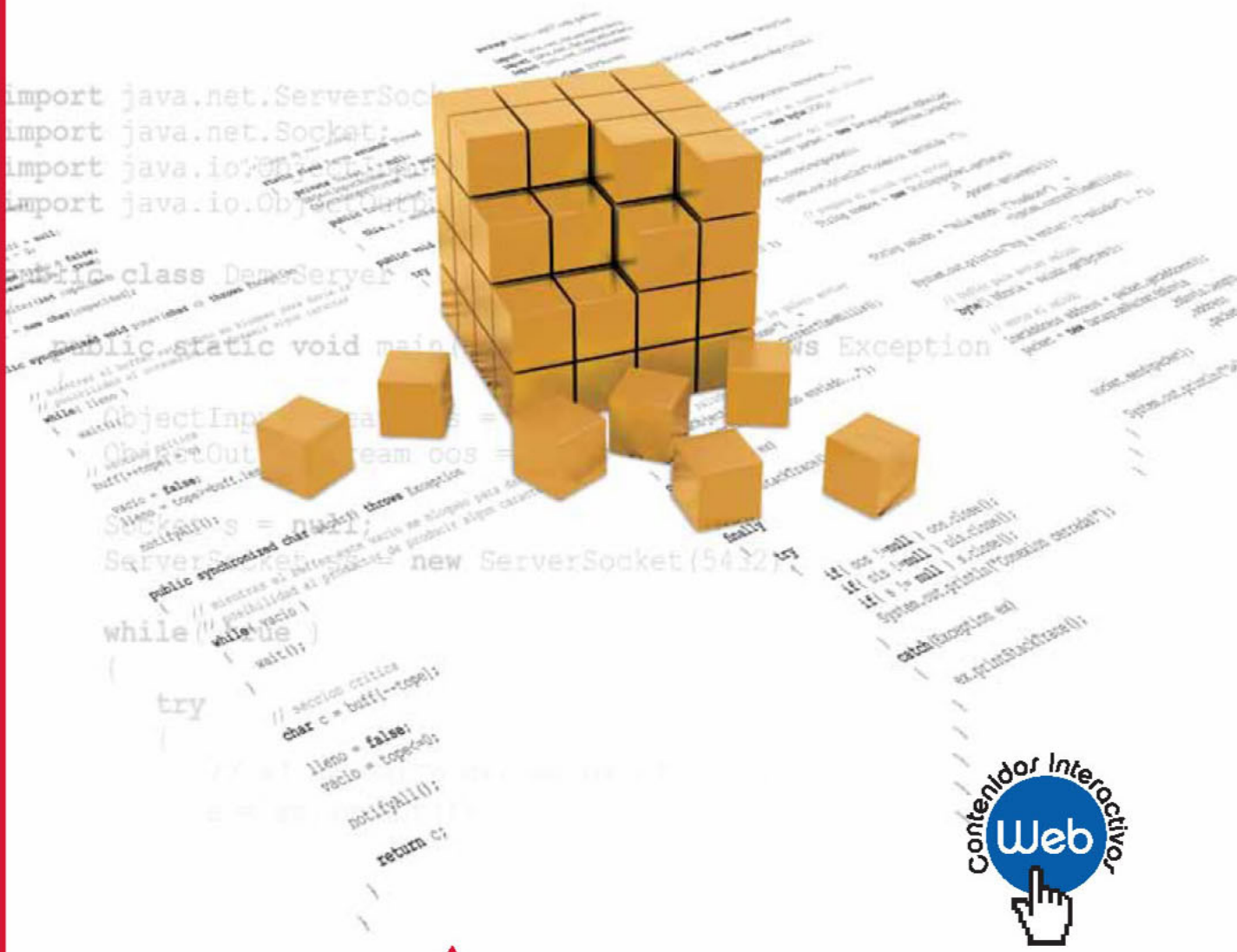
Java a fondo 2da. Edición

ESTUDIO DEL LENGUAJE Y DESARROLLO DE APLICACIONES

ACTUALIZADO A JAVA 7 / INCLUYE INTRODUCCIÓN A HIBERNATE Y SPRING

ING. PABLO AUGUSTO SZNAJDLEDER



libroWeb

Contenidos Interactivos Web

Alfaomega

Java a fondo

Estudio del lenguaje y desarrollo de aplicaciones

Ing. Pablo Augusto Sznajdleder

Alfaomega

Sznajdleder, Pablo Augusto
Java a fondo : estudio del lenguaje y desarrollo de aplicaciones . - 2a ed. - Buenos Aires : Alfaomega Grupo Editor Argentino, 2013.
456 p. ; 24x21 cm.

ISBN 978-987-1609-36-9

1. Informática. I. Título.
CDD 005.3

Edición: Damián Fernández
Revisión de estilo: Vanesa García
Diseño de interiores y portada: Diego Ay
Revisión de armado: Vanesa García

Internet: http://www.alfaomega.com.mx

Paraguay 1307, PB, oficina 11

ISBN 978-987-1609-36-9

Queda hecho el depósito que prevé la ley 11.723

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.
General del Canto 370, Providencia, Santiago, Chile
Tel.: (56-2) 947-9351 – Fax: (56-2) 235-5786 -E-mail: agechile@alfaomega.cl

“Superman”, por Octaviano Sznajdleder

Dedico este trabajo, con todo mi amor, a mi esposa Analía y a mi hijo Octaviano.
Sin ellos, nada tendría sentido.

Dedicado también a la memoria de mi tío Beno que, aunque nunca se lo dije,
siempre ocupó un lugar importante en mi corazón.

Agradecimientos

Quiero agradecer muy especialmente a mi madre Nelly, quien me permitió trabajar en su casa cada vez que Octaviano no me permitió hacerlo en la mía.

