

■ BUENAS  
PRÁCTICAS  
DE PROGRAMACIÓN  
ORIENTADA A OBJETOS EN  
**JAVA**

LUIS EDUARDO BAQUERO REY

MIGUEL ARMANDO HERNÁNDEZ BEJARANO



**LOS LIBERTADORES**  
FUNDACIÓN UNIVERSITARIA



**LOS LIBERTADORES**  
FUNDACIÓN UNIVERSITARIA

Luis Eduardo Baquero Rey, Miguel Armando Hernández Bejarano.  
 Buenas prácticas de programación orientada a objetos en Java /  
 Bogotá: Fundación Universitaria Los Libertadores. Facultad de Ingeniería y Ciencias  
 Básicas, 2017.  
 350 páginas. ISBN: 978-958-9146-68-2

1. Programación orientada a objetos (Computadores).
2. Java (Lenguaje de programación de computadores). I. Título. II. Autores.

005.117

B222b

Fundación Universitaria Los Libertadores. CRAI-Biblioteca "Hernando Santos Castillo"

COLECCIÓN: Ingeniería

AUTORES

© Luis Eduardo Baquero Rey

© Miguel Hernández Bejarano

Primera edición

Bogotá D.C., junio 2016

ISBN: 978-958-9146-68-2 (impreso)

COORDINACIÓN EDITORIAL

Diego Martínez Cárdenas

DISEÑO Y DIAGRAMACIÓN

Amarillo Magenta Estudio Gráfico

IMPRESIÓN

XXXXXXXXXX

Esta obra cumplió con Depósito Legal de acuerdo con el Decreto 460 de 1995.  
 Se prohíbe cualquier tipo de reproducción parcial o completa de este contenido sin  
 autorización expresa de la Editorial de la Fundación Universitaria Los Libertadores.

Prólogo	10
---------	----

## 1 ■ Fundamentos de la Programación Orientada a Objetos 15

1.1 Temática a desarrollar	17
1.2 Introducción	17
1.3 Programación Orientada a Objetos	18
1.4 Principios de la Programación Orientada a Objetos	18
1.5 Ventajas del uso de la programación orientada a objetos	19
1.6 Desventajas del uso de la programación orientada a objetos	20
1.7 Lenguajes de programación orientada a objetos	20
1.8 Clases y objetos	20
1.9 Visibilidad en atributos y métodos	28
1.10 Paquetes	28
1.11 Lecturas recomendadas	29
1.12 Preguntas revisión de conceptos	30
1.13 Ejercicios	30
1.14 Referencias bibliográficas	31

## 2 ■ Componentes de un Programa 32

2.1 Temática a desarrollar	35
2.2 Introducción	35
2.3 Características de un programa Java	36
2.4 Estructura de un programa en Java	36
2.5 Presentación del programa	39
2.6 Construcción de un programa en Java	40
2.7 Palabras reservadas	41
2.8 Los identificadores	41
2.9 Tipos de datos	42
2.10 Variables	44
2.11 Constantes	44
2.12 Comentarios	45

2.13 Definición de variables	46
2.14 Definición de constantes	46
2.15 Operadores aritméticos	47
2.16 Los separadores	48
2.17 Lecturas recomendadas	49
2.18 Preguntas de revisión de conceptos	49
2.19 Ejercicios	49
2.20 Referencias bibliográficas	50

## 3

### ■ Herramientas para el desarrollo de la Programación Orientada a Objetos 51

3.1 Temática a desarrollar	53
3.2 Introducción	53
3.3 Entornos Integrados de Desarrollo (IDE)	54
3.4 Herramientas de modelamiento	58
3.5 Java como lenguaje de programación	59
3.6 Lecturas recomendadas	61
3.7 Preguntas de revisión de conceptos	62
3.8 Ejercicios	62
3.9 Referencias bibliográficas	63

