

# Introducción a Java: Guía de actividades prácticas

Wilson Rojas / Mario Silva



UNIVERSIDAD **EL BOSQUE**

FACULTAD DE INGENIERÍA



# Introducción a Java: Guía de actividades prácticas



# Introducción a Java: Guía de actividades prácticas

Wilson Rojas / Mario Silva



UNIVERSIDAD **EL BOSQUE**

---

FACULTAD DE INGENIERÍA

**001.6424 R64i**

**ROJAS REALES, Wilson**

Introducción a Java: guía de actividades prácticas / Wilson Rojas Reales, Mario Silva Montoya.

-- Bogotá : Universidad El Bosque, 2016.

166 páginas

ISBN 978-958-739-076-6 (Impreso)

ISBN 978-958-739-077-3 (Digital)

1. Java (lenguaje de programación para computadores) 2. Procesamiento electrónico de datos 3. Algoritmos -- Computadores  
4. Programación orientada a objetos (computación) -- Manuales 5. Silva Montoya Mario.



FACULTAD DE INGENIERÍA

**Introducción a Java: guía de actividades prácticas**

ISBN impreso: 978-958-739-076-6

ISBN digital: 978-958-739-077-3

© Universidad El Bosque

Editorial Universidad El Bosque

Wilson Rojas Reales

Mario Silva Montoya

Rector: Rafael Sánchez París

Vicerrectora Académica: María Clara Rangel Galvis

Vicerrector de Investigaciones: Miguel Otero Cadena

Vicerrector Administrativo: Francisco Falla Carrasco

Decano Facultad de Ingeniería:

Julio César Sandoval

Directora Programa de Ingeniería de Sistemas:

Natalia Parra Román

Editorial Universidad El Bosque

Dirección: Av. Cra 9 n°. 131A-02, Torre D, 4.º piso

Teléfono: +57 (1) 648 9000, ext. 1395

Correo electrónico: [editorial@unbosque.edu.co](mailto:editorial@unbosque.edu.co)

Sitio web: [www.uelbosque.edu.co/editorial](http://www.uelbosque.edu.co/editorial)

Editor: Gustavo Zuluaga Hoyos

Dirección gráfica y diseño: Alejandro Gallego

Diagramación: Nicolás González

Corrección de estilo: Anne Leidy Cárdenas Giraldo

Impresión

KIMPRES

Cll 19 sur # 69c-17, PBX: 413 6884, Bogotá, D. C.

Impreso en Colombia

Julio de 2016

Todos los derechos reservados. Esta publicación no puede ser reproducida ni total ni parcialmente, ni entregada o transmitida por un sistema de recuperación de información, en ninguna forma ni por ningún medio, sin la autorización previa de los titulares de los derechos de autor.

## Tabla de contenido //

Introducción .....	13
<b>17 / Actividad 1: Introducción a Java y a los computadores</b>	
• Objetivos .....	17
• Introducción .....	17
• Ejercicios.....	18
<b>21 / Actividad 2: Entornos de desarrollo</b>	
• Objetivo.....	21
• Introducción .....	21
• Materiales para utilizar.....	21
• Prerrequisitos.....	22
• Ejercicios.....	22
• Referencias bibliográficas.....	35
<b>37 / Actividad 3: Cómo resolver un algoritmo</b>	
• Objetivos .....	37
• Introducción .....	37
• Ejercicios.....	38
<b>39 / Actividad 4: Uso de variables</b>	
• Objetivos .....	39
• Introducción .....	39
• Prerrequisitos.....	40
• Ejercicios.....	41
<b>45 / Actividad 5: Mi primer objeto</b>	
• Objetivos .....	43
• Introducción .....	43
• Prerrequisitos.....	44
• Ejercicios.....	44
<b>47 / Actividad 6: Sobrecarga de métodos</b>	
• Objetivos .....	47
• Introducción .....	47
• Prerrequisitos.....	48
• Ejercicios.....	48

<b>51 / Actividad 7: Expresiones aritméticas</b>	
• Objetivos .....	51
• Introducción .....	51
• Prerrequisitos.....	52
• Ejercicios.....	52
<b>55 / Actividad 8: Introducción a Eclipse</b>	
• Objetivos .....	55
• Introducción .....	55
• Prerrequisitos.....	56
• Ejercicios.....	56
<b>59 / Actividad 9: La clase String</b>	
• Objetivos .....	59
• Introducción .....	59
• Prerrequisitos.....	60
• Ejercicios.....	60
<b>63 / Actividad 10: Uso del condicional simple</b>	
• Objetivos .....	63
• Introducción .....	63
• Prerrequisitos.....	64
• Ejercicios.....	65
<b>67 / Actividad 11: Uso del condicional múltiple</b>	
• Objetivos .....	67
• Introducción .....	67
• Prerrequisitos.....	69
• Ejercicios.....	69
<b>71 / Actividad 12: Uso de Ciclos</b>	
• Objetivos .....	71
• Introducción .....	71
• Prerrequisitos.....	73
• Ejercicios.....	73
<b>75 / Actividad 13: Uso de arreglos</b>	
• Objetivos .....	75
• Introducción .....	75
• Prerrequisitos.....	77
• Ejercicios.....	77



<b>81 /</b>	<b>Actividad 14: Arreglo de objetos</b>	
	• Objetivos .....	81
	• Introducción .....	81
	• Prerrequisitos.....	83
	• Ejercicios.....	83
<b>85 /</b>	<b>Actividad 15: Herencia</b>	
	• Objetivos .....	85
	• Introducción .....	85
	• Prerrequisitos.....	86
	• Ejercicios.....	87
<b>89 /</b>	<b>Actividad 16: Polimorfismo</b>	
	• Objetivos .....	89
	• Introducción .....	89
	• Prerrequisitos.....	91
	• Ejercicios.....	92
<b>95 /</b>	<b>Actividad 17: Clases abstractas</b>	
	• Objetivos .....	95
	• Introducción .....	95
	• Prerrequisitos.....	99
	• Ejercicios.....	99
<b>101 /</b>	<b>Actividad 18: Interfaces</b>	
	• Objetivos .....	101
	• Introducción .....	101
	• Prerrequisitos.....	103
	• Ejercicios.....	104
<b>105 /</b>	<b>Actividad 19: Excepciones</b>	
	• Objetivos .....	105
	• Introducción .....	105
	• Prerrequisitos.....	106
	• Ejercicios.....	106
	• Referencias bibliográficas.....	108
<b>109 /</b>	<b>Actividad 20: Archivos y directorios</b>	
	• Objetivos .....	109
	• Introducción .....	109
	• Prerrequisitos.....	110

	• Ejercicios.....	110
	• Referencias bibliográficas.....	113
<b>115 /</b>	<b>Actividad 21: Serialización de objetos</b>	
	• Objetivos.....	115
	• Introducción.....	115
	• Prerrequisitos.....	116
	• Ejercicios.....	116
	• Referencias bibliográficas.....	117
<b>119 /</b>	<b>Actividad 22: Gestión de memoria</b>	
	• Objetivos.....	119
	• Introducción.....	119
	• Prerrequisitos.....	121
	• Ejercicios.....	121
	• Referencias bibliográficas.....	122
<b>123 /</b>	<b>Actividad 23: Recursividad</b>	
	• Objetivos.....	123
	• Introducción.....	123
	• Prerrequisitos.....	124
	• Ejercicios.....	124
	• Referencias bibliográficas.....	125
<b>127 /</b>	<b>Actividad 24: Algoritmos de ordenamiento</b>	
	• Objetivos.....	127
	• Introducción.....	127
	• Prerrequisitos.....	128
	• Ejercicios.....	128
	• Referencias bibliográficas.....	130
<b>131 /</b>	<b>Actividad 25: Algoritmos de búsqueda</b>	
	• Objetivos.....	131
	• Introducción.....	131
	• Prerrequisitos.....	132
	• Ejercicios.....	132
	• Referencias bibliográficas.....	133
<b>135 /</b>	<b>Actividad 26: Hilos</b>	
	• Objetivos.....	135
	• Introducción.....	135

• Prerrequisitos.....	136
• Ejercicios.....	137
• Referencias bibliográficas.....	138
<b>139 / Actividad 27: Sockets</b>	
• Objetivos.....	139
• Introducción.....	139
• Prerrequisitos.....	141
• Ejercicios.....	141
• Referencias bibliográficas.....	146
<b>147 / Actividad 28: Colecciones</b>	
• Objetivos.....	147
• Introducción.....	147
• Prerrequisitos.....	148
• Ejercicios.....	149
• Referencias bibliográficas.....	150
<b>151 / Actividad 29: Genéricos</b>	
• Objetivos.....	151
• Introducción.....	151
• Prerrequisitos.....	155
• Ejercicios.....	155
• Referencias bibliográficas.....	156
<b>157 / Actividad 30: Acceso a base de datos</b>	
• Objetivos.....	157
• Introducción.....	157
• Prerrequisitos.....	164
• Ejercicios.....	164
• Referencias bibliográficas.....	165



# Introducción

---

Más allá de ahondar en el debate sobre cuál debería ser el paradigma apropiado para la enseñanza de la programación en los primeros semestres, hay que lograr el objetivo de brindar a los estudiantes las herramientas y los mecanismos necesarios que les permitan construir programas para resolver problemas. Esto se logra mediante la enseñanza de un determinado lenguaje de programación, de conceptos propios del mismo, y de métodos y técnicas que faciliten abordar cualquier tipo de problema a través de una metodología que parta de su especificación y llegue hasta una solución correcta del mismo.

Actualmente la mayoría de las personas inmersas en el desarrollo de *software* considera que la programación orientada a objetos (POO) es el mejor paradigma para la construcción de sistemas informáticos. Sin embargo, la finalidad de este libro no es enseñar tal paradigma, ni mucho menos al que no lo conoce. Pretende, más bien, ofrecer una serie de actividades prácticas, de modo que los estudiantes, desde su inicio, puedan comprenderlo, y adquieran habilidades en el desarrollo de solu-

---

ciones para diversos escenarios problemáticos a partir de ciertas especificaciones y pautas dadas por el docente en el aula de clases.

Como producto del quehacer docente y del trabajo realizado en el aula de clases por varios años con estudiantes de primeros semestres, el objetivo principal de este libro es, en síntesis, apoyar el aprendizaje de la programación orientada a objetos, a través del uso del lenguaje de programación Java y de actividades prácticas. La selección del lenguaje de programación obedece al auge que Java ha tenido en los últimos años y a su fuerte vínculo con el mundo de la *internet*, dada su constante innovación con relación a la programación para la red. Sin embargo, bien podría utilizarse cualquier otro lenguaje de programación para resolver los ejercicios planteados en cada una de las actividades del libro.

La enseñanza de la programación orientada a objetos se ha convertido en una labor bastante compleja, debido a las exigencias que implica. Este paradigma ha sido considerado, en los últimos años, como uno de los más grandes avances dentro de los lenguajes de programación. Las nuevas características de este tipo de lenguaje han influenciado a la industria de desarrollo de *software* y han permitido la construcción de sistemas robustos y complejos. El aprendizaje de la POO requiere creatividad, dedicación, horas de estudio, lógica y una gran cantidad de aspectos conceptuales. Por todo lo anterior, el estudiante debe dedicar tiempo y esfuerzo al estudio teórico de los conceptos para realizar apropiadamente las actividades prácticas en el aula de clase.

Esta obra se encuentra estructurada en dos partes. La primera de ellas comprende dieciocho actividades prácticas iniciales, cuyo propósito es familiarizar al estudiante con el manejo de conceptos propios de la programación orientada a objetos y su aplicabilidad. La segunda, contiene actividades que buscan profundizar en algunos temas más avanzados.

Las primeras cinco actividades introducen al estudiante en el estudio conceptual de la POO y constituyen la base fundamental para la comprensión conceptual de cualquier tipo de lenguaje orientado a objetos. El estudiante, apoyado en las sesiones teóricas, se familiarizará con el entorno de desarrollo y podrá identificar una serie de conceptos

---

fundamentales. Así mismo, podrá resolver problemas relacionados con cada uno de los temas estudiados. Se sugiere no omitir el estudio de estas primeras cinco actividades.

Las demás actividades están diseñadas para apoyar externamente la labor realizada por el docente en el aula de clases, de tal manera que el estudiante pueda adquirir las aptitudes necesarias para proponer soluciones informáticas. Además, el desarrollo de estas actividades permite profundizar un poco más en los conceptos estudiados en las primeras cinco, puesto que trata de dar solución a problemas de mayor complejidad y permite, por supuesto, adquirir nuevos conocimientos.

Al finalizar las actividades cualquier estudiante tendrá la capacidad de:

- Hacer uso de los conocimientos adquiridos para darle solución a diferentes problemas mediante el uso del lenguaje de programación Java.
- Debatir sobre la sintaxis y la estructura del lenguaje de programación Java, y sobre la definición de métodos, constructores, entre otros conceptos.
- Aplicar los conceptos discutidos a la planificación de diversos programas en el lenguaje de programación Java.
- Analizar y plantear soluciones lógicas bajo el paradigma de la programación orientada a objetos mediante el uso del lenguaje de programación Java.
- Analizar, diseñar, codificar, compilar, ejecutar y corregir programas utilizando un computador, a través de la programación en Java.





---

## Actividad 1 //

# Introducción a Java y a los computadores

## Objetivos

- Comprender la importancia del lenguaje de programación Java en la actualidad y algunos conceptos propios de la programación orientada a objetos.
- Identificar cuáles son las partes principales de un computador.

## Introducción

Se puede decir que un computador es un sistema capaz de procesar grandes volúmenes de información. Recibe datos, órdenes y programas como entrada (por medio de diversos dispositivos) y proporciona un resultado como salida (también por medio de diversos dispositivos). Los computadores pueden estar interconectados, intercambiar mensajes y archivos, y compartir recursos. En la actualidad existen computadores en cualquier parte del mundo, como los celulares, y un sinnúmero de

---

electrodomésticos que incorporan uno o varios procesadores. Sin embargo, los computadores, los celulares y otros dispositivos tienen características especiales y propósitos diferentes.

Un computador está compuesto por un procesador, una memoria, un disco duro, entre otros componentes, y en términos de *software* posee un programa denominado sistema operativo. Un programa no es más que una serie de instrucciones que se ejecutan para determinados fines. Actualmente, la mayor parte del código utilizado para los sistemas operativos se escribe en C++ y en Java.

Hoy en día, Java es uno de los lenguajes de programación más importantes. Todo programa ejecutado en Java debe ser compilado, y el código generado (*bytecodes*) debe ser interpretado por una máquina virtual. El código compilado se podrá ejecutar en máquinas virtuales independientemente de la plataforma en donde se haya realizado el proceso de compilación. Java es un lenguaje de propósito general orientado a objetos. Quienes hayan programado en C o C++ podrán darse cuenta de que su sintaxis es similar, pero se debe tener en cuenta que Java no es una evolución de estos lenguajes de programación.

## Ejercicios

El estudiante debe realizar un informe que incluya la siguiente información:

- ¿Qué es Java y cuál es su versión más reciente?
- Una reseña de la historia de Java.
- ¿Cuáles son las características principales de Java?
- Diferencias entre las versiones SE (*Standard Edition*), EE (*Enterprise Edition*) y ME (*Micro Edition*) de Java.
- ¿Qué es la programación orientada a objetos?
- ¿Qué es una clase y qué es un objeto en la programación orientada a objetos?
- ¿Qué es un código fuente?
- ¿Qué es un código máquina?
- ¿Qué es un *bytecodes*?

- 
- ¿Qué es un compilador?
  - ¿Qué es la máquina virtual de Java?
  - ¿Cuál es la instrucción para invocar el compilador de Java y cómo funciona?
  - ¿Cuál es la instrucción para invocar la máquina virtual de Java y cómo funciona?
  - ¿Qué es un computador?
  - ¿Cuáles son los componentes físicos principales de un computador?
  - ¿Un reproductor de DVD es un computador? Discúptalo.
  - ¿Qué es la consola de comandos?
  - ¿Cómo funcionan las siguientes instrucciones de la consola de comandos?
    - Mostrar los archivos del directorio.
    - Cambiar de directorio.
    - Cambiar de unidad de disco.
    - Subir un nivel de directorio.
    - Cambiar el nombre de un archivo.
    - Borrar un archivo.



---

## Actividad 2 //

# Entornos de desarrollo

### Objetivo

- Instalar y configurar la plataforma Java y su kit de desarrollo JDK.

### Introducción

Java no es sólo un lenguaje de programación, sino también una plataforma que permite el desarrollo y la ejecución de programas en diferentes sistemas y equipos.

Java se encuentra en distintas versiones: JEE (*Enterprise Edition*), JSE (*Standard Edition*), JME (*Micro Edition*), entre otras. La versión que se usará en estas guías es la JSE.

### Material para utilizar

En Java, y en general en cualquier desarrollo de *software*, existe un *software* que provee herramientas de desarrollo para crear programas. Di-

cho *software* se denomina JDK (*Java Development Kit* o Kit de desarrollo de Java) y Oracle es la empresa que lo distribuye actualmente de manera gratuita. La versión que se puede descargar en este momento es la 8 del JDK, la cual se usará en los ejercicios planteados a lo largo de este libro. Para realizar esta actividad se recomienda trabajar en un equipo de pruebas o en una máquina virtual. Si no tiene experiencia en la instalación de Java, es preferible no usar el computador personal.

## Prerrequisitos

- Para realizar este ejercicio es necesario conocer el uso de la consola de comandos de Windows.
- Esta actividad se desarrolla desde un computador con sistema operativo Windows 7. La instalación en sistemas Windows 8, Vista y XP es similar. En esta actividad no se explica la instalación en otros sistemas como Mac OS o Linux.

## Ejercicios

1. Para descargar la última versión del JDK debe ir a la página web de Oracle ([www.oracle.com](http://www.oracle.com)) y seleccionar la sección de descargas (*Downloads*). Allí, debe ir a la opción de Java para desarrolladores (*Java for Developers*).

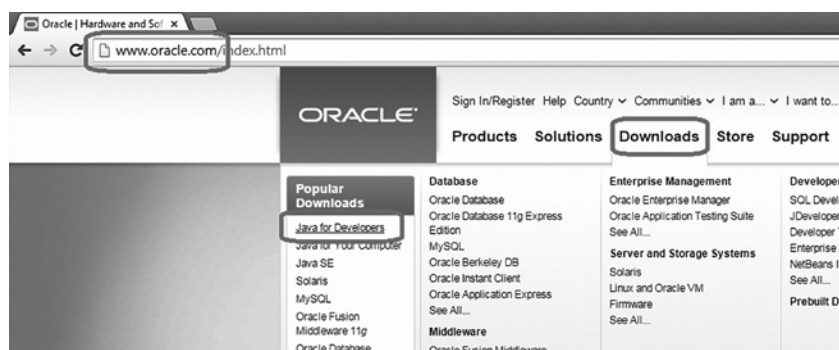
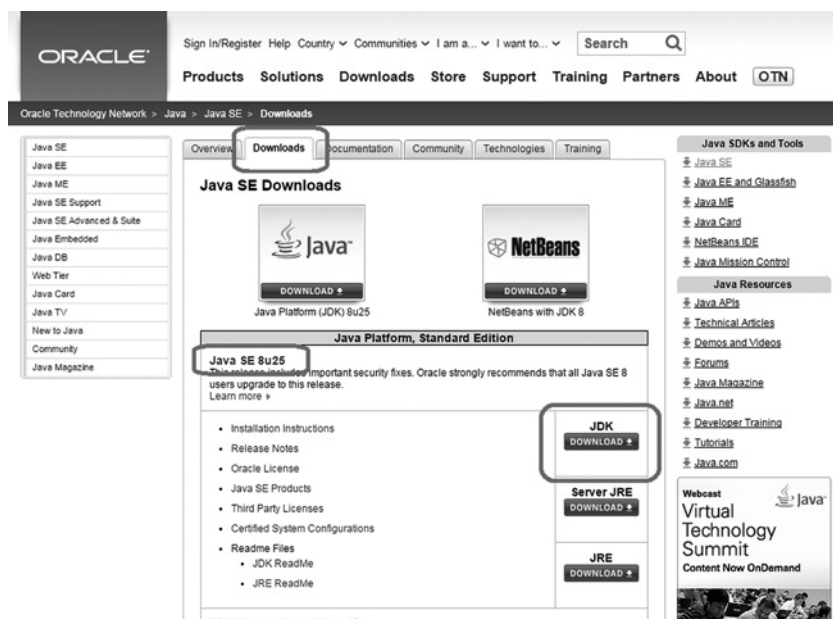


Figura 1. Sitio web de Oracle.

La página lo enviará a la sección de descargas, específicamente a la de Java SE (Java Standard Edition). En este caso debe navegar en la parte inferior de la página web y buscar la opción de “Java SE 8 Update 25” (o la versión de actualización disponible en el momento de desarrollar la actividad).



**Figura 2.** Sitio de descargas del JDK.

En esta sección encontrará las opciones para descargar el JDK y el JRE. No descargue el JRE (*Java Runtime Environment*), entorno de ejecución de Java que permite ejecutar el código generado por el compilador. La versión que debe descargar es el JDK que incluye, por supuesto, el JRE para poder ejecutar el código creado.

