecerNombreEmp (nomEmp)
objEmpleado.establecerHorasTrab (hrsTra)
objEmpleado.establecerCuotaHora (cuoHr)
```

En este paso se procede a colocar los valores leídos en los datos del objeto. A continuación se explica cada acción:

```
objEmpleado.establecerNombreEmp (nomEmp)
```

Se llama al método `establecerNombreEmp` del objeto `objEmpleado`, enviando como parámetro `nomEmp`; es decir, el valor de `nomEmp`, que es Socorro Román Maldonado, se envía al método `establecerNombreEmp`, donde se tiene el parámetro `nom`, el cual recibe este valor que se le envía desde aquí y lo coloca en el dato `nombreEmp`. Observe que `nomEmp` es una variable que reside en el segmento de datos de la memoria y `nombreEmp` reside en el objeto que a su vez reside en la memoria dinámica (Heap).

```
objEmpleado.establecerHorasTrab (hrsTra)
```

Se llama al método `establecerHorasTrab` del objeto `objEmpleado`, enviando como parámetro `hrsTra`; es decir, el valor de `hrsTra`, que es 40, se envía al método `establecerHorasTrab`, donde se tiene el parámetro `horasTr`, el cual recibe este valor que se le envía desde aquí y lo coloca en el dato `horasTrab`. Observe que `hrsTra` es una variable que reside en el segmento de datos de la memoria y `horasTrab` reside en el objeto que a su vez reside en la memoria dinámica (Heap).

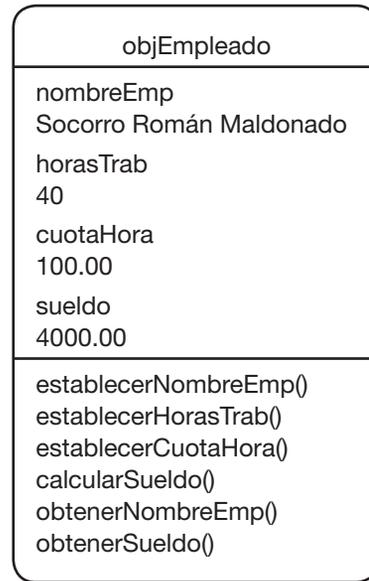
```
objEmpleado.establecerCuotaHora (cuoHr)
```

Se llama al método `establecerCuotaHora` del objeto `objEmpleado`, enviando como parámetro `cuoHr`; es decir, el valor de `cuoHr`, que es 100.00, se envía al método `establecerCuotaHora`, donde se tiene el parámetro `cuotaHr`, el cual recibe este valor que se le envía desde aquí y lo coloca en el dato `cuotaHora`. Observe que `cuoHr` es una variable que reside en el segmento de datos de la memoria y `cuotaHora` reside en el objeto que a su vez reside en la memoria dinámica (Heap).

```
f. Calcular objEmpleado.calcularSueldo ()
```

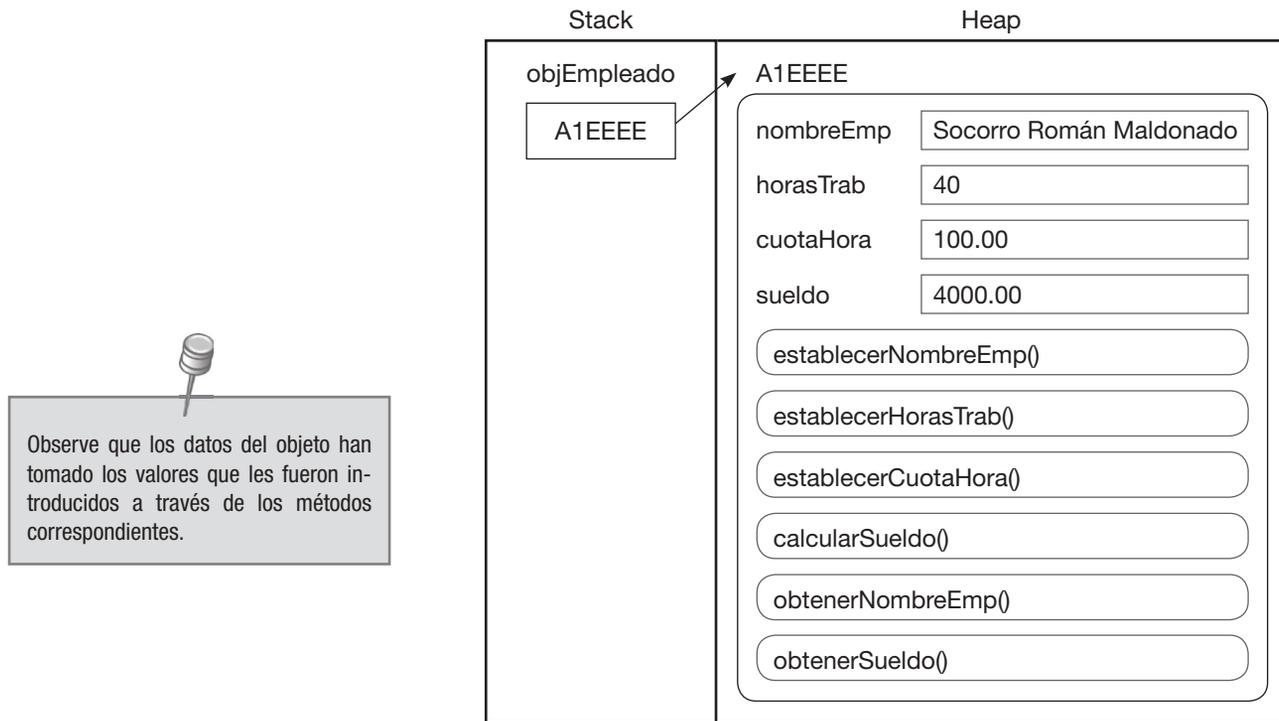
Se llama al método `calcularSueldo` del objeto `objEmpleado`, donde calcula el sueldo multiplicando el valor del dato `horasTrab` por el valor del dato `cuotaHora`. El valor resultante, 4000.00, lo coloca en el dato `sueldo`.

Gráficamente, el objeto nos queda como se muestra en la siguiente figura:



Es decir, los datos del objeto tomaron valores a través de los métodos del mismo objeto.

En la memoria, nuestro ejemplo ahora queda así:



Observe que los datos del objeto han tomado los valores que les fueron introducidos a través de los métodos correspondientes.

Observe que los datos del objeto han tomado los valores que les fueron introducidos a través de los métodos correspondientes.

```
g. Imprimir objEmpleado.obtenerNombreEmp()
   objEmpleado.obtenerSueldo()
```

En este paso se procede a imprimir o dar salida a los valores de los datos del objeto que se requirieron al plantear el problema. Enseguida se explican:

```
objEmpleado.obtenerNombreEmp()
```

Se llama al método `obtenerNombreEmp` del objeto `objEmpleado`, el cual accede y retorna (`return`) el valor del dato `nombreEmp` a este punto para imprimirlo.

```
objEmpleado.obtenerSueldo()
```

Se llama al método `obtenerSueldo` del objeto `objEmpleado`, el cual accede y retorna (`return`) el valor del dato `sueldo` a este punto para imprimirlo.

```
h. Fin Método principal
```

Indica el fin del método principal.

Fin Clase EjecutaEmpleado5

Indica el fin de la clase EjecutaEmpleado5.

Fin

Indica el fin del algoritmo.

Resumiendo:

En la clase controlador `EjecutaEmpleado5`, en el Método `principal()`, es donde se implementa la secuencia de pasos lógica que controla la solución del problema:

1. Se declaran variables normales (estáticas) para dar entrada a los datos.
2. Se declara, crea e inicializa el objeto empleado utilizando el modelo: la clase `Empleado5`. En el objeto creado están empaquetados o encapsulados juntos los datos y los métodos; los datos, que representan la estructura del objeto, sólo pueden accederse a través de los métodos del mismo objeto.
3. Se solicitan los datos.
4. Se leen los datos en las variables declaradas en este método principal, luego se colocan en los datos del objeto a través de los métodos del mismo objeto: `establecerNombreEmp()`, `establecerHorasTrab()`, `establecerCuotaHora()`.
5. Se hace el cálculo del sueldo llamando al método `calcularSueldo()`.
6. Se imprimen los datos de salida, accediendo a los datos del objeto a través de los métodos `obtenerNombreEmp()` y `obtenerSueldo()`.

Esto es, se establece un proceso que le permita al usuario interactuar con la computadora para solucionar su problema introduciendo los datos que se le solicitan y recibiendo los resultados a través de la parte vista, incluida en esta clase controlador.

9.3 Constructores y destructores

Los lenguajes orientados a objetos como Java proporcionan por defecto (default) un método constructor y un proceso destructor para cada clase definida.

Constructor

El método constructor es ejecutado con `new` y su función es asignar espacio y crear el objeto en la memoria dinámica (Heap). Asimismo, inicia los datos con los valores que se hayan asignado en la declaración; o bien, si no fueron iniciados con valores en la declaración, se inician con los valores por defecto (default): los datos numéricos en ceros y los datos cadena (string) en nulos. El constructor tiene el mismo nombre que la clase pero con `()` al final. En el ejemplo del punto anterior la clase es `Empleado5` y su constructor es `Empleado5()`.

Si en nuestro ejemplo quisiéramos iniciar los datos del objeto con valores específicos, necesitaríamos el método constructor:

```

2. Método Empleado5()
  a. nombreEmp = "María"
    horasTrab = 40
    cuotaHora = 100.00
    sueldo = 4000.00
  b. Fin Método Empleado5

```

Los datos se están iniciando con los valores especificados y la numeración de los demás métodos se reacomodaría.



Aunque los lenguajes orientados a objetos permiten definir el constructor y, en el caso del destructor, algunos lenguajes también permiten definirlo y/o llamarlo, en este libro estaremos usando los conceptos por defecto (default) explicados en los párrafos anteriores.

Destructor

El proceso destructor hace lo contrario que el constructor, es decir, libera el espacio ocupado por los objetos creados en la memoria dinámica. El proceso destructor, por defecto (default), entra en acción cada determinado tiempo y destruye los objetos creados que ya no tienen referencias, esto es, que ya no se utilizan.

Nota: Aunque los lenguajes orientados a objetos permiten definir el constructor y, en el caso del destructor, algunos lenguajes también permiten definirlo y/o llamarlo, en este libro estaremos usando los conceptos por defecto (default) explicados en los párrafos anteriores.

9.4 Ejercicios resueltos



Es pertinente recordar que todo algoritmo orientado a objetos se elabora en dos pasos. Primero, se diseña el diagrama de clases aplicando los lineamientos que se explicaron en el capítulo 8. Segundo, se diseña la lógica de cada una de las clases usando pseudocódigo y aplicando los lineamientos que se explicaron en los puntos anteriores de este capítulo.

En este punto se presentan ejercicios resueltos aplicando la metodología propuesta en el capítulo anterior y en éste, pero también se usan conceptos desarrollados en los primeros tres capítulos; los problemas resueltos en este punto son algunos de los ejercicios resueltos del capítulo 3. Es decir, el mismo problema ya lo resolvimos aplicando la lógica básica de la programación en el capítulo 3 y aquí lo estamos resolviendo aplicando la lógica de la programación orientada a objetos.

Nota: Es pertinente recordar que todo algoritmo orientado a objetos se elabora en dos pasos. Primero, se diseña el diagrama de clases aplicando los lineamientos que se explicaron en el capítulo 8. Segundo, se diseña la lógica de cada una de las clases usando pseudocódigo y aplicando los lineamientos que se explicaron en los puntos anteriores de este capítulo.

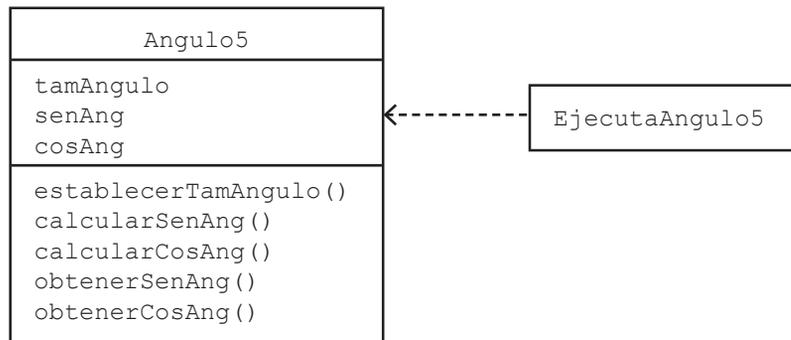
Ejercicio 9.4.1

Elaborar un algoritmo que permita leer el tamaño de un ángulo en radianes, luego que calcule e imprima el seno y coseno.

A continuación se presenta el algoritmo de la solución:

(Primero hágalo usted; después compare la solución)

Diagrama de clases



Algoritmo CALCULOS LOGARITMICOS DE ANGULO

Clase Angulo5

1. Declarar datos
tamAngulo, senAng, cosAng: Real
2. Método establecerTamAngulo(ang: Real)
 - a. tamAngulo = ang
 - b. Fin Método establecerTamAngulo
3. Método calcularSenAng()
 - a. senAng = Seno(tamAngulo)
 - b. Fin Método calcularSenAng
4. Método calcularCosAng()
 - a. cosAng = Coseno(tamAngulo)
 - b. Fin Método calcularCosAng
5. Método obtenerSenAng(): Real
 - a. return senAng
 - b. Fin Método obtenerSenAng
6. Método obtenerCosAng(): Real
 - a. return cosAng
 - b. Fin Método obtenerCosAng

Fin Clase Angulo5

Clase EjecutaAngulo5

1. Método principal()
 - a. Declarar variables
tamAng: Real
 - b. Declarar, crear e iniciar objeto
Angulo5 objAngulo = new Angulo5()
 - c. Solicitar tamaño del ángulo en radianes
 - d. Leer tamAng

```

    e. Establecer objAngulo.establecerTamAngulo (tamAng)
    f. Calcular objAngulo.calcularSenAng ()
        objAngulo.calcularCosAng ()
    g. Imprimir objAngulo.obtenerSenAng ()
        objAngulo.obtenerCosAng ()
    h. Fin Método principal
Fin Clase EjecutaAngulo5
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Angulo5.java y EjecutaAngulo5.java

Explicación:

El algoritmo tiene dos clases: la Clase Angulo5 y la Clase EjecutaAngulo5.

En la Clase Angulo5:

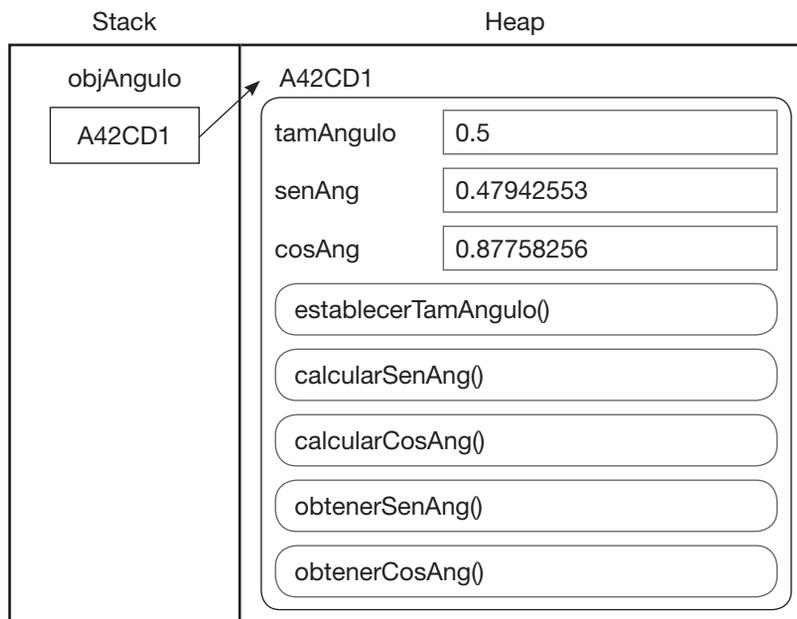
1. Se declaran los datos que representan la estructura de la clase:
 - tamAngulo para el tamaño del ángulo.
 - senAng para el seno.
 - cosAng para el coseno.
 2. Método establecerTamAngulo (ang: Real).
 - Recibe en el parámetro ang el valor que luego coloca en el dato tamAngulo.
 3. Método calcularSenAng ()
 - a. Calcula el seno de tamAngulo llamando a la función Seno (tamAngulo).
 4. Método calcularCosAng ().
 - a. Calcula el coseno de tamAngulo llamando a la función Coseno (tamAngulo).
 5. Método obtenerSenAng () Real.
 - a. Retorna senAng (seno de tamAngulo).
 6. Método obtenerCosAng () Real.
 - a. Retorna cosAng (coseno de tamAngulo).
- Fin de la Clase Angulo5.

En la Clase EjecutaAngulo5, en el Método principal ():

- a. Se declara la variable:
 - tamAng para leer el tamaño del ángulo.
- b. Se declara el objeto objAngulo, usando como base a la Clase Angulo5. Dicho objeto se crea e inicializa mediante el constructor por defecto Angulo5 ().
- c. Se solicita el tamaño del ángulo en radianes.
- d. Se lee en tamAng.
- e. Se llama al Método establecerTamAngulo (tamAng) del objeto objAngulo para colocar el valor de tamAng en el dato tamAngulo.
- f. Se llama al Método calcularSenAng () del objeto objAngulo para calcular el seno del dato tamAngulo.
Se llama al Método calcularCosAng () del objeto objAngulo para calcular el coseno del dato tamAngulo.

- g. Se llama al Método `obtenerSenAng()` del objeto `objAngulo` para acceder e imprimir el valor del dato `senAng`.
Se llama al Método `obtenerCosAng()` del objeto `objAngulo` para acceder e imprimir el valor del dato `cosAng`.
- h. Fin del Método principal.
Fin de la Clase `EjecutaAngulo5`.
Fin del algoritmo.

En la siguiente figura se muestra cómo se ve el objeto en la memoria:



Explicación:

En el Stack se crea una referencia `objAngulo` a la dirección `A42CD1` del Heap, donde se almacena el objeto creado, el cual tiene los datos y métodos que ya fueron explicados. Observe que los datos del objeto han tomado los valores que les fueron introducidos a través de los métodos correspondientes.

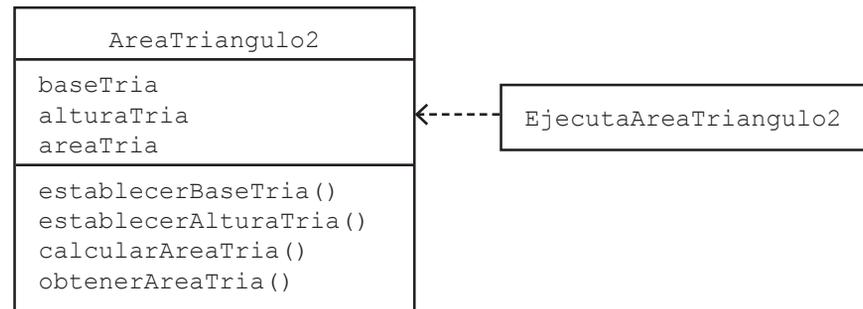
Ejercicio 9.4.2

Elaborar un algoritmo para calcular el área de un triángulo. Se requiere imprimir como salida el área del triángulo. Los datos disponibles para leer como entrada son la base y la altura del triángulo.

El área se calcula:
$$\text{Área} = \frac{\text{Base} \times \text{Altura}}{2}$$

A continuación se presenta el algoritmo de la solución:
(Primero hágalo usted; después compare la solución)

Diagrama de clases



Algoritmo AREA TRIANGULO

Clase AreaTriangulo2

1. Declarar datos

baseTria: Real
 alturaTria: Real
 areaTria: Real

2. Método establecerBaseTria(base: Real)

a. baseTria = base
 b. Fin Método establecerBaseTria

3. Método establecerAlturaTria(altura: Real)

a. alturaTria = altura
 b. Fin Método establecerAlturaTria

4. Método calcularAreaTria()

a. areaTria = (baseTria * alturaTria) / 2
 b. Fin Método calcularAreaTria

5. Método obtenerAreaTria(): Real

a. return = areaTria
 b. Fin Método obtenerAreaTria

Fin Clase AreaTriangulo2

Clase EjecutaAreaTriangulo2

1. Método principal()

a. Declarar variables

basTri, altuTri: Real

b. Declarar, crear e iniciar objeto

AreaTriangulo2 objTriangulo = new AreaTriangulo2()

c. Solicitar Base, Altura

```

d. Leer basTri, altuTri
e. Establecer objTriangulo.establecerBaseTria(basTri)
   objTriangulo.establecerAlturaTria(altuTri)
f. Calcular objTriangulo.calcularAreaTria()
g. Imprimir objTriangulo.obtenerAreaTria()
h. Fin Método principal
Fin Clase EjecutaAreaTriangulo2
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: AreaTriangulo2.java y EjecutaAreaTriangulo2.java



Explicación:

El algoritmo tiene dos clases: la Clase AreaTriangulo2 y la Clase EjecutaAreaTriangulo2.

En la Clase AreaTriangulo2:

1. Se declaran los datos que representan la estructura de la clase:
 baseTria para la base del triángulo.
 alturaTria para la altura del triángulo.
 areaTria para el área del triángulo.
 2. Método establecerBaseTria(base: Real)
 Recibe en el parámetro base el valor que luego coloca en el dato baseTria.
 3. Método establecerAlturaTria(altura: Real).
 Recibe en el parámetro altura el valor que luego coloca en el dato alturaTria.
 4. Método calcularAreaTria().
 Calcula el área del triángulo.
 5. Método obtenerAreaTria() Real.
 Retorna areaTria (área del triángulo).
- Fin de la Clase AreaTriangulo2.

En la Clase EjecutaAreaTriangulo2, en el Método principal():

- a. Se declaran las variables:
 basTri para leer la base del triángulo.
 altuTri para leer la altura del triángulo.
- b. Se declara el objeto objTriangulo, usando como base a la Clase AreaTriangulo2. Dicho objeto se crea e inicializa mediante el constructor por defecto AreaTriangulo2().
- c. Se solicita la base y la altura.
- d. Se leen en basTri y altuTri.
- e. Se llama al Método establecerBaseTria(basTri) del objeto objTriangulo para colocar el valor de basTri en el dato baseTria.
 Se llama al Método establecerAlturaTria(altuTri) del objeto objTriangulo para colocar el valor de altuTri en el dato alturaTria.

- f. Se llama al Método `calcularAreaTria()` del objeto `objTriangulo` para calcular el área del triángulo.
 - g. Se llama al Método `obtenerAreaTria()` del objeto `objTriangulo` para acceder e imprimir el valor del dato `areaTria`.
 - h. Fin del Método principal.
- Fin de la Clase `EjecutaAreaTriangulo2`.
Fin del algoritmo.

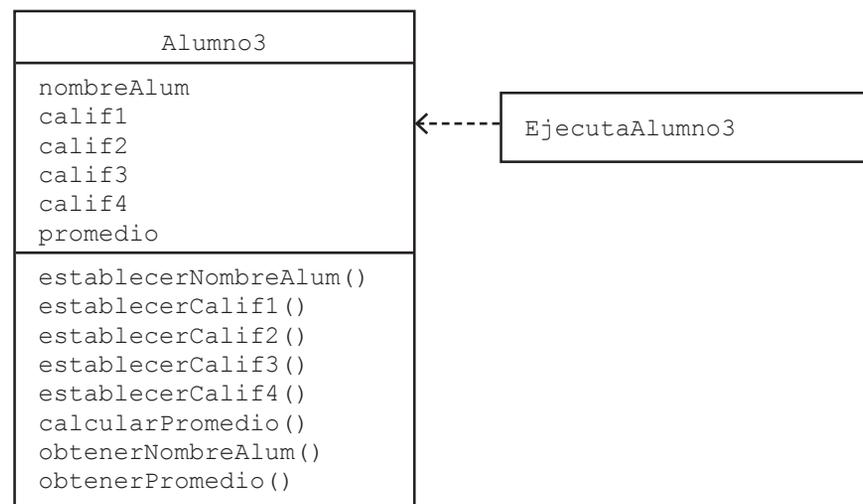
Ejercicio 9.4.3

Elaborar un algoritmo para calcular el promedio de calificaciones de un estudiante. Los datos disponibles para lectura son el nombre, calificación 1, calificación 2, calificación 3 y calificación 4 de cada uno de los cuatro exámenes presentados. La información que se debe imprimir es el nombre y el promedio de las calificaciones. El promedio se obtiene sumando las cuatro calificaciones y dividiendo la suma entre 4.

A continuación se presenta el algoritmo de la solución:

(Primero hágalo usted; después compare la solución)

Diagrama de clases



Algoritmo CALIFICACION ALUMNO

Clase Alumno3

1. Declarar datos
 - nombreAlum: Cadena
 - calif1: Real
 - calif2: Real
 - calif3: Real
 - calif4: Real
 - promedio: Real

```
2. Método establecerNombreAlum(nom: Cadena)
  a. nombreAlum = nom
  b. Fin Método establecerNombreAlum

3. Método establecerCalif1(ca1: Real)
  a. calif1 = ca1
  b. Fin Método establecerCalif1

4. Método establecerCalif2(ca2: Real)
  a. calif2 = ca2
  b. Fin Método establecerCalif2

5. Método establecerCalif3(ca3: Real)
  a. calif3 = ca3
  b. Fin Método establecerCalif3

6. Método establecerCalif4(ca4: Real)
  a. calif4 = ca4
  b. Fin Método establecerCalif4

7. Método calcularPromedio()
  a. promedio = (calif1+ calif2+ calif3+ calif4)/4
  b. Fin Método calcularPromedio

8. Método obtenerNombreAlum(): Cadena
  a. return nombreAlum
  b. Fin Método obtenerNombreAlum

9. Método obtenerPromedio(): Real
  a. return promedio
  b. Fin Método obtenerPromedio
Fin Clase Alumno3

Clase EjecutaAlumno3
1. Método principal()
  a. Declarar variables
     nombre: Cadena
     c1, c2, c3, c4: Real
  b. Declarar, crear e iniciar objeto
     Alumno3 objAlumno = new Alumno3()
  c. Solicitar nombre del alumno, calificación 1,
     calificación 2, calificación 3, calificación 4
  d. Leer nombre, c1, c2, c3, c4
  e. Establecer objAlumno.establecerNombreAlum(nombre)
     objAlumno.establecerCalif1(c1)
     objAlumno.establecerCalif2(c2)
     objAlumno.establecerCalif3(c3)
     objAlumno.establecerCalif4(c4)
  f. Calcular objAlumno.calcularPromedio()
```

```

        g. Imprimir objAlumno.obtenerNombreAlum()
           objAlumno.obtenerPromedio()
        h. Fin Método principal
    Fin Clase EjecutaAlumno3
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Alumno3.java y EjecutaAlumno3.java

Explicación:

El algoritmo tiene dos clases: la Clase Alumno3 y la Clase EjecutaAlumno3.

En la Clase Alumno3:

1. Se declaran los datos que representan la estructura de la clase:
 - nombreAlum para el nombre del alumno.
 - calif1 para la calificación 1 del alumno.
 - calif2 para la calificación 2 del alumno.
 - calif3 para la calificación 3 del alumno.
 - calif4 para la calificación 4 del alumno.
 - promedio para el promedio del alumno.
 2. Método establecerNombreAlum(nom: Cadena)
 - Recibe en el parámetro nom el valor que luego coloca en el dato nombreAlum.
 3. Método establecerCalif1(ca1: Real).
 - Recibe en el parámetro ca1 el valor que luego coloca en el dato calif1.
 4. Método establecerCalif2(ca2: Real).
 - Recibe en el parámetro ca2 el valor que luego coloca en el dato calif2.
 5. Método establecerCalif3(ca3: Real).
 - Recibe en el parámetro ca3 el valor que luego coloca en el dato calif3.
 6. Método establecerCalif4(ca4: Real).
 - Recibe en el parámetro ca4 el valor que luego coloca en el dato calif4.
 7. Método calcularPromedio().
 - Calcula el promedio del alumno.
 8. Método obtenerNombreAlum() Cadena.
 - Retorna nombreAlum (nombre del alumno).
 9. Método obtenerPromedio() Real.
 - Retorna promedio (promedio del alumno).
- Fin de la Clase Alumno3.

En la Clase EjecutaAlumno3, en el Método principal():

- a. Se declaran las variables:
 - nombre para leer el nombre del alumno.
 - c1 para leer la calificación 1 del alumno.
 - c2 para leer la calificación 2 del alumno.
 - c3 para leer la calificación 3 del alumno.
 - c4 para leer la calificación 4 del alumno.

- b. Se declara el objeto `objAlumno`, usando como base a la Clase `Alumno3`. Dicho objeto se crea e inicializa mediante el constructor por defecto `Alumno3()`.
- c. Se solicitan nombre, calificación 1, calificación 2, calificación 3, calificación 4.
- d. Se leen en nombre, `c1`, `c2`, `c3`, `c4`.
- e. Se llama al Método `establecerNombreAlum(nombre)` del objeto `objAlumno` para colocar el valor de nombre en el dato `nombreAlum`.
Se llama al Método `establecerCalif1(c1)` del objeto `objAlumno` para colocar el valor de `c1` en el dato `calif1`.
Se llama al Método `establecerCalif2(c2)` del objeto `objAlumno` para colocar el valor de `c2` en el dato `calif2`.
Se llama al Método `establecerCalif3(c3)` del objeto `objAlumno` para colocar el valor de `c3` en el dato `calif3`.
Se llama al Método `establecerCalif4(c4)` del objeto `objAlumno` para colocar el valor de `c4` en el dato `calif4`.
- f. Se llama al Método `calcularPromedio()` del objeto `objAlumno` para calcular el promedio.
- g. Se llama al Método `obtenerNombreAlum()` del objeto `objAlumno` para acceder e imprimir el valor del dato `nombreAlum`.
Se llama al Método `obtenerPromedio()` del objeto `objAlumno` para acceder e imprimir el valor del dato `promedio`.
- h. Fin del Método principal.
Fin de la Clase `EjecutaAlumno3`
Fin del algoritmo

Tabla 9.1: ejercicios resueltos disponibles en la zona de descarga del capítulo 9 de la Web del libro.

Ejercicio	Descripción
Ejercicio 9.4.4	Calcula el precio de venta de un artículo
Ejercicio 9.4.5	Calcula la hipotenusa de un triángulo rectángulo
Ejercicio 9.4.6	Convierte grados a radianes y viceversa

9.5 Ejercicios propuestos

Como ejercicios propuestos para este punto, se recomiendan, del capítulo 3, algunos de los ejercicios resueltos que no fueron incluidos en este punto; además, todos los ejercicios propuestos en dicho capítulo.

9.6 Resumen de conceptos que debe dominar

- Cómo diseñar algoritmos orientados a objetos aplicando la estructura de secuenciación en pseudocódigo.
- Constructores y destructores.

9.7 Contenido de la página Web de apoyo

El material marcado con asterisco () sólo está disponible para docentes.*

9.7.1 Resumen gráfico del capítulo

9.7.2 Autoevaluación

9.7.3 Programas en Java

9.7.4 Ejercicios resueltos

9.7.5 Power Point para el profesor (*)

10

Programación orientada a objetos aplicando las estructuras de selección

Contenido

- 10.1 Diseño de algoritmos OO usando la selección doble (if-then-else)
- 10.2 Diseño de algoritmos OO usando la selección simple (if-then)
- 10.3 Diseño de algoritmos OO usando la selección múltiple (switch)
- 10.4 Ejercicios resueltos
- 10.5 Ejercicios propuestos
- 10.6 Resumen de conceptos que debe dominar
- 10.7 Contenido de la página Web de apoyo
 - El material marcado con asterisco (*) sólo está disponible para docentes.*
 - 10.7.1 Resumen gráfico del capítulo
 - 10.7.2 Autoevaluación
 - 10.7.3 Programas en Java
 - 10.7.4 Ejercicios resueltos
 - 10.7.5 Power Point para el profesor (*)

Objetivos del capítulo

- Aprender cómo diseñar algoritmos orientados a objetos aplicando las estructuras de selección: if-then-else, if-then y switch.
- Aplicar lo aprendido en la solución de ejercicios resueltos y propuestos.

Competencias

- Competencia general del capítulo
 - *Analizar problemas y diseñar algoritmos que los solucionen aplicando la arquitectura orientada a objetos y la selección if-then-else, if-then y switch.*
- Competencias específicas del capítulo
 - Diseña algoritmos orientados a objetos aplicando la selección doble (if-then-else).
 - Elabora algoritmos orientados a objetos aplicando la selección simple (if-then).
 - Desarrolla algoritmos orientados a objetos aplicando la selección múltiple (switch).

Introducción

Con el estudio del capítulo anterior, usted ya domina los conceptos básicos de la programación orientada a objetos aplicando la estructura de secuenciación y sabe cómo diseñar algoritmos usando dicha estructura.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos mediante la programación orientada a objetos aplicando las estructuras de selección.

Se explica cómo diseñar algoritmos orientados a objetos usando las estructuras de control selección que estudiamos en el capítulo 4, pero ahora inmersas en la arquitectura orientada a objetos, que se estudió en los dos capítulos anteriores.

Recordemos que la selección doble (if-then-else) sirve para plantear situaciones en las que se tienen dos alternativas de acción y se debe escoger una. La selección simple (if-then) sirve para cuando se tiene una alternativa de acción condicionada. Y la selección múltiple (switch) sirve para cuando se tienen múltiples alternativas de acción, de las cuales se debe escoger una.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende. Es por ello que es esencial que ejercite estudiando los problemas planteados en los ejercicios resueltos y propuestos. Al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro, luego verifique sus resultados con los del libro, analice las diferencias y vea sus errores. Al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no está igual que el del libro, no necesariamente está mal. Usted debe ir aprendiendo a analizar las diferencias y a comprender que a veces, aunque haya diferencias, las dos soluciones están correctas.

En el siguiente capítulo se estudia la programación orientada a objetos aplicando las estructuras de repetición.

10.1 Diseño de algoritmos OO usando la selección doble (if-then-else)

En este capítulo se utilizarán las estructuras de selección en pseudocódigo que se estudiaron en el capítulo 4, pero ahora aplicadas conjuntamente con el diagrama de clases y los conceptos de la programación orientada a objetos, es decir, en el diseño de algoritmos orientados a objetos.

En este punto se aplica la selección doble (if-then-else).

Ejemplo:

Calcular el sueldo de un empleado.

Información por imprimir como salida:

Nombre y Sueldo

Datos disponibles para leer:

Nombre, número de horas trabajadas y cuota por hora

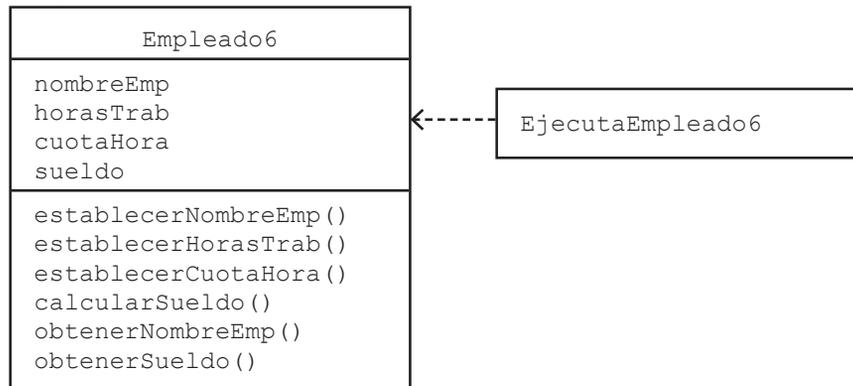
Proceso:

El sueldo se calcula de la forma siguiente:

Si el número de horas trabajadas es mayor que 40, el excedente de 40 se paga al doble de la cuota por hora. En caso de no ser mayor que 40, se paga a la cuota por hora normal.

A continuación se presenta el diagrama de clases de la solución:

Diagrama de clases

**Explicación:**

Este diagrama consta de dos clases: una es la clase `Empleado6`, que es el modelo que representa y soluciona el problema planteado, la cual está formada por:

Los datos:

<code>nombreEmp</code>	Nombre del empleado
<code>horasTrab</code>	Número de horas trabajadas
<code>cuotaHora</code>	Cuota por hora
<code>sueldo</code>	Sueldo del empleado

Y los métodos:

`establecerNombreEmp()`. Para establecer el valor del dato `nombreEmp`.
`establecerHorasTrab()`. Para establecer el valor del dato `horasTrab`.
`establecerCuotaHora()`. Para establecer el valor del dato `cuotaHora`.
`calcularSueldo()`. Para calcular y establecer el valor del dato `sueldo`.
`obtenerNombreEmp()`. Para acceder e imprimir el valor del dato `nombreEmp`.
`obtenerSueldo()`. Para acceder e imprimir el valor del dato `sueldo`.

La otra es la clase `EjecutaEmpleado6`, que es la clase controlador, la cual utiliza el modelo, que es la clase `Empleado6`, para controlar la interacción con el usuario representando y resolviendo su problema.

A continuación se presenta el algoritmo de la solución en pseudocódigo:

Algoritmo CALCULAR SUELDO DOBLE DE UN EMPLEADO

Clase Empleado6

1. Declarar datos

 nombreEmp: Cadena
 horasTrab: Entero
 cuotaHora: Real
 sueldo: Real

2. Método establecerNombreEmp(nom: Cadena)

 a. nombreEmp = nom
 b. Fin Método establecerNombreEmp

3. Método establecerHorasTrab(horasTr: Entero)

 a. horasTrab = horasTr
 b. Fin Método establecerHorasTrab

4. Método establecerCuotaHora(cuotaHr: Real)

 a. cuotaHora = cuotaHr
 b. Fin Método establecerCuotaHora

5. Método calcularSueldo()

 a. if horasTrab <= 40 then
 1. sueldo = horasTrab * cuotaHora
 b. else
 1. sueldo = (40*cuotaHora)+
 ((horasTrab-40)*(cuotaHora*2))
 c. endif
 d. Fin Método calcularSueldo

6. Método obtenerNombreEmp(): Cadena

 a. return nombreEmp
 b. Fin Método obtenerNombreEmp

7. Método obtenerSueldo(): Real

 a. return sueldo
 b. Fin Método obtenerSueldo

Fin Clase Empleado6

Clase EjecutaEmpleado6

1. Método principal()

 a. Declarar variables
 nomEmp: Cadena
 hrsTra: Entero
 cuoHr: Real

 b. Declarar, crear e iniciar objeto
 Empleado6 objEmpleado = new Empleado6()

```

c. Solicitar nombre, número de horas trabajadas,
   cuota por hora
d. Leer nomEmp, hrsTra, cuoHr
e. Establecer objEmpleado.establecerNombreEmp(nomEmp)
   objEmpleado.establecerHorasTrab(hrsTra)
   objEmpleado.establecerCuotaHora(cuoHr)
f. Calcular objEmpleado.calcularSueldo()
g. Imprimir objEmpleado.obtenerNombreEmp()
   objEmpleado.obtenerSueldo()
h. Fin Método principal
Fin Clase EjecutaEmpleado6
Fin

```

En la zona de descarga de la Web del libro está disponible:
Programa en Java: Empleado6.java y EjecutaEmpleado6.java



Explicación:

El algoritmo tiene dos clases: la Clase Empleado6 y la Clase EjecutaEmpleado6.

En la Clase Empleado6:

1. Se declaran los datos que representan la estructura de la clase:
 - nombreEmp para el nombre del empleado.
 - horasTrab para el número de horas trabajadas.
 - cuotaHora para la cuota por hora.
 - sueldo para el sueldo del empleado.
2. Método establecerNombreEmp(nom: Cadena).

Recibe en el parámetro nom el valor que luego coloca en el dato nombreEmp.
3. Método establecerHorasTrab(horasTr: Entero).

Recibe en el parámetro horasTr el valor que luego coloca en el dato horasTrab.
4. Método establecerCuotaHora(cuotaHr: Real).

Recibe en el parámetro cuotaHr el valor que luego coloca en el dato cuotaHora.
5. Método calcularSueldo()
 - a. Si horasTrab <= 40 entonces:
 1. Se calcula el sueldo en forma simple.
 - b. Si no:
 1. Se calcula el sueldo: 40 horas a la cuota simple y el excedente de 40 al doble de la cuota.
 - c. Fin del if.
 - d. Fin del Método calcularSueldo().
6. Método obtenerNombreEmp() Cadena.

Retorna nombreEmp.
7. Método obtenerSueldo() Real.

Retorna sueldo.

Fin de la Clase Empleado6.

En la Clase EjecutaEmpleado6, en el Método principal():

- a. Se declaran las variables:
La variable `nomEmp` para leer el nombre del empleado.
La variable `hrsTra` para leer el número de horas trabajadas.
La variable `cuoHr` para leer la cuota por hora.
 - b. Se declara el objeto `objEmpleado`, usando como base a la Clase `Empleado6`. Dicho objeto se crea e inicializa mediante el constructor por defecto `Empleado6()`.
 - c. Se solicitan nombre, número de horas trabajadas y cuota por hora.
 - d. Se leen en `nomEmp`, `hrsTra`, `cuoHr`.
 - e. Se llama al Método `establecerNombreEmp(nomEmp)` del objeto `objEmpleado` para colocar el valor de `nomEmp` en el dato `nombreEmp`.
Se llama al Método `establecerHorasTrab(hrsTra)` del objeto `objEmpleado` para colocar el valor de `hrsTra` en el dato `horasTrab`.
Se llama al Método `establecerCuotaHora(cuoHr)` del objeto `objEmpleado` para colocar el valor de `cuoHr` en el dato `cuotaHora`.
 - f. Se llama al Método `calcularSueldo()` del objeto `objEmpleado` para calcular el sueldo.
 - g. Se llama al Método `obtenerNombreEmp()` del objeto `objEmpleado` para acceder e imprimir el valor del dato `nombreEmp`.
Se llama al Método `obtenerSueldo()` del objeto `objEmpleado` para acceder e imprimir el valor del dato `sueldo`.
 - h. Fin del Método `principal`.
- Fin de la Clase `EjecutaEmpleado6`.
Fin del algoritmo.

Práctica: En este momento se recomienda practicar haciendo algunos ejercicios resueltos y propuestos para aplicar el if-then-else.

10.2 Diseño de algoritmos OO usando la selección simple (if-then)

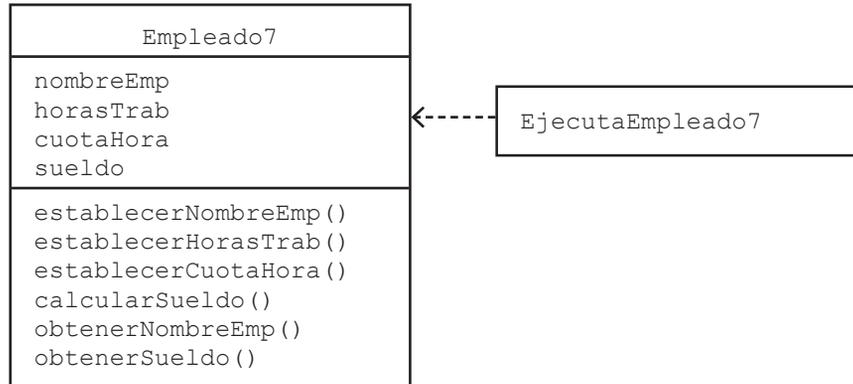
En este punto se aplica la selección simple (if-then).

Ejemplo:

Siguiendo con el mismo problema de calcular el sueldo de un empleado, ahora se otorga un incentivo del 5% si el empleado trabajó más de 40 horas. Esto se agrega, independientemente del cálculo del sueldo.

A continuación se presenta el algoritmo de la solución:

Diagrama de clases



Algoritmo CALCULAR SUELDO CON INCENTIVO DE UN EMPLEADO

Clase Empleado7

1. Declarar datos

```

nombreEmp: Cadena
horasTrab: Entero
cuotaHora: Real
sueldo: Real
  
```

2. Método establecerNombreEmp(nom: Cadena)

```

a. nombreEmp = nom
b. Fin Método establecerNombreEmp
  
```

3. Método establecerHorasTrab(horasTr: Entero)

```

a. horasTrab = horasTr
b. Fin Método establecerHorasTrab
  
```

4. Método establecerCuotaHora(cuotaHr: Real)

```

a. cuotaHora = cuotaHr
b. Fin Método establecerCuotaHora
  
```

5. Método calcularSueldo()

```

a. sueldo = horasTrab * cuotaHora
b. if horasTrab > 40 then
    1. sueldo = sueldo + (sueldo * 0.05)
c. endif
d. Fin Método calcularSueldo
  
```

6. Método obtenerNombreEmp(): Cadena

```

a. return nombreEmp
b. Fin Método obtenerNombreEmp
  
```

7. Método obtenerSueldo(): Real

```

a. return sueldo
b. Fin Método obtenerSueldo
  
```

Fin Clase Empleado7

```

Clase EjecutaEmpleado7
1. Método principal()
  a. Declarar variables
     nomEmp: Cadena
     hrsTra: Entero
     cuoHr: Real
  b. Declarar, crear e iniciar objeto
     Empleado7 objEmpleado = new Empleado7()
  c. Solicitar nombre, número de horas trabajadas,
     cuota por hora
  d. Leer nomEmp, hrsTra, cuoHr
  e. Establecer objEmpleado.establecerNombreEmp(nomEmp)
     objEmpleado.establecerHorasTrab(hrsTra)
     objEmpleado.establecerCuotaHora(cuoHr)
  f. Calcular objEmpleado.calcularSueldo()
  g. Imprimir objEmpleado.obtenerNombreEmp()
     objEmpleado.obtenerSueldo()
  h. Fin Método principal
Fin Clase EjecutaEmpleado7
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Empleado7.java y EjecutaEmpleado7.java

Explicación:

Este algoritmo es casi igual al del punto anterior; lo único diferente es que en el Método calcularSueldo() ahora se hace el cálculo así:

5. Método calcularSueldo()
 - a. Se calcula el sueldo en forma normal ($\text{horasTrab} * \text{cuotaHora}$).
 - b. Si $\text{horasTrab} > 40$ entonces:
 1. Al sueldo se le agrega el 5% del mismo sueldo.
 - c. Fin del if.
 - d. Fin del Método calcularSueldo.



Práctica: En este momento se recomienda practicar haciendo algunos ejercicios resueltos y propuestos para aplicar el if-then.

Práctica: En este momento se recomienda practicar haciendo algunos ejercicios resueltos y propuestos para aplicar el if-then.

10.3 Diseño de algoritmos OO usando la selección múltiple (switch)

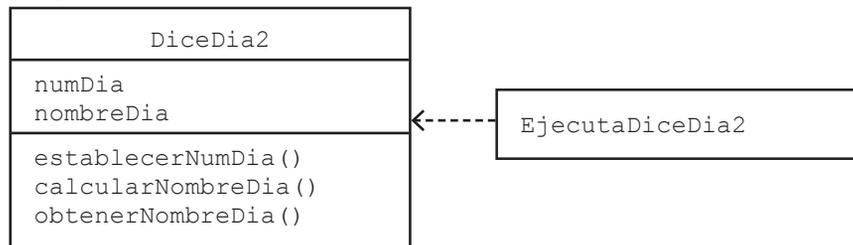
En este punto se aplica la selección múltiple (switch).

Ejemplo:

Elaborar un algoritmo que lea el número de día (un valor entre 1 y 7) e imprima domingo si es 1, lunes si es 2, ..., sábado si es 7.

A continuación se presenta el algoritmo de la solución:

Diagrama de clases



Algoritmo DICE DIA

Clase DiceDia2

1. Declarar datos

numDia: Entero
nombreDia: Cadena

2. Método establecerNumDia(num: Entero)

a. numDia = num
b. Fin Método establecerNumDia

3. Método calcularNombreDia()

a. switch numDia
1: nombreDia = "DOMINGO"
2: nombreDia = "LUNES"
3: nombreDia = "MARTES"
4: nombreDia = "MIERCOLES"
5: nombreDia = "JUEVES"
6: nombreDia = "VIERNES"
7: nombreDia = "SABADO"
b. default
1. nombreDia = "NO ESTA EN EL RANGO DE 1 A 7"
c. endswitch
d. Fin Método calcularNombreDia

4. Método obtenerNombreDia(): Cadena

a. return nombreDia
b. Fin Método obtenerNombreDia

Fin Clase DiceDia2

Clase EjecutaDiceDia2

1. Método principal()

a. Declarar variables
nDia: Entero
b. Declarar, crear e iniciar objeto
DiceDia2 objDia = new DiceDia2()
c. Solicitar número de día
d. Leer nDia

```

e. Establecer objDia.establecerNumDia (nDia)
f. Calcular objDia.calcularNombreDia ()
g. Imprimir objDia.obtenerNombreDia ()
h. Fin Método principal
Fin Clase EjecutaDiceDia2
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: DiceDia2.java y EjecutaDiceDia2.java

Explicación:

El algoritmo tiene dos clases: la Clase DiceDia2 y la Clase EjecutaDiceDia2.

En la Clase DiceDia2:

1. Se declaran los datos que representan la estructura de la clase:
 - numDia para el número de día.
 - nombreDia para el nombre del día.
 2. Método establecerNumDia (num: Entero)
 - Recibe en el parámetro num el valor que luego coloca en el dato numDia.
 3. Método calcularNombreDia
 - Establece el nombre del día:
 - a. switch numDia
 - Si numDia ==1 entonces: nombreDia = "DOMINGO".
 - Si numDia ==2 entonces: nombreDia = "LUNES".
 - Si numDia ==3 entonces: nombreDia = "MARTES".
 - Si numDia ==4 entonces: nombreDia = "MIERCOLES".
 - Si numDia ==5 entonces: nombreDia = "JUEVES".
 - Si numDia ==6 entonces: nombreDia = "VIERNES".
 - Si numDia ==7 entonces: nombreDia = "SABADO".
 - b. Si no es ninguno, entonces:
 1. nombreDia = "NO ESTA EN EL RANGO DE 1 A 7".
 - c. Fin del switch.
 - d. Fin del método.
 - 4. Método obtenerNombreDia () Cadena.
 - Retorna nombreDia (nombre del día).
- Fin de la Clase DiceDia2.

En la Clase EjecutaDiceDia2, en el Método principal ():

- a. Se declara la variable:
 - nDia para leer el número de día.
- b. Se declara el objeto objDia, usando como base a la Clase DiceDia2. Dicho objeto se crea e inicializa mediante el constructor por defecto DiceDia2 ().
- c. Se solicita el número de día.
- d. Se lee en nDia.
- e. Se llama al Método establecerNumDia (nDia) del objeto objDia para colocar el valor de nDia en el dato numDia.

- f. Se llama al Método `calcularNombreDia()` del objeto `objDia` para calcular el nombre del día `nombreDia`.
 - g. Se llama al Método `obtenerNombreDia()` del objeto `objDia` para acceder e imprimir el valor del dato `nombreDia`.
 - h. Fin del Método principal.
- Fin de la Clase `EjecutaDiceDia2`.
Fin del algoritmo.

Práctica: En este momento se recomienda practicar haciendo el ejercicio resuelto y algunos propuestos para aplicar el switch.



Práctica: En este momento se recomienda practicar haciendo el ejercicio resuelto y algunos propuestos para aplicar el switch.

10.4 Ejercicios resueltos

En este punto se presentan ejercicios resueltos aplicando la metodología propuesta en el capítulo anterior y en éste, pero también se usan conceptos desarrollados en los primeros cuatro capítulos; los problemas resueltos en este punto son algunos de los ejercicios resueltos del capítulo 4. Es decir, el mismo problema ya lo resolvimos aplicando la lógica básica de la programación en el capítulo 4 y aquí lo estamos resolviendo aplicando la lógica de la programación orientada a objetos.

Se le recomienda que primero haga usted el algoritmo y después compare su solución con la del libro. En esta parte hay ejercicios donde se utilizan las tres estructuras de selección. En los puntos anteriores, al estudiar cada estructura de selección, se recomienda venir a esta parte y practicar con ejercicios correspondientes a la estructura que esté estudiando, de manera que a esta parte va a venir en varias ocasiones.

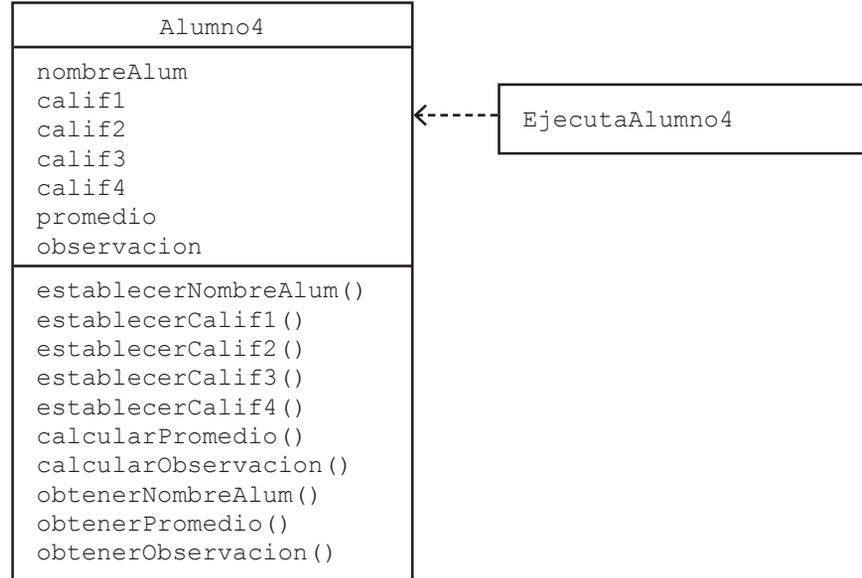
Ejercicio 10.4.1

Elaborar un algoritmo para calcular el promedio de calificaciones de un estudiante. Los datos disponibles para lectura son el nombre, calificación 1, calificación 2, calificación 3 y calificación 4 de cada uno de los cuatro exámenes presentados. La información que se debe imprimir es el nombre, el promedio de las calificaciones y un comentario de “Aprobado” si obtiene 60 o más, o “Reprobado” en caso contrario. El promedio se obtiene sumando las cuatro calificaciones y dividiendo la suma entre 4. Use if-then-else.

A continuación se presenta el algoritmo de la solución:

(Primero hágalo usted; después compare la solución)

Diagrama de clases



Algoritmo CALCULA PROMEDIO DE UN ALUMNO

Clase Alumno4

1. Declarar datos
 - nombreAlum: Cadena
 - calif1: Real
 - calif2: Real
 - calif3: Real
 - calif4: Real
 - promedio: Real
 - observacion: Cadena
2. Método establecerNombreAlum(nom: Cadena)
 - a. nombreAlum = nom
 - b. Fin Método establecerNombreAlum
3. Método establecerCalif1(ca1: Real)
 - a. calif1 = ca1
 - b. Fin Método establecerCalif1
4. Método establecerCalif2(ca2: Real)
 - a. calif2 = ca2
 - b. Fin Método establecerCalif2
5. Método establecerCalif3(ca3: Real)
 - a. calif3 = ca3
 - b. Fin Método establecerCalif3
6. Método establecerCalif4(ca4: Real)
 - a. calif4 = ca4
 - b. Fin Método establecerCalif4

```

7. Método calcularPromedio()
  a. promedio = (calif1+ calif2+ calif3+ calif4)/4
  b. Fin Método calcularPromedio

8. Método calcularObservacion()
  a. if promedio >= 60 then
      1. observacion = "Aprobado"
  b. else
      1. observacion = "Reprobado"
  c. endif
  d. Fin Método calcularObservacion

9. Método obtenerNombreAlum(): Cadena
  a. return nombreAlum
  b. Fin Método obtenerNombreAlum

10. Método obtenerPromedio(): Real
  a. return promedio
  b. Fin Método obtenerPromedio

11. Método obtenerObservacion(): Cadena
  a. return observacion
  b. Fin Método obtenerObservacion
Fin Clase Alumno4

Clase EjecutaAlumno4
1. Método principal()
  a. Declarar variables
     nombre: Cadena
     c1, c2, c3, c4: Real
  b. Declarar, crear e iniciar objeto
     Alumno4 objAlumno = new Alumno4()
  c. Solicitar nombre del alumno, calificación 1,
     calificación 2, calificación 3, calificación 4
  d. Leer nombre, c1, c2, c3, c4
  e. Establecer objAlumno.establecerNombreAlum(nombre)
     objAlumno.establecerCalif1(c1)
     objAlumno.establecerCalif2(c2)
     objAlumno.establecerCalif3(c3)
     objAlumno.establecerCalif4(c4)
  f. Calcular objAlumno.calcularPromedio()
     objAlumno.calcularObservacion()
  g. Imprimir objAlumno.obtenerNombreAlum()
     objAlumno.obtenerPromedio()
     objAlumno.obtenerObservacion()

  h. Fin Método principal
Fin Clase EjecutaAlumno4
Fin

```

En la zona de descarga de la Web del libro está disponible:
 Programa en Java: Alumno4.java y EjecutaAlumno4.java



Explicación:

El algoritmo tiene dos clases: la Clase Alumno4 y la Clase EjecutaAlumno4.

En la Clase Alumno4:

1. Se declaran los datos que representan la estructura de la clase:
 - nombreAlum para el nombre del alumno.
 - calif1 para la calificación 1 del alumno.
 - calif2 para la calificación 2 del alumno.
 - calif3 para la calificación 3 del alumno.
 - calif4 para la calificación 4 del alumno.
 - promedio para el promedio del alumno.
 - observacion en el que se colocará Aprobado o Reprobado.
 2. Método establecerNombreAlum(nom: Cadena).
Recibe en el parámetro nom el valor que luego coloca en el dato nombreAlum.
 3. Método establecerCalif1(ca1: Real).
Recibe en el parámetro ca1 el valor que luego coloca en el dato calif1.
 4. Método establecerCalif2(ca2: Real).
Recibe en el parámetro ca2 el valor que luego coloca en el dato calif2.
 5. Método establecerCalif3(ca3: Real).
Recibe en el parámetro ca3 el valor que luego coloca en el dato calif3.
 6. Método establecerCalif4(ca4: Real).
Recibe en el parámetro ca4 el valor que luego coloca en el dato calif4.
 7. Método calcularPromedio().
Calcula el promedio del alumno.
 8. Método calcularObservacion().
 - a. Si promedio \geq 60 entonces:
 1. Coloca "Aprobado" en observacion.
 - b. Si no:
 1. Coloca "Reprobado" en observacion.
 - c. Fin del if.
 - d. Fin del Método calcularObservacion.
 9. Método obtenerNombreAlum() Cadena.
Retorna nombreAlum (nombre del alumno).
 10. Método obtenerPromedio() Real.
Retorna promedio (promedio del alumno).
 11. Método obtenerObservacion() Cadena.
Retorna observacion.
- Fin de la Clase Alumno4.

En la Clase EjecutaAlumno4, en el Método principal():

- a. Se declaran las variables:
 - nombre para leer el nombre del alumno.
 - c1 para leer la calificación 1 del alumno.
 - c2 para leer la calificación 2 del alumno.
 - c3 para leer la calificación 3 del alumno.
 - c4 para leer la calificación 4 del alumno.

- b. Se declara el objeto `objAlumno`, usando como base a la Clase `Alumno4`. Dicho objeto se crea e inicializa mediante el constructor por defecto `Alumno4()`.
- c. Se solicitan nombre del alumno, calificación 1, calificación 2, calificación 3 y calificación 4.
- d. Se leen en nombre, `c1`, `c2`, `c3`, `c4`.
- e. Se llama al Método `establecerNombreAlum(nombre)` del objeto `objAlumno` para colocar el valor de nombre en el dato `nombreAlum`.
Se llama al Método `establecerCalif1(c1)` del objeto `objAlumno` para colocar el valor de `c1` en el dato `calif1`.
Se llama al Método `establecerCalif2(c2)` del objeto `objAlumno` para colocar el valor de `c2` en el dato `calif2`.
Se llama al Método `establecerCalif3(c3)` del objeto `objAlumno` para colocar el valor de `c3` en el dato `calif3`.
Se llama al Método `establecerCalif4(c4)` del objeto `objAlumno` para colocar el valor de `c4` en el dato `calif4`.
- f. Se llama al Método `calcularPromedio()` del objeto `objAlumno` para calcular el promedio.
Se llama al Método `calcularObservacion()` del objeto `objAlumno` para calcular la observación.
- g. Se llama al Método `obtenerNombreAlum()` del objeto `objAlumno` para acceder e imprimir el valor del dato `nombreAlum`.
Se llama al Método `obtenerPromedio()` del objeto `objAlumno` para acceder e imprimir el valor del dato `promedio`.
Se llama al Método `obtenerObservacion()` del objeto `objAlumno` para acceder e imprimir el valor del dato `observacion`.
- h. Fin del Método principal.
Fin de la Clase `EjecutaAlumno4`.
Fin del algoritmo.

Ejercicio 10.4.2

Elaborar un algoritmo similar al anterior de CALCULAR SUELDO DOBLE DE UN EMPLEADO, pero ahora tomando en cuenta que se tiene otra alternativa: las horas que exceden de 50 se pagan al triple de la cuota por hora. Use if-then-else.

A continuación se presenta el algoritmo:

(Primero hágalo usted; después compare la solución)

La solución es muy similar a la del algoritmo CALCULAR SUELDO DOBLE DE UN EMPLEADO: el diagrama de clases es igual y la Clase `EjecutaEmpleado7` es igual. La Clase `Empleado7` es la que cambia; el Método `calcularSueldo()` queda así:

```

5. Método calcularSueldo()
  a. if horasTrab <= 40 then
    1. sueldo = horasTrab * cuotaHora
  b. else
    1. if horasTrab <= 50 then
      a. sueldo=(40*cuotaHora)+
        ((horasTrab-40)*(cuotaHora*2))

```

```

    2. else
      a. sueldo=(40*cuotaHora)+(10*cuotaHora*2)
        +( (horasTrab-50) *(cuotaHora*3))
    3. endif
c. endif
d. Fin Método calcularSueldo

```

Explicación:

5. Método calcularSueldo().
 - a. Si horasTrab <= 40 entonces:
 1. Calcula el sueldo de la forma simple.
 - b. Si no:
 1. Si horasTrab <= 50 entonces:
 - a. Calcula el sueldo de la forma doble.
 2. Si no:
 - a. Calcula el sueldo de la forma triple.
 3. Fin del if.
 - c. Fin del if.
 - d. Fin Método calcularSueldo.

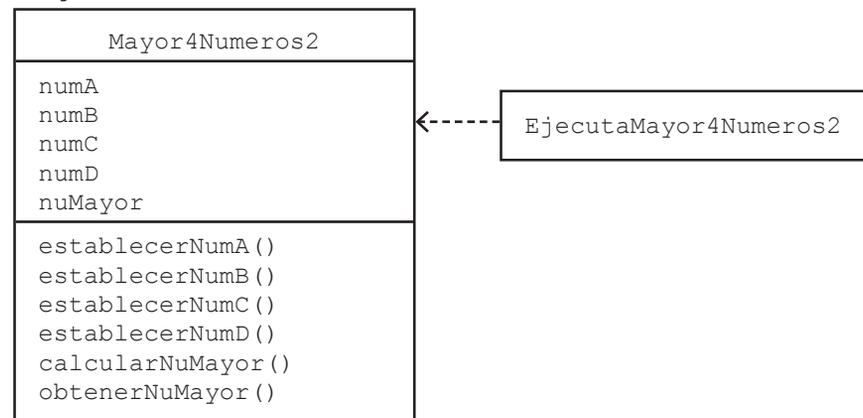
Ejercicio 10.4.3

Elaborar un algoritmo que lea cuatro números y calcule e imprima el mayor. Se supone que son números diferentes. Utilizar if-then-else y AND.

A continuación se presenta el algoritmo de la solución:

(Primero hágalo usted; después compare la solución)

Diagrama de clases



Algoritmo MAYOR 4 NUMEROS

Clase Mayor4Numeros2

1. Declarar datos
 numA, numB, numC, numD, nuMayor: Entero
2. Método establecerNumA(a: Entero)
 - a. numA = a
 - b. Fin Método establecerNumA
3. Método establecerNumB(b: Entero)
 - a. numB = b
 - b. Fin Método establecerNumB
4. Método establecerNumC(c: Entero)
 - a. numC = c
 - b. Fin Método establecerNumC
5. Método establecerNumD(d: Entero)
 - a. numD = d
 - b. Fin Método establecerNumD
6. Método calcularNuMayor()
 - a. if (numA > numB)AND(numA > numC)AND(numA > numD) then
 1. nuMayor = numA
 - b. else
 1. if (numB > numC)AND(numB > numD) then
 - a. nuMayor = numB
 2. else
 - a. if numC > numD then
 1. nuMayor = numC
 - b. else
 1. nuMayor = numD
 - c. endif
 3. endif
 - c. endif
 - d. Fin Método calcularNuMayor
7. Método obtenerNuMayor(): Entero
 - a. return nuMayor
 - b. Fin Método obtenerNuMayor

Fin Clase Mayor4Numeros2

Clase EjecutaMayor4Numeros2

1. Método principal()
 - a. Declarar variables
 n1, n2, n3, n4: Real
 - b. Declarar, crear e iniciar objeto
 Mayor4Numeros2 objMayor4Numeros= new Mayor4Numeros2()
 - c. Solicitar número 1, número 2, número 3, número 4

```

d. Leer n1, n2, n3, n4
e. Establecer objMayor4Numeros.establecerNumA(n1)
    objMayor4Numeros.establecerNumB(n2)
    objMayor4Numeros.establecerNumC(n3)
    objMayor4Numeros.establecerNumD(n4)
f. Calcular objMayor4Numeros.calcularNuMayor()
g. Imprimir objMayor4Numeros.obtenerNuMayor()
h. Fin Método principal
Fin Clase EjecutaMayor4Numeros2
Fin

```



En la zona de descarga de la Web del libro está disponible:

Programa en Java: Mayor4Numeros2.java y EjecutaMayor4Numeros2.java

Explicación:

El algoritmo tiene dos clases: la Clase Mayor4Numeros2 y la Clase EjecutaMayor4Numeros2.

En la Clase Mayor4Numeros2:

1. Se declaran los datos que representan la estructura de la clase:
 - numA, numB, numC, numD para cada uno de los 4 números.
 - nuMayor para número mayor.
2. Método establecerNumA(a: Entero).
 - Recibe en el parámetro a el valor que luego coloca en el dato numA.
3. Método establecerNumB(b: Entero).
 - Recibe en el parámetro b el valor que luego coloca en el dato numB.
4. Método establecerNumC(c: Entero).
 - Recibe en el parámetro c el valor que luego coloca en el dato numC.
5. Método establecerNumD(d: Entero).
 - Recibe en el parámetro d el valor que luego coloca en el dato numD.
6. Método calcularNuMayor().
 - Calcula el mayor en nuMayor:
 - a. Si (numA > numB) y (numA > numC) y (numA > numD) entonces:
 1. Coloca numA en nuMayor.
 - b. Si no:
 1. Si (numB > numC) y (numB > numD) entonces:
 - a. Coloca numB en nuMayor.
 2. Si no:
 - a. Si numC > numD entonces:
 1. Coloca numC en nuMayor.
 - b. Si no:
 1. Coloca numD en nuMayor.
 - c. Fin del if.
 3. Fin del if.

- c. Fin del if.
 - d. Fin Método calcularNuMayor.
7. Método obtenerNuMayor() Entero.
- a. Retorna nuMayor.
- Fin de la Clase Mayor4Numeros2.

En la Clase EjecutaMayor4Numeros2, en el Método principal():

- a. Se declaran las variables:
n1, n2, n3, n4 para leer los 4 números.
 - b. Se declara el objeto objMayor4Numeros, usando como base a la Clase Mayor4Numeros2. Dicho objeto se crea e inicializa mediante el constructor por defecto Mayor4Numeros2().
 - c. Se solicitan número 1, número 2, número 3 y número 4.
 - d. Se leen en n1, n2, n3, n4.
 - e. Se llama al Método establecerNumA(n1) del objeto objMayor4Numeros para colocar el valor de n1 en el dato numA.
Se llama al Método establecerNumB(n2) del objeto objMayor4Numeros para colocar el valor de n2 en el dato numB.
Se llama al Método establecerNumC(n3) del objeto objMayor4Numeros para colocar el valor de n3 en el dato numC.
Se llama al Método establecerNumD(n4) del objeto objMayor4Numeros para colocar el valor de n4 en el dato numD.
 - f. Se llama al Método calcularNuMayor() del objeto objMayor4Numeros para calcular el mayor.
 - g. Se llama al Método obtenerNuMayor() del objeto objMayor4Numeros para acceder e imprimir el valor del dato nuMayor.
 - h. Fin del Método principal.
- Fin de la Clase EjecutaMayor4Numeros2.
Fin del algoritmo.

Tabla 10.1: ejercicios resueltos disponibles en la zona de descarga del capítulo 10 de la Web del libro.

Ejercicio	Descripción
Ejercicio 10.4.4	Obtiene el mayor de 5 números
Ejercicio 10.4.5	Imprime el tipo de ángulo que es
Ejercicio 10.4.6	Realiza cálculos con la segunda ley de Newton
Ejercicio 10.4.7	Realiza cálculos de un cliente

10.5 Ejercicios propuestos

Como ejercicios propuestos para este capítulo se recomiendan, del capítulo 4, algunos de los ejercicios resueltos que no fueron incluidos en éste; además, todos los ejercicios propuestos en dicho capítulo.

10.6 Resumen de conceptos que debe dominar

- Cómo diseñar algoritmos orientados a objetos aplicando las estructuras de selección:
 - if-then-else (selección doble)
 - if-then (selección simple)
 - switch (selección múltiple)

10.7 Contenido de la página Web de apoyo

El material marcado con asterisco () sólo está disponible para docentes.*

10.7.1 Resumen gráfico del capítulo

10.7.2 Autoevaluación

10.7.3 Programas en Java

10.7.4 Ejercicios resueltos

10.7.5 Power Point para el profesor (*)

Programación orientada a objetos aplicando las estructuras de repetición

Contenido

- 11.1 Diseño de algoritmos OO usando la repetición do...while
 - 11.1.1 Contadores y acumuladores
 - 11.1.2 Ejercicios resueltos para do...while
 - 11.1.3 Ejercicios propuestos para do...while
- 11.2 Diseño de algoritmos OO usando la repetición for
 - 11.2.1 Ejercicios resueltos para for
 - 11.2.2 Ejercicios propuestos para for
- 11.3 Diseño de algoritmos OO usando la repetición while
 - 11.3.1 Ejercicios resueltos para while
 - 11.3.2 Ejercicios propuestos para while
- 11.4 Resumen de conceptos que debe dominar
- 11.5 Contenido de la página Web de apoyo
 - El material marcado con asterisco (*) sólo está disponible para docentes.*
 - 11.5.1 Resumen gráfico del capítulo
 - 11.5.2 Autoevaluación
 - 11.5.3 Programas en Java
 - 11.5.4 Ejercicios resueltos
 - 11.5.5 Power Point para el profesor (*)

Objetivos del capítulo

- Aprender a usar las estructuras de repetición do...while, for y while en el diseño de algoritmos orientados a objetos.
- Aplicar lo aprendido en la solución de ejercicios resueltos y propuestos.

Competencias

- Competencia general del capítulo
 - *Analizar problemas y diseñar algoritmos que los solucionen aplicando la arquitectura orientada a objetos y la repetición do...while, for y while.*
- Competencias específicas del capítulo
 - Diseña algoritmos orientados a objetos aplicando la repetición do...while.
 - Elabora algoritmos orientados a objetos aplicando la repetición for.
 - Desarrolla algoritmos orientados a objetos aplicando la repetición while.

Introducción

Con el estudio del capítulo anterior, usted ya domina los conceptos básicos de la programación orientada a objetos aplicando las estructuras de selección y sabe cómo diseñar algoritmos usando dichas estructuras.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos mediante la programación orientada a objetos aplicando las estructuras de repetición.

Se explica cómo diseñar algoritmos orientados a objetos usando las estructuras de control de repetición que estudiamos en el capítulo 5, pero ahora inmersas en la arquitectura orientada a objetos, que se estudió en los capítulos 8, 9 y 10.