## 4

### ■ Métodos constructores, set y get 59

4.1 Temática a desarrollar	67
4.2 Introducción	67
4.3 Métodos	68
4.4 Modificadores de acceso	71
4.5 Métodos set y get	78
4.6 Métodos de instancias	82
4.7 Estructura de un método	83
4.8 Prueba de escritorio	85
4.9 Métodos que no retornan valor	87
4.10 Requerimientos	87

4.11 Un desarrollo completo	89
4.12 Entrada y salida de datos	96
4.13 Entrada y salida estándar	97
4.14 Lecturas recomendadas	108
4.15 Preguntas de revisión de conceptos	108
4.16 Ejercicios	109
4.17 Referencias bibliográficas	110

## 5 ■ Condicionales 111

5.1 Temática a desarrollar	113
5.2 Introducción	113
5.3 Estructuras de control condicionales	114
5.4 Operadores relacionales	115
5.5 Operadores lógicos	116
5.6 Tipos de condicionales	117
5.7 Selección múltiple	135
5.8 Operador condicional (?)	139
5.9 Preguntas de revisión de conceptos	143
5.10 Lecturas recomendadas	143
5.11 Ejercicios	143
5.12 Referencias bibliográficas	149

## 6 ■ Cadenas 151

6.1 Temática a desarrollar	153
6.2 Introducción	153
6.3 Clase String	154
6.4 Obtener cadenas desde las primitivas	160
6.5 Obtener primitivas desde las cadenas	161
6.6 Lecturas recomendadas	171
6.7 Preguntas de revisión de conceptos	171
6.8 Ejercicios	171
6.9 Referencias bibliográficas	173

## 7

### Ciclos

175

7.1 Temática a desarrollar	177
7.2 Introducción	177
7.3 Estructuras de control repetitivas o ciclos	178
7.4 Comparación de los ciclos while, do-while, for	194
7.5 Ciclo for each (for mejorado)	196
7.6 Uso de los ciclos repetitivos	197
7.7 Lecturas recomendadas	197
7.8 Preguntas y ejercicios de revisión de conceptos	198
7.9 Ejercicios	198
7.10 Referencias bibliográficas	199

## 8

### Relaciones entre Clases

201

8.1 Temática a desarrollar	203
8.2 Introducción	203
8.3 Elementos entre las relaciones de clases	204
8.4 La relación de Especialización/Generalización	210
8.5 Dependencia	211
8.6 Lecturas recomendadas	212
8.7 Preguntas de evaluación	212
8.8 Ejercicios	213
8.9 Referencias bibliográficas	217

## 9

### Abstracción y Encapsulamiento

219

9.1 Temática a desarrollar	221
9.2 Introducción	221
9.3 Abstracción	221
9.4 Encapsulamiento	223
9.5 Lecturas recomendadas	227
9.6 Preguntas de revisión de conceptos	227
9.7 Ejercicios	227
9.8 Referencias bibliográficas	229

# 10

## Herencia

231

10.1 Temática a desarrollar	233
10.2 Introducción	233
10.3 Herencia	234
10.4 Representación de la herencia	239
10.5 Sentencia super	244
10.6 Lecturas recomendadas	247
10.7 Preguntas de revisión de conceptos	248
10.8 Ejercicios	248
10.9 Referencias bibliográficas	251

# 11

## Polimorfismo

253

11.1 Temática a desarrollar	255
11.2 Introducción	255
11.3 Polimorfismo	256
11.4 Sobrecarga de métodos	257
11.5 Sobrecarga de constructores	260
11.6 Sobreescritura de métodos	262
11.7 Enlace dinámico	269
11.8 Lecturas recomendadas	272
11.9 Preguntas de revisión de conceptos	272
11.10 Ejercicios	272
11.11 Referencias bibliográficas	275

# 12

## Interface y Clases Abstractas

277

12.1 Temática a desarrollar	279
12.2 Introducción	279
12.3 Interface	280
12.4 Clases abstractas	284
12.5 Preguntas de revisión de conceptos	286
12.6 Ejercicios	286
12.7 Referencias bibliográficas	290