La página web le solicitará que acepte el acuerdo de licencia (*License Agreement*) y que seleccione la versión del JDK que necesite. Actualmente se suele instalar el sistema operativo Windows de 64 *bits*. Si este es su caso, debe escoger la versión Windows x64; en caso contra-

rio, debe ser Windows x86. Si va a hacer la instalación en un computador personal o en otro sistema operativo, escoja la versión que corresponda.

Java SE Development Kit 8u25		
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.		
<input type="radio"/> Accept License Agreement <input checked="" type="radio"/> Decline License Agreement		
Product / File Description	File Size	Download
Linux x86	135.24 MB	jdk-8u25-linux-i586.rpm
Linux x86	154.88 MB	jdk-8u25-linux-i586.tar.gz
Linux x64	135.6 MB	jdk-8u25-linux-x64.rpm
Linux x64	153.42 MB	jdk-8u25-linux-x64.tar.gz
Mac OS X x64	209.13 MB	jdk-8u25-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	137.01 MB	jdk-8u25-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	97.14 MB	jdk-8u25-solaris-sparcv9.tar.gz
Solaris x64 (SVR4 package)	137.11 MB	jdk-8u25-solaris-x64.tar.Z
Solaris x64	94.24 MB	jdk-8u25-solaris-x64.tar.gz
Windows x86	157.26 MB	jdk-8u25-windows-i586.exe
Windows x64	169.62 MB	jdk-8u25-windows-x64.exe

Figura 3. Versiones de descarga del JDK.

Descargue el JDK y guárdelo en un sitio donde pueda ejecutarlo posteriormente; por ejemplo, en su carpeta de descargas o en el escritorio del computador donde esté trabajando. La descarga en una conexión de 5 Mbps debe tardar entre 5 y 10 minutos.

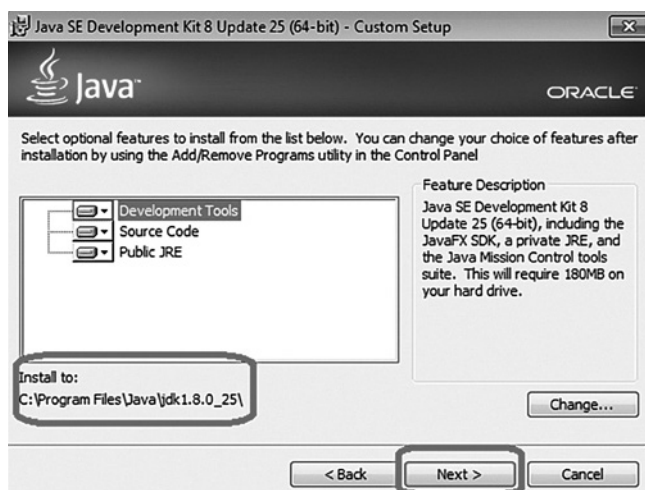
Una vez descargado, haga doble clic en el archivo o ejecútelo. Se debe abrir un asistente de instalación. Haga clic en el botón *Next* (siguiente).





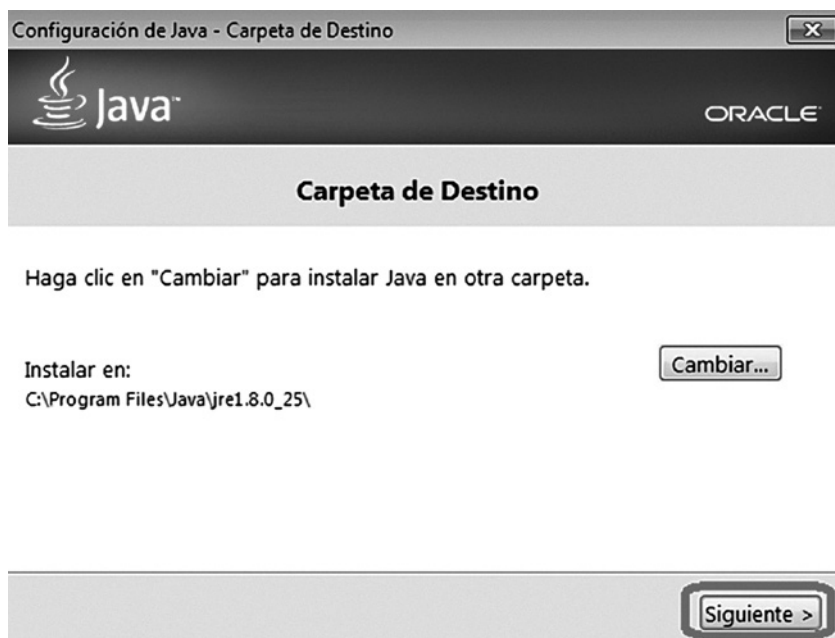
**Figura 4.** Inicio del asistente de instalación.

El asistente le preguntará por los componentes que desea instalar; por recomendación, deberá escogerlos todos. También le pedirá que seleccione la carpeta donde instalará el JDK, y deberá dejar el sitio recomendado. Haga clic en el botón *Next* (siguiente).



**Figura 5.** Opciones de selección de los componentes para instalar.

Ahora iniciará el proceso de instalación de Java. El asistente le preguntará por el lugar donde desea instalar los archivos del JRE de Java (recuerde que este componente es necesario para ejecutar los programas que construya en Java). Debe dejar la ubicación que aparece por defecto. Haga clic en *Next* (siguiente).



**Figura 6.** Sitio de instalación del JRE.

Si hizo todo correctamente, se debe desplegar una ventana indicándole que la instalación del JDK fue exitosa. Haga clic en el botón Close (cerrar) para terminar el asistente.

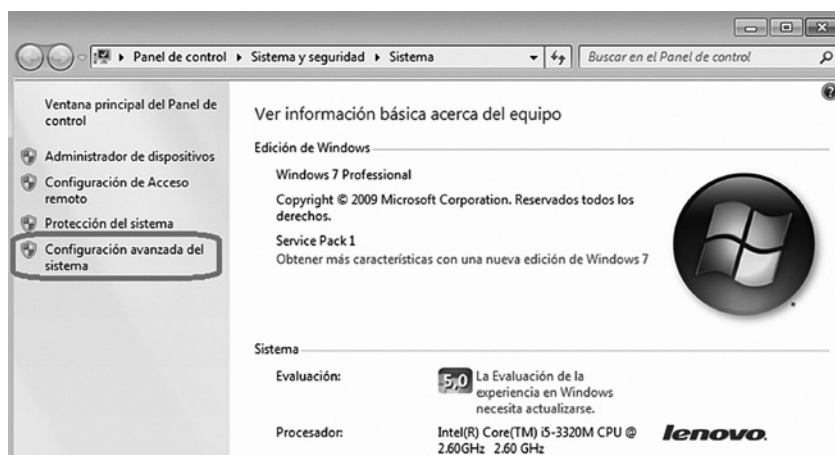


**Figura 7.** *Instalación exitosa del JDK.*

Aún la configuración no ha terminado.

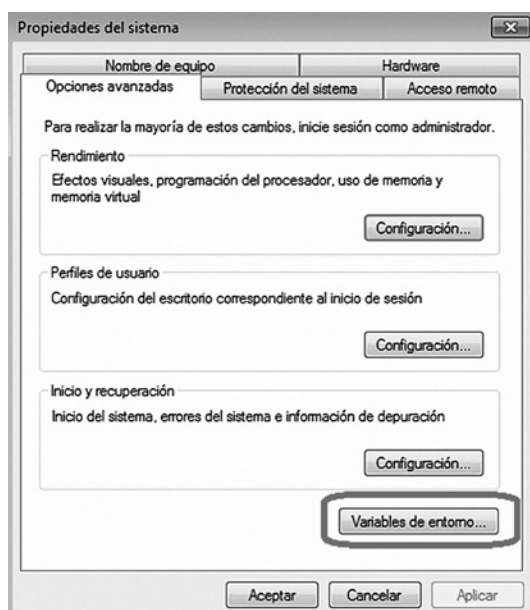
Para invocar el compilador de Java (javac), necesario para convertir el código fuente en programas ejecutables desde cualquier carpeta o entorno de desarrollo (IDE), hay que configurar las variables de entorno del sistema operativo. Las variables de entorno son cadenas o textos que contienen información acerca del entorno para el sistema y para el usuario que inicie sesión. Algunos programas usan la información para determinar dónde se ubican los archivos (como los archivos temporales) [1]. En este caso en particular se debe cambiar la variable `PATH`. Esta variable del sistema utiliza el sistema operativo para buscar los ejecutables necesarios desde la línea de comandos o la ventana terminal [2].

Para cambiar esta variable, debe hacer clic con el botón derecho del mouse en el ícono “Equipo” e ir a “Propiedades”. En el menú que se despliegue, haga clic en “Configuración avanzada del sistema”.



**Figura 8.** Configuración avanzada del sistema.

Debe abrir la ventana “Propiedades del sistema” y seleccionar la pestaña “Opciones avanzadas”. Busque al final de la pestaña el botón “Variables de entorno” y haga clic en él.



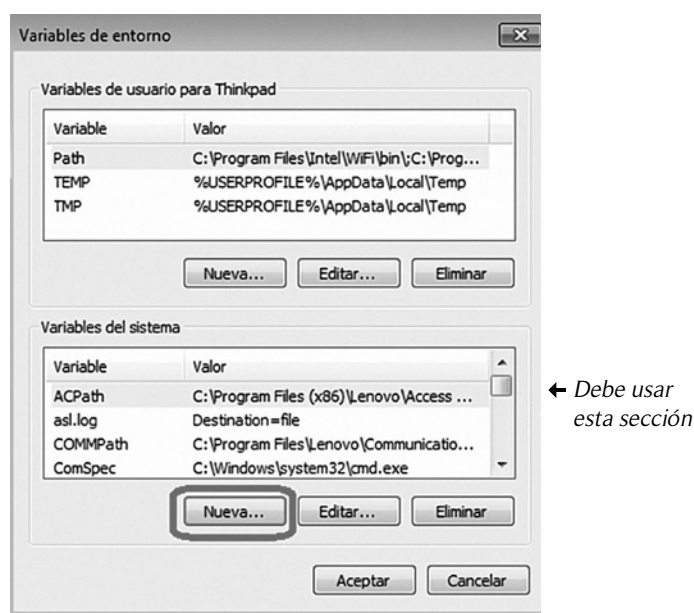
**Figura 9.** Menú de las Propiedades del Sistema.

A continuación se debe desplegar otra ventana llamada “Variables de entorno”.

A partir de este punto debe seguir las instrucciones cuidadosamente. Cualquier cambio incorrecto en las variables de entorno puede hacer que algún programa instalado funcione incorrectamente. De nuevo, se recomienda enfáticamente realizar esta actividad en los computadores de su institución educativa o en un equipo de pruebas (puede ser incluso una máquina virtual) antes de hacerlo en un computador personal o familiar.

En la ventana “Variables de entorno” se usará la zona “Variables del sistema”.

Para empezar, debe crear una nueva variable en las “Variables del sistema”. Haga clic en el botón “Nueva...”



**Figura 10.** Configuración de las variables de entorno.

El nombre de la variable es `JAVA_HOME` (escríbalo en mayúsculas). El valor de la variable es la ubicación donde instaló el JDK. En

computadores con sistema operativo en inglés, la dirección debe ser algo como “C:\Program Files\Java\jdk1.8.0\_25”. En aquellos cuyo sistema operativo está en español, la ubicación debe ser “C:\Archivos de programa\Java\jdk1.8.0\_20”. Verifique que en realidad exista la carpeta: navegue hasta la carpeta para confirmarlo; de no ser así, haga los ajustes necesarios en el valor de la variable. Haga clic en el botón “Aceptar” para guardar y continuar.



**Figura 11.** Creación de la variable `JAVA_HOME`.

Ahora debe modificar la variable `PATH` (no confundir con `Classpath`, esa es otra variable), que se encuentra en el listado de las variables del sistema. Selecciónela y haga clic en “Editar...”

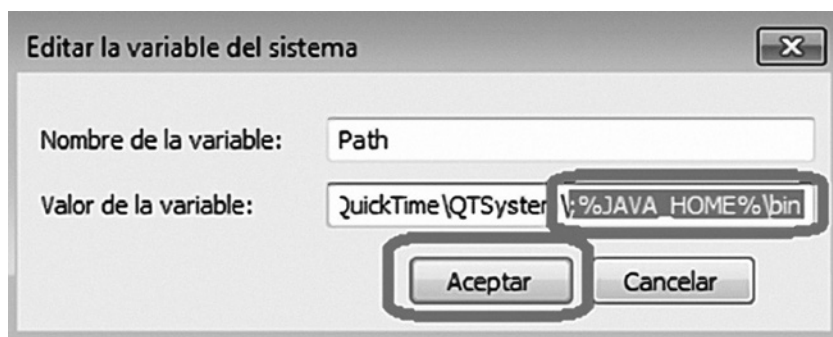


**Figura 12.** Variable PATH (recuerde que se va a modificar, no a eliminar).

Es importante que no borre ni modifique el nombre de la variable. Esto podría hacer que el computador falle. En el valor de la variable vaya al final de la línea que ya existe. Agregue exactamente lo siguiente, incluyendo el punto y coma al inicio de la orden:

```
; %JAVA_HOME%\bin
```

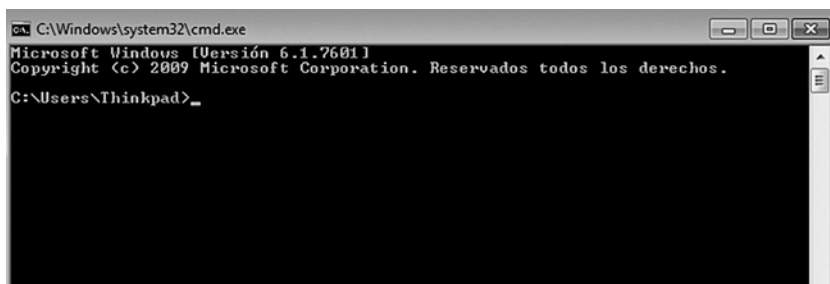
Este valor agregará la carpeta *bin* de Java a la variable PATH. De este modo se asegura que la ventana de comandos y el sistema operativo, en general, puedan encontrar las órdenes necesarias de Java para revisar, compilar y ejecutar su código. Haga clic en “Aceptar” para guardar y continuar.



**Figura 13.** *Modificación de la variable PATH.*

Haga nuevamente clic en “Aceptar” para guardar y cerrar la ventana de “Variables de entorno”. Realice la misma acción en la ventana “Propiedades del sistema”.

Ahora debe probar que la instalación se hizo correctamente. Para esto debe abrir una nueva consola o ventana de comandos, hacer clic en “Inicio” y escribir `cmd`. Debe aparecer la opción de ventana de comandos. Iníciela.



**Figura 14.** *Consola de comandos de Windows.*

Escriba la orden `javac`. Debe obtener el siguiente resultado; de no ser así, debe revisar nuevamente las instrucciones y verificar qué pudo fallar en la configuración.





```
C:\Windows\system32\cmd.exe
Microsoft Windows [Versión 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. Reservados todos los derechos.

C:\Users\Thinkpad>javac
Usage: javac <options> <source files>
where possible options include:
  -g                        Generate all debugging info
  -g:none                  Generate no debugging info
  -g:{lines,vars,source}   Generate only some debugging info
  -nowarn                  Generate no warnings
  -verbose                 Output messages about what the compiler is doing
  -deprecation             Output source locations where deprecated APIs are used
  -classpath <path>       Specify where to find user class files and annotations
  -processorpath <path>   Specify where to find user class files and annotations
  -sourcepath <path>      Specify where to find input source files
  -bootclasspath <path>  Override location of bootstrap class files
  -extdirs <dirs>         Override location of installed extensions
  -endoredirs <dirs>     Override location of endorsed standards path
  -proc:{none,only}       Control whether annotation processing and/or compilation is done.
  -processor <class1>[,<class2>,<class3>...] Names of the annotation processors to run; bypasses default discovery process
  -processorpath <path>   Specify where to find annotation processors
  -parameters             Generate metadata for reflection on method parameters
  -d <directory>          Specify where to place generated class files
  -s <directory>          Specify where to place generated source files
  -h <directory>          Specify where to place generated native header files
  -implicit:{none,class}  Specify whether or not to generate class files for implicitly referenced files
  -encoding <encoding>    Specify character encoding used by source files
  -source <release>        Provide source compatibility with specified release
  -target <release>        Generate class files for specific VM version
  -profile <profile>       Check that API used is available in the specified platform
  -version                Version information
  -help                   Print a synopsis of standard options
  -Akey[=value]            Options to pass to annotation processors
  -X                      Print a synopsis of nonstandard options
  -J<flag>                 Pass <flag> directly to the runtime system
  -Werror                  Terminate compilation if warnings occur
  @<filename>              Read options and filenames from file

C:\Users\Thinkpad>_
```

**Figura 15.** Prueba de ejecución de javac (el compilador de Java).

Si desea, puede hacer la prueba con un primer programa. Abra el “bloc de notas” de su computador y escriba el siguiente código de programa:

```
class HolaMundo
{
    public static void main(String[] args)
    {
        System.out.print ("Hola mundo, soy
el primer programa en Java");
    }
}
```

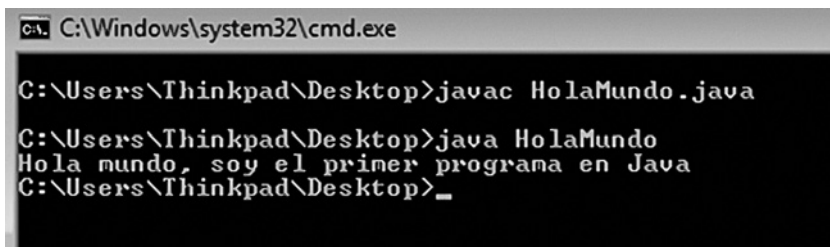
Guarde el archivo con el nombre `HolaMundo.java`. En la ventana de comandos, diríjase a la carpeta donde guardó el archivo que acaba de crear y ejecute la siguiente orden:

```
javac HolaMundo.java
```

En la carpeta donde esté guardado el archivo se debe haber creado un nuevo archivo llamado `HolaMundo.class`. Ahora ejecute la orden:

```
java HolaMundo
```

Debe aparecer el mensaje del código como se ve en la Figura 16:

A screenshot of a Windows command prompt window. The title bar at the top reads "C:\Windows\system32\cmd.exe". The command prompt shows the following sequence of commands and output:  
C:\Users\Thinkpad\Desktop>javac HolaMundo.java  
C:\Users\Thinkpad\Desktop>java HolaMundo  
Hola mundo, soy el primer programa en Java  
C:\Users\Thinkpad\Desktop>\_

**Figura 16.** *Primer programa en Java.*

2. Después de realizar el ejercicio número 1, el estudiante deberá presentar un informe técnico en donde responda a los siguientes interrogantes:
  - a. ¿Qué inconvenientes se presentaron durante el proceso de instalación?
  - b. ¿Cómo fueron solucionados?
  - c. Con respecto a las variables de entorno, ¿qué aprendió?
  - d. ¿Considera usted que el proceso de instalación en el sistema operativo Windows es similar al proceso de instalación en un sistema operativo Linux? Justifique su respuesta.

---

## Referencias bibliográficas

- [1] Microsoft (2014). Cómo administrar variables de entorno en Windows XP. Recuperado de: <http://support.microsoft.com/kb/310519/es> (último acceso: 17/12/2014).
- [2] Oracle (2014). ¿Cómo puedo establecer o cambiar la variable del sistema PATH? Recuperado de <http://www.java.com/es/download/help/path.xml> (último acceso: 17/12 /2014).



---

## Actividad 3 //

# Cómo resolver un algoritmo

### Objetivos

- Comprender el concepto de algoritmo e identificar los conceptos de pseudocódigo y de diagrama de flujo.
- Analizar y aplicar los conceptos estudiados.

### Introducción

Un algoritmo es un conjunto de instrucciones o reglas bien definidas, ordenadas y con un fin específico que permite resolver un problema mediante su ejecución. Este se elabora a través de la combinación de las siguientes estructuras (conocidas como estructuras de control):

- *Secuencia simple*: la secuencia simple es un listado de instrucciones que tiene un orden. Al escribir esta estructura se entiende que es muy importante mantener el orden de las tareas que la componen.

- *Selección simple*: dada una condición (que puede ser verdadera o falsa), la selección simple permite ejecutar o no un conjunto de instrucciones asociadas con la misma.
- *Selección compuesta*: a diferencia de la selección simple, la selección compuesta valida los posibles valores que pueda tener una variable. Según sea dicho valor, el algoritmo ejecutará las instrucciones asociadas con el mismo.
- *Ciclos o iteraciones*: en esta estructura se presenta una situación similar a la de la selección simple, en la que se tiene la condición de que mientras sea verdadera ejecutará cíclicamente las instrucciones asociadas. Cuando su condición cambie a falsa, el ciclo finalizará. Se debe tener cuidado con la definición de esta estructura porque el ciclo debe permitir que, en algún momento, la condición cambie a falsa.

Existen dos formas principales de representar un algoritmo. La primera es mediante una gráfica llamada diagrama de flujo y la segunda, por medio de una escritura informal llamada pseudocódigo.