Mi agradecimiento también a Damián Fernández quien confió en mí y me brindó esta gran oportunidad, y a Carlos Álvarez, que, cuando le propuse hacer la revisión técnica, sin dudarlo me respondió "me interesa revisar tu libro".

Agradezco también a Analía Mora con quién cuento incondicionalmente para investigar y resolver cuestiones relacionadas con la tecnología y el desarrollo de aplicaciones.

Finalmente, quiero agradecer a Pablo Bergonzi, gurú indiscutido de Java en Argentina, quien siempre está dispuesto a darme una mano cada vez que las cosas no funcionan como deberían hacerlo.

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.

Pablo Augusto Sznajdleder

Es Ingeniero en Sistemas de Información, egresado de la Universidad Tecnológica Nacional (UTN-FRBA).

Actualmente, es profesor en la cátedra de "Algoritmos y Estructuras de Datos" en la UTN-FRBA pasando también por la Universidad Nacional de San Martín (UNSAM) y el Instituto de Tecnología ORT Argentina. Trabajó como instructor Java para Sun Mycrosystems, Oracle e Informix/IBM entre otras empresas líderes.

Desde 1995 trabaja en sistemas, principalmente, en el desarrollo de aplicaciones empresariales distribuidas: primero en C/C++ y luego, en Java/JEE.

En 1996 comenzó a trabajar como Instructor Java para Sun Microsystems y, desde el 2000, se desempeña en la búsqueda y selección de RRHH capacitados en tecnología Java poniendo especial atención en la identificación de jóvenes estudiantes sin experiencia laboral previa, pero con gran potencial profesional.

Tiene las certificaciones internacionales *Sun Certified Java Programmer* (SCJP, 1997) y *Sun Certified Java Developer* (SCJD, 1998) y, además, está certificado como Instructor Oficial Java por Sun Microsystems (1997).

Acaba de publicar su libro sobre algoritmos y estructuras de datos: *Algoritmos a fondo, con implementaciones en C y Java*. En el 2010 publicó la primera edición de *Java a fondo*. En el 2009 participó como revisor técnico en el libro: *Análisis y diseño de algoritmos* (López, Jeder, Vega) y, en el 2008, publicó *HolaMundo pascal, Algoritmos y estructuras de datos*, cuyo contenido cubre por completo los temas que abarca la asignatura de igual nombre en la UTN-FRBA.

Revisor técnico: Carlos Álvarez

Es Licenciado en Análisis de Sistemas, egresado de la Facultad de Ingeniería (UBA). Trabajó durante 10 años como consultor en desarrollo y *performance* de aplicaciones y, desde el 2010, es Gerente de calidad técnica de software de despegar.com.

Contenido

Información del contenido de la página Web

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

Capítulo 1
Introducción al lenguaje de programación Java
- Mapa conceptual
- Autoevaluación
- Videotutoriales:
  - Instalar Java y *Eclipse*.
  - Crear y ejecutar nuestro primer programa en *Eclipse*.
  - Pasar argumentos en línea de comandos en *Eclipse*.
  - Compilar y ejecutar un programa Java desde la línea de comandos (sin utilizar *Eclipse*).
- Presentaciones*

Capítulo 2
Programación orientada a objetos
- Mapa conceptual
- Autoevaluación
- Videotutoriales:
  - Utilizar la herramienta "javadoc" para documentar nuestro código fuente.
  - Empaquetar clases utilizando la herramienta "jar".
- Presentaciones*

Capítulo 3
Acceso a bases de datos (JDBC)
- Mapa conceptual
- Autoevaluación
- Videotutorial:
  - Usar *Eclipse* como cliente SQL.
- Presentaciones*

Capítulo 4
Diseño de aplicaciones Java (Parte I)
- Mapa conceptual
- Autoevaluación
- Presentaciones*

Capítulo 5
Interfaz gráfica (GUI)
- Mapa conceptual
- Autoevaluación
- Videotutorial:
  - Utilizar herramientas visuales para diseño y desarrollo de interfaz gráfica.
- Presentaciones*

Capítulo 6
Multithreading (Hilos)
- Mapa conceptual
- Autoevaluación
- Presentaciones*

Capítulo 7
Networking
- Mapa conceptual
- Autoevaluación
- Videotutorial:
  - Compilar y ejecutar una aplicación RMI.
- Presentaciones*

Capítulo 8
Diseño de aplicaciones Java (Parte II)
- Mapa conceptual
- Autoevaluación
- Presentaciones*

Capítulo 9
Estructuras de datos dinámicas
- Mapa conceptual
- Autoevaluación
- Presentaciones*

Capítulo 10
Parametrización mediante archivos XML
- Mapa conceptual
- Autoevaluación
- Presentaciones*

Capítulo 11
Introspección de clases y objetos
- Mapa conceptual
- Autoevaluación
- Presentaciones*

Capítulo 12
Generalizaciones y desarrollo de frameworks
- Mapa conceptual
- Autoevaluación
- Presentaciones*

Capítulo 13
Entrada/Salida
- Mapa conceptual
- Autoevaluación
- Presentaciones*

Capítulo 14
Consideraciones adicionales
- Mapa conceptual
- Autoevaluación
- Presentaciones*

Capítulo 15
Object Relational Mapping (ORM) y persistencia de datos
- Mapa conceptual
- Autoevaluación
- Presentaciones*

Capítulo 16
Inversión del control por inyección de dependencias
- Mapa conceptual
- Autoevaluación
- Presentaciones*

Capítulo 17
Actualización a Java 7
- Mapa conceptual
- Autoevaluación
- Presentaciones*

Código fuente de cada capítulo

Hipervínculos de interés

Fe de erratas

Guía para el docente de las competencias específicas que se desarrollan con este libro

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://virtual.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

	Videotutoriales que complementan la obra.
	Nuevo en Java 7: bajo este ícono se encuentran nuevas formas de resolver algunas cuestiones en Java.
	Conceptos para recordar: bajo este ícono se encuentran definiciones importantes que refuerzan lo explicado en la página.
	Comentarios o información extra: este ícono ayuda a comprender mejor o ampliar el texto principal.
	Contenidos interactivos: indica la presencia de contenidos extra en la Web.