## 13

### Modelado Orientado a Objetos y Aplicaciones de Software

291

13.1 Temática a desarrollar	293
13.2 Introducción	293
13.3 Análisis orientado a objetos	294
13.4 Escenarios	295
13.5 Prototipo	295
13.6 Diagramas de casos de uso	296
13.7 Diseño orientado a objetos	305
13.8 Diagramas UML (Unified Modeling Language)	309
13.9 Aplicaciones de software	311
13.10 Lecturas recomendadas	331
13.11 Preguntas de revisión de conceptos	331
13.12 Ejercicios	331
13.14 Referencias bibliográficas	333

## 14

### Documentación

335

14.1 Temática a desarrollar	337
14.2 Introducción	337
14.3 Javadoc	337
14.4 Javadoc en los IDE	343
14.5 API de Java	346
14.6 Lecturas recomendadas	347
14.7 Preguntas de revisión de conceptos	348
14.8 Ejercicios	348
14.9 Referencias bibliográficas	348

## LISTA DE FIGURAS

**Figura 1.** Concepto de Programación Orientada a Objetos

**Figura 2.** Concepto de Clase

**Figura 3.** Partes del diagrama de clase

**Figura 4.** Concepto de objeto

**Figura 5.** Ejemplos del objeto reloj

**Figura 6.** Diagrama de paquete

**Figura 7.** Diagrama de paquetes con dos clases comunes.

**Figura 8.** Concepto de tipos de datos.

**Figura 9.** Pantalla de carga de Eclipse LUNA

**Figura 10.** Pantalla inicio en Netbeans

**Figura 11.** Pantalla inicio de GreeFoot

**Figura 12.** Pantalla IntelliJ IDEA

**Figura 13.** Concepto de Java

**Figura 14.** Concepto de plataformas de java

**Figura 15.** Concepto de Métodos

**Figura 16.** Concepto de Constructor

**Figura 17.** Concepto de métodos set y get

**Figura 18.** Concepto de las estructuras condicionales

**Figura 19.** Concepto de la clase String

**Figura 20.** Concepto de los ciclos

**Figura 21.** Concepto de relaciones en clases

**Figura 22.** Concepto de abstracción

**Figura 23.** Concepto de encapsulamiento

**Figura 24.** Concepto de Herencia

**Figura 25.** Concepto de polimorfismo

**Figura 26.** Concepto de aplicaciones de software

**Figura 27.** Documentacion en Bluej

**Figura 28.** Generacion Javadoc en eclipse

**Figura 29.** Documentación Javadoc en eclipse

**Figura 30.** Generacion Javadoc en Netbeans

**Figura 31.** Documentación Javadoc en Netbeans

**Figura 32.** API Java 8

## LISTA DE TABLAS

**Tabla 1.** Ejemplos de objetos asociados a su correspondiente clase

**Tabla 2.** Ejemplo de objeto

**Tabla 3.** Ejemplo estructura interna de un Objeto

**Tabla 4.** Palabras reservadas en java

**Tabla 5.** Tipos de datos de java

**Tabla 6.** Operadores aritméticos

**Tabla 7.** Operadores relacionales

**Tabla 8.** Operadores lógicos

**Tabla 9.** Resultado operadores lógicos

**Tabla 10.** Apoyo implementación del método

**Tabla 11.** Apoyo parte de análisis

**Tabla 12.** Ejemplo de cadenas

**Tabla 13.** Metodos básicos

**Tabla 14.** Tipos de cardinalidad

**Tabla 15.** Comparación entre abstracción y encapsulamiento

## PRÓLOGO

En el quehacer diario de la docencia, y más concretamente en el área de la programación, surgen muchos temas, talleres y ejercicios a trabajar con los estudiantes. En tal sentido, este libro es el resultado de varios años de experiencias donde se retroalimentan carencias y éxitos en busca de buenas prácticas de la Programación Orientada a Objetos (POO), para tal efecto se utiliza Java como lenguaje de programación de alto nivel. Está diseñado para cualquier estudiante que desea hacer inmersión en la programación de computadores, independientemente del área o disciplina del conocimiento, pero eso sí, cambiando el paradigma de la programación estructurada a la de los objetos como en la vida real.