## Ejercicios

Construya un algoritmo (en pseudocódigo o en diagrama de flujo) para cada uno de los siguientes items:

- Hacer una llamada a larga distancia.
- Prepararse para ir a la universidad a clase de 7 a.m.
- Elevar cualquier número a la tercera potencia.
- Llevar el marcador de un partido de fútbol durante los 90 minutos de su duración.
- Ordenar a 10 alumnos, por orden de estatura.
- Según el color de un semáforo, determinar qué debe hacer el conductor.
- Determinar cuál es el número mayor entre 2 números.
- Determinar cuál es el número mayor entre 3 números.
- Realizar la multiplicatoria de los números del 47 al 129 (realice un ciclo).

- Organizar un mazo de cartas (no se sabe cuántas cartas tiene) en cuatro pilas, cada una de un palo.

---

## Actividad 4 //

# Uso de variables

## Objetivos

- Aprender sobre las variables.
- Reconocer la sintaxis de las variables y del operador de asignación en Java.
- Conocer algunos de los problemas que presenta el uso de variables en Java.

## Introducción

Una variable es una ubicación de almacenamiento en la memoria de un computador que tiene tipo, nombre y contenido. Las variables son usadas en Java, principalmente, para almacenar números o caracteres (las cadenas de caracteres no se almacenan en variables). El tipo de dato de una variable se escoge según el valor que se va a almacenar. Un número entero se puede almacenar en una variable de tipo `int`, mientras que un carácter se puede almacenar en una variable de tipo `char`.

---

El nombre del dato debe representar el valor que se va a almacenar en la variable. Por ejemplo, si se va a almacenar la edad de un estudiante un buen nombre podría ser `edadEstudiante`, mientras que un mal nombre sería `ee` (refiriéndose a las iniciales de edad de estudiante). El contenido debe estar dentro del rango y el dominio que permite el tipo de dato. Por ejemplo, si una variable se declara de tipo `short` (más pequeño que un `int`), dicha variable puede contener valores enteros entre -32.768 y 32.767. Así, el número 40.000 no se podría almacenar en esta variable.

En Java, una variable se declara de la siguiente forma:

```
tipo nombreDeVariable = valor;
```

Por ejemplo, una variable de tipo “double”, llamada `pi`, con un valor de 3.141592 se declara de la siguiente forma:

```
double pi = 3.141592;
```

La asignación del valor no es obligatoria en el mismo momento de la declaración de la variable. Es posible crear la variable y posteriormente asignarle el valor:

```
double pi;  
pi = 3.141592;
```

Debe tener cuidado al asignar valores a las variables de tipo `char`, ya que en este caso es necesario el uso de las comillas simples:

```
char primeraLetra = 'a';
```

## Prerrequisitos

- Instalar y configurar Java correctamente en un equipo (según la actividad número 2 de este libro).



# Ejercicios

Antes de empezar revise la documentación de Java para verificar cuáles son los valores que se pueden almacenar en las variables de tipo `int`, `byte`, `short`, `long`, `double`, `float`, `char` y `boolean`.

1. En la Tabla 1 cite un ejemplo de un valor que se pueda almacenar y otro que no en una variable declarada.

**Tabla 1.** *Tipos de datos y valores permitidos.*

Tipo de dato	Nombre de la variable	Valor permitido	Valor no permitido
int			
byte			
short			
long			
double			
float			
char			
boolean			

2. Construya una clase pública llamada `ValorPermitido`. Declare un método `main` y todas las variables de la Tabla 1 sin asignar valores a las variables. Trate de compilar el archivo y ejecutarlo; ¿se produjo algún error?
3. Asigne valores erróneos o no permitidos a las variables e intente que salgan en la pantalla; ¿qué error se produjo? Trate de familiarizarse con los mensajes de error que muestre el compilador.

- 
4. Asigne valores correctos a las variables, después compile y ejecute la clase nuevamente. Haga cambios en los valores asignados a las variables para que se familiarice con la manera de asignar valores.
  5. Intente asignar los valores 14.33 y 14,33 al tipo de dato `double` o `float`; ¿cuál de los dos números funcionó? ¿Puede decir por qué?
  6. Intente cambiar el tipo de dato de una variable sin eliminar su declaración inicial. Por ejemplo:

```
int edad = 13;  
short edad = 13;
```

¿Qué sucede ahora?

---

## Actividad 5 //

# Mi primer objeto

### Objetivos

- Reconocer la relación entre un objeto y una clase.
- Familiarizarse con el proceso de modelado y creación de objetos.
- Diseñar clases de acuerdo con ciertos lineamientos.
- Crear programas para integrar los conceptos estudiados.

### Introducción

La programación orientada a objetos procura modelar objetos de la vida real y representarlos en *software*. Para hacer esto, se reconoce que los objetos pueden tener características (color, altura, tamaño) y comportamientos (subir escaleras, permitir entrada, acelerar, frenar). Antes de construir objetos, estos se deben definir y caracterizar. Para ello se hace una clase en la cual se definen las características y los comportamientos de los objetos de la misma clase. Por ejemplo, una fábrica puede hacer

50 carros (objetos); sin embargo, estos carros se fabrican según los planos de un diseñador (en nuestro caso, una clase). En dichos planos se definen las características que van a tener los carros y el comportamiento de los mismos.

En Java las características de los objetos se almacenan en variables de instancia, y los comportamientos se definen en métodos. La construcción de un objeto en Java se conoce como “instanciar” una clase, de manera que un objeto también se conoce como la instancia de una clase. En el ejemplo antes mencionado de los carros y la fábrica, cada carro construido es una instancia de los planos del diseñador fabricado tal y como se definió en los diseños.

## Prerrequisitos

- Instalar y configurar Java correctamente en un equipo (según la actividad práctica número 2 de este libro).

## Ejercicios

En la primera parte de este ejercicio se definirán los objetos tipo `libro` y `autor`. Posteriormente se creará un programa para probar la creación de los libros y los autores, y el correcto funcionamiento de los mismos.

Primero, se definirán los autores: estos tienen nombre y apellido. Si se pregunta por el nombre de un autor, se debe dar el nombre completo. El nombre se le asigna en su creación y no puede ser modificado.

Autor
-nombre: String -nombre: String
+ mostrarNombre ()

Instrucciones:

1. Defina una clase pública llamada `Autor`.
2. Defina una variable de instancia de tipo `String` llamada `nombre`.
3. Defina una variable de instancia de tipo `String` llamada `apellido`.
4. Defina un constructor que reciba el nombre y el apellido del autor e inicie las variables de instancia.
5. Defina un método `mostrarNombre()` que, al ser llamado, muestre el nombre completo del autor (nombre + apellido).

Ahora se definirán los libros. Estos tienen un título, un autor, un ISBN (número de identificación) y un número de páginas. Un libro se puede contruir solamente con estos tres elementos; el número de páginas es opcional. Una vez creado el libro, no se puede modificar ni el nombre, ni el ISBN, ni el autor.

Libro
-titulo: String -autor: Autor -isbn: String -numPaginas: int
+ cambiarNumPaginas(int nuevoNumPaginas) + mostrarNumPaginas() +mostrarIsbn +mostrarTitulo +mostrarAutor

Instrucciones:

1. Defina una clase pública llamada `Libro`.
2. Defina una variable de instancia de tipo `String` llamada `título`.
3. Defina una variable de instancia de tipo `Autor` llamada `autor`.
4. Defina una variable de instancia de tipo `String` llamada `isbn`.

5. Defina una variable de instancia de tipo `int` llamada `numPaginas`.
6. Defina un constructor que reciba el título del libro, el ISBN, el nombre y el apellido del autor, y que inicie estas variables de instancia. Debe tener cuidado con la inicialización del autor.
7. Construya un método `cambiarNumPaginas(int nuevoNumPaginas)` que actualice el número de páginas del libro.
8. Construya un método `mostrarNumPaginas()` que muestre el número de páginas del libro.
9. Construya un método `mostrarIsbn()` que muestre el número ISBN del libro.
10. Construya un método `mostrarTitulo()` que muestre el título del libro.
11. Construya un método `mostrarAutor()` que muestre el autor del libro.

Una vez construidas las clases de estos objetos, debe crear un programa que permita probar el funcionamiento de los mismos. Para esto debe hacer lo siguiente:

1. Defina una clase pública llamada `Biblioteca`.
2. Defina un método `main` (necesario para que la clase se comporte como un programa).

A continuación:

- Construya el libro *La insoportable levedad del ser* de Milan Kundera, con ISBN 35128 y 336 páginas.
- Construya el libro *La metamorfosis* de Franz Kafka, con ISBN 36995 y 224 páginas.
- Construya el libro *El hombre duplicado* de José Saramago, con ISBN 16998 y 407 páginas.
- Consulte el autor del primer libro y verifique que la información sea correcta.
- Consulte el ISBN del segundo libro y verifique que haya sido creado correctamente.

- Consulte el número de páginas del tercer libro. Cambie el número de páginas por 432 y verifique que el cambio se efectúe (use los métodos correspondientes).

---

## Actividad 6 //

# Sobrecarga de métodos

## Objetivos

- Entender por qué dos métodos aparentemente pueden tener el mismo nombre y comportarse de manera diferente.
- Estudiar en qué casos se puede hablar de sobrecarga y en qué casos no.

## Introducción

Cuando se define el comportamiento de un objeto, en algunas ocasiones resulta práctico definir dos comportamientos con el mismo nombre. Por ejemplo, en la práctica es similar sumar dos números que sumar tres: la tarea es simplemente sumar; sin embargo, resulta engorroso tener que poner un nombre a la tarea de sumar dos números y otro a la de sumar tres. Por esta razón, Java permite hacer métodos diferentes con el mismo nombre siempre y cuando se cumpla lo siguiente:

- Que cambie el número de argumentos.
- Que cambien los tipos de datos.

Es necesario aclarar que en Java cambiar únicamente el tipo de dato de retorno del método no habilita la sobrecarga. Para esto se deben cumplir las condiciones antes mencionadas.

## Prerrequisitos

- Instalar y configurar Java correctamente en un equipo (según la actividad práctica núm. 2 de este libro).
- Revisar los conceptos estudiados: clases, objetos, métodos y variables.

## Ejercicios

1. Construya una clase `Calculadora` que permita realizar la suma de dos y tres números. Para cada uno de los comportamientos se elaborará un método. El objeto no tendrá variables de instancia, sino que recibirá los valores y realizará el cálculo.

Calculadora
<pre>+ void suma(int num1, int num 2) + void suma(double num1, int num 2) + void suma(double num1, double num 2)</pre>

Los pasos son los siguientes:

1. Conserve el primer método `suma` en la clase `Calculadora` existente.
2. Modifique el segundo método `suma` para que primero reciba un número de tipo `double`, después uno de tipo `entero` y finalmente muestre en pantalla la suma de los números. El método no retornará ningún dato.



3. Construya un método `suma` que reciba dos números de tipo `double` y muestre en pantalla la suma de los flotantes. El método no retornará ningún dato.
4. Construya una clase llamada `Prueba` que permita crear un objeto de tipo `Calculadora`. Use el método `suma` para realizar la suma de 59 y 90, la de 59.6 y 62, y la de 5.44 y 14.22.

Para comprobar que la sobrecarga no aplica únicamente para el cambio del tipo de retorno del método, construya un método en la clase `Calculadora` que retorne un número entero en un método `suma` y que reciba dos números enteros y retorne la suma de los mismos.

Calculadora
+ void suma(int num1, int num 2) + void suma(double num1, int num 2) + void suma(double num1, double num 2) + void suma(int num1, int num 2)

Al tratar de compilar esta clase, se debió generar un error en tiempo de compilación. Revise de qué se trata.



---

## Actividad 7 //

# Expresiones aritméticas

### Objetivos

- Identificar las expresiones aritméticas en Java.
- Comprender y probar algunas reglas de precedencia.
- Practicar la elaboración de expresiones aritméticas.

### Introducción

En Java, expresiones como:

$$x_1 = \frac{-b + \sqrt{b^2 - 4ac}}{2a}$$

Se pueden escribir de la siguiente manera:

```
x1=-b+Math.sqrt((b*b)-(4*a*c))/(2*a)
```

En Java no existe la representación de ecuaciones, de modo que se debe trabajar con expresiones aritméticas de un nivel (un renglón). Por otra parte, las siguientes expresiones no representan el mismo resultado:

```
var1 = a*2+5-1/4
var1 = (a*(2+5)-1)/4
var1 = (a*2)+5-(1/4)
```

En la primera ecuación se siguen unas reglas llamadas reglas de precedencia, de modo que la expresión tiene un resultado que no es el mismo que el de las otras dos expresiones. Conviene conocer dichas reglas de precedencia.

## Prerrequisitos

- Reconocer las reglas de precedencia en Java.

## Ejercicios

Escribir una expresión en Java que represente las siguientes ecuaciones:

a.  $x_1 = \frac{3a}{b}$

b.  $var1 = a^2 + b^2$

c.  $total = 1 + \frac{1}{a+b}$

d.  $ganancia = p(1 + m)^n$

Sugerencia: Puede usar el método `Math.pow(a, b)`

e.  $valor = \frac{1}{\frac{1}{\frac{1}{x}}}$

$$f. \text{ variable} = p(1 + \frac{r}{100})^n$$

Sugerencia: Puede usar el método `Math.pow(a,b)`

$$g. \text{ distancia} = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Sugerencia: Puede usar los método `Math.pow(a,b)` y `Math.sqrt(a)`

$$h. \text{ total} = (a - b)^4 + \sqrt{7ba^2}$$

Sugerencia: Puede usar los método `Math.pow(a,b)` y `Math.sqrt(a)`

$$i. \text{ p} = \frac{x+1}{2} + \frac{x+3}{4} + \frac{x+7}{5}$$

$$j. \text{ x} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Sugerencia: recuerde que en este caso hay dos soluciones; escriba una para  $x_1$  y otra para  $x_2$

$$k. \text{ area} = \pi r^2$$

Sugerencia: Puede usar la constante `Math.PI`.

$$l. \cos \alpha = (2 \cos \frac{1}{2}(\alpha + \beta) \cos \frac{1}{2}(\alpha - \beta) - \cos \alpha + \beta)$$

Sugerencia: puede usar los métodos `Math.cos(a)` y `Math.sin(a)`.

$$m. \text{ valor} = \text{Ln } 3 + \left(2 \frac{a+b}{2k}\right)$$

Sugerencia: puede usar el método `Math.log(a)`.

$$\text{n. } resultado = \frac{7 - \cos^2(h) + 1}{3 + \sqrt{15}}$$

Sugerencia: puede usar los métodos `Math.cos(a)`, `Math.pow(a,b)` y `Math.sqrt(a)`.

$$\text{o. } p = 2\pi \sqrt{\frac{1}{f}} \left(1 + \frac{\phi}{14+b}\right)$$

Sugerencia: puede usar la constante `Math.PI` y el método `Math.sqrt(a)`.

Indique cuál es el resultado de las siguientes expresiones en Java (tenga en cuenta las reglas de precedencia):

- a. `7 + 2 - 1 / 4`
- b. `5 + 21 * 4 * 4 - 1`
- c. `Math.sqrt(54 * 2 + 7) + 4 % 2`
- d. `2 == 5 + 4 - 5 * 48 / 4`
- e. `1 / 1 / 1 / 7`
- f. `((13 - 7) * (21 - 11)) / ((1 + 1) * (15 - 10)) + 1`
- g. `5 + 4 * 4 + 2 * 7 + 6 * 5`
- h. `7 / 5 * 5 / 9 * 5 / 2 * 4 / 7`
- i. `55 / 5 + 98 - 145 >= 58 + 15 - 4 * 12`
- j. `true && true || false`

---

## Actividad 8 //

# Introducción a Eclipse

## Objetivos

- Identificar el entorno de desarrollo Eclipse.
- Definir el concepto IDE.
- Identificar las partes principales de Eclipse.

## Introducción

Eclipse es un IDE (*Integrated Development Environment* o ambiente de desarrollo integrado). Para usarlo es necesario instalar el JDK (*Java Development Kit*). Recuerde instalar la versión que corresponda a su computador (32 o 64 bits).

Eclipse se encuentra en diferentes versiones. Si trabaja con la versión SE (*Standard Edition*) de Java se recomienda descargar la versión *Eclipse IDE for Java Developers*, óptima para esta edición. La descarga de la versión de Eclipse (32 o 64 bits) depende de la versión que haya instalado del JDK (32 o 64 bits).

No es necesario instalar el programa. La descarga consiste en un archivo comprimido (formato .ZIP), de manera que lo único que debe hacer es descargarlo y descomprimirlo. Se recomienda que lo descomprima en el directorio raíz. Por ejemplo, en la carpeta “C:\eclipse”.

## Prerrequisitos

- Instalar y configurar Java correctamente en un equipo (según la actividad número 2 de este libro).

## Ejercicios

1. Descargue Eclipse desde la página web <http://www.eclipse.org/downloads/>.
2. Seleccione un *workspace* o zona de trabajo (la carpeta en su computador donde debe guardar los proyectos para trabajar) adecuado.
3. Construya un proyecto de Java llamado Primera Prueba.
4. Construya un paquete llamado `com.universidad.informatica.test` en la carpeta `src`.
5. Dentro de este paquete, construya una clase llamada `PrimeraClaseDePrueba`.
6. En este momento deberá ver en el editor de Eclipse el código fuente de una clase vacía. Incluya o modifique la clase para insertar el siguiente código fuente:

```
package com.universidad.informatica.test;
import java.util.Scanner;
public class PrimeraClaseDePrueba {
    public static void main(String[] args) {
        Scanner teclado = new Scanner(System.in);
        System.out.println("Hola, cual es tu
        nombre???");
        String nombre;
        nombre = teclado.nextLine();
    }
}
```



```

        System.out.println("Hola " + nombre + ",
este es un ejemplo de Java");
    }
}

```

7. Guarde la clase. Si tiene errores (aparecen resaltados en rojo) revise qué puede estar fallando.
8. Ejecute el código haciendo clic en el botón de ejecución *run*.
9. Cree una clase llamada `Persona` (puede ser en el mismo paquete) que defina un objeto con las siguientes características:

Persona
<ul style="list-style-type: none"> <li>- nombre: String</li> <li>- apellido: String</li> <li>- edad: int</li> </ul>
<pre> &lt;&lt;constructor&gt;&gt; Persona () &lt;&lt;constructor&gt;&gt; Persona (nuevoNombre: String, nuevoApellido: String) + establecerNombre (nuevoNombre: String) + establecerApellido (nuevoApellido: String) + establecerEdad (nuevaEdad: int) + obtenerNombre (): String + obtenerApellido: String + obtenerEdad (): int + obtenerNombreCompleto (): String </pre>

10. Crea una clase llamada `PruebaPersona` que permita construir dos personas, y probar que sus métodos y constructores funcionen correctamente.



---

## Actividad 9 //

# La clase String

### Objetivos

- Identificar la clase `String` y su utilidad en el uso de cadenas de caracteres.
- Reforzar el concepto de la clase `String` mediante la API de Java.
- Resolver problemas en los que deba tener en cuenta la aplicabilidad de la clase `String`.

### Introducción

El uso de cadenas de texto es quizás una de las actividades más comunes en la programación en Java. Para emplearlas, existe una clase llamada `String`, la cual incluye métodos que permiten realizar tareas con las cadenas de texto, como por ejemplo contar el número de letras de la cadena, generar nuevas cadenas de texto (con letras en mayúscula o minúscula), insertar texto o recortarlo, entre otras funciones.

---

Para mayor información sobre la clase `String` puede visitar la API de Java.

## Prerrequisitos

- Instalar y configurar Java correctamente en un equipo (según la actividad número 2 de este libro).
- Revisar los métodos estudiados sobre la clase `String`, con base en la información aportada por otras fuentes de información (API de Java, sitios web, entre otros).

## Ejercicios

1. Descargue Eclipse desde la página web <http://www.eclipse.org/downloads/>
2. Seleccione un *workspace* o zona de trabajo (la carpeta de su computador donde deberá guardar sus proyectos) adecuado.
3. Construya un proyecto de Java llamado `Prueba String`.
4. Construya un paquete llamado `com.universidad.informatica.test` en la carpeta `src`. Dentro de este paquete, construya una clase llamada `PruebaDeCadenasDeTexto`, en la cual debe hacer lo siguiente:
  - a. Construya una cadena de texto con el contenido: `Hola, bienvenido al curso de Java`. Debe guardar este texto en una variable de referencia llamada `cadena`, de la siguiente manera: `String cadena = 'Hola, bienvenido al curso de Java';`
  - b. Con esta cadena, construya una instrucción en Java para responder las siguientes preguntas:
    - ¿Cuál es la letra que se encuentra en el índice 8? Use el método `charAt`.
    - ¿Es igual a la cadena “Hola, bienvenido al curso de Java”? Use el método `equals` (debe construir otra cadena de texto con el texto que va a comparar).