Prólogo

A partir de los años ochenta, la Humanidad ha comenzado una nueva revolución, equiparable a la Primera Revolución Industrial: estamos hablando de fines del siglo XVIII y principios del XIX de nuestra era, la actual revolución a la que me refiero es la Tecnológica, con la aparición de las primeras computadoras nos referimos a la década de los sesenta y a partir de los ochenta con las primeras computadoras personales.

Luego de este último periodo, hemos vivido avances tecnológicos de manera vertiginosa con cambios que, en un principio, fueron paulatinos pero, ahora, son cada vez más rápidos debido a la aparición de nuevos dispositivos electrónicos como computadoras, celulares, videojuegos, televisores, etcétera.

El hardware es cada vez más pequeño, pero más potente, más eficiente en cuanto a la capacidad en el manejo de la información, siendo atractivo para los usuarios finales en precios, portabilidad, tamaño, resolución de pantalla, peso, etcétera.

En cuanto al software, este ha avanzado más rápido que el hardware con el desarrollo de nuevos lenguajes de programación, lo que ha obligado a las empresas desarrolladoras del hardware a la compactación y eficiencia del mismo.

Uno de los lenguajes de programación que tiene mayor importancia sobre los demás es Java, el cual es dominante en la creación de aplicaciones empresariales, elaboración de videojuegos, sitios Web, etcétera.

Otra característica importante es que Java es multiplataforma, lo que significa que las aplicaciones programadas con este lenguaje pueden "correr" en cualquier sistema operativo y hardware. Esto lo posiciona como el lenguaje de programación más versátil de, al menos, los últimos 15 años.

Java, como lenguaje de Programación Orientada a Objetos (POO) requiere del conocimiento de diversas técnicas (Herencia, Polimorfismo, Abstracción, Encapsulamiento, etc.) las cuales son importantes que el lector domine para poder desarrollar aplicaciones flexibles, mantenibles, robustas y escalables.

Llevado al ámbito empresarial, las aplicaciones Java pueden interactuar con bases de datos, servidores de mensajería asincrónica, servicios de nombres y directorios, *Web services*, etcétera, lo que facilita enormemente las tareas de integración y el desarrollo de sistemas distribuidos.

Por lo expuesto, Java es "mucho más" que un simple lenguaje de programación; y para conocerlo no solo se requiere aprender su sintaxis y semántica. También es fundamental conocer técnicas de diseño que, sumadas a las potencialidades del lenguaje, nos ayuden a desarrollar aplicaciones duraderas en el tiempo.

El libro *Java a fondo* explica cómo desarrollar aplicaciones Java desde un enfoque de diseño y en sintonía con los lineamientos recomendados por los expertos de la industria.

Comienza desde cero y avanza progresivamente incorporando conceptos, que se van reforzando a medida que expone ejemplos sencillos y eficaces.

El libro se diferencia de los demás porque no aburre con extensas e interminables listas de palabras reservadas, operadores, reglas, clases y métodos. Por el contrario, los conocimientos se proporcionan conforme van siendo necesarios, según sea el ejemplo o caso práctico que se esté analizando en cada capítulo.

Aunque Java a fondo solo cubre JSE (*Java Standard Edition*), plantea y explica las técnicas y los patrones de diseño que son pilares en JEE (*Java Enterprise Edition*), dejando al lector en la puerta del desarrollo de aplicaciones empresariales. *Session Facade*, *Model View Controller*, *Factory Method*, *Data Access Object* son algunos de los patrones de diseño que el autor explica con una simpleza que sorprende, haciendo fácil lo que en Internet y en otros libros se presenta como imposible, elitista o para pocos.

Java a fondo hace hincapié en el diseño de aplicaciones desarrolladas en capas lógicas, las cuales proveen sus servicios a las capas anteriores y se nutren de los servicios provistos por las capas posteriores. Me refiero a las famosas "capa de presentación", "capa de negocios" y "capa de acceso a datos".

Cabe destacar que el autor complementa la obra con una serie de tutoriales en video en los que él mismo explica aquellos temas que, dada su naturaleza, resultan difíciles de explicar en papel. Estos videos cubren, desde cómo instalar Eclipse (la herramienta de desarrollo que el autor recomienda utilizar), hasta cómo montar un entorno de objetos distribuidos RMI; pasando por cómo diseñar visualmente interfaces gráficas, cómo usar el *debugger*, cómo conectarse a la base de datos para ejecutar consultas, etcétera.

Finalmente, en esta nueva edición, se incorporan dos capítulos que introducen al lector en el uso de Hibernate y Spring; estos *frameworks* de "persistencia de objetos" e "inyección de dependencias" respectivamente son extensamente usados en la industria de desarrollo y muchas veces se requiere conocerlos para incorporarse al mercado laboral.

Mtro. Jorge Armando Villarreal Ramírez
DOCENTE DE TIEMPO COMPLETO
UNIVERSIDAD DEL VALLE DE MÉXICO / CAMPUS COYOACÁN

Introducción

Java a fondo propone un curso de lenguaje y desarrollo de aplicaciones Java basado en un enfoque totalmente práctico, sin vueltas ni rodeos, y contemplando el aprendizaje basado en competencias.