Recordemos que la repetición `do...while` sirve para plantear situaciones en las que una acción o conjunto de acciones se repetirán mientras se cumpla una condición en un rango de 1 a n veces, la repetición `for` sirve para plantear situaciones en las que una acción o conjunto de acciones se repetirán un número de veces conocido de antemano y la repetición `while` sirve para plantear situaciones en las que una acción o conjunto de acciones se repetirán mientras se cumpla una condición en un rango de cero a n veces.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejercite estudiando los problemas planteados en los ejercicios resueltos y propuestos. Al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro, luego verifique sus resultados con los del libro, analice las diferencias y vea sus errores. Al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no está igual que el del libro, no necesariamente está mal. Usted debe ir aprendiendo a analizar las diferencias y a comprender que a veces, aunque haya diferencias, las dos soluciones están correctas.

En el siguiente capítulo se estudia la estructura de datos denominada arreglos, inmersa en la arquitectura orientada a objetos.

11.1 Diseño de algoritmos OO usando la repetición `do...while`

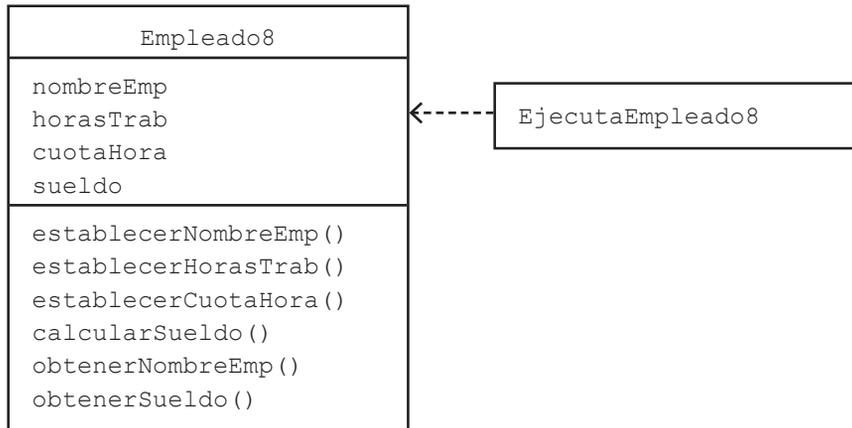
En este punto se utilizará la estructura de repetición `do...while` en pseudocódigo, la misma que se estudió en el apartado 5.1 del capítulo 5, pero ahora aplicada conjuntamente con el diagrama de clases y los conceptos de la programación orientada a objetos, es decir, en el diseño de algoritmos orientados a objetos.

Ejemplo:

Elaborar un algoritmo que calcule e imprima el sueldo de varios empleados. Cada empleado se tratará en forma similar al primer problema que planteamos en el capítulo 3, cuando estudiamos la secuenciación.

A continuación se presenta el algoritmo de la solución:

Diagrama de clases



Algoritmo CALCULAR SUELDO DE VARIOS EMPLEADOS

Clase Empleado8

1. Declarar datos

```

nombreEmp: Cadena
horasTrab: Entero
cuotaHora: Real
sueldo: Real
  
```

2. Método establecerNombreEmp(nom: Cadena)

```

a. nombreEmp = nom
b. Fin Método establecerNombreEmp
  
```

3. Método establecerHorasTrab(horasTr: Entero)

```

a. horasTrab = horasTr
b. Fin Método establecerHorasTrab
  
```

4. Método establecerCuotaHora(cuotaHr: Real)

```

a. cuotaHora = cuotaHr
b. Fin Método establecerCuotaHora
  
```

5. Método calcularSueldo()

```

a. sueldo = horasTrab * cuotaHora
b. Fin Método calcularSueldo
  
```

6. Método obtenerNombreEmp(): Cadena

```

a. return nombreEmp
b. Fin Método obtenerNombreEmp
  
```

7. Método obtenerSueldo(): Real

```

a. return sueldo
b. Fin Método obtenerSueldo
  
```

Fin Clase Empleado8

```

Clase EjecutaEmpleado8
1. Método principal()
  a. Declarar variables
    nomEmp: Cadena
    hrsTra: Entero
    cuoHr: Real
    desea: Carácter
  b. do
    1. Declarar, crear e iniciar objeto
      Empleado8 objEmpleado = new Empleado8()
    2. Solicitar nombre, número de horas trabajadas,
      cuota por hora
    3. Leer nomEmp, hrsTra, cuoHr
    4. Establecer
      objEmpleado.establecerNombreEmp(nomEmp)
      objEmpleado.establecerHorasTrab(hrsTra)
      objEmpleado.establecerCuotaHora(cuoHr)
    5. Calcular objEmpleado.calcularSueldo()
    6. Imprimir objEmpleado.obtenerNombreEmp()
      objEmpleado.obtenerSueldo()
    7. Preguntar "¿Desea procesar otro empleado(S/N)?"
    8. Leer desea
  c. while desea == 'S'
  d. Fin Método principal
Fin Clase EjecutaEmpleado8
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Empleado8.java y EjecutaEmpleado8.java

Explicación:

El algoritmo tiene dos clases: la Clase Empleado8 y la Clase EjecutaEmpleado8.

En la Clase Empleado8:

1. Se declaran los datos que representan la estructura de la clase:
 nombreEmp para el nombre del empleado.
 horasTrab para las horas trabajadas del empleado.
 cuotaHora para la cuota por hora.
 sueldo para el sueldo del empleado.
2. Método establecerNombreEmp(nom: Cadena).
 Recibe en el parámetro nom el valor que luego coloca en el dato nombreEmp.
3. Método establecerHorasTrab(horasTr: Entero).
 Recibe en el parámetro horasTr el valor que luego coloca en el dato horasTrab.
4. Método establecerCuotaHora(cuotaHr: Real).
 Recibe en el parámetro cuotaHr el valor que luego coloca en el dato cuotaHora.

5. Método `calcularSueldo()`.
Calcula el sueldo del empleado.
 6. Método `obtenerNombreEmp()` Cadena.
Retorna `nombreEmp` (nombre del empleado).
 7. Método `obtenerSueldo()` Real.
Retorna sueldo.
- Fin de la Clase `Empleado8`

En la Clase `EjecutaEmpleado8`, en el Método `principal()`:

- a. Se declaran las variables:
 - `nomEmp` para leer el nombre del empleado.
 - `hrsTra` para leer las horas trabajadas.
 - `cuoHr` para leer la cuota por hora.
 - `desea` para controlar el ciclo repetitivo.
 - b. Inicia ciclo `do`:
 1. Se declara el objeto `objEmpleado`, usando como base a la Clase `Empleado8`. Dicho objeto se crea e inicializa mediante el constructor por defecto `Empleado8()`.
Observe que cada vez que entra al ciclo crea un nuevo objeto empleado.
 2. Se solicitan el nombre, número de horas trabajadas y cuota por hora.
 3. Se leen en `nomEmp`, `hrsTra`, `cuoHr`.
 4. Se llama al Método `establecerNombreEmp(nomEmp)` del objeto `objEmpleado` para colocar el valor de `nomEmp` en el dato `nombreEmp`.
Se llama al Método `establecerHorasTrab(hrsTra)` del objeto `objEmpleado` para colocar el valor de `hrsTra` en el dato `horasTrab`.
Se llama al Método `establecerCuotaHora(cuoHr)` del objeto `objEmpleado` para colocar el valor de `cuoHr` en el dato `cuotaHora`.
 5. Se llama al Método `calcularSueldo()` del objeto `objEmpleado` para calcular el sueldo.
 6. Se llama al Método `obtenerNombreEmp()` del objeto `objEmpleado` para acceder e imprimir el valor del dato `nombreEmp`.
Se llama al Método `obtenerSueldo()` del objeto `objEmpleado` para acceder e imprimir el valor del dato `sueldo`.
 7. Se pregunta "¿Desea procesar otro empleado(S/N)?".
 8. Se lee la respuesta en `desea`.
 - c. Fin del ciclo (`while desea == 'S'`). Si se cumple regresa al `do`; si no, se sale del ciclo.
 - d. Fin del Método `principal`.
- Fin de la Clase `EjecutaEmpleado8`.
Fin del algoritmo.

11.1.1 Contadores y acumuladores

Utilizando el concepto de contadores y acumuladores en el problema anterior, ahora se requiere elaborar un algoritmo que permita procesar los empleados e imprima un reporte que tenga el siguiente formato:

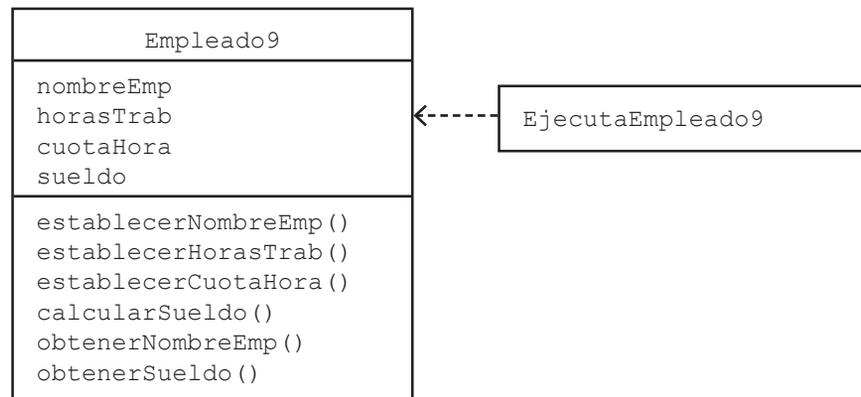
```

                REPORTE DE EMPLEADOS
    NOMBRE                SUELDO
    -----
XXXXXXXXXXXXXXXXXXXXXXXXX  99,999.99
XXXXXXXXXXXXXXXXXXXXXXXXX  99,999.99
    ---                    ---
    ---                    ---

XXXXXXXXXXXXXXXXXXXXXXXXX  99,999.99
TOTAL  999 EMPLEADOS      999,999.99
  
```

A continuación se presenta el algoritmo de la solución:

Diagrama de clases



Algoritmo CALCULAR SUELDO DE VARIOS EMPLEADOS

Clase Empleado9

1. Declarar datos

```

nombreEmp: Cadena
horasTrab: Entero
cuotaHora: Real
sueldo: Real
  
```

2. Método establecerNombreEmp(nom: Cadena)

```

a. nombreEmp = nom
b. Fin Método establecerNombreEmp
  
```

3. Método establecerHorasTrab(horasTr: Entero)

```

a. horasTrab = horasTr
b. Fin Método establecerHorasTrab
  
```

```

4. Método establecerCuotaHora(cuotaHr: Real)
  a. cuotaHr = cuotaHr
  b. Fin Método establecerCuotaHora

5. Método calcularSueldo()
  a. sueldo = horasTrab * cuotaHr
  b. Fin Método calcularSueldo

6. Método obtenerNombreEmp(): Cadena
  a. return nombreEmp
  b. Fin Método obtenerNombreEmp

7. Método obtenerSueldo(): Real
  a. return sueldo
  b. Fin Método obtenerSueldo
Fin Clase Empleado9

Clase EjecutaEmpleado9
1. Método principal()
  a. Declarar variables
     nomEmp: Cadena
     hrsTra, totEmpleados: Entero
     cuoHr, totSueldos: Real
     desea: Carácter
  b. Imprimir encabezado
  c. totEmpleados = 0
     totSueldos = 0
  d. do
     1. Declarar, crear e iniciar objeto
        Empleado9 objEmpleado = new Empleado9()
     2. Solicitar nombre, número de horas trabajadas,
        cuota por hora
     3. Leer nomEmp, hrsTra, cuoHr
     4. Establecer
        objEmpleado.establecerNombreEmp(nomEmp)
        objEmpleado.establecerHorasTrab(hrsTra)
        objEmpleado.establecerCuotaHora(cuoHr)
     5. Calcular objEmpleado.calcularSueldo()
     6. Imprimir objEmpleado.obtenerNombreEmp()
        objEmpleado.obtenerSueldo()
     7. totEmpleados = totEmpleados + 1
        totSueldos = totSueldos + objEmpleado.obtenerSueldo()
     8. Preguntar "¿Desea procesar otro empleado(S/N)?"
     9. Leer desea
  e. while desea == 'S'
  f. Imprimir totEmpleados, totSueldos
  g. Fin Método principal
Fin Clase EjecutaEmpleado9
Fin

```

En la zona de descarga de la Web del libro está disponible:
Programa en Java: Empleado9.java y EjecutaEmpleado9.java



Explicación:

El algoritmo tiene dos clases: la Clase Empleado9 y la Clase EjecutaEmpleado9.

En la Clase Empleado9:

1. Se declaran los datos que representan la estructura de la clase:
nombreEmp para el nombre del empleado.
horasTrab para las horas trabajadas del empleado.
cuotaHora para la cuota por hora.
sueldo para el sueldo del empleado.
2. Método establecerNombreEmp(nom: Cadena).
Recibe en el parámetro nom el valor que luego coloca en el dato nombreEmp.
3. Método establecerHorasTrab(horasTr: Entero).
Recibe en el parámetro horasTr el valor que luego coloca en el dato horasTrab.
4. Método establecerCuotaHora(cuotaHr: Real).
Recibe en el parámetro cuotaHr el valor que luego coloca en el dato cuotaHora.
5. Método calcularSueldo().
Calcula el sueldo del empleado
6. Método obtenerNombreEmp() Cadena.
Retorna nombreEmp (nombre del empleado)
7. Método obtenerSueldo() Real.
Retorna sueldo.

Fin de la Clase Empleado9.

En la Clase EjecutaEmpleado9, en el Método principal():

- a. Se declaran las variables:
nomEmp para leer el nombre del empleado.
hrsTra para leer las horas trabajadas.
cuoHr para leer la cuota por hora.
totEmpleados para contar el total de empleados.
totSueldos para calcular el total de sueldos de todos los empleados.
desea para controlar el ciclo repetitivo.
- b. Imprime el encabezado.
- c. Inicia totEmpleados y totSueldos en 0.
- d. Inicia ciclo do:
 1. Se declara el objeto objEmpleado, usando como base a la Clase Empleado9. Dicho objeto se crea e inicializa mediante el constructor por defecto Empleado9().
Observe que cada vez que entra al ciclo crea un nuevo objeto empleado.
 2. Se solicitan el nombre, número de horas trabajadas y cuota por hora.
 3. Se leen en nomEmp, hrsTra, cuoHr.
 4. Se llama al Método establecerNombreEmp(nomEmp) del objeto objEmpleado para colocar el valor de nomEmp en el dato nombreEmp.
Se llama al Método establecerHorasTrab(hrsTra) del objeto objEmpleado para colocar el valor de hrsTra en el dato horasTrab.

- Se llama al Método establecerCuotaHora(cuoHr) del objeto objEmpleado para colocar el valor de cuoHr en el dato cuotaHora.
5. Se llama al Método calcularSueldo() del objeto objEmpleado para calcular el sueldo.
 6. Se llama al Método obtenerNombreEmp() del objeto objEmpleado para acceder e imprimir el valor del dato nombreEmp.
Se llama al Método obtenerSueldo() del objeto objEmpleado para acceder e imprimir el valor del dato sueldo.
 7. Se incrementa totEmpleados en 1.
Se incrementa totSueldos con el sueldo del empleado, al que se accede llamando al Método obtenerSueldo() del objeto objEmpleado.
 8. Se pregunta “¿Desea procesar otro empleado(S/N)?”.
 9. Se lee la respuesta en desea.
- e. Fin del ciclo (while desea == 'S'). Si se cumple regresa al do; si no, se sale del ciclo.
- f. Imprime totEmpleados y totSueldos.
- g. Fin del Método principal.
- Fin de la Clase EjecutaEmpleado9.
- Fin del algoritmo.

11.1.2 Ejercicios resueltos para do...while

En este punto se presentan algunos de los ejercicios resueltos en el apartado 5.1 del capítulo 5. Es decir, el mismo problema ya lo resolvimos aplicando la lógica básica de la programación en el capítulo 5 y aquí lo estamos resolviendo aplicando la metodología de la programación orientada a objetos.

Cabe aclarar que, por la naturaleza de la repetición do...while, los problemas que se plantean en este capítulo tienen una orientación más administrativa porque esta estructura es más natural para resolver problemas de esta índole. Sin embargo, en el punto siguiente (for), donde los problemas tienen una orientación más hacia ingeniería, también utilizaremos el do... while para solucionar ese tipo de problemas.

Ejercicio 11.1.2.1

Una empresa vende hojas de hielo seco, con las condiciones siguientes:

- Si el cliente es tipo 1 se le descuenta el 5 %
- Si el cliente es tipo 2 se le descuenta el 8 %
- Si el cliente es tipo 3 se le descuenta el 12 %
- Si el cliente es tipo 4 se le descuenta el 15 %

Cuando un cliente realiza una compra se generan los datos siguientes:

- Nombre del cliente
- Tipo de cliente (1, 2, 3, 4)
- Cantidad de hojas compradas
- Precio por hoja



Cabe aclarar que, por la naturaleza de la repetición do...while, los problemas que se plantean en este capítulo tienen una orientación más administrativa porque esta estructura es más natural para resolver problemas de esta índole. Sin embargo, en el punto siguiente (for), donde los problemas tienen una orientación más hacia ingeniería, también utilizaremos el do... while para solucionar ese tipo de problemas.

Elaborar un algoritmo que permita procesar varios clientes e imprima el reporte:

NOMBRE	REPORTE DE CLIENTES		
	SUB.TOTAL	DESCUENTO	NETO A PAGAR
XXXXXXXXXXXXXXXXXXXX	99,999.99	99,999.99	99,999.99
XXXXXXXXXXXXXXXXXXXX	99,999.99	99,999.99	99,999.99
---	---	---	---
XXXXXXXXXXXXXXXXXXXX	99,999.99	99,999.99	99,999.99
TOTAL 999 CLIENTES	999,999.99	999,999.99	999,999.99

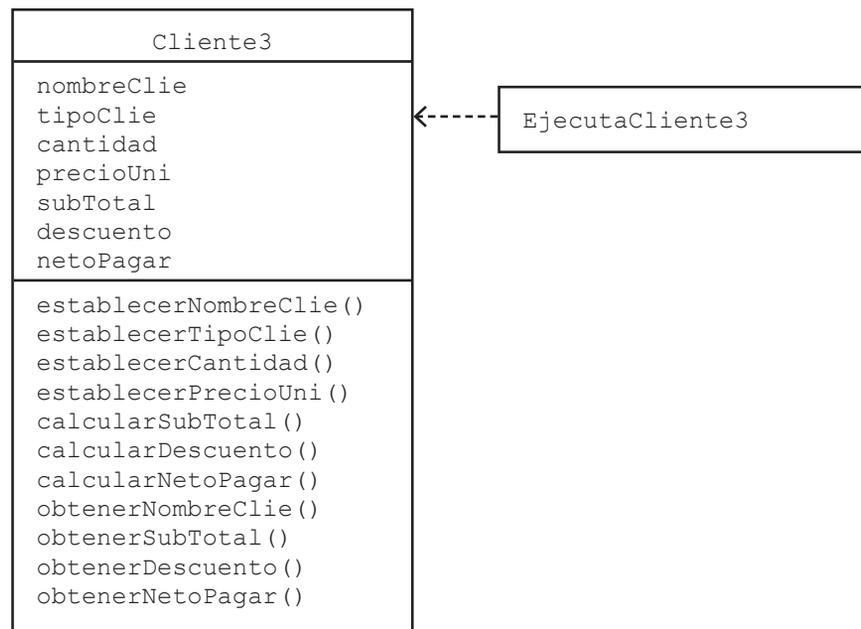
Cálculos:

- Subtotal = Cantidad de hojas X Precio por hoja
- Descuento es el porcentaje correspondiente del Subtotal por pagar
- Neto por pagar = Subtotal – Descuento

A continuación se presenta el algoritmo de la solución:

(Primero hágalo usted; después compare la solución)

Diagrama de clases



Algoritmo CLIENTES HOJAS HIELO SECO

Clase Cliente3

1. Declarar datos

```

nombreClie: Cadena
tipoClie, cantidad: Entero
precioUni, subTotal, descuento, netoPagar: Real
  
```

```
2. Método establecerNombreClie(nom: Cadena)
  a. nombreClie = nom
  b. Fin Método establecerNombreClie

3. Método establecerTipoClie(tip: Entero)
  a. tipoClie = tip
  b. Fin Método establecerTipoClie

4. Método establecerCantidad(can: Entero)
  a. cantidad = can
  b. Fin Método establecerCantidad

5. Método establecerPrecioUni(pre: Real)
  a. precioUni = pre
  b. Fin Método establecerPrecioUni

6. Método calcularSubTotal()
  a. subTotal = cantidad * precioUni
  b. Fin Método calcularSubTotal

7. Método calcularDescuento()
  a. switch tipoClie
    1: descuento = subTotal * 0.05
    2: descuento = subTotal * 0.08
    3: descuento = subTotal * 0.12
    4: descuento = subTotal * 0.15
  b. endswitch
  c. Fin Método calcularDescuento

8. Método calcularNetoPagar()
  a. netoPagar = subTotal - descuento
  b. Fin Método calcularNetoPagar

9. Método obtenerNombreClie(): Cadena
  a. return nombreClie
  b. Fin Método obtenerNombreClie

10. Método obtenerSubTotal(): Real
  a. return subTotal
  b. Fin Método obtenerSubTotal

11. Método obtenerDescuento(): Real
  a. return descuento
  b. Fin Método obtenerDescuento

12. Método obtenerNetoPagar(): Real
  a. return netoPagar
  b. Fin Método obtenerNetoPagar
Fin Clase Cliente3
```

```

Clase EjecutaCliente3
1. Método principal()
  a. Declarar variables
      nomCli: Cadena
      tiCli, cant, totClientes: Entero
      preUni, totSubTot, totDescuento, totNeto: Real
      desea: Carácter
  b. Imprimir encabezado
  c. totClientes = 0
      totSubTot = 0
      totDescuento = 0
      totNeto = 0
  d. do
      1. Declarar, crear e iniciar objeto
          Cliente3 objCliente = new Cliente3()
      2. Solicitar nombre, tipo cliente,
          cantidad, precio unitario
      3. Leer nomCli, tiCli, cant, preUni
      4. Establecer
          objCliente.establecerNombreClie(nomCli)
          objCliente.establecerTipoClie(tiCli)
          objCliente.establecerCantidad(cant)
          objCliente.establecerPrecioUni(preUni)
      5. Calcular objCliente.calcularSubTotal()
          objCliente.calcularDescuento()
          objCliente.calcularNetoPagar()
      6. Imprimir objCliente.obtenerNombreClie()
          objCliente.obtenerSubTotal()
          objCliente.obtenerDescuento()
          objCliente.obtenerNetoPagar()
      7. totClientes = totClientes + 1
          totSubTot = totSubTot +
              objCliente.obtenerSubTotal()
          totDescuento = totDescuento +
              objCliente.obtenerDescuento()
          totNeto = totNeto +
              objCliente.obtenerNetoPagar()
      8. Preguntar "¿Desea procesar otro cliente(S/N)?"
      9. Leer desea
  e. while desea == 'S'
  f. Imprimir totClientes, totSubTot,
      totDescuento, totNeto
  g. Fin Método principal
Fin Clase EjecutaCliente3
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Cliente3.java y EjecutaCliente3.java

Explicación:

El algoritmo tiene dos clases: la Clase Cliente3 y la Clase EjecutaCliente3.

En la Clase Cliente3:

1. Se declaran los datos que representan la estructura de la clase:
nombreClie para el nombre del cliente.
tipoClie para el tipo de cliente.
cantidad para la cantidad de hojas compradas.
precioUni para el precio unitario.
subTotal para el subtotal.
descuento para el descuento.
netoPagar para el neto a pagar.
 2. Método establecerNombreClie(nom: Cadena).
Recibe en el parámetro nom el valor que luego coloca en el dato nombreClie.
 3. Método establecerTipoClie(tipo: Entero).
Recibe en el parámetro tipo el valor que luego coloca en el dato tipoClie.
 4. Método establecerCantidad(can: Entero).
Recibe en el parámetro can el valor que luego coloca en el dato cantidad.
 5. Método establecerPrecioUni(pre: Real).
Recibe en el parámetro pre el valor que luego coloca en el dato precioUni.
 6. Método calcularSubTotal().
Calcula el subtotal por pagar.
 7. Método calcularDescuento().
Calcula el descuento.
 - a. switch tipoClie:
Si tipoClie==1: descuento = subTotal * 0.05.
Si tipoClie==2: descuento = subTotal * 0.08.
Si tipoClie==3: descuento = subTotal * 0.12.
Si tipoClie==4: descuento = subTotal * 0.15.
 - b. endswitch.
 - c. Fin Método calcularDescuento.
 8. Método calcularNetoPagar().
Calcula el neto por pagar.
 9. Método obtenerNombreClie() Cadena.
Retorna nombreClie.
 10. Método obtenerSubTotal() Real.
Retorna subTotal.
 11. Método obtenerDescuento() Real.
Retorna descuento.
 12. Método obtenerNetoPagar() Real.
Retorna netoPagar.
- Fin de la Clase Cliente3.

En la Clase EjecutaCliente3, en el Método principal():

- a. Se declaran las variables:
nomCli para leer el nombre del cliente.
tiCli para leer el tipo de cliente.
cant para leer la cantidad de hojas compradas.
preUni para leer el precio unitario por hoja.

totClientes para leer el total de clientes.
totSubTot calcular el total de los subtotales.
totDescuento calcular el total de los descuentos.
totNeto calcular el total de los netos por pagar
desea para controlar el ciclo repetitivo.

- b. Imprime el encabezado.
- c. Inicia los totalizadores en 0.
- d. Inicia ciclo do:
 1. Se declara el objeto objCliente, usando como base a la Clase Cliente3. Dicho objeto se crea e inicializa mediante el constructor por defecto Cliente3().
Observe que cada vez que entra al ciclo crea un nuevo objeto cliente.
 2. Se solicita nombre, tipo cliente, cantidad, precio unitario.
 3. Lee en nomCli, tiCli, cant, preUni.
 4. Se llama al Método establecerNombreClie(nomCli) del objeto objCliente para colocar el valor de nomCli en el dato nombreClie.
Se llama al Método establecerTipoClie(tiCli) del objeto objCliente para colocar el valor de tiCli en el dato tipoClie.
Se llama al Método establecerCantidad(cant) del objeto objCliente para colocar el valor de cant en el dato cantidad.
Se llama al Método establecerPrecioUni(preUni) del objeto objCliente para colocar el valor de preUni en el dato precioUni.
 5. Se llama al Método calcularSubTotal() del objeto objCliente para calcular el subtotal.
Se llama al Método calcularDescuento() del objeto objCliente para calcular el descuento.
Se llama al Método calcularNetoPagar() del objeto objCliente para calcular el neto por pagar.
 6. Se llama al Método obtenerNombreClie() del objeto objCliente para acceder e imprimir el valor del dato nombreClie.
Se llama al Método obtenerSubTotal() del objeto objCliente para acceder e imprimir el valor del dato subTotal.
Se llama al Método obtenerDescuento() del objeto objCliente para acceder e imprimir el valor del dato descuento.
Se llama al Método obtenerNetoPagar() del objeto objCliente para acceder e imprimir el valor del dato netoPagar.
 7. Incrementa totClientes en 1.
Se incrementa totSubTot con el subtotal del cliente, al que se accede llamando al Método obtenerSubTotal() del objeto objCliente.
Se incrementa totDescuento con el descuento del cliente, al que se accede llamando al Método obtenerDescuento() del objeto objCliente.
Se incrementa totNeto con el neto por pagar del cliente, al que se accede llamando al Método obtenerNetoPagar() del objeto objCliente.
 8. Pregunta “¿Desea procesar otro cliente(S/N)?”.
 9. Se lee la respuesta en desea.
- e. Fin del ciclo (while desea == 'S'). Si se cumple regresa al do; si no, se sale del ciclo.

f. Imprime totClientes, totSubTot, totDescuento, totNeto.

g. Fin Método principal.

Fin de la Clase EjecutaCliente3.

Fin del algoritmo.

Ejercicio 11.1.2.2

Elaborar un algoritmo que proporcione el siguiente reporte:

NOMBRE	ANÁLISIS DE CALIFICACIONES				PROMEDIO
	CAL.1	CAL.2	CAL.3	CAL.4	
XXXXXXXXXXXXXXXXXXXXX	99.99	99.99	99.99	99.99	99.99
XXXXXXXXXXXXXXXXXXXXX	99.99	99.99	99.99	99.99	99.99
---	---	---	---	---	---
XXXXXXXXXXXXXXXXXXXXX	99.99	99.99	99.99	99.99	99.99
PROMEDIOS GENERALES	99.99	99.99	99.99	99.99	99.99

A partir de que se tiene el nombre y la calificación 1, 2, 3 y 4 de varios alumnos, el promedio se obtiene sumando las cuatro calificaciones y dividiendo el resultado entre cuatro.

El promedio general de cada calificación se calcula sumando las calificaciones de todos los alumnos y dividiendo el resultado entre el número de alumnos.

A continuación se presenta el algoritmo de la solución:

(Primero hágalo usted; después compare la solución)

Diagrama de clases



```
Algoritmo CALCULA PROMEDIO DE VARIOS ALUMNOS
Clase Alumno5
1. Declarar datos
    nombreAlum: Cadena
    calif1: Real
    calif2: Real
    calif3: Real
    calif4: Real
    promedio: Real

2. Método establecerNombreAlum(nom: Cadena)
    a. nombreAlum = nom
    b. Fin Método establecerNombreAlum

3. Método establecerCalif1(ca1: Real)
    a. calif1 = ca1
    b. Fin Método establecerCalif1

4. Método establecerCalif2(ca2: Real)
    a. calif2 = ca2
    b. Fin Método establecerCalif2

5. Método establecerCalif3(ca3: Real)
    a. calif3 = ca3
    b. Fin Método establecerCalif3

6. Método establecerCalif4(ca4: Real)
    a. calif4 = ca4
    b. Fin Método establecerCalif4

7. Método calcularPromedio()
    a. promedio = (calif1+ calif2+ calif3+ calif4)/4
    b. Fin Método calcularPromedio

8. Método obtenerNombreAlum(): Cadena
    a. return nombreAlum
    b. Fin Método obtenerNombreAlum

9. Método obtenerCalif1(): Real
    a. return calif1
    b. Fin Método obtenerCalif1

10. Método obtenerCalif2(): Real
    a. return calif2
    b. Fin Método obtenerCalif2

11. Método obtenerCalif3(): Real
    a. return calif3
    b. Fin Método obtenerCalif3
```

12. Método obtenerCalif4(): Real
 - a. return calif4
 - b. Fin Método obtenerCalif4
 13. Método obtenerPromedio(): Real
 - a. return promedio
 - b. Fin Método obtenerPromedio
- Fin Clase Alumno5

Clase EjecutaAlumno5

1. Método principal()
 - a. Declarar variables
nombre: Cadena
totAlumnos: Entero
c1, c2, c3, c4, promCal1, promCal2,
promCal3, promCal4, promProm, totCal1,
totCal2, totCal3, totCal4, totProm: Real
desea: Carácter
 - b. Imprimir encabezado
 - c. totAlumnos = 0
totCal1 = 0, totCal2 = 0, totCal3 = 0,
totCal4 = 0, totProm = 0
 - d. do
 1. Declarar, crear e iniciar objeto
Alumno5 objAlumno = new Alumno5()
 2. Solicitar nombre del alumno, calificación 1,
calificación 2, calificación 3, calificación 4
 3. Leer nombre, c1, c2, c3, c4
 4. Establecer
objAlumno.establecerNombreAlum(nombre)
objAlumno.establecerCalif1(c1)
objAlumno.establecerCalif2(c2)
objAlumno.establecerCalif3(c3)
objAlumno.establecerCalif4(c4)
 5. Calcular objAlumno.calcularPromedio()
 6. Imprimir objAlumno.obtenerNombreAlum()
objAlumno.obtenerCalif1()
objAlumno.obtenerCalif2()
objAlumno.obtenerCalif3()
objAlumno.obtenerCalif4()
objAlumno.obtenerPromedio()
 7. totAlumnos = totAlumnos + 1
 8. totCal1 = totCal1 + objAlumno.obtenerCalif1()
totCal2 = totCal2 + objAlumno.obtenerCalif2()
totCal3 = totCal3 + objAlumno.obtenerCalif3()
totCal4 = totCal4 + objAlumno.obtenerCalif4()
totProm = totProm + objAlumno.obtenerPromedio()
 9. Preguntar "¿Desea procesar otro alumno (S/N)?"
 10. Leer desea

```

e. while desea == 'S'
f. promCal1 = totCal1 / totAlumnos
   promCal2 = totCal2 / totAlumnos
   promCal3 = totCal3 / totAlumnos
   promCal4 = totCal4 / totAlumnos
   promProm = totProm / totAlumnos
g. Imprimir promCal1, promCal2, promCal3,
   promCal4, promProm
h. Fin Método principal
Fin Clase EjecutaAlumno5
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Alumno5.java y EjecutaAlumno5.java

Explicación:

El algoritmo tiene dos clases: la Clase Alumno5 y la Clase EjecutaAlumno5.

En la Clase Alumno5:

1. Se declaran los datos que representan la estructura de la clase:
 nombreAlum para el nombre del alumno.
 calif1 para la calificación 1 del alumno.
 calif2 para la calificación 2 del alumno.
 calif3 para la calificación 3 del alumno.
 calif4 para la calificación 4 del alumno.
 promedio para el promedio del alumno.
2. Método establecerNombreAlum(nom: Cadena).
 Recibe en el parámetro nom el valor que luego coloca en el dato nombreAlum.
3. Método establecerCalif1(ca1: Real).
 Recibe en el parámetro ca1 el valor que luego coloca en el dato calif1.
4. Método establecerCalif2(ca2: Real).
 Recibe en el parámetro ca2 el valor que luego coloca en el dato calif2.
5. Método establecerCalif3(ca3: Real).
 Recibe en el parámetro ca3 el valor que luego coloca en el dato calif3.
6. Método establecerCalif4(ca4: Real).
 Recibe en el parámetro ca4 el valor que luego coloca en el dato calif4.
7. Método calcularPromedio().
 Calcula el promedio del alumno.
8. Método obtenerNombreAlum() Cadena.
 Retorna nombreAlum (nombre del alumno).
9. Método obtenerCalif1() Real.
 Retorna calif1.
10. Método obtenerCalif2() Real.
 Retorna calif2.
11. Método obtenerCalif3() Real.
 Retorna calif3.

12. Método obtenerCalif4() Real.
Retorna calif4.
 13. Método obtenerPromedio() Real.
Retorna promedio (promedio del alumno).
- Fin de la Clase Alumno5.

En la Clase EjecutaAlumno5, en el Método principal():

- a. Se declaran las variables:
 - nombre para leer el nombre del alumno.
 - c1 para leer la calificación 1 del alumno.
 - c2 para leer la calificación 2 del alumno.
 - c3 para leer la calificación 3 del alumno.
 - c4 para leer la calificación 4 del alumno.
 - totAlumnos para contar el total de alumnos.
 - promCal1, promCal2, promCal3, promCal4, promProm para los promedios.
 - totCal1, totCal2, totCal3, totCal4, totProm para los totalizadores.
 - desea para controlar el ciclo repetitivo.
- b. Imprime el encabezado.
- c. Inicia los totalizadores en 0.
- d. Inicia ciclo do:
 1. Se declara el objeto objAlumno, usando como base a la Clase Alumno5. Dicho objeto se crea e inicializa mediante el constructor por defecto Alumno5().
Observe que cada vez que entra al ciclo crea un nuevo objeto alumno.
 2. Se solicitan nombre del alumno, calificación 1, calificación 2, calificación 3 y calificación 4.
 3. Se leen en nombre, c1, c2, c3, c4.
 4. Se llama al Método establecerNombreAlum(nombre) del objeto objAlumno para colocar el valor de nombre en el dato nombreAlum.
Se llama al Método establecerCalif1(c1) del objeto objAlumno para colocar el valor de c1 en el dato calif1.
Se llama al Método establecerCalif2(c2) del objeto objAlumno para colocar el valor de c2 en el dato calif2.
Se llama al Método establecerCalif3(c3) del objeto objAlumno para colocar el valor de c3 en el dato calif3.
Se llama al Método establecerCalif4(c4) del objeto objAlumno para colocar el valor de c4 en el dato calif4.
 5. Se llama al Método calcularPromedio() del objeto objAlumno para calcular el promedio.
 6. Se llama al Método obtenerNombreAlum() del objeto objAlumno para acceder e imprimir el valor del dato nombreAlum.
Se llama al Método obtenerCalif1() del objeto objAlumno para acceder e imprimir el valor del dato calif1.
Se llama al Método obtenerCalif2() del objeto objAlumno para acceder e imprimir el valor del dato calif2.
Se llama al Método obtenerCalif3() del objeto objAlumno para acceder e imprimir el valor del dato calif3.

Se llama al Método `obtenerCalif4()` del objeto `objAlumno` para acceder e imprimir el valor del dato `calif4`.

Se llama al Método `obtenerPromedio()` del objeto `objAlumno` para acceder e imprimir el valor del dato `promedio`.

7. Incrementa `totAlumnos` en 1.
 8. Se incrementa `totCal1` con `calif1`, a la que se accede llamando al Método `obtenerCalif1()` del objeto `objAlumno`.
Se incrementa `totCal2` con `calif2`, a la que se accede llamando al Método `obtenerCalif2()` del objeto `objAlumno`.
Se incrementa `totCal3` con `calif3`, a la que se accede llamando al Método `obtenerCalif3()` del objeto `objAlumno`.
Se incrementa `totCal4` con `calif4`, a la que se accede llamando al Método `obtenerCalif4()` del objeto `objAlumno`.
Se incrementa `totProm` con `promedio`, al que se accede llamando al Método `obtenerPromedio()` del objeto `objAlumno`.
 9. Pregunta “¿Desea procesar otro alumno(S/N)?”.
 10. Se lee la respuesta en `desea`.
 - e. Fin del ciclo (`while desea == 'S'`). Si se cumple regresa al `do`; si no, se sale del ciclo.
 - f. Calcula los promedios `promCal1`, `promCal2`, `promCal3`, `promCal4` y `promProm`.
 - g. Imprime `promCal1`, `promCal2`, `promCal3`, `promCal4`, `promProm`.
 - h. Fin Método principal.
- Fin de la Clase `EjecutaAlumno5`.
Fin del algoritmo.

Tabla 11.1: ejercicios resueltos disponibles en la zona de descarga del capítulo 11 de la Web del libro.

Ejercicio	Descripción
Ejercicio 11.1.2.3	Procesa artículos con inflación
Ejercicio 11.1.2.4	Procesa la producción de varios obreros con dos ciclos

11.1.3 Ejercicios propuestos para `do...while`

Como ejercicios propuestos para este punto se recomiendan, del capítulo 5, del apartado de la repetición `do...while` (5.1), algunos de los ejercicios resueltos que no fueron incluidos aquí, además de todos los ejercicios propuestos en dicho apartado.

11.2 Diseño de algoritmos OO usando la repetición `for`

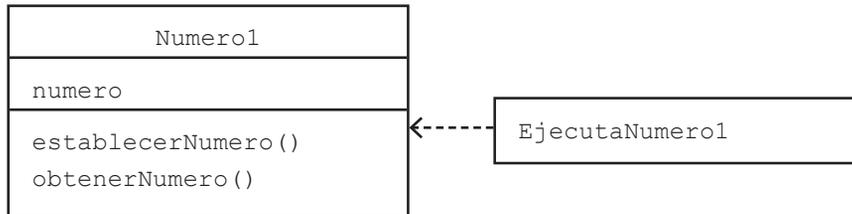
En este punto se utilizará la estructura de repetición `for` en pseudocódigo, misma que se estudió en el apartado 5.2 del capítulo 5, pero ahora aplicada conjuntamente con el diagrama de clases y los conceptos de la programación orientada a objetos, es decir, en el diseño de algoritmos orientados a objetos.

Ejemplo:

Elaborar un algoritmo que lea 20 números y que calcule e imprima el promedio de dichos números.

A continuación se presenta el algoritmo de la solución:

Diagrama de clases



Algoritmo PROMEDIO DE 20 NUMEROS

Clase Numerol

1. Declarar datos
 - numero: Entero
2. Método establecerNumero(nu: Entero)
 - a. numero = nu
 - b. Fin Método establecerNumero
3. Método obtenerNumero(): Entero
 - a. return numero
 - b. Fin Método obtenerNumero

Fin Clase Numerol

Clase EjecutaNumerol

1. Método principal()
 - a. Declarar variables
 - i, num, sumatoria: Entero
 - promedio: Real
 - b. sumatoria = 0
 - c. for i=1; i<=20; i++
 1. Declarar, crear e iniciar objeto
 - Numerol objNumero = new Numerol()
 2. Solicitar número
 3. Leer num
 4. Establecer objNumero.establecerNumero(num)
 5. sumatoria= sumatoria + objNumero.obtenerNumero()
 - d. endfor
 - e. promedio = sumatoria / 20
 - f. Imprimir promedio
 - g. Fin Método principal

Fin Clase EjecutaNumerol

Fin

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Numerol.java y EjecutaNumerol.java



Explicación:

El algoritmo tiene dos clases: la Clase `Numero1` y la Clase `EjecutaNumero1`.

En la Clase `Numero1`:

1. Se declaran los datos que representan la estructura de la clase:
numero para el número.
 2. Método `establecerNumero(nu: Entero)`.
Recibe en el parámetro `nu` el valor que luego coloca en el dato `numero`.
 3. Método `obtenerNumero()`.
Retorna `numero`.
- Fin de la Clase `Numero1`.

En la Clase `EjecutaNumero1`, en el Método `principal()`:

- a. Se declaran las variables:
num para leer el número.
i para controlar el ciclo repetitivo for.
sumatoria para calcular la sumatoria de los números.
promedio para calcular el promedio.
 - b. Inicia sumatoria en 0.
 - c. Inicia ciclo for desde `i=1` hasta 20 con incrementos de 1:
 1. Se declara el objeto `objNumero`, usando como base a la Clase `Numero1`. Dicho objeto se crea e inicializa mediante el constructor por defecto `Numero1()`. Observe que cada vez que entra al ciclo crea un nuevo objeto `numero`.
 2. Solicita el número.
 3. Lee en `num`.
 4. Se llama al Método `establecerNumero(num)` del objeto `objNumero` para colocar el valor de `num` en el dato `numero`.
 5. Se incrementa `sumatoria` con `numero`, al que se accede llamando al Método `obtenerNumero()` del objeto `objNumero`.
 - d. Fin del ciclo for.
 - e. Calcula promedio.
 - f. Imprime promedio.
 - g. Fin Método `principal`.
- Fin de la Clase `EjecutaNumero1`.
Fin del algoritmo.

11.2.1 Ejercicios resueltos para for

En este punto se presentan algunos de los ejercicios resueltos en el apartado 5.2 del capítulo 5. Es decir, el mismo problema ya lo resolvimos aplicando la lógica básica de la programación en el capítulo 5, y aquí lo estamos resolviendo aplicando la metodología de la programación orientada a objetos.

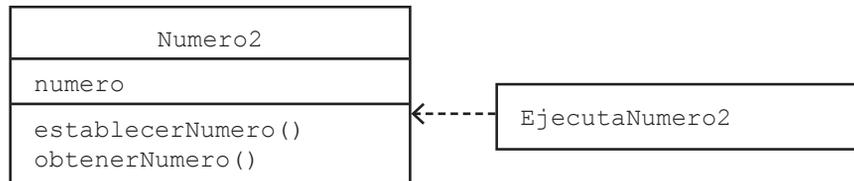
Ejercicio 11.2.1.1

Elaborar un algoritmo que solicite la cantidad de números por procesar y lea la respuesta en `N`. Luego, que lea los `N` números y calcule e imprima el promedio de dichos números.

A continuación se presenta el algoritmo de la solución:

(Primero hágalo usted; después compare la solución)

Diagrama de clases



Algoritmo PROMEDIO DE N NUMEROS

Clase Numero2

1. Declarar datos
 - numero: Entero
2. Método establecerNumero(nu: Entero)
 - a. numero = nu
 - b. Fin Método establecerNumero
3. Método obtenerNumero(): Entero
 - a. return numero
 - b. Fin Método obtenerNumero

Fin Clase Numero2

Clase EjecutaNumero2

1. Método principal()
 - a. Declarar variables
 - i, n, num, sumatoria: Entero
 - promedio: Real
 - b. Solicitar cantidad de números a procesar
 - c. Leer n
 - d. sumatoria = 0
 - e. for i=1; i<=n; i++
 1. Declarar, crear e iniciar objeto
 - Numero2 objNumero = new Numero2()
 2. Solicitar número
 3. Leer num
 4. Establecer objNumero.establecerNumero(num)
 5. sumatoria= sumatoria + objNumero.obtenerNumero()
 - f. endfor
 - g. promedio = sumatoria / n
 - h. Imprimir promedio
 - i. Fin Método principal

Fin Clase EjecutaNumero2

Fin

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Numero2.java y EjecutaNumero2.java



Explicación:

El algoritmo tiene dos clases: la Clase Numero2 y la Clase EjecutaNumero2.

En la Clase Numero2:

1. Se declaran los datos que representan la estructura de la clase:
numero para el número.
 2. Método establecerNumero(nu: Entero)
Recibe en el parámetro nu el valor que luego coloca en el dato numero.
 3. Método obtenerNumero() Entero.
Retorna numero.
- Fin de la Clase Numero2.

En la Clase EjecutaNumero2, en el Método principal():

- a. Se declaran las variables:
num para leer el número.
i para controlar el ciclo repetitivo for.
sumatoria para calcular la sumatoria de los números.
promedio para calcular el promedio.
n para leer la cantidad de números.
 - b. Solicita la cantidad de números por procesar.
 - c. Lee en n.
 - d. Inicia sumatoria en 0.
 - e. Inicia ciclo for desde i=1 hasta n con incrementos de 1:
 1. Se declara el objeto objNumero, usando como base a la Clase Numero2. Dicho objeto se crea e inicializa mediante el constructor por defecto Numero2().
Observe que cada vez que entra al ciclo crea un nuevo objeto numero.
 2. Solicita el número.
 3. Lee en num.
 4. Se llama al Método establecerNumero(num) del objeto objNumero para colocar el valor de num en el dato numero.
 5. Se incrementa sumatoria con numero, al que se accede llamando al Método obtenerNumero() del objeto objNumero.
 - f. Fin del ciclo for.
 - g. Calcula promedio.
 - h. Imprime promedio.
 - i. Fin Método principal.
- Fin de la Clase EjecutaNumero2.
Fin del algoritmo.

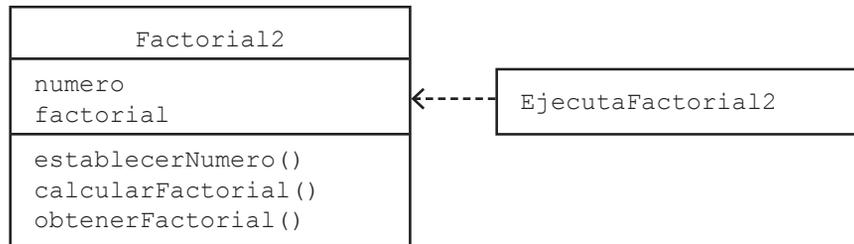
Ejercicio 11.2.1.2

Elabore un algoritmo que lea un valor N, entero y positivo, y que calcule e imprima su factorial. Por ejemplo, si se lee el 5, su factorial es el producto de $5*4*3*2*1$. El factorial de 0 es 1.

A continuación se presenta el algoritmo de la solución:

(Primero hágalo usted; después compare la solución)

Diagrama de clases



Algoritmo FACTORIAL

Clase Factorial2

1. Declarar datos

numero: Entero
factorial: Entero

2. Método establecerNumero(nu: Entero)

a. numero = nu
b. Fin Método establecerNumero

3. Método calcularFactorial()

a. Declarar variables
i: Entero
b. if numero == 0 then
1. factorial = 1
c. else
1. factorial = 1
2. for i=numero; i>=1; i--
a. factorial = factorial * i
3. endfor
d. endif
e. Fin Método calcularFactorial

4. Método obtenerFactorial(): Entero

a. return factorial
b. Fin Método obtenerFactorial

Fin Clase Factorial2

Clase EjecutaFactorial2

1. Método principal()

a. Declarar variables
num: Entero
b. Declarar, crear e iniciar objeto
Factorial2 objFactorial = new Factorial2()
c. Solicitar número
d. Leer num

```

        e. Establecer objFactorial.establecerNumero(num)
        f. Calcular objFactorial.calcularFactorial()
        g. Imprimir objFactorial.obtenerFactorial()
        h. Fin Método principal
    Fin Clase EjecutaFactorial2
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Factorial2.java y EjecutaFactorial2.java

Explicación:

El algoritmo tiene dos clases: la Clase Factorial2 y la Clase EjecutaFactorial2.

En la Clase Factorial2:

1. Se declaran los datos que representan la estructura de la clase:
numero para el número.
factorial para el factorial.
 2. Método establecerNumero(nu: Entero).
Recibe en el parámetro nu el valor que luego coloca en el dato numero.
 3. Método calcularFactorial().
 - a. Declarar variables:
i: Entero.
 - b. Si numero == 0 entonces:
1. factorial = 1.
 - c. Si no:
1. Inicia factorial en 1.
2. Inicia ciclo for desde i=numero hasta 1 con decrementos de 1.
a. Calcula factorial = factorial * i.
3. Fin del for.
 - d. Fin del if.
 - e. Fin Método calcularFactorial.
 4. Método obtenerFactorial().
Retorna factorial.
- Fin de la Clase Factorial2.

En la Clase EjecutaFactorial2, en el Método principal():

- a. Se declara la variable:
num para leer el número.
- b. Se declara el objeto objFactorial, usando como base a la Clase Factorial2. Dicho objeto se crea e inicializa mediante el constructor por defecto Factorial2().
- c. Solicita el número.
- d. Lee en num.

- e. Se llama al Método `establecerNumero (num)` del objeto `objFactorial` para colocar el valor de `num` en el dato `numero`.
 - f. Se llama al Método `calcularFactorial ()` del objeto `objFactorial` para calcular el factorial.
 - g. Se llama al Método `obtenerFactorial ()` del objeto `objFactorial` para acceder e imprimir el valor del dato `factorial`.
 - h. Fin Método principal.
- Fin de la Clase `EjecutaFactorial2`.
Fin del algoritmo.

Tabla 11.2: ejercicios resueltos disponibles en la zona de descarga del capítulo 11 de la Web del libro.

Ejercicio	Descripción
Ejercicio 11.2.1.3	Calcula el factorial a N números
Ejercicio 11.2.1.4	Procesa obreros con dos for
Ejercicio 11.2.1.5	Procesa obreros con un <code>do...while</code> y un <code>for</code>
Ejercicio 11.2.1.6	Procesa obreros con un <code>for</code> y un <code>do...while</code>

11.2.2 Ejercicios propuestos para for

Como ejercicios propuestos para este punto se recomiendan, del capítulo 5, del apartado de la repetición `for` (5.2), algunos de los ejercicios resueltos que no fueron incluidos aquí, además de todos los ejercicios propuestos en dicho apartado.

11.3 Diseño de algoritmos OO usando la repetición while

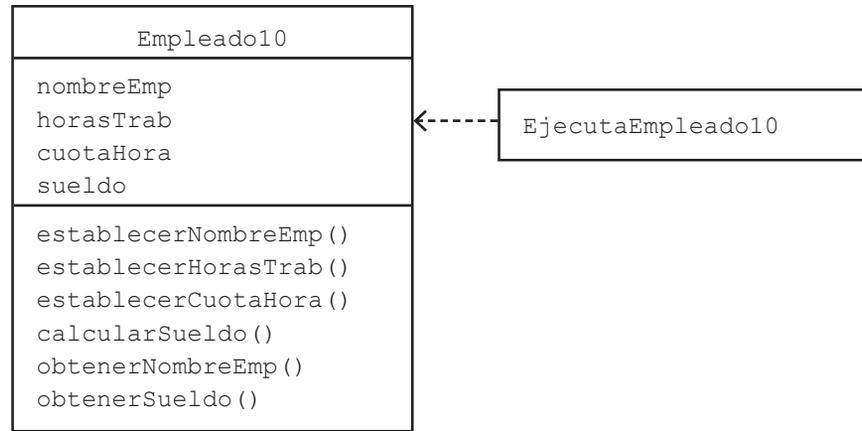
En este punto se utilizará la estructura de repetición `while` en pseudocódigo, misma que se estudió en el apartado 5.3 del capítulo 5, pero ahora aplicada conjuntamente con el diagrama de clases y los conceptos de la programación orientada a objetos, es decir, en el diseño de algoritmos orientados a objetos.

Ejemplo:

Elaborar un algoritmo que permita procesar varios empleados, igual al primer ejemplo del primer apartado (`do...while`) de este capítulo. Por cada empleado se leen los datos: nombre del empleado, número de horas trabajadas y cuota por hora, y se imprime el nombre y sueldo.

A continuación se presenta el algoritmo de la solución:

Diagrama de clases



Algoritmo CALCULAR SUELDO DE VARIOS EMPLEADOS

Clase Empleado10

1. Declarar datos

```

nombreEmp: Cadena
horasTrab: Entero
cuotaHora: Real
sueldo: Real
  
```

2. Método establecerNombreEmp(nom: Cadena)

```

a. nombreEmp = nom
b. Fin Método establecerNombreEmp
  
```

3. Método establecerHorasTrab(horasTr: Entero)

```

a. horasTrab = horasTr
b. Fin Método establecerHorasTrab
  
```

4. Método establecerCuotaHora(cuotaHr: Real)

```

a. cuotaHora = cuotaHr
b. Fin Método establecerCuotaHora
  
```

5. Método calcularSueldo()

```

a. sueldo = horasTrab * cuotaHora
b. Fin Método calcularSueldo
  
```

6. Método obtenerNombreEmp(): Cadena

```

a. return nombreEmp
b. Fin Método obtenerNombreEmp
  
```

7. Método obtenerSueldo(): Real

```

a. return sueldo
b. Fin Método obtenerSueldo
  
```

Fin Clase Empleado10

```

Clase EjecutaEmpleado10
1. Método principal()
  a. Declarar variables
    nomEmp: Cadena
    hrsTra: Entero
    cuoHr: Real
    desea: Carácter
  b. Preguntar "¿Desea procesar empleado (S/N)?"
  c. Leer desea
  d. while desea == 'S'
    1. Declarar, crear e iniciar objeto
      Empleado10 objEmpleado = new Empleado10()
    2. Solicitar nombre, número de horas trabajadas,
      cuota por hora
    3. Leer nomEmp, hrsTra, cuoHr
    4. Establecer
      objEmpleado.establecerNombreEmp(nomEmp)
      objEmpleado.establecerHorasTrab(hrsTra)
      objEmpleado.establecerCuotaHora(cuoHr)
    5. Calcular objEmpleado.calcularSueldo()
    6. Imprimir objEmpleado.obtenerNombreEmp()
      objEmpleado.obtenerSueldo()
    7. Preguntar "¿Desea procesar otro empleado(S/N)?"
    8. Leer desea
  e. endwhile
  f. Fin Método principal
Fin Clase EjecutaEmpleado10
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Empleado10.java y EjecutaEmpleado10.java



Explicación:

El algoritmo tiene dos clases: la Clase Empleado10 y la Clase EjecutaEmpleado10.

En la Clase Empleado10:

1. Se declaran los datos que representan la estructura de la clase:
 - nombreEmp para el nombre del empleado.
 - horasTrab para las horas trabajadas del empleado.
 - cuotaHora para la cuota por hora.
 - sueldo para el sueldo del empleado.
2. Método establecerNombreEmp(nom: Cadena).

Recibe en el parámetro nom el valor que luego coloca en el dato nombreEmp.
3. Método establecerHorasTrab(horasTr: Entero).

Recibe en el parámetro horasTr el valor que luego coloca en el dato horasTrab.

4. Método establecerCuotaHora (cuotaHr: Real).
Recibe en el parámetro cuotaHr el valor que luego coloca en el dato cuotaHr.
 5. Método calcularSueldo ()
Calcula el sueldo del empleado.
 6. Método obtenerNombreEmp () Cadena.
Retorna nombreEmp (nombre del empleado).
 7. Método obtenerSueldo () Real.
Retorna sueldo.
- Fin de la Clase Empleado10.

En la Clase EjecutaEmpleado10, en el Método principal ():

- a. Se declaran las variables:
nomEmp para leer el nombre del empleado.
hrsTra para leer las horas trabajadas.
cuoHr para leer la cuota por hora.
desea para controlar el ciclo repetitivo.
 - b. Pregunta “¿Desea procesar empleado (S/N)?”.
 - c. Lee la respuesta en desea.
 - d. Inicia ciclo while mientras desea == 'S':
 1. Se declara el objeto objEmpleado, usando como base a la Clase Empleado10. Dicho objeto se crea e inicializa mediante el constructor por defecto Empleado10 ().
Observe que cada vez que entra al ciclo crea un nuevo objeto empleado.
 2. Se solicitan el nombre, número de horas trabajadas y cuota por hora.
 3. Se leen en nomEmp, hrsTra, cuoHr.
 4. Se llama al Método establecerNombreEmp (nomEmp) del objeto objEmpleado para colocar el valor de nomEmp en el dato nombreEmp.
Se llama al Método establecerHorasTrab (hrsTra) del objeto objEmpleado para colocar el valor de hrsTra en el dato horasTrab.
Se llama al Método establecerCuotaHora (cuoHr) del objeto objEmpleado para colocar el valor de cuoHr en el dato cuotaHora.
 5. Se llama al Método calcularSueldo () del objeto objEmpleado para calcular el sueldo.
 6. Se llama al Método obtenerNombreEmp () del objeto objEmpleado para acceder e imprimir el valor del dato nombreEmp.
Se llama al Método obtenerSueldo () del objeto objEmpleado para acceder e imprimir el valor del dato sueldo.
 7. Se pregunta “¿Desea procesar otro empleado(S/N)?”.
 8. Se lee la respuesta en desea.
 - e. Fin del while.
 - f. Fin Método principal.
- Fin de la Clase EjecutaEmpleado10.
Fin del algoritmo.

11.3.1 Ejercicios resueltos para while

En este punto se presentan algunos de los ejercicios resueltos en el apartado 5.3 del capítulo 5. Es decir, el mismo problema ya lo resolvimos aplicando la lógica básica de la programación en el capítulo 5, y aquí lo estamos resolviendo aplicando la metodología de la programación orientada a objetos.

Ejercicio 11.3.1.1

El sueldo que perciben los vendedores de una empresa automotriz está integrado de la manera siguiente: el salario mínimo, más \$ 100.00 por cada auto vendido, más el 2 % del valor de los autos vendidos.

Datos que se tienen por cada vendedor:

Nombre del Vendedor : XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

precio auto : 999,999.99

precio auto : 999,999.99

precio auto : 999,999.99

Nombre del Vendedor : XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

precio auto : 999,999.99

precio auto : 999,999.99

precio auto : 999,999.99

.....

Nombre del Vendedor : XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

precio auto : 999,999.99

precio auto : 999,999.99

Como se puede apreciar, se tienen varios vendedores, y por cada vendedor se tiene el nombre y el precio de cada auto que vendió en la quincena. Es posible que algunos vendedores no hayan realizado venta alguna; en tal caso, sólo se tendrá el nombre.

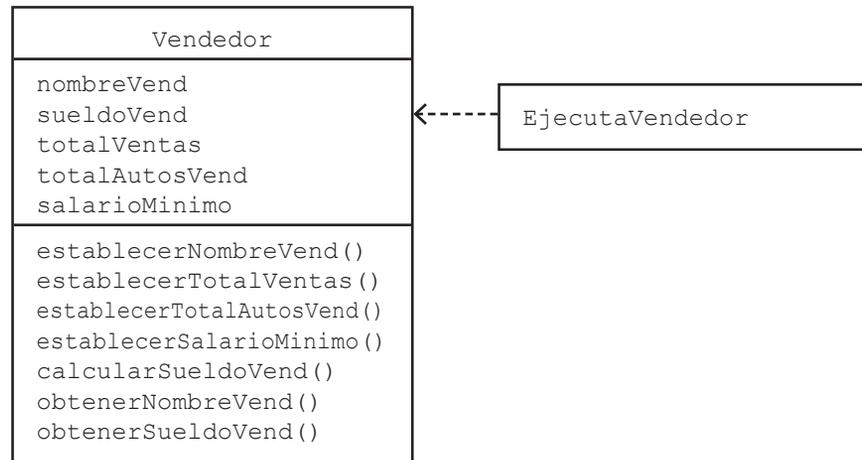
Elaborar un algoritmo que permita leer los datos e imprimir el siguiente reporte:

NÓMINA QUINCENAL	
NOMBRE	SUELDO
XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99,999.99
XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99,999.99
---	---
XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99,999.99
TOTALES 999	999,999.99

A continuación se presenta el algoritmo de la solución:

(Primero hágalo usted; después compare la solución)

Diagrama de clases



Algoritmo VENDEDORES DE AUTOS

Clase Vendedor

1. Declarar datos

```

nombreVend: Cadena
totalAutosVend: Entero
sueldoVend, totalVentas, salarioMinimo: Real
  
```

2. Método establecerNombreVend(nom: Cadena)

```

a. nombreVend = nom
b. Fin Método establecerNombreVend
  
```

3. Método establecerTotalVentas(toVen: Real)

```

a. totalVentas = toVen
b. Fin Método establecerTotalVentas
  
```

4. Método establecerTotalAutosVend(toAuVen: Entero)

```

a. totalAutosVend = toAuVen
b. Fin Método establecerTotalAutosVend
  
```

5. Método establecerSalarioMinimo(saMi: Real)

```

a. salarioMinimo = saMi
b. Fin Método establecerSalarioMinimo
  
```

6. Método calcularSueldoVend()

```

a. sueldoVend = salarioMinimo +
    (totalAutosVend*100.00)+
    (totalVentas*0.02)
b. Fin Método calcularSueldoVend
  
```

7. Método obtenerNombreVend(): Cadena
 - a. return nombreVend
 - b. Fin Método obtenerNombreVend
 8. Método obtenerSueldoVend(): Real
 - a. return sueldoVend
 - b. Fin Método obtenerSueldoVend
- Fin Clase Vendedor

Clase EjecutaVendedor

1. Método principal()
 - a. Declarar variables
 - nombre: Cadena
 - desea, otro: Carácter
 - totAutos, totVend: Entero
 - precioAuto, salMin, sueldo,
 - totSueldos, totVendido: Real
 - b. Solicitar salario mínimo
 - c. Leer salMin
 - d. Imprimir Encabezado
 - e. totSueldos = 0
totVend = 0
 - f. do
 1. Declarar, crear e iniciar objeto
Vendedor objVendedor = new Vendedor()
 2. Solicitar nombre del vendedor
 3. Leer nombre
 4. totAutos = 0
totVendido = 0
 5. Preguntar "¿Hay auto vendido (S/N)?"
 6. Leer otro
 7. while otro == 'S'
 - a. Solicitar precio del auto
 - b. Leer precioAuto
 - c. totAutos = totAutos + 1
totVendido = totVendido + precioAuto
 - d. Preguntar "¿Hay otro auto vendido (S/N)?"
 - e. Leer otro
 8. endwhile
 9. Establecer
objVendedor.establecerNombreVend(nombre)
objVendedor.establecerTotalVentas(totVendido)
objVendedor.establecerTotalAutosVend(totAutos)
objVendedor.establecerSalarioMinimo(salMin)
 10. Calcular objVendedor.calcularSueldoVend()
 11. Imprimir objVendedor.obtenerNombreVend()
objVendedor.obtenerSueldoVend()
 12. totVend = totVend + 1
totSueldos = totSueldos +
objVendedor.obtenerSueldoVend()

```

        13. Preguntar "¿Hay otro vendedor (S/N)?"
        14. Leer desea
    g. while desea == 'S'
    h. Imprimir totVend, totSueldos
    i. Fin Método principal
Fin Clase EjecutaVendedor
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Vendedor.java y EjecutaVendedor.java

Explicación:

El algoritmo tiene dos clases: la Clase Vendedor y la Clase EjecutaVendedor.

En la Clase Vendedor:

1. Se declaran los datos que representan la estructura de la clase:
 nombreVend para el nombre del vendedor.
 sueldoVend para el sueldo del vendedor.
 totalVentas para la cantidad vendida.
 totalAutosVend para los autos vendidos.
 salarioMinimo para el salario mínimo.
 2. Método establecerNombreVend(nom: Cadena).
 Recibe en el parámetro nom el valor que luego coloca en el dato nombreVend.
 3. Método establecerTotalVentas(toVen: Real).
 Recibe en el parámetro toVen el valor que luego coloca en el dato totalVentas
 4. Método establecerTotalAutosVend(toAuVen: Entero).
 Recibe en el parámetro toAuVen el valor que luego coloca en el dato totalAutosVend.
 5. Método establecerSalarioMinimo(saMi: Real).
 Recibe en el parámetro saMi el valor que luego coloca en el dato salarioMinimo.
 6. Método calcularSueldoVend().
 Calcula el sueldo sumando el salario mínimo más 100.00 por cada auto vendido más el 2% del total vendido en dinero.
 7. Método obtenerNombreVend() Cadena.
 Retorna nombreVend.
 8. Método obtenerSueldoVend() Real.
 Retorna sueldoVend.
- Fin de la Clase Vendedor.

En la Clase EjecutaVendedor, en el Método principal():

- a. Se declaran las variables:
 nombre para leer el nombre del vendedor.
 precioAuto para leer el precio de cada auto vendido.

salMin para leer el salario mínimo.
totVend para contar el total de vendedores.
totAutos para calcular el total de autos vendidos.
totSueldos para calcular el total de los sueldos de los vendedores.
totVendido para calcular el total vendido por cada vendedor.
desea y otro para controlar los ciclos repetitivos.

- b. Solicita el salario mínimo quincenal.
- c. Lee en salMin.
- d. Imprime el encabezado.
- e. Inicia totSueldos y totVend en 0.
- f. Inicia ciclo do:
 1. Se declara el objeto objVendedor, usando como base a la Clase Vendedor. Dicho objeto se crea e inicializa mediante el constructor por defecto Vendedor().
Observe que cada vez que entra al ciclo crea un nuevo objeto vendedor.
 2. Solicita el nombre del vendedor.
 3. Lee en nombre.
 4. Inicia totAutos y totVendido en 0.
 5. Pregunta “¿Hay auto vendido (S/N)?”.
 6. Lee en otro la respuesta.
 7. Inicia ciclo while mientras otro == 'S':
 - a. Solicita el precio del auto.
 - b. Lee en precioAuto.
 - c. Incrementa totAutos en 1.
Incrementa totVendido con precioAuto.
 - d. Pregunta “¿Hay otro auto vendido (S/N)?”.
 - e. Lee en otro la respuesta.
 8. Fin del while.
 9. Se llama al Método establecerNombreVend(nombre) del objeto objVendedor para colocar el valor de nombre en el dato nombreVend.
Se llama al Método establecerTotalAutosVend (totAutos) del objeto objVendedor para colocar el valor de totAutos en el dato totalAutosVend.
Se llama al Método establecerTotalVentas (totVendido) del objeto objVendedor para colocar el valor de totVendido en el dato totalVentas.
Se llama al Método establecerSalarioMinimo (salMin) del objeto objVendedor para colocar el valor de salMin en el dato salarioMinimo.
 10. Se llama al Método calcularSueldoVend() del objeto objVendedor para calcular el sueldo.
 11. Se llama al Método obtenerNombreVend() del objeto objVendedor para acceder e imprimir el valor del dato nombreVend.
Se llama al Método obtenerSueldoVend() del objeto objVendedor para acceder e imprimir el valor del dato sueldoVend.
 12. Se incrementa totVend en 1.

Se incrementa `totSueldos` con el sueldo del empleado, al que se accede llamando al Método `obtenerSueldoVend()` del objeto `objVendedor`.

13. Pregunta “¿Hay otro vendedor (S/N)?”.

14. Lee la respuesta en `desea`.

g. Fin del ciclo (`while desea == 'S'`). Si se cumple regresa al `do`; si no, se sale del ciclo.

h. Imprime `totVend`, `totSueldos`.

i. Fin Método principal.

Fin de la Clase `EjecutaVendedor`.

Fin del algoritmo.

Ejercicio 11.3.1.2

La Comisión Nacional del Agua, Delegación Sonora, lleva un registro de las lluvias que se presentan en todas las poblaciones del estado, de manera que se tienen los siguientes datos:

Población 1: X-----X

Lluvia: ----

Lluvia: ----

...

Lluvia: ----

Población 2: X-----X

Lluvia: ----

Lluvia: ----

...

Lluvia: ----

Población N: X-----X

Lluvia: ----

Lluvia: ----

...

Lluvia: ----

Por cada población aparece el dato lluvia tantas veces como lluvias haya habido (este dato está en milímetros cúbicos). Podría darse el caso de que alguna población no traiga ningún dato de lluvia; esto quiere decir que no llovió.

Elaborar un algoritmo que lea estos datos e imprima el reporte:

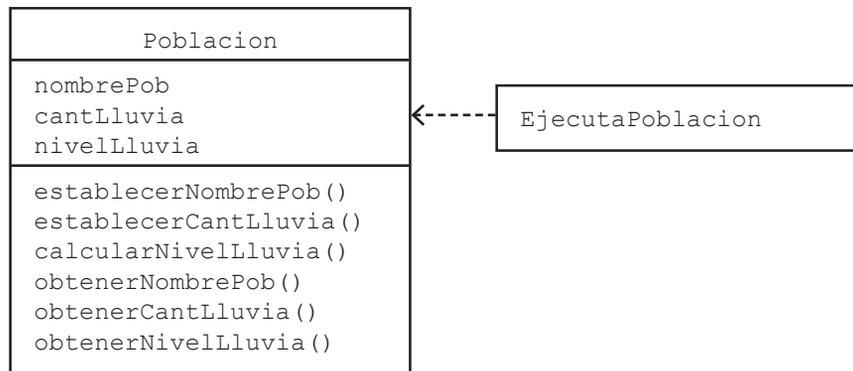
POBLACIÓN	REPORTE DE LLUVIAS	
	TOTAL LLUVIA	NIVEL LLUVIA
XXXXXXXXXXXXXXXXXXXXX	999.99	NULA
XXXXXXXXXXXXXXXXXXXXX	999.99	BAJA
		REGULAR
XXXXXXXXXXXXXXXXXXXXX	999.99	ALTA
TOTAL 999 POBLACIONES	999.99	
TOTAL POBLACIONES DONDE NO LLOVIÓ: 999		

Cálculos:

- TOTAL LLUVIA es el total de lluvia que se presentó en una población. Se calcula sumando la cantidad de milímetros cúbicos de todas las lluvias.
- Se piden dos totales: TOTAL de poblaciones procesadas y el total del total de lluvia en todas las poblaciones. También se pide la cantidad de poblaciones donde no se presentó lluvia alguna.
- El NIVEL LLUVIA es una observación, imprime:
 - NULA si no llovió.
 - BAJA si TOTAL LLUVIA es menor a 100.
 - REGULAR si TOTAL LLUVIA está entre 100 y 250.
 - ALTA si TOTAL LLUVIA es mayor a 250.