- ¿Cuál es el índice donde se encuentra la cadena `curso`? Use el método `indexOf` (debe construir otra cadena de texto con el texto que desea buscar).
  - ¿Cuál es el tamaño de la cadena de texto? Use el método `length`.
- c. Con la misma cadena, construya una instrucción en Java para realizar las siguientes actividades:
- Extraer a otra cadena el texto que empiece con la letra “c” de la palabra “curso”. Use el método `substring`.
  - Extraer a otra cadena el texto que empiece con la palabra “bienvenido” y termine con la palabra “curso”. Use el método `substring`.
  - Generar una nueva cadena con el mismo texto de cadena, pero en minúsculas. Use el método `toLowerCase`.
  - Generar una nueva cadena con el mismo texto de cadena, pero en mayúsculas. Use el método `toUpperCase`.
  - Llamar al usuario por su nombre y construir una nueva cadena por medio del uso de la cadena existente, pero incluyendo el nombre del usuario en el saludo. Por ejemplo, “Hola Mario, bienvenido al curso de Java”. Use el método `concat` y los métodos anteriores.
- d. Construya dos cadenas de caracteres “hola” y “Hola” y compárelas por medio del método `equals`. El resultado indica que las dos palabras no son iguales, ¿por qué?
- e. Usando el método `compareTo`, realizar las comparaciones entre las cadenas:
- “Hola” y “Hola”. Revise el resultado (es un número entero).
  - “Hola” y “HOLA”. Revise el resultado (es un número entero).
  - “Hola” y “hola”. Revise el resultado (es un número entero).
  - “ol” y “Hola”. Revise el resultado (es un número entero).
  - “HOLA” y “HOLA, ¿cómo vas?”. Revise el resultado (es un número entero).

- 
- f. ¿Qué diferencia hay en hacer una comparación de cadenas usando los métodos `equals` y `compareTo`?
  - g. El método `compareTo` hace una comparación lexicográfica, ¿de qué se trata esta comparación?

---

## Actividad 10 //

# Uso del condicional simple

## Objetivos

- Aprender el uso de la estructura del condicional simple en Java.
- Aprender el uso de la instrucción *if*.
- Resolver problemas en los que deba aplicar la instrucción *if*.

## Introducción

La programación estructurada es importante para entender cómo resolver problemas usando la programación orientada a objetos. Según el paradigma de la programación estructurada, un programa se puede escribir solamente mediante el empleo de las tres estructuras básicas de control: secuencias, condicionales y repeticiones. Las conjunciones condicionales denotan condición o necesidad de que se cumpla alguna circunstancia. Para formar una conjunción condicional, se usa la conjunción “si”. Por ejemplo: “Si va a llover, debo llevar paraguas”; “si quiero

pasar el semestre, debo estudiar”. Se observa que en cada oración se cumple la oración subordinada si y solo si la primera es verdadera.

En Java, de manera similar, existe una instrucción para estas expresiones condicionales llamada *if*. La estructura básica para una sola instrucción es:

```
if(condición)
    instrucción;
```

En caso de que sea necesario realizar múltiples instrucciones, se debe armar un cuerpo de instrucciones usando llaves:

```
if(condición)
{ //inicio del cuerpo de instrucciones
instrucción1;
instrucción2;
instrucción3;
instrucción4;
instrucción5;
instrucción6;
} //fin del cuerpo de instrucciones
```

La condición solamente puede ser *true* o *false*. Dicha condición se puede construir usando expresiones que incluyan operadores lógicos. Por ejemplo:

```
a <= b
(7 > variable1) && (9 < variable2)
edad > 18
```

## Prerrequisitos

- Instalar y configurar Java correctamente en un equipo (según la actividad número 2 de este libro).



- Revisar los conceptos estudiados sobre la programación orientada a objetos, específicamente: clases, objetos, métodos y variables.
- Repasar el estudio de las expresiones aritméticas.

## Ejercicios

1. Construya una aplicación que, al ser abierta, le pregunte la edad al usuario. Si el número es menor a 18, debe mostrar el mensaje: “Lo siento, debe ser mayor de edad para poder manejar”; si es mayor o igual a 18, debe mostrar el mensaje: “¡Felicitaciones, cumple con la edad para manejar!”.
2. Construya una aplicación que, al ser abierta, le pregunte al usuario por sus ventas semanales y que, según la información proporcionada, haga lo siguiente:
  - Si son menores a 10.000, debe mostrar el mensaje: “Debe esforzarse más, no se cumple la meta mínima”.
  - Si son iguales o mayores a 10.000 y menores a 30.000, debe mostrar el mensaje: “Muy buen trabajo, debe seguir así. Si se esfuerza un poco más, podría recibir el premio sorpresa”.
  - Si son mayores o iguales a 30.000, debe mostrar el mensaje: “¡Felicitaciones, logró la meta para la sorpresa! Por favor pasar por Recursos Humanos y reclamar su viaje”.
3. Construya una aplicación que, al ser abierta, le solicite al usuario un número entero, y determine si dicho número es par o impar.
4. Construya una aplicación que, al ser abierta, le solicite al usuario ingresar tres números enteros, y que muestre el número mayor de los tres.
5. Construya una aplicación que reciba 10 números enteros y que determine y muestre la cantidad de números negativos y positivos recibidos. Asuma que el valor cero es positivo.

---

6. Construya una aplicación que le pregunte al usuario por sus ingresos anuales y su estado civil (soltero o casado). Debe calcular el valor del impuesto que debe pagar según las siguientes condiciones:

Si es casado:

- Si sus ingresos son menores a \$20.000, el valor que debe pagar corresponde al 5 % de sus ingresos anuales.
- Si sus ingresos son mayores o iguales a \$20.000, el valor que debe pagar corresponde al 3 % de sus ingresos anuales.

Si es soltero

- Si sus ingresos son menores a \$20.000, el valor que debe pagar corresponde al 8 % de sus ingresos anuales.
- Si sus ingresos son mayores o iguales a \$20.000, el valor que debe pagar corresponde al 6% de sus ingresos anuales.

---

## Actividad 11 //

# Uso del condicional múltiple

## Objetivos

- Aprender el uso de la estructura condicional múltiple en Java.
- Adquirir habilidades en el uso de la instrucción *switch*.
- Resolver problemas en los que deba aplicar condicionales múltiples.

## Introducción

En la actividad anterior se evidencia que las conjunciones condicionales denotan condición o necesidad de que se cumpla alguna circunstancia. Una estructura condicional múltiple permite evaluar los diferentes valores que puede tomar una variable. Según su valor, se puede determinar una acción o tarea por realizar.

En Java existe una instrucción para estas expresiones condicionales múltiples, llamada *switch*. Su estructura básica es:

```
switch(variableAEvaluar) {
    case primerValor:
        instrucciones;
        break;
    case segundoValor:
        instrucciones;
        break;
    case tercerValor:
        instrucciones;
        break;
    default:
        instrucciones;
        break;
}
```

La instrucción `switch` permite evaluar variables de tipo `byt`”, `short`, `int`, `char` y `String` (en las primeras versiones no estaba permitido evaluar expresiones de tipo `String`). Como se puede observar, en la estructura básica se evalúan los posibles valores de la variable `variableAEvaluar` y, según el valor que adquiera, se ejecutan las acciones asociadas a cada caso (o *case* en la estructura). Si el valor evaluado no es ninguno de los propuestos en cada *case*, se puede incluir la opción *default* que hace referencia a las tareas o instrucciones que se deben realizar, si el valor de la variable evaluada no es ninguno de los *case* anteriores.

Es necesario aclarar que la instrucción *switch* no se debe (aunque teóricamente sí se puede) usar para evaluar rangos de valores. Los rangos de valores se pueden trabajar mejor con la estructura *if* o con *if else*. Para aclarar esto, suponga que necesita un sistema que le muestre un mensaje para las personas menores de 18 años y otro para las personas mayores de 18. Básicamente, se están evaluando dos rangos. El primer rango de edades va desde los 0 hasta los 18 años, y el segundo va desde los 18 años en adelante. Como se ilustró en la actividad anterior, esto se resuelve fácilmente mediante el uso de la estructura *if*. Si se quisiera resolver este problema con una estructura *switch*, se tendrían que ha-

cer 19 casos, desde los 0 años hasta los 18, y un caso adicional *default* que aplicaría para las demás edades. En este caso, y en general para los rangos de valores, resulta poco práctico usar *switch* para resolver el problema.

## Prerrequisitos

- Instalar y configurar Java correctamente en un equipo (según la actividad número 2 de este libro).
- Realizar la actividad práctica número 10 de este libro.

## Ejercicios

1. Construya una aplicación que le pida al usuario dos números enteros y le pregunte qué desea hacer con ellos; las dos opciones son sumar o multiplicar. El programa deberá entregar la respuesta correcta según la opción seleccionada por el usuario. Si el usuario indica otra opción, deberá mostrar el mensaje: “opción inválida”.
2. Construya una aplicación en Java que detecte la tecla oprimida por el usuario. Si el usuario presiona un número del 0 al 9, el programa debe mostrar como resultado el valor de la tecla oprimida. En caso de presionar cualquier otra tecla, debe mostrarse el mensaje: “tecla no permitida”.

Debe realizar el siguiente ejercicio dos veces. La primera con la instrucción *switch* y la segunda con las instrucciones *if...else* anidadas:

3. Construya una aplicación en Java que le muestre al usuario el nombre del mes, dependiendo del número del mes que ingrese. Ejemplo: si el usuario ingresa 2, deberá mostrar “Febrero”; si el usuario ingresa 11, deberá mostrar “Noviembre”. Si el número ingresado no corresponde a ningún mes, deberá mostrar el mensaje “número de mes inválido”.
4. Construya una aplicación en Java que detecte la tecla oprimida. Si el usuario presiona una vocal (cualquiera), el programa

---

debe mostrar un mensaje que indique: “La letra es una vocal”. Si el usuario presiona un número del 0 al 9, el programa debe mostrar la tecla que presionó. En caso de presionar cualquier otra tecla, debe mostrar el mensaje: “La tecla es una consonante o un símbolo”.

---

Actividad 12 //

# Uso de Ciclos

## Objetivos

- Identificar los conceptos empleados en el uso de ciclos en Java.
- Reconocer las estrategias de contador y de centinela.
- Adquirir habilidades y destrezas en el uso de las instrucciones *while* y *for*.

## Introducción

En relación con el uso de las estructuras de control, aparece una nueva estructura que permite hacer repeticiones de instrucciones: la estructura de ciclos. Los ciclos suelen tener la siguiente estructura:

```
Ciclo(condición) {  
    instrucciones;  
}
```

---

Los ciclos son, básicamente, repeticiones de una o varias instrucciones. Para construir ciclos, se deben tener en cuenta dos cosas:

- La condición para que el ciclo haga las repeticiones —mientras que esta condición se mantenga como verdadera, las instrucciones se repetirán—.
- El cambio de la condición para que las repeticiones dejen de ejecutarse —por lo general, el ciclo debe terminar en algún momento, de modo que la condición debe cambiar en algún momento de verdadera a falsa; para esto, es necesario usar las instrucciones dentro del ciclo, a fin de garantizar su finalización—.

En Java existen dos instrucciones que se pueden emplear para el uso de ciclos: *while* y *for*. La estructura de la instrucción *while* es:

```
while (condición) {  
    instrucciones;  
}
```

La estructura de la instrucción *for* es:

```
for (inicialización; condición; incremento) {  
    instrucciones;  
}
```

El secreto para el uso de los ciclos radica en cómo se controla el número de repeticiones que se deben efectuar en un ciclo. Para esto se emplean dos estrategias: la del contador y la del centinela. La estrategia del contador consiste en indicar cuántas repeticiones (o iteraciones) se requieren por ciclo. De manera explícita, se determina cuántas repeticiones se van a realizar. A modo de ejemplo, el siguiente ciclo:

```
for (int i = 0; i < 10 ; i++) {  
    instrucciones;  
}
```



Este ciclo realiza 10 iteraciones. Se puede observar que, antes de iniciar la primera iteración, es posible determinar el número de iteraciones o repeticiones que se van a realizar.

La estrategia del `centinela` consiste en determinar un valor `centinela` (valor especial) y cuando dicho valor aparezca el ciclo deberá terminar. Como no se puede determinar cuándo va a aparecer el valor `centinela`, se dice que no es posible determinar de manera explícita cuántas repeticiones se van a realizar. A modo de ejemplo, el siguiente ciclo:

```
Scanner teclado = new Scanner (System.in);
valor = nextInt();

while(valor != -1){
    instrucciones;
    valor = nextInt();
}
```

Este ciclo realiza varias iteraciones (repeticiones), pero no se puede determinar cuántas. Dependerá del usuario y de cuándo decida ingresar un valor -1 (que en el ejemplo es el valor `centinela`).

## Prerrequisitos

- Instalar y configurar Java correctamente en un equipo (según la actividad número 2 de este libro).
- Realizar la actividad práctica número 11 de este libro.

## Ejercicios

1. Construya una aplicación que realice la sumatoria de 10 números enteros positivos. Posteriormente determine el valor promedio de los números.

- 
2. Construya una aplicación que realice la sumatoria de números enteros positivos. Aún no se sabe cuántos números se van a ingresar, pero se sabe que si el usuario ingresa el valor -1, finaliza la sumatoria. Determine el valor promedio de los números.
  3. Construya una aplicación que reciba 3 números enteros y determine cuál es el mayor y el menor de los tres.
  4. Construya una aplicación que realice el factorial de un entero no negativo. El factorial de “n” se define como tal “para valores mayores o iguales a 1”.
  5. Construya una aplicación que permita encontrar el número entero mayor sin saber cuántos números se van a comparar.

Actividad 13 //

# Uso de arreglos

## Objetivos

- Familiarizarse con el uso de arreglos.
- Conocer los arreglos como estructuras de almacenamiento.
- Aprender a construir y utilizar arreglos en Java.
- Utilizar el ciclo *for* para llevar a cabo el recorrido de un *array*.
- Utilizar *arrays* unidimensionales y multidimensionales.

## Introducción

Un arreglo se puede entender como un elemento que permite mantener un número fijo de valores de un mismo tipo de dato. Al construir un arreglo, se debe tener en cuenta que:

- Un arreglo puede ser unidimensional o multidimensional.
- Su tamaño es fijo (una vez construido, su tamaño no se puede modificar).

- Un arreglo está compuesto por elementos. Por ejemplo, un arreglo de tamaño 8 es un arreglo con 8 elementos (en este caso, se trata de un arreglo unidimensional). Un arreglo de dimensión 2 x 3 indica que se trata de una matriz de dos filas y tres columnas (en este caso, se trata de un arreglo de dos dimensiones).
- Cada elemento tiene un identificador llamado índice que inicia desde el número 0. Así, el primer elemento tendría el índice 0; el segundo, el índice 1; el tercero, el índice 2, y así sucesivamente. Esto en el caso de un arreglo unidimensional. En los arreglos de dos dimensiones, se tiene un índice para el manejo de las filas y otro para el manejo de las columnas.

Un arreglo se define de manera similar a los objetos. Primero, se necesita una variable de referencia, y luego se construye el objeto con el tamaño que se necesite:

```
//construir el arreglo
int[] primerArreglo;
//asignar memoria
primerArreglo = new int[5];
```

Cada elemento del arreglo se puede utilizar mediante el empleo del índice. Por ejemplo, si se desea dar valor a los elementos 1, 3 y 5 del arreglo se puede hacer algo así:

```
unArreglo[0] = 10;
//recuerde que el primer elemento tiene
//el índice 0
unArreglo[2] = 20;
unArreglo[4] = 30;
```

En el caso de arreglos de dos dimensiones, estos se declaran por medio del uso de un número de corchetes igual a la dimensión del *array*. Ejemplo:

```
// Arrays de doubles
double dosArrays[ ][ ] = new double[512]
[128];
//declaración e inicialización de una matriz
double [ ][ ] mat = {{1,2,3}, {4,5,6}};
```

Para acceder a un elemento dentro de un arreglo de dos dimensiones, el proceso es similar al realizado para un arreglo unidimensional.

```
int valor = matriz[3][1];
```

Se pueden construir arreglos de tipo `byte`, `short`, `long`, `float`, `double`, `boolean`, `char` y `String`.

## Prerrequisitos

- Instalar y configurar Java correctamente en un equipo (según la actividad número 2 de este libro).
- Realizar la actividad número 12 de este libro.

## Ejercicios

1. Construya una aplicación que inicialmente defina un arreglo de 3 números enteros, que después le pida al usuario los tres valores para ingresar al arreglo, y que finalmente muestre en pantalla los valores que tiene el arreglo. Procure hacer esto con ciclos.
2. Construya una aplicación que defina un arreglo de tamaño 10 y llene los elementos con valores de tipo `long`. Después, construya otro arreglo del mismo tamaño y tipo, y copie todos los elementos del primer arreglo en el segundo. Muestre en pantalla los elementos del segundo arreglo.
3. Construya un arreglo de tipo `char` cuyo tamaño será un dato ingresado por el usuario (10 y 20 elementos). Llene los ele-

mentos del arreglo con valores que el usuario le indique (use ciclos). Muestre los valores en pantalla en orden inverso, desde el último elemento hasta el primero.

4. Construya una aplicación que defina un arreglo de 10 elementos enteros (el usuario o el mismo programador puede digitar los valores). Ahora es necesario que elimine los elementos que se encuentren en la ubicación 2, 4 y 8 para que finalmente quede con un arreglo de 7 elementos. Se entiende que un arreglo no puede cambiar de tamaño, ¿qué propone para solucionar este problema?
5. Observe el siguiente fragmento de código e indique qué ocurriría al intentar ejecutarlo:

```
public class HayUnProblema
{
    public static void main(String[] args){
        double data[] = new double[10];
        data[10] = 100;
    }
}
```

6. Observe el siguiente fragmento de código e indique qué ocurriría al intentar ejecutarlo:

```
double data[ ];
data[0] = 29.95;
```

7. Construya un programa que defina un arreglo bidimensional de tipo entero que posea 4 filas y 4 columnas. El usuario deberá ingresar cada uno de los elementos del arreglo. El programa deberá mostrar en pantalla el valor mayor, el valor menor y el valor promedio encontrados en el arreglo.
8. Mediante el uso de la matriz creada en el ejercicio anterior, construya un programa que muestre los elementos ubicados en la diagonal secundaria.

- 
9. Por medio del uso de la matriz creada anteriormente, construya un programa que muestre los elementos ubicados en el cuarto cuadrante (tenga en cuenta el plano cartesiano).
  10. Construya un programa que defina una matriz de cinco filas y cinco columnas. Cada uno de los elementos que conforme la matriz deberá ser ingresado por el usuario y será un valor numérico entero. Al finalizar, el programa deberá mostrar la sumatoria de cada una de las filas y también la sumatoria de cada una de las columnas.





---

Actividad 14 //

# Arreglo de objetos

## Objetivos

- Aplicar el uso de arreglos basados en objetos.
- Crear arreglos de objetos.

## Introducción

En la práctica anterior, en la que se estudiaron los arreglos unidimensionales y bidimensionales, se pudo evidenciar la necesidad de definir el tipo de dato que debe contener un array. Así como se pueden construir arreglos de tipo `byte`, `short`, `long`, `float`, `double`, `boolean`, `char` y `String`, también se puede definir un *array* de objetos. El tratamiento de las posiciones, bien sea para las inserciones, búsquedas, entre otras, es el mismo.

Suponga que se tiene una clase llamada `Persona` con los atributos: identificación, nombre, apellido, edad, sexo. Se podría, entonces, definir la siguiente clase:

```

public class PersonaApp {

    public static void main(String[] args) {

        //Creamos un array de objetos de la cla-
se Persona
        Persona arrayObjetos[]=new Persona[3];

        //Creamos objetos en cada posicion
        arrayObjetos[0]=new Persona(11,"Fer","U-
reña",23, "M");
        arrayObjetos[1]=new Persona(12,"Epi",
"Dermis", 30, "M");
        arrayObjetos[2]=new Persona(13,"Blas",
"Femia", 25, "F");

        //Recorremos el array para encontrar la
suma de las edades
        int sumaEdad=0;
        for (int i=0;i<arrayObjetos.length;i++){
            sumaEdad+=arrayObjetos[i].getEdad();
        }
        System.out.println("La suma de edades es "+su-
maEdad);
    }

}

```

En el ejemplo anterior sólo se tienen tres objetos de la clase `Persona`, pero se podrían definir muchos más. Si ese fuera el caso, se utilizaría el ciclo `for` para crear una serie de objetos y hacer su recorrido. Sería ideal que el usuario ingresara los datos para cada uno de los objetos.

---

## Prerrequisitos

- Instalar y configurar Java correctamente en un equipo (según la actividad número 2 de este libro).
- Realizar la actividad número 13 de este libro.

## Ejercicios

1. Construya una aplicación que inicialmente defina un arreglo de tres objetos (se deja a consideración del estudiante la definición del tipo de objeto); que a continuación le pida al usuario los datos correspondientes para cada uno de los atributos de los tres objetos para ingresar al arreglo, y que finalmente muestre en pantalla los valores del arreglo. Procure hacer esto con ciclos.
2. Defina la clase `Empleado` con los atributos: identificación, nombres, apellidos, sexo, salario. Ahora construya una nueva clase que haga uso de la clase `Empleado` y muestre el promedio de salarios<sup>1</sup>.
3. Con base en la clase `Empleado` utilizada en el ejercicio anterior, construya una nueva clase que haga uso de la clase `Empleado` y muestre sólo los empleados de sexo M (M: masculino, F: femenino). Los datos para cada uno de los objetos deben ser ingresados por el usuario. Utilice una variable `centinella` para el control de ingreso de los datos.
4. Defina la clase `Triangulo` con los siguientes atributos: lado A, lado B, lado C. Ahora construya una nueva clase que haga uso de la clase `Triangulo` y permita el ingreso de 10 objetos

---

<sup>1</sup> Se deben construir 10 objetos de la clase “Empleado” y los datos deben ser ingresados por el usuario.

---

Triangulo. Los datos deben ser ingresados por el usuario. Al final, se debe mostrar sólo la información de los objetos Triangulo que son de tipo equilátero<sup>2</sup>.