El libro comienza desde un nivel "cero" y avanza hasta llegar a temas complejos como Introspección de clases y objetos, Acceso a bases de datos (JDBC), multiprogramación, *networking* y objetos distribuidos (RMI), entre otros.

Se hace hincapié en la teoría de objetos: polimorfismo, clases abstractas, *interfaces* Java y clases genéricas así como en el uso de patrones de diseño que permiten desacoplar las diferentes partes que componen una aplicación para que esta resulte ser mantenible, extensible y escalable.

La obra explica cómo diseñar y desarrollar aplicaciones Java respetando los estándares y lineamientos propuestos por los expertos de la industria lo que la convierte en una herramienta fundamental para obtener las certificaciones internacionales SCJP (*Sun Certified Java Programmer*) y SCJD (*Sun Certified Java Developer*).

Para ayudar a clarificar los conceptos, el autor incluye diagramas UML y una serie de videotutoriales que incrementan notablemente la dinámica del aprendizaje, además de guiar al alumno en el uso de una de las herramientas de desarrollo más utilizadas y difundidas: Eclipse.

Java a fondo puede utilizarse como un libro de referencia o como una guía para desarrollar aplicaciones Java ya que la estructuración de los contenidos fue cuidadosamente pensada para este fin.

Entre los Capítulos 1 y 3, se explica el lenguaje de programación, el paradigma de objetos y JDBC que es la API a través de la cual los programas Java se conectan con las bases de datos.

El Capítulo 4 explica cómo desarrollar una aplicación Java separada en capas lógicas ("presentación", "aplicación" y "acceso a datos") poniendo en práctica los principales patrones de diseño. La aplicación de estudio se conecta a una base de datos e interactúa con el usuario a través de la consola (teclado y pantalla en modo texto).

El Capítulo 5 explica AWT y *Swing* que son las APIs provistas por el lenguaje con las que podemos desarrollar interfaces gráficas, permitiendo que el lector programe una capa de presentación más vistosa y amigable para la aplicación desarrollada en el capítulo anterior.

En los Capítulos 6 y 7, se estudian los conceptos de multiprogramación y *networking*: cómo conectar programas a través de la red utilizando los protocolos UDP y TCP, y RMI.

Con los conocimientos adquiridos hasta este momento, en el Capítulo 8, se vuelve a analizar la aplicación de estudio del Capítulo 4, pero desde un punto de vista físico diferenciando entre capas lógicas y capas físicas e implementando la capa de aplicación detrás de los servicios de un *server*.

Entre los Capítulos 9 y 11, se estudian conceptos de estructuras de datos, *parseo* de contenidos XML e introspección de clases y objetos para luego, en el Capítulo 12, aplicarlos en el análisis y desarrollo de un *framework* que automatiza las tareas rutinarias y repetitivas que hubo que realizar (por ejemplo) para leer archivos XML y para acceder a la base de datos, entre otras cosas.

En el Capítulo 13, se estudian conceptos de entrada y salida (I/O *streams*).

El Capítulo 14 profundiza sobre cuestiones que, adrede, no fueron tratadas para evitar confundir al lector. Principalmente, consideraciones sobre concurrencia, *multithreading* y sobre el uso ciertas clases "*legacy*".

Los Capítulos 15 y 16 introducen al uso de dos *frameworks* ineludibles: Hibernate y Spring; estos *frameworks* de "persistencia de objetos" e "inyección de dependencias" respectivamente son ampliamente usados en la industria del software.

El último capítulo menciona las principales novedades que incluye la API de Java 7; novedades que también se han ido destacando a lo largo de todo el libro.

Para aquellos lectores que no tienen las bases mínimas y necesarias de programación estructurada, se incluye un apéndice de programación inicial. Se ofrece también un apéndice que explica cómo desarrollar *Applets*.

El docente debe saber que, en cada capítulo, se mencionan las competencias específicas a desarrollar y que en la página Web del libro dispone de una guía detallada de las competencias que se desarrollan a lo largo del libro y las evidencias que se pueden recolectar.

1

Introducción al lenguaje de programación Java

Contenido

Objetivos del capítulo

- Desarrollar ejemplos simples que le permitan al lector familiarizarse con el lenguaje de programación Java.
- Identificar los elementos del lenguaje: estructuras, tipos de datos, operadores, *arrays*, etc.
- Conocer la clase `String` y realizar operaciones sobre cadenas de caracteres.
- Entender qué son el JDK, el JRE y la JVM.

Competencias específicas

- Elaborar desarrollos en lenguaje Java para ejemplificar su programación.
- Describir los elementos y requerimientos del lenguaje de programación Java para reconocer sus características.

1.1 Introducción

Java es un lenguaje de programación de propósitos generales. Podemos usar Java para desarrollar el mismo tipo de aplicaciones que programamos con otros lenguajes como C o Pascal.

Habitualmente, tendemos a asociar el término "Java" al desarrollo de páginas de Internet. La gente cree que "Java es un lenguaje para programar páginas Web", pero esto es totalmente falso. La confusión surge porque Java permite "incrustar" programas dentro de las páginas Web para que sean ejecutados en el navegador del usuario. Estos son los famosos *Applets*, que fueron muy promocionados durante los años noventa pero que hoy en día son obsoletos y, prácticamente, quedaron en desuso.

Tampoco debemos confundir Java con *JavaScript*. El primero es el lenguaje de programación que estudiaremos en este libro. El segundo es un lenguaje de *scripting* que permite agregar cierta funcionalidad dinámica en las páginas Web. Nuevamente, la similitud de los nombres puede aportar confusión por eso, vale la pena aclarar que *JavaScript* no tiene nada que ver con Java. Son dos cosas totalmente diferentes.