A continuación se presenta el algoritmo de la solución:
(Primero hágalo usted; después compare la solución)

Diagrama de clases



Algoritmo LLUVIAS

Clase Poblacion

1. Declarar datos

```

nombrePob: Cadena
cantLluvia: Real
nivelLluvia: Cadena
  
```

2. Método establecerNombrePob(nom: Cadena)

```

a. nombrePob = nom
b. Fin Método establecerNombrePob
  
```

3. Método establecerCantLluvia(cant: Real)

```

a. cantLluvia = cant
b. Fin Método establecerCantLluvia
  
```

4. Método calcularNivelLluvia()

```

a. if cantLluvia == 0 then
  1. nivelLluvia = "NULA"
  
```

```
b. endif
c. if (cantLluvia > 0)AND(cantLluvia < 100) then
    1. nivelLluvia = "BAJA"
d. endif
e. if (cantLluvia >= 100)AND(cantLluvia <= 250)then
    1. nivelLluvia = "REGULAR"
f. endif
g. if cantLluvia > 250 then
    1. nivelLluvia = "ALTA"
h. endif
i. Fin Método calcularNivelLluvia

5. Método obtenerNombrePob(): Cadena
a. return nombrePob
b. Fin Método obtenerNombrePob

6. Método obtenerCantLluvia(): Real
a. return cantLluvia
b. Fin Método obtenerCantLluvia

7. Método obtenerNivelLluvia(): Cadena
a. return nivelLluvia
b. Fin Método obtenerNivelLluvia
Fin Clase Poblacion

Clase EjecutaPoblacion
1. Método principal()
a. Declarar variables
    poblacion: Cadena
    otro, hay: Carácter
    totPobla, totPobNoLluvia: Entero
    lluvia, totLluvia, toTotLluvia: Real
b. Imprimir encabezado
c. totPobla = 0
    totPobNoLluvia = 0
    toTotLluvia = 0
d. Preguntar "¿Hay poblacion (S/N)?"
e. Leer hay
f. while hay == 'S'
    1. Declarar, crear e iniciar objeto
        Poblacion objPoblacion = new Poblacion()
    2. Solicitar población
    3. Leer poblacion
    4. totLluvia = 0
    5. Preguntar "¿Hay lluvia (S/N)?"
    6. Leer otro
    7. while otro == 'S'
        a. Solicitar lluvia en milímetros cúbicos
        b. Leer lluvia
```

```

        c. totLluvia = totLluvia + lluvia
        d. Preguntar "¿Hay otra lluvia (S/N)?"
        e. Leer otro
    8. endwhile
    9. Establecer
        objPoblacion.establecerNombrePob(poblacion)
        objPoblacion.establecerCantLluvia(totLluvia)
    10. Calcular objPoblacion.calcularNivelLluvia()
    11. Imprimir objPoblacion.obtenerNombrePob()
        objPoblacion.obtenerCantLluvia()
        objPoblacion.obtenerNivelLluvia()
    12. totPobla = totPobla + 1
        toTotLluvia = toTotLluvia +
            objPoblacion.obtenerCantLluvia()
    13. if totLluvia == 0 then
        a. totPobNoLluvia = totPobNoLluvia + 1
    14. endif
    15. Preguntar "¿Hay otra poblacion (S/N)?"
    16. Leer hay
g. endwhile
h. Imprimir totPobla, toTotLluvia, totPobNoLluvia
i. Fin Método principal
Fin Clase EjecutaPoblacion
Fin

```

En la zona de descarga de la Web del libro está disponible:
Programa en Java: Poblacion.java y EjecutaPoblacion.java



Explicación:

El algoritmo tiene dos clases: la Clase Poblacion y la Clase EjecutaPoblacion.

En la Clase Poblacion:

1. Se declaran los datos que representan la estructura de la clase:
 - nombrePob para el nombre de la población.
 - cantLluvia para la cantidad de lluvia.
 - nivelLluvia para el nivel de lluvia.
2. Método establecerNombrePob(nom: Cadena).
 - Recibe en el parámetro nom el valor que luego coloca en el dato nombrePob.
3. Método establecerCantLluvia(cant: Real).
 - Recibe en el parámetro cant el valor que luego coloca en el dato cantLluvia.
4. Método calcularNivelLluvia().
 - a. Si cantLluvia == 0 entonces:
 1. nivelLluvia = "NULA".
 - b. Fin del if.
 - c. Si (cantLluvia > 0) AND (cantLluvia < 100) entonces:
 1. nivelLluvia = "BAJA".
 - d. Fin del if.

- e. Si `(cantLluvia >= 100)AND(cantLluvia <= 250)` entonces:
 1. `nivelLluvia = "REGULAR"`.
 - f. Fin del if.
 - g. Si `cantLluvia > 250` entonces:
 1. `nivelLluvia = "ALTA"`.
 - h. Fin del if.
 - i. Fin Método `calcularNivelLluvia`.
5. Método `obtenerNombrePob()` Cadena.
Retorna `nombrePob`.
 6. Método `obtenerCantLluvia()` Real.
Retorna `cantLluvia`.
 7. Método `obtenerNivelLluvia()` Cadena.
Retorna `nivelLluvia`.
- Fin de la Clase `Poblacion`.

En la Clase `EjecutaPoblacion`, en el Método `principal()`:

- a. Se declaran las variables:
 - `poblacion` para leer el nombre de la población.
 - `lluvia` para leer la cantidad de cada lluvia que hubo.
 - `totPobla` para contar el total de poblaciones.
 - `totPobNoLluvia` para contar el total de poblaciones donde no llovió.
 - `totLluvia` para calcular el total de lluvia de cada población.
 - `toTotLluvia` para calcular el total de lluvia de todas las poblaciones.
 - `otro` y `hay` para controlar los ciclos repetitivos.
- b. Imprime el encabezado.
- c. Inicia `totPobla`, `totPobNoLluvia` y `toTotLluvia` en 0.
- d. Pregunta "¿Hay poblacion (S/N)?".
- e. Lee la respuesta en `hay`.
- f. Inicia ciclo `while` mientras `hay == 'S'`:
 1. Se declara el objeto `objPoblacion`, usando como base a la Clase `Poblacion`. Dicho objeto se crea e inicializa mediante el constructor por defecto `Poblacion()`.
Observe que cada vez que entra al ciclo crea un nuevo objeto `poblacion`.
 2. Solicita población.
 3. Lee en `poblacion`.
 4. Inicia `totLluvia` en 0.
 5. Pregunta "¿Hay lluvia (S/N)?".
 6. Lee la respuesta en `otro`.
 7. Inicia ciclo `while` mientras `otro == 'S'`:
 - a. Solicita lluvia en milímetros cúbicos.
 - b. Lee en `lluvia`.
 - c. Incrementa `totLluvia` con `lluvia`.
 - d. Pregunta "¿Hay otra lluvia (S/N)?".
 - e. Lee la respuesta en `otro`.
 8. Fin del `while`.

9. Se llama al Método establecerNombrePob(poblacion) del objeto objPoblacion para colocar el valor de poblacion en el dato nombrePob.
Se llama al Método establecerCantLluvia(totLluvia) del objeto objPoblacion para colocar el valor de totLluvia en el dato cantLluvia.
 10. Se llama al Método calcularLluvia() del objeto objPoblacion para calcular el nivel de lluvia.
 11. Se llama al Método obtenerNombrePob() del objeto objPoblacion para acceder e imprimir el valor del dato nombrePob.
Se llama al Método obtenerCantLluvia() del objeto objPoblacion para acceder e imprimir el valor del dato cantLluvia.
Se llama al Método obtenerNivelLluvia() del objeto objPoblacion para acceder e imprimir el valor del dato nivelLluvia.
 12. Incrementa totPobla en 1.
Incrementa toTotLluvia con totLluvia.
 13. Si totLluvia == 0 entonces:
 1. Incrementa totPobNoLluvia en 1.
 14. Fin del if.
 15. Pregunta “¿Hay otra poblacion (S/N)?”.
 16. Lee la respuesta en hay.
- g. Fin del while.
- h. Imprime totPobla, toTotLluvia, totPobNoLluvia.
- i. Fin Método principal.
- Fin de la Clase EjecutaPoblacion.
- Fin del algoritmo.

En la zona de descarga del capítulo 11 de la Web del libro está disponible el ejercicio resuelto.

Ejercicio	Descripción
Ejercicio 11.3.1.3	Procesa la producción de estaciones de trabajo

11.3.2 Ejercicios propuestos para while

Como ejercicios propuestos para este punto se recomiendan, del capítulo 5, del apartado de la repetición while (5.3), algunos de los ejercicios resueltos que no fueron incluidos aquí, además de todos los ejercicios propuestos en dicho punto.

11.4 Resumen de conceptos que debe dominar

- Diseño de algoritmos orientados a objetos usando las estructuras de repetición:
 - do...while
 - for
 - while.

11.5 Contenido de la página Web de apoyo

El material marcado con asterisco () sólo está disponible para docentes.*

11.5.1 Resumen gráfico del capítulo

11.5.2 Autoevaluación

11.5.3 Programas en Java

11.5.4 Ejercicios resueltos

11.5.5 Power Point para el profesor (*)

12

Programación orientada a objetos aplicando arreglos

Contenido

- 12.1 Diseño de algoritmos OO usando arreglos unidimensionales
 - 12.1.1 Ejercicios propuestos para arreglos unidimensionales
- 12.2 Diseño de algoritmos OO usando arreglos bidimensionales
 - 12.2.1 Ejercicios propuestos para arreglos bidimensionales
- 12.3 Diseño de algoritmos OO usando arreglos tridimensionales
 - 12.3.1 Ejercicios propuestos para arreglos tridimensionales
- 12.4 Diseño de algoritmos OO usando arreglos tetradimensionales
 - 12.4.1 Ejercicios propuestos para arreglos tetradimensionales
- 12.5 Resumen de conceptos que debe dominar
- 12.6 Contenido de la página Web de apoyo
 - El material marcado con asterisco (*) sólo está disponible para docentes.*
 - 12.6.1 Resumen gráfico del capítulo
 - 12.6.2 Autoevaluación
 - 12.6.3 Programas en Java
 - 12.6.4 Ejercicios resueltos
 - 12.6.5 Power Point para el profesor (*)

Objetivos del capítulo

- Aprender a usar arreglos unidimensionales, bidimensionales, tridimensionales y tetradimensionales en el diseño de algoritmos orientados a objetos.
- Aplicar lo aprendido en la solución de ejercicios resueltos y propuestos.

Competencias

- Competencia general del capítulo
 - *Analizar problemas y diseñar algoritmos que los solucionen aplicando la arquitectura orientada a objetos y arreglos.*
- Competencias específicas del capítulo
 - Diseña algoritmos orientados a objetos aplicando arreglos unidimensionales.
 - Elabora algoritmos orientados a objetos aplicando arreglos bidimensionales.
 - Desarrolla algoritmos orientados a objetos aplicando arreglos tridimensionales.
 - Diseña algoritmos orientados a objetos aplicando arreglos tetradimensionales.

Introducción

Con el estudio del capítulo anterior, usted ya domina los conceptos básicos de la programación orientada a objetos aplicando las estructuras de repetición y sabe cómo diseñar algoritmos usando dichas estructuras.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos usando la estructura de datos denominada arreglos, inmersa en la arquitectura orientada a objetos.

Se explica cómo diseñar algoritmos orientados a objetos usando la estructura denominada arreglos que se estudió en el capítulo 6, pero ahora inmersa en la arquitectura orientada a objetos.

Recordemos que el arreglo es un tipo de dato estructurado formado por un conjunto de elementos de un mismo tipo de datos y que puede almacenar más de un valor a la vez, con la condición de que todos los elementos deben ser del mismo tipo de dato, es decir, se puede tener un arreglo de datos enteros, reales, cadenas, etcétera. Asimismo, los arreglos se clasifican de acuerdo con el número de dimensiones que tienen. Así, se tienen los unidimensionales, los bidimensionales y los multidimensionales (de más de dos dimensiones); dentro de éstos están los tridimensionales, tetradimensionales, etcétera.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejercite estudiando los problemas planteados en los ejercicios resueltos y propuestos. Al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro, luego verifique sus resultados con los del libro, analice las diferencias y vea sus errores. Al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no está igual que el del libro, no necesariamente está mal. Usted debe ir aprendiendo a analizar las diferencias y a comprender que a veces, aunque haya diferencias, las dos soluciones están correctas.

En el siguiente capítulo se estudia la herencia, que es uno de los conceptos fundamentales de la programación orientada a objetos.

12.1 Diseño de algoritmos OO usando arreglos unidimensionales

En este punto se utilizarán los arreglos unidimensionales en pseudocódigo, mismos que se estudiaron en el capítulo 6, pero ahora aplicados conjuntamente con el diagrama de clases y los conceptos de la programación orientada a objetos, es decir, en el diseño de algoritmos orientados a objetos.

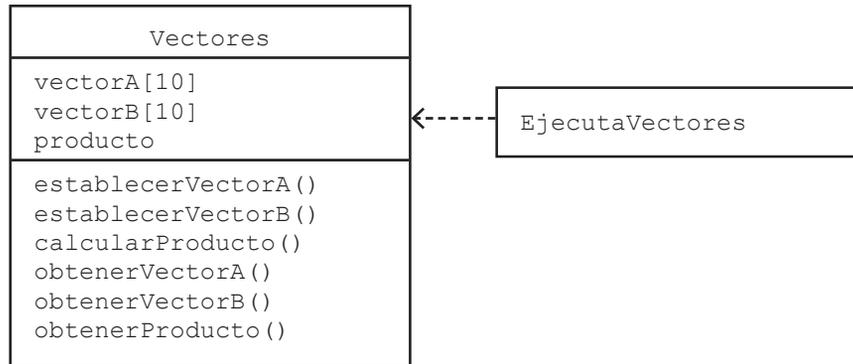
Ejercicio 12.1.1

Elaborar un algoritmo que permita leer un vector de 10 números en un arreglo A de 10 elementos, lo mismo para un arreglo B, y calcule e imprima el producto de A x B. Para obtener el producto de dos vectores se multiplica el elemento 1 del vector A por el elemento 1 del vector B, el 2 de A por el 2 de B, y así sucesivamente,

obteniéndose la sumatoria de los productos. El resultado no es un vector, sino un valor simple.

A continuación se presenta el algoritmo de la solución:

Diagrama de clases



Algoritmo PRODUCTO DE VECTORES

Clase Vectores

1. Declarar datos
 - vectorA: Arreglo[10] Entero
 - vectorB: Arreglo[10] Entero
 - producto: Entero
2. Método establecerVectorA(vecA[]: Entero)
 - a. vectorA = vecA
 - b. Fin Método establecerVectorA
3. Método establecerVectorB(vecB[]: Entero)
 - a. vectorB = vecB
 - b. Fin Método establecerVectorB
4. Método calcularProducto()
 - a. Declarar variables
 - r: Entero
 - b. producto = 0
 - c. for r=0; r<=9; r++
 1. producto = producto + (vectorA[r] * vectorB[r])
 - d. endfor
 - e. Fin Método calcularProducto
5. Método obtenerVectorA(): Arreglo[] Entero
 - a. return vectorA
 - b. Fin Método obtenerVectorA
6. Método obtenerVectorB(): Arreglo[] Entero
 - a. return vectorB
 - b. Fin Método obtenerVectorB

```

7. Método obtenerProducto(): Entero
  a. return producto
  b. Fin Método obtenerProducto
Fin Clase Vectores

Clase EjecutaVectores
1. Método principal()
  a. Declarar variables
    vectA, vectA2: Arreglo[10] Entero
    vectB, vectB2: Arreglo[10] Entero
    i: Entero
  b. Declarar, crear e iniciar objeto
    Vectores objVectores = new Vectores()
  c. for i=0; i<=9; i++
    1. Solicitar vector A[i]
    2. Leer vectA[i]
  d. endfor
  e. for i=0; i<=9; i++
    1. Solicitar vector B[i]
    2. Leer vectB[i]
  f. endfor
  g. Establecer
    objVectores.establecerVectorA(vectA)
    objVectores.establecerVectorB(vectB)
  h. Calcular objVectores.calcularProducto()
  i. Obtener
    vectA2 = objVectores.obtenerVectorA()
    vectB2 = objVectores.obtenerVectorB()
  j. Imprimir encabezado
  k. for i=0; i<=9; i++
    1. Imprimir vectA2[i], vectB2[i]
  l. endfor
  m. Imprimir objVectores.obtenerProducto()
  n. Fin Método principal
  Fin Clase EjecutaVectores
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: Vectores.java y EjecutaVectores.java

Explicación:

El algoritmo tiene dos clases: la Clase Vectores y la Clase EjecutaVectores.

En la Clase Vectores:

1. Se declaran los datos que representan la estructura de la clase:
 - vectorA para el vector A de números.
 - vectorB para el vector B de números.
 - producto para el producto de los vectores.

2. Método establecerVectorA(`vecA[]: Entero`).
Recibe en el parámetro `vecA` los valores que luego coloca en el dato `vectorA`.
 3. Método establecerVectorB(`vecB[]: Entero`).
Recibe en el parámetro `vecB` los valores que luego coloca en el dato `vectorB`.
 4. Método `calcularProducto()`.
 - a. Declara la variable `r` para controlar el ciclo.
 - b. Se inicia `producto` en 0.
 - c. Inicia ciclo `for` desde `r=0` hasta 9 con incrementos de 1.
 1. Se multiplican los elementos `r` de ambos vectores y el resultado se incrementa en `producto`.
 - d. Fin del ciclo `for`.
 - e. Fin Método `calcularProducto`.
 5. Método `obtenerVectorA(): Arreglo[] Entero`.
Retorna `vectorA`.
 6. Método `obtenerVectorB(): Arreglo[] Entero`.
Retorna `vectorB`.
 7. Método `obtenerProducto(): Entero`.
Retorna `producto`.
- Fin de la Clase `Vectores`.

En la Clase `EjecutaVectores`, en el Método `principal()`:

- a. Se declaran las variables:
 - `vectA` para leer los números del vector A.
 - `vectB` para leer los números del vector B.
 - `vectA2` para recibir los números del vector A del objeto.
 - `vectB2` para recibir los números del vector B del objeto.
 - `i` para controlar el ciclo repetitivo.
- b. Se declara el objeto `objVectores`, usando como base a la Clase `Vectores`.
Dicho objeto se crea e inicializa mediante el constructor por defecto `Vectores()`.
- c. Inicia ciclo `for` desde `i=0` hasta 9 con incrementos de 1.
 1. Solicita vector A.
 2. Lee en `vectA[i]`.
- d. Fin del ciclo `for`.
- e. Inicia ciclo `for` desde `i=0` hasta 9 con incrementos de 1.
 1. Solicita vector B.
 2. Lee en `vectB[i]`.
- f. Fin del ciclo `for`.
- g. Se llama al Método `establecerVectorA(vectA)` del objeto `objVectores` para colocar el valor de `vectA` en el dato `vectorA`.
Se llama al Método `establecerVectorB(vectB)` del objeto `objVectores` para colocar el valor de `vectB` en el dato `vectorB`.
- h. Se llama al Método `calcularProducto()` del objeto `objVectores` para calcular el producto.
- i. Llama al Método `obtenerVectorA()` del objeto `objVectores` para obtener el dato `vectorA` y colocarlo en `vectA2`, que es como se procesará de aquí en adelante en este método.

Llama al Método `obtenerVectorB()` del objeto `objVectores` para obtener el dato `vectorB` y colocarlo en `vectB2`, que es como se procesará de aquí en adelante en este método.

- j. Imprime el encabezado.
 - k. Inicia ciclo for desde `i=0` hasta 9 con incrementos de 1.
 - 1. Imprime el vector `vectA2[i]` y el vector `vectB2[i]`.
 - l. Fin del ciclo for.
 - m. Se llama al Método `obtenerProducto()` del objeto `objVectores` para acceder e imprimir el valor del dato `producto`.
 - n. Fin del Método principal.
- Fin de la Clase `EjecutaVectores`.
Fin del algoritmo.

Explicación resumida:

En la Clase `EjecutaVectores`, para cada arreglo que se va a leer se declara una variable, `vectA` y `vectB`, y en ambas se leen los dos vectores respectivamente.

Enseguida, se declara, crea e inicia el objeto `objVectores`.

Luego, llamando los métodos establecer del objeto, se envían al objeto los dos vectores y se establecen en `vectorA` y `vectorB`.

Se llama al método que calcula el producto de los dos vectores.

Se traen los datos del objeto llamando los métodos `obtener` y los arreglos `vectorA` y `vectorB` se reciben en `vectA2` y `vectB2`. Para cada arreglo que se va a imprimir, en la Clase `Ejecuta` se declara una variable para recibir el arreglo que viene del objeto. En este caso se declaró `vectA2` para recibir `vectorA` y `vectB2` para recibir `vectorB`. Los arreglos no se pueden imprimir directamente llamando al método `obtener`; es por ello que para cada arreglo que se va a imprimir se declara una segunda variable, en la cual se colocan los datos traídos del objeto con los métodos `obtener`.

A continuación se imprimen los arreglos (vectores) que fueron recibidos en `vectA2` y `vectB2` y finalmente se imprime el producto, directamente trayéndolo con el método `obtener`.

Ejercicio 12.1.2

Elaborar un algoritmo que permita leer 15 números en un arreglo de 15 elementos, calcular la media e imprimir cada elemento del arreglo y la desviación que tiene respecto a la media de la siguiente forma:

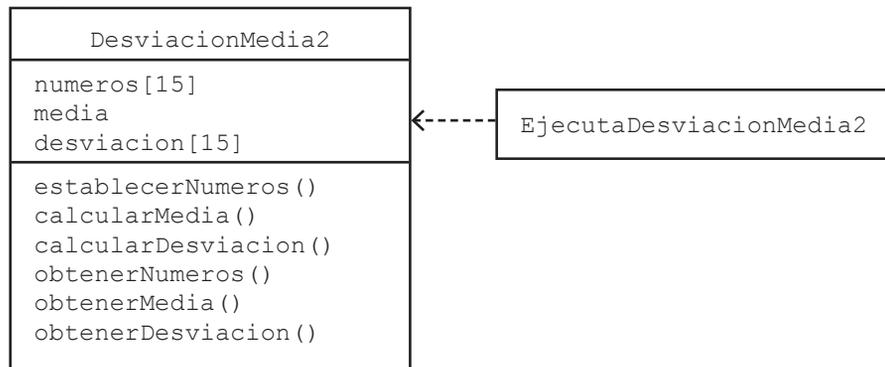
NÚMERO	DESVIACIÓN
---	---
---	---
---	---
---	---
---	---
MEDIA =	---

Cálculos:

- MEDIA se obtiene la sumatoria de los 15 elementos del arreglo y se divide entre 15.
- DESVIACIÓN de cada elemento se obtiene restándole al elemento la MEDIA.

(Primero hágalo usted; después compare la solución)

Diagrama de clases



Algoritmo DESVIACION DE MEDIA

Clase DesviacionMedia2

1. Declarar datos
 - numeros: Arreglo[15] Entero
 - media: Real
 - desviacion: Arreglo[15] Real
2. Método establecerNumeros(n[]: Entero)
 - a. numeros = n
 - b. Fin Método establecerNumeros
3. Método calcularMedia()
 - a. Declarar variables
 - r, sumatoria: Entero
 - b. sumatoria = 0
 - c. for r=0; r<=14; r++
 1. sumatoria = sumatoria + numeros[r]
 - d. endfor
 - e. media = sumatoria / r
 - f. Fin Método calcularMedia
4. Método calcularDesviacion()
 - a. Declarar variables
 - j: Entero
 - b. for j=0; j<=14; j++
 1. desviacion[j] = numeros[j] - media
 - c. endfor
 - d. Fin Método calcularDesviacion
5. Método obtenerNumeros(): Arreglo[] Entero
 - a. return numeros
 - b. Fin Método obtenerNumeros

```

6. Método obtenerMedia(): Real
  a. return media
  b. Fin Método obtenerMedia

7. Método obtenerDesviacion(): Arreglo[] Real
  a. return desviacion
  b. Fin Método obtenerDesviacion
Fin Clase DesviacionMedia2

Clase EjecutaDesviacionMedia2
1. Método principal()
  a. Declarar variables
    nums, nums2: Arreglo[15] Entero
    desv: Arreglo[15] Real
    i: Entero
  b. Declarar, crear e iniciar objeto
    DesviacionMedia2 objNumeros = new DesviacionMedia2()
  c. for i=0; i<=14; i++
    1. Solicitar nums[i]
    2. Leer nums[i]
  d. endfor
  e. Establecer
    objNumeros.establecerNumeros(nums)
  f. Calcular objNumeros.calcularMedia()
    objNumeros.calcularDesviacion()
  g. Obtener
    nums2 = objNumeros.obtenerNumeros()
    desv = objNumeros.obtenerDesviacion()
  h. Imprimir encabezado
  i. for i=0; i<=14; i++
    1. Imprimir nums2[i], desv[i]
  j. endfor
  k. Imprimir objNumeros.obtenerMedia()
  l. Fin Método principal
Fin Clase EjecutaDesviacionMedia2
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: DesviacionMedia2.java y EjecutaDesviacionMedia2.java

Explicación:

El algoritmo tiene dos clases: la Clase DesviacionMedia2 y la Clase EjecutaDesviacionMedia2.

En la Clase DesviacionMedia2:

1. Se declaran los datos que representan la estructura de la clase:
 - numeros para el arreglo de números.
 - media para la media de los números.
 - desviacion para el arreglo de la desviación de los números.

2. Método establecerNumeros($n[]$: Entero).
Recibe en el parámetro n los valores que luego coloca en el dato `numeros`.
 3. Método calcularMedia().
 - a. Declara las variables:
 - r para controlar el ciclo.
 - `sumatoria` para calcular la sumatoria de los números.
 - b. Se inicia `sumatoria` en 0.
 - c. Inicia ciclo for desde $r=0$ hasta 14 con incrementos de 1.
 1. Incrementa `sumatoria` con `numeros[r]`.
 - d. Fin del ciclo for.
 - e. Calcula `media = sumatoria / r`.
 - f. Fin del Método `calcularMedia`.
 4. Método calcularDesviacion ().
 - a. Declara la variable j para controlar el ciclo.
 - b. Inicia ciclo for desde $j=0$ hasta 14 con incrementos de 1.
 1. Calcula `desviacion[j] = numeros[j] - media`.
 - c. Fin del ciclo for.
 - d. Fin del Método `calcularDesviacion`.
 5. Método obtenerNumeros(): Arreglo[] Entero.
Retorna `numeros`.
 6. Método obtenerMedia(): Real.
Retorna `media`.
 7. Método obtenerDesviacion(): Arreglo[] Real.
Retorna `desviacion`.
- Fin de la Clase `DesviacionMedia2`.

En la Clase `EjecutaDesviacionMedia2`, en el Método `principal()`:

- a. Se declaran las variables:
 - `nums` para leer el arreglo de números.
 - `nums2` para recibir los números del arreglo del objeto.
 - `desv` para recibir las desviaciones de los números del objeto.
 - i para controlar el ciclo repetitivo.
- b. Se declara el objeto `objNumeros`, usando como base a la Clase `DesviacionMedia2`. Dicho objeto se crea e inicializa mediante el constructor por defecto `DesviacionMedia2()`.
- c. Inicia ciclo for desde $i=0$ hasta 14 con incrementos de 1.
 1. Solicita `nums[i]`.
 2. Lee en `nums[i]`.
- d. Fin del ciclo for.
- e. Se llama al Método `establecerNumeros(nums)` del objeto `objNumeros` para colocar el valor de `nums` en el dato `numeros`.
- f. Se llama al Método `calcularMedia()` del objeto `objNumeros` para calcular la media.
Se llama al Método `calcularDesviacion()` del objeto `objNumeros` para calcular la desviación.

- g. Llama al Método `obtenerNumeros()` del objeto `objNumeros` para obtener el dato `numeros` y colocarlo en `nums2`, que es como se procesará de aquí en adelante en este método.
Llama al Método `obtenerDesviacion()` del objeto `objNumeros` para obtener el dato `desviacion` y colocarlo en `desv`, que es como se procesará de aquí en adelante en este método.
 - h. Imprime encabezado.
 - i. Inicia ciclo for desde `i=0` hasta 14 con incrementos de 1.
 - 1. Imprime cada numero `nums2[i]` y su desviación de la media `desv[i]`.
 - j. Fin del ciclo for.
 - k. Se llama al Método `obtenerMedia()` del objeto `objNumeros` para acceder e imprimir el valor del dato `media`.
 - l. Fin del Método principal.
- Fin de la Clase `EjecutaDesviacionMedia2`.
Fin del algoritmo.

Tabla 12.1: muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 12 de la Web del libro.

Ejercicio	Descripción
Ejercicio 12.1.3	Lee dos arreglos y los suma.
Ejercicio 12.1.4	Procesa la producción de 30 días de un obrero.

12.1.1 Ejercicios propuestos para arreglos unidimensionales

Como ejercicios propuestos para este punto se recomiendan, del capítulo 6, algunos de los ejercicios resueltos que no fueron incluidos en este apartado, además de todos los ejercicios propuestos en dicho capítulo sobre arreglos unidimensionales.

12.2 Diseño de algoritmos OO usando arreglos bidimensionales

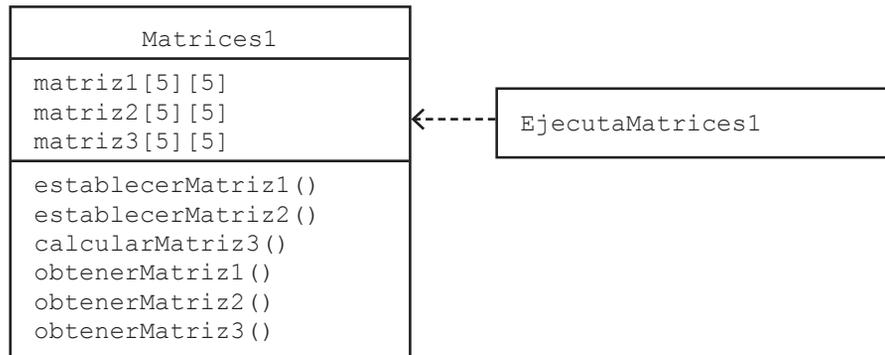
En este punto se utilizarán los arreglos bidimensionales en pseudocódigo, mismos que se estudiaron en el capítulo 6, pero ahora aplicados conjuntamente con el diagrama de clases y los conceptos de la programación orientada a objetos, es decir, en el diseño de algoritmos orientados a objetos.

Ejercicio 12.2.1

Elaborar un algoritmo que lea números enteros para los elementos de dos matrices de 5×5 y que calcule cada elemento de una tercera matriz sumando los elementos correspondientes de las dos anteriores. Al final, imprimir las tres matrices.

A continuación se presenta el algoritmo de la solución:

Diagrama de clases



Algoritmo SUMA MATRICES

Clase Matrices1

1. Declarar datos

```

matriz1: Arreglo[5][5] Entero
matriz2: Arreglo[5][5] Entero
matriz3: Arreglo[5][5] Entero
  
```

2. Método establecerMatriz1(mat1[][]: Entero)

```

a. matriz1 = mat1
b. Fin Método establecerMatriz1
  
```

3. Método establecerMatriz2(mat2[][]: Entero)

```

a. matriz2 = mat2
b. Fin Método establecerMatriz2
  
```

4. Método calcularMatriz3()

```

a. Declarar variables
   r, c: Entero
b. for r=0; r<=4; r++
   1. for c=0; c<=4; c++
      a. matriz3[r][c]=matriz1[r][c]+matriz2[r][c]
   2. endfor
c. endfor
d. Fin Método calcularMatriz3
  
```

5. Método obtenerMatriz1(): Arreglo[][] Entero

```

a. return matriz1
b. Fin Método obtenerMatriz1
  
```

6. Método obtenerMatriz2(): Arreglo[][] Entero

```

a. return matriz2
b. Fin Método obtenerMatriz2
  
```

7. Método obtenerMatriz3(): Arreglo[][] Entero

```

a. return matriz3
b. Fin Método obtenerMatriz3
  
```

Fin Clase Matrices1

```

Clase EjecutaMatrices1
1. Método principal()
  a. Declarar variables
    matr1, matr12: Arreglo[5][5] Entero
    matr2, matr22: Arreglo[5][5] Entero
    matr3: Arreglo[5][5] Entero
    ren, col: Entero
  b. Declarar, crear e iniciar objeto
    Matrices1 objMatrices = new Matrices1()
  c. for ren=0; ren<=4; ren++
    1. for col=0; col<=4; col++
      a. Solicitar matr1[ren][col]
      b. Leer matr1[ren][col]
    2. endfor
  d. endfor
  e. for ren=0; ren<=4; ren++
    1. for col=0; col<=4; col++
      a. Solicitar matr2[ren][col]
      b. Leer matr2[ren][col]
    2. endfor
  f. endfor
  g. Establecer
    objMatrices.establecerMatriz1(matr1)
    objMatrices.establecerMatriz2(matr2)
  h. Calcular objMatrices.calcularMatriz3()
  i. Obtener
    matr12 = objMatrices.obtenerMatriz1()
    matr22 = objMatrices.obtenerMatriz2()
    matr3 = objMatrices.obtenerMatriz3()
  j. for ren=0; ren<=4; ren++
    1. for col=0; col<=4; col++
      a. Imprimir matr12[ren][col]
    2. endfor
  k. endfor
  l. for ren=0; ren<=4; ren++
    1. for col=0; col<=4; col++
      a. Imprimir matr22[ren][col]
    2. endfor
  m. endfor
  n. for ren=0; ren<=4; ren++
    1. for col=0; col<=4; col++
      a. Imprimir matr3[ren][col]
    2. endfor
  o. endfor
  p. Fin Método principal
  Fin Clase EjecutaMatrices1
Fin

```



En la zona de descarga de la Web del libro está disponible:
 Programa en Java: Matrices1.java y EjecutaMatrices1.java

Explicación:

El algoritmo tiene dos clases: la Clase `Matrices1` y la Clase `EjecutaMatrices1`.

En la Clase `Matrices1`:

1. Se declaran los datos que representan la estructura de la clase:
 - `matriz1` que es un arreglo de 5 x 5 elementos de tipo entero para la matriz 1.
 - `matriz2` que es un arreglo de 5 x 5 elementos de tipo entero para la matriz 2.
 - `matriz3` que es un arreglo de 5 x 5 elementos de tipo entero para la matriz 3.
 2. Método `establecerMatriz1(mat1[][]: Entero)`.
Recibe en el parámetro `mat1` los valores que luego coloca en el dato `matriz1`.
 3. Método `establecerMatriz2(mat2[][]: Entero)`.
Recibe en el parámetro `mat2` los valores que luego coloca en el dato `matriz2`.
 4. Método `calcularMatriz3()`.
 - a. Se declaran las variables `r, c` para controlar los ciclos.
 - b. Inicia ciclo for desde `r=0` hasta 4 con incrementos de 1.
 1. Inicia ciclo for desde `c=0` hasta 4 con incrementos de 1.
 - a. Calcula `matriz3[r][c]` sumando `matriz1[r][c]` + `matriz2[r][c]`.
 2. Fin del ciclo for.
 - c. Fin del ciclo for.
 - d. Fin Método `calcularMatriz3`.
 5. Método `obtenerMatriz1(): Arreglo[][] Entero`.
Retorna `matriz1`.
 6. Método `obtenerMatriz2(): Arreglo[][] Entero`.
Retorna `matriz2`.
 7. Método `obtenerMatriz3(): Arreglo[][] Entero`.
Retorna `matriz3`.
- Fin de la Clase `Matrices1`.

En la Clase `EjecutaMatrices1`, en el Método `principal()`:

- a. Se declaran las variables:
 - `matr1` para leer la matriz 1 de números.
 - `matr12` para recibir la matriz 1 de números.
 - `matr2` para leer la matriz 2 de números.
 - `matr22` para recibir la matriz 2 de números.
 - `matr3` para recibir la matriz 3 de números.
 - `ren` y `col` para controlar los ciclos repetitivos.
- b. Se declara el objeto `objMatrices`, usando como base a la Clase `Matrices1`.
Dicho objeto se crea e inicializa mediante el constructor por defecto `Matrices1()`.
- c. Inicia ciclo for desde `ren=0` hasta 4 con incrementos de 1.
 1. Inicia ciclo for desde `col=0` hasta 4 con incrementos de 1.
 - a. Solicita `matr1[ren][col]`.
 - b. Lee en `matr1[ren][col]`.
 2. Fin del ciclo for.
- d. Fin del ciclo for.
- e. Inicia ciclo for desde `ren=0` hasta 4 con incrementos de 1.

1. Inicia ciclo for desde `col=0` hasta 4 con incrementos de 1.
 - a. Solicita `matr2[ren][col]`.
 - b. Lee en `matr2[ren][col]`.
2. Fin del ciclo for.
- f. Fin del ciclo for.
- g. Se llama al Método `establecerMatriz1(matr1)` del objeto `objMatrices` para colocar los valores de `matr1` en el dato `matriz1`.
Se llama al Método `establecerMatriz2(matr2)` del objeto `objMatrices` para colocar los valores de `matr2` en el dato `matriz2`.
- h. Se llama al Método `calcularMatriz3()` del objeto `objMatrices` para calcular los valores de los elementos de la `matriz3`.
- i. Llama al Método `obtenerMatriz1()` del objeto `objMatrices` para obtener el dato `matriz1` y colocarlo en `matr12`, que es como se procesará de aquí en adelante en este método.
Llama al Método `obtenerMatriz2()` del objeto `objMatrices` para obtener el dato `matriz2` y colocarlo en `matr22`, que es como se procesará de aquí en adelante en este método.
Llama al Método `obtenerMatriz3()` del objeto `objMatrices` para obtener el dato `matriz3` y colocarlo en `matr3`, que es como se procesará de aquí en adelante en este método.
- j. Inicia ciclo for desde `ren=0` hasta 4 con incrementos de 1.
 1. Inicia ciclo for desde `col=0` hasta 4 con incrementos de 1.
 - a. Imprime la matriz `matr12[ren][col]`.
 2. Fin del ciclo for.
- k. Fin del ciclo for.
- l. Inicia ciclo for desde `ren=0` hasta 4 con incrementos de 1.
 1. Inicia ciclo for desde `col=0` hasta 4 con incrementos de 1.
 - a. Imprime la matriz `matr22[ren][col]`.
 2. Fin del ciclo for.
- m. Fin del ciclo for.
- n. Inicia ciclo for desde `ren=0` hasta 4 con incrementos de 1.
 1. Inicia ciclo for desde `col=0` hasta 4 con incrementos de 1.
 - a. Imprime la matriz `matr3[ren][col]`.
 2. Fin del ciclo for.
- o. Fin del ciclo for.
- p. Fin Método principal.
Fin de la Clase `EjecutaMatrices1`.
Fin del algoritmo.

Ejercicio 12.2.2

Se tienen los datos de 20 obreros. Por cada obrero se tiene el nombre y la cantidad de unidades fabricadas por cada uno de los seis meses del semestre. Elaborar un algoritmo que lea estos datos en dos arreglos: uno unidimensional para los nombres de los 20 obreros; otro, bidimensional, en el que se tendrán 20 renglones (uno para cada obrero) por 6 columnas (una para la producción de cada mes):

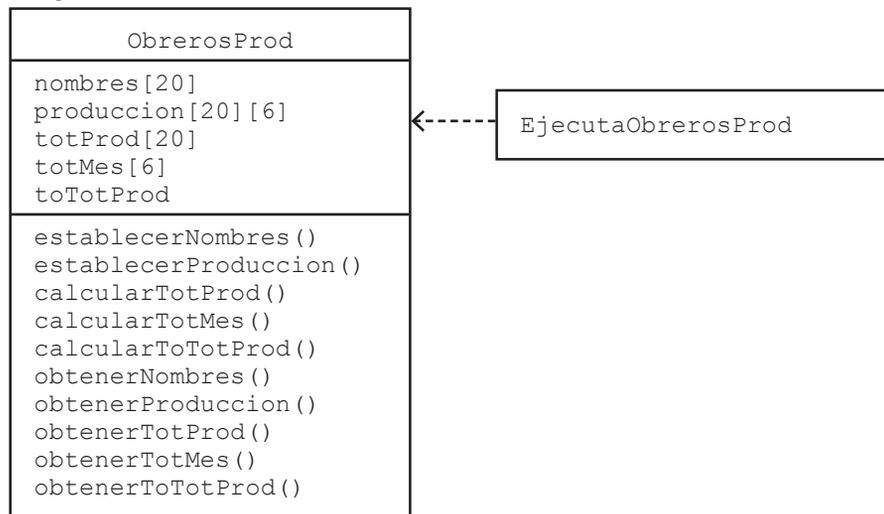
	nombres	produccion					
		0	1	2	3	4	5
0							
1							
2							
-							
19							

Además, se requiere que imprima el reporte siguiente:

NOMBRE DEL OBRERO	REPORTE SEMESTRAL DE PRODUCCIÓN						TOT. PROD.
	MES1	MES2	MES3	MES4	MES5	MES6	
XXXXXXXXXXXXXXXXXXXX	999	999	999	999	999	999	999
XXXXXXXXXXXXXXXXXXXX	999	999	999	999	999	999	999
-							
XXXXXXXXXXXXXXXXXXXX	999	999	999	999	999	999	999
TOTAL							999

A continuación se presenta el algoritmo de la solución:

Diagrama de clases



Algoritmo PRODUCCION 20 OBREROS

Clase ObrerosProd

1. Declarar datos

```

nombres: Arreglo[20] Cadena
produccion: Arreglo[20][6] Entero
totProd: Arreglo[20] Entero
totMes: Arreglo[6] Entero
toTotProd: Entero
  
```

2. Método establecerNombres (nom[]: Cadena)
 - a. nombres = nom
 - b. Fin Método establecerNombres
3. Método establecerProduccion (pro[][]: Entero)
 - a. produccion = pro
 - b. Fin Método establecerProduccion
4. Método calcularTotProd()
 - a. Declarar variables
r, c: Entero
 - b. for r=0; r<=19; r++
 1. totProd[r] = 0
 2. for c=0; c<=5; c++
 - a. totProd[r] = totProd[r] +
produccion[r][c]
 3. endfor
 - c. endfor
 - d. Fin Método calcularTotProd
5. Método calcularTotMes()
 - a. Declarar variables
re, co: Entero
 - b. for co=0; co<=5; co++
 1. totMes[co] = 0
 2. for re=0; re<=19; re++
 - a. totMes[co] = totMes[co] +
produccion[re][co]
 3. endfor
 - c. endfor
 - d. Fin Método calcularTotMes
6. Método calcularToTotProd()
 - a. Declarar variables
i: Entero
 - b. toTotProd = 0
 - c. for i=0; i<=5; i++
 1. toTotProd = toTotProd + totMes[i]
 - d. endfor
 - e. Fin Método calcularToTotProd
7. Método obtenerNombres(): Arreglo[] Cadena
 - a. return nombres
 - b. Fin Método obtenerNombres
8. Método obtenerProduccion(): Arreglo[][] Entero
 - a. return produccion
 - b. Fin Método obtenerProduccion

```
9. Método obtenerTotProd(): Arreglo[] Entero
  a. return totProd
  b. Fin Método obtenerTotProd

10. Método obtenerTotMes(): Arreglo[] Entero
  a. return totMes
  b. Fin Método obtenerTotMes

11. Método obtenerToTotProd(): Entero
  a. return toTotProd
  b. Fin Método obtenerToTotProd
Fin Clase ObrerosProd

Clase EjecutaObrerosProd
1. Método principal()
  a. Declarar variables
    obreros, obreros2: Arreglo[20] Cadena
    prod, prod2: Arreglo[20][6] Entero
    toPro: Arreglo[20] Entero
    toMeses: Arreglo[6] Entero
    ren, col: Entero
  b. Declarar, crear e iniciar objeto
    ObrerosProd objObrero = new ObrerosProd()
  c. for ren=0; ren<=19; ren++
    1. Solicitar nombre del obrero [ren]
    2. Leer obreros[ren]
    3. for col=0; col<=5; col++
      a. Solicitar producción del día [ren][col]
      b. Leer prod[ren][col]
    4. endfor
  d. endfor
  e. Establecer
    objObrero.establecerNombres(obreros)
    objObrero.establecerProduccion(prod)
  f. Calcular objObrero.calcularTotProd()
    objObrero.calcularTotMes()
    objObrero.calcularToTotProd()
  g. Obtener
    obreros2 = objObrero.obtenerNombres()
    prod2 = objObrero.obtenerProduccion()
    toPro = objObrero.obtenerTotProd()
    toMeses = objObrero.obtenerTotMes()
  h. Imprimir encabezado
  i. for ren=0; ren<=19; ren++
    1. Imprimir obreros2[ren]
    2. for col=0; col<=5; col++
      a. Imprimir prod2[ren][col]
    3. endfor
    4. Imprimir toPro[ren]
```

```

        j. endfor
        k. for col=0; col<=5; col++
            1. Imprimir toMeses[col]
        l. endfor
        m. Imprimir objObrero.obtenerToTotProd()
        n. Fin Método principal
    Fin Clase EjecutaObrerosProd
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: ObrerosProd.java y EjecutaObrerosProd.java

Explicación:

El algoritmo tiene dos clases: la Clase ObrerosProd y la Clase EjecutaObrerosProd.

En la Clase ObrerosProd:

1. Se declaran los datos que representan la estructura de la clase:
 - nombres para el arreglo de los nombres de los obreros.
 - produccion para el arreglo de la producción de los obreros.
 - totProd para el arreglo del total de la producción de los obreros.
 - totMes para el arreglo del total de la producción por mes de los obreros.
 - toTotProd el total de la producción de todos los obreros.
2. Método establecerNombres (nom[] : Cadena).
 - Recibe en el parámetro nom los valores que luego coloca en el dato nombres.
3. Método establecerProduccion (pro[][] : Entero).
 - Recibe en el parámetro pro los valores que luego coloca en el dato produccion.
4. Método calcularTotProd().
 - a. Se declaran las variables r y c para controlar los ciclos.
 - b. Inicia ciclo for desde r=0 hasta 19 con incrementos de 1.
 1. Se inicia totProd[r] en 0.
 2. Inicia ciclo for desde c=0 hasta 5 con incrementos de 1.
 - a. Se incrementa totProd[r] con produccion[r][c].
 3. Fin del ciclo for.
 - c. Fin del ciclo for.
 - d. Fin Método calcularTotProd.
5. Método calcularTotMes().
 - a. Se declaran las variables re y co para controlar los ciclos.
 - b. Inicia ciclo for desde co=0 hasta 5 con incrementos de 1.
 1. Se inicia totMes[co] en 0.
 2. Inicia ciclo for desde re=0 hasta 19 con incrementos de 1.
 - a. Se incrementa totMes[co] con produccion[re][co].
 3. Fin del ciclo for.
 - c. Fin del ciclo for.
 - d. Fin Método calcularTotMes.

6. Método `calcularToTotProd()`.
 - a. Se declara la variable `i` para controlar el ciclo.
 - b. Se inicia `toTotProd` en 0.
 - c. Inicia ciclo `for` desde `i=0` hasta 5 con incrementos de 1.
 1. Se incrementa `toTotProd` con `totMes[i]`.
 - d. Fin del ciclo `for`.
 - e. Fin Método `calcularToTotProd`.
7. Método `obtenerNombres(): Arreglo[] Cadena`.
Retorna nombres.
8. Método `obtenerProduccion(): Arreglo[][] Entero`.
Retorna produccion.
9. Método `obtenerTotProd(): Arreglo[] Entero`.
Retorna `totProd`.
10. Método `obtenerTotMes(): Arreglo[] Entero`.
Retorna `totMes`.
11. Método `obtenerToTotProd(): Entero`.
Retorna `toTotProd`.

Fin de la Clase `ObrerosProd`.

En la Clase `EjecutaObrerosProd`, en el Método `principal()`:

- a. Se declaran las variables:
 - `obreros` para leer los nombres de los obreros.
 - `prod` para leer la producción de los obreros.
 - `obreros2` para recibir los nombres de los obreros del objeto.
 - `prod2` para recibir la producción de los obreros del objeto.
 - `toPro` para recibir el total de la producción de los obreros del objeto.
 - `toMeses` para recibir el total por mes de la producción de los obreros del objeto.
 - `ren` y `col` para controlar los ciclos repetitivos.
- b. Se declara el objeto `objObrero`, usando como base a la Clase `ObrerosProd`.
Dicho objeto se crea e inicializa mediante el constructor por defecto `ObrerosProd()`.
- c. Inicia ciclo `for` desde `ren=0` hasta 19 con incrementos de 1.
 1. Solicita el nombre del obrero.
 2. Se lee en `obreros[ren]`.
 3. Inicia ciclo `for` desde `col=0` hasta 5 con incrementos de 1.
 - a. Solicita la producción del día.
 - b. Se lee en `prod[ren][col]`.
 4. Fin del ciclo `for`.
- d. Fin del ciclo `for`.
- e. Se llama al Método `establecerNombres(obreros)` del objeto `objObrero` para colocar el valor de `obreros` en el dato `nombres`.
Se llama al Método `establecerProduccion(prod)` del objeto `objObrero` para colocar el valor de `prod` en el dato `produccion`.
- f. Se llama al Método `calcularTotProd()` del objeto `objObrero` en el cual calcula el total de la producción de cada uno de los 20 obreros.
Se llama al Método `calcularTotMes()` del objeto `objObrero` en el cual calcula el total de la producción de cada mes de los 20 obreros.

Se llama al Método `calcularToTotProd()` del objeto `objObrero` en el cual calcula el total del total de la producción de los 20 obreros.

- g. Llama al Método `obtenerNombres()` del objeto `objObrero` para obtener el dato `nombres` y colocarlo en `obreros2`, que es como se procesará de aquí en adelante en este método.

Llama al Método `obtenerProduccion()` del objeto `objObrero` para obtener el dato `produccion` y colocarlo en `prod2`, que es como se procesará de aquí en adelante en este método.

Llama al Método `obtenerTotProd()` del objeto `objObrero` para obtener el dato `totProd` y colocarlo en `toPro`, que es como se procesará de aquí en adelante en este método.

Llama al Método `toMeses()` del objeto `objObrero` para obtener el dato `toMes` y colocarlo en `toMeses`, que es como se procesará de aquí en adelante en este método.

- h. Imprime encabezado.
- i. Inicia ciclo for desde `ren=0` hasta 19 con incrementos de 1.
1. Imprime el nombre del obrero que está en `obreros2[ren]`.
 2. Inicia ciclo for desde `col=0` hasta 5 con incrementos de 1.
 - a. Imprime la producción `prod2[ren][col]`.
 3. Fin del ciclo for.
 4. Imprime el total de la producción del obrero `toPro[ren]`.
- j. Fin del ciclo for.
- k. Inicia ciclo for desde `col=0` hasta 5 con incrementos de 1.
1. Imprime el total de la producción del mes `toMeses[col]`.
- l. Fin del ciclo for.
- m. Se llama al Método `obtenerToTotProd()` del objeto `objObrero` para acceder e imprimir el valor del dato `toTotProd`.
- n. Fin del Método principal.
- Fin de la Clase `EjecutaObrerosProd`.
- Fin del algoritmo.



A este problema se le puede agregar que al final imprima los nombres del obrero más productivo y del menos productivo.

Nota: A este problema se le puede agregar que al final imprima los nombres del obrero más productivo y del menos productivo.

Tabla 12.2: muestra los ejercicios resueltos disponibles en la zona de descarga del capítulo 12 de la Web del libro.

Ejercicio	Descripción
Ejercicio 12.2.3	Multiplica matriz por vector columna.
Ejercicio 12.2.4	Suma por renglones y columnas una matriz de números.

12.2.1 Ejercicios propuestos para arreglos bidimensionales

Como ejercicios propuestos para este apartado se recomiendan, del capítulo 6, algunos de los ejercicios resueltos que no fueron incluidos aquí, además de todos los ejercicios propuestos en dicho capítulo sobre arreglos bidimensionales.

12.3 Diseño de algoritmos OO usando arreglos tridimensionales

En este punto se utilizarán los arreglos tridimensionales en pseudocódigo, mismos que se estudiaron en el capítulo 6, pero ahora aplicados conjuntamente con el diagrama de clases y los conceptos de la programación orientada a objetos, es decir, en el diseño de algoritmos orientados a objetos.

Ejercicio 12.3.1

Una compañía manufacturera tiene 6 plantas, en cada planta tiene 4 estaciones de trabajo y por cada estación de trabajo se tiene la producción (número de unidades fabricadas) de cada uno de los 5 días laborables de la semana. Elaborar un algoritmo que lea los datos de la producción en un arreglo de tres dimensiones. La primera dimensión la conforman las plantas, la segunda dimensión, las estaciones de trabajo y la tercera, los días de producción. Esquemáticamente:

		Día				
		0	1	2	3	4
Planta 0	Estación 0					
	Estación 1					
	Estación 2					
	Estación 3					
Planta 1						
...						
Planta 5						

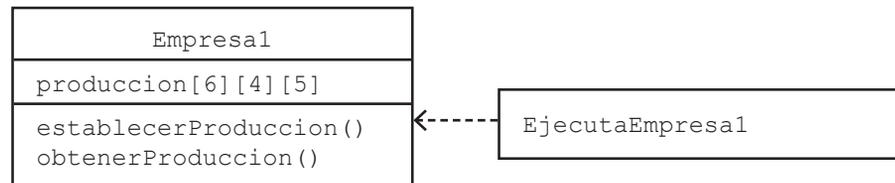
Una vez leídos los datos, que imprima el siguiente reporte:

REPORTE SEMANAL DE PRODUCCIÓN						
PLANTA 1						
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	TOTAL
ESTACIÓN-1	--	--	--	--	--	--
ESTACIÓN-2	--	--	--	--	--	--
ESTACIÓN-3	--	--	--	--	--	--
ESTACIÓN-4	--	--	--	--	--	--
TOTALES	--	--	--	--	--	--

PLANTA 6						
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	TOTAL
OBRAERO-1	--	--	--	--	--	--
OBRAERO-2	--	--	--	--	--	--
OBRAERO-3	--	--	--	--	--	--
OBRAERO-4	--	--	--	--	--	--
TOTALES	--	--	--	--	--	--
TOTAL GENERAL DE PRODUCCIÓN						--

A continuación se presenta el algoritmo de la solución:

Diagrama de clases



Algoritmo ARREGLO TRES DIMENSIONES

Clase Empresal

1. Declarar datos

 produccion: Arreglo[6][4][5] Entero

2. Método establecerProduccion(pro[][][]: Entero)

 a. produccion = pro

 b. Fin Método establecerProduccion

3. Método obtenerProduccion(): Arreglo[][][] Entero

 a. return produccion

 b. Fin Método obtenerProduccion

Fin Clase Empresal

Clase EjecutaEmpresal

1. Método principal()

 a. Declarar variables

 prod, prod2: Arreglo[6][4][5] Entero

 pla, est, dia, totEst, totDia, totPlanta,

 totProd: Entero

```

b. Declarar, crear e iniciar objeto
   Empresa1 objEmpresa = new Empresa1()
c. for pla=0; pla<=5; pla++
   1. for est=0; est<=3; est++
      a. for dia=0; dia<=4; dia++
         1. Solicitar prod[pla][est][dia]
         2. Leer prod[pla][est][dia]
      b. endfor
   2. endfor
d. endfor
e. Establecer
   objEmpresa.establecerProduccion(prod)
f. Obtener
   prod2 = objEmpresa.obtenerProduccion()
g. Imprimir primer renglón del encabezado
h. totProd = 0
i. for pla=0; pla<=5; pla++
   1. Imprimir encabezado
   2. for est=0; est<=3; est++
      a. totEst = 0
      b. for dia=0; dia<=4; dia++
         1. Imprimir prod2[pla][est][dia]
         2. totEst = totEst +
            prod2[pla][est][dia]
      c. endfor
      d. Imprimir totEst
   3. endfor
   4. totPlanta = 0
   5. for dia=0; dia<=4; dia++
      a. totDia = 0
      b. for est=0; est<=3; est++
         1. totDia = totDia +
            prod2[pla][est][dia]
      c. endfor
      d. Imprimir totDia
      e. totPlanta = totPlanta + totDia
   6. endfor
   7. Imprimir totPlanta
   8. totProd = totProd + totPlanta
j. endfor
k. Imprimir totProd
l. Fin Método principal
Fin Clase EjecutaEmpresa1
Fin

```

En la zona de descarga de la Web del libro está disponible:
 Programa en Java: Empresa1.java y EjecutaEmpresa1.java



Explicación:

El algoritmo tiene dos clases: la Clase `Empresal` y la Clase `EjecutaEmpresal`.

En la Clase `Empresal`:

1. Se declaran los datos que representan la estructura de la clase:
`produccion` para la producción de las 6 plantas de las 4 estaciones de trabajo y los 5 días hábiles de la semana.
 2. Método `establecerProduccion(pro[][][]: Entero)`.
Recibe en el parámetro `pro` los valores que luego coloca en el dato `produccion`.
 3. Método `obtenerProduccion(): Arreglo[][][] Entero`.
Retorna `produccion`.
- Fin de la Clase `Empresal`.

En la Clase `EjecutaEmpresal`, en el Método `principal()`:

- a. Se declaran las variables:
`prod` para leer la producción de las 6 plantas de las 4 estaciones de trabajo y los 5 días hábiles de la semana.
`prod2` para recibir la producción de las 6 plantas de las 4 estaciones de trabajo y los 5 días hábiles de la semana que regresa del objeto.
`pla`, `est` y `dia` para controlar los ciclos repetitivos.
`totEst` para calcular el total de producción de cada estación.
`totDia` para calcular el total de producción de cada día.
`totPlanta` para calcular el total de producción de cada planta.
`totProd` para calcular el total de producción de las 6 plantas.
- b. Se declara el objeto `objEmpresa`, usando como base a la Clase `Empresal`.
Dicho objeto se crea e inicializa mediante el constructor por defecto `Empresal()`.
- c. Inicia ciclo for desde `pla=0` hasta 5 con incrementos de 1,
 1. Inicia ciclo for desde `est=0` hasta 3 con incrementos de 1,
 - a. Inicia ciclo for desde `dia=0` hasta 4 con incrementos de 1,
 1. Solicita producción de planta, estación y día,
 2. Lee en `prod[pla][est][dia]`,
 - b. Fin del ciclo for,
 2. Fin del ciclo for
- d. Fin del ciclo for.
- e. Se llama al Método `establecerProduccion(prod)` del objeto `objEmpresa` para colocar los valores de `prod` en el dato `produccion`.
- f. Llama al Método `obtenerProduccion()` del objeto `objEmpresa` para obtener el dato `produccion` y colocarlo en `prod2`, que es como se procesará de aquí en adelante en este método.
- g. Imprimir primer renglón del encabezado.
- h. Inicia `totProd` en 0.
- i. Inicia ciclo for desde `pla=0` hasta 5 con incrementos de 1.
 1. Imprimir encabezado.
 2. Inicia ciclo for desde `est=0` hasta 3 con incrementos de 1.

- a. Inicia totEst en 0.
 - b. Inicia ciclo for desde dia=0 hasta 4 con incrementos de 1.
 1. Imprimir prod2[pla][est][dia].
 2. Incrementa totEst con prod2[pla][est][dia].
 - c. Fin del ciclo for.
 - d. Imprime totEst.
 3. Fin del ciclo for.
 4. Inicia totPlanta en 0.
 5. Inicia ciclo for desde dia=0 hasta 4 con incrementos de 1.
 - a. Inicia totDia en 0.
 - b. Inicia ciclo for desde est=0 hasta 3 con incrementos de 1.
 1. Incrementa totDia con prod2[pla][est][dia].
 - c. Fin del ciclo for.
 - d. Imprime totDia.
 - e. Incrementa totPlanta con totDia.
 6. Fin del ciclo for.
 7. Imprime totPlanta.
 8. Incrementa totProd con totPlanta.
 - j. Fin del ciclo for.
 - k. Imprime totProd.
 - l. Fin Método principal.
- Fin de la Clase EjecutaEmpresa1.
Fin del algoritmo.

12.3.1 Ejercicios propuestos para arreglos tridimensionales

Como ejercicios propuestos para este apartado se recomiendan, del capítulo 6, algunos de los ejercicios resueltos que no fueron incluidos aquí, además de todos los ejercicios propuestos en dicho capítulo sobre arreglos tridimensionales.

12.4 Diseño de algoritmos OO usando arreglos tetradimensionales

En este punto se utilizarán los arreglos tetradimensionales en pseudocódigo, mismos que se estudiaron en el capítulo 6, pero ahora aplicados conjuntamente con el diagrama de clases y los conceptos de la programación orientada a objetos, es decir, en el diseño de algoritmos orientados a objetos.

Ejercicio 12.4.1

Una compañía manufacturera tiene 6 plantas; en cada planta tiene 3 estaciones de trabajo, en cada estación de trabajo se tienen 4 obreros y por cada obrero se tiene la producción (número de unidades fabricadas) de cada uno de los 5 días laborables de la semana. Elaborar un algoritmo que lea los datos de la producción en un arreglo de cuatro dimensiones. La primera dimensión la conforman las plantas, la segunda dimensión, las estaciones de trabajo, la tercera dimensión, los obreros y la cuarta, los días de producción. Esquemáticamente:

		Estación 1					Estación 2	Estación 3
		Día						
		1	2	3	4	5		
Planta 1	Obrero 1							
	Obrero 2							
	Obrero 3							
	Obrero 4							
Planta 2								
...								
Planta 6								

REPORTE SEMANAL DE PRODUCCIÓN

PLANTA 1						
ESTACIÓN 1						
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	TOTAL
OBRERO-1	--	--	--	--	--	--
OBRERO-2	--	--	--	--	--	--
OBRERO-3	--	--	--	--	--	--
OBRERO-4	--	--	--	--	--	--
TOTALES	--	--	--	--	--	--

ESTACIÓN 2						
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	TOTAL
OBRERO-1	--	--	--	--	--	--
OBRERO-2	--	--	--	--	--	--
OBRERO-3	--	--	--	--	--	--
OBRERO-4	--	--	--	--	--	--
TOTALES	--	--	--	--	--	--

ESTACIÓN 3						
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	TOTAL
OBRERO-1	--	--	--	--	--	--
OBRERO-2	--	--	--	--	--	--
OBRERO-3	--	--	--	--	--	--
OBRERO-4	--	--	--	--	--	--
TOTALES	--	--	--	--	--	--

--

PLANTA 6						
ESTACIÓN 1						
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	TOTAL
OBRERO-1	--	--	--	--	--	--
OBRERO-2	--	--	--	--	--	--
OBRERO-3	--	--	--	--	--	--
OBRERO-4	--	--	--	--	--	--
TOTALES	--	--	--	--	--	--

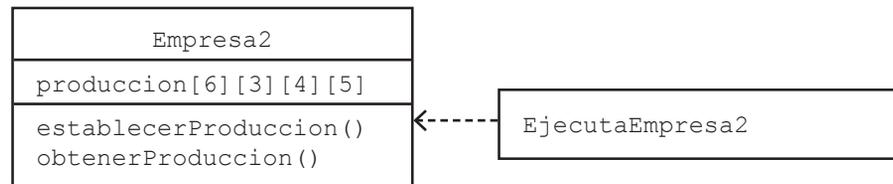
ESTACIÓN 2						
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	TOTAL
OBRERO-1	--	--	--	--	--	--
OBRERO-2	--	--	--	--	--	--
OBRERO-3	--	--	--	--	--	--
OBRERO-4	--	--	--	--	--	--
TOTALES	--	--	--	--	--	--

ESTACIÓN 3						
	DÍA 1	DÍA 2	DÍA 3	DÍA 4	DÍA 5	TOTAL
OBRERO-1	--	--	--	--	--	--
OBRERO-2	--	--	--	--	--	--
OBRERO-3	--	--	--	--	--	--
OBRERO-4	--	--	--	--	--	--
TOTALES	--	--	--	--	--	--

TOTAL GENERAL DE PRODUCCION						--
------------------------------------	--	--	--	--	--	----

A continuación se presenta el algoritmo de la solución:

Diagrama de clases



Algoritmo ARREGLO CUATRO DIMENSIONES

Clase Empresa2

1. Declarar datos
 - produccion: Arreglo[6][3][4][5] Entero
2. Método establecerProduccion(pro[][][] []: Entero)
 - a. produccion = pro
 - b. Fin Método establecerProduccion
3. Método obtenerProduccion(): Arreglo[][][] [] Entero
 - a. return produccion
 - b. Fin Método obtenerProduccion

Fin Clase Empresa2

Clase EjecutaEmpresa2

1. Método principal()
 - a. Declarar variables
 - prod, prod2: Arreglo[6][3][4][5] Entero
 - pla, est, obr, dia, totEst, totDia, totObr,
 - totPlanta, totProd: Entero
 - b. Declarar, crear e iniciar objeto
 - Empresa2 objEmpresa = new Empresa2()
 - c. for pla=0; pla<=5; pla++
 1. for est=0; est<=2; est++
 - a. for obr=0; obr<=3; obr++
 1. for dia=0; dia<=4; dia++
 - a. Solicitar prod[pla][est][obr][dia]
 - b. Leer prod[pla][est][obr][dia]
 2. endfor
 - b. endfor
 2. endfor
 - d. endfor
 - e. Establecer
 - objEmpresa.establecerProduccion(prod)
 - f. Obtener
 - prod2 = objEmpresa.obtenerProduccion()
 - g. Imprimir primer renglón del encabezado
 - h. totProd = 0
 - i. for pla=0; pla<=5; pla++