---

<sup>2</sup> En la misma clase “Triángulo” debe definir un método llamado “tipo-Triangulo” que reciba como parámetro los tres valores (correspondientes a cada uno de los lados) ingresados por el usuario y retorne *true* en caso de que se trate de un triángulo equilátero.

# Herencia

## Objetivos

- Identificar los conceptos relacionados con la herencia.
- Comprender la forma de derivar una subclase.
- Aplicar los conceptos teóricos a la solución de algoritmos.

## Introducción

La herencia es una de las características importantes de la programación orientada a objetos. Es un tipo de relación entre una clase más general (superclase) y otra más especializada (subclase). La subclase hereda todos los atributos y métodos de la superclase. El método se escribe una vez y puede ser utilizado por todas las subclases existentes. En el caso de los atributos, estos son compartidos tanto por la superclase como por todas las subclases.

En Java, todas las clases heredan de una clase base llamada *Object*. Esta es la única que no posee una clase padre. Para indicar que una

clase hereda de otra es necesario utilizar la palabra reservada *extends*. Para mayor claridad veamos el siguiente ejemplo:

Suponga que existe la clase *Persona* con los atributos: nombre, profesión y ocupación:

```
public class Persona {
    protected String nombre;
    protected String profesion;
    protected String ocupacion;

    public Persona() {
        System.out.println("Constructor:
Dentro de la clase Persona");
        nombre = ""; profesion = ""; ocupa-
ción="";
    }
    // Otras sentencias
}
```

Ahora debe crear la clase *Empleado*. Un *Empleado* también puede ser una *Persona*; entonces se puede aplicar el concepto de herencia “extendiendo” la clase *Persona*.

```
public class Empleado extends Persona {
    public Empleado() {
        System.out.println("Constructor:
Dentro de la clase Empleado");
    }
    // Otras sentencias
}
```

## Prerrequisitos

- Instalar y configurar Java correctamente en un equipo (según la actividad práctica número 2 de este libro).

- Realizar lecturas complementarias sobre el concepto de herencia.

## Ejercicios

1. Investigue sobre los diferentes métodos que ofrece la clase *Object*.
2. Construya un ejemplo con el lenguaje de programación Java en el que se muestren evidencias de, por lo menos, tres de los métodos de la clase *Object*.
3. ¿Cuántas subclasses se pueden construir a partir de una superclase ya implementada?
4. Identifique la superclase y la subclase en cada una de las siguientes parejas de clases:
  - Empleado, Gerente.
  - Polígono, Triángulo.
  - Estudiante Graduado, Estudiante.
  - Persona, Estudiante.
  - Vehículo, Carro.
5. Explique el significado de la palabra reservada *Super*.
6. Implemente una superclase *Factura*. A continuación, construya las clases *FacturaCredito* y *FacturaContado* heredadas de la clase *Factura*. Una *Factura* tiene un número consecutivo y una fecha de creación. Una *FacturaCredito* tiene un plazo máximo para realizar su pago, y una *FacturaContado* tiene dos medios de pago: efectivo o tarjeta débito. Construya las clases correspondientes, sus constructores y los métodos `toString` para cada una de las clases. Finalmente construya una nueva clase que permita hacer uso de las clases creadas inicialmente.





---

Actividad 16 //

# Polimorfismo

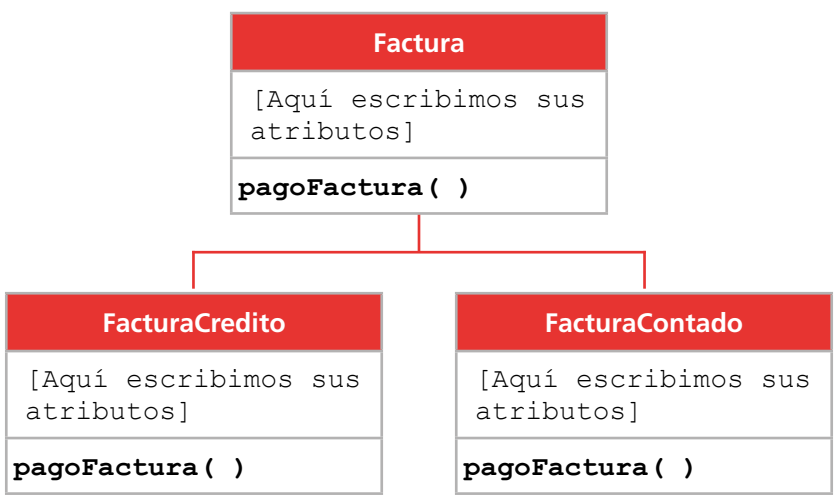
## Objetivos

- Comprender la importancia del concepto de polimorfismo en la programación orientada a objetos.
- Analizar y resolver problemas en los que sea aplicable el concepto de polimorfismo.

## Introducción

Uno de los pilares fundamentales de la programación orientada a objetos es el polimorfismo. Conceptualmente, se trata de la propiedad que tienen los objetos de adquirir varias formas. En pocas palabras, el polimorfismo permite enviar el mismo mensaje a distintos objetos de naturaleza heterogénea. Esta propiedad permite programar de una manera más general, en lugar de hacerlo de una forma específica, y simplifica la programación.

Por ejemplo, dada la superclase `Factura` se podría definir un método llamado `pagoFactura` para las clases derivadas `FacturaCredito` y `FacturaContado`. El método `pagoFactura` de las subclases se implementa de manera diferente.



En el gráfico anterior se observa que el método `pagoFactura ( )` tiene una funcionalidad distinta según el objeto que reciba el mensaje. Se podría tener el siguiente fragmento de código:

```
FacturaCredito fCre = new FacturaCredito(...);
FacturaContado fCon = new FacturaContado(...);
.....
fCre.pagoFactura ( );
fCon.pagoFactura ( );
```

La clase padre comparte el método `pagoFactura ( )` con las subclases y estas, a la vez, tienen un método propio que se puede implementar de forma diferente. Ahora, si se tuviese una lista del objeto `Factura` que contuviese objetos de tipo `FacturaCredito` o `FacturaContado`, se podría pensar que son objetos de distinto tipo, lo cual es correcto; pero recordemos que estas dos clases heredan de la

clase `Factura`, por lo tanto son del mismo tipo aunque implementen de forma diferente el método `pagoFactura()`.

Observe el siguiente fragmento de código:

```
List<Factura> factura = new ArrayList<Factura>( );
FacturaCredito fCre = new FacturaCredito(...);
FacturaContado fCon = new FacturaContado(...);

factura.add(fCre);
factura.add(fCon);

for(int x=0; x < factura.size(); x++)
    factura.get(x).pagoFactura();
```

En el recorrido de la lista se puede apreciar que se obtiene la posición del objeto sin importar su tipo. Gracias al polimorfismo se invoca el método `pagoFactura()` sin necesidad de conocer la clase desde la cual se hace el llamado. El compilador es el que se encarga de evaluar el tipo de objeto que se recupera en cada posición de la lista y de invocar el método correspondiente. El uso del polimorfismo ayuda en gran medida como punto de partida para el estudio de la genericidad en Java, que será una de las actividades prácticas de este libro. En Java, existen tres formas de implementar el polimorfismo.

## Prerrequisitos

- Comprender el uso y la importancia de la herencia en la programación orientada a objetos.
- Desarrollar la actividad práctica número 15 de este libro.
- Comprender el concepto de sobrecarga de métodos.
- Investigar cuáles son las tres formas de implementar el polimorfismo en el lenguaje de programación Java.

---

## Ejercicios

1. En la sección introductoria de este capítulo se ha explicado un ejemplo de manera breve. Este ejemplo no está implementado al ciento por ciento. El estudiante debe realizar la implementación completa como ejercicio y tener en cuenta los siguientes aspectos:
  - La factura tiene un atributo denominado `valorFactura`.
  - Para realizar el pago de una factura a crédito se debe incrementar el 5% al valor de la factura, si este se efectúa entre los 0 y 30 días. Se debe incrementar el 10%, si el pago se realiza entre los 31 y 60 días, y el 20% si se realiza entre los 61 y 90 días. La empresa no otorga más días de plazo.
  - En el caso de las facturas de contado, la empresa otorga un descuento del 10 % sobre el valor de las mismas cuando su pago se realiza los 5 primeros días posteriores a la fecha de su emisión, y un 5% cuando se efectúa entre el sexto y décimo día.
2. Una empresa del sector tecnológico desea tener un sistema de información para administrar toda la información de su personal. Se conoce la siguiente información de cada empleado: número de identificación, nombres, apellidos, fecha de ingreso, sueldo básico mensual, porcentaje promedio de cumplimiento en los proyectos que participa y número de proyectos en los que participa. Desde el punto de vista de la contratación, se tienen empleados con contrato a término fijo, con contrato indefinido y por prestación de servicios. Para determinar el sueldo mensual de cada empleado, se tiene en cuenta:
  - Si el contrato es a término indefinido y el porcentaje de cumplimiento en proyectos es superior al 90 %, se incrementará el 10 % al sueldo básico mensual.
  - Si el contrato es a término fijo y el porcentaje de cumplimiento en proyectos es superior al 90 %, se incrementará el 8 % al sueldo básico mensual.

- Si el contrato es por prestación de servicios y el porcentaje de cumplimiento en proyectos es superior al 90%, se incrementará el 2 % por cada proyecto en el que participe.
3. Una empresa se dedica al alquiler de vehículos de distintos tipos. Para cada alquiler se guarda un registro con los datos básicos del solicitante, la fecha inicial y final del alquiler, el cilindraje del motor del vehículo en centímetros cúbicos, el modelo del vehículo, su placa y su marca. Para determinar el valor del alquiler, el procedimiento es el siguiente: el número de días de alquiler (incluyendo la fecha inicial y final) se multiplica por un valor (que se obtiene al multiplicar 10 por el cilindraje del vehículo. El cilindraje puede ser 1,4 cc, 1,6 cc, 2,0 cc, 1,8 cc) y por un factor fijo (2). Sin embargo, se tiene una tabla de descuentos dependiendo del modelo del vehículo:
- Si el modelo del vehículo es 2010 en adelante y el cilindraje es 1,8 o 2,0, se realiza un descuento del 10 % sobre el valor del alquiler.
  - Si el modelo del vehículo está entre los años 2005 y 2009, y el cilindraje es de 1,4 o 1,6, se realiza un descuento del 15 % sobre el valor del alquiler siempre y cuando el número de días de alquiler supere los 30 días.
  - Si el modelo del vehículo está entre los años 2005 y 2009, pero su cilindraje es de 1,8 o 2,0, el descuento es del 20 % sobre el alquiler del vehículo, siempre y cuando el número de días de alquiler supere los 15.
  - Para cualquier otro tipo de casos no existe descuento.
  - Mediante el uso adecuado de la herencia, construya las clases necesarias y la clase principal para calcular el valor del alquiler de cualquier tipo de vehículo.
4. ¿En qué consiste “el principio de sustitución de Liskov” (Bárbara H. Liskov & Stephen N. Zilles en *Programming with Abstract Data Types*) y cuál es su concepto con respecto al mismo?



Actividad 17 //

# Clases abstractas

## Objetivos

- Aprender a crear y a utilizar clases y métodos abstractos.
- Aplicar los conceptos estudiados sobre clases y métodos abstractos a la solución de problemas.

## Introducción

Para definir un método abstracto, basta con escribir la declaración del método sin el cuerpo (sin su implementación) y utilizar la palabra reservada `abstract`. Por ejemplo:

```
public abstract void calcularArea( );
```

Los métodos abstractos se declaran cuando se espera que dos o más subclases definan una funcionalidad (o varias) similar en diferentes

---

modos, por medio de diferentes implementaciones (aplicación del polimorfismo). Una clase abstracta es una clase cualquiera que contiene por lo menos un método abstracto. Cuando se define una clase abstracta, esta no se puede instanciar. De ser así, se generaría un error en el tiempo de compilación. La implementación de los métodos abstractos definidos en la clase abstracta se lleva a cabo en otra clase. Las clases que harán uso de la clase abstracta deberán utilizar la palabra reservada *extends* (aplicación de herencia).

A continuación, un ejemplo sencillo de una clase abstracta:

```
public abstract class EjemploSencillo {
    public void unoMetodo() {
        System.out.println("Ejemplo Uno...");
    }
    public void dosMetodo() {
        System.out.println("Ejemplo dos...");
    }
    // Queremos que este metodo sea imple-
mentado
    // por una clase concreta.
    public abstract void metodoAbstracto();
}
```

Cuando una clase extiende la clase abstracta `EjemploSencillo` debe implementar el método abstracto `metodoAbstracto`. Un ejemplo podría ser el siguiente:

```
public class UsoEjSencillo extends Ejemplo-
Sencillo {
    public void metodoAbstracto() {
        System.out.println("Implementando el
método abstracto");
    }
}
```



Se podrían definir otras subclases que se extiendan de la clase `EjemploSencillo` y proporcionar diversas implementaciones para los métodos abstractos. En caso de que una subclase de una clase abstracta no defina e implemente sus métodos abstractos, también tendrá que declararse como una clase abstracta. Observe este ejemplo en el cual se define una clase abstracta y un método abstracto:

```
public abstract class ActividadDiaria{

    private String unAtributo;

    // constructores, getter y setter
    public void competir() {
        System.out.println("Competencia
(Clase Padre)");
    }

    // METODO ABSTRACTO => no se implementa
en la //clase abstracta pero si en la clases hijas
    public abstract void entrenar();
}
```

La clase `ActividadDiaria`, al tratarse de una clase “abstracta”, podría adoptar diferentes formas a partir de la definición de otras clases. Se puede definir una clase `Futbolista` que haga uso de la clase abstracta, y la definición del método abstracto podría ser totalmente diferente con respecto a la definición de otra clase.

```
public class Futbolista extends ActividadDia-
ria {

    private int numCamiseta;

    // constructor, getter y setter
```

```

        @Override
        public void entrenar() {
            System.out.println("Realiza un en-
entrenamiento (Clase Futbolista)");
        }
        @Override
        public void competir() {
            System.out.println("Juega un Partido
(Clase Futbolista)");
        }
    }
}

```

Otro ejemplo de clase abstracta:

```

public class Boxeador extends ActividadDiaria {

    private String colorPantaloneta;

    // constructor, getter y setter
    @Override
    public void entrenar() {
        System.out.println("Realiza un en-
entrenamiento (Clase Boxeador)");
    }
    @Override
    public void competir() {
        System.out.println("Participa en una
competencia (Clase Boxeador)");
    }
}

```

Importante: en las clases ejemplificadas anteriormente se observa que, en una línea antes de los métodos `competir` y `entrenar`, aparece la etiqueta `@override`. Esta anotación se utiliza para indicar el

código en el que se está “sobre-escribiendo o especializando” un método que se encuentra en la clase “padre” y que se quiere redefinir en la clase “hija”. La etiqueta `@override` sólo se encuentra en los métodos de las clases “hijas” que están definidos en la clase “padre”. Esto significa que cuando se haga el llamado a esos métodos, las clases “hijas” ejecutarán el método redefinido en la clase “hija”, y las que no lo hayan redefinido ejecutarán el método de la clase “padre”.

## Prerrequisitos

- Tener claridad sobre los conceptos de herencia y polimorfismo.
- Desarrollar las actividades número 15 y 16 de este libro.

## Ejercicios

1. Defina una clase abstracta denominada `Figura`. En ella, defina también los métodos abstractos `perimetro` y `area`. Defina ahora las clases `Cuadrado` y `Circulo` que se extiendan de la clase `Figura`. Implemente los métodos abstractos de la clase `Figura` en las subclases `Cuadrado` y `Circulo`. Por último, defina una clase principal que permita al usuario ingresar datos y que permita realizar un test sobre las clases definidas previamente.  
Recuerde que no se pueden crear instancias de una clase que ha sido declarada previamente como abstract. Suponga que no existe tal restricción: ¿qué ocurriría si se hace el llamado a un método abstracto de un objeto?
2. Defina la clase abstracta `Empleado`. A partir de ella defina las subclases `EmpleadoFijo` y `EmpleadoTemporal`. Defina el método abstracto `calcularSalario` en la clase `Empleado`. Para el caso de un empleado fijo, el salario se calcula de la siguiente manera: salario básico mensual más una bonificación adicional. En el caso del empleado temporal, el salario está definido como: salario básico mensual más el va-

lor obtenido por horas extras trabajadas (valor hora trabajada es un valor que debe ser ingresado por el usuario).

A partir de este enunciado el estudiante debe implementar la totalidad del ejercicio y crear una clase principal para llevar a cabo el test completo del programa que se implementará.

3. Use las clases del ejercicio anterior para crear un método abstracto denominado `visualizar` en la clase `Empleado`. Dependiendo del tipo de empleado, el método `visualizar` tendrá que mostrar unos u otros datos personales (será necesario implementar de manera específica cada clase derivada). Una vez realizado lo anterior implemente al ciento por ciento las tres clases (con todos sus atributos y métodos) y desarrolle un programa que permita crear un objeto de tipo `EmpleadoFijo` y otro de tipo `EmpleadoTemporal` para ingresar información al usuario y muestre la información por medio del método `visualizar`.
4. Suponga que se tiene declarada la siguiente clase abstracta:

```
public abstract class EjemploClaseAbstracta{
    public abstract void cualquierMetodoAbstracto( );
    public int valor (int otroValor){
        return otroValor * 4;
    }
}
```

Ahora indique si las siguientes instrucciones son o no válidas. Deberá justificar su respuesta:

- a. `EjemploClaseAbstracta mC = new EjemploClaseAbstracta( );`
- b. `EjemploClaseAbstracta nC = null;`

# Interfaces

## Objetivos

- Reconocer la importancia del uso de interfaces en la programación orientada a objetos.
- Identificar la sintaxis utilizada para la creación de una interfaz.
- Aplicar los conceptos estudiados para resolver problemas mediante el uso del lenguaje de programación Java.

## Introducción

Una interfaz define una forma estándar y pública de especificar el comportamiento de una clase. Al definir una interfaz, todos sus métodos deben ser abstractos, y al definir una clase concreta esta deberá implementar la interfaz, es decir, implementar todos sus métodos.

En definitiva, una interfaz es como una clase *abstract* en la que todos sus métodos son *abstract*. Los métodos sin implementar al interior

---

de una interfaz indican cómo será el comportamiento sin definir (implementación). Con el propósito de ilustrar de manera práctica el concepto, veamos el siguiente ejemplo:

En el cuento de “Los tres cerditos y el lobo” se puede identificar la clase `Cerdo` y la clase `Lobo`. Ambas son subclases de la clase `Animal`. Hay acciones que no podrían heredarse de la clase `Animal`, por ejemplo la acción `Gritar`. Esta acción, según el cuento, no es común a las clases `Cerdo` o `Lobo`. Una solución podría ser definir una interfaz que proporcione los diferentes métodos (o acciones) de la clase `Cerdo` y también, si así se desea, implemente los métodos (o acciones) de la clase `Lobo`. De esta forma, la clase `Cerdo` implementará su método `Gritar` de una manera y la clase `Lobo` de otra.

Una interfaz se define de la siguiente manera:

```
public interface [NombreDeLaInterface] {  
    // métodos sin implementación  
}
```

En el siguiente ejemplo se muestra una interfaz que define la relación entre dos objetos:

```
public interface RelacionObjetos {  
    public boolean esMayor(Object a, Object b);  
    public boolean esMenor(Object a, Object b);  
    public boolean esIgual(Object a, Object b);  
}
```

Para crear una clase concreta que implemente una interfaz, se debe usar la palabra reservada *implements*:

```
public class Linea implements RelacionObjetos {  
    private double linea1;  
    private double linea2;
```

```

        private double linea3;
        private double linea4;

        public Linea(double l1, double l2, double l3, double l4){
            this.linea1=l1;
            this.linea2=l2;
            this.linea3=l3;
            this.linea4=l4;
        }

        public double obtenerLongitud(){
            double longitud= Math.sqrt((linea2 -
linea1) * (linea2 - linea1) + (linea4 - linea3) *
(linea4 - linea3));
            return longitud;
        }

        public boolean esMayor(Object a, Object b){
            double aLongitud = ((Linea)a).obtenerLongitud( );
            double bLongitud = ((Linea)b).obtenerLongitud( );
            return (aLongitud > bLongitud);
        }
    }

```

*Nota:* se deja a consideración del estudiante complementar el ejercicio anterior.

## Prerrequisitos

- Revisar los conceptos herencia, polimorfismo, clases abstractas y métodos abstractos.

- Desarrollar la actividad 17 de este libro.

## Ejercicios

1. Elabore un cuadro comparativo entre interfaz y clase abstracta en donde se describan sus ventajas y desventajas, y se exponga cuándo es apropiado utilizar la una o la otra.
2. ¿Podría una clase implementar varias interfaces? Justifique su respuesta.
3. ¿En qué escenarios resulta de mayor utilidad definir una interfaz antes que una clase abstracta?
4. ¿Qué comentarios podría emitir con relación al siguiente fragmento de código?

```
public interface Algo {  
    private abstract void unMetodo( );  
}
```

5. Defina una clase denominada `ArrayEntero` que declare un atributo de tipo `integer[ ]` y que implemente una interfaz llamada `Operación`. La clase `ArrayEntero` deberá implementar, además, el método `asignar` que consiste en asignar de manera aleatoria diez valores numéricos enteros al atributo declarado. Así mismo, deberá crear una clase principal que permita realizar el test a las clases definidas previamente. A continuación se muestra la implementación de la interfaz:

```
public interface Operación {  
    int valorMinimo( );  
    int valorMáximo( );  
    int producto( );  
    int sumatoria( );  
}
```



Actividad 19 //

# Excepciones

## Objetivos

- Reconocer la importancia del uso de interfaces en la programación orientada a objetos.
- Identificar la sintaxis utilizada para la creación de una interfaz.
- Aplicar los conceptos estudiados para resolver problemas mediante el uso del lenguaje de programación Java.