JSE (*Java Standard Edition*) incluye al lenguaje de programación, el *runtime* y un conjunto de herramientas de desarrollo.
JEE (*Java Enterprise Edition*) básicamente es una biblioteca orientada al desarrollo de aplicaciones empresariales.

No obstante, podemos utilizar Java para desarrollar páginas Web. La tecnología Java que permite construir este tipo de aplicaciones está basada en el desarrollo de Servlets, pero esto es parte de lo que se conoce como JEE (*Java Enterprise Edition*) y excede el alcance de este libro.

Solo mencionaremos que JEE es un conjunto de bibliotecas que permiten desarrollar "aplicaciones empresariales" con Java. Es decir que para programar con JEE primero debemos conocer el lenguaje de programación Java.

Java, como lenguaje de programación, se caracteriza por dos puntos bien definidos:

- Es totalmente orientado a objetos.
- La sintaxis del lenguaje es casi idéntica a la del lenguaje C++.

Más allá de esto, debemos mencionar que incluye una biblioteca muy extensa (árbol de clases) que provee funcionalidad para casi todo lo que el programador pueda necesitar. Esto abarca desde manejo de cadenas de caracteres (*strings*) hasta *Sockets* (redes, comunicaciones), interfaz gráfica, etcétera.

1.2 Comencemos a programar

Desarrollaremos un programa Java que escribe la frase "`Hola Mundo !!!`" en la consola (pantalla).

```
package libro.cap01;

public class HolaMundo
{
   public static void main(String[] args)
   {
      System.out.println("Hola Mundo !!!");
   }
}
```

Antes de pasar a analizar el código del programa debo solicitarle al lector que sea paciente y que me permita pasar por alto la explicación de algunas de las palabras o sentencias que utilizaremos en los ejemplos de este primer capítulo. Este es el caso de las palabras `public`, `static`, `class`, `package`, etc. Todas estas palabras serán explicadas en detalle en el capítulo de programación orientada a objetos.

Ahora sí analicemos el código de nuestro programa.

Un programa Java es una clase (`class`) que contiene el método (o función) `main`.

Este método tiene que ser definido con los modificadores `public`, `static`, `void` y debe recibir un `String[]` como parámetro.

Los bloques de código se delimitan con `{ }` (llaves) y las sentencias finalizan con `;` (punto y coma).

Podemos ver también que el programa comienza definiendo un `package`. Por último, con `System.out.println` escribimos el texto que vamos a mostrar en la consola.

En Java siempre codificamos clases y cada clase debe estar contenida dentro de un archivo de texto con el mismo nombre que la clase y con extensión **.java**. Así, nuestro programa debe estar codificado en un archivo llamado **HolaMundo.java** (respetando mayúsculas y minúsculas).

Cada clase debe estar contenida dentro de un archivo de texto con el mismo nombre que la clase y con extensión **.java**.

1.2.1 El Entorno Integrado de Desarrollo (IDE)

Si bien podemos editar nuestro código utilizando cualquier editor de texto y luego compilarlo en línea de comandos, lo recomendable es utilizar una herramienta que nos ayude en todo el proceso de programación.

Una IDE (*Integrated Development Enviroment*) es una herramienta que permite editar programas, compilarlos, depurarlos, documentarlos, ejecutarlos, etc.

Para trabajar con Java existen en el mercado diferentes IDE. Algunas son de código abierto (*open source*) como *Eclipse* y *NetBeans* y otras son pagas e impulsadas por las empresas de tecnología como *JBuilder* (de *Borland*), *JDeveloper* (de *Oracle*), *WebPhere* (de IBM), etcétera.

En este libro utilizaremos *Eclipse* que, además de ser gratis, es (para mi gusto) la mejor herramienta de programación de todos los tiempos.

Para no "ensuciar" el contenido del capítulo con una excesiva cantidad de imágenes y contenidos básicos que (probablemente) para el lector podrían resultar triviales se proveen videotutoriales en los que se explica en detalle cómo instalar Java y *Eclipse* y cómo utilizar *Eclipse* para escribir y ejecutar programas Java. Si el lector no tiene conocimientos previos de programación se provee también un apéndice donde se explican las bases necesarias para que pueda comprender los contenidos expuestos en este libro.

Instalar Java y *Eclipse*.

Crear y ejecutar nuestro primer programa en *Eclipse*.

1.3 Estructuras de control

A continuación, analizaremos una serie de programas simples que nos ayudarán a ver cómo definir variables, cómo utilizar estructuras de decisión, estructuras iterativas, cómo comentar código, etcétera.

1.3.1 Entrada y salida de datos por consola

Llamamos "consola" al conjunto compuesto por la pantalla (en modo texto) y el teclado de la computadora donde se ejecutará nuestro programa. Así, cuando hablemos de "ingreso de datos por consola" nos estaremos refiriendo al teclado y cuando hablemos de "mostrar datos por consola" nos referiremos a la pantalla (siempre en modo texto).

El siguiente programa pide al usuario que ingrese su nombre, lee el dato por teclado y luego lo muestra en la consola.

```
package libro.cap01;

import java.util.Scanner;

public class HolaMundoNombre
{
   public static void main(String[] args)
   {
      // esta clase permite leer datos por teclado
      Scanner scanner = new Scanner(System.in);

      // mensaje para el usuario
      System.out.print("Ingrese su nombre: ");

      // leemos un valor entero por teclado
      String nom = scanner.nextLine();

      // mostramos un mensaje y luego el valor leido
      System.out.println("Hola Mundo: " + nom);
   }
}
```

Diferencia entre: `System.out.print` y `System.out.println`. El primero imprime en la consola el valor del argumento que le pasamos. El segundo hace lo mismo pero, agrega un salto de línea al final.