```

1. Imprimir encabezado de planta
2. totPlanta = 0
3. for est=0; est<=2; est++
    a. Imprimir encabezado de estación
    b. for obr=0; obr<=3; obr++
        1. Imprimir 'OBRERO-',obr
        2. totObr = 0
        3. for dia=0; dia<=4; dia++
            a. Imprimir
                prod2[pla][est][obr][dia]
            b. totObr = totObr +
                prod2[pla][est][obr][dia]
        4. endfor
        5. Imprimir totObr
    c. endfor
    d. totEst = 0
    e. for dia=0; dia<=4; dia++
        1. totDia = 0
        2. for obr=0; obr<=3; obr++
            a. totDia = totDia +
                prod2[pla][est][obr][dia]
        3. endfor
        4. Imprimir totDia
        5. totEst = totEst + totDia
    f. endfor
    g. Imprimir totEst
    h. totPlanta = totPlanta + totEst
4. endfor
5. Imprimir totPlanta
6. totProd = totProd + totPlanta
j. endfor
k. Imprimir TotProd
1. Fin Método principal
Fin Clase EjecutaEmpresa2
Fin

```

En la zona de descarga de la Web del libro está disponible:
Programa en Java: Empresa2.java y EjecutaEmpresa2.java



Explicación:

El algoritmo tiene dos clases: la Clase Empresa2 y la Clase EjecutaEmpresa2.

En la Clase Empresa2:

1. Se declaran los datos que representan la estructura de la clase:
produccion para la producción de las 6 plantas de las 3 estaciones de trabajo de los 4 obreros y los 5 días hábiles de la semana.

2. Método establecerProduccion (pro [] [] [] []: Entero).
Recibe en el parámetro pro los valores que luego coloca en el dato produccion.
 3. Método obtenerProduccion(): Arreglo [] [] [] [] Entero.
Retorna produccion.
- Fin de la Clase Empresa2.

En la Clase EjecutaEmpresa2, en el Método principal():

- a. Se declaran las variables:
 - prod para leer la producción de las 6 plantas de las 3 estaciones de trabajo de los 4 obreros y los 5 días hábiles de la semana.
 - prod2 para recibir la producción de las 6 plantas de las 3 estaciones de trabajo de los 4 obreros y los 5 días hábiles de la semana que regresa del objeto.
 - pla, est, obr y dia para controlar los ciclos repetitivos.
 - totEst para calcular el total de producción de cada estación.
 - totDia para calcular el total de producción de cada día.
 - totObr para calcular el total de producción de cada obrero.
 - totPlanta para calcular el total de producción de cada planta.
 - totProd para calcular el total de producción de las 6 plantas.
- b. Se declara el objeto objEmpresa, usando como base a la Clase Empresa2. Dicho objeto se crea e inicializa mediante el constructor por defecto Empresa2().
- c. Inicia ciclo for desde pla=0 hasta 5 con incrementos de 1.
 1. Inicia ciclo for desde est=0 hasta 2 con incrementos de 1.
 - a. Inicia ciclo for desde obr=0 hasta 3 con incrementos de 1.
 1. Inicia ciclo for desde dia=0 hasta 4 con incrementos de 1.
 - a. Solicita producción de planta, estación, obrero y día.
 - b. Lee en prod[pla][est][obr][dia].
 2. Fin del ciclo for.
 - b. Fin del ciclo for.
 2. Fin del ciclo for.
- d. Fin del ciclo for.
- e. Se llama al Método establecerProduccion(prod) del objeto objEmpresa para colocar los valores de prod en el dato produccion.
- f. Llama al Método obtenerProduccion() del objeto objEmpresa para obtener el dato produccion y colocarlo en prod2, que es como se procesará de aquí en adelante en este método.
- g. Imprimir primer renglón del encabezado.
- h. Inicia totProd en 0.
- i. Inicia ciclo for desde pla=0 hasta 5 con incrementos de 1.
 1. Imprimir encabezado de planta.
 2. Inicia totPlanta en 0.
 3. Inicia ciclo for desde est=0 hasta 2 con incrementos de 1.
 - a. Imprimir encabezado de estación.
 - b. Inicia ciclo for desde obr=0 hasta 3 con incrementos de 1.
 1. Imprime 'OBRERO-', obr.
 2. Inicia totObr en 0.

3. Inicia ciclo for desde `dia=0` hasta 4 con incrementos de 1.
 - a. Imprime `prod2[pla][est][obr][dia]`.
 - b. Incrementa `totObr` con `prod2[pla][est][obr][dia]`.
 4. Fin del ciclo for.
 5. Imprime `totObr`.
 - c. Fin del ciclo for.
 - d. Inicia `totEst` en 0.
 - e. Inicia ciclo for desde `dia=0` hasta 4 con incrementos de 1.
 1. Inicia `totDia` en 0.
 2. Inicia ciclo for desde `obr=0` hasta 3 con incrementos de 1.
 - a. Incrementa `totDia` con `prod2[pla][est][obr][dia]`.
 3. Fin del ciclo for.
 4. Imprime `totDia`.
 5. Incrementa `totEst` con `totDia`.
 - f. Fin del ciclo for.
 - g. Imprime `totEst`.
 - h. Incrementa `totPlanta` con `totEst`.
 4. Fin del ciclo for.
 5. Imprimir `totPlanta`.
 6. Incrementa `totProd` con `totPlanta`.
 - j. Fin del ciclo for.
 - k. Imprime `TotProd`.
 - l. Fin Método principal.
- Fin de la Clase `EjecutaEmpresa2`.
Fin del algoritmo.

12.4.1 Ejercicios propuestos para arreglos tetradimensionales

Como ejercicios propuestos para este apartado se recomiendan, del capítulo 6, algunos de los ejercicios resueltos que no fueron incluidos aquí, además de todos los ejercicios propuestos en dicho capítulo sobre arreglos tetradimensionales.

12.5 Resumen de conceptos que debe dominar

- Diseño de algoritmos orientados a objetos usando arreglos
 - unidimensionales
 - bidimensionales
 - tridimensionales
 - tetradimensionales

12.6 Contenido de la página Web de apoyo

El material marcado con asterisco () sólo está disponible para docentes.*

12.6.1 Resumen gráfico del capítulo

12.6.2 Autoevaluación

12.6.3 Programas en Java

12.6.4 Ejercicios resueltos

12.6.5 Power Point para el profesor (*)

13

Programación orientada a objetos usando herencia

Contenido

- 13.1 Herencia
- 13.2 Diseño del diagrama de clases con herencia
 - 13.2.1 Superclases y subclases
- 13.3 Diseño de algoritmos OO usando herencia
- 13.4 Ejercicios resueltos
- 13.5 Ejercicios propuestos
- 13.6 Resumen de conceptos que debe dominar
- 13.7 Contenido de la página Web de apoyo
 - El material marcado con asterisco (*) sólo está disponible para docentes.*
 - 13.7.1 Resumen gráfico del capítulo
 - 13.7.2 Autoevaluación
 - 13.7.3 Programas en Java
 - 13.7.4 Ejercicios resueltos
 - 13.7.5 Power Point para el profesor (*)

Objetivos del capítulo

- Aprender el concepto de herencia y su uso en el diseño de algoritmos orientados a objetos.
- Aplicar lo aprendido en la solución de ejercicios resueltos y propuestos.

Competencias

- Competencia general del capítulo
 - *Analizar problemas y diseñar algoritmos que los solucionen aplicando la arquitectura orientada a objetos y la herencia.*
- Competencias específicas del capítulo
 - Define el concepto de herencia.
 - Describe el diseño del diagrama de clases con herencia.
 - Define los conceptos de superclases y subclases.
 - Diseña algoritmos orientados a objetos aplicando herencia.

Introducción

Con el estudio del capítulo anterior, usted ya domina la estructura de datos denominada arreglos, inmersa en la arquitectura orientada a objetos.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos usando la herencia, que es uno de los conceptos fundamentales de la programación orientada a objetos.

Se explica cómo diseñar algoritmos orientados a objetos usando la herencia.

Se expone el mecanismo de herencia, el cual consiste en que objetos descendientes heredan las características y comportamiento de lo que se llama objetos ancestros, aplicado en el contexto de la programación de computadoras.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejercite estudiando los problemas planteados en los ejercicios resueltos y propuestos. Al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro, luego verifique sus resultados con los del libro, analice las diferencias y vea sus errores. Al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no está igual que el del libro, no necesariamente está mal. Usted debe ir aprendiendo a analizar las diferencias y a comprender que a veces, aunque haya diferencias, las dos soluciones están correctas.

En el siguiente capítulo se estudia el polimorfismo, que es uno de los conceptos fundamentales de la programación orientada a objetos.

13.1 Herencia

La idea fundamental de la programación orientada a objetos es identificar los objetos presentes en nuestro problema y sus relaciones para formar una jerarquía de objetos: algo similar a un árbol genealógico; es decir, primero identificar al objeto más genérico de la especie y luego identificar otros objetos derivados de este primer objeto, y así sucesivamente, de esos nuevos objetos identificar otros objetos derivados de ellos. Los objetos derivados son objetos descendientes que heredan las características y comportamiento de lo que se llama objetos ancestros (o sus ascendientes).

Ejemplo:

En cierta empresa se tienen empleados, los cuales se dividen en dos tipos: empleados por horas, a los que se les paga de acuerdo al número de horas trabajadas y a una cuota que se les paga por hora, y empleados asalariados, a quienes se les paga de acuerdo a un sueldo fijo mensual.

En este problema podemos identificar dos objetos:

Un objeto es: EmpleadoPorHoras, con los datos:

nombreEmp	Nombre del empleado.
deptoEmp	Departamento en el que labora.
puestoEmp	Puesto que desempeña.
horasTrab	Número de horas que trabajó.

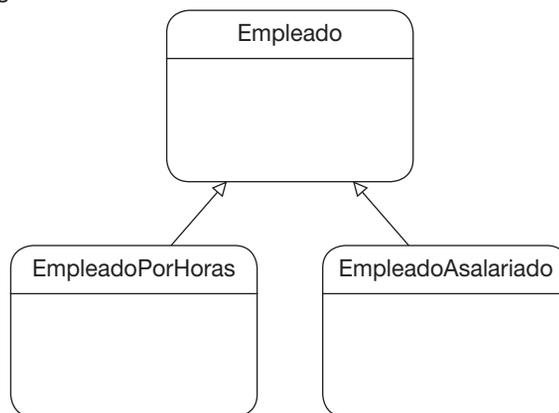
cuotaHora	Cuota que se le paga por hora.
sueldoQnaHoras	Sueldo quincenal del empleado por horas, calculado con horas trabajadas y cuota por hora.

Y el otro objeto es: EmpleadoAsalariado, con los datos:

nombreEmp	Nombre del empleado.
deptoEmp	Departamento en el que labora.
puestoEmp	Puesto que desempeña.
sueldoMensual	Sueldo fijo mensual que se le paga.
sueldoQnaAsal	Sueldo quincenal del empleado asalariado, calculado con el sueldo mensual.

Podemos apreciar que los objetos EmpleadoPorHoras y EmpleadoAsalariado tienen en común los datos nombreEmp, deptoEmp y puestoEmp. El resto de los datos son diferentes.

Cuando se tiene una situación como la expuesta, podemos utilizar el mecanismo de herencia mediante un proceso de abstracción en el que definimos un objeto más abstracto que contenga lo que es común a los demás objetos para, a partir de éste, derivar a los otros objetos mediante el mecanismo de herencia, como se muestra en el siguiente diagrama:



Explicación:

La flecha indica que “se deriva de” o bien que “hereda de”; por ejemplo, que EmpleadoPorHoras hereda de Empleado y EmpleadoAsalariado hereda de Empleado.

Se tiene el objeto más genérico: Empleado, con los datos:

nombreEmp	Nombre del empleado.
deptoEmp	Departamento en el que labora.
puestoEmp	Puesto que desempeña.

Se tiene el objeto: EmpleadoPorHoras, con los datos:

horasTrab	Número de horas que trabajó.
cuotaHora	Cuota que se le paga por hora.
sueldoQnaHoras	Sueldo quincenal del empleado por horas.

Y, mediante el mecanismo de herencia, “hereda de” Empleado los datos de éste; por tanto, no es necesario definirlos aquí.

También se tiene el objeto: EmpleadoAsalariado, con los datos:

SueldoMensual Sueldo fijo mensual que se le paga.

sueldoQnaAsal Sueldo quincenal del empleado asalariado.

Que mediante el mecanismo de herencia, “hereda de” Empleado los datos de éste; por tanto, no es necesario definirlos aquí.

Con la jerarquía de objetos antes esquematizada, tenemos que el objeto EmpleadoPorHoras es un objeto descendiente de Empleado, y lo mismo el objeto EmpleadoAsalariado.

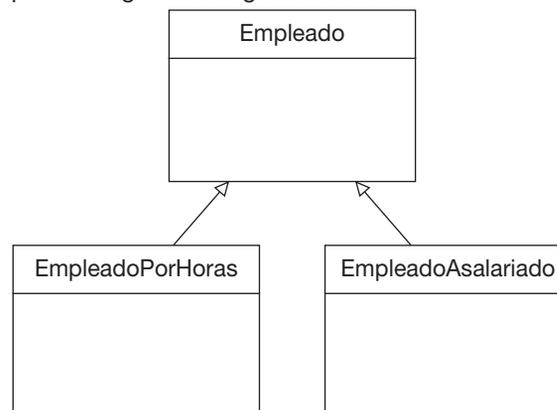
Los objetos descendientes heredan las características y comportamiento de lo que se llama objetos ancestros. Esto quiere decir que si tenemos el objeto Empleado con los datos ya detallados, al definir el objeto EmpleadoPorHoras éste hereda todo lo que contiene su ancestro Empleado y sólo resta por definir lo que es específico del objeto EmpleadoPorHoras, que son los datos horasTrab, cuotaHora y sueldoQnaHoras. Es decir que, al identificarse como descendiente de Empleado, ya no es necesario definir las características de Empleado, porque las hereda de éste.

Al definir el objeto EmpleadoAsalariado, éste hereda los datos de su ancestro Empleado, y sólo resta por definir lo que es específico del objeto EmpleadoAsalariado, que es sueldoMensual y sueldoQnaAsal.

En resumen, la programación orientada a objetos es el proceso de construir árboles genealógicos para estructuras de datos. De ese modo, una de las cosas más importantes que la programación orientada a objetos adiciona a los lenguajes tradicionales es el mecanismo de herencia, mediante el cual tipos de datos heredan las características de tipos más simples y más genéricos.

13.2 Diseño del diagrama de clases con herencia

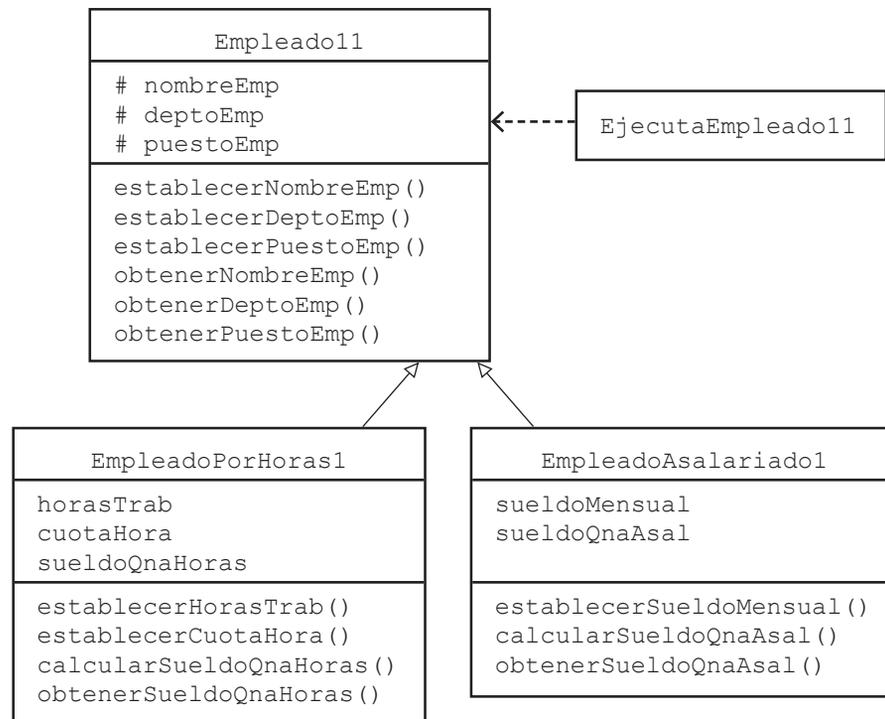
Ahora se debe dibujar el diagrama de clases tomando como base el diagrama de jerarquía de objetos elaborado líneas antes. Como ya se explicó, cada objeto identificado se representa mediante una clase y, utilizando el rectángulo para representar las clases, nos queda el siguiente diagrama de clases:



Como se puede observar, se ha formado una jerarquía de clases donde tenemos una primera clase, la clase principal o más general: la clase Empleado, a la cual se le conoce como la superclase. Luego, de la superclase Empleado se derivan dos clases, a las que se les llama subclases porque son clases que se derivan de otra clase; éstas son: la clase EmpleadoPorHoras y la clase EmpleadoAsalariado.

Nota: Recuerde que la flecha indica que “se deriva de” o bien que “hereda de”; por ejemplo, que la clase EmpleadoPorHoras “hereda de” Empleado y la clase EmpleadoAsalariado “hereda de” Empleado.

Aplicando el concepto de diseñar la estructura de cada clase, nos queda el siguiente diagrama de clases:



Explicación:

Se tiene la clase controlador EjecutaEmpleado11, la cual utiliza al modelo, que está formado por tres clases jerarquizadas:

1. La clase Empleado11, que es la superclase o clase principal, que tiene los datos nombreEmp, deptoEmp y puestoEmp, y los métodos establecerNombreEmp(), establecerDeptoEmp(), establecerPuestoEmp(), obtenerNombreEmp(), obtenerDeptoEmp() y obtenerPuestoEmp() para establecer y obtener cada uno de los datos respectivamente. Empleado11 es la superclase que se usa para derivar subclases a través del mecanismo de herencia; es por ello que a sus datos se le antepuso el símbolo #, el cual indica que el dato es protegido (protected); los datos deben ser protegidos para que se puedan heredar.



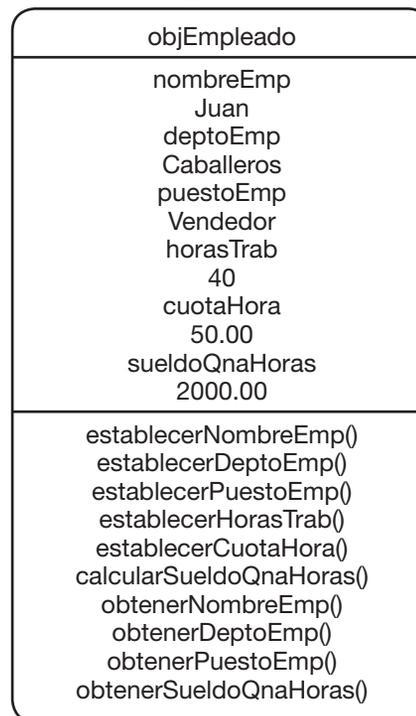
Recuerde que la flecha indica que “se deriva de” o bien que “hereda de”; por ejemplo, que la clase EmpleadoPorHoras “hereda de” Empleado y la clase EmpleadoAsalariado “hereda de” Empleado.

2. La clase `EmpleadoPorHoras1`, que es una subclase que se deriva de la superclase `Empleado11`, tiene los datos `horasTrab`, `cuotaHora` y `suedoQnaHoras`, y los métodos `establecerHorasTrab()`, `establecerCuotaHora()`, `calcularSueldoQnaHoras()` y `obtenerSueldoQnaHoras()` para establecer las horas trabajadas, establecer la cuota por hora, calcular el sueldo quincenal del empleado por horas y obtener el valor del sueldo quincenal respectivamente. Al derivarse de `Empleado11`, `EmpleadoPorHoras1` hereda los datos y métodos de `Empleado11` a través del mecanismo de herencia.

3. La clase `EmpleadoAsalariado1`, que es una subclase que se deriva de la superclase `Empleado11`, tiene los datos `sueldoMensual` y `suedoQnaAsal`, y los métodos `establecerSueldoMensual()`, `calcularSueldoQnaAsal()` y `obtenerSueldoQnaAsal()` para establecer el sueldo mensual, calcular el sueldo quincenal del empleado asalariado y para obtener el sueldo quincenal respectivamente. Al derivarse de `Empleado11`, `EmpleadoAsalariado1` hereda los datos y métodos de `Empleado11` a través del mecanismo de herencia.

13.2.1 Superclases y subclases

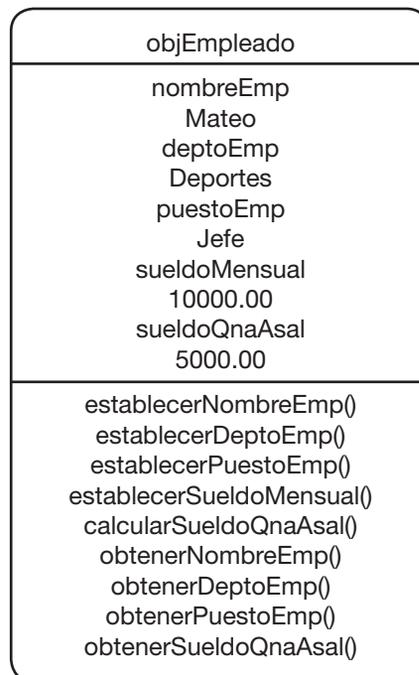
Las superclases no se usan para ser instanciadas, es decir, no se usan para generar objetos a partir de ellas. Las superclases se utilizan para derivar otras clases (subclases), formando jerarquías de clases a través del mecanismo de herencia. En nuestro ejemplo, `Empleado11` es una superclase que se utiliza para derivar las subclases `EmpleadoPorHoras1` y `EmpleadoAsalariado1`. En consecuencia, la clase `Empleado11` no se usa para generar objetos; sólo se generan objetos de las subclases `EmpleadoPorHoras1` y `EmpleadoAsalariado1`. Así, al generar un objeto de la clase `EmpleadoPorHoras1` tendrá la siguiente estructura:



Explicación:

Al crear un objeto o instancia de la clase EmpleadoPorHoras1, el objeto generado tiene los datos y métodos de la clase EmpleadoPorHoras1 y también los datos y métodos de la clase Empleado11 porque, al diseñar el diagrama de clases, existe una relación de herencia entre Empleado11, que es la superclase, y EmpleadoPorHoras1, que es una subclase que se deriva de Empleado11.

Así, al generar un objeto de la clase EmpleadoAsalariado1 éste tendrá la siguiente estructura:

*Explicación:*

Al crear un objeto o instancia de la clase EmpleadoAsalariado1, el objeto generado tiene los datos y métodos de la clase EmpleadoAsalariado1 y también los datos y métodos de la clase Empleado11 porque, al diseñar el diagrama de clases, existe una relación de herencia entre Empleado11, que es la superclase, y EmpleadoAsalariado1, que es una subclase que se deriva de Empleado11.

13.3 Diseño de algoritmos OO usando herencia

En este apartado se toma como base el diagrama de clases diseñado en el punto anterior y se procede a diseñar el algoritmo en pseudocódigo; es decir, se diseña la lógica de cada clase usando pseudocódigo, involucrando el concepto de herencia. A continuación se tiene el algoritmo:

Algoritmo CALCULAR SUELDO DE VARIOS EMPLEADOS

Clase Empleado1

1. Declarar datos
 - # nombreEmp: Cadena
 - # deptoEmp: Cadena
 - # puestoEmp: Cadena
2. Método establecerNombreEmp(nom: Cadena)
 - a. nombreEmp = nom
 - b. Fin Método establecerNombreEmp
3. Método establecerDeptoEmp(dep: Cadena)
 - a. deptoEmp = dep
 - b. Fin Método establecerDeptoEmp
4. Método establecerPuestoEmp(pue: Cadena)
 - a. puestoEmp = pue
 - b. Fin Método establecerPuestoEmp
5. Método obtenerNombreEmp(): Cadena
 - a. return nombreEmp
 - b. Fin Método obtenerNombreEmp
6. Método obtenerDeptoEmp(): Cadena
 - a. return deptoEmp
 - b. Fin Método obtenerDeptoEmp
7. Método obtenerPuestoEmp(): Cadena
 - a. return puestoEmp
 - b. Fin Método obtenerPuestoEmp

Fin Clase Empleado1

Clase EmpleadoPorHoras1 hereda de Empleado1

1. Declarar datos
 - horasTrab: Entero
 - cuotaHora: Real
 - sueldoQnaHoras: Real
2. Método establecerHorasTrab(horasTr: Entero)
 - a. horasTrab = horasTr
 - b. Fin Método establecerHorasTrab
3. Método establecerCuotaHora(cuotaHr: Real)
 - a. cuotaHora = cuotaHr
 - b. Fin Método establecerCuotaHora
4. Método calcularSueldoQnaHoras()
 - a. sueldoQnaHoras = horasTrab * cuotaHora
 - b. Fin Método calcularSueldoQnaHoras
5. Método obtenerSueldoQnaHoras(): Real
 - a. return sueldoQnaHoras
 - b. Fin Método obtenerSueldoQnaHoras

Fin Clase EmpleadoPorHoras1

```
Clase EmpleadoAsalariado1 hereda de Empleado11
1. Declarar datos
    sueldoMensual: Real
    sueldoQnaAsal: Real

2. Método establecerSueldoMensual(sdo: Real)
    a. sueldoMensual = sdo
    b. Fin Método establecerSueldoMensual

3. Método calcularSueldoQnaAsal()
    a. sueldoQnaAsal = sueldoMensual / 2
    b. Fin Método calcularSueldoQnaAsal

4. Método obtenerSueldoQnaAsal(): Real
    a. return sueldoQnaAsal
    b. Fin Método obtenerSueldoQnaAsal
Fin Clase EmpleadoAsalariado1

Clase EjecutaEmpleado11
1. Método principal()
    a. Declarar variables
        nomEmp, depto, puesto: Cadena
        hrsTra, tipoEmp: Entero
        cuoHr, sdoMen: Real
        desea: Carácter
    b. do
        1. Imprimir Menu y solicitar tipo de empleado
            Tipos de empleado
            1. Empleado por horas
            2. Empleado asalariado
            Teclee tipo:
        2. Leer tipoEmp
        3. Solicitar nombre, departamento, puesto
        4. Leer nomEmp, depto, puesto
        5. if tipoEmp == 1 then
            a. Declarar, crear e iniciar objeto
                EmpleadoPorHoras1 objEmpleado = new
                    EmpleadoPorHoras1()
            b. Solicitar número de horas trabajadas,
                cuota por hora
            c. Leer hrsTra, cuoHr
            d. Establecer
                objEmpleado.establecerNombreEmp(nomEmp)
                objEmpleado.establecerDeptoEmp(depto)
                objEmpleado.establecerPuestoEmp(puesto)
                objEmpleado.establecerHorasTrab(hrsTra)
                objEmpleado.establecerCuotaHora(cuoHr)
            e. Calcular
                objEmpleado.calcularSueldoQnaHoras()
```

```

        f. Imprimir objEmpleado.obtenerNombreEmp()
           objEmpleado.obtenerDeptoEmp()
           objEmpleado.obtenerPuestoEmp()
           objEmpleado.obtenerSueldoQnaHoras()
    6. else
        a. Declarar, crear e iniciar objeto
           EmpleadoAsalariado1 objEmpleado = new
               EmpleadoAsalariado1()
        b. Solicitar sueldo mensual
        c. Leer sdoMen
        d. Establecer
           objEmpleado.establecerNombreEmp(nomEmp)
           objEmpleado.establecerDeptoEmp(depto)
           objEmpleado.establecerPuestoEmp(puesto)
           objEmpleado.establecerSueldoMensual(sdoMen)
        e. Calcular
           objEmpleado.calcularSueldoQnaAsal()
        f. Imprimir objEmpleado.obtenerNombreEmp()
           objEmpleado.obtenerDeptoEmp()
           objEmpleado.obtenerPuestoEmp()
           objEmpleado.obtenerSueldoQnaAsal()
    7. endif
    8. Preguntar "¿Desea procesar otro empleado(S/N)?"
    9. Leer desea
    c. while desea == 'S'
    d. Fin Método principal
Fin Clase EjecutaEmpleado11
Fin

```



En la zona de descarga de la Web del libro está disponible:

Programa en Java: Empleado11.java, EmpleadoPorHoras1.java,
EmpleadoAsalariado1.java y EjecutaEmpleado11.java

Explicación:

Este algoritmo tiene cuatro clases, y entre ellas la superclase `Empleado11` (la superclase no se usa para crear instancias u objetos; en consecuencia, sólo se utilizará para derivar otras clases a partir de ella).

En la superclase `Empleado11`:

- Se declaran los datos que representan la estructura de la clase:
`nombreEmp` para el nombre del empleado (# protegido para heredarlo).
`deptoEmp` para el departamento del empleado (# protegido para heredarlo).
`puestoEmp` para el puesto del empleado (# protegido para heredarlo).
- Método `establecerNombreEmp(nom: Cadena)`.
 Recibe en el parámetro `nom` el valor que luego coloca en el dato `nombreEmp`.
- Método `establecerDeptoEmp(dep: Cadena)`.
 Recibe en el parámetro `dep` el valor que luego coloca en el dato `deptoEmp`.

4. Método establecerPuestoEmp(pue: Cadena).
Recibe en el parámetro pue el valor que luego coloca en el dato puestoEmp.
 5. Método obtenerNombreEmp(): Cadena.
Retorna nombreEmp.
 6. Método obtenerDeptoEmp(): Cadena.
Retorna deptoEmp.
 7. Método obtenerPuestoEmp(): Cadena.
Retorna puestoEmp.
- Fin de la Clase Empleado11.

En la Clase EmpleadoPorHoras1 hereda de Empleado11 (las palabras “hereda de” quieren decir que la clase EmpleadoPorHoras1 se “deriva de” la superclase Empleado11 aplicando el mecanismo de herencia):

1. Se declaran los datos que representan la estructura de la clase:
horasTrab para las horas trabajadas del empleado.
cuotaHora para la cuota por hora.
sueldoQnaHoras para el sueldo quincenal del empleado.
 2. Método establecerHorasTrab(horasTr: Entero).
Recibe en el parámetro horasTr el valor que luego coloca en el dato horasTrab.
 3. Método establecerCuotaHora(cuotaHr: Real).
Recibe en el parámetro cuotaHr el valor que luego coloca en el dato cuotaHora.
 4. Método calcularSueldoQnaHoras().
Calcula el sueldo quincenal del empleado.
 5. Método obtenerSueldoQnaHoras(): Real.
Retorna sueldoQnaHoras.
- Fin de la Clase EmpleadoPorHoras1.

En la Clase EmpleadoAsalariado1 hereda de Empleado11 (las palabras “hereda de” quieren decir que la clase EmpleadoAsalariado1 se “deriva de” la superclase Empleado11 aplicando el mecanismo de herencia):

1. Se declaran los datos que representan la estructura de la clase:
sueldoQnaAsal para el sueldo quincenal del empleado.
sueldoMensual para el sueldo mensual del empleado.
 2. Método establecerSueldoMensual(sdo: Real).
Recibe en el parámetro sdo el valor que luego coloca en el dato sueldoMensual.
 3. Método calcularSueldoQnaAsal().
Calcula el sueldo quincenal del empleado.
 4. Método obtenerSueldoQnaAsal(): Real.
Retorna sueldoQnaAsal.
- Fin de la Clase EmpleadoAsalariado1.

En la Clase EjecutaEmpleado11, en el Método principal:

- a. Se declaran las variables:
 - nomEmp para leer el nombre del empleado.
 - depto para leer el departamento del empleado.
 - puesto para leer el puesto del empleado.
 - tipoEmp para leer el tipo de empleado.
 - hrsTra para leer las horas trabajadas.
 - cuoHr para leer la cuota por hora.
 - sdoMen para leer el sueldo mensual.
 - desea para controlar el ciclo repetitivo.
- b. Inicia ciclo do:
 1. Imprime el menú y solicita el tipo de empleado:
 - Tipos de empleado
 - 1. Empleado por horas
 - 2. Empleado asalariado
 - Teclee tipo:
 2. Lee en tipoEmp.
 3. Solicita nombre, departamento y puesto.
 4. Lee en nomEmp, depto, puesto.
 5. Si tipoEmp == 1 entonces:
 - a. Se declara el objeto objEmpleado, usando como base a la Clase EmpleadoPorHoras1. Dicho objeto se crea e inicializa mediante el constructor por defecto EmpleadoPorHoras1().
 - b. Solicita número de horas trabajadas y cuota por hora.
 - c. Lee en hrsTra, cuoHr.
 - d. Se llama al Método establecerNombreEmp (nomEmp) del objeto objEmpleado para colocar el valor de nomEmp en el dato nombreEmp. Se llama al Método establecerDeptoEmp (depto) del objeto objEmpleado para colocar el valor de depto en el dato deptoEmp. Se llama al Método establecerPuestoEmp (puesto) del objeto objEmpleado para colocar el valor de puesto en el dato puestoEmp. Se llama al Método establecerHorasTrab (hrsTra) del objeto objEmpleado para colocar el valor de hrsTra en el dato horasTrab. Se llama al Método establecerCuotaHora (cuoHr) del objeto objEmpleado para colocar el valor de cuoHr en el dato cuotaHora.
 - e. Se llama al Método calcularSueldoQnaHoras () del objeto objEmpleado para calcular el sueldo quincenal.
 - f. Se llama al Método obtenerNombreEmp () del objeto objEmpleado para acceder e imprimir el valor del dato nombreEmp. Se llama al Método obtenerDeptoEmp () del objeto objEmpleado para acceder e imprimir el valor del dato deptoEmp. Se llama al Método obtenerPuestoEmp () del objeto objEmpleado para acceder e imprimir el valor del dato puestoEmp. Se llama al Método obtenerSueldoQnaHoras () del objeto objEmpleado para acceder e imprimir el valor del dato sueldoQnaHoras.

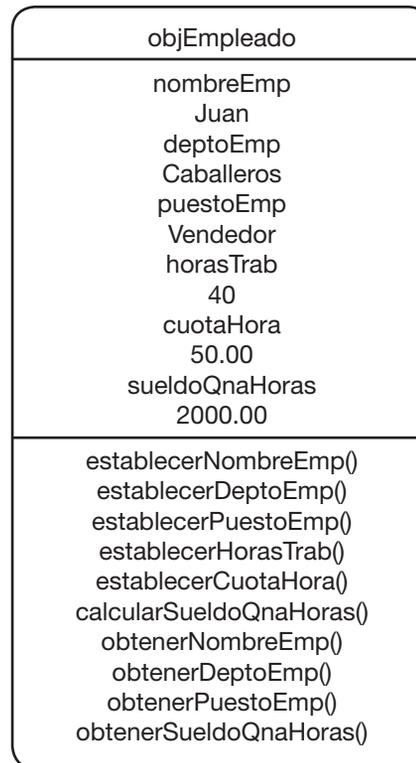
6. Si no:
 - a. Se declara el objeto `objEmpleado`, usando como base a la Clase `EmpleadoAsalariado1`. Dicho objeto se crea e inicializa mediante el constructor por defecto `EmpleadoAsalariado1()`.
 - b. Solicita sueldo mensual.
 - c. Lee en `sdoMen`.
 - d. Se llama al Método `establecerNombreEmp(nomEmp)` del objeto `objEmpleado` para colocar el valor de `nomEmp` en el dato `nombreEmp`. Se llama al Método `establecerDeptoEmp(depto)` del objeto `objEmpleado` para colocar el valor de `depto` en el dato `deptoEmp`. Se llama al Método `establecerPuestoEmp(puesto)` del objeto `objEmpleado` para colocar el valor de `puesto` en el dato `puestoEmp`. Se llama al Método `establecerSueldoMensual(sdoMen)` del objeto `objEmpleado` para colocar el valor de `sdoMen` en el dato `sueldoMensual`.
 - e. Se llama al Método `calcularSueldoQnaAsal()` del objeto `objEmpleado` para calcular el sueldo quincenal.
 - f. Se llama al Método `obtenerNombreEmp()` del objeto `objEmpleado` para acceder e imprimir el valor del dato `nombreEmp`. Se llama al Método `obtenerDeptoEmp()` del objeto `objEmpleado` para acceder e imprimir el valor del dato `deptoEmp`. Se llama al Método `obtenerPuestoEmp()` del objeto `objEmpleado` para acceder e imprimir el valor del dato `puestoEmp`. Se llama al Método `obtenerSueldoQnaAsal()` del objeto `objEmpleado` para acceder e imprimir el valor del dato `sueldoQnaAsal`.
7. Fin del if.
8. Pregunta “¿Desea procesar otro empleado(S/N)?”.
9. Lee la respuesta en `desea`.
- c. Fin del ciclo (`while desea == 'S'`). Si se cumple regresa al do; si no, se sale del ciclo.
- d. Fin Método principal.

Fin de la Clase EjecutaEmpleado11.

Fin del algoritmo.

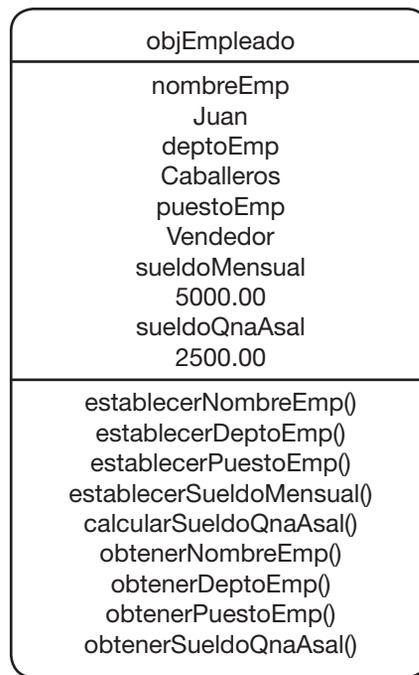
Resumiendo:

Cada vez que entra al ciclo procesa un empleado y pregunta si es empleado por horas o asalariado. Si el tipo de empleado es por horas, genera un objeto llamado `objEmpleado`. Este objeto se generará tomando como base la subclase `EmpleadoPorHoras1`, que a su vez hereda de `Empleado11`, quedando un objeto con la siguiente estructura:



Es decir, queda con los datos y métodos definidos tanto en la subclase `EmpleadoPorHoras1` como en la Clase `Empleado11`, que es la superclase de la cual hereda `EmpleadoPorHoras1`.

Por el contrario, si el tipo de empleado es asalariado, genera un objeto llamado `objEmpleado`. Este objeto se generará tomando como base la subclase `EmpleadoAsalariado1`, que a su vez hereda de `Empleado11`, quedando un objeto con la siguiente estructura:



Es decir, queda con los datos y métodos definidos tanto en la subclase `EmpleadoAsalariado1` como en la Clase `Empleado11`, que es la superclase de la cual hereda `EmpleadoAsalariado1`.

13.4 Ejercicios resueltos

Ejercicio 13.4.1

Elaborar un algoritmo que ofrezca un menú de opciones mediante el cual se pueda escoger calcular el área de las figuras geométricas: triángulo, cuadrado, rectángulo y círculo. Una vez seleccionada la opción, que permita solicitar y leer el nombre de la figura y los datos necesarios para calcular el área correspondiente e imprima el nombre de la figura y el área.

$$\text{Área de triángulo} = \frac{\text{Base} \times \text{Altura}}{2}$$

$$\text{Área de cuadrado} = \text{Lado}^2$$

$$\text{Área de círculo} = \pi r^2$$

$$\text{Área de rectángulo} = \text{Base} \times \text{Altura}$$

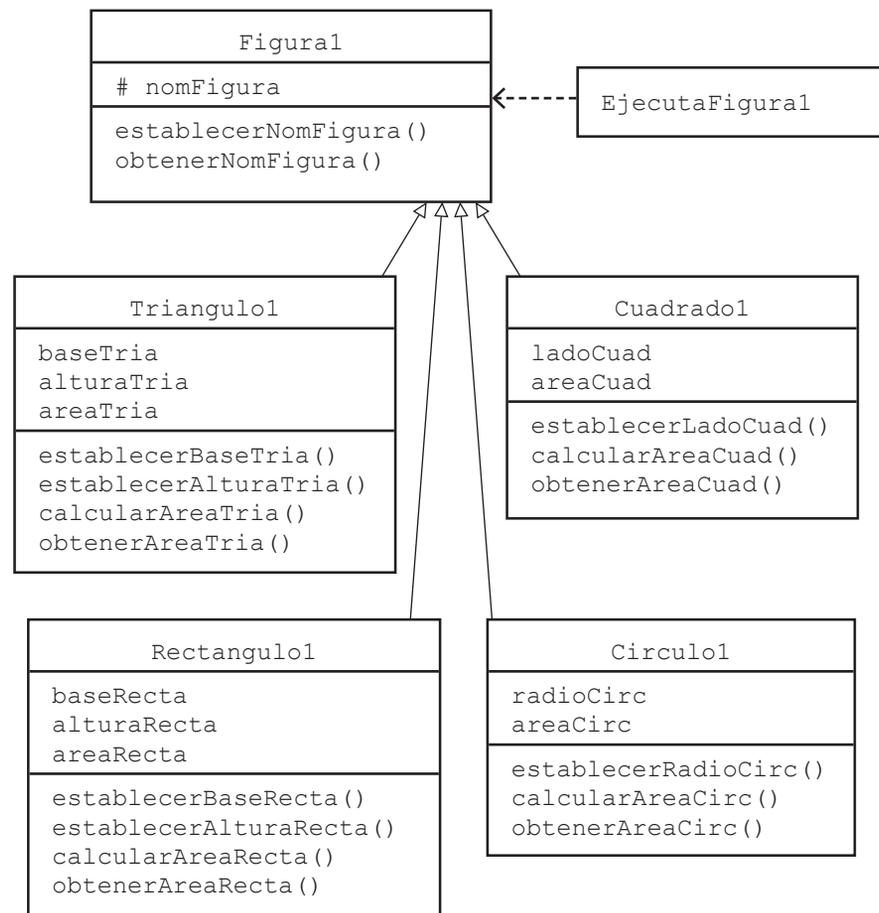
Debe ofrecer el siguiente menú de opciones, donde está solicitando la opción deseada:

AREAS FIGURAS GEOMETRICAS:
1. TRIANGULO 2. CUADRADO 3. RECTANGULO 4. CIRCULO 5. FIN
ESCOGER OPCIÓN

A continuación se tiene el diagrama de clases de la solución:

(Primero hágalo usted; después compare la solución)

Diagrama de clases



Explicación:

Se tiene la clase controlador `EjecutaFigural`, la cual utiliza al modelo, que está formado por cinco clases jerarquizadas:

1. La clase `Figural`, que es la superclase o clase principal, tiene el dato `nomFigura` y los métodos `establecerNomFigura()` y `obtenerNomFigura()` para establecer y obtener el dato respectivamente. `Figural` es la superclase que sólo se utiliza para derivar otras clases (subclases) a través del mecanismo de herencia; es por ello que a su dato `nomFigura` se le antepuso el símbolo `#`, el cual indica que el dato es protegido (`protected`), pues los datos deben ser protegidos para que se puedan heredar.
2. La clase `Triangulo1`, que es una subclase que se deriva de la superclase `Figural`, tiene los datos `baseTria`, `alturaTria` y `areaTria`, y los métodos `establecerBaseTria()`, `establecerAlturaTria()`, `calcularAreaTria()` y `obtenerAreaTria()` para establecer la base, establecer la altura, calcular el área y obtener el valor del área del triángulo respectivamente. Al derivarse de `Figural`, `Triangulo1` hereda el dato y los métodos de `Figural` a través del mecanismo de herencia.
3. La clase `Cuadrado1`, que es una subclase que se deriva de la superclase `Figural`, tiene los datos `ladoCuad` y `areaCuad`, y los métodos `establecerLadoCuad()`, `calcularAreaCuad()` y `obtenerAreaCuad()` para establecer el lado, calcular el área y obtener el valor del área del cuadrado respectivamente. Al derivarse de `Figural`, `Cuadrado1` hereda el dato y los métodos de `Figural` a través del mecanismo de herencia.
4. La clase `Rectangulo1`, que es una subclase que se deriva de la superclase `Figural`, tiene los datos `baseRecta`, `alturaRecta` y `areaRecta`, y los métodos `establecerBaseRecta()`, `establecerAlturaRecta()`, `calcularAreaRecta()` y `obtenerAreaRecta()` para establecer la base, establecer la altura, calcular el área y obtener el valor del área del rectángulo respectivamente. Al derivarse de `Figural`, `Rectangulo1` hereda el dato y los métodos de `Figural` a través del mecanismo de herencia.
5. La clase `Circulo1`, que es una subclase que se deriva de la superclase `Figural`, tiene los datos `radioCirc` y `areaCirc`, y los métodos `establecerRadioCirc()`, `calcularAreaCirc()` y `obtenerAreaCirc()` para establecer el radio, calcular el área y obtener el valor del área del círculo respectivamente. Al derivarse de `Figural`, `Circulo1` hereda el dato y los métodos de `Figural` a través del mecanismo de herencia.

A continuación se tiene el algoritmo de la solución en pseudocódigo:

(Primero hágalo usted; después compare la solución)

```

Algoritmo AREAS FIGURAS GEOMETRICAS
Clase Figural
  1. Declarar datos
      # nomFigura: Cadena

  2. Método establecerNomFigura(nom: Cadena)
      a. nomFigura = nom
      b. Fin Método establecerNomFigura

```

```
3. Método obtenerNomFigura(): Cadena
  a. return nomFigura
  b. Fin Método obtenerNomFigura
Fin Clase Figural

Clase Triangulo1 hereda de Figural
1. Declarar datos
  baseTria: Real
  alturaTria: Real
  areaTria: Real

2. Método establecerBaseTria(base: Real)
  a. baseTria = base
  b. Fin Método establecerBaseTria

3. Método establecerAlturaTria(altura: Real)
  a. alturaTria = altura
  b. Fin Método establecerAlturaTria

4. Método calcularAreaTria()
  a. areaTria = (baseTria * alturaTria) / 2
  b. Fin Método calcularAreaTria

5. Método obtenerAreaTria(): Real
  a. return areaTria
  b. Fin Método obtenerAreaTria
Fin Clase Triangulo1

Clase Cuadrado1 hereda de Figural
1. Declarar datos
  ladoCuad: Real
  areaCuad: Real

2. Método establecerLadoCuad(lado: Real)
  a. ladoCuad = lado
  b. Fin Método establecerLadoCuad

3. Método calcularAreaCuad()
  a. areaCuad = Potencia(ladoCuad, 2)
  b. Fin Método calcularAreaCuad

4. Método obtenerAreaCuad(): Real
  a. return areaCuad
  b. Fin Método obtenerAreaCuad
Fin Clase Cuadrado1
```

Clase Rectangulo1 hereda de Figural

1. Declarar datos
 - baseRecta: Real
 - alturaRecta: Real
 - areaRecta: Real
2. Método establecerBaseRecta(base: Real)
 - a. baseRecta = base
 - b. Fin Método establecerBaseRecta
3. Método establecerAlturaRecta(altura: Real)
 - a. alturaRecta = altura
 - b. Fin Método establecerAlturaRecta
4. Método calcularAreaRecta()
 - a. areaRecta = baseRecta * alturaRecta
 - b. Fin Método calcularAreaRecta
5. Método obtenerAreaRecta(): Real
 - a. return areaRecta
 - b. Fin Método obtenerAreaRecta

Fin Clase Rectangulo1

Clase Circulo1 hereda de Figural

1. Declarar datos
 - radioCirc: Real
 - areaCirc: Real
2. Método establecerRadioCirc(radio: Real)
 - a. radioCirc = radio
 - b. Fin Método establecerRadioCirc
3. Método calcularAreaCirc()
 - a. areaCirc = 3.1416 * Potencia(radioCirc, 2)
 - b. Fin Método calcularAreaCirc
4. Método obtenerAreaCirc(): Real
 - a. return areaCirc
 - b. Fin Método obtenerAreaCirc

Fin Clase Circulo1

Clase EjecutaFigural

1. Método principal()
 - a. Declarar variables
 - nombre: Cadena
 - opcion: Entero
 - bas, alt, rad, lad: Real

b. do

1. Imprimir MENU

AREAS FIGURAS GEOMETRICAS:
1. TRIANGULO 2. CUADRADO 3. RECTANGULO 4. CIRCULO 5. FIN
ESCOGER OPCIÓN

2. Leer opcion

3. Solicitar nombre de la figura

4. Leer nombre

5. switch opcion

1: a. Declarar, crear e iniciar objeto

```
Triangulo1 objFigura = new Triangulo1()
```

b. Solicitar Base, Altura

c. Leer bas, alt

d. Establecer

```
objFigura.establecerNomFigura(nombre)
```

```
objFigura.establecerBaseTria(bas)
```

```
objFigura.establecerAlturaTria(alt)
```

e. Calcular `objFigura.calcularAreaTria()`

f. Imprimir `objFigura.obtenerNomFigura()`

```
objFigura.obtenerAreaTria()
```

2: a. Declarar, crear e iniciar objeto

```
Cuadrado1 objFigura = new Cuadrado1()
```

b. Solicitar Lado

c. Leer lad

d. Establecer

```
objFigura.establecerNomFigura(nombre)
```

```
objFigura.establecerLadoCuad(lad)
```

e. Calcular `objFigura.calcularAreaCuad()`

f. Imprimir `objFigura.obtenerNomFigura()`

```
objFigura.obtenerAreaCuad()
```

3: a. Declarar, crear e iniciar objeto

```
Rectangulo1 objFigura = new Rectangulo1()
```

b. Solicitar Base, Altura

c. Leer bas, alt

d. Establecer

```
objFigura.establecerNomFigura(nombre)
```

```
objFigura.establecerBaseRecta(bas)
```

```
objFigura.establecerAlturaRecta(alt)
```

e. Calcular `objFigura.calcularAreaRecta()`

f. Imprimir `objFigura.obtenerNomFigura()`

```
objFigura.obtenerAreaRecta()
```

```

4: a. Declarar, crear e iniciar objeto
    Circulo1 objFigura = new Circulo1()
    b. Solicitar Radio
    c. Leer rad
    d. Establecer
      objFigura.establecerNomFigura(nombre)
      objFigura.establecerRadioCirc(rad)
    e. Calcular objFigura.calcularAreaCirc()
    f. Imprimir objFigura.obtenerNomFigura()
      objFigura.obtenerAreaCirc()

6. endswitch
c. while opcion != 5
d. Fin Método principal
Fin Clase EjecutaFigural
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Figura1.java, Triangulo1.java, Cuadrado1.java, Rectangulo1.java, Circulo1.java y EjecutaFigura1.java



Explicación:

Este algoritmo tiene la superclase `Figural` (la superclase no se usa para crear instancias u objetos; sólo se utilizará para derivar otras clases a partir de ella).

En la superclase `Figural`:

1. Se declaran los datos que representan la estructura de la clase:
`nomFigura` para el nombre de la figura.
2. Método `establecerNomFigura(nom: Cadena)`.
 Recibe en el parámetro `nom` el valor que luego coloca en el dato `nomFigura`.
3. Método `obtenerNomFigura(): Cadena`.
 Retorna `nomFigura`.

Fin de la Clase `Figural`.

En la Clase `Triangulo1` hereda de `Figural` (las palabras “hereda de” quieren decir que la clase `Triangulo1` se “deriva de” la superclase `Figural` aplicando el mecanismo de herencia):

1. Se declaran los datos que representan la estructura de la clase:
`baseTria` para la base del triángulo.
`alturaTria` para la altura del triángulo.
`areaTria` para el área del triángulo.
2. Método `establecerBaseTria(base: Real)`.
 Recibe en el parámetro `base` el valor que luego coloca en el dato `baseTria`.
3. Método `establecerAlturaTria(altura: Real)`.
 Recibe en el parámetro `altura` el valor que luego coloca en el dato `alturaTria`.
4. Método `calcularAreaTria()`.
 Calcula el área del triángulo.
5. Método `obtenerAreaTria(): Real`.
 Retorna `areaTria`.

Fin de la Clase `Triangulo1`.

En la Clase Cuadrado1 hereda de Figura1 (las palabras “hereda de” quieren decir que la clase Cuadrado1 se “deriva de” la superclase Figura1 aplicando el mecanismo de herencia):

1. Se declaran los datos que representan la estructura de la clase:
ladoCuad para el lado del cuadrado.
areaCuad para el área del cuadrado.
 2. Método establecerLadoCuad(lado: Real).
Recibe en el parámetro lado el valor que luego coloca en el dato ladoCuad.
 3. Método calcularAreaCuad().
Calcula el área del cuadrado.
 4. Método obtenerAreaCuad(): Real.
Retorna areaCuad.
- Fin de la Clase Cuadrado1.

En la Clase Rectangulo1 hereda de Figura1 (las palabras “hereda de” quieren decir que la clase Rectangulo1 se “deriva de” la superclase Figura1 aplicando el mecanismo de herencia):

1. Se declaran los datos que representan la estructura de la clase:
baseRecta para la base del rectángulo.
alturaRecta para la altura del rectángulo.
areaRecta para el área del rectángulo.
 2. Método establecerBaseRecta(base: Real).
Recibe en el parámetro base el valor que luego coloca en el dato baseRecta.
 3. Método establecerAlturaRecta(altura: Real).
Recibe en el parámetro altura el valor que luego coloca en el dato alturaRecta.
 4. Método calcularAreaRecta().
Calcula el área del rectángulo.
 5. Método obtenerAreaRecta(): Real.
Retorna areaRecta.
- Fin de la Clase Rectangulo1.

En la Clase Circulo1 hereda de Figura1 (las palabras “hereda de” quieren decir que la clase Circulo1 se “deriva de” la superclase Figura1 aplicando el mecanismo de herencia):

1. Se declaran los datos que representan la estructura de la clase:
radioCirc para el radio del círculo.
areaCirc para el área del círculo.
 2. Método establecerRadioCirc(radio: Real).
Recibe en el parámetro radio el valor que luego coloca en el dato radioCirc.
 3. Método calcularAreaCirc().
Calcula el área del círculo.
 4. Método obtenerAreaCirc(): Real.
Retorna areaCirc.
- Fin de la Clase Circulo1.

En la Clase EjecutaFigura1, en el Método principal:

- a. Se declaran las variables:
 - nombre para leer el nombre de la figura.
 - opcion para leer la opción que desee.
 - bas para leer la base.
 - alt para leer la altura.
 - rad para leer el radio.
 - lad para leer el lado .
- b. Inicia ciclo do:
 1. Imprime el menú de opciones, que incluye solicitar opción.
 2. Lee la opción seleccionada en opcion.
 3. Solicita el nombre de la figura.
 4. Lee en nombre.
 5. Inicia switch opcion:
 - 1: a. Se declara el objeto objFigura, usando como base a la Clase Triangulo1. Dicho objeto se crea e inicializa mediante el constructor por defecto Triangulo1().
 - b. Solicita base y altura.
 - c. Lee en bas y alt.
 - d. Se llama al Método establecerNomFigura(nombre) del objeto objFigura para colocar el valor de nombre en el dato nomFigura. Se llama al Método establecerBaseTria(bas) del objeto objFigura para colocar el valor de bas en el dato baseTria. Se llama al Método establecerAlturaTria(alt) del objeto objFigura para colocar el valor de alt en el dato alturaTria.
 - e. Se llama al Método calcularAreaTria() del objeto objFigura para calcular el área del triángulo.
 - f. Se llama al Método obtenerNomFigura() del objeto objFigura para acceder e imprimir el valor del dato nomFigura. Se llama al Método obtenerAreaTria() del objeto objFigura para acceder e imprimir el valor del dato areaTria.
 - 2: a. Se declara el objeto objFigura, usando como base a la Clase Cuadrado1. Dicho objeto se crea e inicializa mediante el constructor por defecto Cuadrado1().
 - b. Solicita lado.
 - c. Lee en lad.
 - d. Se llama al Método establecerNomFigura(nombre) del objeto objFigura para colocar el valor de nombre en el dato nomFigura. Se llama al Método establecerLadoCuad(lad) del objeto objFigura para colocar el valor de lad en el dato ladoCuad.
 - e. Se llama al Método calcularAreaCuad() del objeto objFigura para calcular el área del cuadrado.
 - f. Se llama al Método obtenerNomFigura() del objeto objFigura para acceder e imprimir el valor del dato nomFigura. Se llama al Método obtenerAreaCuad() del objeto objFigura para acceder e imprimir el valor del dato areaCuad.
- 3: a. Se declara el objeto objFigura, usando como base a la Clase Rectangulo1. Dicho objeto se crea e inicializa mediante el constructor por defecto Rectangulo1().

- b. Solicita base y altura.
 - c. Lee en bas y alt.
 - d. Se llama al Método establecerNomFigura (nombre) del objeto objFigura para colocar el valor de nombre en el dato nomFigura. Se llama al Método establecerBaseRecta (bas) del objeto objFigura para colocar el valor de bas en el dato baseRecta. Se llama al Método establecerAlturaRecta (alt) del objeto objFigura para colocar el valor de alt en el dato alturaRecta.
 - e. Se llama al Método calcularAreaRecta () del objeto objFigura para calcular el área del rectángulo.
 - f. Se llama al Método obtenerNomFigura () del objeto objFigura para acceder e imprimir el valor del dato nomFigura. Se llama al Método obtenerAreaRecta () del objeto objFigura para acceder e imprimir el valor del dato areaRecta.
- 4:
- a. Se declara el objeto objFigura, usando como base a la Clase Circulo1. Dicho objeto se crea e inicializa mediante el constructor por defecto Circulo1 ().
 - b. Solicita el radio.
 - c. Lee en rad.
 - d. Se llama al Método establecerNomFigura (nombre) del objeto objFigura para colocar el valor de nombre en el dato nomFigura. Se llama al Método establecerRadioCirc (rad) del objeto objFigura para colocar el valor de rad en el dato radioCirc.
 - e. Se llama al Método calcularAreaCirc () del objeto objFigura para calcular el área del círculo.
 - f. Se llama al Método obtenerNomFigura () del objeto objFigura para acceder e imprimir el valor del dato nomFigura. Se llama al Método obtenerAreaCirc () del objeto objFigura para acceder e imprimir el valor del dato areaCirc.
6. Fin del switch,
- c. Fin del ciclo (while opcion != 5). Si se cumple regresa al do; si no, se sale del ciclo.
 - d. Fin del Método principal.
- Fin de la Clase EjecutaFigura1.
Fin del algoritmo.



En la zona de descarga del capítulo 13 de la Web del libro se encuentra el ejercicio resuelto.

Ejercicio 13.4.2

En este ejercicio se debe elaborar un algoritmo que soluciona el mismo problema del ejercicio anterior, pero ahora se le agrega que también calcule e imprima el perímetro de las figuras.

13.5 Ejercicios propuestos

Nota: Recuerde que un algoritmo orientado a objetos se diseña en dos pasos: diseño del diagrama de clases y diseño del algoritmo enseudocódigo.

1. Elaborar un algoritmo que ofrezca un menú de opciones mediante el cual se pueda escoger calcular el volumen de las figuras geométricas: cubo, cilindro, cono y esfera. Una vez seleccionada la opción, que permita solicitar y leer el nombre de la figura y los datos necesarios para calcular el volumen correspondiente e imprima el nombre de la figura y el volumen.

Volumen de cubo = $Arista^3$

Volumen de cilindro = $\pi r^2 h$

Volumen de cono = $\frac{1}{3} \pi r^2 h$

Volumen de esfera = $\frac{4}{3} \pi r^3$

Debe ofrecer el siguiente menú de opciones, donde está solicitando la opción deseada:

AREAS FIGURAS GEOMETRICAS:
1. CUBO 2. CILINDRO 3. CONO 4. ESFERA 5. FIN
ESCOGER OPCIÓN

La idea es que se use una superclase Figura que contendrá el dato nombre y los métodos para establecerlo y obtenerlo. De esa superclase derivar cuatro subclases: Cubo, Cilindro, Cono y Esfera, en cada una de las cuales se heredarán el dato y los métodos de la superclase; además, cada subclase de éstas deberá tener sus propios datos y métodos para establecer los datos necesarios, calcular el volumen correspondiente y obtenerlo para imprimirlo. Asimismo, deberá haber una clase controlador que permita leer los datos y utilice el modelo para representar y solucionar el problema.

2. Elaborar un algoritmo que ofrezca un menú de opciones mediante el cual se pueda escoger calcular el área de las figuras geométricas: trapecio, rombo, y paralelogramo. Una vez seleccionada la opción, que permita solicitar y leer el nombre de la figura y los datos necesarios para calcular el área correspondiente e imprima el nombre de la figura y el área.

$$\text{Área de trapecio} = \frac{(\text{BaseMayor} + \text{BaseMenor}) \text{ Altura}}{2}$$

$$\text{Área de rombo} = \frac{\text{DiagonalMayor} \times \text{DiagonalMenor}}{2}$$

$$\text{Área de paralelogramo} = \text{Base} \times \text{Altura}$$

Debe ofrecer el siguiente menú de opciones, donde está solicitando la opción deseada:

AREAS FIGURAS GEOMETRICAS:
1. TRAPECIO 2. ROMBO 3. PARALELOGRAMO 5. FIN
ESCOGER OPCIÓN

La idea es que se use una superclase Figura que contendrá el dato nombre y los métodos para establecerlo y obtenerlo. De esa superclase derivar tres subclases: Trapecio, Rombo y Paralelogramo, en cada una de las cuales se heredarán el dato y los métodos de la superclase; además, cada subclase de éstas deberá tener sus propios datos y métodos para establecer los datos necesarios, calcular el área correspondiente y obtenerla para imprimirla. Asimismo, deberá haber una clase controlador que permita leer los datos y utilice el modelo para representar y solucionar el problema.

3. Elaborar un algoritmo similar al anterior, sólo que ahora, además del área, para cada una de las figuras deberá calcular e imprimir el perímetro.
- Perímetro de trapecio = BaseMayor + BaseMenor + Lado1 + Lado2
 Perímetro de rombo = 4 x Lado
 Perímetro de paralelogramo = (2 x Base) + (2 x Altura)
4. En una empresa automotriz se tienen tres tipos de empleados: administrativos, mecánicos y vendedores. En general, para todos los empleados se tienen los datos RFC (Registro Federal de Contribuyentes), el nombre, el departamento y el puesto. En particular, para el empleado administrativo se tiene el dato sueldo mensual; para el mecánico se tiene el precio del trabajo tantas veces como trabajos haya realizado; y para el vendedor se tiene el precio del auto por cada auto que vendió.

El sueldo quincenal se calcula:

Para el administrativo, sueldo mensual entre 2.

Para el mecánico, el 4% del valor total de los trabajos realizados.

Para el vendedor, el salario mínimo más 2% del valor de la venta realizada.

Elaborar un algoritmo que permita procesar los empleados de la empresa e imprimir el reporte:

REPORTE DE NÓMINA QUINCENAL				
RFC	NOMBRE	DEPTO.	PUESTO	SUELDO QUINCENA
XXXXX	XXXXXXXXXXXXXXXXXX	XXXXXXX	XXXXXXX	99,999.99
--				
XXXXX	XXXXXXXXXXXXXXXXXX	XXXXXXX	XXXXXXX	99,999.99
TOTAL	999 EMPLEADOS			99,999.99

La idea es que se use una superclase Empleado que contendrá los datos RFC (Registro Federal de Causantes), el nombre, el departamento y el puesto, y los métodos para establecer y obtener cada uno de los datos. De esa superclase derivar tres subclases: EmpAdmvo, EmpMecanico y EmpVendedor, en cada una de las cuales se heredarán los datos y los métodos de la superclase. Además, cada subclase de éstas deberá tener sus propios datos y métodos para establecer los datos necesarios, calcular el sueldo quincenal correspondiente y obtenerlo para imprimirlo. Asimismo, deberá haber una clase controlador que permita leer los datos y utilice el modelo para representar y solucionar el problema.

- Elaborar un algoritmo similar al anterior, pero para una fábrica en la cual se tienen dos tipos de empleados: directivos y técnicos. Usted investigue o establezca los datos disponibles y la forma de pagarle a cada uno de los tipos de empleados. Hágalo de manera que tenga una superclase y dos subclases que hereden de la superclase.
- Elaborar un algoritmo similar al anterior, pero para un gobierno de un municipio o de un estado en el cual se tienen dos tipos de empleados: de confianza y sindicalizados. Usted investigue o establezca los datos disponibles y la forma de pagarle a cada uno de los tipos de empleados. Hágalo de manera que tenga una superclase y dos subclases que hereden de la superclase.
- En un banco se tienen clientes que son inversionistas: unos tienen cuenta de ahorro, otros tienen inversión en pagaré y otros tienen cuenta maestra. Cada tipo de inversión paga intereses de manera diferente; usted investigue o establezca cómo lo hará cada tipo de inversión. Elaborar un algoritmo que tenga una superclase Inversionista y tres subclases que hereden de la superclase, una para cada tipo de inversión: CuentaAhorro, Pagare, CuentaMaestra. Por cada tipo de cuenta se tiene el número de cliente, el nombre del cliente, el número de cuenta, el capital invertido, la tasa de interés anual y el plazo de la inversión. Imprimir el siguiente reporte:

REPORTE DE INVERSIONES			
No. Cliente	Nombre	No. Cuenta	Interés ganado
999999999	XXXXXXXXXXXXXXXXXX	9999999	99,999.99
--			
999999999	XXXXXXXXXXXXXXXXXX	9999999	99,999.99
TOTAL	999 INVERSIONES		99,999.99

8. En un banco se tienen clientes deudores: unos tienen préstamo personal, otros tienen préstamo hipotecario y otros tienen préstamo de automóvil. Cada tipo de préstamo paga intereses de manera diferente; usted investigue o establezca cómo lo hará cada tipo de préstamo. Elaborar un algoritmo que tenga una superclase `ClienteDeudor` y tres subclases que hereden de la superclase, una para cada tipo de préstamo: `PrestamoPersonal`, `PrestamoHipotecario`, `PrestamoAuto`. Por cada tipo de préstamo se tiene el número de cliente, el nombre del cliente, el número de cuenta, el capital prestado, la tasa de interés anual y el plazo de la inversión. Imprimir el siguiente reporte:

REPORTE DE CLIENTES DEUDORES			
No. Cliente	Nombre	No. Cuenta	Interés por pagar
999999999	XXXXXXXXXXXXXXXXXX	9999999	99,999.99
--			
999999999	XXXXXXXXXXXXXXXXXX	9999999	99,999.99
TOTAL 999 CLIENTES			99,999.99

13.6 Resumen de conceptos que debe dominar

- Herencia
- Diseño del diagrama de clases con herencia
- Superclases y subclases
- Diseño de algoritmos OO usando herencia

13.7 Contenido de la página Web de apoyo

El material marcado con asterisco (*) sólo está disponible para docentes.

13.7.1 Resumen gráfico del capítulo

13.7.2 Autoevaluación

13.7.3 Programas en Java

13.7.4 Ejercicios resueltos

13.7.5 Power Point para el profesor (*)

14

Programación orientada a objetos usando polimorfismo

Contenido

- 14.1 Polimorfismo
- 14.2 Diseño del diagrama de clases con polimorfismo
 - 14.2.1 Clases abstractas
- 14.3 Diseño de algoritmos OO usando polimorfismo
- 14.4 Ejercicios resueltos
- 14.5 Ejercicios propuestos
- 14.6 Resumen de conceptos que debe dominar
- 14.7 Contenido de la página Web de apoyo
 - El material marcado con asterisco (*) sólo está disponible para docentes.*
 - 14.7.1 Resumen gráfico del capítulo
 - 14.7.2 Autoevaluación
 - 14.7.3 Programas en Java
 - 14.7.4 Ejercicios resueltos
 - 14.7.5 Power Point para el profesor (*)

Objetivos del capítulo

- Aprender el concepto de polimorfismo y su uso en el diseño de algoritmos orientados a objetos.
- Aplicar lo aprendido en la solución de ejercicios resueltos y propuestos.

Competencias

- Competencia general del capítulo
 - *Analizar problemas y diseñar algoritmos que los solucionen aplicando la arquitectura orientada a objetos y polimorfismo.*
- Competencias específicas del capítulo
 - Define el concepto de polimorfismo.
 - Describe el diseño del diagrama de clases con polimorfismo.
 - Define los conceptos de métodos abstractos y clases abstractas.
 - Diseña algoritmos orientados a objetos aplicando polimorfismo.

Introducción

Con el estudio del capítulo anterior usted ya domina la herencia.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos usando el polimorfismo, que es uno de los conceptos fundamentales de la programación orientada a objetos.

Se explica cómo diseñar algoritmos orientados a objetos usando el polimorfismo.

Se expone el polimorfismo, el cual consiste en que en un mismo algoritmo algo se puede hacer de muchas formas.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejercite estudiando los problemas planteados en los ejercicios resueltos y propuestos. Al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro, luego verifique sus resultados con los del libro, analice las diferencias y vea sus errores. Al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no está igual que el del libro, no necesariamente está mal. Usted debe ir aprendiendo a analizar las diferencias y a comprender que a veces, aunque haya diferencias, las dos soluciones están correctas.

En el siguiente capítulo se estudian los registros y archivos.

14.1 Polimorfismo

Los métodos que hemos utilizado hasta este momento son métodos estáticos. Se les denomina así por el hecho de que el compilador asigna y resuelve todas las referencias en tiempo de compilación, es decir, los métodos se definen y se implementan en el mismo lugar, y así actuarán siempre.

Como hemos visto, objetos y métodos estáticos pueden ser una poderosa herramienta para organizar la complejidad de un programa. Sin embargo, hay ciertos problemas que no se resuelven con este tipo de métodos. La solución es que las referencias se resuelvan en tiempo de ejecución; para hacerlo se requieren ciertos mecanismos. Los lenguajes orientados a objetos proporcionan esos mecanismos a través del soporte de métodos abstractos (virtuales o dinámicos).

Los métodos abstractos (virtuales o dinámicos) implementan una muy poderosa herramienta de generalización llamada polimorfismo. Polimorfismo significa muchas formas, y es justamente eso: una forma de dar a un método un nombre que es compartido hacia arriba y hacia abajo en la jerarquía del objeto, donde cada objeto de la jerarquía implementa el método en forma apropiada a él mismo.

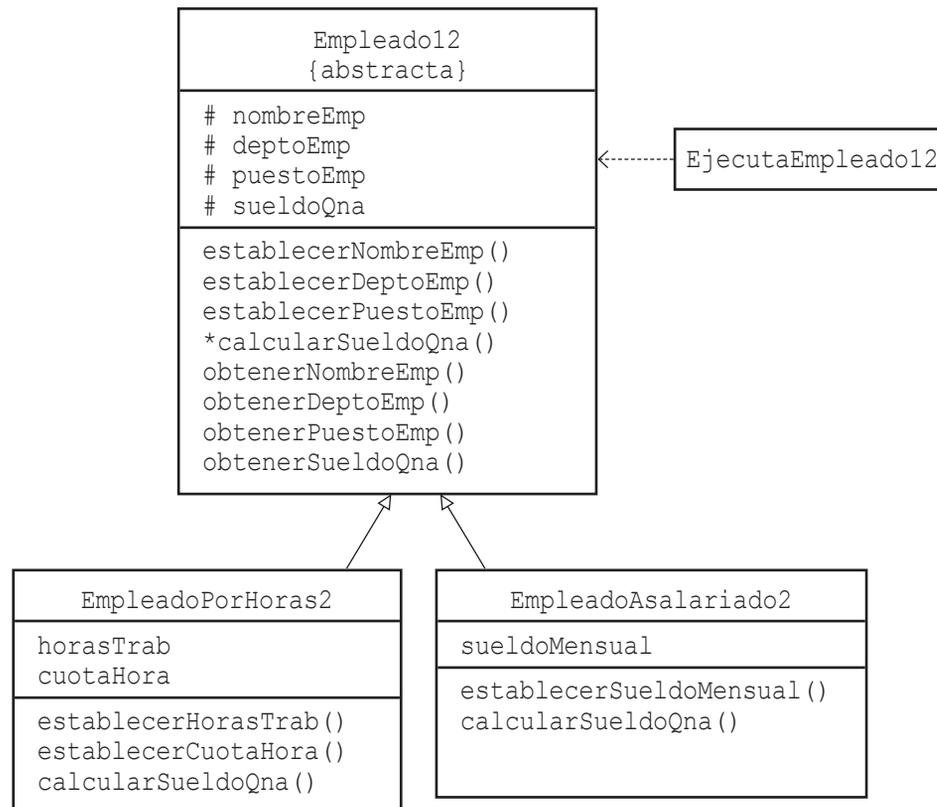
La diferencia entre una llamada a un método estático y una llamada a un método abstracto es la diferencia entre una decisión hecha ahora y una decisión que se hará después. Cuando se codifica un método estático, en esencia se está diciendo al compilador “tú sabes lo que yo quiero; llámalo”. Por otro lado, al hacer una llamada a un método abstracto (virtual o dinámico), es como si se le dijera al compilador “todavía no sabes lo que quiero; cuando llegue el momento, pide la forma que corresponda”.

Al utilizar objetos polimórficos y métodos abstractos (virtuales o dinámicos), los usuarios del objeto pueden añadir características que se ajusten a sus necesidades. Esta nueva noción de tomar el código de alguien, añadiéndole nuevas funciones y sin modificar el código fuente, es a lo que se le llama extensibilidad. La extensibilidad es una forma de crecer más, dada por la herencia; se hereda todo lo que el tipo ancestro tiene y se añaden las nuevas funciones que se necesiten. El acoplamiento tardío permite juntar el nuevo código con el viejo (el ya definido) en tiempo de ejecución. Así, la extensión del código existente es sin coseduras y no cuesta más, en términos de rendimiento, que un viaje rápido al lugar donde están los métodos abstractos.

14.2 Diseño del diagrama de clases con polimorfismo

En este punto y el siguiente se presenta el ejemplo de empleados que se desarrolló en el capítulo anterior, pero ahora dándole un enfoque apropiado para usar y explicar el concepto de polimorfismo. En este punto se diseña el diagrama de clases.

Diagrama de clases



Explicación:

Se tiene la clase controlador EjecutaEmpleado12, la cual utiliza al modelo, que está formado por tres clases jerarquizadas:

1. La clase Empleado12, que es la superclase o clase principal, se está definiendo como una clase abstracta y tiene los datos nombreEmp, deptoEmp, puestoEmp y sueldoQna, a los que se les antepuso el símbolo #, el cual indica que el dato es protegido (protected), pues los datos deben ser protegidos para que se puedan heredar; asimismo, esta clase tiene los métodos para establecer y obtener cada uno de los datos. En el caso de *calcularSueldoQna(), con el asterisco se define como un método abstracto, esto es, se define sólo el encabezado del método y cada objeto de las subclases derivadas lo implementará de acuerdo con las especificaciones que le corresponda: el objeto generado con EmpleadoPorHoras2 lo calculará utilizando horasTrab y cuotaHora, mientras que el objeto generado con EmpleadoAsalariado2 lo calculará utilizando sueldoMensual. Aplicando el polimorfismo se tiene más de una forma (dos) de calcular el sueldo.

2. La clase EmpleadoPorHoras2, que es una subclase que se deriva de la superclase abstracta Empleado12, tiene los datos horasTrab y cuotaHora, y los métodos establecerHorasTrab(), establecerCuotaHora() y calcularSueldoQna() para establecer las horas trabajadas, establecer la cuota por hora y calcular el sueldo quincenal respectivamente. Al derivarse de Empleado12, EmpleadoPorHoras2 hereda los datos y métodos de Empleado12 a través del mecanismo de herencia y, como está heredando el método abstracto *calcularSueldoQna(), aquí lo implementa utilizando la forma de calcular el sueldo quincenal correspondiente al empleado por horas.

3. La clase EmpleadoAsalariado2 que es una subclase que se deriva de la superclase abstracta Empleado12, tiene el dato sueldoMensual, así como el método establecerSueldoMensual(), que establece el sueldo mensual y el método calcularSueldoQna() que calcula el sueldo quincenal. Al derivarse de Empleado12, EmpleadoAsalariado2 hereda los datos y métodos de Empleado12 a través del mecanismo de herencia y, como está heredando el método abstracto *calcularSueldoQna(), aquí lo implementa utilizando la forma de calcular el sueldo quincenal correspondiente al empleado asalariado.

14.2.1 Clases abstractas

Las clases abstractas son clases que tienen al menos un método abstracto, los cuales se utilizan para implementar el polimorfismo. Como ya se ha mencionado, polimorfismo significa muchas formas y se aplica cuando alguna tarea o función, tiene más de una forma diferente de hacerse, veamos cómo se aplica esto en nuestro ejemplo:

La función calcular sueldo quincenal tiene dos formas distintas de hacerse: una para el empleado por horas, usando el número de horas trabajadas y cuota por hora, y la otra para el empleado asalariado, con el sueldo mensual.

En la clase Empleado12 se define el método abstracto *calcularSueldoQna(), el cual es un método abstracto porque aquí no se implementa: sólo se define su encabezado.

Luego, en cada una de las subclases EmpleadoPorHoras2 y EmpleadoAsalariado2, ambas derivadas de Empleado12, se implementa el método abstracto *calcularSueldoQna(). Esto es porque al empleado por horas el sueldo se le calcula de una forma y al empleado asalariado se le calcula de otra forma.

Así, el método *calcularSueldoQna() tiene dos formas distintas de implementarse. Es por ello que en la superclase abstracta Empleado12 se define pero no se implementa, y será en el momento de ejecución cuando se establezca la forma como se hará el cálculo del sueldo, dependiendo del tipo de empleado que se esté procesando, es decir, de una forma si es empleado por horas, o de la otra forma si es empleado asalariado.

Las clases abstractas son clases incompletas por el hecho de que los métodos abstractos no se implementan en dichas clases sino que se implementan en subclases derivadas de las clases abstractas, por lo cual no se pueden crear instancias u objetos de las clases abstractas.