## Introducción

Una excepción es un evento que ocurre durante el proceso de ejecución de un programa y que interrumpe el flujo normal de las sentencias. Java hace uso de excepciones para gestionar el tratamiento de errores en sus programas. Por ejemplo: de acceso a posiciones inválidas de un arreglo, de falta de memoria en el sistema, de apertura de un archivo que no existe, entre muchos otros. Muchas clases de errores pueden generar

excepciones. Desde problemas de hardware, como la avería de un disco duro, hasta simples errores de programación.

En Java las excepciones están diseñadas para detectar y corregir errores. Cuando se presenta un error, una aplicación no debe “morirse”; por esto debe existir un mecanismo que permita controlar los programas. El manejo de excepciones también es una estructura de control diseñada para que los programas construidos puedan gestionar el tratamiento de condiciones anormales.

## Prerrequisitos

- Se recomienda tener en cuenta lecturas adicionales para mayor claridad en el manejo de excepciones.

## Ejercicios

1. Una de las excepciones más comunes es `OutOfMemoryError`. Determine cuál o cuáles son los motivos por los que se genera ese tipo de excepción.
2. ¿Qué diferencia existe entre las excepciones revisadas y las de ejecución?
3. Observe el siguiente fragmento de código e indique qué sucedería al ejecutarlo:

```
String men1 = "hola";  
int men2 = Integer.parseInt(men1);
```

4. Escriba un programa que contenga un método denominado `characterException` que realice la siguiente acción:  
Reciba como parámetros una cadena `String` y un entero `int`. Si el entero está entre cero (0) y la longitud de la cadena [puede hacer uso del método `length()` de la clase `String`], el programa mostrará el caracter en la posición correspondiente [puede utilizar el método `charAt()` de la

clase `String`]. En caso contrario, construirá y arrojará una excepción de tipo `Exception`.

5. Cree una clase denominada `ClaseGenericaExcepcion` con un método `main( )` que genere un objeto de la clase `exception` dentro de un bloque `try`. Proporcione al constructor de `exception` un argumento `string`. Capture la excepción dentro de una cláusula `catch` y muestre en pantalla el argumento `String`. Añada una cláusula `finally` y muestre un mensaje para demostrar qué ocurre.
6. Observe el siguiente código e indique qué se mostrará en pantalla cuando se ejecute el programa:

```
class Ejemplo01 {
    private static int metodo() {
        int valor = 0;
        try {
            valor = valor + 1;
            valor = valor + Integer.parseInt("W");

            valor = valor + 1;
            System.out.println("Valor al final del try: " + valor);
        } catch (NumberFormatException e) {
            valor = valor + Integer.parseInt("42");

            System.out.println("Valor al final del catch: " + valor);
        } finally {
            valor = valor + 1;
            System.out.println("Valor al final del finally: " + valor);
        }
        valor = valor + 1;
    }
}
```

```

        System.out.println("Valor antes
del return: " + valor);
        return valor;
    }

    public static void main(String[] args) {
        try {
            System.out.println(metodo());
        } catch (Exception e) {
            System.err.println("Exception en me-
todo()");
            e.printStackTrace();
        }
    }
}

```

## Referencias bibliográficas

- Eckel, B. (2007). Tratamiento de errores mediante excepciones. *Piensa en Java* (pp. 277-315). Madrid: Prentice-Hall.
- Horton, I. (2011). Exceptions. *Beginning Java* (7 ed.) (pp. 279-306) (s.l.): John Wiley & Sons, Inc.
- Horstmann, C. (2010). Input/Output and exception handling. *Big Java* (4 ed.) (pp. 481-490) (s.l.): John Wiley & Sons, Inc.

---

Actividad 20 //

# Archivos y directorios

## Objetivos

- Identificar las clases utilizadas para la gestión de archivos y de directorios.
- Emplear las clases necesarias para almacenar y recuperar información en un archivo.
- Crear excepciones en el acceso a archivos.
- Diferenciar el tratamiento de los archivos de caracteres frente a los archivos de *bytes*.

## Introducción

En el paquete “java.io” de Java existen diversas clases que facilitan el trabajo con archivos, bien sean de caracteres o de *bytes* (binarios), o de acceso secuencial o aleatorio. Los archivos se clasifican según su contenido. Los de caracteres o de texto son aquellos creados exclusivamente con caracteres; esto significa que pueden ser creados y visualizados me-

---

dante cualquier editor de texto que ofrezca el sistema operativo, como por ejemplo: Notepad, vi, Edit, entre otros.

Los archivos binarios (o de *bytes*) son aquellos que no contienen caracteres reconocibles, sino otro tipo de información como música, imágenes, videos, entre otros. Estos archivos sólo pueden ser abiertos por aplicaciones concretas que entiendan cómo están almacenados los *bytes* dentro de los mismos y que reconozcan la información que contienen.

Finalmente, están los archivos de tipo secuencial y los de tipo aleatorio, denominados así por la forma en la que se accede a los datos dentro los mismos. La información almacenada en los archivos secuenciales es una secuencia de *bytes* (o caracteres), de modo que para acceder al *byte* (o carácter) “i-ésimo” es necesario pasar antes por todos los anteriores (“i-1”). En los archivos aleatorios, a diferencia de los secuenciales, se puede acceder directamente a una posición concreta del archivo sin tener que hacer el recorrido por los datos anteriores. Un ejemplo de acceso aleatorio en programación es el uso de *arrays*.

## Prerrequisitos

- Antes de realizar la práctica, se aconseja leer sobre las siguientes clases:  
`FileReader`, `FileWriter`, `FileInputStream`, `FileOutputStream`, `RandomAccessFile`, `DirFilter`.
- Hacer las lecturas sugeridas en la bibliografía.
- Recordar el uso y la importancia de la gestión de excepciones.
- Emplear las instrucciones correctas del lenguaje de programación Java para la gestión de excepciones.

## Ejercicios

1. ¿Qué pasaría si se intenta abrir un archivo para escritura, pero el archivo (o dispositivo) está protegido contra escritura? Intenta desarrollar un programa para realizar esta prueba.

2. Escriba un programa que pregunte al usuario el nombre de un archivo y muestre el número de caracteres, palabras y líneas que contenga.
3. Escriba un programa denominado `ConcatenarArchivos` que concatene el contenido de varios archivos en uno solo. Por ejemplo:

```
java ConcatenarArchivos archivo1.txt ar-
chivo2.txt archivo3.txt todos.txt
El archivo todos.txt contendrá los archivos: archivo1.
txt, archivo2.txt y archivo3.txt.
```

4. Escriba un programa denominado `Buscador` que explore en todos los archivos dados como argumentos y muestre todas las líneas que contengan una palabra reservada enviada como argumento. Por ejemplo:

```
java Buscador programa archivo1.txt ar-
chivo2.txt archivo3.txt
```

Como salida debe mostrarse, para cada archivo, la línea en donde se encuentre la palabra buscada. En el ejemplo se quiere buscar la palabra “programa” en los archivos: `archivo1.txt`, `archivo2.txt` y `archivo3.txt`.

5. A continuación se presentan dos bloques de código. Encuentre el error en cada caso y explique brevemente cómo podría resolverlo:

- Suponga que `empresa`, `cuenta` y `sueldo` están declaradas:

```
ObjectOutputStream outputStream;
outputStream.writeInt(cuenta);
```

```
outputStream.writeChars(empresa);
outputStream.writeDouble(sueldo);
```

- Las siguientes instrucciones permiten la lectura de un registro almacenado en el archivo “clientes.txt.” La variable `inRegistros` es utilizada para referirse al archivo.

```
Scanner inRegistros = new Scanner(new
File("clients.txt"));
RegistroCliente registro = (Registro-
Cliente) inRegistros.readObject( );
```

6. Observe el siguiente programa (suponga que existe el archivo “a.txt” y que su contenido es la palabra “Hola”): ¿cuál sería la salida o resultado final cuando se ejecute el programa?

```
import java.io.FileInputStream;
import java.io.IOException;

public class Ejemplo01 {

    public static void main(String[] args) {
        try {
            FileInputStream f = new Fi-
leInputStream("a.txt");
            System.out.println(f.
read());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```



---

## Referencias bibliográficas

- Eckel, B. (2007). Entrada y salida. *Piensa en Java* (pp. 587-638). Madrid: Prentice-Hall.
- Horton, I. (2010). Input/Output and exception handling. *Big Java* (4 ed.) (pp. 467-503) (s.l) John Wiley & Sons, Inc.
- Horstmann, C. (2010). Input / Output and exception handling. *Big Java* (4 ed.) (pp. 467-503 (s.l): John Wiley & Sons, Inc.



---

Actividad 21 //

# Serialización de objetos

## Objetivos

- Diseñar y construir clases serializables.
- Aplicar los conceptos de serialización para escribir objetos sobre un archivo y recuperar datos del mismo.
- Identificar elementos no serializables.

## Introducción

Java facilita el almacenamiento y la transmisión del estado de un objeto mediante un mecanismo conocido con el nombre de serialización. La serialización de un objeto consiste en generar una secuencia de *bytes* lista para su almacenamiento o transmisión. Después, por medio de la deserialización, se puede reconstruir el estado original del objeto.

El concepto de serialización puede ser algo confuso. Para aclararlo un poco, veamos el siguiente ejemplo: cuando se construye un

---

documento (por ejemplo, un documento en formato Excel) y se almacena en el disco duro, lo que se está haciendo realmente es serializar los datos de tal manera que se pueda obtener una copia posteriormente. La serialización permite que la transferencia de datos a través de la red sea mucho más eficiente y más fácil:

Para serializar un objeto, este debe implementar la interfaz “`java.io.Serializable`” (que lo único que hace es marcar el objeto como serializable, sin necesidad de implementar ningún método); en caso contrario, no se podrá hacer la serialización del objeto. La interfaz serializable no tiene métodos, sólo es usada para informar a la JVM de que un objeto va a ser serializado. (Fragmento tomado de: [http://catarina.udlap.mx/u\\_dl\\_a/tales/documentos/lis/montero\\_g\\_g/capitulo2.pdf](http://catarina.udlap.mx/u_dl_a/tales/documentos/lis/montero_g_g/capitulo2.pdf))

Java utiliza su propio formato para serializar los objetos y almacenarlos en un archivo; por ende, este último no puede ser leído tan fácilmente con un editor de textos. Las actividades planteadas para esta práctica permiten interpretar los conceptos inherentes al proceso de serialización de objetos.

## Prerrequisitos

- Dominar el manejo de archivos y la gestión de excepciones.
- Hacer las lecturas sugeridas en la bibliografía.

## Ejercicios

1. Defina una clase `Empleado` con los atributos nombre, apellido, edad y salario. Escriba en un archivo binario los datos de un objeto de la clase `Empleado`.
2. Después de realizar el ejercicio anterior, intente editar (puede utilizar el bloc de notas) el archivo generado y responda a la siguiente pregunta: ¿los datos almacenados en el archivo son legibles? Si la respuesta es negativa, ¿cuál es su opinión al respecto?

3. Con base en la clase `Empleado` diseñada en el ejercicio inicial de esta práctica, intente leer los objetos almacenados; ¿qué conclusiones puede emitir?
4. Con base en los ejercicios anteriores resuelva el mismo ejercicio, pero ahora evite la lectura de por lo menos dos de los atributos definidos (debe utilizar la palabra reservada *transient*).
5. ¿A qué se debe, en ocasiones, la aparición de la excepción `java.io.InvalidClassException`? Justifique su respuesta.
6. ¿En qué momento se podría generar una excepción `NotSerializableException`? Justifique su respuesta.

## Referencias bibliográficas

- Eckel, B. (2007). Serialización de objetos. *Piensa en Java* (4 ed.) (pp.639 – 649). Madrid: Prentice-Hall.
- Horton, I. (2011). Serializing Objects. *Beginning Java*. (7 ed.) (pp.451-468) (s.l.): John Wiley & Sons, Inc.
- Deitel, p., y Deitel, H. (2012). Files Streams and Object Serialization. *Java How to Program*. (9 ed.) (pp.719-764) (s.l.): Prentice-Hall.



Actividad 22 //

# Gestión de memoria

## Objetivos

- Entender el funcionamiento de la administración de memoria realizada por la JVM (*Java Virtual Machine*).
- Identificar las excepciones de tipo `OutOfMemory` y `StackOverflow`.
- Evaluar el uso de los parámetros de configuración utilizados por la JVM.
- Aplicar algunos de los parámetros de configuración utilizados por la JVM.

## Introducción

La JVM (*Java Virtual Machine*) es la encargada de ejecutar los programas en Java, y de reservar, asignar y liberar la memoria utilizada por la aplicación que se esté ejecutando en el momento. En lenguaje C, la respon-

sabilidad de la gestión de memoria de los programas es del programador. En este lenguaje es necesario utilizar instrucciones como `malloc` y `calloc` para asignar memoria dinámica, y la función `dispose` para liberar la memoria que ha sido asignada y que puede ser reutilizada. En Java, en cambio, la gestión dinámica de la memoria es responsabilidad del mismo lenguaje. Las variables son asignadas en la memoria `Heap` en la medida en que se crean y dicha memoria se libera de manera automática cuando ya no es necesaria, por medio de un proceso interno propio de la JVM denominado *Garbage Collector*.

En Java, a escala macro, se pueden identificar tres (3) zonas de memoria: la zona de datos, la memoria `Heap` y la memoria `Stack`. *Garbage Collector* (o recolector de basura) es un proceso automático de baja prioridad que se ejecuta dentro de la JVM y se encarga de liberar aquella memoria del `Heap` que ya no se utiliza y que, por lo tanto, podría ser utilizada por otros programas. Un objeto es candidato para ser eliminado cuando desde el `Stack` ninguna variable hace referencia al mismo.

El siguiente código permite hacer peticiones de recursos de memoria mediante un ciclo iterativo. Se observa que el `array` de 10 MB es ligeramente mayor a 10 MB a causa de la sobrecarga del mismo:

```
public class MemoryCrash{
    public static void main(String[] args){
        int max = 100;
        Runtime rt = Runtime.getRuntime();
        Object[ ] objects = new Object[max];
        for(int i=0; i < max; i++){
            System.out.println("Tenemos " + i *
10 + " MB y añadimos 10MB");
            Objects[i] = new long[10 * 1024 * 128];
            System.out.println("Memoria libre: "
+ rt.freeMemory( ));
            System.out.println("Memoria total: "
+ rt.totalMemory( ));
        }
    }
}
```



```

        Thread.sleep(2 * 1000);
    } catch (InterruptedException ie) {
        System.out.println("Inte-
rrumpido.");
    }
}
}
}
}

```

(Tomado de <https://rubensawordpress.com/2006/10/03/java-jvm-memory-mana/>)

Cuando ejecute el código se generará una excepción de tipo `OutOfMemory`.

## Prerrequisitos

Para ampliar el conocimiento sobre la gestión de memoria realizada por la JVM, se recomienda leer sobre los siguientes temas:

- Zonas que hacen parte de la memoria “Heap”.
- Zonas que hacen parte de la memoria “Non-Heap”.
- Parámetros utilizados para la configuración de la JVM.

## Ejercicios

1. Una excepción de tipo `OutOfMemoryError` se puede generar por varios motivos. Investigue los diferentes tipos existentes y emita un comentario personal sobre cada uno de ellos. Por ejemplo, podría dar una alternativa para su solución.
2. Investigue en qué casos se podría generar una fuga de memoria (*Memory Leak*).
3. Investigue sobre las diferentes herramientas existentes para llevar a cabo un análisis de monitoreo sobre el funcionamiento de la memoria mientras un programa es ejecutado por la JVM. Podría elaborar un cuadro en donde describa ventajas,

---

desventajas, características de cada herramienta, entre otros aspectos. Como apoyo, estas son algunas de las herramientas: *VisualVM*, *Memory Analyzer (MAT)*, *GCViewer*, *JProfile*, *JConsole*.

4. Investigue sobre la clase `Runtime` que ofrece Java y sus diferentes métodos. Por ejemplo: `freeMemory`, `gc`, `maxMemory`, `exec`, `totalMemory`.
5. Ejecute el código descrito en la parte introductoria y use como parámetros “-Xms2m” y “-Xmx64m”. Luego, vuelva a ejecutar el código, pero en esta segunda ocasión utilice como parámetros “-Xms512m”. ¿Qué conclusiones podría emitir?
6. Con base en los resultados del ejercicio anterior, modifique los parámetros de tal forma que el programa termine de manera correcta.
7. Investigue sobre los diferentes tipos de algoritmos de *Garbage Collector*: ¿qué características posee cada uno de ellos? Mencione ventajas, desventajas, entre otros aspectos.

## Referencias bibliográficas

Shirazi, J. (2000). *Profiling Tools*. Java Performance Tuning (pp. 21-76). USA: O'Really & Associates, Inc.

# Recursividad

## Objetivos

- Identificar los conceptos caso base y caso recursivo.
- Resolver problemas mediante la aplicación de métodos recursivos.

## Introducción

La recursividad es una técnica de programación muy eficaz, que puede ser utilizada en algunas ocasiones en reemplazo de la iteración. Esta técnica permite que el algoritmo se invoque a sí mismo para resolver una versión más pequeña del problema original. Un diseño bien elaborado de un algoritmo permite solucionar problemas complejos y de forma eficiente. Sin embargo, un diseño incorrecto podría generar consecuencias bastante desagradables en la programación. Por ejemplo, podría producir un ciclo iterativo infinito.

Todo algoritmo recursivo debe contener un caso base y un caso recursivo. El primero permite la salida o finalización del método recursivo, y el segundo consiste en la llamada al método recursivo.

## Prerrequisitos

- Tener claridad sobre los conceptos relacionados con la gestión de excepciones.
- Hacer lecturas sobre el significado y la aplicabilidad de los parámetros de la JVM.

## Ejercicios

1. Observe el siguiente fragmento de código:

```
class Ejercicio {  
    public static void main(String [ ] args){  
        metodo1(4);  
    }  
    public static void metodo1(int n){  
        if (n > 2) System.out.print('X');  
        else{  
            metodo1(n-1);  
            System.out.print('O');  
        }  
    }  
}
```

Ahora, haga lo siguiente:

- Identifique el caso base.
  - Identifique el caso recursivo.
  - Cuando se ejecute el programa, ¿cuál será el resultado?
2. Reescriba el programa anterior sin hacer uso de “recursividad”. Compruebe que arroje el mismo resultado cuando se ingrese el valor entero cuatro (4).

3. Elabore un programa que contenga un método denominado `valorRecursivo` que reciba como argumento un valor entero “n” y que cumpla con la condición de ser mayor o igual a uno (1). El método deberá retornar el valor recursivo del número de la siguiente forma:

```
valorRecursivo (1) = 1
valorRecursivo (n) = (n - 3) + valorRe-
cursivo( n - 1)
```

4. Elabore un programa mediante la clase `BigInteger` que haga uso de la recursión y muestre por pantalla el factorial de los números comprendidos entre 105 y 108.
5. Construya un método recursivo que reciba como parámetros dos números enteros positivos “a” y “b”. Como resultado, deberá mostrar la cantidad de veces que se encuentra el entero “b” en el entero “a”. Por ejemplo, si el valor de  $a=1231231233$  y  $b=3$ , el resultado será 4.
6. Investigue sobre el concepto de recursividad directa e indirecta: ¿en qué tipo de situaciones se aplica cada uno?

## Referencias bibliográficas

- Joyanes Aguilar, L., y Zahonero Martinez, I. (2008). Recursividad. *Estructuras de datos en Java* (pp.125-133). Madrid: Mc Graw-Hill.
- Allen Weiss, M. (2000). Recursividad. *Estructuras de datos en Java* (pp.165-211). (s.l.): Addison Wesley-Pearson.
- Deitel, P, y Deitel, H. (2012). Recursion. *Java How to Program* (9 ed.) (pp.765-797) (s.l.): Prentice-Hall.



---

Actividad 24 //

# Algoritmos de ordenamiento

## Objetivos

- Estudiar el funcionamiento de los algoritmos de ordenamiento.
- Determinar la eficiencia de los algoritmos de ordenamiento y su relación con componentes del hardware como la memoria RAM y el procesador.
- Aplicar alguno de los algoritmos de ordenamiento para el desarrollo de programas.

## Introducción

Los algoritmos de ordenamiento, como bien lo dice su nombre, permiten ordenar vectores o matrices con valores de tipo numérico o de `String`. Su estudio es muy importante, específicamente para aplicar al número de comparaciones e iteraciones que dependerá en gran medida de la cantidad de datos de entrada. Los algoritmos tienen diferencias que

---

los hacen más o menos eficientes, dependiendo de la rapidez y eficacia demostrada por cada uno de ellos. Los algoritmos básicos de ordenación más simples y básicos son:

- Algoritmos de ordenación por selección.
- Algoritmos de ordenación por inserción.
- Algoritmos de ordenación por burbuja.

También existen algoritmos recursivos que son un poco más complejos, requieren mayor análisis y son más rápidos que los iterativos. Para este tipo de algoritmos se puede utilizar la técnica “divide y vencerás”. Al ser estos un poco más efectivos, es posible que el tiempo de ejecución y de ordenación sea más óptimo.

## Prerrequisitos

- Realizar las lecturas propuestas como referencia.
- Tener disponibles los diferentes algoritmos de ordenamiento que se utilizarán.
- Estudiar previamente el método `currentTimeMillis()` de la clase `System` bajo el entorno de programación Java.
- Conocer el tratamiento de excepciones y el manejo de archivos.

## Ejercicios

1. Desarrolle un programa que genere de manera aleatoria 8.192 números enteros positivos (los números deben ser generados entre 1 y 10.000 y no deben existir números repetidos). Los números deben ser almacenados en un `array` a medida que se generen (no se debe utilizar la estructura `ArrayList`). Una vez almacenados en el `array`, debe utilizar cada uno de los algoritmos de ordenamiento descritos en la siguiente tabla y calcular el tiempo empleado sólo en el proceso de ordenamiento.

Será necesario diligenciar la siguiente tabla:



Algoritmo	Número de comparaciones	Número de intercambios	Tiempo empleado (en segundos)
Burbuja			
Selección			
Inserción			
Quicksort iterativo			
Quicksort recursivo			
Shell sort			
Merge sort			

Con base en los resultados anteriores, responda los siguientes interrogantes:

- ¿Qué características (memoria RAM, procesador) posee la máquina en la que se ejecutaron los algoritmos?
- Si los algoritmos hubiesen sido ejecutados en una máquina de características superiores a la utilizada, ¿el tiempo de ejecución sería el mismo? ¿Cuál es su opinión?
- Si los algoritmos hubiesen sido ejecutados de manera simultánea en la misma máquina, descrita en el literal a, ¿el tiempo de ejecución sería el mismo? ¿Cuál es su opinión?
- Con el apoyo de Excel, genere una gráfica con base en el algoritmo utilizado vs. tiempo empleado. ¿Qué conclusiones puede emitir con base en esta gráfica?
- ¿Cuáles son el peor y el mejor caso en cada uno de los algoritmos utilizados?
- Explique brevemente, y adjunte gráficos de ser posible, los cambios realizados en los algoritmos de ordenamiento, toma-

dos como base para la obtención de la información requerida en el diligenciamiento de la tabla descrita anteriormente.

- Suponga que se tienen “N” parejas de datos como entrada, donde el primer dato corresponde a un número y el segundo es uno de tres colores (rojo, azul o amarillo). Elabore un programa que permita ordenar (se puede seleccionar cualquier algoritmo de ordenamiento) “N” parejas por el segundo dato (los colores amarillo, azul o rojo), de tal forma que siempre que haya colores iguales se deberá tener en cuenta el primer dato (el número). Por ejemplo:

Datos de entrada: (1, azul), (10, rojo), (4, azul), (6, amarillo), (9, rojo).

La salida del programa debe ser: (6, amarillo), (1, azul), (4, azul), (9, rojo), (10, rojo).

2. Construya un programa que le permita al usuario ingresar dos valores numéricos enteros. El primero corresponderá al número de filas, y el segundo al número de columnas de una matriz. Una vez hayan sido ingresados los datos de la matriz, se deben ordenar primero por filas y luego por columnas. Se debe mostrar la matriz resultante.
3. Construya un programa que contenga un método denominado `ordenarPalabras`. El método deberá recibir un texto finalizado en un punto (“String” de caracteres de máximo 200 caracteres de longitud). Como resultado, se deberá mostrar al usuario el mismo texto ordenado por cada una de las palabras que lo conforman, en orden ascendente.
4. Investigue sobre los algoritmos de ordenamiento: `Mezcla directa` y `Radix Sort`.

## Referencias bibliográficas

Joyanes Aguilar, L., y Zahonero Martínez, I. (2008). Algoritmos de ordenamiento. *Estructuras de datos en Java* (pp.161-179). Madrid: Mc Graw-Hill.

- Allen Weiss, M. (2000). Algoritmos de ordenación. *Estructuras de datos en Java* (pp. 213-246). (s.l.): Addison Wesley-Pearson.
- Horstmann, C. (2010). Sorting and Searching. *Big-Java* (pp.596-609). (s.l.): Wiley John Wiley & Sons, Inc.
- 

Actividad 25 //

# Algoritmos de búsqueda

## Objetivos

- Identificar la diferencia entre el proceso de búsqueda lineal y binaria.
- Determinar cuál de los algoritmos de búsqueda es más eficiente.
- Entender el proceso que realiza cada uno de los algoritmos.
- Emplear los algoritmos de búsqueda en la construcción de programas.

## Introducción

El algoritmo de búsqueda lineal y el algoritmo de búsqueda binaria son dos de los algoritmos más populares y utilizados. El algoritmo de búsqueda lineal busca un elemento en un *array* de manera secuencial (podría ser en un archivo). Si el elemento o clave que se busca no existe en el *array*, el algoritmo verifica cada elemento y, cuando termina de recorrer el *array*, informa que la clave o el elemento no se encuentra. Si la clave o el elemen-

to se encuentra en el *array*, el algoritmo verifica cada uno de los elementos hasta encontrar la clave y retorna el índice del elemento encontrado.

El algoritmo de búsqueda binaria es mucho más eficiente que el algoritmo de búsqueda secuencial, pero tiene como prerrequisito que el *array* se encuentre ordenado. El algoritmo realiza la primera iteración en la mitad del *array* para verificar la existencia de una clave. Si dicho elemento es la clave que se busca, el algoritmo termina en ese momento.

Suponga que el *array* se encuentra ordenado de menor a mayor; entonces, si la clave que se busca es menor que el elemento ubicado en la mitad del *array*, se descartan los elementos que se encuentran en la segunda mitad del *array* y el algoritmo sólo concentra su búsqueda en la primera mitad. Cada iteración del algoritmo verifica el elemento que se encuentra en la mitad de la porción del *array* que se esté trabajando; es decir, en cada iteración se descarta una parte del *array*.

## Prerrequisitos

- Tener conocimiento sobre la gestión de excepciones y el manejo de archivos.
- Adquirir habilidades en el funcionamiento y uso de los algoritmos de ordenamiento.

## Ejercicios

1. Construya un programa que genere de manera aleatoria 8.192 números enteros positivos en el rango de 4.568 a 15.560, y que los almacene en un *array* de igual tamaño. Deberá tener en cuenta que en el *array* no se permite almacenar números repetidos.
2. Con base en el ejercicio anterior, construya un programa que haga uso del algoritmo de búsqueda binaria para permitir la búsqueda de un valor dado como dato de entrada. En la construcción del ejercicio se debe calcular el tiempo empleado en el proceso de búsqueda y, al finalizar el proceso, se debe visualizar el mismo.

3. Con base en el primer ejercicio de esta práctica, construya un programa que haga uso del algoritmo de búsqueda secuencial y que permita la búsqueda de un valor dado como dato de entrada. En la construcción del ejercicio se debe calcular el tiempo empleado en el proceso de búsqueda y, al finalizar el proceso, se debe visualizar el mismo.
4. Emita sus propias conclusiones con base en los resultados arrojados en los dos ejercicios anteriores.
5. Cada una de las líneas o registros que conforman un archivo de texto contienen una palabra (podría estar conformada por letras mayúsculas, minúsculas, números, símbolos especiales, etc.). Se requiere construir un programa que permita al usuario ingresar una cadena de texto y buscarla al interior del archivo. Como resultado, se debe indicar el número de la línea en donde fue encontrada la cadena (en caso de encontrarla). Tenga en cuenta que la cadena podría estar varias veces al interior del archivo.
6. Construya un programa que permita ingresar como datos de entrada una serie de parejas “x”, “y” que formen un punto en el plano cartesiano. El programa deberá mostrar como resultado la mayor distancia existente entre los puntos ingresados por el usuario y, además, permitir visualizar la pareja de puntos<sup>3</sup>.

## Referencias bibliográficas

- Joyanes Aguilar, L., y Zahonero Martinez, I. (2008). Búsqueda en listas: búsqueda secuencial y binaria. *Estructuras de datos en Java*. Madrid: Mc Graw-Hill, 2008, pp.182-185.
- Allen Weiss, M. (2000). Algoritmos de Ordenación. *Estructuras de datos en Java* (pp.213-246). (s.l.): Addison Wesley-Pearson.
- Horstmann, C. (2010). Sorting and Searching. *Big-Java*. (pp. 614-619). (s.l.): Wiley John Wiley & Sons, Inc.

---

<sup>3</sup> El programa deberá solicitarle al usuario la cantidad de puntos.



Actividad 26 //

# Hilos

## Objetivos

- Reconocer la importancia del uso y de la gestión de hilos bajo el entorno de programación Java en el desarrollo de aplicaciones de *software*.
- Identificar las clases utilizadas para la creación de hilos.
- Crear hilos utilizando la interfaz `Runnable` y la clase `Threads`.
- Analizar el proceso de sincronización de hilos y el control de los mismos por medio de prioridades.

## Introducción

La mayor parte de las veces el software ejecuta diversas tareas de manera simultánea o casi simultánea. El concepto de hilos está relacionado con el *software*, puesto que la generación de estos no se produce por *hardware*. Sin embargo, al disponer de una CPU *multicore* se podrían

---

realizar tantas tareas como *cores* tenga cada CPU. Cuando se hace referencia a un hilo de ejecución, se habla de la ejecución secuencial de un grupo o conjunto de instrucciones que hacen parte de un programa. En Java, cada programa tiene un hilo por defecto.

En un momento determinado, un hilo puede estar en uno de los siguientes estados: *running* (en curso de ejecución, en control de la CPU); *ready* (puede ejecutarse pero no se le ha dado la orden); *resumed* (listo para ejecutarse después de haber estado suspendido o bloqueado); *suspended* (de manera voluntaria, permite que otros hilos se ejecuten); *blocked* (esperando por algún otro recurso o que ocurra un evento).

La prioridad de un hilo está definida por un valor entero positivo entre uno (1) y diez (10). La prioridad determina qué hilo recibe el control de la CPU y consigue ser ejecutado primero. A mayor valor de prioridad, mayor es la oportunidad de ser ejecutado primero. Si dos hilos poseen la misma prioridad, la ejecución dependerá en gran medida del sistema operativo. Cuando varios hilos son ejecutados y acceden a un mismo objeto, se podrían presentar resultados inesperados si dichos hilos no se encuentran sincronizados. El proceso de sincronización es un mecanismo que permite bloquear un hilo de forma exclusiva, es decir, que no se pueda interrumpir.

Cuando los hilos acceden a un recurso común, la forma de evitar eventos inesperados es mediante el uso del mecanismo de sincronización. Por ejemplo: en el proceso de gestión de archivos se podría utilizar un hilo para llevar a cabo la lectura de su contenido y otro para la escritura de información sobre el mismo.

## Prerrequisitos

- Tener conocimiento sobre la gestión de excepciones y el manejo de archivos.
- Realizar las lecturas recomendadas.



## Ejercicios

1. Cree un programa con 4 hilos. Cuando ejecute el programa, cada uno de los hilos debe mostrar cinco mensajes por pantalla indicando la fecha y hora del sistema en el momento de la visualización. Entre cada visualización, cada hilo debe esperar 1000 milisegundos.
2. Cree un programa con 2 hilos. El programa debe simular el manejo de una cuenta de ahorros, el ingreso y la extracción de dinero de la cuenta. La cuenta debe tener un capital inicial (valor que debe ser ingresado por el usuario) y debe realizar 10 ingresos y 5 extracciones. Tanto el monto por concepto de ingreso como el de extracción debe ser un valor entero aleatorio entre 1 y el capital inicial ingresado por el usuario.
3. Modifique el programa anterior de manera que sea desarrollado por medio del concepto de sincronización de hilos.
4. Elabore un programa que use un solo hilo. El programa debe permitirle la captura de un texto por teclado y usted debe verificar si el texto ingresado está escrito en letras mayúsculas o minúsculas.
5. Construya un programa que permita la captura de una palabra ingresada como parámetro y escriba cada una de las letras que la conforman, en orden aleatorio. Para lograr tal efecto, debe asignar un hilo a cada letra. Este debe detenerse durante un tiempo aleatorio y después mostrar el carácter.
6. Observe el siguiente fragmento de código y suponga su ejecución:

```
HiloRunnable h1 = new HiloRunnable("Ho-  
la, inicio de hilo");  
HiloRunnable h2 = new HiloRunnable("Fi-  
naliza el hilo");  
h1.run( );  
h2.run( );
```

---

Explique si es posible que ambos hilos se ejecuten al mismo tiempo.

7. En Java, cualquier interfaz gráfica tiene más de un hilo. ¿Cómo podría probar y demostrar este caso?

## Referencias bibliográficas

Deitel, P., y Deitel, H. (2012). Multithreading. *Java How to Program* (9 ed.) (pp.1045-1117) (s.l.) Prentice Hall.

Horstmann, C. (2010). Multithreading. *Big-Java*. (pp.801-838). (S.L.): Wiley John Wiley & Sons, Inc.

Actividad 27 //

# Sockets

## Objetivos

- Adquirir los conocimientos básicos sobre los protocolos TCP y UDP.
- Construir una aplicación Java a través del uso de sockets y datagramas.
- Construir un *server* multihilos sencillo.

## Introducción

La transmisión de datos entre las redes de computadores se realiza por medio de la pila de protocolos TCP/IP. Dicha pila consta de una serie de capas, tal como se registra en el siguiente diagrama:

Nivel de aplicación (http, ftp, ssh, telnet)
Nivel de transporte (tcp, udp)
Nivel de red (ip, icmp, arp)
Nivel de enlace (ethernet)
Nivel físico

Cuando se desarrolla una aplicación en Java en red estamos trabajando con el nivel de aplicación, pero es posible implementar programas a más bajo nivel (nivel de transporte). En este caso, es importante resaltar las diferencias existentes entre dos protocolos: TCP (*Transmission Control Protocol*) y UDP (*User Datagram Protocol*).

#### TCP:

- Es un protocolo orientado a conexión.
- Entre dos computadores, el flujo de bytes es fiable (llegada en orden, correcta, sin perder nada, control de flujo, control de congestión, entre otros aspectos).
- Protocolos en el nivel de aplicación que usan TCP: telnet, http, ftp.

#### UDP:

- Es un protocolo no orientado a conexión.
- Envía paquetes de datos (datagramas) independientes, sin garantía de llegada correcta.
- Protocolos a nivel de aplicación que usan UDP: ping, tftp.

#### Sockets

Si se quiere establecer una comunicación entre dos procesos que se ejecutan en dos máquinas diferentes, una de las aplicaciones deberá ofrecer un servicio (servidor) y otra lo debe solicitar (cliente). Un computador puede tener una o varias conexiones físicas a la red y diversos servidores pueden estar escuchándose en el mismo computador. Cuando un cliente realiza una petición por medio de una de las conexiones físicas, uno de los puertos permite tanto a TCP como a UDP direccionar los datos a la aplicación correcta entre las diversas que se estén ejecu-

tando en el mismo equipo. Por lo tanto, cualquier aplicación que haga las veces de servidor debe tener configurado un puerto para poder recibir las peticiones.

Los datos que se transmitan a través de la red tendrán información que permita identificar la máquina (32 bits) y el puerto (16 bits) a los que van dirigidos. En definitiva, un *socket* es un extremo de un enlace de comunicación bidireccional entre dos programas que se comunican a través de la red.

## Prerrequisitos

- Tener conocimiento sobre la gestión de excepciones y el manejo de archivos.
- Tener en cuenta la lectura recomendada.

## Ejercicios

A continuación se muestra el código fuente de la implementación de un “cliente” mediante el uso de UDP.

```
import java.net.DatagramPacket;
import java.net.DatagramSocket;
import java.net.InetAddress;
import java.net.SocketTimeoutException;

public class ClienteUDP {
    static final int SERVER_PORT = 8000;
    static final String SERVER = "localhost";
    static final int CLIENT_PORT = 12000;
    static final String CLIENT = "localhost";
    static final String mensaje = "PRUEBA DE SOCKETS";

    public static void main(String[] args) {
```

```

        try {
            InetAddress dirServidor = InetAddress.getByName(SERVER);
            DatagramSocket sDatagram = new DatagramSocket();
            sDatagram.setSoTimeout(30000);
            // Preparar datagrama y enviar
            DatagramPacket dgramEnv = new DatagramPacket(mensaje.getBytes(),
                                                            mensaje.length(), dirServidor, SERVER_PORT);
            sDatagram.send(dgramEnv);
            System.out.println("CLIENTE: Enviando"
                               + new String(dgramEnv.getData()) + " a "
                               + dgramEnv.getAddress().toString() + " : "
                               + dgramEnv.getPort());
            // Preparamos el datagrama de recepcion
            byte array[] = new byte[1024];
            DatagramPacket dgramRec = new DatagramPacket(array, array.length);
            sDatagram.receive(dgramRec);
            System.out.println("CLIENTE: Recibido"
                               + new String(dgramRec.getData(), 0, dgramRec.getLength())
                               + " de " + dgramRec.getAddress().toString() + " : "
                               + dgramRec.getPort());

            sDatagram.close();

        } catch (SocketTimeoutException e) {

```

```

        System.err.println("30 segs sin re-
cibir nada");

        } catch (Exception e) {
            System.err.println("Error: " + e.
getMessage());
        }

    }
}

```

A continuación se muestra parte de la implementación del “servidor”:

```

package com.unbosque.edu.co.sockets;

public class ServidorUDP {

    public static void main(String[] args) {
        try {
            // Crear el socket del servidor
            // Estructura while - infinito -
            // preparar datagrama para recibir
            // preparar datagrama a enviar y se
envia

            // Cerrar el socket del servidor
        } catch (Exception e) {
            System.err.println("Error : " + e.
getMessage());
            e.printStackTrace();

        }
    }
}

```

1. Ingrese las instrucciones correspondientes en la clase `ServidorUDP` en las líneas que aparecen comentadas.
2. ¿Cómo se obtiene la dirección del “servidor”?
3. ¿Qué ocurre si no se especifica la IP del servidor cuando se crea el `DatagramSocket`?
4. ¿Qué pasaría si el tamaño del mensaje que se está recibiendo es mayor al tamaño del *buffer* que ha sido reservado previamente?
5. Sin alterar el código fuente del “servidor”, ¿se podrían ejecutar los dos al mismo tiempo? ¿Qué pasaría?
6. ¿Se podrían ejecutar varios “clientes” para comunicarse con el mismo “servidor”?

En cuanto al protocolo TCP, observe la siguiente implementación de un “cliente”:

```
import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.io.PrintWriter;
import java.net.InetAddress;
import java.net.Socket;
import java.net.SocketTimeoutException;

public class ClienteTCP {
    static final String SERVER_HOST = "localhost";
    static final int SERVER_PORT = 8082;
    static final String mensaje = "PRUEBA DE SOCKETS TCP";

    public static void main(String[] args) {
        try {
            InetAddress dirServidor = InetAddress.getByName(SERVER_HOST);
            Socket socket = new Socket(dirServidor, SERVER_PORT);
```



```

        socket.setSoTimeout(30000);

        System.out.println("CLIENTE: cone-
xion realizada con "
                        + dirServidor.toString() +
" al puerto " + SERVER_PORT);
        // Canal de entrada
        BufferedReader sEntrada = new Buffere-
redReader(new InputStreamReader(
                        socket.getInputStream()));
        // Canal de salida
        PrintWriter sSalida = new PrintWri-
ter(socket.getOutputStream(),
                        true);

        System.out.println("CLIENTE : En-
viando " + mensaje);
        sSalida.println(mensaje);

        String recibido = sEntrada.readLine();
        System.out.println("CLIENTE : Reci-
bido " + recibido);

        socket.close();

    } catch (SocketTimeoutException e) {
        System.err.println("30 segs sin re-
cibir nada");