La clase `Scanner` permite leer datos a través del teclado. Para no confundir al lector simplemente aceptaremos que el uso de esta clase es necesario y no lo explicaremos aquí, pero obviamente quedará explicado y comprendido a lo largo del libro.

Luego mostramos un mensaje indicando al usuario que debe ingresar su nombre a través del teclado. A continuación, leemos el nombre que el usuario vaya a ingresar y lo almacenamos en la variable `nom`. Por último, mostramos un mensaje compuesto por un texto literal `"Hola Mundo: "` seguido del valor contenido en la variable `nom`.

Notemos la diferencia entre: `System.out.print` y `System.out.println`. El primero imprime en la consola el valor del argumento que le pasamos. El segundo hace lo mismo, pero agrega un salto de línea al final.

1.3.2 Definición de variables

Podemos definir variables en cualquier parte del código simplemente indicando el tipo de datos y el nombre de la variable (identificador).

Identificadores válidos son:

```
fecha
iFecha
fechaNacimiento
fecha_nacimiento
fecha3
_fecha
```

Identificadores NO válidos son:

```
3fecha
fecha-nacimiento
fecha+nacimiento
-fecha
```

1.3.3 Comentarios en el código

Java admite los mismos tipos de comentarios que C: comentarios "en línea" que comienzan con `//` y comentarios en bloque delimitados por `/*` y `*/`.

Java soporta comentarios *in-line* (de una sola línea) y comentarios de varias líneas.

Comentarios de una sola línea:

```
// esto es una linea de codigo comentada
```

Comentarios de más de una línea:

```
/*
Estas son varias
lineas de codigo
comentadas
*/
```

El siguiente programa pide al usuario que ingrese su nombre, edad y altura. Estos datos deben ingresarse separados por un espacio en blanco. Luego los muestra por consola.

```
package libro.cap01;

import java.util.Scanner;

public class HolaMundoNombre
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);

      // Mensaje para el usuario
      System.out.print("Ingrese nombre edad altura: ");

      // leemos el nombre
      String nom = scanner.next();

      // leemos la edad
      int edad= scanner.nextInt();

      // leemos la altura
      double altura = scanner.nextDouble();

      // mostramos los datos por consola
      System.out.println("Nombre: "+nom
                         +" Edad: "+edad
                         +" Altura : "+altura);
   }
}
```

Este ejemplo ilustra el uso de datos de diferentes tipos: `String`, `int` y `double`. También muestra que podemos definir variables en cualquier parte del código fuente (igual que en C++) y, por último, muestra cómo concatenar datos de diferentes tipos para emitir la salida del programa.

1.3.4 Estructuras de decisión

En Java disponemos de tres estructuras de decisión o estructuras condicionales:

Decisión simple: `if`

Decisión múltiple: `switch`

Decisión *in-line*: `a>b ? "a es Mayor" : "a es Menor"`

Comenzaremos analizando la sentencia `if` cuya estructura es la siguiente:

```
if( condicion )
{
    accion1;
}
else
{
    accion2;
}
```

Ejemplo: ¿es mayor de 21 años?

En el siguiente ejemplo, utilizamos un `if` para determinar si el valor (`edad`) ingresado por el usuario es mayor o igual que 21.

```
package libro.cap01;

import java.util.Scanner;

public class EsMayorQue21
{
   public static void main(String[] args)
   {
      Scanner scanner=new Scanner(System.in);

      System.out.print("Ingrese su edad: ");
      int edad=scanner.nextInt();

      if( edad >= 21 )
      {
         System.out.println("Ud. es mayor de edad !");
      }
      else
      {
         System.out.println("Ud. es es menor de edad");
      }
   }
}
```

Ejemplo: ¿es par o impar?

El siguiente programa pide al usuario que ingrese un valor entero e indica si el valor ingresado es par o impar.

Recordemos que un número es par si es divisible por 2. Es decir que el resto en dicha división debe ser cero. Para esto, utilizaremos el operador `%` (operador módulo, retorna el resto de la división).

```
package libro.cap01;

import java.util.Scanner;

public class ParOImpar
{

   public static void main(String[] args)
   {
      Scanner scanner=new Scanner(System.in);

      System.out.print("Ingrese un valor: ");
      int v = scanner.nextInt();

      // obtenemos el resto de dividir v por 2
      int resto = v % 2;

      // para preguntar por igual utilizamos ==
      if( resto == 0 )
      {
         System.out.println(v+" es Par");
      }
      else
      {
         System.out.println(v+" es Impar");
      }
   }
}
```

Para resolver este problema, primero obtenemos el resto que se origina al dividir `v` por 2. Luego preguntamos si este resto es igual a cero. Notemos que el operador para comparar es `==` (igual igual). Java (como C) utiliza `==` como operador de comparación ya que el operador `=` (igual) se utiliza para la asignación.

Java (como C) utiliza `==` como operador de comparación ya que el operador `=` (igual) se utiliza para la asignación.

El `if` *in-line* tiene la siguiente estructura:

```
condicion ? retornoPorTrue : retornoPorFalse;
```

Lo anterior debe leerse de la siguiente manera: si se verifica la `condicion` entonces se retorna la expresión ubicada entre el `?` (signo de interrogación) y los `:` (dos puntos). Si la `condicion` resulta falsa entonces se retorna la expresión ubicada después de los `:` (dos puntos).