14.3 Diseño de algoritmos OO usando polimorfismo

En este punto se toma como base el diagrama de clases diseñado en el apartado anterior y se procede a diseñar el algoritmo en pseudocódigo; es decir, se diseña la lógica de cada clase usando pseudocódigo, involucrando el concepto de polimorfismo. A continuación se tiene el algoritmo:

Algoritmo CALCULAR SUELDO DE VARIOS EMPLEADOS

Clase abstracta Empleado12

1. Declarar datos

```
# nombreEmp: Cadena
# deptoEmp: Cadena
# puestoEmp: Cadena
# sueldoQna: Real
```

2. Método establecerNombreEmp(nom: Cadena)

```
a. nombreEmp = nom
b. Fin Método establecerNombreEmp
```

3. Método establecerDeptoEmp(dep: Cadena)

```
a. deptoEmp = dep
b. Fin Método establecerDeptoEmp
```

4. Método establecerPuestoEmp(pue: Cadena)

```
a. puestoEmp = pue
b. Fin Método establecerPuestoEmp
```

5. Método abstracto calcularSueldoQna()

6. Método obtenerNombreEmp(): Cadena

```
a. return nombreEmp
b. Fin Método obtenerNombreEmp
```

7. Método obtenerDeptoEmp(): Cadena

```
a. return deptoEmp
b. Fin Método obtenerDeptoEmp
```

8. Método obtenerPuestoEmp(): Cadena

```
a. return puestoEmp
b. Fin Método obtenerPuestoEmp
```

```
9. Método obtenerSueldoQna(): Real
  a. return sueldoQna
  b. Fin Método obtenerSueldoQna
Fin Clase Empleado12

Clase EmpleadoPorHoras2 hereda de Empleado12
1. Declarar datos
  horasTrab: Entero
  cuotaHora: Real

2. Método establecerHorasTrab(horasTr: Entero)
  a. horasTrab = horasTr
  b. Fin Método establecerHorasTrab

3. Método establecerCuotaHora(cuotaHr: Real)
  a. cuotaHora = cuotaHr
  b. Fin Método establecerCuotaHora

4. Método calcularSueldoQna()
  a. sueldoQna = horasTrab * cuotaHora
  b. Fin Método calcularSueldoQna
Fin Clase EmpleadoPorHoras2

Clase EmpleadoAsalariado2 hereda de Empleado12
1. Declarar datos
  sueldoMensual: Real

2. Método establecerSueldoMensual(sdo: Real)
  a. sueldoMensual = sdo
  b. Fin Método establecerSueldoMensual

3. Método calcularSueldoQna()
  a. sueldoQna = sueldoMensual / 2
  b. Fin Método calcularSueldoQna
Fin Clase EmpleadoAsalariado2

Clase EjecutaEmpleado12
1. Método principal()
  a. Declarar variables
    nomEmp, depto, puesto: Cadena
    hrsTra, tipoEmp: Entero
    cuoHr, sdoMen: Real
    desea: Carácter
  b. do
    1. Imprimir Menu y solicitar tipo de empleado
      Tipos de empleado
      1. Empleado por horas
      2. Empleado asalariado
      Teclee tipo:
    2. Leer tipoEmp
    3. Solicitar nombre, departamento, puesto
    4. Leer nomEmp, depto, puesto
```

```

5. if tipoEmp == 1 then
    a. Declarar, crear e iniciar objeto
       EmpleadoPorHoras2 objEmpleado = new
           EmpleadoPorHoras2()
    b. Solicitar número de horas trabajadas,
       cuota por hora
    c. Leer hrsTra, cuoHr
    d. Establecer
       objEmpleado.establecerNombreEmp(nomEmp)
       objEmpleado.establecerDeptoEmp(depto)
       objEmpleado.establecerPuestoEmp(puesto)
       objEmpleado.establecerHorasTrab(hrsTra)
       objEmpleado.establecerCuotaHora(cuoHr)
    e. Calcular objEmpleado.calcularSueldoQna()
    f. Imprimir objEmpleado.obtenerNombreEmp()
       objEmpleado.obtenerDeptoEmp()
       objEmpleado.obtenerPuestoEmp()
       objEmpleado.obtenerSueldoQna()
6. else
    a. Declarar, crear e iniciar objeto
       EmpleadoAsalariado2 objEmpleado = new
           EmpleadoAsalariado2()
    b. Solicitar sueldo mensual
    c. Leer sdoMen
    d. Establecer
       objEmpleado.establecerNombreEmp(nomEmp)
       objEmpleado.establecerDeptoEmp(depto)
       objEmpleado.establecerPuestoEmp(puesto)
       objEmpleado.establecerSueldoMensual(sdoMen)
    e. Calcular objEmpleado.calcularSueldoQna()
    f. Imprimir objEmpleado.obtenerNombreEmp()
       objEmpleado.obtenerDeptoEmp()
       objEmpleado.obtenerPuestoEmp()
       objEmpleado.obtenerSueldoQna()
7. endif
8. Preguntar "¿Desea procesar otro empleado(S/N)?"
9. Leer desea
    c. while desea == 'S'
    d. Fin Método principal
Fin Clase EjecutaEmpleado12
Fin

```

En la zona de descarga de la Web del libro está disponible:
 Programa en Java: Empleado12.java, EmpleadoPorHoras2.java,
 EmpleadoAsalariado2.java y EjecutaEmpleado12.java



Explicación:

Este algoritmo tiene cuatro clases, y entre ellas la Clase abstracta `Empleado12` (que una clase es abstracta significa que tiene al menos un método abstracto, por tanto no se pueden crear instancias u objetos de dicha clase y sólo se utilizará para derivar otras clases a partir de ella).

En la Clase abstracta `Empleado12`:

1. Se declaran los datos que representan la estructura de la clase:
`nombreEmp` para el nombre del empleado (# protegido para heredarlo).
`deptoEmp` para el departamento del empleado (# protegido para heredarlo).
`puestoEmp` para el puesto del empleado (# protegido para heredarlo).
`sueldoQna` para el sueldo quincenal del empleado (# protegido para heredarlo).
2. Método `establecerNombreEmp(nom: Cadena)`.
Recibe en el parámetro `nom` el valor que luego coloca en el dato `nombreEmp`.
3. Método `establecerDeptoEmp(dep: Cadena)`.
Recibe en el parámetro `dep` el valor que luego coloca en el dato `deptoEmp`.
4. Método `establecerPuestoEmp(pue: Cadena)`.
Recibe en el parámetro `pue` el valor que luego coloca en el dato `puestoEmp`.
5. Método abstracto `calcularSueldoQna()`.
Se define el método abstracto `calcularSueldoQna()` para calcular el sueldo del empleado. Abstracto significa que aquí se define sólo el encabezado del método, esto es, sin acciones. Será en otro momento donde se implementará completamente el método, de acuerdo al tipo de empleado que corresponda. Aplicando el polimorfismo se tiene más de una forma (dos) de calcular el sueldo.
6. Método `obtenerNombreEmp(): Cadena`.
Retorna `nombreEmp`.
7. Método `obtenerDeptoEmp(): Cadena`.
Retorna `deptoEmp`.
8. Método `obtenerPuestoEmp(): Cadena`.
Retorna `puestoEmp`.
9. Método `obtenerSueldoQna(): Real`.
Retorna `sueldoQna`.

Fin de la Clase `Empleado12`

En la Clase `EmpleadoPorHoras2` hereda de `Empleado12` (las palabras “hereda de” quieren decir que la clase `EmpleadoPorHoras2` se “deriva de” la superclase `Empleado12` aplicando el mecanismo de herencia):

1. Se declaran los datos que representan la estructura de la clase:
`horasTrab` para las horas trabajadas del empleado.
`cuotaHora` para la cuota por hora.
2. Método `establecerHorasTrab(horasTr: Entero)`.
Recibe en el parámetro `horasTr` el valor que luego coloca en el dato `horasTrab`.
3. Método `establecerCuotaHora(cuotaHr: Real)`.
Recibe en el parámetro `cuotaHr` el valor que luego coloca en el dato `cuotaHora`.
4. Método `calcularSueldoQna()`.
Calcula el sueldo quincenal del empleado. Aquí se implementa el método abstracto en el que calcula el sueldo quincenal de la forma que le corresponde al empleado por horas, es decir, multiplicando número de horas por cuota por hora.

Fin de la Clase `EmpleadoPorHoras2`

En la Clase `EmpleadoAsalariado2` hereda de `Empleado12` (las palabras “hereda de” quieren decir que la clase `EmpleadoAsalariado2` se “deriva de” la superclase `Empleado12` aplicando el mecanismo de herencia):

1. Se declara el dato que representa la estructura de la clase:
 `sueldoMensual` para el sueldo mensual del empleado.
2. Método `establecerSueldoMensual(sdo: Real)`.
 Recibe en el parámetro `sdo` el valor que luego coloca en el dato `sueldoMensual`.
3. Método `calcularSueldoQna()`.
 Calcula el sueldo quincenal del empleado. Aquí se implementa el método abstracto en el que calcula el sueldo quincenal de la forma que le corresponde al empleado asalariado, es decir, dividiendo el sueldo mensual entre 2.

Fin de la Clase `EmpleadoAsalariado2`.

En la Clase `EjecutaEmpleado12`, en el Método principal:

- a. Se declaran las variables:
 `nomEmp` para leer el nombre del empleado.
 `depto` para leer el departamento del empleado.
 `puesto` para leer el puesto del empleado.
 `tipoEmp` para leer el tipo de empleado.
 `hrsTra` para leer las horas trabajadas.
 `cuoHr` para leer la cuota por hora.
 `sdoMen` para leer el sueldo mensual.
 `desea` para controlar el ciclo repetitivo.
- b. Inicia ciclo do:
 1. Imprime el menú y solicita el tipo de empleado:
 Tipos de empleado
 1. Empleado por horas
 2. Empleado asalariado Teclee tipo:
 2. Lee en `tipoEmp`.
 3. Solicita nombre, departamento y puesto.
 4. Lee en `nomEmp`, `depto`, `puesto`.
 5. Si `tipoEmp == 1` entonces:
 - a. Se declara el objeto `objEmpleado`, usando como base a la Clase `EmpleadoPorHoras2`. Dicho objeto se crea e inicializa mediante el constructor por defecto `EmpleadoPorHoras2()`.
 - b. Solicita número de horas trabajadas y cuota por hora.
 - c. Lee en `hrsTra`, `cuoHr`.
 - d. Se llama al Método `establecerNombreEmp(nomEmp)` del objeto `objEmpleado` para colocar el valor de `nomEmp` en el dato `nombreEmp`.
 Se llama al Método `establecerDeptoEmp(depto)` del objeto `objEmpleado` para colocar el valor de `depto` en el dato `deptoEmp`.
 Se llama al Método `establecerPuestoEmp(puesto)` del objeto `objEmpleado` para colocar el valor de `puesto` en el dato `puestoEmp`.
 Se llama al Método `establecerHorasTrab(hrsTra)` del objeto `objEmpleado` para colocar el valor de `hrsTra` en el dato `horasTrab`.

- Se llama al Método establecerCuotaHora(cuoHr) del objeto objEmpleado para colocar el valor de cuoHr en el dato cuotaHora.
- e. Se llama al Método calcularSueldoQna() del objeto objEmpleado para calcular el sueldo quincenal, como corresponde al empleado por horas.
 - f. Se llama al Método obtenerNombreEmp() del objeto objEmpleado para acceder e imprimir el valor del dato nombreEmp.
Se llama al Método obtenerDeptoEmp() del objeto objEmpleado para acceder e imprimir el valor del dato deptoEmp.
Se llama al Método obtenerPuestoEmp() del objeto objEmpleado para acceder e imprimir el valor del dato puestoEmp.
Se llama al Método obtenerSueldoQna() del objeto objEmpleado para acceder e imprimir el valor del dato sueldoQna.
6. Si no:
 - a. Se declara el objeto objEmpleado, usando como base a la Clase EmpleadoAsalariado2. Dicho objeto se crea e inicializa mediante el constructor por defecto EmpleadoAsalariado2().
 - b. Solicita sueldo mensual.
 - c. Lee en sdoMen.
 - d. Se llama al Método establecerNombreEmp(nomEmp) del objeto objEmpleado para colocar el valor de nomEmp en el dato nombreEmp.
Se llama al Método establecerDeptoEmp(depto) del objeto objEmpleado para colocar el valor de depto en el dato deptoEmp.
Se llama al Método establecerPuestoEmp(puesto) del objeto objEmpleado para colocar el valor de puesto en el dato puestoEmp.
Se llama al Método establecerSueldoMensual(sdoMen) del objeto objEmpleado para colocar el valor de sdoMen en el dato sueldoMensual.
 - e. Se llama al Método calcularSueldoQna() del objeto objEmpleado para calcular el sueldo quincenal como corresponde al empleado asalariado.
 - f. Se llama al Método obtenerNombreEmp() del objeto objEmpleado para acceder e imprimir el valor del dato nombreEmp.
Se llama al Método obtenerDeptoEmp() del objeto objEmpleado para acceder e imprimir el valor del dato deptoEmp.
Se llama al Método obtenerPuestoEmp() del objeto objEmpleado para acceder e imprimir el valor del dato puestoEmp.
Se llama al Método obtenerSueldoQna() del objeto objEmpleado para acceder e imprimir el valor del dato sueldoQna.
 7. Fin del if.
 8. Pregunta “¿Desea procesar otro empleado(S/N)?”.
 9. Lee en desea la respuesta.
- c. Fin del ciclo (while desea == 'S'). Si se cumple regresa al do; si no, se sale del ciclo.
 - d. Fin Método principal.
Fin de la Clase EjecutaEmpleado12.
Fin del algoritmo.

Nota: Este problema de empleados es el mismo que se planteó en el capítulo anterior. La diferencia en su solución es que aquí el dato `sueldoQna` está definido en la superclase abstracta `Empleado12`; asimismo, el método `*calcularSueldoQna()` como un método abstracto y el método `obtenerSueldoQna()`. Que el método es abstracto significa que aquí está definido, pero no implementado, y el cálculo del sueldo quincenal se realizará ya sea como está implementado en la subclase `EmpleadoPorHoras2` o de la forma como está implementado en la subclase `EmpleadoAsalariado2`, de acuerdo al tipo de empleado que se esté procesando, aplicando el polimorfismo.

14.4 Ejercicios resueltos

Ejercicio 14.4.1

Elaborar un algoritmo que ofrezca un menú de opciones mediante el cual se pueda escoger calcular el área de las figuras geométricas: triángulo, cuadrado, rectángulo y círculo. Una vez seleccionada la opción, que permita solicitar y leer el nombre de la figura y los datos necesarios para calcular el área correspondiente e imprimir el nombre de la figura y el área.

$$\text{Área de triángulo} = \frac{\text{Base} \times \text{Altura}}{2}$$

$$\text{Área de cuadrado} = \text{Lado}^2$$

$$\text{Área de círculo} = \pi r^2$$

$$\text{Área de rectángulo} = \text{Base} \times \text{Altura}$$

Debe ofrecer el siguiente menú de opciones, donde está solicitando la opción deseada:

AREAS FIGURAS GEOMETRICAS:
1. TRIANGULO 2. CUADRADO 3. RECTANGULO 4. CIRCULO 5. FIN
ESCOGER OPCIÓN

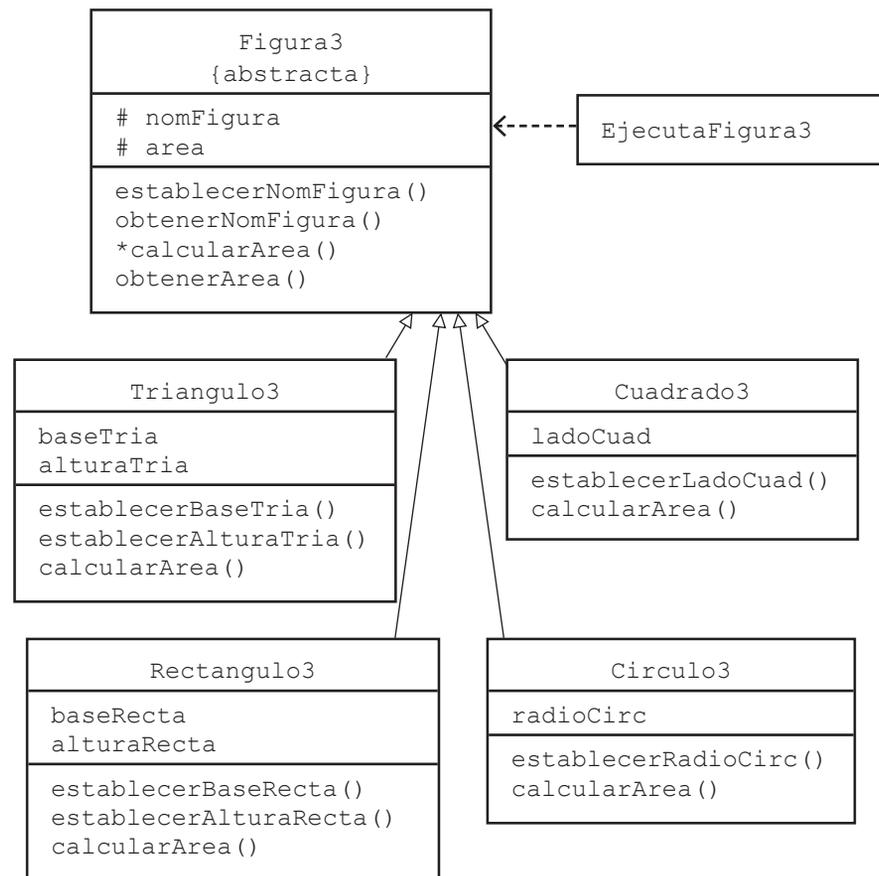


Este problema de empleados es el mismo que se planteó en el capítulo anterior. La diferencia en su solución es que aquí el dato `sueldoQna` está definido en la superclase abstracta `Empleado12`; asimismo, el método `*calcularSueldoQna()` como un método abstracto y el método `obtenerSueldoQna()`. Que el método es abstracto significa que aquí está definido, pero no implementado, y el cálculo del sueldo quincenal se realizará ya sea como está implementado en la subclase `EmpleadoPorHoras2` o de la forma como está implementado en la subclase `EmpleadoAsalariado2`, de acuerdo al tipo de empleado que se esté procesando, aplicando el polimorfismo.

A continuación se tiene el algoritmo de la solución:

(Primero hágalo usted; después compare la solución)

Diagrama de clases



Explicación:

Se tiene la clase controlador `EjecutaFigura3`, la cual utiliza al modelo, que está formado por cinco clases jerarquizadas:

1. La clase `Figura3`, que es la superclase o clase principal, se está definiendo como una clase abstracta y tiene el dato `nomFigura`, al que se le antepuso el símbolo `#`, el cual indica que el dato es protegido (protected), pues los datos deben ser protegidos para que se puedan heredar; asimismo, esta clase tiene los métodos `establecerNomFigura()` y `obtenerNomFigura()` para establecer y obtener el dato respectivamente. También tiene el dato `area` y los métodos `*calcularArea()` y `obtenerArea()` para calcular el área y obtener el valor del dato respectivamente. En el caso de `*calcularArea()`, con el asterisco se define como un método abstracto, esto es, se define sólo el encabezado del método y cada objeto de las clases derivadas lo implementará de acuerdo con las especificaciones que le corresponda:

el objeto generado con `Triangulo3` lo calculará de una forma; el objeto generado con `Cuadrado3` lo calculará de otra forma; el objeto generado con `Rectangulo3` lo calculará de otra forma y el objeto generado con `Circulo3` lo calculará de otra forma. Aplicando el polimorfismo se tiene más de una forma (cuatro) de calcular el área.

2. La clase `Triangulo3`, que es una subclase que se deriva de la superclase abstracta `Figura3`, tiene los datos `baseTria` y `alturaTria`, y los métodos `establecerBaseTria()`, `establecerAlturaTria()` y `calcularArea()` para establecer la base, establecer la altura y calcular el área respectivamente. Al derivarse de `Figura3`, `Triangulo3` hereda los datos y métodos de `Figura3` a través del mecanismo de herencia y, como está heredando el método abstracto `*calcularArea()`, aquí lo implementa utilizando la forma de calcular el área correspondiente al triángulo.

3. La clase `Cuadrado3`, que es una subclase que se deriva de la superclase `Figura3`, tiene el dato `ladoCuad` y los métodos `establecerLadoCuad()` y `calcularArea()` para establecer el lado y calcular el área respectivamente. Al derivarse de `Figura3`, `Cuadrado3` hereda los datos y métodos de `Figura3` a través del mecanismo de herencia y, como está heredando el método abstracto `*calcularArea()`, aquí lo implementa utilizando la forma de calcular el área correspondiente al cuadrado.

4. La clase `Rectangulo3`, que es una subclase que se deriva de la superclase `Figura3`, tiene los datos `baseRecta` y `alturaRecta`, y los métodos `establecerBaseRecta()`, `establecerAlturaRecta()` y `calcularArea()` para establecer la base, establecer la altura y calcular el área respectivamente. Al derivarse de `Figura3`, `Rectangulo3` hereda los datos y métodos de `Figura3` a través del mecanismo de herencia y, como está heredando el método abstracto `*calcularArea()`, aquí lo implementa utilizando la forma de calcular el área correspondiente al rectángulo.

5. La clase `Circulo3`, que es una subclase que se deriva de la superclase `Figura3`, tiene el dato `radioCirc` y los métodos `establecerRadioCirc()` y `calcularArea()` para establecer el radio y calcular el área respectivamente. Al derivarse de `Figura3`, `Circulo3` hereda los datos y métodos de `Figura3` a través del mecanismo de herencia y, como está heredando el método abstracto `*calcularArea()`, aquí lo implementa utilizando la forma de calcular el área correspondiente al círculo.

A continuación se tiene el algoritmo de la solución en pseudocódigo:

(Primero hágalo usted; después compare la solución)

```

Algoritmo AREAS FIGURAS GEOMETRICAS
Clase abstracta Figura3
1. Declarar datos
    # nomFigura: Cadena
    # area: Real

2. Método establecerNomFigura(nom: Cadena)
    a. nomFigura = nom
    b. Fin Método establecerNomFigura

```

```
3. Método obtenerNomFigura(): Cadena
  a. return nomFigura
  b. Fin Método obtenerNomFigura

4. Método abstracto calcularArea()

5. Método obtenerArea(): Real
  a. return area
  b. Fin Método obtenerArea
Fin Clase Figura3

Clase Triangulo3 hereda de Figura3
1. Declarar datos
  baseTria: Real
  alturaTria: Real

2. Método establecerBaseTria(base: Real)
  a. baseTria = base
  b. Fin Método establecerBaseTria

3. Método establecerAlturaTria(altura: Real)
  a. alturaTria = altura
  b. Fin Método establecerAlturaTria

4. Método calcularArea()
  a. area = (baseTria * alturaTria) / 2
  b. Fin Método calcularArea
Fin Clase Triangulo3

Clase Cuadrado3 hereda de Figura3
1. Declarar datos
  ladoCuad: Real

2. Método establecerLadoCuad(lado: Real)
  a. ladoCuad = lado
  b. Fin Método establecerLadoCuad

3. Método calcularArea()
  a. area = ladoCuad * ladoCuad
  b. Fin Método calcularArea
Fin Clase Cuadrado3

Clase Rectangulo3 hereda de Figura3
1. Declarar datos
  baseRecta: Real
  alturaRecta: Real

2. Método establecerBaseRecta(base: Real)
  a. baseRecta = base
  b. Fin Método establecerBaseRecta
```

3. Método establecerAlturaRecta(altura: Real)
 - a. alturaRecta = altura
 - b. Fin Método establecerAlturaRecta

4. Método calcularArea()
 - a. area = baseRecta * alturaRecta
 - b. Fin Método calcularArea

Fin Clase Rectangulo3

Clase Circulo3 hereda de Figura3

1. Declarar datos
 - radioCirc: Real
2. Método establecerRadioCirc(radio: Real)
 - a. radioCirc = radio
 - b. Fin Método establecerRadioCirc
3. Método calcularArea()
 - a. area = 3.1416 * (radioCirc * radioCirc)
 - b. Fin Método calcularArea

Fin Clase Circulo3

Clase EjecutaFigura3

1. Método principal()
 - a. Declarar variables
 - nombre: Cadena
 - opcion: Entero
 - bas, alt, rad, lad: Real
 - b. do
 1. Imprimir MENU

AREAS FIGURAS GEOMETRICAS:

- | |
|--|
| <ol style="list-style-type: none"> 1. TRIANGULO 2. CUADRADO 3. RECTANGULO 4. CIRCULO 5. FIN |
|--|

ESCOGER OPCIÓN

2. Leer opcion
3. Solicitar nombre de la figura
4. Leer nombre
5. switch opcion
 - 1: a. Declarar, crear e iniciar objeto
 - Triangulo3 objFigura = new Triangulo3()
 - b. Solicitar Base, Altura
 - c. Leer bas, alt

```

        d. Establecer
           objFigura.establecerNomFigura(nombre)
           objFigura.establecerBaseTria(bas)
           objFigura.establecerAlturaTria(alt)
        e. Calcular objFigura.calcularArea()
        f. Imprimir objFigura.obtenerNomFigura()
           objFigura.obtenerArea()

2: a. Declarar, crear e iniciar objeto
    Cuadrado3 objFigura = new Cuadrado3()
    b. Solicitar Lado
    c. Leer lad
    d. Establecer
       objFigura.establecerNomFigura(nombre)
       objFigura.establecerLadoCuad(lad)
    e. Calcular objFigura.calcularArea()
    f. Imprimir objFigura.obtenerNomFigura()
       objFigura.obtenerArea()

3: a. Declarar, crear e iniciar objeto
    Rectangulo3 objFigura= new Rectangulo3()
    b. Solicitar Base, Altura
    c. Leer bas, alt
    d. Establecer
       objFigura.establecerNomFigura(nombre)
       objFigura.establecerBaseRecta(bas)
       objFigura.establecerAlturaRecta(alt)
    e. Calcular objFigura.calcularArea()
    f. Imprimir objFigura.obtenerNomFigura()
       objFigura.obtenerArea()

4: a. Declarar, crear e iniciar objeto
    Circulo3 objFigura = new Circulo3()
    b. Solicitar Radio
    c. Leer rad
    d. Establecer
       objFigura.establecerNomFigura(nombre)
       objFigura.establecerRadioCirc(rad)
    e. Calcular objFigura.calcularArea()
    f. Imprimir objFigura.obtenerNomFigura()
       objFigura.obtenerArea()

        6. endswitch
    c. while opcion != 5
    d. Fin Método principal
    Fin Clase EjecutaFigura3
Fin

```



En la zona de descarga de la Web del libro está disponible:

Programa en Java: Figura3.java, Triangulo3.java, Cuadrado3.java, Rectangulo3.java, Circulo3.java y EjecutaFigura3.java

Explicación:

Este algoritmo tiene seis clases, entre ellas la Clase abstracta `Figura3` (que una clase es abstracta significa que tiene al menos un método abstracto, por tanto no se pueden crear instancias u objetos de dicha clase y sólo se utilizará para derivar otras clases a partir de ella).

En la Clase abstracta `Figura3`:

1. Se declaran los datos que representan la estructura de la clase:
`nomFigura` para el nombre de la figura.
`area` para el área de la figura.
2. Método `establecerNomFigura(nom: Cadena)`.
Recibe en el parámetro `nom` el valor que luego coloca en el dato `nomFigura`.
3. Método `obtenerNomFigura(): Cadena`.
Retorna `nomFigura`.
4. Método abstracto `calcularArea()`.
Se define el método abstracto `calcularArea()` para calcular el área de la figura. Abstracto significa que aquí se define sólo el encabezado del método, es decir, sin acciones. Será en otro momento donde se implementará completamente el método, de acuerdo a la figura que corresponda. Aplicando el polimorfismo se tiene más de una forma (cuatro) de calcular el área.
5. Método `obtenerArea(): Real`.
Retorna `area`.

Fin de la Clase `Figura3`.

En la Clase `Triangulo3` hereda de `Figura3` (las palabras “hereda de” quieren decir que la clase `Triangulo3` se “deriva de” la superclase `Figura3` aplicando el mecanismo de herencia):

1. Se declaran los datos que representan la estructura de la clase:
`baseTria` para la base del triángulo.
`alturaTria` para la altura del triángulo.
2. Método `establecerBaseTria(base: Real)`.
Recibe en el parámetro `base` el valor que luego coloca en el dato `baseTria`.
3. Método `establecerAlturaTria(altura: Real)`.
Recibe en el parámetro `altura` el valor que luego coloca en el dato `alturaTria`.
4. Método `calcularArea()`.
Calcula el área. Aquí se implementa el método abstracto en el que calcula el área de la forma que le corresponde al triángulo, es decir, multiplicando `base` por `altura` y dividiendo entre 2.

Fin de la Clase `Triangulo3`.

En la Clase `Cuadrado3` hereda de `Figura3` (las palabras “hereda de” quieren decir que la clase `Cuadrado3` se “deriva de” la superclase `Figura3` aplicando el mecanismo de herencia):

1. Se declaran los datos que representan la estructura de la clase:

- ladoCuad para el lado del cuadrado.
2. Método establecerLadoCuad(lado: Real).
Recibe en el parámetro lado el valor que luego coloca en el dato ladoCuad.
 3. Método calcularArea().
Calcula el área. Aquí se implementa el método abstracto en el que calcula el área de la forma que le corresponde al cuadrado, es decir, elevando el lado al cuadrado.
- Fin de la Clase Cuadrado3

En la Clase Rectangulo3 hereda de Figura3 (las palabras “hereda de” quieren decir que la clase Rectangulo3 se “deriva de” la superclase Figura3 aplicando el mecanismo de herencia):

1. Se declaran los datos que representan la estructura de la clase:
baseRecta para la base del rectángulo.
alturaRecta para la altura del rectángulo.
2. Método establecerBaseRecta(base: Real).
Recibe en el parámetro base el valor que luego coloca en el dato baseRecta.
3. Método establecerAlturaRecta(altura: Real).
Recibe en el parámetro altura el valor que luego coloca en el dato alturaRecta.
4. Método calcularArea().
Calcula el área. Aquí se implementa el método abstracto en el que calcula el área de la forma que le corresponde al rectángulo, es decir, multiplicando base por altura.

Fin de la Clase Rectangulo3

En la Clase Circulo3 hereda de Figura3 (las palabras “hereda de” quieren decir que la clase Circulo3 se “deriva de” la superclase Figura3 aplicando el mecanismo de herencia):

1. Se declaran los datos que representan la estructura de la clase:
radioCirc para el radio del círculo.
2. Método establecerRadioCirc(radio: Real).
Recibe en el parámetro radio el valor que luego coloca en el dato radioCirc.
3. Método calcularArea().
Calcula el área. Aquí se implementa el método abstracto en el que calcula el área de la forma que le corresponde al círculo, es decir, multiplicando pi por radio al cuadrado.

Fin de la Clase Circulo3.

En la Clase EjecutaFigura3, en el Método principal:

- a. Se declaran las variables:
nombre para leer el nombre de la figura.
opcion para leer la opción que desee.
bas para leer la base.
alt para leer la altura.

rad para leer el radio.

lad para leer el lado.

b. Inicia ciclo do:

1. Imprime el menú de opciones, el cual incluye solicitar la opción.
2. Lee en opcion.
3. Solicita el nombre de la figura.
4. Lee en nombre.
5. Inicia switch opcion:
 - 1: a. Se declara el objeto objFigura, usando como base a la Clase Triangulo3. Dicho objeto se crea e inicializa mediante el constructor por defecto Triangulo3().
 - b. Solicita Base y Altura.
 - c. Lee en bas y alt.
 - d. Se llama al Método establecerNomFigura(nombre) del objeto objFigura para colocar el valor de nombre en el dato nomFigura. Se llama al Método establecerBaseTria(bas) del objeto objFigura para colocar el valor de bas en el dato baseTria. Se llama al Método establecerAlturaTria(alt) del objeto objFigura para colocar el valor de alt en el dato alturaTria.
 - e. Se llama al Método calcularArea() del objeto objFigura para calcular el área del triángulo.
 - f. Se llama al Método obtenerNomFigura() del objeto objFigura para acceder e imprimir el valor del dato nomFigura. Se llama al Método obtenerArea() del objeto objFigura para acceder e imprimir el valor del dato area.
 - 2: a. Se declara el objeto objFigura, usando como base a la Clase Cuadrado3. Dicho objeto se crea e inicializa mediante el constructor por defecto Cuadrado3().
 - b. Solicita Lado.
 - c. Lee en lad.
 - d. Se llama al Método establecerNomFigura(nombre) del objeto objFigura para colocar el valor de nombre en el dato nomFigura. Se llama al Método establecerLadoCuad(lad) del objeto objFigura para colocar el valor de lad en el dato ladoCuad.
 - e. Se llama al Método calcularArea() del objeto objFigura para calcular el área del cuadrado.
 - f. Se llama al Método obtenerNomFigura() del objeto objFigura para acceder e imprimir el valor del dato nomFigura. Se llama al Método obtenerArea() del objeto objFigura para acceder e imprimir el valor del dato area.
 - 3: a. Se declara el objeto objFigura, usando como base a la Clase Rectangulo3. Dicho objeto se crea e inicializa mediante el constructor por defecto Rectangulo3().
 - b. Solicita Base y Altura.
 - c. Lee en bas y alt.
 - d. Se llama al Método establecerNomFigura(nombre) del objeto objFigura para colocar el valor de nombre en el dato nomFigura. Se llama al Método establecerBaseRecta(bas) del objeto objFigura para colocar el valor de bas en el dato baseRecta.



Este problema de áreas de figuras geométricas es el mismo que se planteó en el capítulo anterior. La diferencia en su solución es que aquí el dato `area` está definido en la superclase abstracta `Figura3`; asimismo, el método `*calcularArea()` como un método abstracto y el método `obtenerArea()`. Que el método es abstracto significa que aquí está definido, pero no implementado, y el cálculo del área se realizará ya sea como está implementado en la subclase `Triangulo3`, `Cuadrado3`, `Rectangulo3` o `Circulo3`, de acuerdo al tipo de figura que se esté procesando, aplicando el polimorfismo.

- e. Se llama al Método `calcularArea()` del objeto `objFigura` para calcular el área del rectángulo.
 - f. Se llama al Método `obtenerNomFigura()` del objeto `objFigura` para acceder e imprimir el valor del dato `nomFigura`.
Se llama al Método `obtenerArea()` del objeto `objFigura` para acceder e imprimir el valor del dato `area`.
- 4:
- a. Se declara el objeto `objFigura`, usando como base a la Clase `Circulo3`. Dicho objeto se crea e inicializa mediante el constructor por defecto `Circulo3()`.
 - b. Solicita el radio.
 - c. Lee en rad.
 - d. Se llama al Método `establecerNomFigura(nombre)` del objeto `objFigura` para colocar el valor de `nombre` en el dato `nomFigura`.
Se llama al Método `establecerRadioCirc(rad)` del objeto `objFigura` para colocar el valor de `rad` en el dato `radioCirc`.
 - e. Se llama al Método `calcularArea()` del objeto `objFigura` para calcular el área del círculo.
 - f. Se llama al Método `obtenerNomFigura()` del objeto `objFigura` para acceder e imprimir el valor del dato `nomFigura`.
Se llama al Método `obtenerArea()` del objeto `objFigura` para acceder e imprimir el valor del dato `area`.
6. Fin del switch.
- c. Fin del ciclo (`while opcion != 5`). Si se cumple regresa al `do`; si no, se sale del ciclo.
- d. Fin del Método principal.
- Fin de la Clase `EjecutaFigura3`.
- Fin del algoritmo.

Nota: Este problema de áreas de figuras geométricas es el mismo que se planteó en el capítulo anterior. La diferencia en su solución es que aquí el dato `area` está definido en la superclase abstracta `Figura3`; asimismo, el método `*calcularArea()` como un método abstracto y el método `obtenerArea()`. Que el método es abstracto significa que aquí está definido, pero no implementado, y el cálculo del área se realizará ya sea como está implementado en la subclase `Triangulo3`, `Cuadrado3`, `Rectangulo3` o `Circulo3`, de acuerdo al tipo de figura que se esté procesando, aplicando el polimorfismo.



En la zona de descarga del capítulo 14 de la Web del libro se encuentra el ejercicio resuelto.

Ejercicio 14.4.2

En este ejercicio se elabora un algoritmo que soluciona el mismo problema anterior, pero ahora se le agrega que también calcule e imprima el perímetro de las figuras.

14.5 Ejercicios propuestos

Nota: Recuerde que un algoritmo orientado a objetos se diseña en dos pasos: diseño del diagrama de clases y diseño del algoritmo en pseudocódigo.

1. Elaborar un algoritmo que ofrezca un menú de opciones mediante el cual se pueda escoger calcular el volumen de las figuras geométricas cubo, cilindro, cono y esfera. Una vez seleccionada la opción, que permita solicitar y leer el nombre de la figura y los datos necesarios para calcular el volumen correspondiente e imprima el nombre de la figura y el volumen.

Volumen de cubo = Arista^3

Volumen de cilindro = $\pi r^2 h$

Volumen de cono = $\frac{1}{3} \pi r^2 h$

Volumen de esfera = $\frac{4}{3} \pi r^3$

Debe ofrecer el siguiente menú de opciones, donde está solicitando la opción deseada:

VOLUMENES FIGURAS GEOMETRICAS:
1. CUBO 2. CILINDRO 3. CONO 4. ESFERA 5. FIN
ESCOGER OPCIÓN

La idea es que se use una superclase abstracta Figura que contendrá el dato nombre y los métodos para establecerlo y obtenerlo; además, el dato volumen, un método abstracto para calcular el volumen y un método para obtenerlo e imprimirlo. De esa superclase se deben derivar cuatro subclases: Cubo, Cilindro, Cono y Esfera, en cada una de las cuales se heredarán los datos y los métodos de la superclase Figura. Cada subclase de éstas deberá tener sus propios datos y los métodos necesarios para establecerlos, además del método para calcular el volumen de la figura correspondiente. En virtud de que calcular volumen es un método abstracto heredado de la superclase abstracta Figura, cada una de las subclases derivadas lo deberá implementar de acuerdo a la forma que le corresponda, aplicando el polimorfismo.

2. Elaborar un algoritmo que ofrezca un menú de opciones mediante el cual se pueda escoger calcular el área de las figuras geométricas trapecio, rombo y paralelogramo. Una vez seleccionada la opción, que permita solicitar y leer el nombre de la figura y los datos necesarios para calcular el área correspondiente e imprima el nombre de la figura y el área.

$$\text{Área de trapecio} = \frac{(\text{BaseMayor} + \text{BaseMenor}) \text{Altura}}{2}$$

$$\text{Área de rombo} = \frac{\text{DiagonalMayor} \times \text{DiagonalMenor}}{2}$$

$$\text{Área de paralelogramo} = \text{Base} \times \text{Altura}$$

Debe ofrecer el siguiente menú de opciones, donde está solicitando la opción deseada:

AREAS FIGURAS GEOMETRICAS:
1. TRAPECIO 2. ROMBO 3. PARALELOGRAMO 5. FIN
ESCOGER OPCIÓN

La idea es que se use una superclase abstracta Figura que contendrá el dato nombre y los métodos para establecerlo y obtenerlo; además, el dato área, un método abstracto para calcular el área y un método para obtenerlo e imprimirlo. De esa superclase se deben derivar tres subclases: Trapecio, Rombo y Paralelogramo, en cada una de las cuales se heredarán los datos y los métodos de la superclase. Cada subclase de éstas deberá tener sus propios datos y los métodos necesarios para establecerlos, además del método para calcular el área de la figura correspondiente. En virtud de que calcular área es un método abstracto heredado de la superclase abstracta Figura, cada una de las subclases lo deberá implementar de acuerdo a la forma que le corresponda, aplicando el polimorfismo.

3. Elaborar un algoritmo similar al anterior, sólo que ahora, además del área, para cada una de las figuras deberá calcular e imprimir el perímetro.

$$\text{Perímetro de trapecio} = \text{BaseMayor} + \text{BaseMenor} + \text{Lado1} + \text{Lado2}$$

$$\text{Perímetro de rombo} = 4 \times \text{Lado}$$

$$\text{Perímetro de paralelogramo} = (2 \times \text{Base}) + (2 \times \text{Altura})$$

Ahora en la superclase abstracta Figura, además de lo del problema anterior, se tendrá el dato perímetro, el método abstracto para calcular perímetro y un método para obtenerlo e imprimirlo. En virtud de que calcular perímetro es un método abstracto heredado de la superclase abstracta Figura, cada una de las subclases lo deberá implementar de acuerdo a la forma que le corresponda, aplicando el polimorfismo.

4. En una empresa automotriz se tienen tres tipos de empleados: administrativos, mecánicos y vendedores. En general, para todos los empleados se tienen los datos RFC (Registro Federal de Causantes), el nombre, el departamento y el puesto. En particular; para el empleado administrativo se tiene el dato del sueldo mensual; para el mecánico se tiene el precio del trabajo, tantas veces como trabajos haya realizado; y para el vendedor se tiene el precio del auto, por cada

auto que vendió.

El sueldo quincenal se calcula:

Para el administrativo, sueldo mensual entre 2.

Para el mecánico, el 4% del valor total de los trabajos realizados.

Para el vendedor, el salario mínimo más 2% del valor de la venta realizada.

Elaborar un algoritmo que permita procesar los empleados de la empresa e imprimir el reporte:

REPORTE DE NÓMINA QUINCENAL				
RFC	NOMBRE	DEPTO.	PUESTO	SUELDO QUINCENAL
XXXXX	XXXXXXXXXXXXXXXXXX	XXXXXXX	XXXXXXX	99,999.99

XXXXX	XXXXXXXXXXXXXXXXXX	XXXXXXX	XXXXXXX	99,999.99
TOTAL	999 EMPLEADOS			99,999.99

La idea es que se use una superclase Empleado que contendrá los datos RFC (Registro Federal de Causantes), el nombre, el departamento y el puesto, y los métodos para establecer y obtener cada uno de los datos; además, el dato sueldo quincenal, un método abstracto para calcular el sueldo quincenal y un método para obtenerlo e imprimirlo. De esa superclase se deben derivar tres subclases: EmpAdmvo, EmpMecanico y EmpVendedor, en cada una de las cuales se heredarán los datos y los métodos de la superclase. Cada subclase de éstas deberá tener sus propios datos y los métodos necesarios para establecerlos, además del método calcular el sueldo quincenal del empleado correspondiente. En virtud de que calcular sueldo quincenal es un método abstracto heredado de la superclase abstracta Empleado, cada una de las subclases lo deberá implementar de acuerdo a la forma que le corresponda, aplicando el polimorfismo.

- Elaborar un algoritmo similar al anterior, pero para una fábrica en la cual se tienen dos tipos de empleados: directivos y técnicos. Usted investigue o establezca los datos disponibles y la forma de pagarle a cada uno de los tipos de empleados. Hágalo de manera que tenga una superclase y dos subclases que hereden de la superclase.
- Elaborar un algoritmo similar al anterior, pero para un gobierno de un municipio o de un estado en el cual se tienen dos tipos de empleados: de confianza y sindicalizados. Usted investigue o establezca los datos disponibles y la forma de pagarle a cada uno de los tipos de empleados. Hágalo de manera que tenga una superclase y dos subclases que hereden de la superclase.
- En un banco se tienen clientes que son inversionistas: unos tienen cuenta de ahorro, otros tienen inversión en pagaré y otros tienen cuenta maestra. Cada tipo de inversión paga intereses de manera diferente; usted investigue o establezca cómo lo hará cada tipo de inversión. Elaborar un algoritmo que tenga una superclase Inversionista y tres subclases que hereden de la superclase, una para cada tipo de inversión: CuentaAhorro, Pagare, CuentaMaestra. Por cada tipo de cuenta se tiene el número de cliente, el nombre del cliente, el número de cuenta, el capital invertido, la tasa de interés anual y el plazo de la inversión. Imprimir el siguiente reporte:

REPORTE DE INVERSIONES			
NO. CLIENTE	NOMBRE	NO. CUENTA	INTERÉS GANADO
999999999	XXXXXXXXXXXXXXXXXX	9999999	99,999.99

999999999	XXXXXXXXXXXXXXXXXX	9999999	99,999.99
TOTAL 999	INVERSIONES		99,999.99

La idea es que se aplique el polimorfismo en calcular interés ganado; esto es, que en la superclase se tenga el dato `interesGanado` y un método abstracto que lo calcule, y en cada subclase `CuentaAhorro`, `Pagare`, `CuentaMaestra` se implemente de acuerdo a la forma que le corresponda calcular el interés ganado.

8. En un banco se tienen clientes deudores: unos tienen préstamo personal, otros tienen préstamo hipotecario y otros tienen préstamo de automóvil. Cada tipo de préstamo paga intereses de manera diferente; usted investigue o establezca cómo lo hará cada tipo de préstamo. Elaborar un algoritmo que tenga una superclase `ClienteDeudor` y tres subclases que hereden de la superclase, una para cada tipo de préstamo: `PrestamoPersonal`, `PrestamoHipotecario`, `PrestamoAuto`. Por cada tipo de préstamo se tiene el número de cliente, el nombre del cliente, el número de cuenta, el capital prestado, la tasa de interés anual y el plazo de la inversión. Imprimir el siguiente reporte:

REPORTE DE CLIENTES DEUDORES			
NO. CLIENTE	NOMBRE	NO. CUENTA	INTERÉS POR PAGAR
999999999	XXXXXXXXXXXXXXXXXX	9999999	99,999.99
--			
--			
999999999	XXXXXXXXXXXXXXXXXX	9999999	99,999.99
TOTAL 999	CLIENTES		99,999.99

La idea es que se aplique el polimorfismo en calcular interés por pagar; esto es, que en la superclase se tenga el dato `interesAPagar` y un método abstracto que lo calcule, y en cada subclase `PrestamoPersonal`, `PrestamoHipotecario`, `PrestamoAuto` se implemente de acuerdo a la forma que le corresponda calcular el interés por pagar.

14.6 Resumen de conceptos que debe dominar

- Polimorfismo
- Diseño del diagrama de clases con polimorfismo
- Clases abstractas
- Diseño de algoritmos OO usando polimorfismo

14.7 Contenido de la página Web de apoyo

El material marcado con asterisco () sólo está disponible para docentes.*

14.7.1 Resumen gráfico del capítulo

14.7.2 Autoevaluación

14.7.3 Programas en Java

14.7.4 Ejercicios resueltos

14.7.5 Power Point para el profesor (*)

15

Registros y archivos

Contenido

- 15.1 Organización de archivos
- 15.2 Manejo de registros en seudocódigo
- 15.3 Operaciones para el manejo de archivos en seudocódigo
- 15.4 Proceso de un archivo secuencial
- 15.5 Proceso de un archivo directo
- 15.6 Ejercicios resueltos
- 15.7 Ejercicios propuestos
- 15.8 Resumen de conceptos que debe dominar
- 15.9 Contenido de la página Web de apoyo
 - El material marcado con asterisco (*) sólo está disponible para docentes.*
 - 15.9.1 Resumen gráfico del capítulo
 - 15.9.2 Autoevaluación
 - 15.9.3 Programas en Java
 - 15.9.4 Ejercicios resueltos
 - 15.9.5 Power Point para el profesor (*)

Objetivos del capítulo

- Aprender los conceptos de registros y archivos.
- Estudiar la organización de archivos.
- Aprender cómo manejar registros y archivos en seudocódigo.
- Estudiar el proceso de archivos con organización secuencial y organización directa.
- Aplicar lo aprendido en el desarrollo de algoritmos para programas grandes conformando sistemas de información.

Competencias

- Competencia general del capítulo
 - *Analizar problemas y diseñar algoritmos que los solucionen aplicando la arquitectura orientada a objetos, registros y archivos.*
- Competencias específicas del capítulo
 - Define los conceptos de registros y archivos.
 - Describe la organización de archivos.
 - Detalla cómo manejar registros y archivos en seudocódigo.
 - Diseña algoritmos orientados a objetos aplicando el proceso de un archivo secuencial.
 - Elabora algoritmos orientados a objetos aplicando el proceso de un archivo directo.

Introducción

Con el estudio del capítulo anterior, usted ya domina el polimorfismo.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos utilizando las estructuras de datos: registros y archivos.

Se explica que un dato describe un atributo o característica de un objeto o sujeto de proceso de datos, que el registro es un tipo de dato formado por un conjunto de datos que describen a un objeto o sujeto de proceso de datos y que el archivo es un tipo de dato formado por un conjunto de registros que contienen datos acerca de un mismo objeto o sujeto de proceso de datos.

Se puntualiza que los archivos se almacenan en medios externos, como son los discos magnéticos duros o flexibles, dispositivos ópticos, cintas magnéticas, entre otros, mismos que permiten el almacenamiento permanente de datos para usos futuros y constituyen un elemento esencial en el uso de las computadoras.

Se define que la organización de archivos es el procedimiento que sirve para relacionar la forma de registrar los datos con la forma de procesarlos, que existen diversas formas de organización como son la secuencial, directa o relativa, secuencial indexado, indexado no secuencial, entre otras. Se expone el manejo de registros en seudocódigo, las operaciones para el manejo de archivos en seudocódigo, características de los archivos, tipos de archivos, operaciones sobre archivos y operaciones con registros. Se estudia el proceso de un archivo con organización secuencial y el proceso de un archivo con organización directa. Se presenta el desarrollo de algoritmos para programas grandes conformando sistemas de información.

Es pertinente recordar que si el estudiante no hace algoritmos, no aprende; es por ello que es esencial que ejercite estudiando los problemas planteados en los ejercicios resueltos y propuestos. Al estudiar los ejercicios resueltos se le recomienda que primero diseñe usted la solución, sin ver la solución propuesta en el libro, luego verifique sus resultados con los del libro, analice las diferencias y vea sus errores. Al principio es normal que cometa errores, poco a poco deberá ir eliminándolos.

Si su algoritmo no está igual que el del libro, no necesariamente está mal. Usted debe ir aprendiendo a analizar las diferencias y a comprender que a veces, aunque haya diferencias, las dos soluciones están correctas.

En el siguiente capítulo se estudian otros temas.

Los tipos de datos que se han tratado hasta este momento están limitados por residir sólo en la memoria principal. Esto significa que si se apaga la computadora desaparecen. Otra limitante es la cantidad de datos que se podrían almacenar, ya que este tipo de memoria es muy limitada por su alto costo.

En este capítulo estudiaremos la estructura de datos llamada archivos, los cuales se almacenan en medios externos como son los discos magnéticos fijos, disquetes (discos flexibles) magnéticos removibles, discos compactos, cintas magnéticas, memorias USB, entre otros, mismos que permiten el almacenamiento permanente de datos para usos futuros y constituyen un elemento esencial en el uso de las computadoras.

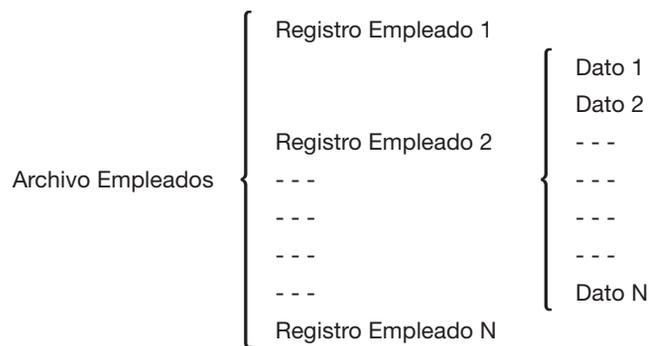
Un **archivo** es un conjunto ordenado de registros que contienen datos acerca de un mismo objeto o sujeto de proceso de datos.

Un **registro lógico** es un conjunto de datos que describen a un objeto o sujeto de proceso de datos.

Un **dato** describe un atributo o característica de un objeto o sujeto de proceso de datos.

Ejemplo:

Un archivo Empleados está formado por un conjunto de registros, donde cada registro representa a un empleado y está compuesto por un conjunto de datos elementales que describen al empleado. Esquemmatizando, tenemos:



Otra forma de esquematizarlo sería:

Archivo Empleados

	Número	Nombre	Depto.	Puesto	Sueldo
Registro 1	10	IGNACIO CAMACHO	1	1	1200
Registro 2	20	MEDARDO OBESO	1	2	1345
Registro 3	30	PORFIRIO LOPEZ	2	1	1750
Registro 4	40	CAMEROLY OBESO	2	3	1450
Registro 5	50	CELEDONIO FLORES	3	1	3400

Registro N	90	ANTONIO HIGUERA	4	5	1250

Nota: El Depto 1 puede ser Compras, el 2 Contabilidad, el 3 Ventas, etcétera. El Puesto 1 puede ser Director, el 2 Sub-director, el 3 Secretaria, el 4 Auxiliar, etcétera.

La **llave o clave** es un dato o grupo de datos dentro del registro que sirve para identificar de manera única a cada registro. En el caso del archivo Empleados, el número es la llave que identifica a cada empleado.



El Depto 1 puede ser Compras, el 2 Contabilidad, el 3 Ventas, etcétera. El Puesto 1 puede ser Director, el 2 Sub-director, el 3 Secretaria, el 4 Auxiliar, etcétera.

El **registro físico o bloque** es la unidad de datos de lectura o escritura de un medio físico de archivo de datos, el cual puede contener varios registros lógicos. Su tamaño depende de las características físicas de la computadora. En el caso de las micros es fijo, generalmente de 256; en las minis, de 512; en las computadoras grandes es variable, y el usuario puede definirlo en múltiplos de 256.

El **factor de bloque** es el número de registros lógicos que contiene un registro físico o bloque.

De aquí en adelante, cuando mencionemos la palabra **registro** nos estaremos refiriendo al concepto de **registro lógico**.

Características de los archivos

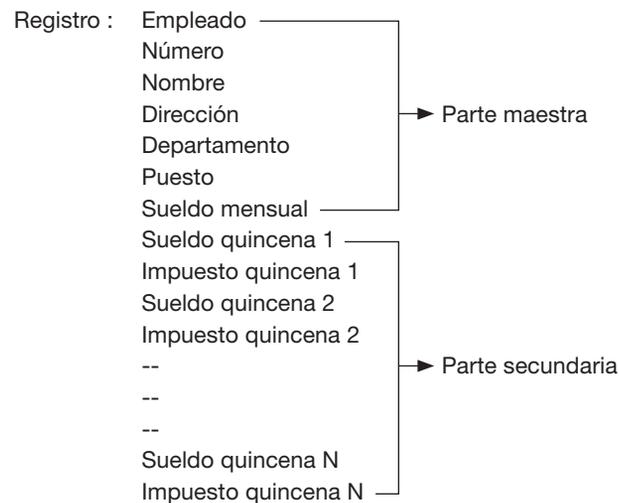
- Son independientes con respecto a los programas.
- Un archivo puede ser utilizado por distintos programas en diversos momentos.
- La información almacenada es permanente.
- Tienen gran capacidad de almacenamiento.
- La recuperación de datos se hace con rapidez.
- Su índice de confiabilidad es muy alto.

Partes de un registro

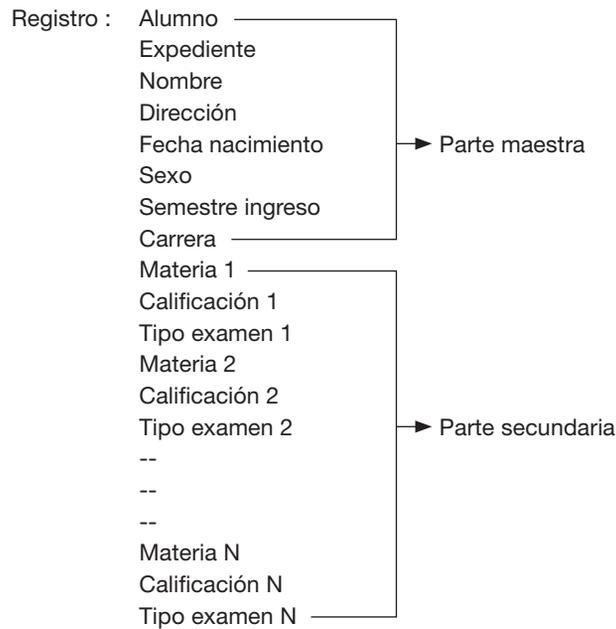
Parte maestra: Está definida por la información que está sujeta a pocos cambios y sirve como historia, identificación y referencia.

Parte(s) secundaria(s): Está definida por la información que está sujeta a cambios constantes, sobre todo cambios de expansión.

Ejemplo:



Ejemplo :



Tipos de archivos

Archivos maestros: Contienen registros con información general de poca variación. Sirven como identificación, referencia, estadística, historia. Ejemplo: archivo maestro de empleados, archivo de alumnos, archivo de artículos, archivo de clientes, etcétera.

Archivos de transacciones: Contienen registros con datos que describen las operaciones de la organización. Surgen como resultado de algún proceso de transacciones y sirven para preparar documentos de movimientos diarios, semanales, quincenales, etcétera. Generalmente son temporales y se utilizan para actualizar los archivos maestros. Ejemplos: archivo de movimientos de almacén, archivo de ventas, archivo de abonos, archivo de inscripción, etcétera.

Archivos de trabajo: Se utilizan como auxiliares en el proceso de los dos tipos de archivos anteriores. Su duración normalmente es menor a la duración de la ejecución de un programa.

Operaciones sobre archivos

Creación	Crear el archivo y escritura de sus registros.
Consulta	Lectura de registros.
Actualización	Inserción (altas), supresión (bajas) y modificación (cambios) en algunos registros.
Clasificación	Reordenamiento de los registros de acuerdo con cierto criterio.
Borrado	Eliminación total del archivo.

Operaciones con registros

Inserción	Añadir un nuevo registro al archivo.
Supresión	Quitar un registro del archivo.
Consulta	Leer el contenido de un registro.
Actualización	Cambiar datos de un registro.

15.1 Organización de archivos

Es el procedimiento que sirve para relacionar la forma de registrar los datos con la forma de procesarlos. La organización de archivos proporciona los métodos para la creación, actualización y acceso de archivos.

Tipos de organización de archivos

- Secuencial
- Directa o relativa
- Secuencial indexado
- Indexado no secuencial
- Otras

El registro de los datos es la operación que consiste en introducir los datos al archivo con el propósito de que queden almacenados para usos posteriores, y puede ser:

- Secuencial
- Directo
- En desorden

El proceso o acceso consiste en buscar datos en el archivo con el propósito de consultarlos o modificarlos, y puede ser:

- Secuencial
- Directo

Organización secuencial

Esta forma de organización de archivos tiene las siguientes características:

- Los registros se almacenan físicamente de acuerdo con una llave dada en secuencia ascendente o descendente.
- Es muy eficaz para el proceso periódico en lotes.
- Cuando se requiere buscar un registro determinado, es necesario recorrer todo el archivo en forma secuencial, uno a uno, hasta encontrarlo.
- Es conveniente que el último registro del archivo sea un centinela que indique el fin.
- No es eficiente para la búsqueda de información.
- Si se encuentra posicionado en un determinado registro, no se puede regresar; se debe reiniciar el proceso desde el principio.
- Si quiere dar de baja un registro, no puede hacerlo directamente. Sólo se marca como dado de baja. Posteriormente, se deberá hacer un proceso especial de reorganización para que los registros se den de baja físicamente.

- Cuando se desea actualizar el archivo con altas, bajas y cambios, es necesario crear otro archivo con las modificaciones, luego pasar los dos archivos completos, creando uno nuevo actualizado.

Ejemplo:

Se tiene el archivo Empleados, en el que se han registrado los datos en orden secuencial de acuerdo al Número, como se muestra a continuación:

Archivo Empleados:

Número	Nombre	Depto.	Puesto	Sueldo
1	MARICELA VILLA	1	1	1200
2	EDILIA CAMACHO	1	2	1450
3	DANIA CAMACHO	1	3	1750
4	JUAN DE DIOS LOPEZ	2	1	1450
5	CAROLINA OBESO	2	2	2300
6	MANUEL MEZA	3	1	3000
7	ARACELI LOPEZ	3	2	1500

Organización directa o relativa

Cuando hablamos de organización directa o de acceso directo a un archivo, lo que quiere decir es que los datos se localizan en una posición conocida por el dispositivo, el cual es de acceso directo. Los registros se almacenan en una dirección relativa conocida, y siempre de acuerdo con la llave. Así, el registro con la llave número 1 se guarda a partir del byte 0 del archivo, el registro con la llave 2 se guarda a partir del byte siguiente después de donde se almacenó el registro anterior y así sucesivamente, como se muestra a continuación: el registro de un empleado ocupa 48 bytes (4 bytes el número, 4 el departamento, 4 el puesto, 4 el sueldo y 32 el nombre, considerando 30 para los caracteres del nombre y 2 que usa el lenguaje de programación para indicar la cantidad de caracteres del dato).

Archivo Empleados:

Dirección del byte	Número	Nombre	Depto.	Puesto	Sueldo
0	1	JESUS CRISTO	4	1	1200
48	2	MARIA ALIMATEA	4	2	1450
96	3	CAMEROLY O. LOPEZ	8	1	1750
144	4	PEDRO APOSTOL	9	1	1450
192	5	CLAUDIA OBESO	9	2	2300

N-1*48	N	MARIA L. LOPEZ	10	3	2500

Nota: N es la llave Número de empleado.

Para el acceso a los registros, se debe conocer la llave y se busca la posición correspondiente de acuerdo al número de empleado y a los 48 bytes que ocupa el registro del empleado.



N es la llave Número de empleado.

15.2 Manejo de registros enseudocódigo

Como se explicó al inicio de este capítulo, el concepto de registro está íntimamente ligado al concepto de archivos. Sin embargo, es posible manejarlo en forma independiente. El registro es un tipo de dato estructurado constituido por un conjunto de elementos individuales (datos) que pueden ser de diferentes tipos de datos.

Definición de registros

El registro se define como una clase, como se muestra a continuación:

```

Clase NomRegistro
1. Declarar datos
   dato1: Tipo de dato
   dato2: Tipo de dato
   datoN: Tipo de dato

2. Método establecerDato1(da1: Tipo de dato)
   a. dato1 = da1
   b. Fin Método establecerDato1

3. Método establecerDato2(da2: Tipo de dato)
   a. dato2 = da2
   b. Fin Método establecerDato2

4. Método establecerDatoN(daN: Tipo de dato)
   a. datoN = daN
   b. Fin Método establecerDatoN

5. Método obtenerDato1(): Tipo de dato
   a. return dato1
   b. Fin Método obtenerDato1

6. Método obtenerDato2(): Tipo de dato
   a. return dato2
   b. Fin Método obtenerDato2

7. Método obtenerDatoN(): Tipo de dato
   a. return datoN
   b. Fin Método obtenerDatoN
Fin Clase NomRegistro

```

En donde:

Clase	Indica el inicio de la definición de una clase.
NomRegistro	Es el nombre mediante el cual se identificará la clase que representa un registro.
Declarar datos	Indica que se declaran los datos de la clase que representa al registro.
dato1, dato2, datoN	Son los identificadores de los datos individuales que forman el registro, cada uno con su definición particular de tipo de dato.

establecerDato1, establecerDato2, establecerDatoN	Son los métodos setters que establecen los valores a cada dato del registro representado en esta clase.
obtenerDato1, obtenerDato2, obtenerDatoN	Son los métodos getters que obtienen los valores de cada dato del registro representado en esta clase.
Fin Clase	Indica que se termina la definición de la clase que representa al registro.

Proceso de un registro

Una vez que se tiene definida una clase que representa un registro, se puede utilizar a través de otra clase (ejecuta), como se muestra a continuación:

```

Clase EjecutaNomRegistro
1. Método principal()
  a. Declarar variables
    dat1: Tipo de dato
    dat2: Tipo de dato
    datN: Tipo de dato
  b. Declarar, crear e iniciar objeto
    NomRegistro objRegistro = new NomRegistro()
  c. Solicitar dato 1, dato 2, dato N
  d. Leer dat1, dat2, datN
  e. Establecer
    objRegistro.establecerDato1(dat1)
    objRegistro.establecerDato2(dat2)
    objRegistro.establecerDatoN(datN)
  f. Imprimir
    objRegistro.obtenerDato1()
    objRegistro.obtenerDato2()
    objRegistro.obtenerDatoN()
  g. Fin Método principal
Fin Clase EjecutaNomRegistro

```

Explicación:

Se tiene la Clase EjecutaNomRegistro, la cual utilizará a la clase NomRegistro. Esta clase tiene el Método principal(), en el cual:

- Se declaran las variables:
dat1: Tipo de dato
dat2: Tipo de dato
datN: Tipo de dato
- Se declara, crea e inicia el objeto objRegistro utilizando como base la clase modelo NomRegistro y el método constructor por defecto NomRegistro(). Todo esto se hace con el operador new.
Se crea una instancia de la clase NomRegistro, en la cual se representa y maneja el registro como un objeto.

- c. Se solicitan los datos: dato 1, dato 2, dato N.
 - d. Se leen en dat1, dat2, datN.
 - e. Se llama al Método establecerDato1 (dat1) del objeto objRegistro para colocar el valor de dat1 en el dato dato1.
Se llama al Método establecerDato2 (dat2) del objeto objRegistro para colocar el valor de dat2 en el dato dato2.
Se llama al Método establecerDatoN (datN) del objeto objRegistro para colocar el valor de datN en el dato datoN.
 - f. Se llama al Método obtenerDato1 () del objeto objRegistro para acceder e imprimir el valor del dato1.
Se llama al Método obtenerDato2 () del objeto objRegistro para acceder e imprimir el valor del dato2.
Se llama al Método obtenerDatoN () del objeto objRegistro para acceder e imprimir el valor del datoN.
 - g. Fin del Método principal.
- Fin de la Clase EjecutaNomRegistro.
Fin del algoritmo.

Ejemplo:

Elaborar un algoritmo que lea e imprima los datos de varios empleados utilizando el registro regEmpleado, que contiene los datos:

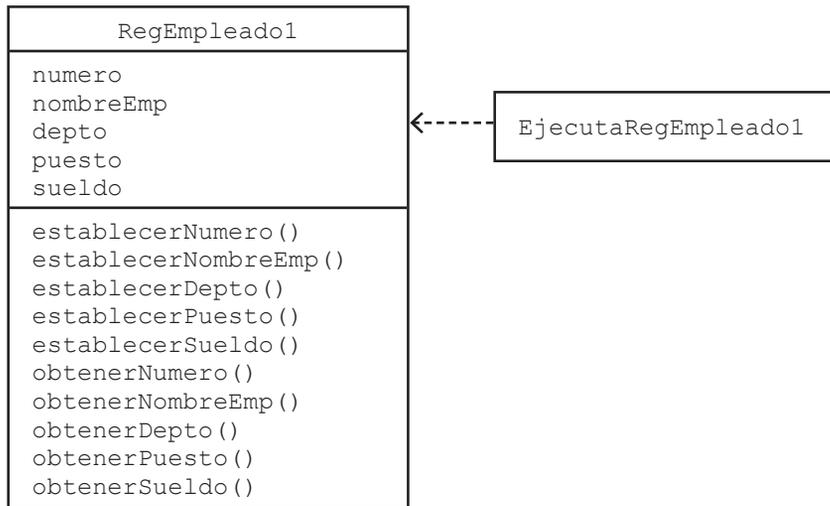
- numero
- nombreEmp
- depto
- puesto
- sueldo

En donde:

numero es tipo Entero
nombreEmp es Cadena
depto es Entero
puesto es Entero
sueldo es Real

A continuación tenemos el algoritmo de la solución en dos pasos: primero el diagrama de clases y después el algoritmo en seudocódigo.

Diagrama de clases



Algoritmo LEE E IMPRIME REGISTROS DE EMPLEADOS

Clase RegEmpleado1

1. Declarar datos

```

numero: Entero
nombreEmp: Cadena
depto : Entero
puesto: Entero
sueldo: Real
  
```

2. Método establecerNumero(num: Entero)

```

a. numero = num
b. Fin Método establecerNumero
  
```

3. Método establecerNombreEmp(nom: Cadena)

```

a. nombreEmp = nom
b. Fin Método establecerNombreEmp
  
```

4. Método establecerDepto(dep: Entero)

```

a. depto = dep
b. Fin Método establecerDepto
  
```

5. Método establecerPuesto(pue: Entero)

```

a. puesto = pue
b. Fin Método establecerPuesto
  
```

6. Método establecerSueldo(sue: Real)

```

a. sueldo = sue
b. Fin Método establecerSueldo
  
```

7. Método obtenerNumero(): Entero

```

a. return numero
b. Fin Método obtenerNumero
  
```

```

8. Método obtenerNombreEmp(): Cadena
  a. return nombreEmp
  b. Fin Método obtenerNombreEmp

9. Método obtenerDepto(): Entero
  a. return depto
  b. Fin Método obtenerDepto

10. Método obtenerPuesto(): Entero
  a. return puesto
  b. Fin Método obtenerPuesto

11. Método obtenerSueldo(): Real
  a. return sueldo
  b. Fin Método obtenerSueldo
Fin Clase RegEmpleado1

Clase EjecutaRegEmpleado1
1. Método principal()
  a. Declarar variables
     numEmp: Entero
     nomEmp: Cadena
     depEmp: Entero
     pueEmp: Entero
     sueEmp: Real
     desea : Carácter
  b. do
     1. Declarar, crear e iniciar objeto
        RegEmpleado1 objEmpleado = new RegEmpleado1()
     2. Solicitar número, nombre, departamento, puesto, sueldo
     3. Leer numEmp, nomEmp, depEmp, pueEmp, sueEmp
     4. Establecer
        objEmpleado.establecerNumero(numEmp)
        objEmpleado.establecerNombreEmp(nomEmp)
        objEmpleado.establecerDepto(depEmp)
        objEmpleado.establecerPuesto(pueEmp)
        objEmpleado.establecerSueldo(sueEmp)
     5. Imprimir
        objEmpleado.obtenerNumero()
        objEmpleado.obtenerNombreEmp()
        objEmpleado.obtenerDepto()
        objEmpleado.obtenerPuesto()
        objEmpleado.obtenerSueldo()
     6. Preguntar "¿Desea procesar otro empleado(S/N)?"
     7. Leer desea
  c. while desea == 'S'
  d. Fin Método principal
Fin Clase EjecutaRegEmpleado1
Fin

```



En la zona de descarga de la Web del libro está disponible:
 Programa en Java: RegEmpleado1.java y EjecutaRegEmpleado1.java

Explicación:

El algoritmo tiene dos clases: la Clase `RegEmpleado1` y la Clase `EjecutaRegEmpleado1`.

En la Clase `RegEmpleado1`:

1. Se declaran los datos que representan la estructura de la clase:
 - `numero` para el dato número del empleado.
 - `nombreEmp` para el nombre del empleado.
 - `depto` para el departamento del empleado.
 - `puesto` para el puesto del empleado.
 - `sueldo` para el sueldo mensual del empleado.
 2. Método `establecerNumero(num: Entero)`.
Recibe en el parámetro `num` el valor que luego coloca en el dato `numero`.
 3. Método `establecerNombreEmp(nom: Cadena)`.
Recibe en el parámetro `nom` el valor que luego coloca en el dato `nombreEmp`.
 4. Método `establecerDepto(dep: Entero)`.
Recibe en el parámetro `dep` el valor que luego coloca en el dato `depto`.
 5. Método `establecerPuesto(pue: Entero)`.
Recibe en el parámetro `pue` el valor que luego coloca en el dato `puesto`.
 6. Método `establecerSueldo(sue: Real)`.
Recibe en el parámetro `sue` el valor que luego coloca en el dato `sueldo`.
 7. Método `obtenerNumero(): Entero`.
Retorna `numero`.
 8. Método `obtenerNombreEmp(): Cadena`.
Retorna `nombreEmp` (nombre del empleado).
 9. Método `obtenerDepto(): Entero`.
Retorna `depto`.
 10. Método `obtenerPuesto(): Entero`.
Retorna `puesto`.
 11. Método `obtenerSueldo() Real`.
Retorna `sueldo`.
- Fin de la Clase `RegEmpleado1`.

En la Clase `EjecutaRegEmpleado1`, en el Método `principal()`:

- a. Se declaran las variables:
 - `numEmp` para leer el número del empleado.
 - `nomEmp` para leer el nombre del empleado.
 - `depEmp` para leer el departamento del empleado.
 - `pueEmp` para leer el puesto del empleado.
 - `sueEmp` para leer el sueldo mensual del empleado.
 - `desea` para controlar el ciclo repetitivo.
- b. Inicia ciclo `do`:
 1. Se declara el objeto `objEmpleado`, usando como base a la Clase `RegEmpleado1`. Dicho objeto se crea e inicializa mediante el constructor por defecto `RegEmpleado1()`.
Observe que cada vez que entra al ciclo crea una nueva instancia de la Clase `RegEmpleado1`, en la cual se representa y maneja como un objeto (`objEmpleado`) un registro de un empleado.

2. Se solicitan número, nombre del empleado, departamento, puesto y sueldo mensual.
 3. Se leen en numEmp, nomEmp, depEmp, pueEmp, sueEmp.
 4. Se llama al Método establecerNumero(numEmp) del objeto objEmpleado para colocar el valor de numEmp en el dato numero.
Se llama al Método establecerNombreEmp(nomEmp) del objeto objEmpleado para colocar el valor de nomEmp en el dato nombreEmp.
Se llama al Método establecerDepto(depEmp) del objeto objEmpleado para colocar el valor de depEmp en el dato depto.
Se llama al Método establecerPuesto(pueEmp) del objeto objEmpleado para colocar el valor de pueEmp en el dato puesto.
Se llama al Método establecerSueldo(sueEmp) del objeto objEmpleado para colocar el valor de sueEmp en el dato sueldo.
 5. Se llama al Método obtenerNumero() del objeto objEmpleado para acceder e imprimir el valor del dato numero.
Se llama al Método obtenerNombreEmp() del objeto objEmpleado para acceder e imprimir el valor del dato nombreEmp.
Se llama al Método obtenerDepto() del objeto objEmpleado para acceder e imprimir el valor del dato depto.
Se llama al Método obtenerPuesto() del objeto objEmpleado para acceder e imprimir el valor del dato puesto.
Se llama al Método obtenerSueldo() del objeto objEmpleado para acceder e imprimir el valor del dato sueldo.
 6. Se pregunta “¿Desea procesar otro empleado(S/N)?”.
 7. Se lee la respuesta en desea.
- c. Fin del ciclo (while desea == 'S'). Si se cumple regresa al do; si no, se sale del ciclo.
- d. Fin del Método principal.
- Fin de la Clase EjecutaRegEmpleado1.
- Fin del algoritmo.