    } catch (Exception e) {
        System.err.println("Error: " + e.
getMessage());
    }
}

```

7. Construya la implementación del servidor TCP
8. Implemente un “servidor” (TCP) que reciba valores enteros desde un “cliente”. Una vez recibidos los parámetros, el “servidor” deberá presentar el resultado de las operaciones: suma, resta, multiplicación y división de los valores recibidos.
9. ¿Cuál es la diferencia entre un objeto `Socket` y un objeto `ServerSocket`?
10. ¿En qué situaciones se podría presentar una excepción de tipo `UnknownHostException`?
11. Utilice un archivo de texto para construir un `cliente` (puede ser UDP o TCP) que haga la lectura de cada uno de los registros que conforman el archivo. Cada registro leído deberá ser enviado al `servidor`, que, al recibir el registro, deberá mostrar los datos recibidos y la longitud de los mismos. .

## Referencias bibliográficas

- Deitel P, y Deitel, H. (2012). Networking. *Java How to Program* (9 ed.) (pp. 1118-1170) (s.l.) Prentice-Hall.
- Horstmann, C. (2010). Internet Networking. *Big-Java*. (pp.839-864) (s.l.) Wiley John Wiley & Sons, Inc.

Actividad 28 //

# Colecciones

## Objetivos

- Reconocer la importancia de las colecciones implementadas en Java.
- Identificar qué clases de colecciones e interfaces se deben seleccionar para un diseño apropiado, dado un determinado escenario.

## Introducción

En Java, el *framework* de colecciones se representa mediante interfaces. La clasificación de ellas viene dada por:

- *Listas*: se trata de una secuencia o lista ordenada. Este tipo de estructura permite duplicados y es de acceso aleatorio. Posee un iterador `ListIterator` que permite modificar la lista en cualquier sentido, ordenar sus elementos de acuerdo con un criterio predefinido y adicionar elementos sin ningún

---

tipo de restricción. Sin embargo, tiene bajo rendimiento en algunas operaciones. Entre los diferentes tipos de listas están: `ArrayList`, `LinkedList`, `Vector` y `CopyOnWriteArrayList`.

- **Sets:** en este tipo de colección no se permiten dos elementos iguales. Los sets o conjuntos permiten la adición y eliminación de elementos, o determinar si un elemento se encuentra o no en la colección. Sin embargo, la estructura no es de acceso aleatorio. No todas las implementaciones de `set` permiten el ordenamiento y aquellas que lo permiten, lo hacen de manera ineficiente. Tipos de sets: `HashSet`, `LinkedHashSet`, `TreeSet`, `EnumSet`, `CopyOnWriteArraySet`, `ConcurrentSkipListSet`.
- **Maps:** se trata de colecciones que asocian un valor con una clave. Sin embargo, no se permite tener dos claves iguales. Cualquier tipo de dato puede ser utilizado tanto para la clave como para el valor: objetos, datos primitivos, otro tipo de colección, entre otros. Internamente hace uso de un `set` para garantizar la no existencia de claves duplicadas. Tipos de `maps`: `HashMap`, `LinkedHashMap`, `TreeMap`, `EnumMap`, `WeakHashMap`, `HashTable`, `ConcurrentHashMap`.
- **Colas:** son colecciones que permiten la creación de colas LIFO o FIFO. No permiten el acceso de manera aleatoria y, según su implementación, posibilitan la manipulación de objetos al inicio o al final de la estructura. Este tipo de estructura es muy eficiente cuando se trata de recuperar el primer o último elemento de la misma, pero muy ineficiente cuando se intenta acceder a un elemento específico. Tipos de colas: `ArrayDeque`, `LinkedBlockingDeque`, `LinkedList`, `PriorityQueue`, `PriorityBlockingQueue`.

## Prerrequisitos

- Tener conocimientos sobre la gestión de excepciones y el manejo de archivos.

## Ejercicios

1. Diseñe e implemente una aplicación de un programa basado en un conjunto que utilice una colección de DVD. Los datos para cada DVD tendrán un título, una categoría, un año de liberación, un autor y un precio.  
El usuario deberá ser capaz de agregar un nuevo DVD, eliminar un DVD, modificar la información del DVD y mostrar la información de todos los DVD que pertenezcan a una categoría específica.  
Se recomienda que al ejecutar el programa principal se muestre al usuario un menú de opciones.
2. Construya un programa que tome como datos de entrada el contenido de un archivo de texto (cada registro contiene una palabra) y utilice un mapa para almacenar los totales de frecuencia para todas las palabras que conforman el archivo. Una vez terminado el proceso, se debe visualizar el contenido del mapa.
3. ¿Cuál es la diferencia entre las colecciones *set* y *map*?
4. ¿Se podrían almacenar en un *map* dos llaves distintas con el mismo valor?
5. Utilice el mismo archivo de texto del segundo ejercicio e inserte cada registro tanto en un *HashSet* como en un *TreeSet*. Determine el tiempo empleado para cada una de las estructuras. ¿Cuál de ellas es más rápida?
6. ¿Qué sucedería al insertar un valor (tipo de dato primitivo) a una colección?
7. ¿Podrían mostrarse todos los elementos de una colección sin utilizar un iterador? En caso afirmativo, ¿cómo?

8. Construya un programa que permita ingresar un texto a un usuario (longitud máxima 100 caracteres) y que determine y muestre el número de palabras duplicadas en el texto. Ignore los signos de puntuación y tenga en cuenta que las letras mayúsculas difieren de las minúsculas (“Hola” es diferente a “hola”).
9. Se tiene una clase denominada `Vehiculo` con los atributos: placa, modelo, marca, color. Construya el código necesario tanto para la clase `Vehiculo` como para el programa principal, con el fin de hacer lo siguiente:
  - Crear un “ArrayList de vehículos”.
  - Insertar objetos `Vehiculo`.
  - Ordenar la lista de `Vehiculos` por el atributo placa (se debe utilizar la interfaz *Comparable*).
  - Mostrar de forma ordenada y ascendentemente la lista de vehículos.

## Referencias bibliográficas

- Deitel, P, y Deitel, H. (2012). Generic Collections. *Java How to Program* (9ed.) (pp.830-872) (s.l.) Prentice Hall, 2012
- Horstmann, C. (2010). Advanced Data Structures. *Big-Java*. (pp.665-674) (s.l.) Wiley John Wiley & Sons, Inc.

# Genéricos

## Objetivos

- Identificar la sintaxis y la terminología apropiada para hacer uso de la genericidad en Java.
- Reconocer la importancia de la programación genérica como mecanismo para la reutilización de *software*.
- Aplicar los conocimientos de genericidad en la construcción de métodos genéricos e identificar la reutilización de *software*.

## Introducción

Con la aparición de Java 1.5 apareció también el concepto de “genéricos”. Estos han sido utilizados por otros lenguajes durante años, como por ejemplo en C++, donde se evidencian mediante las plantillas. En el caso de Java, algunas de sus clases fueron sustituidas por sus equivalentes genéricos. *Generics* es un esquema compilador que, de manera pre-

---

via, informa sobre el tipo de colección y evita el casting. Se pueden crear y utilizar tipos de elementos (pueden ser clases o interfaces) de un modo genérico cuando se mantiene un código común y el resto se parametriza.

El uso de genéricos permite la creación de clases y métodos en los cuales los tipos de datos (las clases) sobre los que actúan constituyen un parámetro adicional. Por ello, cualquier clase o método creado se debe adaptar a distintos tipos de datos de forma automática. Analicemos un ejemplo para ilustrar el funcionamiento de los genéricos en Java. La clase “ArrayList” permite almacenar una lista de objetos y la estructura gestiona los datos con base en una matriz:

```
ArrayList aL = new ArrayList();  
aL.add("Esto es un String");  
aL.add(new Integer(10));  
aL.add("y ahora, otro String");
```

Al observar el código se puede evidenciar que la clase “ArrayList” permite almacenar objetos de cualquier tipo. En el código anterior se almacenan dos objetos *String* y un tipo *Integer*. Antes de JDK 5 no era posible restringir este comportamiento y esto causaba múltiples errores en el proceso de codificación. Si se quisiera recuperar la información almacenada en el “ArrayList”, sería necesario verificar el elemento al que se quisiera acceder para manipular el tipo de dato (problemas de *casting*) o en caso contrario se presentaría una excepción.

Realmente, la variación radica en indicar el tipo de dato tanto en la colección como en el iterador que se crea. Ejemplo:

```
ArrayList<String> aL = new ArrayList<String>();  
// AL colocar <String> estamos indicando que  
la lista  
// solo acepta Strings  
ListIterator<String> iterador = aL.listIterator();
```



Observe un ejemplo completo. Se evidencia claramente la ausencia de *castings* en el código:

```
ArrayList<String> aL=new ArrayList<String>();
aL.add("String Uno");
aL.add("String Dos");
aL.add("String Tres");
aL.add("String Cuatro");
Iterator<String> it=aL.iterator();
while(it.hasNext()){
    String s=it.next();
    System.out.println(s);
}
```

Ahora, el mismo código con un bucle *for...each*:

```
ArrayList<String> aL=new ArrayList<String>();
aL.add("String Uno");
aL.add("String Dos");
aL.add("String Tres");
aL.add("String Cuatro");
Iterator<String> it=aL.iterator();
for(String s: aL){
    System.out.println(s);
}
```

Ahora, un último ejemplo completo:

```
class EjemploGenerico<T> {
    T obj;

    public EjemploGenerico(T o) {
        obj = o;
    }
}
```

```

        public void tipoClase() {
            System.out.println("El tipo de T es
" + obj.getClass().getName());
        }
    }

    public class EjemploG {
        public static void main(String args[]) {
            // Instancia de EjemploGenerico para In-
            teger.
            EjemploGenerico<Integer> intObj = new
            EjemploGenerico<Integer>(45);
            intObj.tipoClase();

            // Instancia de EjemploGenerico para
            String.
            EjemploGenerico<String> strObj = new Ejemplo-
            Generico<String>("Prueba");
            strObj.tipoClase();

        }
    }
}

```

En el ejemplo anterior, "T" es el tipo genérico que será reemplazado por un tipo real, nombre que recibe el parámetro genérico y que será reemplazado por el tipo real que se le pasará a la clase. Al ejecutar el ejercicio, la salida será la siguiente:

```

EL tipo de T es java.lang.Integer
EL tipo de T es java.lang.String

```

Es importante tener en cuenta que los genéricos en Java funcionan solamente con objetos.

## Prerrequisitos

- Tener conocimiento sobre la gestión de excepciones.
- Haber adquirido destreza en el manejo de colecciones.

## Ejercicios

1. En el último ejemplo presentado en la introducción, ¿qué pasaría si se intenta ejecutar el siguiente código?  
`EjemploGenerico<int> myOb = new EjemploGenerico<int>(51);`
2. Investigue el motivo por el cual se podría generar la excepción `ClassCastException`.
3. Construya una clase genérica llamada `OperacionArray` con las siguientes funcionalidades:
  - Mostrar el elemento menor de un *array*.
  - Mostrar el elemento mayor de un *array*.
  - Buscar un objeto en el *array* y mostrar la última posición de aparición del objeto en el array o -1 si no existe.
4. Construya una clase llamada `EjecutarOperacion` que haga uso de la clase `OperacionArray` para un array de *Integer*.
5. Se requiere crear un método genérico para la clase anterior `OperaciónArray` que elimine la última aparición en el *array* de un objeto dado como parámetro. El método deberá mostrar la posición del objeto eliminado o un valor null en caso de no encontrarlo.
6. Construya una clase denominada `MatrizGenerica` con las siguientes características:
  - Debe tener un constructor que reciba por parámetro el número de filas y columnas de la matriz.
  - Debe contener un método que reciba por parámetro el número de fila, número de columna y un elemento para inser-

tar. El elemento es de tipo genérico y debe ser insertado en la posición indicada.

- Debe contener un método que reciba por parámetro el número de fila y el número de columna, y que muestre el elemento que se encuentra en esa posición. El elemento devuelto es de tipo genérico.
  - Debe contener un método que muestre el número de columnas que contenga la matriz.
  - Debe contener un método que muestre el número de filas que contenga la matriz.
  - Debe contener un método `toString` que muestre en forma de *string* la información almacenada en la matriz.
7. ¿Cuál es la diferencia entre una clase genérica y un método genérico?
  8. Implemente un método estático genérico denominado `IntercambioPareja`. El método debe recibir un par de objetos como parámetros y mostrar una nueva pareja con el primer y segundo elemento intercambiados.
  9. Implemente una versión genérica del algoritmo de búsqueda secuencial.
  10. Implemente una versión genérica de la clase `LinkedList`.

## Referencias bibliográficas

- Deitel P, y Deitel, H. (2012). Generic Classes and Methods. *Java How to Program* (9 ed.) (pp.873-903) (s.l.) Prentice Hall.
- Horstmann, C. (2010). Generic Programming. *Big-Java* (pp.723-732) (s.l.) Wiley John Wiley & Sons, Inc.

---

Actividad 30 //

# Acceso a base de datos

## Objetivos

- Identificar las sentencias básicas utilizadas por el estándar SQL (*Structured Query Language*).
- Aprender la API JDBC que permite la conexión entre programas Java y de estos con las bases de datos.
- Aplicar los conceptos estudiados en las sesiones teóricas para integrar un programa Java y una base de datos que ejecute sentencias SQL.

## Introducción

Java ofrece diversos mecanismos para llevar a cabo el proceso de manipulación y conexión a bases de datos (inserción, borrado, modificación y actualización de registros). Ahora bien, como es necesario tener conocimientos básicos sobre bases de datos relacionales y SQL, aunque este

---

no no sea el objeto de este libro, ofrecemos a continuación una breve introducción al tema.

Las bases de datos almacenan información en forma de tablas. Una tabla está conformada por un conjunto de filas y columnas. Las filas se denominan registros y las columnas campos. Al “mapear” estos conceptos, se puede decir que los campos constituyen cada uno de los atributos de una clase o entidad Java, y que los registros hacen referencia a cada uno de los valores para cada atributo en particular. Suponga que tiene la entidad o clase `Persona` con los atributos: identificación (*integer*), nombres (*string*), apellidos (*string*), fecha de nacimiento (*date*). Cada uno de estos atributos constituye un “campo” y cada registro de esta tabla (*fila*) representará una entidad de tipo `Persona`. Para acceder a la información almacenada en la tabla de la base de datos, se utiliza la sentencia `SELECT` con algunos parámetros. Por ejemplo, si la tabla se llama `Persona`, entonces, para obtener todos los registros almacenados en la tabla, se define la sentencia:

```
SELECT * FROM persona (o también: SELECT identificación,
nombres, apellidos, fechaNac FROM persona).
```

Para el almacenamiento de registros proponemos el siguiente ejemplo:

```
INSERT INTO persona (identificación, nombres, apellidos, fechaNac) VALUES (4, “Leonel”, “Murcia”, “05/05/1969”).
```

La sentencia anterior es equivalente a:

```
INSERT INTO persona VALUES (4, “Leonel”, “Murcia”, “05/05/1969”).
```

La sentencia `UPDATE` se utiliza para modificar registros, pero se debe tener mucho cuidado con su uso. Es necesario definir criterios de

actualización porque de lo contrario se afectaría la totalidad de registros almacenados en la tabla. Por ejemplo, si se quiere modificar la fecha de nacimiento de un registro en particular, se debe ingresar la siguiente sentencia:

```
UPDATE persona SET fechaNac="05/05/2000" WHERE identificación=4.
```

Ingresar la sentencia anterior sin la cláusula WHERE afectaría la totalidad de los registros almacenados en la tabla "persona". Para borrar registros de una tabla, se utiliza la sentencia DELETE de la siguiente manera:

```
DELETE FROM persona WHERE identificación=4.
```

Con la sentencia DELETE ocurre algo similar a lo que ocurre con la sentencia UPDATE. El programador debe tener cuidado en el uso de estas sentencias.

Java utiliza la API JDBC (*Java Database Connectivity*) para la conexión entre los programas y las bases de datos. La API JDBC se encuentra en el paquete *java.sql* y dentro de ella hay una serie de interfaces que definen la manera de establecer la conexión entre sentencias, la forma de ejecutarlas, entre otros aspectos. Antes de ejecutar un programa será necesario disponer de un conjunto de clases que implementen todas las interfaces necesarias. A este conjunto de clases se lo denomina *driver*. Con JDBC es posible conectar cualquier programa desarrollado en Java con cualquier base de datos, siempre y cuando se tenga el *driver* específico.

A continuación, un breve ejemplo. Se presenta una clase "BDAd-min.java" que contiene un constructor con los datos necesarios para lograr la conexión a la base de datos (en este caso, se trata de una base de datos construida con "PostgreSQL") y una serie de métodos para manipular la información de cada una de las tablas que conformen la BD.

---

```

import java.net.InetAddress;
import java.net.UnknownHostException;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;

public class BDAdmin {
    // conexion a la base de datos
    private static Connection con = null;

    /*****
        nombre      BDAdmin
        descripcion es el constructor de la clase.
        *****/

    /***/

    public BDAdmin() {
        try {
            String driverName = "org.postgresql.
Driver";

            String url = "jdbc:postgresql://lo-
calhost:5432/EJEMPLO";
            String usr = "postgres";
            String cla = "root";
            // Load the JDBC driver
            Class.forName(driverName);

            // Create a connection to the data-
base
            con = DriverManager.getConnec-
tion(url,usr,cla);

```



```

        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Método para verificar la conexión a la BD
    public boolean Conectado() {
        if (con != null ) {
            return true;
        }

        return false;
    }

    public void desconectar(){
        try {
            con.close();
        }

        catch (Exception ex){}
    }

    // Metodo para realizar una consulta a la
    base de datos de cualquier tabla.
    // Parametro de entrada : la cadena con la
    instruccion del SELECT
    // Parametro de salida: devuelve 1 si hay un
    error y 0 si se encuentran datos

    public static ResultSet ejecutarConsulta(S-
    tring consulta) throws Exception{
        Statement st = null;
        ResultSet rs = null;
        st = con.createStatement();
        rs = st.executeQuery( consulta );
        return rs; // Si la consulta se realiza

```

```

correctamente devuelve 0
    } // fin del metodo

    // Devuelve un resulset de registros
    public ResultSet BuscarRegistro(String sql){
        ResultSet registros =null;
        try{
            registros = ejecutarConsulta(sql);
        }catch( Exception e ){
            e.printStackTrace();
        }
        return registros;
    }

    /*****
    *****/
    descripcion permite hacer la insercion
de registros en la bd
parametro una instruccion sql
    *****/
    /
    public void ejecutarUpdate(String consulta)
    throws SQLException {
        Statement st = con.createStatement();
        st.executeUpdate(consulta);
    }

    // Metodo para obtener el nombre del equipo.
    public static String ObtenerEquipo() throws
    UnknownHostException{
        // Obtengo el nombre del equipo y la IP
        InetAddress Address = InetAddress.getLocal-
        Host();
        // Obtengo solo el nombre del equipo

```

```

        String NEquipo = Address.getHostName();

        return NEquipo;
    }
}

```

Ahora se tiene una clase principal que permite realizar una consulta de registros sobre la tabla Persona (comentada anteriormente). Al ejecutar el ejercicio como salida, sólo habrá un mensaje que indicará la existencia de registros:

```

import java.sql.ResultSet;
import java.sql.SQLException;

public class Prueba {

    public static void main(String[] args) throws
SQLException {
        // TODO Auto-generated method stub

        BDAdmin bd = new BDAdmin();
        if (bd.Conectado()) {
            String sentencia="SELECT * FROM per-
sona";

            ResultSet rsUsuario = bd.BuscarRe-
gistro(sentencia);

            if (rsUsuario.next()){
                System.out.println("Si hay re-
gistros");
            }
        } else {
            System.out.println("No conectado");
        }
    }
}

```

---

## Prerrequisitos

- Tener conocimiento sobre la gestión de excepciones mediante el bloque *Try-Catch*.
- Haber adquirido habilidades en el manejo de colecciones.
- Realizar lecturas adicionales sobre el estándar SQL.
- Diseñar un modelo de bases de datos (sugerencia: “MySQL”, “PostgreSQL”). Deberá tener a su disposición el *driver* respectivo.

## Ejercicios

1. Con base en el ejemplo de la parte introductoria modifique la clase principal para mostrar los registros almacenados en la tabla “persona”<sup>4</sup>.
2. Tome el ejemplo modelo y modifique nuevamente la clase principal para que, dados un número de identificación y un nombre, se modifique o actualice la información del registro de acuerdo con el número de identificación ingresado por el usuario. Si el registro no existe, deberá mostrarse un mensaje que indique que no existe.
3. Desarrolle un ejercicio similar al anterior, pero ahora borre un registro. El usuario ingresará un número de identificación y, si se encuentra almacenado en la tabla, se deberá borrar. De lo contrario, se deberá mostrar un mensaje al usuario que le indique que no existe el registro.
4. Investigue el motivo por el cual se podrían generar las excepciones: “SQLException y SQLWarning”.
5. Investigue sobre la sentencia “PreparedStatement”.

---

<sup>4</sup> Previamente debe haber creado la BD con la tabla “persona” y almacenar algunos registros.

- 
6. ¿Qué diferencias existen entre las sentencias: “executeQuery”, “executeUpdate” y “execute”?
  7. La gestión de transacciones hace uso de los métodos `commit` y `rollback`. Investigue sobre estos métodos y desarrolle los ejercicios 2 y 3 de ésta práctica aplicando estos métodos.

## Referencias bibliográficas

- Deitel P, y Deitel, H. (2012). Accessing Databases with JDBC. *Java How to Program* (9 ed.) (pp.1171-1234). (s.l.): Prentice Hall.
- Horstmann, C. (2010). Relational Databases. *Big-Java* (pp.865-904) (s.l.) Wiley John Wiley & Sons, Inc.

# Introducción a Java: Guía de actividades prácticas

Fue editado y publicado por la  
Editorial Universidad El Bosque. Julio de 2016  
Bogotá D. C., Colombia

Los estudiantes de carreras universitarias relacionadas con la informática, como ingeniería de sistemas o ingeniería electrónica, a menudo suelen encontrarse con dificultades en temas como el manejo de Java, claves en el desarrollo de sus habilidades como programadores. Sin embargo, al recurrir a Internet o a otros libros para clarificar sus dificultades, es común que se enfrenten con más teoría o con tutoriales que no logran resolver adecuadamente sus dudas sobre la programación orientada a objetos. Por esta razón, los autores de este libro han querido plasmar una serie de actividades prácticas, desarrolladas de manera secuencial, que buscan ofrecer a los estudiantes de ingeniería los elementos indispensables para el aprendizaje del lenguaje de programación Java.

ISBN: 978-958-739-076-6



9 789587 390766



UNIVERSIDAD EL BOSQUE

EDITORIAL