Ejemplo: ¿es par o impar? (utilizando `if` *in-line*)

```
package libro.cap01;

import java.util.Scanner;

public class ParOImpar2
{
```

```
   public static void main(String[] args)
   {
      Scanner scanner=new Scanner(System.in);


      System.out.print("Ingrese un valor: ");
      int v=scanner.nextInt();

      // obtenemos el resto de dividir v por 2
      int resto= v % 2;

      // utilizando un if in-line
      String mssg = (resto == 0 ) ? "es Par": "es Impar";

      // muestro resultado
      System.out.println(v+" "+mssg);
   }
}
```

La decisión múltiple `switch` tiene la siguiente estructura:

```
switch( variableEntera )
{
   case valor1:
      accionA;
      accionB;
        :
      break;
   case valor2:
      accionX;
      accionY;
        :
      break;
    :
   default:
      masAcciones;
}
```

En Java 7 la sentencia `switch` admite evaluar cadenas de caracteres.

Es muy importante recordar poner siempre el `break` al finalizar cada `case`.

Dependiendo del valor de `variableEntera`, el programa ingresará por el `case`, cuyo valor coincide con el de la variable. Se ejecutarán todas las acciones desde ese punto hasta el final, salvo que se encuentre una sentencia `break` que llevará al control del programa hasta la llave que cierra el `switch`. Por este motivo (al igual que en C), es muy importante recordar poner siempre el `break` al finalizar cada `case`.

Ejemplo: muestra día de la semana.

En el siguiente programa, pedimos al usuario que ingrese un día de la semana (entre 1 y 7) y mostramos el nombre del día. Si ingresa cualquier otro valor informamos que el dato ingresado es incorrecto.

```
package libro.cap01;

import java.util.Scanner;

public class DemoSwitch

   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Ingrese un dia de la semana (numero): ");
      int v = scanner.nextInt();

      String dia;

      switch( v )
      {
         case 1:
            dia = "Lunes";
            break;
         case 2:
            dia = "Martes";
            break;
         case 3:
            dia = "Miercoles";
            break;
         case 4:
            dia = "Jueves";
            break;
         case 5:
            dia = "Viernes";
            break;
         case 6:
            dia = "Sabado";
            break;
         case 7:
            dia = "Domingo";
            break;
         default:
            dia = "Dia incorrecto... El valor debe ser entre 1 y 7.";
      }

      System.out.println(dia);
   }
}
```

Como vemos, el `switch` permite decidir entre diferentes opciones (siempre deben ser numéricas).

Dependiendo de cuál sea el valor ingresado por el usuario el programa optará por el `case` correspondiente. En caso de que el usuario haya ingresado un valor para el cual no hemos definido ningún `case` entonces el programa ingresará por `default`.

Notemos también que utilizamos la sentencia `break` para finalizar cada `case`. Esto es muy importante ya que si no la utilizamos el programa, luego de entrar al `case` correspondiente, seguirá secuencialmente ejecutando todas las sentencias posteriores. Si el lector conoce algo de lenguaje C, esto no le llamará la atención ya que funciona exactamente igual.

1.3.5 Estructuras iterativas

Disponemos de tres estructuras iterativas: el `while`, el `do-while` y el `for`.

Nuevamente, para aquellos que conocen lenguaje C estas instrucciones son idénticas.

Comencemos por analizar el uso del `while` cuya estructura es la siguiente:

```
while( condicion )
{
    accion1;
    accion2;
    :
}
```

El ciclo itera mientras `condicion` resulte verdadera.

Ejemplo: muestra números naturales.

El siguiente programa utiliza un `while` para mostrar los primeros *n* números naturales. El usuario ingresa el valor *n* por teclado.

```
package libro.cap01;

import java.util.Scanner;

public class PrimerosNumeros1
{
   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);

      // leo el valor de n
      int n = scanner.nextInt();

      int i = 1;

      while( i <= n )
      {
         // muestro el valor de i
         System.out.println(i);

         // incremento el valor de i
         i++;
      }
   }
}
```

Vemos que el ciclo `while` itera mientras que el valor de `i` sea menor o igual que el valor de `n` (que fue ingresado por teclado). Por cada iteración mostramos el valor de la variable `i` y luego la incrementamos.

Analicemos el ciclo `do-while` cuya estructura es la siguiente:

```
do
{
    accion1;
    accion2;
    :
}
while( condicion );
```

Este ciclo también itera mientras se verifique la `condicion`, pero a diferencia del ciclo anterior en este caso la entrada al ciclo no está condicionada; por lo tanto, las acciones encerradas entre el `do` y el `while` se ejecutarán al menos una vez.

Ejemplo: muestra números naturales (utilizando `do-while`).

```
package libro.cap01;

import java.util.Scanner;

public class PrimerosNumeros2
{
   public static void main(String[] args)
   {
      Scanner scanner=new Scanner(System.in);
      int n = scanner.nextInt();

      int i = 1;

      do
      {
         System.out.println(i);
         i++;
      }
      while( i <= n );
   }
}
```

Por último, veremos el ciclo `for` cuya estructura es la siguiente:

```
for( inicializacion; condicion; incremento )
{
    accion1;
    accion2;
    :
}
```

Este ciclo tiene tres secciones separadas por `;` (punto y coma). En la primera sección, se define e inicializa una variable entera que llamaremos variable de control. En la segunda sección, se especifica una condición lógica que (frecuentemente) estará en función de esta variable. En la tercera sección, se define el incremento de la variable de control.

El `for` de Java es exactamente el mismo `for` de C++.

Ejemplo: muestra números naturales (utilizando `for`).

```
package libro.cap01;

import java.util.Scanner;

public class PrimerosNumeros3
{
   public static void main(String[] args)
   {
      Scanner scanner=new Scanner(System.in);
      int n = scanner.nextInt();
```

```
        for( int i=1; i<=n; i++ )
        {
           System.out.println(i);
        }
    }
}
```

1.4 Otros elementos del lenguaje

En esta sección estudiaremos otros elementos del lenguaje Java tales como tipos de datos, definición de constantes, *arrays*, etcétera.