15.3 Operaciones para el manejo de archivos en pseudocódigo

La estructura de datos archivo está formada por un conjunto de objetos, donde cada objeto representa un registro, y se almacena en un dispositivo auxiliar de almacenamiento como discos magnéticos, duros, flexibles, dispositivos ópticos, cintas, etcétera. Es decir, los datos que se registran en archivos quedan almacenados en forma permanente. Cuando se crea un archivo, no se indica su tamaño en número de componentes: se inicia con el byte 0 (cero), cuando se escribe sobre él, se crea el siguiente byte y así sucesivamente, la única limitante en el número de componentes (bytes) del archivo, es la capacidad del dispositivo de almacenamiento. A continuación se explican las operaciones para el manejo de archivos en pseudocódigo.

Creación de archivo secuencial

Cuando se va a utilizar un archivo, primero es necesario crearlo, a fin de que le sea asignado un lugar en el dispositivo físico de acuerdo con los datos que contendrá, y asignar algunas identificaciones que requiere la computadora. En caso de crear

un archivo que ya existe, significaría destruir el existente y comenzar de nuevo. En el momento en que se crea un archivo, no contiene ningún componente; en consecuencia, lo único que se puede hacer es escribir en éste. El apuntador se coloca en el primer componente, en el byte 0 (cero).

Formato:

```
Crear (nomArchivo, nomFisico, Secuencial)
```

En donde:

Crear	Identifica la operación de crear inicialmente el archivo.
nomArchivo	Es el nombre del archivo que se creará, mediante el cual se identificará.
nomFisico	Es el nombre físico (en el disco) del archivo que se creará. Debe incluir el path. Ejemplo: "C:/arEmp.dat".
Secuencial	Es el tipo de organización del archivo: secuencial.

Ejemplo:

```
Crear (empleados, "C:/arEmp.dat", Secuencial)
```

Explicación:

Se crea el archivo `empleados` con organización secuencial. Su nombre físico en el disco es "C:/arEmp.dat"; sólo se puede escribir en él.

Creación de archivo directo

También conocido como de acceso aleatorio (Random). Cuando se va a utilizar un archivo, primero es necesario crearlo, a fin de que le sea asignado un lugar en el dispositivo físico de acuerdo con los datos que contendrá, y asignar algunas identificaciones que requiere la computadora. En caso de crear un archivo que ya existe, significaría destruir el existente y comenzar de nuevo. En el momento en que se crea un archivo, no contiene ningún componente; en consecuencia, lo único que se puede hacer es escribir en éste. El apuntador se coloca en el primer componente, en el byte 0 (cero).

Formato:

```
Crear (nomArchivo, nomFisico, Directo, "modo")
```

En donde:

Crear	Identifica la operación de crear inicialmente el archivo.
nomArchivo	Es el nombre del archivo que se creará, mediante el cual se identificará.
nomFisico	Es el nombre físico (en el disco) del archivo que se creará. Debe incluir el path. Ejemplo: "C:/arEmp.dat".

Directo modo	Es el tipo de organización del archivo: directo. El modo puede ser: w Indica que es sólo de escritura. Se usa al crear el archivo. El apuntador se coloca en el primer componente, que está vacío; por ello sólo se puede escribir. r Indica que es sólo de lectura. El apuntador se coloca en el primer componente y sólo se puede leer. rw Indica que es de lectura/escritura. El apuntador se coloca en el primer componente y se puede leer o escribir, o se puede mover el apuntador para posteriormente leer o escribir.
--------------	--

Ejemplo:

```
Crear (empleados, "C:/arEmp.dat", Directo, "rw")
```

Explicación:

Se crea el archivo `empleados` con organización directa. Su nombre físico en el disco es `"C:/arEmp.dat"`, en modo lectura/escritura.

Abrir archivo secuencial

Esta operación permite preparar (abrir) un archivo existente para ser utilizado. El apuntador se coloca en el primer componente y sólo permite leer del archivo. En caso de que el archivo no exista habrá error.

Formato:

```
Abrir (nomArchivo, nomFisico, Secuencial)
```

En donde:

Abrir	Identifica la operación de apertura de archivo.
nomArchivo	Es el nombre que identifica al archivo.
nomFisico	Es el nombre físico (en el disco) del archivo que se abrirá.
Secuencial	Indica que es un archivo con organización secuencial.

Ejemplo:

```
Abrir (empleados, "C:/arEmp.dat", Secuencial)
```

Explicación:

Se abre el archivo `empleados` con organización secuencial. Su nombre físico en el disco es `"C:/arEmp.dat"`; sólo se puede leer de él.

Abrir archivo directo

Esta operación permite preparar (abrir) un archivo existente para ser utilizado. El apuntador se coloca en el primer componente y permite leer y escribir sobre el archivo; es decir, se pueden hacer altas, bajas, cambios, consultas, etcétera. En caso de que el archivo no exista habrá error.

Formato:

```
Abrir (nomArchivo, nomFisico, Directo, "modo")
```

En donde:

Abrir	Identifica la operación de apertura de archivo.
nomArchivo	Es el nombre que identifica al archivo.
nomFisico	Es el nombre físico (en el disco) del archivo que se abrirá.
modo	El modo puede ser: <ul style="list-style-type: none">r Indica que es sólo de lectura. El apuntador se coloca en el primer componente y sólo se puede leer.rw Indica que es de lectura/escritura. El apuntador se coloca en el primer componente y se puede leer o escribir, o se puede mover el apuntador para posteriormente leer o escribir.

Ejemplo:

```
Abrir (empleados, "C:/arEmp.dat", Directo, "rw")
```

Explicación:

Se abre el archivo `empleados` con organización directa. Su nombre físico en el disco es `"C:/arEmp.dat"`. Se abre en modo lectura/escritura, y se puede leer o escribir en éste.

Escritura de objetos (registros)

La operación de escribir o imprimir un objeto, que representa un registro, consiste en grabarlo a partir del byte donde esté ubicado el apuntador del archivo en ese momento. Después de imprimir, el apuntador avanza al siguiente byte del archivo.

Formato:

```
Imprimir (nomArchivo, objRegistro)
```

En donde:

Imprimir	Identifica la acción de escritura.
nomArchivo	Es el nombre de identificación del archivo en el que se hará la escritura.
objRegistro	Es el nombre de identificación del objeto (registro) que contiene los datos que se grabarán a partir del byte actual del archivo, mismos que debieron ser previamente establecidos en el objeto.

Ejemplo:

```
Imprimir (empleados, objEmpleado)
```

Explicación:

Se imprime en el archivo `empleados` el objeto (registro) `objEmpleado`.

Lectura de objetos (registros)

Esta operación permite leer un objeto a partir del byte donde se encuentra ubicado actualmente el apuntador del archivo. Después de la lectura, el apuntador avanza al siguiente byte. Cuando se lee el último componente el apuntador avanza al fin del archivo (EOF). Si trata de leer cuando el apuntador está en el fin del archivo (EOF) habrá error.

Formato:

```
Leer (nomArchivo, objRegistro)
```

En donde:

<code>Leer</code>	Identifica la operación de lectura.
<code>nomArchivo</code>	Es el nombre de identificación del archivo.
<code>objRegistro</code>	Es el nombre de identificación del objeto en el cual se va a colocar el objeto (registro) leído.

Ejemplo:

```
Leer (empleados, objEmpleado)
```

Explicación:

Se lee el objeto (registro) del componente actual del archivo `empleados` y se coloca en el objeto (registro) `objEmpleado`.

Localización de componente (encontrar)

Es una operación que permite colocar o mover el apuntador del archivo en un determinado componente (byte). Esta operación sólo aplica para archivos con organización directa.

Formato:

```
nomArchivo.encontrar(n)
```

En donde:

<code>encontrar</code>	Identifica la acción de localizar un componente.
<code>nomArchivo</code>	Es el nombre del archivo en el que se hará.
<code>n</code>	Es el número del componente (byte) en el que se colocará el apuntador del archivo. Puede ser una variable, constante o expresión de tipo entero, entre 0 y $n-1$, donde n es el tamaño del archivo en número de bytes.

Ejemplo:

```
empleados.encontrar(50)
```

Explicación:

Coloca el apuntador del archivo `empleados` en el byte número 50.

Nota: Si el archivo `empleados` tiene n bytes utilizados (del 0 al $n-1$), y se desea agregar más bytes, se hace moviendo el apuntador más allá del último byte utilizado ($n-1$), así:

```
empleados.encontrar(n)
```

Observación: Como el último byte es el $n-1$, el nuevo que se agregará es el n .

Cerrar archivos

Esta operación permite liberar (cerrar) de uso un archivo que esté abierto. Todo archivo debe ser cerrado al terminarse de usar.

Formato:

```
Cerrar (nomArchivo)
```

En donde:

<code>Cerrar</code>	Identifica la operación de cierre del archivo.
<code>nomArchivo</code>	Es el nombre del archivo que se cerrará.

Ejemplo:

```
Cerrar (empleados)
```

Explicación:

Cierra o libera de uso el archivo identificado como `empleados`, el cual se encuentra abierto por haber sido creado o abierto.

Funciones para el proceso de archivos

`Eof (nomArchivo)` Es una función booleana que proporciona un valor verdadero (`true`) si el apuntador está al final del archivo; en caso contrario, proporciona el valor falso (`false`).

`TamañoAr (nomArchivo)` Proporciona el número de componentes (bytes) que tiene el archivo.

`nomArchivo.PosApuntador()` Proporciona el número del componente en el que se encuentra actualmente el apuntador del archivo.

`Renombra (nomArchivo, "nuevoNombre")` Permite renombrar o cambiar el nombre de un archivo.

`BorrarAr (nomArchivo)` Permite borrar un archivo.

`TruncaAr (nomArchivo)` Permite truncar el tamaño del archivo suprimiendo los elementos después de la posición actual del apuntador.



Si el archivo `empleados` tiene n bytes utilizados (del 0 al $n-1$), y se desea agregar más bytes, se hace moviendo el apuntador más allá del último byte utilizado ($n-1$), así:

```
empleados.encontrar(n)
```

Observación: Como el último byte es el $n-1$, el nuevo que se agregará es el n .

15.4 Proceso de un archivo secuencial

Cuando se va a utilizar un archivo con organización secuencial, primero debe crearse el archivo, enseguida se le hacen altas de registros (objetos) y posteriormente puede agrandarse añadiéndole más registros (objetos) con las nuevas altas que se generen. El último registro (objeto) debe ser una marca o “centinela” que indique el fin del archivo. Un archivo con organización secuencial puede ser actualizado con altas, bajas y cambios mediante la creación de otro archivo que contenga estos movimientos. Posteriormente, se hace un proceso especial de actualización en el que se crea un nuevo archivo actualizado. Finalmente, el archivo es utilizado para leer sus registros (objetos) y emitir reportes o listados de sus datos. A continuación se explica en qué consiste cada uno de dichos procesos:

Creación

Esta operación permite crear el archivo dejándolo listo para recibir altas de registros (objetos). Se dan de alta registros (objetos). Después de dar todas las altas, al final del archivo se crea un último registro (objeto) que funge como centinela o indicador de fin del archivo. Este proceso se hace una sola vez.

Expansión

Un archivo secuencial existente, como el creado con el proceso anterior, puede agrandarse agregándole nuevos registros (objetos) a partir del último, el cual funge como centinela de fin. Se le agregan los nuevos registros (objetos) o altas a partir del último registro (objeto) del archivo y, al final, se coloca el indicador (centinela) de fin de archivo de nuevo. Este proceso se hace tantas veces como necesidad se tenga de agregar nuevos registros (objetos) al archivo. Para evitar una complejidad innecesaria, este proceso no se ejemplificará en este punto; se hará en el siguiente apartado en el proceso de un archivo directo.

Actualización con altas, bajas y cambios

Cuando se tiene un archivo secuencial y se desea actualizar con altas, bajas y cambios, es necesario:

1. Crear otro archivo con los movimientos, el cual contendrá, además de los datos del archivo original, otro dato que es el tipo de movimiento que puede ser alta, baja o cambio (A, B, C), incluyendo al final el centinela de fin.
2. Procesar los dos archivos anteriores. Este proceso hace que los movimientos se reflejen en el archivo original y generan un nuevo archivo: el original actualizado.

Para evitar una complejidad innecesaria, este proceso no se ejemplificará en este punto; se hará en el siguiente apartado en el proceso de un archivo directo.

Obtención de reportes

Un archivo al que ya se le dieron de alta registros (objetos) puede ser utilizado para emitir información en forma de reportes o listados de sus datos. A partir de un archivo secuencial, el procedimiento es el siguiente: abrir el archivo, leer el primer registro, hacer los cálculos necesarios y enlistarlo; enseguida se lee el siguiente registro, se hacen cálculos y se enlistan, y así sucesivamente hasta encontrar el fin del archivo.

Ejemplo:

Obtener un reporte del archivo Empleados con el siguiente formato:

CATÁLOGO DE EMPLEADOS				
NÚMERO	NOMBRE	DEPTO.	PUESTO	SUELDO
99	XXXXXXXXXXXXXXXXXXXX	99	99	99,999.99
99	XXXXXXXXXXXXXXXXXXXX	99	99	99,999.99

99	XXXXXXXXXXXXXXXXXXXX	99	99	99,999.99
TOTAL	999 EMPLEADOS			999,999.99

Luego, con base en el archivo Empleados, emitir el reporte de nómina quincenal:

NÓMINA QUINCENAL				
NÚMERO	NOMBRE	S. BRUTO	IMPUESTO	S. NETO
99	XXXXXXXXXXXXXXXXXXXX	99,999.99	99,999.99	99,999.99
99	XXXXXXXXXXXXXXXXXXXX	99,999.99	99,999.99	99,999.99

99	XXXXXXXXXXXXXXXXXXXX	99,999.99	99,999.99	99,999.99
TOTALES	999 EMPLEADOS	999,999.99	999,999.99	999,999.99

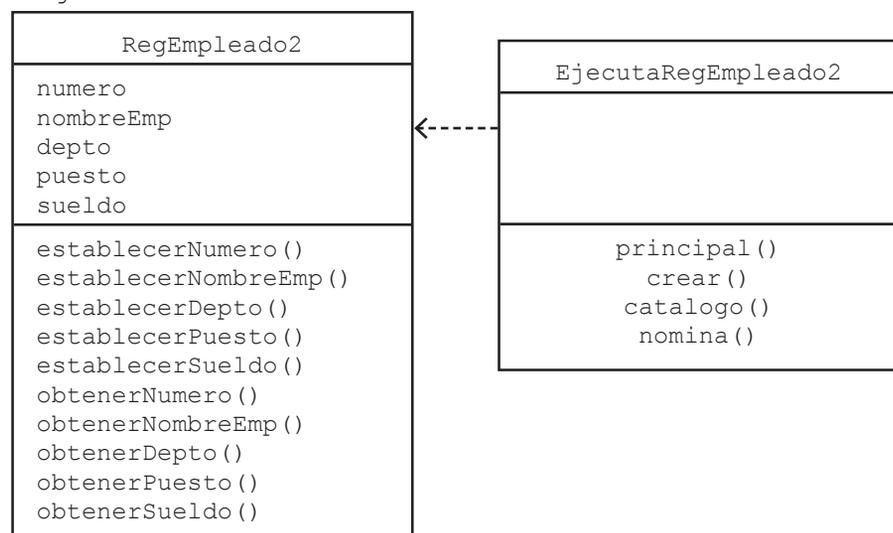
El impuesto es el 5 % sobre el excedente del salario mínimo.

Ejemplo:

A continuación se elabora un algoritmo que permite procesar el archivo de empleados esquematizado al principio de este capítulo, con organización secuencial, en el cual se aplican los procesos de creación del archivo de empleados y emisión de los reportes de catálogo de empleados y de nómina quincenal.

Se presenta el algoritmo de la solución en dos pasos: primero el diagrama de clases y después el algoritmo en pseudocódigo:

Diagrama de clases



```
Algoritmo SISTEMA DE NOMINA SECUENCIAL
Clase RegEmpleado2
1. Declarar datos
   numero: Entero
   nombreEmp: Cadena
   depto : Entero
   puesto: Entero
   sueldo: Real

2. Método establecerNumero(num: Entero)
   a. numero = num
   b. Fin Método establecerNumero

3. Método establecerNombreEmp(nom: Cadena)
   a. nombreEmp = nom
   b. Fin Método establecerNombreEmp

4. Método establecerDepto(dep: Entero)
   a. depto = dep
   b. Fin Método establecerDepto

5. Método establecerPuesto(pue: Entero)
   a. puesto = pue
   b. Fin Método establecerPuesto

6. Método establecerSueldo(sue: Real)
   a. sueldo = sue
   b. Fin Método establecerSueldo

7. Método obtenerNumero(): Entero
   a. return numero
   b. Fin Método obtenerNumero

8. Método obtenerNombreEmp(): Cadena
   a. return nombreEmp
   b. Fin Método obtenerNombreEmp

9. Método obtenerDepto(): Entero
   a. return depto
   b. Fin Método obtenerDepto

10. Método obtenerPuesto(): Entero
   a. return puesto
   b. Fin Método obtenerPuesto

11. Método obtenerSueldo(): Real
   a. return sueldo
   b. Fin Método obtenerSueldo
Fin Clase RegEmpleado2
```

Clase EjecutaRegEmpleado2

1. Método principal()

a. Declarar variables

opcion: Entero

b. do

1. Imprimir MENU

SISTEMA DE NOMINA
1. CREAR ARCHIVO 2. CATALOGO EMPLEADOS 3. NOMINA QUINCENAL 4. FIN
ESCOGER OPCIÓN

2. Leer opcion

3. switch opcion

1: crear()

2: catalogo()

3: nomina()

4. endswitch

c. while opcion != 4

d. Fin Método principal

2. Método crear()

a. Declarar variables

numEmp: Entero

nomEmp: Cadena

depEmp: Entero

pueEmp: Entero

sueEmp: Real

desea : Carácter

b. Crear (empleados, "C:/arEmp1.dat", Secuencial)

c. do

1. Declarar, crear e iniciar objeto

RegEmpleado2 objEmpleado = new EmpleadoReg2()

2. Imprimir PANTALLA de captura

SISTEMA DE NOMINA
NUMERO: NOMBRE: DEPARTAMENTO: PUESTO: SUELDO:
¿OTRA ALTA (S/N)? :

3. Leer numEmp, nomEmp, depEmp, pueEmp, sueEmp

4. while Longitud(nomEmp) < 30

a. nomEmp = nomEmp + " "

```

5. endwhile
6. Establecer
    objEmpleado.establecerNumero(numEmp)
    objEmpleado.establecerNombreEmp(nomEmp)
    objEmpleado.establecerDepto(depEmp)
    objEmpleado.establecerPuesto(pueEmp)
    objEmpleado.establecerSueldo(sueEmp)
7. Imprimir(empleados, objEmpleado)
8. Leer desea
d. while desea == 'S'
e. numEmp = 0
   nomEmp = "FIN"
   depEmp = 0
   pueEmp = 0
   sueEmp = 0
f. Declarar, crear e iniciar objeto
   RegEmpleado2 objEmpleado = new RegEmpleado2()
g. Establecer
   objEmpleado.establecerNumero(numEmp)
   objEmpleado.establecerNombreEmp(nomEmp)
   objEmpleado.establecerDepto(depEmp)
   objEmpleado.establecerPuesto(pueEmp)
   objEmpleado.establecerSueldo(sueEmp)
h. Imprimir(empleados, objEmpleado)
i. Cerrar(empleados)
j. Fin Método crear

3. Método catalogo()
a. Declarar variables
   totEmpleados: Entero
   totSueldos: Real
b. Abrir(empleados, "C:/arEmp1.dat", Secuencial)
c. Imprimir encabezado
d. totEmpleados = 0
   totSueldos = 0
e. Declarar, crear e iniciar objeto
   RegEmpleado2 objEmpleado = new RegEmpleado2()
f. Leer(empleados, objEmpleado)
g. while objEmpleado.obtenerNombreEmp() != "FIN"
   1. Imprimir
      objEmpleado.obtenerNumero()
      objEmpleado.obtenerNombreEmp()
      objEmpleado.obtenerDepto()
      objEmpleado.obtenerPuesto()
      objEmpleado.obtenerSueldo()
   2. totEmpleados = totEmpleados + 1
   3. totSueldos = totSueldos + objEmpleado.obtenerSueldo()
   4. Crear e iniciar objeto
      objEmpleado = new RegEmpleado2()
   5. Leer(empleados, objEmpleado)

```

```

        h. endwhile
        i. Imprimir totEmpleados, totSueldos
        j. Cerrar (empleados)
        k. Fin Método catalogo

4. Método nomina()
  a. Declarar variables
      totEmpleados: Entero
      bruto, impuesto, neto, totBruto, totImpuesto,
      totNeto, salMin: Real
  b. Abrir (empleados, "C:/arEmpl.dat", Secuencial)
  c. Solicitar salario mínimo quincenal
  d. Leer salMin
  e. Imprimir encabezado
  f. totEmpleados = 0; totBruto = 0; totImpuesto = 0;
      totNeto=0
  g. Declarar, crear e iniciar objeto
      RegEmpleado2 objEmpleado = new RegEmpleado2()
  h. Leer (empleados, objEmpleado)
  i. while objEmpleado.obtenerNombreEmp() != "FIN"
      1. bruto = objEmpleado.obtenerSueldo() / 2
      2. if bruto > salMin then
          a. impuesto = (bruto - salMin) * 0.05
      3. else
          a. impuesto = 0
      4. endif
      5. neto = bruto - impuesto
      6. Imprimir objEmpleado.obtenerNumero()
          objEmpleado.obtenerNombreEmp()
          bruto, impuesto, Neto
      7. totEmpleados = totEmpleados + 1
          totBruto = totBruto + bruto
          totImpuesto = totImpuesto + impuesto
          totNeto = totNeto + neto
      8. Crear e iniciar objeto
          objEmpleado = new EmpleadoReg2()
      9. Leer (empleados, objEmpleado)
  j. endwhile
  k. Imprimir totEmpleados, totBruto, totImpuesto, totNeto
  l. Cerrar (empleados)
  m. Fin Método nomina
Fin Clase EjecutaRegEmpleado2
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: RegEmpleado2.java y EjecutaRegEmpleado2.java

Explicación:

El algoritmo tiene dos clases: la Clase RegEmpleado2 y la Clase EjecutaRegEmpleado2.

La Clase `RegEmpleado2` representa un objeto (registro) de empleado y cada vez que se va a procesar un objeto empleado se creará utilizándola. En esta clase:

1. Se declaran los datos que representan la estructura de la clase:
 - `numero` para el dato número del empleado.
 - `nombreEmp` para el nombre del empleado.
 - `depto` para el departamento del empleado.
 - `puesto` para el puesto del empleado.
 - `sueldo` para el sueldo mensual del empleado.
 2. Método `establecerNumero(num: Entero)`.
Recibe en el parámetro `num` el valor que luego coloca en el dato `numero`.
 3. Método `establecerNombreEmp(nom: Cadena)`.
Recibe en el parámetro `nom` el valor que luego coloca en el dato `nombreEmp`.
 4. Método `establecerDepto(dep: Entero)`.
Recibe en el parámetro `dep` el valor que luego coloca en el dato `depto`.
 5. Método `establecerPuesto(pue: Entero)`.
Recibe en el parámetro `pue` el valor que luego coloca en el dato `puesto`.
 6. Método `establecerSueldo(sue: Real)`.
Recibe en el parámetro `sue` el valor que luego coloca en el dato `sueldo`.
 7. Método `obtenerNumero(): Entero`.
Retorna `numero`.
 8. Método `obtenerNombreEmp() Cadena`.
Retorna `nombreEmp` (nombre del empleado).
 9. Método `obtenerDepto(): Entero`.
Retorna `depto`.
 10. Método `obtenerPuesto(): Entero`.
Retorna `puesto`.
 11. Método `obtenerSueldo() Real`.
Retorna `sueldo`.
Observe que para cada dato se tiene un método setter y un getter, para establecer y acceder el valor del dato respectivamente.
- Fin de la Clase `RegEmpleado2`.

En la Clase `EjecutaRegEmpleado2`:

Se procesa el archivo de empleados y, en su momento, se van a generar objetos utilizando la Clase `RegEmpleado2`.

En esta clase se tienen los métodos: `principal()`, `crear()`, `catalogo()` y `nomina()`.

Nota: En el diagrama de clases, la Clase `EjecutaRegEmpleado2` se está esquematizando de forma diferente de como lo estuvimos haciendo en los capítulos anteriores porque dicha clase ahora, además del método `principal`, tiene otros métodos y es necesario indicarlos en el diagrama. Observe que la parte de datos o atributos queda vacía porque no es necesario indicar las variables requeridas en esta clase en el diagrama debido a que ésta es la clase controladora y no se usa para crear objetos.



En el diagrama de clases, la Clase `EjecutaRegEmpleado2` se está esquematizando de forma diferente de como lo estuvimos haciendo en los capítulos anteriores porque dicha clase ahora, además del método `principal`, tiene otros métodos y es necesario indicarlos en el diagrama. Observe que la parte de datos o atributos queda vacía porque no es necesario indicar las variables requeridas en esta clase en el diagrama debido a que ésta es la clase controladora y no se usa para crear objetos.

1. En el Método `principal()`:
 - a. Se declara la variable `opcion` para leer la opción que se escoja.
 - b. Inicia ciclo `do`:
 1. Imprimir el menú de opciones donde se solicita la opción.
 2. Se lee la respuesta en `opcion`.
 3. Inicia `switch` `opcion`:
 - Si `opcion` es 1, entonces: Llama al Método `crear()`.
 - Si `opcion` es 2, entonces: Llama al Método `catalogo()`.
 - Si `opcion` es 3, entonces: Llama al Método `nomina()`.
 4. Fin del `switch`.
 - c. Fin ciclo `do...while`. Si `opcion != 4` va al `do`; si no, se sale del ciclo `do...while`.
 - d. Fin Método `principal`.

2. En el Método `crear()`:
 - a. Se declaran las variables:
 - `numEmp` para leer el número del empleado.
 - `nomEmp` para leer el nombre del empleado.
 - `depEmp` para leer el departamento del empleado.
 - `pueEmp` para leer el puesto del empleado.
 - `sueEmp` para leer el sueldo mensual del empleado.
 - `desea` para controlar el ciclo repetitivo.
 - b. Se crea el archivo `empleados`, que tiene el nombre físico en el disco “C:/arEmp1.dat”, con organización secuencial.
 - c. Inicia ciclo `do`:
 1. Se declara el objeto `objEmpleado`, usando como base a la Clase `RegEmpleado2`. Dicho objeto se crea e inicializa mediante el constructor por defecto `RegEmpleado2()`.
Observe que cada vez que entra al ciclo crea una nueva instancia de la Clase `RegEmpleado2`, en la cual se representa y maneja como un objeto (`objEmpleado`) un registro de un empleado.
 2. Se solicitan los datos del empleado: número, nombre, departamento, puesto y sueldo mensual.
 3. Se leen en `numEmp`, `nomEmp`, `depEmp`, `pueEmp`, `sueEmp`.
 4. `while` se revisa si el nombre leído en `nomEmp` no tiene 30 caracteres. Si es así, entra al ciclo; si no, se sale del ciclo.
 - a. Se rellena con un espacio cada carácter que le falte para tener los 30 ocupados.
 5. `endwhile`.

Nota: Esto se hace porque no todos los nombres tienen la misma cantidad de caracteres y se está estableciendo ocupar 30 caracteres. Los que no quepan se abrevian y los que queden cortos (menos de 30) se rellenan con espacios para usar los 30.

 6. Se llama al Método `establecerNumero(numEmp)` del objeto `objEmpleado` para colocar el valor de `numEmp` en el dato `numero`.
Se llama al Método `establecerNombreEmp(nomEmp)` del objeto `objEmpleado` para colocar el valor de `nomEmp` en el dato `nombreEmp`.



Esto se hace porque no todos los nombres tienen la misma cantidad de caracteres y se está estableciendo ocupar 30 caracteres. Los que no quepan se abrevian y los que queden cortos (menos de 30) se rellenan con espacios para usar los 30.

- Se llama al Método establecerDepto (depEmp) del objeto objEmpleado para colocar el valor de depEmp en el dato depto.
- Se llama al Método establecerPuesto (pueEmp) del objeto objEmpleado para colocar el valor de pueEmp en el dato puesto.
- Se llama al Método establecerSueldo (sueEmp) del objeto objEmpleado para colocar el valor de sueEmp en el dato sueldo.
7. Se imprime en el archivo empleados el objeto objEmpleado, el cual contiene el registro con los datos del empleado que se está procesando.
 - d. Fin del ciclo (while desea == 'S'). Si se cumple regresa al do; si no, se sale del ciclo.
 - e. En las variables en las que se leen los datos del empleado se colocan ceros en los datos numéricos y FIN en el nombre.
 - f. Se declara y genera un último objeto objEmpleado, usando como base a la Clase RegEmpleado2.
 - g. Se establecen en el objeto objEmpleado, formando un objeto que va a fungir como centinela de fin del archivo de empleados.
 - h. Se imprime en el archivo empleados el objeto objEmpleado, el cual contiene el objeto que va a fungir como centinela de fin del archivo de empleados.
 - i. Se cierra y libera de uso el archivo empleados.
 - j. Fin del Método crear.
3. En el Método catalogo():
 - a. Se declaran las variables:
 - totEmpleados para contar el total de empleados.
 - totSueldos para calcular el total de sueldos.
 - b. Se abre el archivo empleados, que tiene el nombre físico en el disco "C:/arEmp1.dat", con organización secuencial. El apuntador del archivo se coloca en el byte 0.
 - c. Se imprime el encabezado.
 - d. Se inician totEmpleados y totSueldos en 0.
 - e. Se declara el objeto objEmpleado, usando como base a la Clase RegEmpleado2. Dicho objeto se crea e inicializa mediante el constructor por defecto RegEmpleado2().
 - f. Se lee, del archivo empleados, el objeto actual que contiene el registro con los datos de un empleado (objeto) y se coloca en objEmpleado. Es decir, se lee el primer objeto (registro).
 - g. Inicia ciclo while objEmpleado.obtenerNombreEmp() != "FIN". Se pregunta si no se ha encontrado el fin del archivo. Si es así, entra al ciclo; si no, se sale del ciclo.
 1. Imprime los datos del objeto objEmpleado, accediéndolos al llamar a los métodos: obtenerNumero(), obtenerNombreEmp(), obtenerDepto(), obtenerPuesto(), obtenerSueldo().
 2. Incrementa totEmpleados en 1.
 3. Incrementa totSueldos con objEmpleado.obtenerSueldo(). Se llama al Método obtenerSueldo() del objeto objEmpleado para acceder al valor del dato sueldo e incrementarlo en totSueldos.

4. Se crea un nuevo objeto `objEmpleado`, usando como base a la Clase `RegEmpleado2`.
 5. Se lee, del archivo `empleados`, el objeto actual que contiene el registro con los datos del siguiente empleado (objeto) y se coloca en `objEmpleado`. Es decir, se lee el siguiente objeto (registro).
 - h. Fin del ciclo `while`.
 - i. Imprime los totales `totEmpleados`, `totSueldos`.
 - j. Se cierra y libera de uso el archivo `empleados`.
 - k. Fin del Método `catalogo`.
4. En el Método `nomina()`:
- a. Se declaran las variables:
 - `totEmpleados` para contar el total de empleados.
 - `bruto` para calcular el sueldo bruto quincenal.
 - `impuesto` para calcular el impuesto que se va a descontar.
 - `neto` para calcular el neto que se va a pagar.
 - `totBruto` para calcular el total de sueldo bruto.
 - `totImpuesto` para calcular el total de impuesto.
 - `totNeto` para calcular el total de neto.
 - `salMin` para leer el salario mínimo quincenal.
 - b. Se abre el archivo `empleados`, que tiene el nombre físico en el disco "C:/arEmp1.dat", con organización secuencial. El apuntador del archivo se coloca en el byte 0.
 - c. Se solicita el salario mínimo quincenal.
 - d. Se lee en `salMin`.
 - e. Se imprime el encabezado.
 - f. Se inician `totEmpleados`, `totBruto`, `totImpuesto` y `totNeto` en 0.
 - g. Se declara el objeto `objEmpleado`, usando como base a la Clase `RegEmpleado2`. Dicho objeto se crea e inicializa mediante el constructor por defecto `RegEmpleado2()`.
 - h. Se lee, del archivo `empleados`, el objeto actual que contiene el registro con los datos de un empleado (objeto) y se coloca en `objEmpleado`. Es decir, se lee el primer objeto(registro).
 - i. Inicia ciclo `while objEmpleado.obtenerNombreEmp() != "FIN"`. Se pregunta si no se ha encontrado el fin del archivo. Si es así, entra al ciclo; si no, se sale del ciclo.
 1. Calcula `bruto = objEmpleado.obtenerSueldo() / 2`.
 2. Si `bruto > salMin` entonces:
 - a. Calcula `impuesto = (bruto - salMin) * 0.05`.
 3. Si no (else):
 - a. Calcula `impuesto = 0`.
 4. Fin del `if`.
 5. Calcula `neto = bruto - impuesto`.
 6. Imprime los datos del objeto `objEmpleado`, accediéndolos al llamar a los métodos: `obtenerNumero()`, `obtenerNombreEmp()` y a los datos calculados: `bruto`, `impuesto` y `neto`.

7. Incrementa `totEmpleados` con 1, `totBruto` con bruto, `totImpuesto` con impuesto y `totNeto` con neto.
8. Se crea un nuevo objeto `objEmpleado`, usando como base a la Clase `RegEmpleado2`.
9. Se lee, del archivo `empleados`, el objeto actual que contiene el registro con los datos del siguiente empleado (objeto) y se coloca en `objEmpleado`. Es decir, se lee el siguiente objeto (registro).
- j. Fin del ciclo `while`.
- k. Imprime los totales `totEmpleados`, `totBruto`, `totImpuesto`, `totNeto`.
- l. Se cierra y libera de uso el archivo `empleados`.
- m. Fin del Método `nomina`.

Fin de la Clase `EjecutaRegEmpleado2`.

Fin del algoritmo.

Emisión de reportes con cortes de control

Si suponemos que los registros del archivo están ordenados por departamento, es posible obtener reportes agrupados por departamento, es decir, que todos los empleados del primer departamento aparezcan juntos, al inicio, con totales por departamento; después, los del siguiente departamento y así sucesivamente hasta llegar al final, donde habrá un gran total que agrupe a todos los departamentos.

Ejemplo:

Supongamos que ya tenemos el archivo de empleados, con organización secuencial y su nombre físico en el disco "C:/arEmp1.dat", y contiene los objetos (registros) siguientes:

Número	Nombre	Depto.	Puesto	Sueldo
1	MARICELA VILLA	1	1	1200
2	EDILIA CAMACHO	1	2	1450
3	DANIA CAMACHO	1	3	1750
4	JUAN DE DIOS LOPEZ	2	1	1450
5	MANUEL SAIZ	2	2	2300
6	MANUEL MEZA	3	1	3000
7	ARACELI LOPEZ	3	2	1500

Emitir el catálogo de empleados con cortes de control nos arrojaría el siguiente reporte:

CATALOGO DE EMPLEADOS				
NUM.	NOMBRE	DEPTO.	PUESTO	SUELDO
1	MARICELA VILLA	1	1	1200.00
2	EDILIA CAMACHO	1	2	1450.00
3	DANIA CAMACHO	1	3	1750.00
TOTAL DEPTO. 1	3 EMPLEADOS			4400.00
4	JUAN DE DIOS LOPEZ	2	1	1450.00
5	MANUEL SAIZ	2	2	2300.00
TOTAL DEPTO. 2	2 EMPLEADOS			3750.00
6	MANUEL MEZA	3	1	3000.00
7	ARACELI LOPEZ	3	2	1500.00
TOTAL DEPTO. 3	2 EMPLEADOS			4500.00
TOTAL GENERAL	7 EMPLEADOS			12650.00

Los datos se imprimen agrupados por departamento. Por cada departamento se requieren dos datos totalizadores: total de empleados y total de sueldos. Al final se requiere un total general con los mismos datos que por departamento, sólo que agrupando a todos los departamentos.

A continuación se elabora un método que emite el catálogo de empleados con cortes de control:

```

6. Método catalogoConCortes()
  a. Declarar variables
      totEmpleados, totEmpDep, deptoProceso: Entero
      totSueldos, totSueldosDep: Real
  b. Abrir (empleados, "C:/arEmpl.dat", Secuencial)
  c. Imprimir encabezado
  d. totEmpleados = 0
      totSueldos = 0
  e. Declarar, crear e iniciar objeto
      RegEmpleado2 objEmpleado = new RegEmpleado2()
  f. Leer (empleados, objEmpleado)
  g. while objEmpleado.obtenerNombreEmp() != "FIN"
      1. totEmpDep = 0 ; totSueldosDep = 0
      2. deptoProceso = objEmpleado.obtenerDepto()
      3. while objEmpleado.obtenerDepto() == deptoProceso
          a. Imprimir
              objEmpleado.obtenerNumero()
              objEmpleado.obtenerNombreEmp()
              objEmpleado.obtenerDepto()
              objEmpleado.obtenerPuesto()
              objEmpleado.obtenerSueldo()
          b. totEmpDep = totEmpDep + 1
          c. totSueldosDep = totSueldosDep +
              objEmpleado.obtenerSueldo()
          d. Crear e iniciar objeto
              objEmpleado = new RegEmpleado2()
          e. Leer (empleados, objEmpleado)

```

```

4. endwhile
5. Imprimir totEmpDep, totSueldosDep
6. totEmpleados = totEmpleados + totEmpDep
   totSueldos = totSueldos + totSueldosDep
h. endwhile
i. Imprimir totEmpleados, totSueldos
j. Cerrar (empleados)
k. Fin Método catalogoConCortes

```

Nota: Este método que emite el catálogo con cortes se le puede agregar al algoritmo anterior de este punto.

Explicación:

6. En el Método `catalogoConCortes()`:
 - a. Se declaran las variables:
 - `totEmpleados` para contar el total de empleados global
 - `totEmpDep` para contar el total de empleados por departamento
 - `deptoProceso` para controlar el departamento que se esta procesando
 - `totSueldos` para calcular el total de sueldos global
 - `totSueldosDep` para calcular el total de sueldos por departamento
 - b. Se abre el archivo `empleados`, que tiene el nombre físico en el disco "C:/arEmp1.dat", con organización secuencial. El apuntador del archivo se coloca en el byte 0.
 - c. Imprime el encabezado.
 - d. Se inician `totEmpleados` y `totSueldos` en 0.
 - e. Se declara el objeto `objEmpleado`, usando como base a la Clase `RegEmpleado2`. Dicho objeto se crea e inicializa mediante el constructor por defecto `RegEmpleado2()`.
 - f. Se lee, del archivo `empleados`, el objeto actual que contiene el registro con los datos de un empleado (objeto) y se coloca en `objEmpleado`. Es decir, se lee el primer objeto (registro).
 - g. Inicia ciclo `while objEmpleado.obtenerNombreEmp() != "FIN"`. Se pregunta si no se ha encontrado el fin del archivo. Si es así, entra al ciclo; si no, se sale del ciclo.
 1. Inicia `totEmpDep` y `totSueldosDep` en 0.
 2. Coloca `objEmpleado.obtenerDepto()` en `deptoProceso`.
 3. Inicia ciclo `while objEmpleado.obtenerDepto() == deptoProceso`
Se pregunta si el departamento en proceso es igual al departamento que trae el empleado leído. Si es así, entra al ciclo; si no, se sale del ciclo.
 - a. Imprime los datos del objeto `objEmpleado`, accediéndolos al llamar a los métodos: `obtenerNumero()`, `obtenerNombreEmp()`, `obtenerDepto()`, `obtenerPuesto()`, `obtenerSueldo()`.
 - b. Incrementa `totEmpleados` en 1.
 - c. Incrementa `totSueldos` con `objEmpleado.obtenerSueldo()`. Se llama al Método `obtenerSueldo()` del objeto `objEmpleado` para acceder al valor del dato sueldo e incrementarlo en `totSueldos`.



Este método que emite el catálogo con cortes se le puede agregar al algoritmo anterior de este punto.

- d. Se crea un nuevo objeto `objEmpleado`, usando como base a la Clase `RegEmpleado2`.
- e. Se lee, del archivo `empleados`, el objeto actual que contiene el registro con los datos del siguiente empleado (objeto) y se coloca en `objEmpleado`. Es decir, se lee el siguiente objeto (registro).
- 4. Fin del ciclo `while`.
- 5. Imprime `totEmpDep` y `totSueldosDep`.
- 6. Incrementa `totEmpleados` con `totEmpDep`.
Incrementa `totSueldos` con `totSueldosDep`.
- h. Fin del ciclo `while`.
- i. Imprime los totales `totEmpleados`, `totSueldos`.
- j. Se cierra y libera de uso el archivo `empleados`.
- k. Fin del Método `catalogoConCortes`.

Ejercicio:

Considerando el mismo archivo de empleados, emitir el siguiente reporte con cortes de control:

NUM.	NOMBRE	NOMINA QUINCENAL DEPTO.	S. BRUTO	IMPUESTO	S. NETO
1	MARICELA VILLA	1	9999.99	9999.99	9999.99
2	EDILIA CAMACHO	1	9999.99	9999.99	9999.99
3	DANIA CAMACHO	1	9999.99	9999.99	9999.99
TOTAL DEPTO. 1	3 EMPLEADOS		9999.99	9999.99	9999.99
4	JUAN DE DIOS LOPEZ	2	9999.99	9999.99	9999.99
5	MANUEL SAIZ	2	9999.99	9999.99	9999.99
TOTAL DEPTO. 2	2 EMPLEADOS		9999.99	9999.99	9999.99
6	MANUEL MEZA	3	9999.99	9999.99	9999.99
7	ARACELI LOPEZ	3	9999.99	9999.99	9999.99
TOTAL DEPTO. 3	2 EMPLEADOS		9999.99	9999.99	9999.99
TOTAL GENERAL	7 EMPLEADOS		9999.99	9999.99	9999.99

S. BRUTO es el sueldo que trae en el archivo dividido entre 2, porque en el mes hay dos quincenas.

IMPUESTO es el 5% sobre el excedente del bruto sobre el salario mínimo quincenal.

S. NETO es el S. BRUTO menos el IMPUESTO.

Además, se requieren totales por departamento y un total general de los datos: total de empleados, total de sueldo bruto, total de impuesto y total de sueldo neto.

A continuación se elabora un método que emite la nómina quincenal con cortes de control:

```

7. Método nominaConCortes()
a. Declarar variables
totEmpleados, totEmpDep, deptoProceso: Entero
bruto, impuesto, neto, totBruto, totImpuesto,
totNeto, totBrutoDep, totImpuestoDep, totNetoDep,
salMin: Real

```

```

b. Abrir (empleados, "C:/arEmp1.dat", Secuencial)
c. Solicitar salario mínimo quincenal
d. Leer salMin
e. Imprimir encabezado
f. totEmpleados = 0; totBruto = 0; totImpuesto = 0;
   totNeto=0
g. Declarar, crear e iniciar objeto
   RegEmpleado2 objEmpleado = new RegEmpleado2()
h. Leer (empleados, objEmpleado)
i. while objEmpleado.obtenerNombreEmp() != "FIN"
    1. totEmpDep = 0 ; totBrutoDep = 0
       totImpuestoDep = 0; totNetoDep = 0
    2. deptoProceso = objEmpleado.obtenerDepto()
    3. while objEmpleado.obtenerDepto() == deptoProceso
       a. bruto = objEmpleado.obtenerSueldo() / 2
       b. if bruto > salMin then
          1. impuesto = (bruto - salMin) * 0.05
       c. else
          1. impuesto = 0
       d. endif
       e. neto = bruto - impuesto
       f. Imprimir objEmpleado.obtenerNumero()
          objEmpleado.obtenerNombreEmp()
          objEmpleado.obtenerDepto()
          bruto, impuesto, Neto
       g. totEmpDep = totEmpDep + 1
          totBrutoDep = totBrutoDep + bruto
          totImpuestoDep = totImpuestoDep + impuesto
          totNetoDep = totNetoDep + neto
       h. Crear e iniciar objeto
          objEmpleado = new RegEmpleado2()
       i. Leer (empleados, objEmpleado)
    4. endwhile
    5. Imprimir totEmpDep, totBrutoDep,
       totImpuestoDep, totNetoDep
    6. totEmpleados = totEmpleados + totEmpDep
       totBruto = totBruto + totBrutoDep
       totImpuesto = totImpuesto + totImpuestoDep
       totNeto = totNeto + totNetoDep
j. endwhile
k. Imprimir totEmpleados, totBruto, totImpuesto, totNeto
l. Cerrar (empleados)
m. Fin Método nominaConCortes

```



Este método que emite la nómina quincenal con cortes se le puede agregar al algoritmo anterior de este punto.

Nota: Este método que emite la nómina quincenal con cortes se le puede agregar al algoritmo anterior de este punto.

Explicación:

7. En el Método `nominaConCortes()`:
 - a. Se declaran las variables:
 - `totEmpleados` para contar el total de empleados global.
 - `totEmpDep` para contar el total de empleados por departamento.
 - `deptoProceso` para controlar el departamento que se esta procesando.
 - `bruto` para calcular el sueldo quincenal.
 - `impuesto` para calcular el impuesto.
 - `neto` para calcular el neto.
 - `totBruto` para el total de sueldo bruto global.
 - `totImpuesto` para el total de impuesto global.
 - `totNeto` para el total de sueldo neto global.
 - `totBrutoDep` para el total de sueldo bruto por departamento.
 - `totImpuestoDep` para el total de impuesto por departamento.
 - `totNetoDep` para el total de sueldo neto por departamento.
 - `salMin` para leer el salario mínimo quincenal.
 - b. Se abre el archivo `empleados`, que tiene el nombre físico en el disco "C:/arEmp1.dat", con organización secuencial. El apuntador del archivo se coloca en el byte 0.
 - c. Se solicita el salario mínimo quincenal.
 - d. Se lee en `salMin`.
 - e. Imprime el encabezado.
 - f. Se inician `totEmpleados`, `totBruto`, `totImpuesto` y `totNeto` en 0.
 - g. Se declara el objeto `objEmpleado`, usando como base a la Clase `RegEmpleado2`. Dicho objeto se crea e inicializa mediante el constructor por defecto `RegEmpleado2()`.
 - h. Se lee, del archivo `empleados`, el objeto actual que contiene el registro con los datos de un empleado (objeto) y se coloca en `objEmpleado`. Es decir, se lee el primer objeto (registro).
 - i. Inicia ciclo `while objEmpleado.obtenerNombreEmp() != "FIN"`. Se pregunta si no se ha encontrado el fin del archivo. Si es así, entra al ciclo; si no, se sale del ciclo.
 1. Inicia `totEmpDep`, `totBrutoDep`, `totImpuestoDep` y `totNetoDep` en 0.
 2. Coloca `objEmpleado.obtenerDepto()` en `deptoProceso`.
 3. Inicia ciclo `while objEmpleado.obtenerDepto() == deptoProceso`. Se pregunta si el departamento en proceso es igual al departamento que trae el empleado leído. Si es así, entra al ciclo; si no, se sale del ciclo.
 - a. Calcula `bruto = objEmpleado.obtenerSueldo() / 2`.
 - b. Si `bruto > salMin` entonces:
 1. Calcula `impuesto = (bruto - salMin) * 0.05`.
 - c. Si no (else):
 1. Calcula `impuesto = 0`.
 - d. Fin del if.
 - e. Calcula `neto = bruto - impuesto`.
 - f. Imprime los datos del objeto `objEmpleado`, accediéndolos al llamar a los métodos: `obtenerNumero()`, `obtenerNombreEmp()`, `obtenerDepto()` y los datos calculados: `bruto`, `impuesto` y `neto`.

- g. Incrementa `totEmpDep` con 1, `totBrutoDep` con `bruto`, `totImpuestoDep` con `impuesto` y `totNetoDep` con `neto`.
 - h. Se crea un nuevo objeto `objEmpleado`, usando como base a la Clase `RegEmpleado2`
 - i. Se lee, del archivo `empleados`, el objeto actual que contiene el registro con los datos del siguiente empleado (objeto) y se coloca en `objEmpleado`. Es decir, se lee el siguiente objeto (registro).
4. Fin del ciclo `while`.
 5. Imprime `totEmpDep`, `totBrutoDep`, `totImpuestoDep` y `totNetoDep`.
 6. Incrementa `totEmpleados` con `totEmpDep`, `totBruto` con `totBrutoDep`, `totImpuesto` con `totImpuestoDep` y `totNeto` con `totNetoDep`.
- j. Fin del ciclo `while`.
 - k. Imprime los totales `totEmpleados`, `totBruto`, `totImpuesto` y `totNeto`.
 - l. Se cierra y libera de uso el archivo `empleados`.
 - m. Fin del Método `nominaConCortes`.

15.5 Proceso de un archivo directo

Cuando se va a utilizar un archivo con organización directa, es aconsejable crear los registros del archivo con ceros y nulos y, al final, se coloca una marca o centinela de fin de archivo. Después de esto se pueden hacer altas, bajas y cambios de forma directa. En caso de ser necesario, el archivo puede agrandarse, agregándole más registros (objetos) en ceros y nulos, recorriendo la marca o centinela de fin del archivo.

Creación

Esta operación, como su nombre lo indica, nos permite crear el archivo. A continuación se apartan lugares para los registros (objetos), grabando ceros en los datos numéricos y nulos en los datos cadena de caracteres. Al final del archivo se crea un último registro que funge como centinela o indicador de fin del archivo.

Expansión

Esta operación, como su nombre lo indica, nos permite expandir el archivo. Enseguida del último registro (objeto), se apartan nuevos lugares para los registros (objetos) que se agregarán, grabando ceros en los datos numéricos y nulos en los datos cadena de caracteres. Al final del archivo se crea un último registro que funge como centinela o indicador de fin del archivo.

Altas

Las altas al archivo se harán de la siguiente forma: el empleado (objeto) número 1 se grabará a partir del byte 0, ocupando cierta cantidad de bytes, de acuerdo al tamaño del registro; el empleado 2 se grabará a partir del siguiente byte, después de donde se terminó de grabar el anterior registro, y así sucesivamente.

Bajas

Para dar de baja a un empleado, se teclea su número, se lee el registro (objeto), se despliegan (imprimen) los datos y se pregunta si está seguro de darlos de baja. Si es así, se graba en su lugar un registro (objeto) con ceros y nulos.

Cambios

Para realizar cambios en algunos de los datos de cada registro (objeto) es necesario indicar el número del empleado. Se lee el registro (objeto) correspondiente, se despliegan los datos y se realizan los cambios.

Consultas

Para consultar los datos de un empleado (objeto) se solicita el número, se lee, y luego se busca el registro (objeto) de ese empleado en el archivo; después se leen y se imprimen o despliegan los datos.

Obtención de reportes

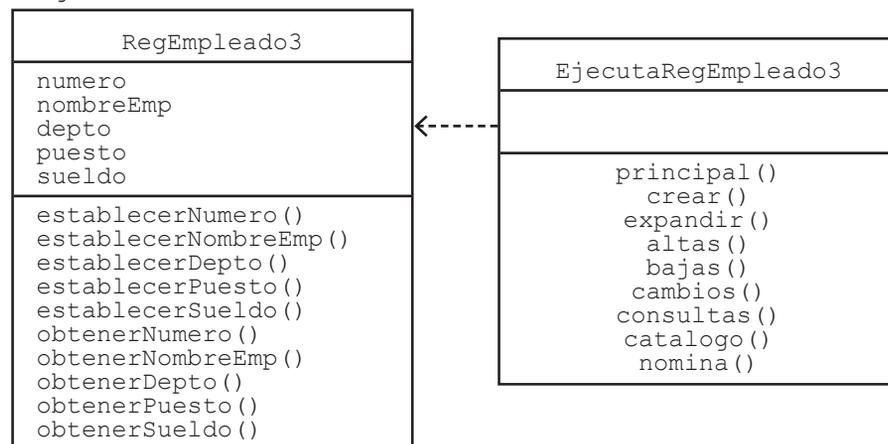
Un archivo al que ya se le dieron de alta registros (objetos) puede ser utilizado para emitir información en forma de reportes o listados de sus datos. A partir de un archivo directo, el procedimiento es el siguiente: abrir el archivo, leer el primer registro (objeto), hacer los cálculos necesarios y enlistarlo, enseguida leer el siguiente registro (objeto), hacer cálculos y enlistarlo, y así sucesivamente hasta encontrar el fin del archivo.

Ejemplo:

A continuación se elabora un algoritmo similar al del punto anterior, sólo que ahora permite procesar un archivo con organización directa para aplicar los conceptos antes descritos. En el método principal se ofrece un menú con las opciones: crear el archivo, expandir el archivo, hacerle altas, bajas, cambios, consultas al archivo, emitir el reporte catálogo de empleados y emitir el reporte nómina quincenal.

A continuación se tiene el algoritmo de la solución:

Diagrama de clases



```
Algoritmo SISTEMA DE NOMINA DIRECTO
Clase RegEmpleado3
1. Declarar datos
    numero: Entero
    nombreEmp: Cadena
    depto : Entero
    puesto: Entero
    sueldo: Real

2. Método establecerNumero(num: Entero)
    a. numero = num
    b. Fin Método establecerNumero

3. Método establecerNombreEmp(nom: Cadena)
    a. nombreEmp = nom
    b. Fin Método establecerNombreEmp

4. Método establecerDepto(dep: Entero)
    a. depto = dep
    b. Fin Método establecerDepto

5. Método establecerPuesto(pue: Entero)
    a. puesto = pue
    b. Fin Método establecerPuesto

6. Método establecerSueldo(sue: Real)
    a. sueldo = sue
    b. Fin Método establecerSueldo

7. Método obtenerNumero(): Entero
    a. return numero
    b. Fin Método obtenerNumero

8. Método obtenerNombreEmp(): Cadena
    a. return nombreEmp
    b. Fin Método obtenerNombreEmp

9. Método obtenerDepto(): Entero
    a. return depto
    b. Fin Método obtenerDepto

10. Método obtenerPuesto(): Entero
    a. return puesto
    b. Fin Método obtenerPuesto

11. Método obtenerSueldo(): Real
    a. return sueldo
    b. Fin Método obtenerSueldo
Fin Clase RegEmpleado3
```

Clase EjecutaRegEmpleado3

1. Método principal()

a. Declarar variables
opcion: Entero

b. do

1. Imprimir MENU

SISTEMA DE NOMINA
1. CREAR ARCHIVO 2. EXPANDIR ARCHIVO 3. ALTAS 4. BAJAS 5. CAMBIOS 6. CONSULTAS 7. CATALOGO EMPLEADOS 8. NOMINA QUINCENAL 9. FIN
ESCOGER OPCIÓN

2. Leer opcion

3. switch opcion

1: crear()

2: expandir()

3: altas()

4: bajas()

5: cambios()

6: consultas()

7: catalogoConCortes()

8: nominaConCortes()

4. endswitch

c. while opcion != 9

d. Fin Método principal

2. Método crear()

a. Declarar variables

i, numRegs: Entero

numEmp: Entero

nomEmp: Cadena

depEmp: Entero

pueEmp: Entero

sueEmp: Real

b. Crear (empleados, "C:/arEmp2.dat", Directo, "rw")

c. Solicitar cantidad de registros que tendrá el archivo

d. Leer numRegs

e. numEmp = 0

nomEmp = "

depEmp = 0

pueEmp = 0

sueEmp = 0

```

f. Declarar, crear e iniciar objeto
   RegEmpleado3 objEmpleado = new RegEmpleado3()
g. Establecer
   objEmpleado.establecerNumero(numEmp)
   objEmpleado.establecerNombreEmp(nomEmp)
   objEmpleado.establecerDepto(depEmp)
   objEmpleado.establecerPuesto(pueEmp)
   objEmpleado.establecerSueldo(sueEmp)
h. for (i=1; i<=numRegs; i++)
   1. Imprimir (empleados, objEmpleado)
      empleados.Imprimir(objEmpleado.obtenerNumero())
      empleados.Imprimir(objEmpleado.obtenerNombreEmp())
      empleados.Imprimir(objEmpleado.obtenerDepto())
      empleados.Imprimir(objEmpleado.obtenerPuesto())
      empleados.Imprimir(objEmpleado.obtenerSueldo())
i. endfor
j. nomEmp = "FIN"
k. objEmpleado.establecerNombreEmp(nomEmp)
l. Imprimir (empleados, objEmpleado)
   empleados.Imprimir(objEmpleado.obtenerNumero())
   empleados.Imprimir(objEmpleado.obtenerNombreEmp())
   empleados.Imprimir(objEmpleado.obtenerDepto())
   empleados.Imprimir(objEmpleado.obtenerPuesto())
   empleados.Imprimir(objEmpleado.obtenerSueldo())
m. Cerrar (empleados)
n. Fin Método crear

3. Método expandir()
a. Declarar variables
   i, numRegs, apuntador: Entero
   numEmp: Entero
   nomEmp: Cadena
   depEmp: Entero
   pueEmp: Entero
   sueEmp: Real
b. Abrir (empleados, "C:/arEmp2.dat", Directo, "rw")
c. Solicitar cantidad de registros que agregará al
   archivo
d. Leer numRegs
e. Declarar, crear e iniciar objeto
   RegEmpleado3 objEmpleado = new RegEmpleado3()
f. apuntador = empleados.PosApuntador()
g. Leer (empleados, objEmpleado)
   objEmpleado.establecerNumero(empleados.LeerEntero())
   objEmpleado.establecerNombreEmp(empleados.LeerCadena())
   objEmpleado.establecerDepto(empleados.LeerEntero())
   objEmpleado.establecerPuesto(empleados.LeerEntero())
   objEmpleado.establecerSueldo(empleados.LeerReal())

```

```

h. while objEmpleado.obtenerNombreEmp() != "FIN"
    1. apuntador = empleados.PosApuntador()
    2. Leer (empleados, objEmpleado)
        objEmpleado.establecerNumero(empleados.LeerEntero())
        objEmpleado.establecerNombreEmp(empleados.LeerCadena())
        objEmpleado.establecerDepto(empleados.LeerEntero())
        objEmpleado.establecerPuesto(empleados.LeerEntero())
        objEmpleado.establecerSueldo(empleados.LeerReal())
i. endwhile
j. empleados.encontrar(apuntador)
k. numEmp = 0
   nomEmp = " "
   depEmp = 0
   pueEmp = 0
   sueEmp = 0
l. Crear e iniciar objeto
   objEmpleado = new RegEmpleado3()
m. Establecer
   objEmpleado.establecerNumero(numEmp)
   objEmpleado.establecerNombreEmp(nomEmp)
   objEmpleado.establecerDepto(depEmp)
   objEmpleado.establecerPuesto(pueEmp)
   objEmpleado.establecerSueldo(sueEmp)
n. for (i=1; i<=numRegs; i++)
    1. Imprimir (empleados, objEmpleado)
       empleados.Imprimir(objEmpleado.obtenerNumero())
       empleados.Imprimir(objEmpleado.obtenerNombreEmp())
       empleados.Imprimir(objEmpleado.obtenerDepto())
       empleados.Imprimir(objEmpleado.obtenerPuesto())
       empleados.Imprimir(objEmpleado.obtenerSueldo())
o. endfor
p. nomEmp = "FIN"
q. objEmpleado.establecerNombreEmp(nomEmp)
r. Imprimir (empleados, objEmpleado)
   empleados.Imprimir(objEmpleado.obtenerNumero())
   empleados.Imprimir(objEmpleado.obtenerNombreEmp())
   empleados.Imprimir(objEmpleado.obtenerDepto())
   empleados.Imprimir(objEmpleado.obtenerPuesto())
   empleados.Imprimir(objEmpleado.obtenerSueldo())
s. Cerrar (empleados)
t. Fin Método expandir

4. Método altas()
a. Declarar variables
   numEmp: Entero
   nomEmp: Cadena
   depEmp: Entero
   pueEmp: Entero
   sueEmp: Real
   desea : Carácter

```

- b. Abrir (empleados, "C:/arEmp2.dat", Directo, "rw")
- c. do

- 1. Declarar, crear e iniciar objeto
RegEmpleado3 objEmpleado = new RegEmpleado3()
- 2. Imprimir PANTALLA de captura

SISTEMA DE NOMINA ALTAS ARCHIVO DE EMPLEADOS
NUMERO: NOMBRE: DEPARTAMENTO: PUESTO: SUELDO:
¿OTRA BAJA (S/N)? :

- 3. Leer numEmp, nomEmp, depEmp, pueEmp, sueEmp
 - 4. while Longitud(nomEmp) < 30
 - a. nomEmp = nomEmp + " "
 - 5. endwhile
 - 6. Establecer
 - objEmpleado.establecerNumero(numEmp)
 - objEmpleado.establecerNombreEmp(nomEmp)
 - objEmpleado.establecerDepto(depEmp)
 - objEmpleado.establecerPuesto(pueEmp)
 - objEmpleado.establecerSueldo(sueEmp)
 - 7. empleados.encontrar((objEmpleado.obtenerNumero()-1)*48)
 - 8. Imprimir (empleados, objEmpleado)
 - empleados.Imprimir(objEmpleado.obtenerNumero())
 - empleados.Imprimir(objEmpleado.obtenerNombreEmp())
 - empleados.Imprimir(objEmpleado.obtenerDepto())
 - empleados.Imprimir(objEmpleado.obtenerPuesto())
 - empleados.Imprimir(objEmpleado.obtenerSueldo())
 - 9. Leer desea
 - d. while desea == 'S'
 - e. Cerrar (empleados)
 - f. Fin Método altas
5. Método bajas()
- a. Declarar variables
 - desea, seguro : Carácter
 - num: Entero
 - numEmp: Entero
 - nomEmp: Cadena
 - depEmp: Entero
 - pueEmp: Entero
 - sueEmp: Real
 - b. Abrir (empleados, "C:/arEmp2.dat", Directo, "rw")
 - c. do
 - 1. Declarar, crear e iniciar objeto
RegEmpleado3 objEmpleado = new RegEmpleado3()

2. Imprimir PANTALLA de captura

SISTEMA DE NOMINA BAJAS ARCHIVO DE EMPLEADOS
NUMERO: NOMBRE: DEPARTAMENTO: PUESTO: SUELDO:
SON LOS DATOS; ¿SEGURO (S/N) ?
¿OTRA BAJA (S/N) ? :

3. Leer num

4. empleados.encontrar((num-1)*48)

5. Leer (empleados, objEmpleado)

```
objEmpleado.establecerNumero(empleados.LeerEntero())
objEmpleado.establecerNombreEmp(empleados.LeerCadena())
objEmpleado.establecerDepto(empleados.LeerEntero())
objEmpleado.establecerPuesto(empleados.LeerEntero())
objEmpleado.establecerSueldo(empleados.LeerReal())
```

6. Imprimir

```
objEmpleado.obtenerNombreEmp()
objEmpleado.obtenerDepto()
objEmpleado.obtenerPuesto()
objEmpleado.obtenerSueldo()
```

7. Preguntar si esta seguro

8. Leer seguro

9. if seguro == 'S' then

```
a. numEmp = 0
   nomEmp = ""
   depEmp = 0
   pueEmp = 0
   sueEmp = 0
```

b. empleados.encontrar((num-1)*48)

c. Imprimir (empleados, objEmpleado)

```
empleados.Imprimir(objEmpleado.obtenerNumero())
empleados.Imprimir(objEmpleado.obtenerNombreEmp())
empleados.Imprimir(objEmpleado.obtenerDepto())
empleados.Imprimir(objEmpleado.obtenerPuesto())
empleados.Imprimir(objEmpleado.obtenerSueldo())
```

10. endif

11. Leer desea

d. while desea == 'S'

e. Cerrar (empleados)

f. Fin Método bajas

6. Método cambios()
 - a. Declarar variables
 - desea : Carácter
 - num, opcion2: Entero
 - numEmp: Entero
 - nomEmp: Cadena
 - depEmp: Entero
 - pueEmp: Entero
 - sueEmp: Real
 - b. Abrir (empleados, "C:/arEmp2.dat", Directo, "rw")
 - c. do

1. Declarar, crear e iniciar objeto
RegEmpleado3 objEmpleado = new RegEmpleado3()
2. Imprimir PANTALLA de captura

SISTEMA DE NOMINA CAMBIOS ARCHIVO DE EMPLEADOS
NUMERO: 1: NOMBRE: 2: DEPARTAMENTO: 3: PUESTO: 4: SUELDO:
DATO A CAMBIAR (1,2,3,4,0=FIN) :
¿OTRO CAMBIO (S/N)? :

3. Leer num
4. empleados.encontrar((num-1)*48)
5. Leer (empleados, objEmpleado)
 - objEmpleado.establecerNumero(empleados.LeerEntero())
 - objEmpleado.establecerNombreEmp(empleados.LeerCadena())
 - objEmpleado.establecerDepto(empleados.LeerEntero())
 - objEmpleado.establecerPuesto(empleados.LeerEntero())
 - objEmpleado.establecerSueldo(empleados.LeerReal())
6. Imprimir
 - objEmpleado.obtenerNombreEmp()
 - objEmpleado.obtenerDepto()
 - objEmpleado.obtenerPuesto()
 - objEmpleado.obtenerSueldo()
7. Leer opcion2
8. while (opcion2>0)AND(opcion2<5)
 - a. switch opcion2
 - 1: Leer nomEmp
Establecer
objEmpleado.establecerNombreEmp(nomEmp)
 - 2: Leer depEmp
Establecer
objEmpleado.establecerDepto(depEmp)

```

        3: Leer pueEmp
           Establecer
              objEmpleado.establecerPuesto(pueEmp)
        4: Leer sueEmp
           Establecer
              objEmpleado.establecerSueldo(sueEmp)
      b. endswitch
      c. Leer opcion2
9. endwhile
10. empleados.encontrar((num-1)*48)
11. Imprimir (empleados, objEmpleado)
    empleados.Imprimir(objEmpleado.obtenerNumero())
    empleados.Imprimir(objEmpleado.obtenerNombreEmp())
    empleados.Imprimir(objEmpleado.obtenerDepto())
    empleados.Imprimir(objEmpleado.obtenerPuesto())
    empleados.Imprimir(objEmpleado.obtenerSueldo())
12. Leer desea
d. while desea == 'S'
e. Cerrar (empleados)
f. Fin Método cambios
7. Método consultas()
  a. Declarar variables
     desea : Carácter
     num: Entero
     numEmp: Entero
     nomEmp: Cadena
     depEmp: Entero
     pueEmp: Entero
     sueEmp: Real
  b. Abrir (empleados, "C:/arEmp2.dat", Directo, "rw")
  c. do
      1. Declarar, crear e iniciar objeto
         RegEmpleado3 objEmpleado = new RegEmpleado3()
      2. Imprimir PANTALLA de captura

```

SISTEMA DE NOMINA CONSULTAS ARCHIVO DE EMPLEADOS
NUMERO: NOMBRE: DEPARTAMENTO: PUESTO: SUELDO:
¿OTRA CONSULTA (S/N) ? :

```

      3. Leer num
      4. empleados.encontrar((num-1)*48)

```

```

5. Leer (empleados, objEmpleado)
   objEmpleado.establecerNumero(empleados.LeerEntero())
   objEmpleado.establecerNombreEmp(empleados.LeerCadena())
   objEmpleado.establecerDepto(empleados.LeerEntero())
   objEmpleado.establecerPuesto(empleados.LeerEntero())
   objEmpleado.establecerSueldo(empleados.LeerReal())
6. Imprimir
   objEmpleado.obtenerNombreEmp()
   objEmpleado.obtenerDepto()
   objEmpleado.obtenerPuesto()
   objEmpleado.obtenerSueldo()
7. Leer desea
d. while desea == 'S'
e. Cerrar (empleados)
f. Fin Método consultas

8. Método catalogoConCortes()
a. Declarar variables
   totEmpleados, totEmpDep, deptoProceso: Entero
   totSueldos, totSueldosDep: Real
b. Abrir (empleados, "C:/arEmp2.dat", Directo, "rw")
c. totEmpleados = 0
   totSueldos = 0
d. Imprimir encabezado
e. Declarar, crear e iniciar objeto
   RegEmpleado3 objEmpleado = new RegEmpleado3()
f. Leer (empleados, objEmpleado)
   objEmpleado.establecerNumero(empleados.LeerEntero())
   objEmpleado.establecerNombreEmp(empleados.LeerCadena())
   objEmpleado.establecerDepto(empleados.LeerEntero())
   objEmpleado.establecerPuesto(empleados.LeerEntero())
   objEmpleado.establecerSueldo(empleados.LeerReal())
g. while objEmpleado.obtenerNombreEmp() != "FIN"
   1. totEmpDep = 0 ; totSueldosDep = 0
   2. deptoProceso = objEmpleado.obtenerDepto()
   3. while objEmpleado.obtenerDepto() == deptoProceso
      a. Imprimir
         objEmpleado.obtenerNumero()
         objEmpleado.obtenerNombreEmp()
         objEmpleado.obtenerDepto()
         objEmpleado.obtenerPuesto()
         objEmpleado.obtenerSueldo()
      b. totEmpDep = totEmpDep + 1
      c. totSueldosDep = totSueldosDep +
         objEmpleado.obtenerSueldo()
      d. Crear e iniciar objeto
         objEmpleado = new RegEmpleado3()
      e. Leer (empleados, objEmpleado)
         objEmpleado.establecerNumero(empleados.LeerEntero())

```

```

        objEmpleado.establecerNombreEmp (empleados.LeerCadena ())
        objEmpleado.establecerDepto (empleados.LeerEntero ())
        objEmpleado.establecerPuesto (empleados.LeerEntero ())
        objEmpleado.establecerSueldo (empleados.LeerReal ())
    4. endwhile
    5. Imprimir totEmpDep, totSueldosDep
    6. totEmpleados = totEmpleados + totEmpDep
       totSueldos = totSueldos + totSueldosDep
h. endwhile
i. Imprimir totEmpleados, totSueldos
j. Cerrar (empleados)
k. Fin Método catalogoConCortes
9. Método nominaConCortes ()
a. Declarar variables
   totEmpleados, totEmpDep, deptoProceso: Entero
   bruto, impuesto, neto, totBruto, totImpuesto,
   totNeto, totBrutoDep, totImpuestoDep, totNetoDep,
   salMin: Real
b. Abrir (empleados, "C:/arEmp2.dat", Directo, "rw")
c. totEmpleados = 0; totBruto = 0; totImpuesto = 0;
   totNeto=0
d. Solicitar Salario mínimo quincenal
e. Leer salMin
f. Imprimir encabezado
g. Declarar, crear e iniciar objeto
   RegEmpleado3 objEmpleado = new RegEmpleado3 ()
h. Leer (empleados, objEmpleado)
   objEmpleado.establecerNumero (empleados.LeerEntero ())
   objEmpleado.establecerNombreEmp (empleados.LeerCadena ())
   objEmpleado.establecerDepto (empleados.LeerEntero ())
   objEmpleado.establecerPuesto (empleados.LeerEntero ())
   objEmpleado.establecerSueldo (empleados.LeerReal ())
i. while objEmpleado.obtenerNombreEmp () != "FIN"
    1. totEmpDep = 0 ; totBrutoDep = 0
       totImpuestoDep = 0; totNetoDep = 0
    2. deptoProceso = objEmpleado.obtenerDepto ()
    3. while objEmpleado.obtenerDepto () == deptoProceso
       a. bruto = objEmpleado.obtenerSueldo () / 2
       b. if bruto > salMin then
          1. impuesto = (bruto - salMin) * 0.05
       c. else
          1. impuesto = 0
       d. endif
       e. neto = bruto - impuesto
       f. Imprimir objEmpleado.obtenerNumero ()
          objEmpleado.obtenerNombreEmp ()
          objEmpleado.obtenerDepto ()
          bruto, impuesto, Neto

```

```

g. totEmpDep = totEmpDep + 1
   totBrutoDep = totBrutoDep + bruto
   totImpuestoDep = totImpuestoDep + impuesto
   totNetoDep = totNetoDep + neto
h. Crear e iniciar objeto
   objEmpleado = new RegEmpleado3()
i. Leer (empleados, objEmpleado)
   objEmpleado.establecerNumero(empleados.LeerEntero())
   objEmpleado.establecerNombreEmp(empleados.LeerCadena())
   objEmpleado.establecerDepto(empleados.LeerEntero())
   objEmpleado.establecerPuesto(empleados.LeerEntero())
   objEmpleado.establecerSueldo(empleados.LeerReal())
4. endwhile
5. Imprimir totEmpDep, totBrutoDep,
   totImpuestoDep, totNetoDep
6. totEmpleados = totEmpleados + totEmpDep
   totBruto = totBruto + totBrutoDep
   totImpuesto = totImpuesto + totImpuestoDep
   totNeto = totNeto + totNetoDep
j. endwhile
k. Imprimir totEmpleados, totBruto, totImpuesto, totNeto
l. Cerrar (empleados)
m. Fin Método nominaConCortes
Fin Clase EjecutaRegEmpleado3
Fin

```

Explicación:

En la zona de descarga de la Web del libro está disponible:
Programa en Java: RegEmpleado3.java y EjecutaRegEmpleado3.java

**Explicación:**

El algoritmo tiene dos clases: la Clase RegEmpleado3 y la Clase EjecutaRegEmpleado3.

La Clase RegEmpleado3 representa un registro de empleado y, cada vez que se va a procesar un objeto empleado, se creará utilizándola. En esta clase:

1. Se declaran los datos que representan la estructura de la clase:
 - numero para el dato número del empleado.
 - nombreEmp para el nombre del empleado.
 - depto para el departamento del empleado.
 - puesto para el puesto del empleado.
 - sueldo para el sueldo mensual del empleado.
2. Método establecerNumero(num: Entero).
Recibe en el parámetro num el valor que luego coloca en el dato numero.
3. Método establecerNombreEmp(nom: Cadena).
Recibe en el parámetro nom el valor que luego coloca en el dato nombreEmp.
4. Método establecerDepto(dep: Entero).
Recibe en el parámetro dep el valor que luego coloca en el dato depto.

5. Método establecerPuesto(pue: Entero).
Recibe en el parámetro pue el valor que luego coloca en el dato puesto.
 6. Método establecerSueldo(sue: Real).
Recibe en el parámetro sue el valor que luego coloca en el dato sueldo.
 7. Método obtenerNumero(): Entero.
Retorna numero.
 8. Método obtenerNombreEmp(): Cadena.
Retorna nombreEmp (nombre del empleado).
 9. Método obtenerDepto(): Entero.
Retorna depto.
 10. Método obtenerPuesto(): Entero.
Retorna puesto.
 11. Método obtenerSueldo() Real.
Retorna sueldo.
Observe que para cada dato se tiene un método setter y un getter, para establecer y acceder el valor del dato respectivamente.
- Fin de la Clase RegEmpleado3.

En la Clase EjecutaRegEmpleado3:

Se procesa el archivo empleados y en su momento se van a generar objetos utilizando la Clase RegEmpleado3.

En esta clase se tienen los métodos: principal(), crear(), expandir(), altas(), bajas(), cambios(), consultas(), catalogoConCortes() y nominaConCortes().

1. En el Método principal():
 - a. Se declara la variable opcion para leer la opción que se escoja.
 - b. Inicia ciclo do:
 1. Imprime el menú de opciones donde se solicita la opción.
 2. Se lee la respuesta en opcion.
 3. Inicia switch opcion:
 - Si opcion es 1, entonces : Llama al Método crear().
 - Si opcion es 2, entonces : Llama al Método expandir().
 - Si opcion es 3, entonces : Llama al Método altas().
 - Si opcion es 4, entonces : Llama al Método bajas().
 - Si opcion es 5, entonces : Llama al Método cambios().
 - Si opcion es 6, entonces : Llama al Método consultas().
 - Si opcion es 7, entonces : Llama al Método catalogoConCortes().
 - Si opcion es 8, entonces : Llama al Método nominaConCortes().
 4. Fin del switch.
 - c. Fin ciclo do...while. Si opcion != 9 va al do; si no, se sale del ciclo do...while.
 - d. Fin Método principal.

2. En el Método `crear()`:
 - a. Se declaran las variables:
 - `i` para controlar el ciclo `for`.
 - `numRegs` para leer la cantidad de registros con que se creará el archivo.
 - `numEmp` para manejar el dato número del empleado.
 - `nomEmp` para manejar el dato nombre del empleado.
 - `depEmp` para manejar el dato departamento del empleado.
 - `pueEmp` para manejar el dato puesto del empleado.
 - `sueEmp` para manejar el dato sueldo mensual del empleado.
 - b. Se crea el archivo `empleados`, que tiene el nombre físico en el disco “C:/arEmp2.dat”, con organización directa y con modo lectura/escritura.
 - c. Se solicita la cantidad de registros con que se creará el archivo.
 - d. Se lee en `numRegs`.
 - e. Se inician las variables que representan los datos de un registro (objeto) empleado: `numEmp` en 0, `nomEmp` en “”, `depEmp` en 0, `pueEmp` en 0 y `sueEmp` en 0.
 - f. Se declara el objeto `objEmpleado`, usando como base a la Clase `RegEmpleado3`. Dicho objeto se crea e inicializa mediante el constructor por defecto `RegEmpleado3()`.
 - g. Se llama al Método `establecerNumero(numEmp)` del objeto `objEmpleado` para colocar el valor de `numEmp` en el dato `numero`.
Se llama al Método `establecerNombreEmp(nomEmp)` del objeto `objEmpleado` para colocar el valor de `nomEmp` en el dato `nombreEmp`.
Se llama al Método `establecerDepto(depEmp)` del objeto `objEmpleado` para colocar el valor de `depEmp` en el dato `depto`.
Se llama al Método `establecerPuesto(pueEmp)` del objeto `objEmpleado` para colocar el valor de `pueEmp` en el dato `puesto`.
Se llama al Método `establecerSueldo(sueEmp)` del objeto `objEmpleado` para colocar el valor de `sueEmp` en el dato `sueldo`.
 - h. Inicia ciclo `for` desde `i=1` hasta `i<=numRegs` con incrementos de 1.
 1. Se imprime en el archivo `empleados` el objeto `objEmpleado`, el cual contiene el registro con los datos en ceros y nulos, para apartar lugar al objeto (registro) del empleado que le corresponderá ir aquí.
 - i. Fin del ciclo `for`.
 - j. Al `nomEmp` le coloca “FIN”.
 - k. Se llama al Método `establecerNombreEmp(nomEmp)` del objeto `objEmpleado` para colocar el valor de `nomEmp` (“FIN”) en el dato `nombreEmp`.
 - l. Se imprime en el archivo `empleados` el objeto `objEmpleado`, el cual contiene el registro con los datos en ceros y “FIN” en el `nombreEmp`. Es la marca o centinela que indica el fin del archivo.
 - m. Se cierra y libera de uso el archivo `empleados`.
 - n. Fin del Método `crear`.
3. En el Método `expandir()`:
 - a. Se declaran las variables:
 - `i` para controlar el ciclo `for`.
 - `numRegs` para leer la cantidad de registros que se agregarán al archivo.
 - `apuntador` para manejar la posición del apuntador del archivo.

- numEmp para manejar el dato número del empleado.
- nomEmp para manejar el dato nombre del empleado.
- depEmp para manejar el dato departamento del empleado.
- pueEmp para manejar el dato puesto del empleado.
- sueEmp para manejar el dato sueldo mensual del empleado.
- b. Se abre el archivo `empleados`, que tiene el nombre físico en el disco “C:/arEmp2.dat”, con organización directa y con modo lectura/escritura.
- c. Se solicita la cantidad de registros que se le agregarán al archivo.
- d. Se lee en `numRegs`.
- e. Se declara el objeto `objEmpleado`, usando como base a la Clase `RegEmpleado3`. Dicho objeto se crea e inicializa mediante el constructor por defecto `RegEmpleado3()`.
- f. Se obtiene la posición actual del apuntador con `empleados.PosApuntador()`, y se guarda en `apuntador`.
- g. Se lee, del archivo `empleados`, el objeto actual que contiene el registro con los datos del siguiente empleado (objeto) y se coloca en `objEmpleado`. Es decir, se lee el primer objeto (registro).
- h. Inicia ciclo `while objEmpleado.obtenerNombreEmp() != "FIN"`. Se pregunta si no se ha encontrado el fin del archivo. Si es así, entra al ciclo; si no, se sale del ciclo.
 - 1. Se obtiene la posición actual del apuntador con `empleados.PosApuntador()` y se guarda en `apuntador`.
 - 2. Se lee, del archivo `empleados`, el objeto actual que contiene el registro con los datos del siguiente empleado (objeto) y se coloca en `objEmpleado`. Es decir, se lee el siguiente objeto (registro).
- i. Fin del ciclo `while`.
- j. Se coloca el apuntador del archivo en el número de byte que tiene `apuntador`.
- k. Se inician las variables que representan los datos de un registro (objeto) empleado: `numEmp` en 0, `nomEmp` en “”, `depEmp` en 0, `pueEmp` en 0 y `sueEmp` en 0.
- l. Se declara el objeto `objEmpleado`, usando como base a la Clase `RegEmpleado3`. Dicho objeto se crea e inicializa mediante el constructor por defecto `RegEmpleado3()`.
- m. Se llama al Método `establecerNumero(numEmp)` del objeto `objEmpleado` para colocar el valor de `numEmp` en el dato `numero`.
Se llama al Método `establecerNombreEmp(nomEmp)` del objeto `objEmpleado` para colocar el valor de `nomEmp` en el dato `nombreEmp`.
Se llama al Método `establecerDepto(depEmp)` del objeto `objEmpleado` para colocar el valor de `depEmp` en el dato `depto`.
Se llama al Método `establecerPuesto(pueEmp)` del objeto `objEmpleado` para colocar el valor de `pueEmp` en el dato `puesto`.
Se llama al Método `establecerSueldo(sueEmp)` del objeto `objEmpleado` para colocar el valor de `sueEmp` en el dato `sueldo`.
- n. Inicia ciclo `for` desde `i=1` hasta `i<=numRegs` con incrementos de 1.
 - 1. Se imprime en el archivo `empleados` el objeto `objEmpleado`, el cual contiene el registro con los datos en ceros y nulos, para apartar lugar al objeto (registro) del empleado que le corresponderá ir aquí.

- o. Fin del ciclo for.
 - p. A `nomEmp` le coloca "FIN".
 - q. Se llama al Método `establecerNombreEmp(nomEmp)` del objeto `objEmpleado` para colocar el valor de `nomEmp` ("FIN") en el dato `nombreEmp`.
 - r. Se imprime en el archivo empleados el objeto `objEmpleado`, el cual contiene el registro con los datos en ceros y "FIN" en el dato `nombreEmp`. Es la marca o centinela que indica el fin del archivo.
 - s. Se cierra y libera de uso el archivo empleados.
 - t. Fin del Método `expandir`.
4. En el Método `altas()`:
- a. Se declaran las variables:
 - `numEmp` para leer el número del empleado.
 - `nomEmp` para leer el nombre del empleado.
 - `depEmp` para leer el departamento del empleado.
 - `pueEmp` para leer el puesto del empleado.
 - `sueEmp` para leer el sueldo mensual del empleado.
 - `desea` para controlar el ciclo repetitivo
 - b. Se abre el archivo `empleados`, que tiene el nombre físico en el disco "C:/arEmp2.dat", con organización directa y con modo lectura/escritura.
 - c. Inicia ciclo do:
 - 1. Se declara el objeto `objEmpleado`, usando como base a la Clase `RegEmpleado3`. Dicho objeto se crea e inicializa mediante el constructor por defecto `RegEmpleado3()`.
 - 2. Se solicitan los datos del empleado: número, nombre, departamento, puesto y sueldo mensual.
 - 3. Se leen en `numEmp`, `nomEmp`, `depEmp`, `pueEmp`, `sueEmp`.
 - 4. Inicia ciclo while para revisar si el nombre leído en `nomEmp` no tiene 30 caracteres. Si es así, entra al ciclo; si no, se sale del ciclo.
 - a. Se rellena con un espacio cada carácter que le falte para tener los 30 ocupados.
 - 5. Fin del while.

Nota: Esto se hace porque no todos los nombres tienen la misma cantidad de caracteres y se está estableciendo ocupar 30 caracteres. Los que no quepan se abrevian y los que queden cortos (menos de 30) se rellenan con espacios para usar los 30.

 - 6. Se llama al Método `establecerNumero(numEmp)` del objeto `objEmpleado` para colocar el valor de `numEmp` en el dato `numero`.
Se llama al Método `establecerNombreEmp(nomEmp)` del objeto `objEmpleado` para colocar el valor de `nomEmp` en el dato `nombreEmp`.
Se llama al Método `establecerDepto(depEmp)` del objeto `objEmpleado` para colocar el valor de `depEmp` en el dato `depto`.
Se llama al Método `establecerPuesto(pueEmp)` del objeto `objEmpleado` para colocar el valor de `pueEmp` en el dato `puesto`.
Se llama al Método `establecerSueldo(sueEmp)` del objeto `objEmpleado` para colocar el valor de `sueEmp` en el dato `sueldo`.
 - 7. Se coloca el apuntador del archivo empleados en el número de byte que le corresponde, el cual se calcula así: al número de empleado se

- le resta 1 y se multiplica por 48, que es la cantidad de bytes que ocupa un objeto (registro) de empleado.
8. Se imprime en el archivo `empleados` el objeto `objEmpleado`, el cual contiene el registro con los datos del empleado que se está procesando.
 9. Lee la respuesta (¿Otra alta?) en `desea`.
 - d. Fin del ciclo (`while desea == 'S'`). Si se cumple regresa al `do`; si no, se sale del ciclo.
 - e. Se cierra y libera de uso el archivo `empleados`.
 - f. Fin del Método `altas`.
5. En el Método `bajas()`:
- a. Se declaran las variables:
 - `desea` para controlar el ciclo repetitivo.
 - `seguro` para preguntar si está seguro de dar de baja al empleado.
 - `numEmp` para leer el número del empleado.
 - `nomEmp` para leer el nombre del empleado.
 - `depEmp` para leer el departamento del empleado.
 - `pueEmp` para leer el puesto del empleado.
 - `sueEmp` para leer el sueldo mensual del empleado.
 - b. Se abre el archivo `empleados`, que tiene el nombre físico en el disco “C:/arEmp2.dat”, con organización directa y con modo lectura/escritura.
 - c. Inicia ciclo `do`:
 1. Se declara el objeto `objEmpleado`, usando como base a la Clase `RegEmpleado3`. Dicho objeto se crea e inicializa mediante el constructor por defecto `RegEmpleado3()`.
 2. Imprime pantalla de captura de datos.
 3. Lee el número de empleado que se va a dar de baja en `num`.
 4. Se coloca el apuntador del archivo `empleados` en el número de byte que le corresponde, el cual se calcula así: al número de empleado se le resta 1 y se multiplica por 48, que es la cantidad de bytes que ocupa un objeto (registro) de empleado.
 5. Se lee, del archivo `empleados`, el objeto actual que contiene el registro con los datos del empleado (objeto) y se coloca en `objEmpleado`.
 6. Imprime los datos del objeto empleado que se dará de baja.
 7. Pregunta si está seguro que lo quiere dar de baja.
 8. Lee la respuesta en `seguro`.
 9. Si `seguro` es igual a ‘S’, entonces:
 - a. Se inician las variables que representan los datos de un registro (objeto) empleado: `numEmp` en 0, `nomEmp` en “”, `depEmp` en 0, `pueEmp` en 0 y `sueEmp` en 0.
 - b. Se coloca el apuntador del archivo `empleados` en el número de byte que le corresponde, el cual se calcula así: al número de empleado se le resta 1 y se multiplica por 48, que es la cantidad de bytes que ocupa un objeto (registro) de empleado.
 - c. Se imprime en el archivo `empleados` el objeto `objEmpleado`, el cual contiene el registro con los datos en ceros y nulo, con lo cual lo borra o da de baja.
 10. Fin del `if`.

11. Lee la respuesta (¿Otra baja?) en desea.
 - d. Fin del ciclo (`while desea == 'S'`). Si se cumple regresa al `do`; si no, se sale del ciclo.
 - e. Se cierra y libera de uso el archivo `empleados`.
 - f. Fin del Método `bajas`.
6. En el Método `cambios()`:
- a. Se declaran las variables:
 - `desea` para controlar el ciclo repetitivo.
 - `num` para leer el número del empleado que se dará de baja.
 - `opcion2` para leer número del dato a cambiar.
 - `numEmp` para leer el número del empleado del objeto (registro).
 - `nomEmp` para leer el nombre del empleado.
 - `depEmp` para leer el departamento del empleado.
 - `pueEmp` para leer el puesto del empleado.
 - `sueEmp` para leer el sueldo mensual del empleado.
 - b. Se abre el archivo `empleados`, que tiene el nombre físico en el disco “C:/arEmp2.dat”, con organización directa y con modo lectura/escritura.
 - c. Inicia ciclo `do`:
 1. Se declara el objeto `objEmpleado`, usando como base a la Clase `RegEmpleado3`. Dicho objeto se crea e inicializa mediante el constructor por defecto `RegEmpleado3()`.
 2. Imprime pantalla de captura de datos.
 3. Lee el número de empleado al que se le harán cambios en `num`.
 4. Se coloca el apuntador del archivo `empleados` en el número de byte que le corresponde, el cual se calcula así: al número de empleado se le resta 1 y se multiplica por 48, que es la cantidad de bytes que ocupa un objeto (registro) de empleado.
 5. Se lee, del archivo `empleados`, el objeto actual que contiene el registro con los datos del empleado (objeto) y se coloca en `objEmpleado`.
 6. Imprime los datos del objeto empleado al que se le harán cambios.
 7. Lee el número de dato que será cambiado en `opcion2`.
 8. Inicia `while`. Si (`opcion2>0`) y (`opcion2<5`) entra al ciclo; si no, se sale del ciclo.
 - a. Inicia `switch opcion2`:
 - Si `opcion2` es 1, entonces: Lee el nuevo `nomEmp` y lo establece en el objeto `objEmpleado.establecerNombreEmp(nomEmp)`.
 - Si `opcion2` es 1, entonces: Lee el nuevo `depEmp` y lo establece en el objeto `objEmpleado.establecerDepto(depEmp)`.
 - Si `opcion2` es 1, entonces: Lee el nuevo `pueEmp` y lo establece en el objeto `objEmpleado.establecerPuesto(pueEmp)`.
 - Si `opcion2` es 1, entonces: Lee el nuevo `sueEmp` y lo establece en el objeto `objEmpleado.establecerSueldo(sueEmp)`.
 - b. Fin del `switch`.
 - c. Lee el número de dato que será cambiado en `opcion2`.
 9. Fin del ciclo `while`.
 10. Se coloca el apuntador del archivo `empleados` en el número de byte que le corresponde, el cual se calcula así: al número de empleado

- se le resta 1 y se multiplica por 48, que es la cantidad de bytes que ocupa un objeto (registro) de empleado.
11. Se imprime en el archivo `empleados` el objeto `objEmpleado`, el cual contiene el registro con los datos cambiados.
 12. Lee la respuesta (¿Otro cambio?) en `desea`.
- d. Fin del ciclo (`while desea == 'S'`). Si se cumple regresa al do; si no, se sale del ciclo.
 - e. Se cierra y libera de uso el archivo `empleados`.
 - f. Fin del Método `cambios`.
7. En el Método `consultas()`:
- a. Se declaran las variables:
 - `desea` para controlar el ciclo repetitivo.
 - `num` para leer el número del empleado que se consultará.
 - `numEmp` para leer el número del empleado del objeto (registro).
 - `nomEmp` para leer el nombre del empleado.
 - `depEmp` para leer el departamento del empleado.
 - `pueEmp` para leer el puesto del empleado.
 - `sueEmp` para leer el sueldo mensual del empleado.
 - b. Se abre el archivo `empleados`, que tiene el nombre físico en el disco “C:/arEmp2.dat”, con organización directa y con modo lectura/escritura.
 - c. Inicia ciclo do:
 1. Se declara el objeto `objEmpleado`, usando como base a la Clase `RegEmpleado3`. Dicho objeto se crea e inicializa mediante el constructor por defecto `RegEmpleado3()`.
 2. Imprime pantalla de captura de datos.
 3. Lee el número de empleado que se consultará en `num`.
 4. Se coloca el apuntador del archivo `empleados` en el número de byte que le corresponde, el cual se calcula así: al número de empleado se le resta 1 y se multiplica por 48, que es la cantidad de bytes que ocupa un objeto (registro) de empleado.
 5. Se lee, del archivo `empleados`, el objeto actual que contiene el registro con los datos del empleado (objeto) y se coloca en `objEmpleado`.
 6. Imprime los datos del objeto empleado.
 7. Lee la respuesta (¿Otro cambio?) en `desea`.
 - d. Fin del ciclo (`while desea == 'S'`). Si se cumple regresa al do; si no, se sale del ciclo.
 - e. Se cierra y libera de uso el archivo `empleados`.
 - f. Fin del Método `consultas`.
8. El Método `catalogoConCortes()`:
Ya se explicó en el punto de emisión de reportes con cortes de control.
9. El Método `nominaConCortes()`:
Ya se explicó en el punto de emisión de reportes con cortes de control.

Fin de la Clase `EjecutaRegEmpleado3`.

Fin del algoritmo.

Observaciones:

Aquí tenemos un algoritmo completo que permite manejar un pequeño sistema de nómina aplicando todas las operaciones sobre registros y archivos. Sin embargo, el algoritmo no contempla validaciones. Por ejemplo:

- Al crear el archivo no se da la oportunidad de retractarse, y se puede crear un archivo ya existente erróneamente, perdiéndose los datos que ya tenía.
- Al hacer un alta no se verifica si el empleado ya está dado de alta, así que se puede dar de alta de nuevo. Tampoco se revisa si el número de empleado cabe en el archivo de acuerdo al número de registros con que fué creado o expandido el archivo.
- Al hacer la captura de datos no se detecta la ocurrencia de errores, y mucho menos recuperarse de éstos. Por ejemplo, si se lee un dato de tipo entero, no se debe permitir que se tecleen letras o símbolos especiales, etcétera. Al leer el nombre de una persona no deben permitirse números o símbolos especiales.
- Al hacer una consulta, baja o cambio, debería dar un mensaje de error si se teclea un número de empleado que no existe.

A manera de ejercicio se recomienda que elabore el mismo sistema de nómina pero ahora contemplando algunas o todas las validaciones antes mencionadas.

15.6 Ejercicios resueltos

Tabla 15.1: muestra los ejercicios resueltos disponibles en la zona de descarga del Capítulo 15 de la Web del libro.

Ejercicio	Descripción
Ejercicio 15.6.1	Procesa un archivo de obreros
Ejercicio 15.6.2	Procesa un archivo de vendedores

15.7 Ejercicios propuestos

1. Hacer un algoritmo que permita crear un archivo de artículos y realizar altas, bajas, cambios y consultas. Los datos que se tienen de cada artículo son los siguientes:

- Número
- Descripción
- Precio anterior
- Precio actual

Asimismo, emitir el siguiente listado de inflación de artículos:

ANÁLISIS DE INFLACIÓN			
ARTÍCULO	PRECIO ANTERIOR	PRECIO ACTUAL	PTJE. INFLACIÓN
XXXXXXXXXXXX	99,999.99	99,999.99	99.99
XXXXXXXXXXXX	99,999.99	99,999.99	99.99
---	---	---	---
XXXXXXXXXXXX	99,999.99	99,999.99	99.99

Promedio de inflación: 99.99

Artículo con mayor inflación: XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX

Porcentaje mayor de inflación: 99.99

Forma de calcular el porcentaje de inflación:

$$\text{PTJE. INFLACIÓN} = \frac{\text{Precio actual} - \text{Precio anterior}}{\text{Precio anterior}} \times 100$$

2. Elaborar un algoritmo que permita crear un archivo de alumnos y realizar altas, bajas, cambios y consultas. Los datos que se tiene de cada alumno son los siguientes:

- Número
- Nombre
- Calificación 1
- Calificación 2
- Calificación 3
- Calificación 4

También debe emitir siguiente reporte:

NOMBRE	ANÁLISIS DE CALIFICACIONES				PROMEDIO
	CAL.1	CAL.2	CAL.3	CAL.4	
XXXXXXXXXXXXXXXXXXXXX	99.99	99.99	99.99	99.99	99.99
XXXXXXXXXXXXXXXXXXXXX	99.99	99.99	99.99	99.99	99.99
---	---	---	---	---	---
---	---	---	---	---	---
XXXXXXXXXXXXXXXXXXXXX	99.99	99.99	99.99	99.99	99.99
PROMEDIOS GENERALES	99.99	99.99	99.99	99.99	99.99

El promedio general de cada calificación se calcula sumando las calificaciones de todos los alumnos y dividiendo el resultado entre el número de alumnos.

3. Hacer un algoritmo igual al del ejercicio de empleados desarrollado en este capítulo, pero debe añadir un archivo que contenga la siguiente tabla para el cálculo del impuesto:

	Lím.Inferior	Lím.Superior	Cuota Fija	Ptje.Impuesto
nivel 1				
nivel 2				
nivel 3				
nivel 15				

Proceso para calcular el impuesto:

Tomando como base el excedente del sueldo bruto sobre el salario mínimo:

- a) Se busca el nivel en el que se encuentra dicho excedente.
- b) Se obtiene la diferencia entre el excedente y el límite inferior del nivel; a ésta se le aplica el porcentaje de impuesto y se le suma la cuota fija.

4. Hacer un algoritmo que permita crear un archivo de pedidos y realizar altas, bajas, cambios y consultas. Los datos de cada pedido son los siguientes:

- Nombre del cliente
- Dirección
- Artículo
- Cantidad
- Precio unitario

Además, deberá emitir el siguiente listado de pedidos:

LISTADO DE PEDIDOS				
NOMBRE DEL CLIENTE	ARTÍCULO	CANTIDAD	PRECIO.UNIT.	TOTAL
XXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	999	99,999.99	999,999.99
XXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	999	99,999.99	999,999.99
XXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	999	99,999.99	999,999.99

XXXXXXXXXXXXXXXXXXXXX	XXXXXXXXXX	999	99,999.99	999,999.99
TOTAL	999		999,999.99	9999,999.99

5. Hacer un algoritmo que permita crear un archivo de obreros y realizar altas, bajas, cambios y consultas. Los datos de cada obrero son los siguientes:

- Nombre
- Producción de los 12 meses en un arreglo de 12 elementos

Asimismo, debe emitir un premio anual al obrero más productivo. El premio se calcula de la siguiente manera: cantidad más alta producida menos cantidad más baja multiplicado por 10. Imprimir:

Nombre del obrero: XX

Valor del premio: 99,999,999.99

6. Hacer un algoritmo que permita crear un archivo de cuentas de cheques y realizar altas, bajas, cambios y consultas. Los datos de cada cuenta son:

- Número de cuenta
- Nombre del cliente
- Saldo

Además, debe permitir que se hagan depósitos y retiros. Se requiere que emita el siguiente reporte de estados de cuenta:

ESTADOS DE CUENTA		
NÚM. CUENTA	NOMBRE DEL CLIENTE	SALDO
99999999-9	XXXXXXXXXXXXXXXXXXXXX	999,999.99
99999999-9	XXXXXXXXXXXXXXXXXXXXX	999,999.99
99999999-9	XXXXXXXXXXXXXXXXXXXXX	999,999.99

99999999-9	XXXXXXXXXXXXXXXXXXXXX	999,999.99
TOTAL	999 CLIENTES	9,999,999.99

7. Elaborar un algoritmo que maneje el sistema de nómina que se elaboró anteriormente, sólo que, en lugar de imprimir el departamento y puesto como un número, que lo haga en forma de nombre del departamento y nombre de puesto.

Debe manejar un archivo de departamentos y otro de puestos, a los que se les debe hacer altas, bajas y cambios, y de ahí obtener los nombres tomando como base los números. Los datos de estos nuevos archivos son:

<u>departamentos</u>	<u>puestos</u>
depto	puesto
nombreDepto	nombrePuesto

8. Elaborar un algoritmo que maneje un sistema de control de usuarios del agua potable de una localidad y permita crear, hacer altas, bajas y cambios a un archivo de usuarios y otro de colonias. El archivo de usuarios contiene los datos de todos los usuarios de la localidad, la cual se divide en varias colonias; es por ello que cada usuario tiene un dato que es la clave de la colonia a la que pertenece. El archivo de colonias contiene los nombres de todas las colonias de la localidad. También se manejará un archivo de tarifas, que contendrá un arreglo de 8 elementos: en el elemento 1 tendrá la tarifa que le corresponde al usuario tipo 1, en el elemento 2 la tarifa para el usuario tipo 2, y así sucesivamente. Cada vez que se desee cambiar las tarifas, simplemente se crea el archivo con las tarifas correspondientes.

<u>usuarios</u>	<u>colonias</u>	<u>tarifas</u>
numero	claveColonia	tarifa[8]
tipoUsuario	colonia	
nombre		
claveColonia		
direccion		
consumo		
pago		

En donde:

numero	Es el número de identificación del usuario.
tipoUsuario	Es el tipo de usuario (1,2,3,4,5,6,7,8).
nombre	Es el nombre del usuario.
claveColonia	Es la clave de identificación de la colonia.
direccion	Es la dirección del domicilio del usuario.
consumo	Es el consumo en metros cúbicos de agua del usuario en el período correspondiente.
pago	Es el pago realizado por el usuario por el período correspondiente. Si está en cero, es que no ha pagado.
colonia	Es el nombre de la colonia en la que vive el usuario.
tarifa[8]	Es un arreglo que tiene 8 elementos: en el 1 tiene la tarifa que le corresponde al usuario tipo 1, en el 2 la que le corresponde al usuario tipo 2, etcétera.

Para la captura del consumo se debe hacer un proceso especial, al igual que para la captura de los pagos. El algoritmo (programa) también debe proporcionar la posibilidad de hacer consultas de usuarios y de colonias con dar la clave correspondiente. Además, debe emitir los siguientes reportes:

REPORTE DE PAGOS			
NÚMERO	NOMBRE USUARIO	COLONIA	PAGO
---	X-----X	X-----X	----
---			----
---	X-----X	X-----X	----

	TOTAL COLONIA - USUARIOS		----
	---		----
	TOTAL COLONIA - USUARIOS		----
	TOTAL GENERAL - USUARIOS		----

Nota: Para efectos de este reporte, el archivo debe estar ordenado por colonia, es decir, todos los usuarios de la colonia 1 deben estar juntos al principio del archivo, luego siguen los de la colonia 2, y así sucesivamente.

Se pide el total de pagos hechos por cada colonia y el total general.

REPORTE DE PAGOS NO REALIZADOS			
NÚMERO	NOMBRE USUARIO	COLONIA	DEBE
---	X-----X	X-----X	----
---			----
---	X-----X	X-----X	----

	TOTAL COLONIA - USUARIOS		----
	---		----
	TOTAL COLONIA - USUARIOS		----
	TOTAL GENERAL - USUARIOS		----

Nota: Para efectos de este reporte, el archivo debe estar ordenado por colonia, es decir, todos los usuarios de la colonia 1 deben estar juntos al principio del archivo, luego siguen los de la colonia 2, y así sucesivamente.

Se pide el total de pagos no hechos, es decir, lo que debe cada usuario, totalizado por cada colonia y en general.

REPORTE DE FACTURACIÓN						
NÚM.	NOMBRE	COLONIA	CONSUMO	IMPORTE	SOBRE CONSUMO	TOTAL
---	X-----X	X-----X	----	----	----	----

---	X-----X	X-----X	----	----	----	----
	TOTAL COLONIA - USUARIOS	----	----	----	----	----

	TOTAL COLONIA - USUARIOS	----	----	----	----	----
	TOTAL GENERAL - USUARIOS	----	----	----	----	----

Nota: Para efectos de este reporte, el archivo debe estar ordenado por colonia, es decir, todos los usuarios de la colonia 1 deben estar juntos al principio del archivo, luego siguen los de la colonia 2, y así sucesivamente.



Para efectos de este reporte, el archivo debe estar ordenado por colonia, es decir, todos los usuarios de la colonia 1 deben estar juntos al principio del archivo, luego siguen los de la colonia 2, y así sucesivamente.



Para efectos de este reporte, el archivo debe estar ordenado por colonia, es decir, todos los usuarios de la colonia 1 deben estar juntos al principio del archivo, luego siguen los de la colonia 2, y así sucesivamente.

En donde:

CONSUMO	Se lee junto con los datos del registro del usuario y es la cantidad de metros cúbicos de agua que consumió el usuario.
IMPORTE	Se calcula multiplicando el consumo por la tarifa correspondiente, la cual debe ser obtenida del arreglo de tarifas de acuerdo al tipo de usuario.
SOBRECONSUMO	Si consumió arriba de 50 metros cúbicos, se le cobra un 10% de sobreconsumo; si pasó de 100, un 20%; si pasó de 200, un 30% más; y si pasó de 300, un 50% más.
TOTAL	Es el total por pagar. Se suman el IMPORTE y SOBRECONSUMO.

Se pide el total de consumo, importe, sobreconsumo y total por cada colonia y en general.



Se debe listar un recibo por cada usuario.

RECIBO DEL AGUA	
NÚMERO:	Recibo Número:
USUARIO:	
DIRECCIÓN:	
COLONIA:	Páguese antes de
C.P.:	
TIPO USUARIO:	
CONSUMO:	IMPORTE:
TARIFA:	SOBRECONSUMO:
	TOTAL A PAGAR:

Nota: Se debe listar un recibo por cada usuario.

9. Elaborar un algoritmo que maneje un sistema de control de ventas de libros de una editorial y permita crear y hacer altas, bajas y cambios a los archivos de libros, autores, ventas, clientes, ciudades, países y estados. A continuación se describen los datos que contiene cada uno de los archivos:

<u>libros</u>	<u>clientes</u>	<u>autores</u>
claveLibro	claveCliente	claveAutor
titulo	razonSocial	nombre
claveAutor	direccionCte	direccionAutor
edicion	claveCiudad	claveCiudad
fechaEdicion	cpCliente	cpAutor
existencia	claveEstado	claveEstado
stockMinimo	clavePais	clavePais
stockMaximo	telCliente	telAutor
nivelReorden	faxCliente	faxAutor
precioVenta		

ventas	ciudades	estados
claveLibro	claveCiudad	claveEstado
claveCliente	ciudad	estado
cantidad		
precio		
fecha		
<hr/>		
países		
clavePais		
pais		

En donde:

claveLibro	Es la clave de identificación del libro.
titulo	Es el título del libro.
claveAutor	Clave que identifica al autor.
edicion	Es el número de edición del libro.
fechaEdicion	Es la fecha de edición.
existencia	Es la cantidad de unidades que hay en existencia.
stockMinimo	Es la cantidad mínima que se debe tener en existencia.
stockMaximo	Es la cantidad máxima que se debe tener en existencia.
nivelReorden	Es la cantidad de unidades en existencia que indica que se debe ordenar más unidades (reimprimir).
precioVenta	Es el precio unitario de venta del libro.
claveCliente	Es la clave de identificación del cliente.
razonSocial	Es la razón social del cliente.
direccionCte	Es la dirección del cliente.
claveCiudad	Es la clave de la ciudad donde radica el cliente.
cpCliente	Es el código postal del domicilio del cliente.
claveEstado	Es la clave del estado o entidad federativa donde radica el cliente.
clavePais	Es la clave del país donde radica el cliente.
telCliente	Es el número telefónico del cliente.
faxCliente	Es el número de fax del cliente.
nombre	El nombre del autor.
direccionAutor	Es la dirección del autor.
claveCiudad	Es la clave de la ciudad donde radica el autor.
cpAutor	Es el código postal del domicilio del autor.
claveEstado	Es la clave del estado o entidad federativa donde radica el autor.
clavePais	Es la clave del país donde radica el autor.
telAutor	Es el número telefónico del autor.

faxAutor	Es el número de fax del autor.
cantidad	Es la cantidad de libros comprados.
precio	Es el precio unitario por libro comprado.
fecha	Es la fecha de la compra.
ciudad	Es el nombre de la ciudad.
estado	Es el nombre del estado o entidad federativa.
pais	Es el nombre del país.

El algoritmo también debe proporcionar la posibilidad de hacer consultas a los datos de cada uno de los archivos con dar la clave correspondiente. Además, debe emitir los siguientes reportes:

REPORTE DE LIBROS								
CLAVE LIBRO	TÍTULO	AUTOR	EXISTENCIA	PRECIO VENTA	VALOR EXISTENCIA	STOCK MÍNIMO	STOCK MÁXIMO	
---	X-----X	X-----X	----	----	----	----	----	----
.								
---	X-----X	X-----X	----	----	----	----	----	----
TOTAL	-- LIBROS	----	----	----	----	----	----	

REPORTE DE VENTAS POR AUTOR					
CLAVE AUTOR	AUTOR	TÍTULO LIBRO	TOT. UNID. VENDIDAS	PRECIO UNITARIO	TOTAL VENDIDO
----	X-----X	X-----X	----	----	----

----	X-----X	X-----X	----	----	----
TOTAL AUTOR	-- LIBROS		----	----	----

TOTAL AUTOR	-- LIBROS		----	----	----
TOTAL GENERAL	-- LIBROS		----	----	----

REPORTE DE VENTAS POR PAÍSES				
CLAVE LIBRO	TÍTULO LIBRO	AUTOR	PAÍS	TOT. UNIDADES VENDIDAS
----	X-----X	X-----X	X-----X	----

----	X-----X	X-----X	X-----X	----
TOTAL LIBRO	-- PAÍSES		----	

TOTAL LIBRO	-- PAÍSES		----	
TOTAL GENERAL	-- LIBROS		----	

REPORTE DE VENTAS DETALLADO										
CLAVE LIBRO	TÍTULO LIBRO	AUTOR	CLIENTE	CIUDAD	ESTADO	PAÍS	TOT. UNID. VEND.	PREC. UNIT.	TOTAL	
----	X-----X	X-----X	X-----X	X-----X	X-----X	X-----X	----	----	----	
----	X-----X	X-----X	X-----X	X-----X	X-----X	X-----X	----	----	----	
TOTAL LIBRO -- VENTAS		----	----	----						

TOTAL LIBRO -- VENTAS		----	----	----						

TOTAL GENERAL -- LIBROS		----	----	----						

10. Desarrollar un algoritmo para el manejo de un sistema de control de inventarios que permita crear y realizar altas, bajas y cambios a un archivo de artículos y a uno de movimientos que contengan los siguientes datos:

articulos	movimientos
numero	numero
linea	tipoMovimiento
descripcion	fechaMovimiento
localizacion	cantidad
unidadDeMedida	precioUnitario
existencia	
stockMinimo	
stockMaximo	
precioDeCompra	
precioDeVenta	
fechaUltCompra	
fechaUltVenta	
proveedor1	
proveedor2	

Además, dicho algoritmo deberá permitirnos emitir los siguientes reportes:

CATÁLOGO DE ARTÍCULOS						
NÚMERO LÍNEA	UNIDAD MEDIDA DESCRIPCIÓN	LOCALIZACION EXISTENCIA	STOCK MÍN. STOCK MAX.	PCIO.ULT.CPA PCIO.ULT.VTA	FCHA.ÚLT.ENT FCHA.ÚLT.SAL	PROV1 PROV2
99999	XXXXXXXXXXXXXX	XXXXXXXXXX	9999	999,999.99	99/99/99	XXXXXX
99	XXXXXXXXXXXXXX	9999	9999	999,999.99	99/99/99	XXXXXX

TOTAL LÍNEA	99	999	ARTÍCULOS			

TOTAL LÍNEA	99	999	ARTÍCULOS			

TOTAL GENERAL		9999	ARTÍCULOS			

EXISTENCIA VALORIZADA

LÍNEA	NÚMERO	DESCRIPCIÓN	EXISTENCIA	PCIO.CPA	PCIO.VTA	EXIST. VALORIZADA	
						PCIO.CPA	PCIO.VTA
99	99999	XXXXXXXXXX	9999	99999.99	99999.99	99999.99	99999.99

	TOTAL LÍNEA 99	999 ARTÍCULOS				99999.99	99999.99

	TOTAL LÍNEA 99	999 ARTÍCULOS				99999.99	99999.99

	TOTAL GENERAL	9999 ARTÍCULOS				99999.99	99999.99

ARTÍCULOS CON POCO MOVIMIENTO

LÍNEA	NÚMERO	DESCRIPCIÓN	UNIDAD MEDIDA	EXIST.	PCIO.CPA	PCIO.VTA	FECH.ÚLT	FECH.ÚLT
							COMPRA	VENTA
99	99999	XXXXXXXXXX	XXXXXX	9999	99999.99	99999.99	99/99/99	99/99/99

	TOTAL LÍNEA 99	999 ARTÍCULOS						

	TOTAL LÍNEA 99	999 ARTÍCULOS						

	TOTAL GENERAL	9999 ARTÍCULOS						

LISTA DE PRECIOS

LÍNEA	NÚMERO	DESCRIPCIÓN	UNIDAD MEDIDA	PRECIO VENTA
99	99999	XXXXXXXXXXXXXXXXXX	XXXXXXXXXXXXXX	999,999.99

	TOTAL LÍNEA 99	999 ARTÍCULOS		

	TOTAL LÍNEA 99	999 ARTÍCULOS		

	TOTAL GENERAL	9999 ARTÍCULOS		

LISTADO DE PEDIDOS

LÍNEA	NÚMERO	DESCRIPCIÓN	UNIDAD MEDIDA	CANTIDAD	FECHA PEDIDO	FECHA RECIBIDO	PROVEEDOR
99	99999	XXXXXXXXXX	XXXXXX	9999	99/99/99	99/99/99	XXXXXXXXXX

	TOTAL PEDIDOS	999 ARTÍCULOS					

PRODUCTOS BAJO MÍNIMO					
LÍNEA	NÚMERO	DESCRIPCIÓN	UNIDAD MEDIDA	CANTIDAD	COSTO ESTIMADO
99	99999	XXXXXXXXXXXXX	XXXXXXXXXXXXX	9999	999,999.99

TOTAL LÍNEA 99		999 ARTÍCULOS			9,999,999.99

TOTAL LÍNEA 99		999 ARTÍCULOS			9,999,999.99

TOTAL GENERAL		9999 ARTÍCULOS			99,999,999.99

MOVIMIENTOS DEL MES						
LÍNEA	NÚMERO	DESCRIPCIÓN	FECHA MOV.	TIPO	CANTIDAD	PRECIO
99	99999	XXXXXXXXXXXXXXXXX	99/99/99	X	9999	999,999.99

TOTAL DE MOVIMIENTOS		999				

TOTAL DE MOVIMIENTOS		999				

TOTAL LINEA 99		999 ARTÍCULOS				

TOTAL LINEA 99		999 ARTÍCULOS				

TOTAL GENERAL		9999 ARTÍCULOS				

El algoritmo deberá considerar las siguientes pantallas y menús:

SISTEMA DE INVENTARIOS

SISTEMA DE INVENTARIOS MENU PRINCIPAL
1. CREAR ARCHIVOS 2. ACTUALIZACION ARCH. ARTICULOS 3. CAPTURA DE MOVIMIENTOS 4. CONSULTAS DE ARTICULOS 5. EMISION DE REPORTES 6. FIN
ESCOGER OPCION:

SISTEMA DE INVENTARIOS CREACION DE ARCHIVOS
1. CREAR ARCH. ARTICULOS 2. CREAR ARCH. MOVIMIENTOS 3. FIN

ESCOGER OPCION:

SISTEMA DE INVENTARIOS ACTUALIZACION ARCH. ARTICULOS

- | |
|---|
| 1. ALTAS DE ARTICULOS
2. BAJAS DE ARTICULOS
3. CAMBIOS DE ARTICULOS
4. FIN |
|---|

ESCOGER OPCION:

SISTEMA DE INVENTARIOS ALTAS DE ARTICULOS
--

NUMERO: LINEA: DESCRIPCION: UNIDAD MEDIDA: LOCALIZACION: EXISTENCIA: STOCK MINIMO: STOCK MAXIMO: PRECIO ULTIMA COMPRA: PRECIO DE VENTA: FECHA ULTIMA ENTRADA: FECHA ULTIMA SALIDA: PROVEEDOR 1: PROVEEDOR 2:

ESCOGER OPCION:

SISTEMA DE INVENTARIOS ALTAS DE ARTICULOS
NUMERO: LINEA: DESCRIPCION: UNIDAD MEDIDA: LOCALIZACION: EXISTENCIA: STOCK MINIMO: STOCK MAXIMO: PRECIO ULTIMA COMPRA: PRECIO DE VENTA: FECHA ULTIMA ENTRADA: FECHA ULTIMA SALIDA: PROVEEDOR 1: PROVEEDOR 2:
¿OTRA ALTA (S/N)?

SISTEMA DE INVENTARIOS BAJAS DE ARTICULOS
NUMERO: LINEA: DESCRIPCION: UNIDAD MEDIDA: LOCALIZACION: EXISTENCIA: STOCK MINIMO: STOCK MAXIMO: PRECIO ULTIMA COMPRA: PRECIO DE VENTA: FECHA ULTIMA ENTRADA: FECHA ULTIMA SALIDA: PROVEEDOR 1: PROVEEDOR 2:
¿OTRA BAJA (S/N)?

SISTEMA DE INVENTARIOS CAMBIOS DE ARTICULOS
NUMERO: 1: LINEA: 2: DESCRIPCION: 3: UNIDAD MEDIDA: 4: LOCALIZACION: 5: EXISTENCIA: 6: STOCK MINIMO: 7: STOCK MAXIMO: 8: PRECIO ULTIMA COMPRA: 9: PRECIO DE VENTA: 10: FECHA ULTIMA ENTRADA: 11: FECHA ULTIMA SALIDA: 12: PROVEEDOR 1: 13: PROVEEDOR 2:
NUMERO DE DATO POR CAMBIAR (0=FIN):
¿OTRO CAMBIO (S/N)? :

SISTEMA DE INVENTARIOS CAPTURA DE MOVIMIENTOS
NUMERO: DESCRIPCION: TIPO DE MOVIMIENTO (E=ENTRADA s=SALIDA): FECHA: CANTIDAD (UNIDADES): PRECIO:
¿OTRO MOVIMIENTO (S/N)? :

SISTEMA DE INVENTARIOS CONSULTAS DE ARTICULOS
NUMERO: LINEA : DESCRIPCION: UNIDAD MEDIDA: LOCALIZACION: EXISTENCIA: STOCK MINIMO: STOCK MAXIMO: PRECIO ULTIMA COMPRA: PRECIO DE VENTA: FECHA ULTIMA ENTRADA: FECHA ULTIMA SALIDA: PROVEEDOR 1: PROVEEDOR 2:
¿OTRA CONSULTA (S/N)?

SISTEMA DE INVENTARIOS EMISION DE REPORTES
1. CATALOGO DE ARTICULOS 2. EXISTENCIA VALORIZADA 3. ARTICULOS CON POCO MOVIMIENTO 4. LISTA DE PRECIOS 5. ARTICULOS BAJO MINIMO 6. LISTADO DE PEDIDOS 7. FIN
ESCOGER OPCION:

15.8 Resumen de conceptos que debe dominar

- Registros y archivos (conceptos)
- Organización de archivos
- Manejo de registros en seudocódigo
- Operaciones para el manejo de archivos en seudocódigo
- Proceso de un archivo secuencial
- Proceso de un archivo directo

15.9 Contenido de la página Web de apoyo

El material marcado con asterisco () sólo está disponible para docentes.*

15.9.1 Resumen gráfico del capítulo

15.9.2 Autoevaluación

15.9.3 Programas en Java

15.9.4 Ejercicios resueltos

15.9.5 Power Point para el profesor (*)

16

Otros temas

Contenido

- 16.1 Clasificación (ordenación) de datos
- 16.2 Uso del tipo static
- 16.3 Uso del this
- 16.4 Recursividad
- 16.5 Graficación
- 16.6 Resumen de conceptos que debe dominar
- 16.7 Contenido de la página Web de apoyo
 - El material marcado con asterisco (*) sólo está disponible para docentes.*
 - 16.7.1 Resumen gráfico del capítulo
 - 16.7.2 Autoevaluación
 - 16.7.3 Programas en Java
 - 16.7.4 Power Point para el profesor (*)

Objetivos del capítulo

- Estudiar la clasificación (ordenación) de datos
- Aprender a usar el tipo static
- Estudiar cómo usar el this
- Aprender a usar la recursividad
- Estudiar cómo usar la graficación

Competencias

- Competencia general del capítulo
 - *Analizar problemas y diseñar algoritmos que los solucionen aplicando la arquitectura orientada a objetos y otros temas.*
- Competencias específicas del capítulo
 - Diseña algoritmos orientados a objetos aplicando clasificación (ordenación) de datos.
 - Elabora algoritmos orientados a objetos aplicando el tipo static.
 - Desarrolla algoritmos orientados a objetos aplicando this.
 - Describe el uso de la recursividad.
 - Detalla el uso de la graficación.

Introducción

Con el estudio del capítulo anterior, usted ya domina las estructuras de datos: registros y archivos, y cómo diseñar algoritmos usando esas estructuras.

El objetivo de este capítulo es que usted sea capaz de elaborar algoritmos utilizando otros temas que permiten reforzar la técnica presentada, los cuales sirven como introducción a otros temas más avanzados y hacen que la metodología de este libro sea útil para diseñar programas que van más allá del alcance de esta obra.

Se explica que la clasificación (ordenación) de datos quiere decir ordenar los datos en orden secuencial, ya sea en orden ascendente o descendente y que existen diversos métodos que permiten llevar a cabo la clasificación u ordenamiento de datos. Asimismo, se muestra el uso del método denominado intercambio o burbuja.

Se presenta el uso del tipo static (estático) y se explica el uso del this.

Se expone que los lenguajes de programación soportan la recursividad; ésta significa que un elemento puede definirse en términos de sí mismo.

Se explica el uso de diversas funciones para graficar figuras como círculos, arcos, barras, etcétera.

Éste es el último capítulo del libro.

16.1 Clasificación (ordenación) de datos

Clasificar quiere decir ordenar los datos en orden secuencial, ya sea en orden ascendente o descendente. Por ejemplo, si se tienen los valores: 3 7 2 9 4 1, se dice que están desordenados porque no existe un orden secuencial en la forma en que están organizados. Al ordenarlos en forma secuencial ascendente quedan en el siguiente orden: 1 2 3 4 7 9; o bien, si se ordenan en forma descendente, quedan: 9 7 4 3 2 1.

Métodos de ordenamiento o clasificación

Existen diversos métodos que permiten llevar a cabo la clasificación u ordenamiento de datos. Por ejemplo, existe un método denominado intercambio o burbuja, el cual consiste en el siguiente proceso:

1. El primer elemento se compara con el segundo; si es mayor el primero que el segundo, se intercambian, es decir, el primero se coloca en la posición del segundo y el segundo en la posición del primero.
2. El segundo elemento se compara con el tercero; si es mayor, se intercambian. Luego se compara el tercero con el cuarto, haciendo lo propio, y así sucesivamente hasta el último elemento del arreglo.
3. Repetir el paso 1 y 2 hasta que se dé el caso de que no se haya hecho ningún intercambio, lo que significa que el arreglo ya quedó ordenado. De ese modo termina el proceso de clasificación.

A este método se le conoce como burbuja porque los elementos que están en una posición más abajo de acuerdo a su magnitud tienden a “subir” hacia la posición que les corresponde, algo similar a lo que sucede en una botella de alguna bebida gaseosa donde las burbujas de aire suben.

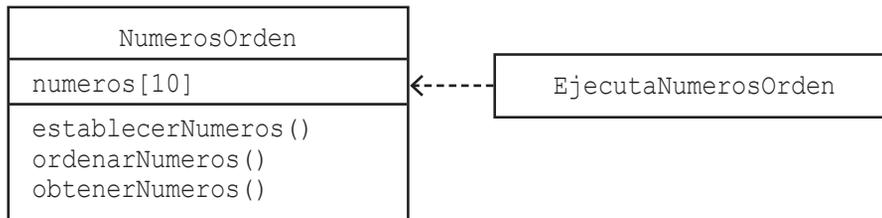
Ejemplo:

Elaborar un algoritmo que permita leer 10 números enteros en un arreglo de 10 elementos, imprima el arreglo, luego clasifique sus elementos en orden ascendente y lo imprima.

A continuación se tiene el algoritmo de la solución:

(Primero hágalo usted; después compare la solución)

Diagrama de clases



Algoritmo CLASIFICA (ORDENA) ARREGLO DE NUMEROS

Clase NumerosOrden

1. Declarar datos
 - numeros: Arreglo[10] Entero
2. Método establecerNumeros(num[]: Entero)
 - a. numeros = num
 - b. Fin Método establecerNumeros
3. Método ordenarNumeros()
 - a. Declarar variables
 - auxiliar, c, bandera: Entero
 - b. do
 1. bandera = 0
 2. for c=1; c<=9; c++
 - a. if numeros[c-1] > numeros[c] then
 1. auxiliar = numeros[c]
 2. numeros[c] = numeros[c-1]
 3. numeros[c-1] = auxiliar
 4. bandera = 1
 - b. endif
 3. endfor
 - c. while bandera != 0
 - d. Fin Método ordenarNumeros
4. Método obtenerNumeros(): Arreglo[] Entero
 - a. return numeros
 - b. Fin Método obtenerNumeros

Fin Clase NumerosOrden

```

Clase EjecutaNumerosOrden
1. Método principal()
  a. Declarar variables
     nums, numsOrdenado: Arreglo[10] Entero
     i: Entero
  b. Declarar, crear e iniciar objeto
     NumerosOrden objNumeros = new NumerosOrden()
  c. for i=0; i<=9; i++
     1. Solicitar nums[i]
     2. Leer nums[i]
  d. endfor
  e. for i=0; i<=9; i++
     1. Imprimir nums[i]
  f. endfor
  g. Establecer
     objNumeros.establecerNumeros(nums)
  h. Clasificar objNumeros.ordenarNumeros()
  i. Obtener
     numsOrdenado = objNumeros.obtenerNumeros()
  j. for i=0; i<=9; i++
     1. Imprimir numsOrdenado[i]
  k. endfor
  l. Fin Método principal
Fin Clase EjecutaNumerosOrden
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: NumerosOrden.java y EjecutaNumerosOrden.java

Explicación:

El algoritmo tiene dos clases: la Clase NumerosOrden y la Clase EjecutaNumerosOrden.

En la Clase NumerosOrden:

1. Se declaran los datos que representan la estructura de la clase: numeros para el arreglo de números.
2. Método establecerNumeros(num[]: Entero).
Recibe en el parámetro num los valores que luego coloca en el dato numeros.
3. Método ordenarNumeros().
 - a. Se hacen las declaraciones de variables.
 - b. Inicia ciclo de clasificación do:
 1. Se pone la bandera en 0.
 2. Inicia ciclo for desde c = 1 hasta 9.
 - a. Si el valor del elemento numeros[c-1] > numeros[c] entonces:
 1. En auxiliar se coloca el elemento numeros[c].

2. En `numeros[c]` se coloca el valor del elemento `numeros[c-1]`.
 3. En `numeros[c-1]` se coloca el valor de `auxiliar`.
En los pasos 1, 2 y 3 se intercambian los elementos `numeros[c-1]` y `numeros[c]`, es decir, se ordenan.
 4. Se coloca la bandera en 1 (se apaga la bandera), lo cual indica que se ha hecho un intercambio; por tanto, el arreglo todavía no ha quedado ordenado.
 - b. Fin del `if`.
 3. Fin del `for`.
 - c. Fin del `do while` `bandera != 0`. Cuando en este punto la bandera esté en 0, significa que el arreglo ya está ordenado.
 - d. Fin Método `ordenarNumeros`.
 4. Método `obtenerNumeros(): Arreglo[] Entero`.
Retorna `numeros`.
- Fin de la Clase `NumerosOrden`.

En la Clase `EjecutaNumerosOrden`, en el Método `principal()`:

- a. Se declaran las variables:
`nums` para leer el arreglo de números.
`numsOrdenado` para recibir los números del arreglo del objeto ordenado.
`i` para controlar el ciclo repetitivo.
 - b. Se declara el objeto `objNumeros`, usando como base la clase `NumerosOrden`.
Dicho objeto se crea e inicializa mediante el constructor por defecto `Numero-
sOrden()`.
 - c. Inicia ciclo `for` desde `i=0` hasta 9 con incrementos de 1.
 1. Solicita `nums[i]`.
 2. Lee en `nums[i]`.
 - d. Fin del ciclo `for`.
 - e. Inicia ciclo `for` desde `i=0` hasta 9 con incrementos de 1.
 1. Imprime cada número de `nums[i]`; imprime el arreglo de números leído.
 - f. Fin del ciclo `for`.
 - g. Se llama al Método `establecerNumeros(nums)` del objeto `objNumeros` para colocar el valor de `nums` en el dato `numeros`.
 - h. Se llama al Método `ordenarNumeros()` del objeto `objNumeros` para ordenar el arreglo de números.
 - i. Llama al Método `obtenerNumeros()` del objeto `objNumeros` para obtener el dato `numeros` y colocarlo en `numsOrdenado`, que es como se procesará de aquí en adelante en este método.
 - j. Inicia ciclo `for` desde `i=0` hasta 9 con incrementos de 1.
 1. Imprime cada número de `numsOrdenado[i]`; imprime el arreglo de números ordenado.
 - k. Fin del ciclo `for`.
 - l. Fin del Método `principal`.
- Fin de la Clase `EjecutaNumerosOrden`.
Fin del algoritmo.

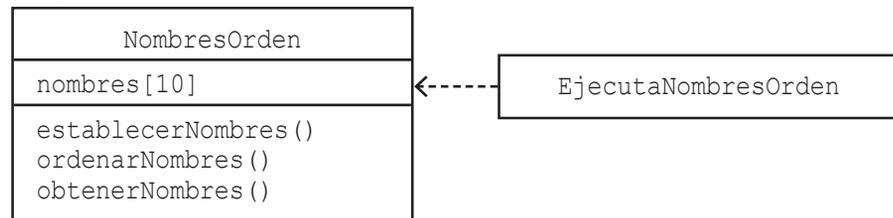
Otro ejemplo:

Elaborar un algoritmo similar al anterior, excepto que ahora se deberán leer y clasificar 10 nombres de personas.

A continuación se tiene el algoritmo de la solución:

(Primero hágalo usted; después compare la solución)

Diagrama de clases



Algoritmo CLASIFICA (ORDENA) ARREGLO DE NOMBRES

Clase NombresOrden

1. Declarar datos

nombres: Arreglo[10] Cadena

2. Método establecerNombres(nom[]: Cadena)

a. nombres = nom

b. Fin Método establecerNombres

3. Método ordenarNombres()

a. Declarar variables

auxiliar: Cadena

c, bandera: Entero

b. do

1. bandera = 0

2. for c=1; c<=9; c++

a. if nombres[c-1] > nombres[c] then

1. auxiliar = nombres[c]

2. nombres[c] = nombres[c-1]

3. nombres[c-1] = auxiliar

4. bandera = 1

b. endif

3. endfor

c. while bandera != 0

d. Fin Método ordenarNombres

4. Método obtenerNombres(): Arreglo[] Cadena

a. return nombres

b. Fin Método obtenerNombres

Fin Clase NombresOrden

```

Clase EjecutaNombresOrden
1. Método principal()
  a. Declarar variables
     noms, nomsOrdenado: Arreglo[10] Cadena
     i: Entero
  b. Declarar, crear e iniciar objeto
     NombresOrden objNombres = new NombresOrden()
  c. for i=0; i<=9; i++
     1. Solicitar noms[i]
     2. Leer noms[i]
  d. endfor
  e. for i=0; i<=9; i++
     1. Imprimir noms[i]
  f. endfor
  g. Establecer
     objNombres.establecerNombres(noms)
  h. Ordenar objNombres.ordenarNombres()
  i. Obtener
     nomsOrdenado = objNombres.obtenerNombres()
  j. for i=0; i<=9; i++
     1. Imprimir nomsOrdenado[i]
  k. endfor
  l. Fin Método principal
Fin Clase EjecutaNombresOrden
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: NombresOrden.java y EjecutaNombresOrden.java



Explicación:

El algoritmo tiene dos clases: la Clase NombresOrden y la Clase EjecutaNombresOrden.

En la Clase NombresOrden:

1. Se declaran los datos que representan la estructura de la clase: nombres para el arreglo de nombres.
2. Método establecerNombres(nom[]: Cadena).
Recibe en el parámetro nom los valores que luego coloca en el dato nombres.
3. Método ordenarNombres().
 - a. Se hacen las declaraciones de variables.
 - b. Inicia ciclo de clasificación do:
 1. Se pone la bandera en 0.
 2. Inicia ciclo for desde c = 1 hasta 9.
 - a. Si el valor del elemento nombres[c-1] > nombres[c] entonces:
 1. En auxiliar se coloca el elemento nombres[c].
 2. En nombres[c] se coloca el valor del elemento nombres[c-1].
 3. En nombres[c-1] se coloca el valor de auxiliar.

En los pasos 1, 2 y 3 se intercambian los elementos `nombres[c-1]` y `nombres[c]`, es decir, se ordenan.

4. Se coloca la bandera en 1 (se apaga la bandera).
 - b. Fin del if.
 3. Fin del for.
 - c. Fin del do while `bandera != 0`. Cuando en este punto la bandera esté en 0, significa que el arreglo ya está ordenado.
 - d. Fin Método `ordenarNombres`.
 4. Método `obtenerNombres(): Arreglo[] Cadena`.
Retorna `nombres`.
- Fin de la Clase `NombresOrden`.

En la Clase `EjecutaNombresOrden`, en el Método `principal()`:

- a. Se declaran las variables:
`noms` para leer el arreglo de nombres.
`nomsOrdenado` para recibir los nombres del arreglo del objeto ordenado.
`i` para controlar el ciclo repetitivo.
 - b. Se declara el objeto `objNombres`, usando como base la Clase `NombresOrden`. Dicho objeto se crea e inicializa mediante el constructor por defecto `NombresOrden()`.
 - c. Inicia ciclo for desde `i=0` hasta 9 con incrementos de 1.
 1. Solicita `noms[i]`.
 2. Lee en `noms[i]`.
 - d. Fin del ciclo for.
 - e. Inicia ciclo for desde `i=0` hasta 9 con incrementos de 1.
 1. Imprime cada nombre de `noms[i]`; imprime el arreglo de nombres leído.
 - f. Fin del ciclo for.
 - g. Se llama al Método `establecerNombres(noms)` del objeto `objNombres` para colocar el valor de `noms` en el dato `nombres`.
 - h. Se llama al Método `ordenarNombres()` del objeto `objNombres` para ordenar el arreglo de nombres.
 - i. Llama al Método `obtenerNombres()` del objeto `objNombres` para obtener el dato `nombres` y colocarlo en `nomsOrdenado`, que es como se procesará de aquí en adelante en este método.
 - j. Inicia ciclo for desde `i=0` hasta 9 con incrementos de 1.
 1. Imprime cada nombre de `nomsOrdenado[i]`; imprime el arreglo de nombres ordenado.
 - k. Fin del ciclo for.
 - l. Fin del Método `principal`.
- Fin de la Clase `EjecutaNombresOrden`.
Fin del algoritmo.

Clasificación (ordenación) de un archivo

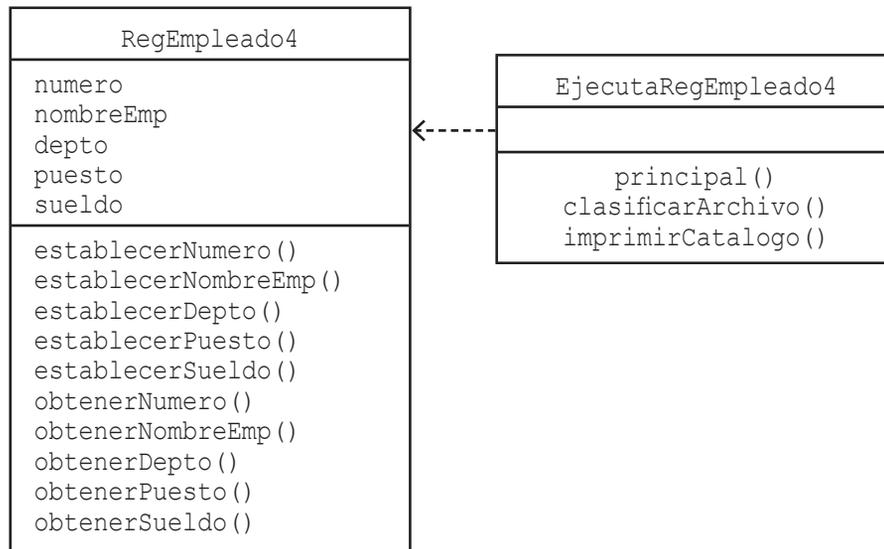
Ahora aplicaremos el método de clasificación en un archivo.

Ejemplo:

Se tiene el archivo de empleados (arEmp2.dat) que fue creado en el capítulo 15 en el punto de proceso de un archivo directo, el cual está ordenado por número de empleado. Se requiere tomar los registros de dicho archivo y crear otro archivo (arEmp2.cla), donde se coloquen en orden secuencial ascendente por nombre, y luego imprimir el catálogo de empleados clasificado por nombre, tomando los datos de dicho archivo.

A continuación se presenta el algoritmo de la solución en dos pasos: primero el diagrama de clases y después el algoritmo en pseudocódigo.

Diagrama de clases



Algoritmo CLASIFICA ARCHIVO DE EMPLEADOS POR NOMBRE

Clase RegEmpleado4

1. Declarar datos

```

numero: Entero
nombreEmp: Cadena
depto : Entero
puesto: Entero
sueldo: Real
  
```

2. Método establecerNumero(num: Entero)

```

a. numero = num
b. Fin Método establecerNumero
  
```

```
3. Método establecerNombreEmp(nom: Cadena)
  a. nombreEmp = nom
  b. Fin Método establecerNombreEmp

4. Método establecerDepto(dep: Entero)
  a. depto = dep
  b. Fin Método establecerDepto

5. Método establecerPuesto(pue: Entero)
  a. puesto = pue
  b. Fin Método establecerPuesto

6. Método establecerSueldo(sue: Real)
  a. sueldo = sue
  b. Fin Método establecerSueldo

7. Método obtenerNumero(): Entero
  a. return numero
  b. Fin Método obtenerNumero

8. Método obtenerNombreEmp(): Cadena
  a. return nombreEmp
  b. Fin Método obtenerNombreEmp

9. Método obtenerDepto(): Entero
  a. return depto
  b. Fin Método obtenerDepto

10. Método obtenerPuesto(): Entero
  a. return puesto
  b. Fin Método obtenerPuesto

11. Método obtenerSueldo(): Real
  a. return sueldo
  b. Fin Método obtenerSueldo
Fin Clase RegEmpleado4

Clase EjecutaRegEmpleado4
1. Método principal()
  a. clasificarArchivo()
  b. imprimirCatalogo()
  c. Fin Método principal

2. Método clasificarArchivo()
  a. Declarar variables
     nombres, direcciones: Arreglo[100] Cadena
     auxiliar: Cadena
     i, c, bandera, posicion, auxiliar2: Entero
  b. Abrir (empleados, "C:/arEmp2.dat", Directo, "rw")
  c. posicion = empleados.PosApuntador()
```

d. Declarar, crear e iniciar objeto

```
RegEmpleado4 objEmpleado = new RegEmpleado4()
```

e. Leer (empleados, objEmpleado)

```
objEmpleado.establecerNumero(empleados.LeerEntero())
objEmpleado.establecerNombreEmp(empleados.LeerCadena())
objEmpleado.establecerDepto(empleados.LeerEntero())
objEmpleado.establecerPuesto(empleados.LeerEntero())
objEmpleado.establecerSueldo(empleados.LeerReal())
```

f. $i = -1$

g. while objEmpleado.obtenerNombreEmp() != "FIN"

```
1.  $i = i + 1$ 
2. nombres[i] = objEmpleado.obtenerNombreEmp()
3. direcciones[i] = posicion
4. posicion = empleados.PosApuntador()
5. objEmpleado = new RegEmpleado4()
6. Leer (empleados, objEmpleado)
   objEmpleado.establecerNumero(empleados.LeerEntero())
   objEmpleado.establecerNombreEmp(empleados.LeerCadena())
   objEmpleado.establecerDepto(empleados.LeerEntero())
   objEmpleado.establecerPuesto(empleados.LeerEntero())
   objEmpleado.establecerSueldo(empleados.LeerReal())
```

h. endwhile

i. do

```
1. bandera = 0
2. for c=1; c<=i; c++
   a. if nombres[c-1] > nombres[c] then
      1. auxiliar = nombres[c]
      2. auxiliar2 = direcciones[c]
      3. nombres[c] = nombres[c-1]
      4. direcciones[c] = direcciones[c-1]
      5. nombres[c-1] = auxiliar
      6. direcciones[c-1] = auxiliar2
      7. bandera = 1
   b. endif
3. endfor
```

j. while bandera != 0

k. Crear (empleados2, "C:/arEmp2.cla", Directo, "rw")

l. for c=0; c<=i; c++

```
1. empleados.encontrar(direcciones[c])
2. Crear e iniciar objeto
   objEmpleado = new RegEmpleado4()
3. Leer (empleados, objEmpleado)
   objEmpleado.establecerNumero(empleados.LeerEntero())
   objEmpleado.establecerNombreEmp(empleados.LeerCadena())
   objEmpleado.establecerDepto(empleados.LeerEntero())
   objEmpleado.establecerPuesto(empleados.LeerEntero())
   objEmpleado.establecerSueldo(empleados.LeerReal())
```

```
4. Imprimir (empleados2, objEmpleado)
    empleados2.Imprimir(objEmpleado.obtenerNumero())
    empleados2.Imprimir(objEmpleado.obtenerNombreEmp())
    empleados2.Imprimir(objEmpleado.obtenerDepto())
    empleados2.Imprimir(objEmpleado.obtenerPuesto())
    empleados2.Imprimir(objEmpleado.obtenerSueldo())
m. endfor
n. Crear e iniciar objeto
    objEmpleado = new RegEmpleado4()
o. Establecer
    objEmpleado.establecerNumero(0)
    objEmpleado.establecerNombreEmp("FIN")
    objEmpleado.establecerDepto(0)
    objEmpleado.establecerPuesto(0)
    objEmpleado.establecerSueldo(0)
p. Imprimir (empleados2, objEmpleado)
    empleados2.Imprimir(objEmpleado.obtenerNumero())
    empleados2.Imprimir(objEmpleado.obtenerNombreEmp())
    empleados2.Imprimir(objEmpleado.obtenerDepto())
    empleados2.Imprimir(objEmpleado.obtenerPuesto())
    empleados2.Imprimir(objEmpleado.obtenerSueldo())
q. Cerrar (empleados)
r. Cerrar (empleados2)
s. Fin Método clasificarArchivo

3. Método imprimirCatalogo()
a. Declarar variables
    totEmpleados: Entero
    totSueldos: Real
b. Abrir (empleados2, "C:/arEmp2.cla", Directo, "rw")
c. totEmpleados = 0
    totSueldos = 0
d. Imprimir encabezado
e. Declarar, crear e iniciar objeto
    RegEmpleado4 objEmpleado = new RegEmpleado4()
f. Leer (empleados2, objEmpleado)
    objEmpleado.establecerNumero(empleados2.LeerEntero())
    objEmpleado.establecerNombreEmp(empleados2.LeerCadena())
    objEmpleado.establecerDepto(empleados2.LeerEntero())
    objEmpleado.establecerPuesto(empleados2.LeerEntero())
    objEmpleado.establecerSueldo(empleados2.LeerReal())
g. while objEmpleado.obtenerNombreEmp() != "FIN"
    1. Imprimir
        objEmpleado.obtenerNumero()
        objEmpleado.obtenerNombreEmp()
        objEmpleado.obtenerDepto()
        objEmpleado.obtenerPuesto()
        objEmpleado.obtenerSueldo()
    2. totEmpleados = totEmpleados + 1
    3. totSueldos = totSueldos + objEmpleado.obtenerSueldo()
```

```

4. Crear e iniciar objeto
   objEmpleado = new EmpleadoReg4()
5. Leer (empleados2, objEmpleado)
   objEmpleado.establecerNumero(empleados2.LeerEntero())
   objEmpleado.establecerNombreEmp(empleados2.LeerCadena())
   objEmpleado.establecerDepto(empleados2.LeerEntero())
   objEmpleado.establecerPuesto(empleados2.LeerEntero())
   objEmpleado.establecerSueldo(empleados2.LeerReal())

h. endwhile
i. Imprimir totEmpleados, totSueldos
j. Cerrar (empleados2)
k. Fin Método imprimirCatalogo
Fin Clase EjecutaRegEmpleado4
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: RegEmpleado4.java y EjecutaRegEmpleado4.java



Explicación:

Se utiliza el archivo “arEmp2.dat”, el cual fue creado en el capítulo 15, y se crea un nuevo archivo “arEmp2.cla”, el cual contendrá los registros clasificados por nombre.