1.4.1 Tipos de datos

Java provee los siguientes tipos de datos:

Tipo	Descripción	Longitud
`byte`	entero con signo	1 *byte*
`char`	entero sin signo	2 *bytes*
`short`	entero con signo	2 *bytes*
`int`	entero con signo	4 *bytes*
`long`	entero con signo	8 *bytes*
`float`	punto flotante	4 *bytes*
`double`	punto flotante	8 *bytes*
`boolean`	lógico (admite `true` o `false`)	1 *byte*
`String`	objeto, representa una cadena de caracteres	

En Java no existe el modificador *unsigned* como en C. Por esta razón, los tipos de datos enteros admiten o no valores negativos según se muestra en esta tabla.

Si el lector conoce algo de lenguaje C, podrá observar que en ambos lenguajes los tipos de datos son prácticamente los mismos.

En otros lenguajes las longitudes de los tipos de datos pueden variar dependiendo de la arquitectura y del compilador que se esté utilizando. Es decir: en C (por ejemplo) no siempre se reservará la misma cantidad de memoria para una variable de tipo `int`.

En Java (dado que los programas Java corren dentro de una máquina virtual) las longitudes siempre serán las expresadas en la tabla anterior. Este tema será tratado en detalle más adelante.

1.4.2 Algunas similitudes y diferencias con C y C++

Para aquellos lectores que tienen conocimientos de estos lenguajes, voy a marcar algunos puntos que les resultarán útiles. A quien no conozca nada de C o C++, considero que también podrá resultarle de utilidad leer esta sección, pero si el lector nota que esta lectura le resulta algo confusa entonces le recomiendo directamente pasarla por alto.

El modificador `unsigned` no existe en Java. Los tipos enteros tienen o no tienen bit de signo (según la tabla anterior) y esto no puede alterarse.

El tipo de datos `boolean`: en Java existe este tipo de datos que admite los valores `true` y `false`. En C y C++, se utiliza el tipo `int` como booleano aceptando que el valor cero es `false` y cualquier otro valor distinto de cero es `true`. Esto en Java no es aceptado. Los `int` no tienen valor de verdad.

Operadores unarios y binarios son los mismos que en C, por lo tanto, las siguientes sentencias son válidas tanto en C como en Java:

```
int i=0;

// incrementa en 1 el valor de i pero retorna el valor anterior
i++;

// incrementa en 1 el valor de i y retorna nuevo valor
++i;

i+=10;  // suma 10 al valor de i
i-=5;   // resta 5 al valor de i
i*=3;   // incrementa 3 veces el valor de i
```

Recolección automática de memoria: esta es una diferencia clave entre Java y C/C++. En Java el programador no tiene la responsabilidad de liberar la memoria que va quedando desreferenciada. Esta tarea es automática y la lleva a cabo un proceso llamado *Garbage Collector* (el "recolector de basura").

1.4.3 Definición de constantes

Las constantes se definen fuera de los métodos utilizando el modificador `final`.

Habitualmente, se las define como públicas y estáticas (`public`, `static`). Más adelante, explicaremos el significado de `public`, `static` y "método".

Ejemplo: muestra día de la semana (utilizando constantes).

```
package libro.cap01;

import java.util.Scanner;

public class DemoConstantes
{
   // definimos las constantes
   public static final int LUNES = 1;
   public static final int MARTES = 2;
   public static final int MIERCOLES = 3;
   public static final int JUEVES = 4;
   public static final int VIERNES = 5;
   public static final int SABADO = 6;
   public static final int DOMINGO = 7;

   public static void main(String[] args)
   {
      Scanner scanner = new Scanner(System.in);

      System.out.print("Ingrese un dia de la semana (numero): ");
      int v = scanner.nextInt();

      String dia;

      switch( v )
      {
         case LUNES:
            dia = "Lunes";
            break;
```

```
                case MARTES:
                    dia = "Martes";
                    break;
                case MIERCOLES:
                    dia = "Miercoles";
                    break;
                case JUEVES:
                    dia = "Jueves";
                    break;
                case VIERNES:
                    dia = "Viernes";
                    break;
                case SABADO:
                    dia = "Sabado";
                    break;
                case DOMINGO:
                    dia = "Domingo";
                    break;
                default:
                    dia = "Dia incorrecto... Ingrese un valor entre 1 y 7.";
            }

            System.out.println(dia);
        }
    }
```

1.4.4 Arrays

En Java los *arrays* comienzan siempre desde cero.

Un *array* es un conjunto de variables del mismo tipo cuyas direcciones de memoria son contiguas. Esto permite definir un nombre para el *array* (conjunto de variables) y acceder a cada elemento del conjunto (a cada variable) a través del nombre común (nombre del *array*) más un subíndice que especifica la posición relativa del elemento al que queremos acceder.

En Java los *arrays* comienzan siempre desde cero y se definen de la siguiente manera:

```
// define un array de 10 elementos enteros numerados de 0 a 9
int arr[] = new int[10];
```

También podemos construir un *array* de *n* elementos, siendo *n* una variable.

```
int n = 10;
int arr[] = new int[n];
```

Debe quedar claro que el *array* es estático. Una vez definido su tamaño este será fijo. No se pueden agregar ni eliminar elementos en un *array*.

Para acceder a un elemento del *array*, lo hacemos a través de un subíndice.

```
// asigno el numero 123 en la posicion 5 del array arr
arr[5] = 123;
```

Ejemplo: almacena valores en un *array*.

En el siguiente ejemplo, definimos un *array* de 10 enteros. Luego pedimos al usuario que ingrese valores numéricos (no más de diez) y los guardamos en el *array*. Por último, recorremos el *array* para mostrar su contenido.