Para hacer la clasificación se utilizan dos arreglos: *nombres* y *direcciones*. Se lee el primer registro del archivo “arEmp2.dat” y se coloca el nombre en el arreglo *nombres*; por otro lado, el número de byte del archivo en el que se encuentra se coloca en el arreglo *direcciones*, es decir, se colocan en los arreglos el nombre y la dirección que ocupa en el archivo original. Esto se hace para todos los registros del archivo. Los arreglos se definieron de 100 elementos, porque se está suponiendo que el archivo no tiene más de 100 elementos. Si se hace un algoritmo para clasificar un archivo de alrededor de 400 elementos, los arreglos deben definirse de un poco más, por ejemplo, de 500.

Se hace el ordenamiento (clasificación) de los datos de los arreglos, se va moviendo el nombre en el arreglo *nombres* y la dirección correspondiente en el arreglo *direcciones*.

Una vez que los datos han quedado ordenados en los arreglos, se procede a crear el nuevo archivo “arEmp2.cla”, luego se toma la primera dirección del arreglo *direcciones* y se lee ese componente del archivo original “arEmp2.dat”. Posteriormente se graban los datos del registro en el nuevo archivo “arEmp2.cla”, se toma la siguiente dirección y se hace lo propio, y así sucesivamente hasta que se terminen las direcciones.

Después de lo anterior tenemos el nuevo archivo “arEmp2.cla” con los datos de los empleados ordenados por nombre.

Por último se procede a enlistar los registros del nuevo archivo y se emite el catálogo de empleados clasificado por nombre.

Clasificación de un archivo por dos datos concatenados

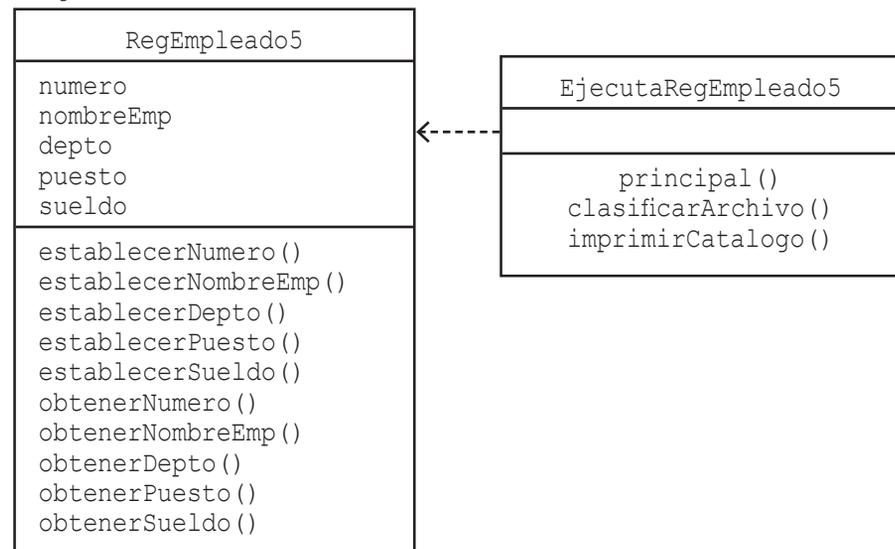
Ahora aplicaremos el método de clasificación en un archivo por dos datos concatenados.

Ejemplo:

Se tiene el archivo de empleados (arEmp2.dat) que fue creado en el capítulo 15 en el punto de proceso de un archivo directo, el cual está ordenado por número de empleado. Se requiere tomar los registros de dicho archivo y crear otro archivo (arEmp3.cla), donde se coloquen en orden secuencial ascendente por departamento y por nombre, y luego imprimir el catálogo de empleados clasificado por departamento y por nombre. Es decir, se clasifica por departamento y, dentro de cada departamento, los empleados estarán ordenados por nombre.

A continuación se presenta el algoritmo de la solución en dos pasos: primero el diagrama de clases y después el algoritmo en pseudocódigo.

Diagrama de clases



Algoritmo CLASIFICA ARCHIVO DE EMPLEADOS POR DEPTO. Y NOMBRE

Clase RegEmpleado5

1. Declarar datos

```

numero: Entero
nombreEmp: Cadena
depto : Entero
puesto: Entero
sueldo: Real
  
```

```
2. Método establecerNumero(num: Entero)
  a. numero = num
  b. Fin Método establecerNumero

3. Método establecerNombreEmp(nom: Cadena)
  a. nombreEmp = nom
  b. Fin Método establecerNombreEmp

4. Método establecerDepto(dep: Entero)
  a. depto = dep
  b. Fin Método establecerDepto

5. Método establecerPuesto(pue: Entero)
  a. puesto = pue
  b. Fin Método establecerPuesto

6. Método establecerSueldo(sue: Real)
  a. sueldo = sue
  b. Fin Método establecerSueldo

7. Método obtenerNumero(): Entero
  a. return numero
  b. Fin Método obtenerNumero

8. Método obtenerNombreEmp(): Cadena
  a. return nombreEmp
  b. Fin Método obtenerNombreEmp

9. Método obtenerDepto(): Entero
  a. return depto
  b. Fin Método obtenerDepto

10. Método obtenerPuesto(): Entero
  a. return puesto
  b. Fin Método obtenerPuesto

11. Método obtenerSueldo(): Real
  a. return sueldo
  b. Fin Método obtenerSueldo
Fin Clase RegEmpleado5

Clase EjecutaRegEmpleado5
1. Método principal()
  a. clasificarArchivo()
  b. imprimirCatalogo()
  c. Fin Método principal

2. Método clasificarArchivo()
  a. Declarar variables
     deptosNombres, direcciones: Arreglo[100] Cadena
     auxiliar: Cadena
     i, c, bandera, posicion, auxiliar2: Entero
```

```

b. Abrir (empleados, "C:/arEmp2.dat", Directo, "rw")
c. posicion = empleados.PosApuntador()
d. Declarar, crear e iniciar objeto
   RegEmpleado5 objEmpleado = new RegEmpleado5()
e. Leer (empleados, objEmpleado)
   objEmpleado.establecerNumero (empleados.LeerEntero())
   objEmpleado.establecerNombreEmp (empleados.LeerCadena())
   objEmpleado.establecerDepto (empleados.LeerEntero())
   objEmpleado.establecerPuesto (empleados.LeerEntero())
   objEmpleado.establecerSueldo (empleados.LeerReal())
f. i = -1
g. while objEmpleado.obtenerNombreEmp() != "FIN"
   1. i = i + 1
   2. deptosNombres[i]= objEmpleado.obtenerDepto() +
                        objEmpleado.obtenerNombreEmp()
   3. direcciones[i] = posicion
   4. posicion = empleados.PosApuntador()
   5. objEmpleado = new RegEmpleado5()
   6. Leer (empleados, objEmpleado)
      objEmpleado.establecerNumero (empleados.LeerEntero())
      objEmpleado.establecerNombreEmp (empleados.LeerCadena())
      objEmpleado.establecerDepto (empleados.LeerEntero())
      objEmpleado.establecerPuesto (empleados.LeerEntero())
      objEmpleado.establecerSueldo (empleados.LeerReal())
h. endwhile
i. do
   1. bandera = 0
   2. for c=1; c<=i; c++
      a. if deptosNombres[c-1]>deptosNombres[c] then
         1. auxiliar = deptosNombres[c]
         2. auxiliar2 = direcciones[c]
         3. deptosNombres[c] = deptosNombres[c-1]
         4. direcciones[c] = direcciones[c-1]
         5. deptosNombres[c-1] = auxiliar
         6. direcciones[c-1] = auxiliar2
         7. bandera = 1
      b. endif
   3. endfor
j. while bandera != 0
k. Crear (empleados2, "C:/arEmp3.cla", Directo, "rw")
l. for c=0; c<=i; c++
   1. empleados.encontrar(direcciones[c])
   2. Crear e iniciar objeto
      objEmpleado = new RegEmpleado5()
   3. Leer (empleados, objEmpleado)
      objEmpleado.establecerNumero (empleados.LeerEntero())
      objEmpleado.establecerNombreEmp (empleados.LeerCadena())
      objEmpleado.establecerDepto (empleados.LeerEntero())
      objEmpleado.establecerPuesto (empleados.LeerEntero())
      objEmpleado.establecerSueldo (empleados.LeerReal())

```

```
4. Imprimir (empleados2, objEmpleado)
    empleados2.Imprimir(objEmpleado.obtenerNumero())
    empleados2.Imprimir(objEmpleado.obtenerNombreEmp())
    empleados2.Imprimir(objEmpleado.obtenerDepto())
    empleados2.Imprimir(objEmpleado.obtenerPuesto())
    empleados2.Imprimir(objEmpleado.obtenerSueldo())
m. endfor
n. Crear e iniciar objeto
    objEmpleado = new RegEmpleado5()
o. Establecer
    objEmpleado.establecerNumero(0)
    objEmpleado.establecerNombreEmp("FIN")
    objEmpleado.establecerDepto(0)
    objEmpleado.establecerPuesto(0)
    objEmpleado.establecerSueldo(0)
p. Imprimir (empleados2, objEmpleado)
    empleados2.Imprimir(objEmpleado.obtenerNumero())
    empleados2.Imprimir(objEmpleado.obtenerNombreEmp())
    empleados2.Imprimir(objEmpleado.obtenerDepto())
    empleados2.Imprimir(objEmpleado.obtenerPuesto())
    empleados2.Imprimir(objEmpleado.obtenerSueldo())
q. Cerrar (empleados)
r. Cerrar (empleados2)
s. Fin Método clasificarArchivo

3. Método imprimirCatalogo()
a. Declarar variables
    totEmpleados: Entero
    totSueldos: Real
b. Abrir (empleados2, "C:/arEmp3.cla", Directo, "rw")
c. totEmpleados = 0
    totSueldos = 0
d. Imprimir encabezado
e. Declarar, crear e iniciar objeto
    RegEmpleado5 objEmpleado = new RegEmpleado5()
f. Leer (empleados2, objEmpleado)
    objEmpleado.establecerNumero(empleados2.LeerEntero())
    objEmpleado.establecerNombreEmp(empleados2.LeerCadena())
    objEmpleado.establecerDepto(empleados2.LeerEntero())
    objEmpleado.establecerPuesto(empleados2.LeerEntero())
    objEmpleado.establecerSueldo(empleados2.LeerReal())
g. while objEmpleado.obtenerNombreEmp() != "FIN"
    1. Imprimir
        objEmpleado.obtenerNumero()
        objEmpleado.obtenerNombreEmp()
        objEmpleado.obtenerDepto()
        objEmpleado.obtenerPuesto()
        objEmpleado.obtenerSueldo()
    2. totEmpleados = totEmpleados + 1
```

```

3. totSueldos = totSueldos +
           objEmpleado.obtenerSueldo()
4. Crear e iniciar objeto
   objEmpleado = new EmpleadoReg5()
5. Leer (empleados2, objEmpleado)
   objEmpleado.establecerNumero(empleados2.LeerEntero())
   objEmpleado.establecerNombreEmp(empleados2.LeerCadena())
   objEmpleado.establecerDepto(empleados2.LeerEntero())
   objEmpleado.establecerPuesto(empleados2.LeerEntero())
   objEmpleado.establecerSueldo(empleados2.LeerReal())

h. endwhile
i. Imprimir totEmpleados, totSueldos
j. Cerrar (empleados2)
k. Fin Método imprimirCatalogo
Fin Clase EjecutaRegEmpleado5
Fin

```



En la zona de descarga de la Web del libro está disponible:
Programa en Java: RegEmpleado5.java y EjecutaRegEmpleado5.java

Explicación:

El algoritmo de clasificación es el mismo que el anterior, excepto que en el Método `clasificarArchivo()`, en lugar de la variable arreglo nombres, se usa `deptosNombres`, en la cual hay un momento en el que se hace lo siguiente:

```

deptosNombres[c]= objEmpleado.obtenerDepto() +
                  objEmpleado.obtenerNombreEmp()

```

En el elemento `c` del arreglo `deptosNombres` se coloca el departamento concatenado con el nombre del empleado. Así, al hacer el ordenamiento, se contempla en primer término el departamento y en segundo término el nombre.

Nota: En este punto se ha dado una idea general de la clasificación y se ha estudiado un método de los más sencillos. Existen varios libros de estructuras de datos que tratan el tema con toda profundidad, lo cual va más allá del alcance de este libro. Sin embargo, es relevante la aplicación en la clasificación del archivo.

16.2 Uso del tipo static

Los datos y métodos que se declaran como estáticos (`static`) son únicos para toda la clase. Esto quiere decir que no pertenecen a ninguna instancia u objeto de la clase, pero pueden ser vistos y utilizados por todas las instancias de la clase. Se describirá su aplicación usando como base el ejemplo del apartado 11.1.1 Contadores y acumuladores.

Ejemplo:

Elaborar un algoritmo que permita procesar varios empleados e imprima un reporte que tenga el formato:



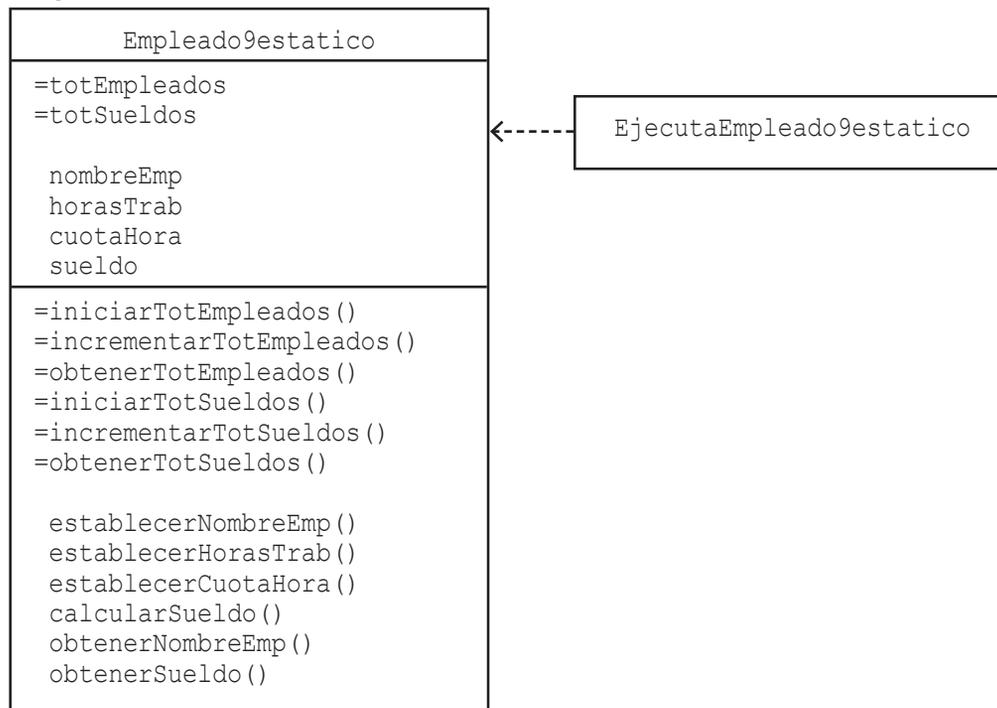
En este punto se ha dado una idea general de la clasificación y se ha estudiado un método de los más sencillos. Existen varios libros de estructuras de datos que tratan el tema con toda profundidad, lo cual va más allá del alcance de este libro. Sin embargo, es relevante la aplicación en la clasificación del archivo.

REPORTE DE EMPLEADOS	
NOMBRE	SUELDO
XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99,999.99
XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99,999.99

XXXXXXXXXXXXXXXXXXXXXXXXXXXX	99,999.99
TOTAL 999 EMPLEADOS	999,999.99

A continuación se presenta el algoritmo de la solución en dos pasos: primero el diagrama de clases y después el algoritmo en pseudocódigo.

Diagrama de clases



Algoritmo CALCULAR SUELDO DE VARIOS EMPLEADOS

Clase Empleado9estatico

1. Declarar datos

Estatico totEmpleados: Entero
Estatico totSueldos: Real

nombreEmp: Cadena
horasTrab: Entero
cuotaHora: Real
sueldo: Real

```
2. Método Estatico iniciarTotEmpleados()
  a. totEmpleados = 0
  b. Fin Método iniciarTotEmpleados

3. Método Estatico incrementarTotEmpleados()
  a. totEmpleados = totEmpleados + 1
  b. Fin Método incrementarTotEmpleados

4. Método Estatico obtenerTotEmpleados(): Entero
  a. return totEmpleados
  b. Fin Método obtenerTotEmpleados

5. Método Estatico iniciarTotSueldos()
  a. totSueldos = 0
  b. Fin Método iniciarTotSueldos

6. Método Estatico incrementarTotSueldos(sdo: Real)
  a. totSueldos = totSueldos + sdo
  b. Fin Método incrementarTotSueldos

7. Método Estatico obtenerTotSueldos(): Real
  a. return totSueldos
  b. Fin Método obtenerTotSueldos

8. Método establecerNombreEmp(nom: Cadena)
  a. nombreEmp = nom
  b. Fin Método establecerNombreEmp

9. Método establecerHorasTrab(horasTr: Entero)
  a. horasTrab = horasTr
  b. Fin Método establecerHorasTrab

10. Método establecerCuotaHora(cuotaHr: Real)
  a. cuotaHora = cuotaHr
  b. Fin Método establecerCuotaHora

11. Método calcularSueldo()
  a. sueldo = horasTrab * cuotaHora
  b. Fin Método calcularSueldo

12. Método obtenerNombreEmp(): Cadena
  a. return nombreEmp
  b. Fin Método obtenerNombreEmp

13. Método obtenerSueldo(): Real
  a. return sueldo
  b. Fin Método obtenerSueldo
Fin Clase Empleado9estatico
```

```

Clase EjecutaEmpleado9estatico
1. Método principal()
  a. Declarar variables
    nomEmp: Cadena
    hrsTra: Entero
    cuoHr: Real
    desea: Carácter
  b. Imprimir encabezado
  c. Iniciar los totales
    Empleado9estatico.iniciarTotEmpleados()
    Empleado9estatico.iniciarTotSueldos()
  d. do
    1. Declarar, crear e iniciar objeto
      Empleado9estatico objEmpleado = new Empleado9estatico()
    2. Solicitar nombre, número de horas trabajadas,
      cuota por hora
    3. Leer nomEmp, hrsTra, cuoHr
    4. Establecer
      objEmpleado.establecerNombreEmp (nomEmp)
      objEmpleado.establecerHorasTrab (hrsTra)
      objEmpleado.establecerCuotaHora (cuoHr)
    5. Calcular objEmpleado.calcularSueldo()
    6. Imprimir objEmpleado.obtenerNombreEmp()
      objEmpleado.obtenerSueldo()
    7. Incrementar totales
      Empleado9estatico.incrementarTotEmpleados()
      Empleado9estatico.incrementarTotSueldos(objEmpleado.obtenerSueldo())
    8. Preguntar "¿Desea procesar otro empleado(S/N)?"
    9. Leer desea
  e. while desea == 'S'
  f. Imprimir Empleado9estatico.obtenerTotEmpleados()
      Empleado9estatico.obtenerTotSueldos()
  g. Fin Método principal
Fin Clase EjecutaEmpleado9estatico
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Empleado9estatico.java y EjecutaEmpleado9estatico.java



Explicación:

El algoritmo que estamos tomando como ejemplo, que desarrollamos en el capítulo 11, consta de las clases `Empleado9` y `EjecutaEmpleado9`, por lo cual en esta solución estamos definiendo las clases `Empleado9estatico` y `EjecutaEmpleado9estatico`. Recordando la solución del capítulo 11, tenemos que en la Clase `EjecutaEmpleado9` se procesan varios empleados; para cada empleado se crea un objeto usando la Clase `Empleado9`, resaltando que el cálculo de los totales de empleados y de sueldos se realiza en la Clase `EjecutaEmpleado9`. Sin embargo, es pertinente enfatizar que el ideal de la programación orientada a objetos es que en

la clase ejecuta no se hagan cálculos, o que se hagan lo menos posible. Es por ello que en este punto se aplica el uso del tipo estático (static) para lograr hacer todos los cálculos en la clase modelo `Empleado9estatico`, y no hacer ningún cálculo en la clase ejecutora `EjecutaEmpleado9estatico`. A continuación se explica lo concerniente al uso de los datos estáticos:

En la Clase `Empleado9estatico`

1. Se declaran los datos
Estatico `totEmpleados: Entero`
Estatico `totSueldos: Real`

Los totales `totEmpleados` y `totSueldos` se declaran como estáticos, lo que significa que no pertenecen a una instancia específica de la clase, sino que son datos que pueden ser utilizados por todos los objetos o instancias creados tomando como base esta clase.

También se declaran los datos:

```
nombreEmp: Cadena
horasTrab: Entero
cuotaHora: Real
sueldo: Real
```

Los datos `nombreEmp`, `horasTrab`, `cuotaHora` y `sueldo` son los datos normales (privados) de la clase, como los que hemos utilizado en los puntos y capítulos anteriores, y que servirán para representar la estructura de los objetos que sean creados con esta clase.

2. Método Estatico `iniciarTotEmpleados()`
 - a. `totEmpleados = 0`
 - b. Fin Método `iniciarTotEmpleados`

Es un método estático que inicia en 0 el dato estático `totEmpleados`.

3. Método Estatico `incrementarTotEmpleados()`
 - a. `totEmpleados = totEmpleados + 1`
 - b. Fin Método `incrementarTotEmpleados`

Es un método estático que incrementa en 1 el dato estático `totEmpleados`.

4. Método Estatico `obtenerTotEmpleados(): Entero`
 - a. `return totEmpleados`
 - b. Fin Método `obtenerTotEmpleados`

Es un método estático que retorna el valor del dato estático `totEmpleados`.

5. Método Estatico iniciarTotSueldos()
 - a. totSueldos = 0
 - b. Fin Método iniciarTotSueldos

Es un método estático que inicia en 0 el dato estático totSueldos.

6. Método Estatico incrementarTotSueldos(sdo: Real)
 - a. totSueldos = totSueldos + sdo
 - b. Fin Método incrementarTotSueldos

Es un método estático que incrementa el dato estático totSueldos con el valor de sdo, que es un parámetro que recibe el valor que se le envía cuando es llamado este método.

7. Método Estatico obtenerTotSueldos(): Real
 - a. return totSueldos
 - b. Fin Método obtenerTotSueldos

Es un método estático que retorna el valor del dato estático totSueldos.

8. El método establecerNombreEmp(nom: Cadena)
9. El método establecerHorasTrab(horasTr: Entero)
10. El método establecerCuotaHora(cuotaHr: Real)
11. El método calcularSueldo()
12. El método obtenerNombreEmp(): Cadena
13. El método obtenerSueldo(): Real

Del paso 8 al 13 son los métodos normales que ya conocemos.

Fin Clase Empleado9estatico

En la Clase EjecutaEmpleado9estatico

1. En el Método principal()
 - a. Se declaran las variables
 - b. Se imprime el encabezado
 - c. Se inician los totales
Empleado9estatico.iniciarTotEmpleados()

Se llama al método estático iniciarTotEmpleados() de la clase Empleado9estatico, que inicia el totEmpleados en 0. Observe que se usa directamente de la clase y no de algún objeto.

```
Empleado9estatico.iniciarTotSueldos()
```

Se llama al método estático `iniciarTotSueldos()` de la clase `Empleado9estatico`, que inicia el `totSueldos` en 0. Observe que se usa directamente de la clase y no de algún objeto.

- d. do Inicia el ciclo do:
1. Se declara, crea e inicia el objeto `objEmpleado`
`Empleado9estatico objEmpleado = new Empleado9estatico()`
 2. Se solicitan nombre, número de horas trabajadas, cuota por hora
 3. Se leen en `nomEmp`, `hrsTra`, `cuoHr`
 4. Se establecen en el objeto los datos leídos
`objEmpleado.establecerNombreEmp(nomEmp)`
`objEmpleado.establecerHorasTrab(hrsTra)`
`objEmpleado.establecerCuotaHora(cuoHr)`
 5. Se llama al método para calcular el sueldo
 6. Se imprimen el nombre y el sueldo
 7. Se incrementan los totales
`Empleado9estatico.incrementarTotEmpleados()`

Se llama al método estático `incrementarTotEmpleados()` de la clase `Empleado9estatico`, que incrementa `totEmpleados` en 1. Observe que se usa directamente de la clase y no de algún objeto.

```
Empleado9estatico.incrementarTotSueldos  
(objEmpleado.obtenerSueldo())
```

Se llama al método estático `incrementarTotSueldos()` de la clase `Empleado9estatico`, que incrementa el `totSueldos` con el sueldo, el cual se obtiene del objeto con `objEmpleado.obtenerSueldo()` y se envía como parámetro. Observe que se usa directamente de la clase y no de algún objeto.

8. Pregunta "¿Desea procesar otro empleado(S/N)?"
 9. Lee la respuesta en `desea`
- e. while Cierre del ciclo do.
- f. Imprime los totales
`Empleado9estatico.obtenerTotEmpleados()`

Se llama al método estático `obtenerTotEmpleados()` de la clase `Empleado9estatico`, que retorna el valor de `totEmpleados` para imprimirlo. Observe que se usa directamente de la clase y no de algún objeto.

```
Empleado9estatico.obtenerTotSueldos()
```

Se llama al método estático `obtenerTotSueldos()` de la clase `Empleado9estatico`, que retorna el valor de `totSueldos` para imprimirlo. Observe que se usa directamente de la clase y no de algún objeto.

```
g. Fin Método principal
Fin Clase EjecutaEmpleado9estatico
Fin
```

En otras palabras, al declarar los datos `totEmpleados` y `totSueldos` como estáticos en la clase `Empleado9estatico`, pueden ser usados por todos los objetos creados con esta clase y así, por ejemplo, el dato sueldo de cada objeto `objEmpleado`, creado en la clase `EjecutaEmpleado9estatico`, se envía a la clase `Empleado9estatico` para que se incremente el dato `totSueldos`. De esa forma se logra calcular el total de sueldos en la clase modelo `Empleado9estatico` y no en la clase ejecutora `EjecutaEmpleado9estatico`.

Otro ejemplo:

Tomando como base el ejercicio 11.1.2.1 que desarrollamos en el capítulo 11, que procesa varios clientes que compran hojas de hielo seco, vamos a trabajar usando el tipo `static` (estático).

A continuación se tiene el diagrama de clases:

Diagrama de clases



A continuación se tiene el algoritmo en pseudocódigo:

```
Algoritmo CLIENTES HOJAS HIELO SECO
Clase Cliente3estatico
1. Declarar datos
    Estatico totClientes: Entero
    Estatico totSubTotal, totDescuento, totNetoPagar: Real

    nombreClie: Cadena
    tipoClie, cantidad: Entero
    precioUni, subTotal, descuento, netoPagar: Real

2. Método Estatico iniciarTotClientes()
    a. totClientes = 0
    b. Fin Método iniciarTotClientes

3. Método Estatico incrementarTotClientes()
    a. totClientes = totClientes + 1
    b. Fin Método incrementarTotClientes

4. Método Estatico obtenerTotClientes(): Entero
    a. return totClientes
    b. Fin Método obtenerTotClientes

5. Método Estatico iniciarTotSubTotal()
    a. totSubTotal = 0
    b. Fin Método iniciarTotSubTotal

6. Método Estatico incrementarTotSubTotal(suTo: Real)
    a. totSubTotal = totSubTotal + suTo
    b. Fin Método incrementarTotSubTotal

7. Método Estatico obtenerTotSubTotal(): Real
    a. return totSubTotal
    b. Fin Método obtenerTotSubTotal

8. Método Estatico iniciarTotDescuento()
    a. totDescuento = 0
    b. Fin Método iniciarTotDescuento

9. Método Estatico incrementarTotDescuento(desc: Real)
    a. totDescuento = totDescuento + desc
    b. Fin Método incrementarTotDescuento

10. Método Estatico obtenerTotDescuento(): Real
    a. return totDescuento
    b. Fin Método obtenerTotDescuento
```

11. Método Estatico iniciarTotNetoPagar()
 - a. totNetoPagar = 0
 - b. Fin Método iniciarTotNetoPagar
12. Método Estatico incrementarTotNetoPagar(nePag: Real)
 - a. totNetoPagar = totNetoPagar + nePag
 - b. Fin Método incrementarTotNetoPagar
13. Método Estatico obtenerTotNetoPagar(): Real
 - a. return totNetoPagar
 - b. Fin Método obtenerTotNetoPagar
14. Método establecerNombreClie(nom: Cadena)
 - a. nombreClie = nom
 - b. Fin Método establecerNombreClie
15. Método establecerTipoClie(tip: Entero)
 - a. tipoClie = tip
 - b. Fin Método establecerTipoClie
16. Método establecerCantidad(can: Entero)
 - a. cantidad = can
 - b. Fin Método establecerCantidad
17. Método establecerPrecioUni(pre: Real)
 - a. precioUni = pre
 - b. Fin Método establecerPrecioUni
18. Método calcularSubTotal()
 - a. subTotal = cantidad * precioUni
 - b. Fin Método calcularSubTotal
19. Método calcularDescuento()
 - a. switch tipoClie
 - 1: descuento = subTotal * 0.05
 - 2: descuento = subTotal * 0.08
 - 3: descuento = subTotal * 0.12
 - 4: descuento = subTotal * 0.15
 - b. endswitch
 - c. Fin Método calcularDescuento
20. Método calcularNetoPagar()
 - a. netoPagar = subTotal - descuento
 - b. Fin Método calcularNetoPagar
21. Método obtenerNombreClie(): Cadena
 - a. return nombreClie
 - b. Fin Método obtenerNombreClie

22. Método obtenerSubTotal(): Real
 - a. return subTotal
 - b. Fin Método obtenerSubTotal
 23. Método obtenerDescuento(): Real
 - a. return descuento
 - b. Fin Método obtenerDescuento
 24. Método obtenerNetoPagar(): Real
 - a. return netoPagar
 - b. Fin Método obtenerNetoPagar
- Fin Clase Cliente3estatico

Clase EjecutaCliente3estatico

1. Método principal()
 - a. Declarar variables
 - nomCli: Cadena
 - tiCli, cant: Entero
 - preUni: Real
 - desea: Carácter
 - b. Imprimir encabezado
 - c. Iniciar los totales
 - Cliente3estatico.iniciarTotClientes()
 - Cliente3estatico.iniciarTotSubTotal()
 - Cliente3estatico.iniciarTotDescuento()
 - Cliente3estatico.iniciarTotNetoPagar()
 - d. do
 1. Declarar, crear e iniciar objeto
 - Cliente3estatico objCliente = new Cliente3estatico()
 2. Solicitar nombre, tipo cliente, cantidad, precio unitario
 3. Leer nomCli, tiCli, cant, preUni
 4. Establecer
 - objCliente.establecerNombreClie(nomCli)
 - objCliente.establecerTipoClie(tiCli)
 - objCliente.establecerCantidad(cant)
 - objCliente.establecerPrecioUni(preUni)
 5. Calcular objCliente.calcularSubTotal()
 - objCliente.calcularDescuento()
 - objCliente.calcularNetoPagar()
 6. Imprimir objEmpleado.obtenerNombreClie()
 - objEmpleado.obtenerSubTotal()
 - objEmpleado.obtenerDescuento()
 - objEmpleado.obtenerNetoPagar()
 7. Incrementar totales
 - Cliente3estatico.incrementarTotClientes()
 - Cliente3estatico.incrementarTotSubTotal(objCliente.obtenerSubTotal())
 - Cliente3estatico.incrementarTotDescuento(objCliente.obtenerDescuento())
 - Cliente3estatico.incrementarTotNetoPagar(objCliente.obtenerNetoPagar())

```

            8. Preguntar "¿Desea procesar otro cliente(S/N)?"
            9. Leer desea
    e. while desea == 'S'
    f. Imprimir Cliente3estatico.obtenerTotClientes()
        Cliente3estatico.obtenerTotSubTotal()
        Cliente3estatico.obtenerTotDescuento()
        Cliente3estatico.obtenerTotNetoPagar()
    g. Fin Método principal
    Fin Clase EjecutaCliente3estatico
Fin

```



En la zona de descarga de la Web del libro está disponible:

Programa en Java: Cliente3estatico.java y EjecutaCliente3estatico.java

Explicación:

El algoritmo que estamos tomando como ejemplo, que desarrollamos en el capítulo 11, consta de las clases `Cliente3` y `EjecutaCliente3`, por lo cual en esta solución estamos definiendo las clases `Cliente3estatico` y `EjecutaCliente3estatico`. Recordando la solución del capítulo 11, tenemos que en la Clase `EjecutaCliente3` se procesan varios clientes y para cada cliente se crea un objeto usando la clase `Cliente3`, resaltando que el cálculo de los totales de clientes, de subtotales, de descuentos y de netos por pagar se realizan en la clase `EjecutaCliente3`. Sin embargo, es pertinente enfatizar que el ideal de la programación orientada a objetos es que en la clase ejecuta no se hagan cálculos o que se hagan lo menos posible. Es por ello que en este punto se aplica el uso del tipo estático (`static`) para lograr hacer todos los cálculos en la clase modelo `Cliente3estatico` y no hacer ningún cálculo en la clase ejecutora `EjecutaCliente3estatico`.

Nota: Algunos de los algoritmos que elaboramos en los puntos y capítulos anteriores, desde el 11, pueden ser diseñados utilizando el tipo estático (`static`).

A manera de práctica, se le recomienda que tome algunos de los algoritmos de los capítulos anteriores (desde el 11) y que los elabore incluyendo el uso del tipo estático, como se explicó en este punto.

16.3 Uso del `this`

La palabra `this` se utiliza en la programación orientada a objetos para hacer referencia a elementos de la misma clase. Se describirá su aplicación usando como base el ejemplo del apartado 11.1.1 Contadores y acumuladores, que desarrollamos en el capítulo 11 y que también usamos como base en el punto anterior.

A continuación se presenta el algoritmo de la solución en pseudocódigo:

Algoritmo CALCULAR SUELDO DE VARIOS EMPLEADOS

Clase Empleado9this

1. Declarar datos
 - nombreEmp: Cadena
 - horasTrab: Entero
 - cuotaHora: Real
 - sueldo: Real
2. Método establecerNombreEmp(nombreEmp: Cadena)
 - a. this.nombreEmp = nombreEmp
 - b. Fin Método establecerNombreEmp
3. Método establecerHorasTrab(horasTrab: Entero)
 - a. this.horasTrab = horasTrab
 - b. Fin Método establecerHorasTrab
4. Método establecerCuotaHora(cuotaHora: Real)
 - a. this.cuotaHora = cuotaHora
 - b. Fin Método establecerCuotaHora
5. Método calcularSueldo()
 - a. sueldo = horasTrab * cuotaHora
 - b. Fin Método calcularSueldo
6. Método obtenerNombreEmp(): Cadena
 - a. return nombreEmp
 - b. Fin Método obtenerNombreEmp
7. Método obtenerSueldo(): Real
 - a. return sueldo
 - b. Fin Método obtenerSueldo

Fin Clase Empleado9this

Clase EjecutaEmpleado9this

1. Método principal()
 - a. Declarar variables
 - nombreEmp: Cadena
 - horasTrab, totEmpleados: Entero
 - cuotaHora, totSueldos: Real
 - desea: Carácter
 - b. Imprimir encabezado
 - c. totEmpleados = 0
 - totSueldos = 0
 - d. do
 1. Declarar, crear e iniciar objeto
Empleado9this objEmpleado = new Empleado9this()
 2. Solicitar nombre, número de horas trabajadas,
cuota por hora

```

3. Leer nombreEmp, horasTrab, cuotaHora
4. Establecer
   objEmpleado.establecerNombreEmp(nombreEmp)
   objEmpleado.establecerHorasTrab(horasTrab)
   objEmpleado.establecerCuotaHora(cuotaHora)
5. Calcular objEmpleado.calcularSueldo()
6. Imprimir objEmpleado.obtenerNombreEmp()
   objEmpleado.obtenerSueldo()
7. totEmpleados = totEmpleados + 1
   totSueldos = totSueldos +
       objEmpleado.obtenerSueldo()
8. Preguntar "¿Desea procesar otro empleado(S/N)?"
9. Leer desea
e. while desea == 'S'
f. Imprimir totEmpleados, totSueldos
g. Fin Método principal
Fin Clase EjecutaEmpleado9this
Fin

```



En la zona de descarga de la Web del libro está disponible:

Programa en Java: Empleado9this.java y EjecutaEmpleado9this.java

Explicación:

Este algoritmo ya lo hicimos en el capítulo 11, y recordemos que por cada dato que se lee, por ejemplo el nombre del empleado, se usan tres nombres de variables diferentes: uno a nivel de la clase modelo Empleado9: nombreEmp, otro en el parámetro del método establecerNombreEmp(nom: Cadena): nom, y otro en la clase ejecuta EjecutaEmpleado9: nomEmp. De esa forma lo hicimos en los capítulos anteriores (desde el 9) para evitar confusiones. Ahora lo estamos haciendo con el uso de this. Veamos qué cambia.

En la clase Empleado9this, en el Método establecerNombreEmp(nombreEmp: Cadena), el parámetro tiene el mismo nombre que el dato declarado a nivel de la clase, que es nombreEmp, y en el método tenemos la expresión:

```
a. this.nombreEmp = nombreEmp
```

En donde:

this.nombreEmp Con la palabra this se indica que se refiere al identificador declarado a nivel de la clase.

nombreEmp Es el identificador declarado en el método como parámetro.

Hace lo siguiente: al ser llamado el Método establecerNombreEmp(nombreEmp: Cadena), recibe el valor que se le envía en el parámetro nombreEmp y lo establece o coloca en el dato this.nombreEmp, en el cual this indica que se refiere al identificador declarado a nivel de la clase. De esta forma se puede usar el mismo nombre

tanto a nivel de la clase como en el método y se evita la confusión con el uso de `this`. La idea es que no se puede hacer esto: `a.nombreEmp = nombreEmp`, porque al declarar una variable en el método igual a una ya declarada a nivel de la clase, la del método oculta a la otra. Debido a que en el método sólo puede usarse la variable que fue declarada ahí; esa expresión es incorrecta.

Así, para cada dato que se va a leer, como es el caso del nombre, podemos usar el mismo identificador tanto a nivel de la clase como en el método, y con el `this` los diferenciamos.

Ya que tenemos claro lo anterior, en el método principal de la clase `EjecutaEmpleado9this` podremos utilizar, para representar y leer el nombre, el mismo identificador `nombreEmp` que, como está en otro método de otra clase, aunque tenga el mismo nombre `nombreEmp`, es diferente a los antes mencionados, porque está en diferente contexto.

Asimismo, para el otro dato que se lee en este algoritmo, que es el número de horas trabajadas, se usa el `this` para utilizar el mismo nombre tanto a nivel de la clase como en el método y diferenciarlos, así:

```
3. Método establecerHorasTrab(horasTrab: Entero)
   a. this.horasTrab = horasTrab
```

En consecuencia, en la clase ejecuta, que es donde se lee, también se usa `horasTrab`.

Sucede lo propio para el tercer dato que se lee en este algoritmo, que es la cuota por hora. Se usa el `this` para utilizar el mismo nombre tanto a nivel de la clase como en el método y diferenciarlos, así:

```
4. Método establecerCuotaHora(cuotaHora: Real)
   a. this.cuotaHora = cuotaHora
```

En consecuencia, en la clase ejecuta, que es donde se lee, también se usa `cuotaHora`.

Otro ejemplo:

Tomando como base el ejercicio 11.1.2.1 que desarrollamos en el capítulo 11, el cual procesa varios clientes que compran hojas de hielo seco, ahora vamos a trabajar usando el `this`.

A continuación se tiene el algoritmo en pseudocódigo:

```
Algoritmo CLIENTES HOJAS HIELO SECO
Clase Cliente3this
1. Declarar datos
   nombreClie: Cadena
   tipoClie, cantidad: Entero
   precioUni, subTotal, descuento, netoPagar: Real
```

```
2. Método establecerNombreClie(nombreClie: Cadena)
  a. this.nombreClie = nombreClie
  b. Fin Método establecerNombreClie

3. Método establecerTipoClie(tipoClie: Entero)
  a. this.tipoClie = tipoClie
  b. Fin Método establecerTipoClie

4. Método establecerCantidad(cantidad: Entero)
  a. this.cantidad = cantidad
  b. Fin Método establecerCantidad

5. Método establecerPrecioUni(precioUni: Real)
  a. this.precioUni = precioUni
  b. Fin Método establecerPrecioUni

6. Método calcularSubTotal()
  a. subTotal = cantidad * precioUni
  b. Fin Método calcularSubTotal

7. Método calcularDescuento()
  a. switch tipoClie
    1: descuento = subTotal * 0.05
    2: descuento = subTotal * 0.08
    3: descuento = subTotal * 0.12
    4: descuento = subTotal * 0.15
  b. endswitch
  c. Fin Método calcularDescuento

8. Método calcularNetoPagar()
  a. netoPagar = subTotal - descuento
  b. Fin Método calcularNetoPagar

9. Método obtenerNombreClie(): Cadena
  a. return nombreClie
  b. Fin Método obtenerNombreClie

10. Método obtenerSubTotal(): Real
  a. return subTotal
  b. Fin Método obtenerSubTotal

11. Método obtenerDescuento(): Real
  a. return descuento
  b. Fin Método obtenerDescuento

12. Método obtenerNetoPagar(): Real
  a. return netoPagar
  b. Fin Método obtenerNetoPagar
Fin Clase Cliente3this
```

Clase EjecutaCliente3this

1. Método principal()

a. Declarar variables

```
nombreClie: Cadena
tipoClie, cantidad, totClientes: Entero
precioUni, totSubTot, totDescuento, totNeto: Real
desea: Carácter
```

b. Imprimir encabezado

c. totClientes = 0

```
totSubTot = 0
```

```
totDescuento = 0
```

```
totNeto = 0
```

d. do

1. Declarar, crear e iniciar objeto

```
Cliente3this objCliente = new Cliente3this()
```

2. Solicitar nombre, tipo cliente,

```
cantidad, precio unitario
```

3. Leer nombreClie, tipoClie, cantidad, precioUni

4. Establecer

```
objCliente.establecerNombreClie(nombreClie)
```

```
objCliente.establecerTipoClie(tipoClie)
```

```
objCliente.establecerCantidad(cantidad)
```

```
objCliente.establecerPrecioUni(precioUni)
```

5. Calcular objCliente.calcularSubTotal()

```
objCliente.calcularDescuento()
```

```
objCliente.calcularNetoPagar()
```

6. Imprimir objCliente.obtenerNombreClie()

```
objCliente.obtenerSubTotal()
```

```
objCliente.obtenerDescuento()
```

```
objCliente.obtenerNetoPagar()
```

7. totClientes = totClientes + 1

```
totSubTot = totSubTot + objCliente.obtenerSubTotal()
```

```
totDescuento = totDescuento +
```

```
objCliente.obtenerDescuento()
```

```
totNeto = totNeto + objCliente.obtenerNetoPagar()
```

8. Preguntar "¿Desea procesar otro cliente (S/N)?"

9. Leer desea

e. while desea == 'S'

f. Imprimir totClientes, totSubTot,

```
totDescuento, totNeto
```

g. Fin Método principal

Fin Clase EjecutaCliente3this

Fin

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Cliente3this.java y EjecutaCliente3this.java



Nota: Todos los algoritmos que elaboramos en los apartados y capítulos anteriores (desde el 9) pueden ser diseñados utilizando la palabra this. Desde ahora usted puede hacer sus algoritmos de esta forma si le parece mejor y no le provoca confusión.



Todos los algoritmos que elaboramos en los apartados y capítulos anteriores (desde el 9) pueden ser diseñados utilizando la palabra this. Desde ahora usted puede hacer sus algoritmos de esta forma si le parece mejor y no le provoca confusión.

A manera de práctica, se le recomienda que tome algunos de los algoritmos de los capítulos anteriores (desde el 9) y los elabore incluyendo el uso de this como se explicó en este punto.

16.4 Recursividad

Los lenguajes de programación soportan la recursividad. Ésta significa que un elemento puede definirse en términos de sí mismo. Se implementa este concepto a través de métodos, los cuales pueden llamarse a sí mismos.

Ejemplo:

Elaborar un algoritmo que lea un número e imprima los números desde ese número hasta 1 con decrementos de 1. Utilizar un método recursivo, es decir, que se llame a sí mismo.

```
Algoritmo RECURSIVIDAD1
Clase Recursividad1
1. Método principal()
  a. Declarar variables
     num: Entero
  b. Solicitar un número
  c. Leer num
  d. imprimirNumero(num)
  e. Fin Método principal

2. Método imprimirNumero(Val x: Entero)
  a. Imprimir x
  b. if x > 1 then
     1. x = x - 1
     2. imprimirNumero(x)
  c. endif
  d. Fin Método imprimirNumero
Fin Clase Recursividad1
Fin
```



En la zona de descarga de la Web del libro está disponible:

Programa en Java: Recursividad1.java

Explicación:

En el Método principal() se lee num, luego se llama al Método imprimirNumero(num), enviándole num como parámetro. En el Método imprimirNumero(Val x: Entero) se recibe el parámetro enviado en el parámetro x y se imprime el valor de x. Luego, si x es mayor que 1, le disminuye 1 a x y llama al Método imprimirNumero(x) con el nuevo valor de x como parámetro, es decir, se llama a sí mismo; si x no es mayor a 1, se termina el if y el método, y vuelve al Método principal(). Imprime desde num hasta 1, con decrementos de 1.

Veamos otro ejemplo:

Elaborar un algoritmo que lea un número e imprima los números desde 1 hasta el número con incrementos de 1. Utilizar un método recursivo.

```

Algoritmo RECURSIVIDAD2
Clase Recursividad2
  1. Declarar variables de clase
     num: Entero

  2. Método principal()
     a. Solicitar número
     b. Leer num
     c. imprimirNumero(1)
     d. Fin Método principal

  3. Método imprimirNumero(Val x: Entero)
     a. Imprimir x
     b. if x < num then
         1. x = x + 1
         2. imprimirNumero(x)
     c. endif
     d. Fin Método imprimirNumero
Fin Clase Recursividad2
Fin

```

En la zona de descarga de la Web del libro está disponible:

Programa en Java: Recursividad2.java



Explicación:

En el Método principal() se lee num y luego se llama al Método imprimirNumero(1), enviándole 1 como parámetro. En el Método imprimirNumero(Val x: Entero) se recibe el parámetro enviado en el parámetro x y se imprime el valor de x. Luego, si x es menor que num (el número leído en el otro método), le incrementa 1 a x y llama al Método imprimirNumero(x) con el nuevo valor de x como parámetro, es decir, se llama a sí mismo; si x no es menor que num, se termina el if y el método, y vuelve al Método principal(). Imprime desde 1 hasta num, con incrementos de 1.

16.5 Graficación

Los lenguajes de programación como Pascal, C, C++, Java, C#, etcétera, permiten dibujar figuras geométricas. A esto se le llama graficación.

Al desarrollar un programa, si usted tiene necesidad de graficar, podrá hacerlo con funciones como las siguientes: drawLine, drawRect, drawArc, drawPolygon, drawPie, entre otras, que en los lenguajes se representan en inglés. Sin embargo, en esta metodología, que es en seudocódigo, las vamos a manejar en forma castellanizada, como se muestra a continuación:

Arco

Dibuja un arco circular desde el ángulo inicial hasta el ángulo final, utilizando (x, y) como punto central y el radio. Sintaxis:

```
Arco(x, y, anguloInicial, anguloFinal, radio)
```

Barra

Dibuja una barra rectangular desde x1, y1 hasta x2, y2, utilizando el relleno y color actual. Sintaxis:

```
Barra(x1, y1, x2, y2)
```

Barra3D

Dibuja una barra en tres dimensiones de x1, y1 hasta x2, y2, con la profundidad prof, y tapa indica si lleva o no tapa, utilizando el relleno y color actual. Sintaxis:

```
Barra3D(x1, y1, x2, y2, prof, tapa)
```

Circulo

Dibuja un círculo utilizando (x, y) como punto central. Sintaxis:

```
Circulo(x, y, radio)
```

CierraGraph

Quita de uso el sistema gráfico. Sintaxis:

```
CierraGraph
```

DetectaGraph

Verifica el hardware y determina el manejador y modo de graficación que se va a utilizar. Sintaxis:

```
DetectaGraph(manejadorGraf, modoGraf)
```

DibujaPoli

Dibuja un polígono. Sintaxis:

```
DibujaPoli(numPuntos, puntos)
```

Elipse

Dibuja una elipse desde el ángulo inicial hasta el ángulo final, utilizando (x, y) como punto central. Sintaxis:

```
Elipse(x, y, anguloInicial, anguloFinal, xRadio, yRadio)
```

IniciaGraph

Inicia el sistema de gráficas y coloca el hardware en modo gráfico. Sintaxis: IniciaGraph(manejador, modo, ruta)

Línea

Dibuja una línea desde (x1, y1) hasta (x2, y2). Sintaxis:

Línea (x1, y1, x2, y2)

RebPastel

Dibuja y rellena una rebanada de pastel desde el ángulo inicial hasta el ángulo final.

Sintaxis:

RebPastel (x, y, ánguloInicial, ánguloFinal, radio)

Rectángulo

Dibuja un rectángulo desde el punto (x1, y1) hasta (x2, y2). Sintaxis:

Rectángulo (x1, y1, x2, y2)

Nota: Entre otras funciones que usted puede añadir de acuerdo a las que contenga el manejador gráfico del lenguaje que esté utilizando, por ejemplo: RellenaElipse, RellenaPoli, etcétera.

16.6 Resumen de conceptos que debe dominar

- Clasificación (ordenación) de datos
- Uso del tipo static
- Uso del this
- Recursividad
- Graficación

16.7 Contenido de la página Web de apoyo

El material marcado con asterisco () sólo está disponible para docentes.*

16.7.1 Resumen gráfico del capítulo

16.7.2 Autoevaluación

16.7.3 Programas en Java

16.7.4 Power Point para el profesor (*)

A

Apéndice A: Conclusiones y recomendaciones

Conclusiones y recomendaciones

Con el estudio de la metodología y fundamentos de programación que le presento en este libro, el estudiante aprenderá la lógica de la programación orientada a objetos sin estar “casado” con ningún lenguaje, esto es, utilizando la técnica seudocódigo, que es un seudolenguaje que permite diseñar programas o elaborar algoritmos orientados a objetos.

La idea es que este seudolenguaje sea lo más general posible, para que los algoritmos que se diseñen, puedan traducirse más o menos con la misma facilidad a cualquier lenguaje orientado a objetos. Sin embargo, es imposible que haya la misma distancia entre este seudolenguaje y todos los lenguajes orientados a objetos, por tanto, debe ser un poco más parecido a alguno de los lenguajes, en este caso he escogido que la técnica seudocódigo, sea sutilmente más parecida al lenguaje Java, porque es el lenguaje que prácticamente se ha convertido en el estándar de la programación orientada a objetos.

La intención es que esta metodología pueda ser aprendida en un primer semestre de nivel licenciatura en alrededor de 80 horas de clase, y luego en un segundo semestre recomendando que se estudie un lenguaje orientado a objetos como Java u otro, para implementar la metodología en dicho lenguaje. De hecho, en un futuro no muy lejano, espero poner a su disposición, un siguiente libro que explicará cómo implementar esta metodología en lenguaje Java.

Cuando una persona aprende a programar usando un seudolenguaje como el que le presento en este libro, le será más fácil estar cambiando de lenguaje de programación, simplemente hay que estar añadiendo al seudolenguaje los nuevos conceptos que se van generando a través de los nuevos lenguajes de programación.

Estimado lector, si Usted ya sabe programar en Java, quizás espere ver en esta metodología temas como arreglos de objetos, entre otros; es por ello que quiero comentarle que no los he incluido, porque creo que deben estudiarse en un siguiente curso de programación orientada a objetos en lenguaje Java.

La metodología que le presento en este libro no es restrictiva y no la debemos considerar completamente terminada, usted amigo lector, está en libertad de añadirle palabras y conceptos que usted vea en los lenguajes y que considere necesarios para sus algoritmos (programas), por ejemplo:

- Funciones para manejar interfaz gráfica de usuario como: Button, Label, CheckBox, ComboBox, List, FlowLayout, BorderLayout, GridLayout, Panel, Menu, Frame, etcétera.
- Funciones para trabajar en redes.
- Funciones para manejar imágenes, animación y audio.
- Manipulación de bits.
- Colecciones.
- Conectividad de bases de datos.
- Etcétera.

B

Apéndice B: Cómo evolucionar de la programación estructurada a la programación orientada a objetos

Cómo evolucionar de la programación estructurada a la programación orientada a objetos

Este autor considera que si una persona ya maneja la lógica básica de la programación, ya sea la programación estructurada en pseudocódigo u otra técnica de diseño de algoritmos, no le debe estorbar, sino que ya trae adelantado algo, y el estudio de la programación orientada a objetos, es lo que sigue en forma natural; es decir, que es la siguiente estructura de la programación que debe aprender.

Para estudiantes que ya llevaron un curso de metodología de la programación estructurada y/o un lenguaje como Pascal, C u otro, deben hacer un repaso rápido de los capítulos del 2 al 7, para adaptar los conceptos y estructuras que manejan, con la forma y estilo que se usa en este libro. En esos capítulos se presenta, lo que se estudia en un primer curso de lógica de programación con técnicas estructuradas, pero enfocando la estructura del algoritmo en forma apropiada a la programación orientada a objetos, si antes, un algoritmo nos quedaba de una forma como esta:

```

Algoritmo CALCULA SUELDO DE UN EMPLEADO
1. Declaraciones
   Variables
      NombreEmp: Cadena[30]
      HorasTrab: Entero
      CuotaHora, Sueldo: Real
2. Solicitar nombre del empleado, número de horas
   Trabajadas, cuota por hora
3. Leer NombreEmp, HorasTrab, CuotaHora
4. Calcular Sueldo = HorasTrab * CuotaHora
5. Imprimir NombreEmp, Sueldo
6. Fin

```

Ahora nos queda de la siguiente forma:

```

Algoritmo CALCULA SUELDO DE UN EMPLEADO
Clase Empleado1
1. Método principal()
   a. Declarar variables
      nombreEmp: Cadena
      horasTrab: Entero
      cuotaHora, sueldo: Real
   b. Solicitar nombre del empleado, número de horas
      trabajadas, cuota por hora
   c. Leer nombreEmp, horasTrab, cuotaHora
   d. Calcular sueldo = horasTrab * cuotaHora
   e. Imprimir nombreEmp, sueldo
   f. Fin Método principal
Fin Clase Empleado1
Fin

```

Esto es, usando una clase y dentro de la clase el método principal, que es donde están las acciones del algoritmo.

En el caso de las estructuras de control, en este libro se les está dando un estilo más parecido a lenguaje C y Java, a continuación se presentan los formatos que se utilizan:

La selección simple o “SI ACASO” se representa:

```
if condición then
    Acción(es)
endif
```

La selección doble o “SI ACASO – SINO” se representa:

```
if condición then
    Acción(es)
else
    Acción(es)
endif
```

La selección múltiple o “SEGUN” se representa:

```
switch selector
    1: Acción(es)
    2: Acción(es)
    3: Acción(es)
    4: Acción(es)
default
    Acción(es)
endswitch
```

La repetición DO - UNTIL o “HACER - HASTA” se representa:

```
do
    Acción(es)
while condición
```

La repetición FOR o “PARA” se representa:

```
for contador=valorInicial; condición; incremento
    Acción(es)
endfor
```

La repetición DOWHILE o “MIENTRAS” se representa:

```
while condición
    Acción(es)
endwhile
```

Asimismo, a los módulos, en este libro, se les denomina métodos que no regresan valor, y a las funciones definidas por el usuario, se les denomina métodos que regresan valor.

Los capítulos del 8 en adelante, deben estudiarse en forma detallada, porque es donde se presenta la metodología de la programación orientada a objetos en forma completa, y podrá ver por ejemplo, el ejercicio mencionado líneas antes, como se diseña en el capítulo 9, donde la estructura del algoritmo queda de una forma muy diferente, porque ahí tiene la arquitectura orientada a objetos.

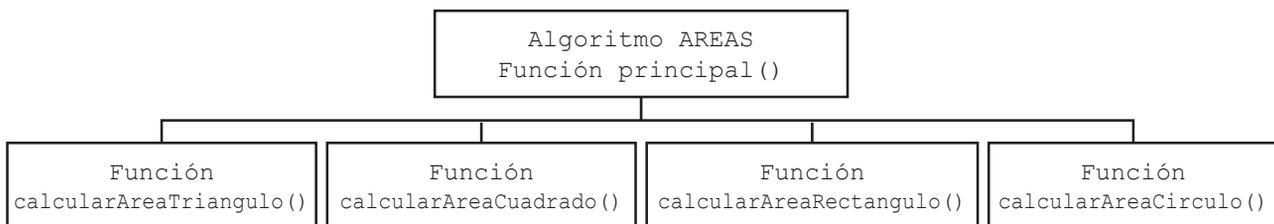
Diferencia entre la programación estructurada y la orientada a objetos

Para resaltar la diferencia esencial entre los dos paradigmas, a continuación se presenta un ejemplo resuelto usando la programación estructurada, y luego se hace referencia a la solución que tiene usando la programación orientada a objetos.

El problema es el ejercicio 13.4.1 planteado en el punto 13.4 Ejercicios resueltos, del capítulo 13, en el que se plantea, elaborar un algoritmo que ofrezca un menú de opciones, mediante el cual se pueda escoger calcular el área de las figuras geométricas: triángulo, cuadrado, rectángulo y círculo. Una vez seleccionada la opción, deben solicitar los datos necesarios, leerlos, hacer el cálculo correspondiente e imprimirlo.

Solución aplicando la programación estructurada

Usando la programación estructurada, primero se diseña el diagrama general de la solución usando la técnica diseño descendente (Top Down Design) u otra similar:



El cual contiene la función principal y cuatro funciones subordinadas que implementan cada una de las tareas a resolver: sumar, restar, multiplicar y dividir.

Enseguida se diseña la lógica de la solución usando pseudocódigo:

Algoritmo AREAS

1. Función principal()
 - a. Declarar variables
opcion: Entero
 - b. do
 1. Imprimir MENU

AREAS FIGURAS GEOMETRICAS
1. TRIANGULO
2. CUADRADO
3. RECTANGULO
4. CIRCULO
5. FIN
ESCOGER OPCIÓN
 2. Leer opcion
 3. switch opcion
 - 1: calcularAreaTriangulo()
 - 2: calcularAreaCuadrado()
 - 3: calcularAreaRectangulo()
 - 4: calcularAreaCirculo()
 4. endswitch
 - c. while opcion != 5
 - d. Fin Función principal
2. Función calcularAreaTriangulo()
 - a. Declarar variables
base, altura, areaTria: Real
 - b. Solicitar Base, Altura
 - c. Leer base, altura
 - d. $areaTria = (base * altura) / 2$
 - e. Imprimir areaTria
 - f. Fin Función calcularAreaTriangulo
3. Función calcularAreaCuadrado()
 - a. Declarar variables
lado, areaCuad: Real
 - b. Solicitar Lado
 - c. Leer lado
 - d. $areaCuad = Potencia(lado, 2)$
 - e. Imprimir areaCuad
 - f. Fin Función calcularAreaCuadrado

```

4. Función calcularAreaRectangulo()
  a. Declarar variables
     areaRec, base, altura: Real
  b. Solicitar Base, Altura
  c. Leer base, altura
  d. areaRec = base * altura
  e. Imprimir areaRec
  f. Fin Función calcularAreaRectangulo

5. Función calcularAreaCirculo()
  a. Declarar
  Constantes
  PI = 3.14159265
  Variables
  areaCirc, radio: Real
  b. Solicitar Radio
  c. Leer radio
  d. areaCirc = PI * Potencia(radio,2)
  e. Imprimir areaCirc
  f. Fin Función calcularAreaCirculo

Fin

```

Explicación:

Se tiene la función principal(), en la cual se ofrece un menú y se solicita la opción, luego de acuerdo a la opción escogida se llama la función correspondiente.

En la función calcularAreaTriangulo(), se hace el proceso de leer la base y la altura, calcular el área del triángulo e imprimirla.

En la función calcularAreaCuadrado(), se hace el proceso de leer el lado, calcular el área del cuadrado e imprimirla.

En la función calcularAreaRectangulo(), se hace el proceso de leer la base y la altura, calcular el área del rectángulo e imprimirla.

En la función calcularAreaCirculo(), se hace el proceso de leer el radio, calcular el área del círculo e imprimirla.

Solución aplicando la programación orientada a objetos

Usando la programación orientada a objetos, primero se diseña el diagrama de clases, que contiene la estructura general de la solución:

Ver el diagrama de clases en el ejercicio 13.4.1 del punto 13.4 Ejercicios resueltos del capítulo 13.

En donde, destaca que en lugar usar el del enfoque de diseño descendente (Top Down Design), aquí se utiliza el diagrama de clases.

Enseguida se diseña la lógica de la solución usando pseudocódigo:

Ver el algoritmo en pseudocódigo en el ejercicio 13.4.1 del punto 13.4 Ejercicios resueltos del capítulo 13.

En donde, destaca el uso de objetos que se crean en base a clases.

Resumiendo

La diferencia esencial de la programación orientada a objetos con respecto a la programación estructurada, son los conceptos y el enfoque de diseño del diagrama de clases, en lugar del enfoque de diseño descendente (Top Down Design). Lo que es totalmente diferente en la POO, es en la arquitectura general de los programas, los cuales, ya no están formados por un conjunto de módulos o funciones, como sucedía con la programación estructurada, sino, por un conjunto de objetos, generados a partir de las clases del diagrama de clases, donde los objetos contienen un conjunto de métodos (equivalentes a módulos y funciones), y éstos están formados por instrucciones.

C

Apéndice C: Algoritmos sin usar etiquetas

Algoritmos sin usar etiquetas

Aunque este autor considera que es mejor utilizar etiquetas porque ayudan a desarrollar disciplina y orden en los estudiantes, virtudes que los maestros debemos estimular, sobre todo en la etapa de formación de los aprendices de programación. En este apéndice se presentan algunos algoritmos de los elaborados en los capítulos del 3 al 13, sólo que aquí se muestra cómo quedan sin utilizar el concepto de etiquetas. Esto es con la finalidad de que el lector pueda observar la diferencia y opte por utilizar la forma que le parezca mejor, esto, una vez que haya aprendido a programar.

Nota: A algunos maestros no les gusta usar etiquetas para enumerar los pasos de los algoritmos, sin embargo, estimado maestro, si Usted está utilizando este libro como texto, es conveniente que utilice los ejemplos como están planteados en el libro, es decir, con etiquetas, porque si Usted explica los algoritmos sin etiquetas, y luego cuando los alumnos estudien el libro y vean los algoritmos con etiquetas, seguramente les causará confusión; y en consecuencia, la utilidad del libro podría ser limitada.

A continuación se presentan algunos algoritmos sin usar etiquetas.

Ejercicio 3.4.1 (Capítulo 3)

```
Algoritmo CALCULOS LOGARITMICOS DE ANGULO
Clase Angulo1
  Método principal()
    Declarar variables
      tamAngulo, senAng, cosAng: Real
    Solicitar tamaño del ángulo en radianes
    Leer tamAngulo
    Calcular senAng = Seno(tamAngulo)
      cosAng = Coseno(tamAngulo)
    Imprimir senAng, cosAng
  Fin Método principal
Fin Clase Angulo1
Fin
```

Ejercicio 4.1.4.1 (Capítulo 4)

```
Algoritmo CALCULA PROMEDIO DE UN ALUMNO
Clase Alumno2
  Método principal()
    Declarar variables
      nombreAlum: Cadena
      calif1, calif2, calif3, calif4, promedio: Real
      observacion: Cadena
    Solicitar nombre del alumno, calificación 1,
      calificación 2, calificación 3,
      calificación 4
```

```
Leer nombreAlum, calif1, calif2, calif3, calif4
Calcular promedio = (calif1+calif2+calif3+calif4)/4
if promedio >= 60 then
    observacion = "Aprobado"
else
    observacion = "Reprobado"
endif
Imprimir nombreAlum, promedio, observacion
Fin Método principal
Fin Clase Alumno2
Fin
```

Ejercicio 5.3.3.1 (Capítulo 5)

```
Algoritmo VENDEDORES DE AUTOS
Clase VendedoresAutos
Método principal()
    Declarar variables
        nombreVend: Cadena
        desea, otro: Carácter
        totAutos, totVend: Entero
        precioAuto, salMin, sueldo,
        totSueldos, totVendido: Real
    Solicitar el salario mínimo
    Leer salMin
    Imprimir Encabezado
    totSueldos = 0
    totVend = 0
    do
        Solicitar el Nombre del vendedor
        Leer nombreVend
        totAutos = 0
        totVendido = 0
        Preguntar "¿Hay auto vendido (S/N)?"
        Leer otro
        while otro == 'S'
            Solicitar el precio del auto
            Leer precioAuto
            totAutos = totAutos + 1
            totVendido = totVendido + precioAuto
            Preguntar "¿Hay otro auto vendido (S/N)?"
            Leer otro
        endwhile
        sueldo = salMin+(totAutos*100)+(totVendido*0.02)
        Imprimir nombreVend, sueldo
        totVend = totVend + 1
        totSueldos = totSueldos + sueldo
    do
```

```

        Preguntar "¿Hay otro vendedor (S/N)?"
        Leer desea
        while desea == 'S'
            Imprimir totVend, totSueldos
        Fin Método principal
    Fin Clase VendedoresAutos
Fin

```

Ejercicio 11.1.2.1 (Capítulo 11)

```

Algoritmo CLIENTES HOJAS HIELO SECO
Clase Cliente3
    Declarar datos
        nombreClie: Cadena
        tipoClie, cantidad: Entero
        precioUni, subTotal, descuento, netoPagar: Real

    Método establecerNombreClie(nom: Cadena)
        nombreClie = nom
    Fin Método establecerNombreClie

    Método establecerTipoClie(tip: Entero)
        tipoClie = tip
    Fin Método establecerTipoClie

    Método establecerCantidad(can: Entero)
        cantidad = can
    Fin Método establecerCantidad

    Método establecerPrecioUni(pre: Real)
        precioUni = pre
    Fin Método establecerPrecioUni

    Método calcularSubTotal()
        subTotal = cantidad * precioUni
    Fin Método calcularSubTotal

    Método calcularDescuento()
        switch tipoClie
            1: descuento = subTotal * 0.05
            2: descuento = subTotal * 0.08
            3: descuento = subTotal * 0.12
            4: descuento = subTotal * 0.15
        endswitch
    Fin Método calcularDescuento

```

```
Método calcularNetoPagar()
    netoPagar = subTotal - descuento
Fin Método calcularNetoPagar

Método obtenerNombreClie() Cadena
    return nombreClie
Fin Método obtenerNombreClie

Método obtenerSubTotal() Real
    return subTotal
Fin Método obtenerSubTotal

Método obtenerDescuento() Real
    return descuento
Fin Método obtenerDescuento

Método obtenerNetoPagar() Real
    return netoPagar
Fin Método obtenerNetoPagar
Fin Clase Cliente3

Clase EjecutaCliente3
Método principal()
    Declarar variables
        nomCli: Cadena
        tiCli, cant, totClientes: Entero
        preUni, totSubTot, totDescuento, totNeto: Real
        desea: Carácter
    Imprimir encabezado
    totClientes = 0
    totSubTot = 0
    totDescuento = 0
    totNeto = 0
    do
        Declarar, crear e iniciar objeto
            Cliente3 objCliente = new Cliente3()
        Solicitar Nombre, Tipo cliente,
            Cantidad, Precio unitario
        Leer nomCli, tiCli, cant, preUni
        Establecer
            objCliente.establecerNombreClie(nomCli)
            objCliente.establecerTipoClie(tiCli)
            objCliente.establecerCantidad(cant)
            objCliente.establecerPrecioUni(preUni)
        Calcular objCliente.calcularSubTotal()
            objCliente.calcularDescuento()
            objCliente.calcularNetoPagar()
```

```
        Imprimir objCliente.obtenerNombreClie()
            objCliente.obtenerSubTotal()
            objCliente.obtenerDescuento()
            objCliente.obtenerNetoPagar()
    totClientes = totClientes + 1
    totSubTot = totSubTot +
        objCliente.obtenerSubTotal()
    totDescuento = totDescuento +
        objCliente.obtenerDescuento()
    totNeto = totNeto +
        objCliente.obtenerNetoPagar()
    Preguntar "¿Desea procesar otro cliente(S/N)?"
    Leer desea
    while desea == 'S'
        Imprimir totClientes, totSubTot,
            totDescuento, totNeto
    Fin Método principal
    Fin Clase EjecutaCliente3
Fin
```

Bibliografía

- Alcalde, E. y García, M.**, *Metodología de la programación*, España, Mc Graw Hill, 1987.
- Bell, D. y Parr, M.**, *Java para estudiantes 6ª edición*, México, Prentice Hall, 2011.
- Booch, G.**, *Análisis y diseño orientado a objetos con aplicaciones Segunda edición*, USA, México, Addison-Wesley/Díaz de Santos, 1996.
- Booch, G., Rumbaugh, J. y Jacobson, I.**, *UML El lenguaje unificado de modelado*, España, Addison Wesley, 1999.
- Ceballos, F.J.**, *Java 2 Curso de programación 3ª edición actualizada*, México, Alfaomega-Rama, 2006.
- Dahl, O.J., Dijkstra, E.W., y Hoare, C.A.R.**, *Structured programming*, USA, Academic Press, 1972.
- Deitel, H.M. y Deitel, P.J.**, *Como programar en C# Segunda edición*, México, Pearson Prentice Hall, 2007.
- Deitel, H.M. y Deitel, P.J.**, *Como programar en Java 7ª edición*, México, Pearson Prentice Hall, 2008.
- Horstmann, C.S. y Cornell, G.**, *Core Java 2 Volume I Fundamentals 5th edition*, USA, The Sun Microsystems Press Prentice Hall, 2000.
- Horton, I.**, *Beginning Java 2*, USA, Wrox Press Inc., 2000.
- Jacobson, I., Booch, G. y Rumbaugh, J.**, *UML El proceso unificado de desarrollo de software*, España, Addison Wesley, 2000.
- Joyanes, A.L.**, *Fundamentos de programación 3ª edición*, España, Mc Graw Hill, 2003.
- Joyanes, A.L.**, *Problemas de metodología de la programación*, España, Mc Graw Hill, 1990.
- Joyanes, A.L.**, *Programación orientada a objetos Segunda edición*, España, Osborne Mc Graw Hill, 1998.
- Joyanes, A.L. y Fernández, A.M.**, *Java 2 Manual de programación*, España, Mc Graw Hill, 2001.
- Korthage, R.**, *Lógica y algoritmos*, México, Limusa, 1967.
- Lemay, L. y Cadenhead, R.**, *Aprendiendo Java en 21 días*, México, Pearson Prentice Hall, 1999.
- López, R.L.**, *Programación estructurada y orientada a objetos un enfoque algorítmico Tercera edición*, México, Alfaomega, 2011.
- López, R.L.**, *Metodología de la programación orientada a objetos*, México, Alfaomega, 2006.
- Meyer, B.**, *Construcción de software orientado a objetos Segunda edición*, España, Prentice Hall, 1999.
- Rumbaugh, J., Jacobson, I. y Booch, G.**, *UML El lenguaje unificado de modelado. Manual de referencia*, España, Addison Wesley, 2000.
- Schildt, Herbert**, *Fundamentos de programación en Java 2*, Colombia, Osborne Mc Graw Hill, 2002.

Schildt, Herbert, *Java 2 Manual de referencia*, España, Osborne Mc Graw Hill, 2001.

Spencer, D., *Problemas para resolver con computadoras*, México, Limusa, 1985.

Warnier, J.D., *Construcción de programas*, España, Editores Técnicos Asociados, 1979.

Watkins, R., *Solución de problemas por medio de computadoras*, México, Limusa, 1984.

Wu, T.C., *Programación en Java. Introducción a la programación orientada a objetos*, México, Mc Graw Hill, 2008.