Se tienen en cuenta elementos de buenas prácticas de la programación con la finalidad de hacer fácil la lectura del código, así como su mantenimiento, facilitando la escalabilidad del mismo, la reutilización y la integración de manera homogénea, estandarizando el desarrollo, basado en convenios y reglas sencillas, y aportando higiene en la codificación de los programas fuente.

Con el propósito de lograr lo anterior, esta obra se divide en catorce capítulos, cada uno distribuido de la siguiente manera: relación general de la temática a desarrollar, una introducción al tema central donde se complementa con un Concepto de, el desarrollo de la temática respectiva incluyendo ejemplos, se recomiendan unas lecturas que posibilitan ampliar el tema, unas preguntas de revisión de conceptos, ejercicios propuestos y finalmente se propone una bibliografía complementaria.

En el **capítulo 1** se tratan temas fundamentales de este paradigma, como el de la programación orientada a objetos, los principios de la POO y los conceptos de clase, objeto, atributo y método.

En el **capítulo 2** se estudian los componentes de un programa donde se involucran identificadores, tipos de datos, palabras reservadas, manejo de comentarios, variables, constantes, las expresiones, la estructura de un programa de java, la entrada y la salida de datos y los tipos de operadores.

En el **capítulo 3** se estudian las herramientas necesarias para el desarrollo de la programación orientada a objetos, como los entornos integrados de desarrollo, herramientas de modelamiento y el lenguaje de programación Java.

En el **capítulo 4** se hace referencia a los métodos, qué son los métodos constructores, los métodos set y get y los métodos modificadores.

En el **capítulo 5** se todo lo relacionado con los condicionales, en su orden, las estructuras de control, los condicionales simple, compuesto y anidado, los operadores lógicos, selección múltiple y el operador condicional ?.

En el **capítulo 6** se estudia el tema relacionado con el manejo de cadenas en Java, incluyendo los métodos asociados a la clase String.

El **capítulo 7** hace referencia a los ciclos o instrucciones repetitivas que se pueden trabajar en un programa Java.

En el **capítulo 8** se estudian las diferentes relaciones entre clases, como la cardinalidad, asociación, agregación, composición, generalización y dependencia.

En el **capítulo 9** se involucran los temas de abstracción y encapsulamiento, como mecanismos de la programación orientada a objetos.

En el **capítulo 10** se estudia el concepto de herencia y la terminología asociada a este concepto, como lo que es una superclase, subclase, protected, extends y super.

En el **capítulo 11** el mecanismo de la programación orientada a objetos denominado polimorfismo, incluyendo, polimorfismo de subtipado, sobrecarga de métodos y sobreescritura de métodos.

En el **capítulo 12** se continúa con el concepto y manejo de interface y clases abstractas en Java.

En el **capítulo 13** se estudian los aspectos básicos a tener en cuenta a la hora de desarrollar un proyecto o aplicación de software de comienzo a fin, desde el análisis orientado a objetos, el diseño, el modelamiento y la aplicación de software final.

En el **capítulo 14** por último, se presentan opciones desde el lenguaje de programación Java para la respectiva documentación de los programas y aplicaciones, con herramientas con Javadoc, soporte Javadoc para los IDE y la API de Java 8.

El estudiante interesado en el tema, podría orientarlo también a cualquier otro lenguaje de programación de alto nivel, diseñado para trabajar con el paradigma de la programación orientada a objetos.





# CAPÍTULO 1

**Fundamentos de la Programación**

**ORIENTADA A OBJETOS**





## 1.1 TEMÁTICA A DESARROLLAR

- » Programación orientada a objetos
- » Principios de la POO
- » Clase
- » Objeto
- » Atributo
- » Método

## 1.2 INTRODUCCIÓN

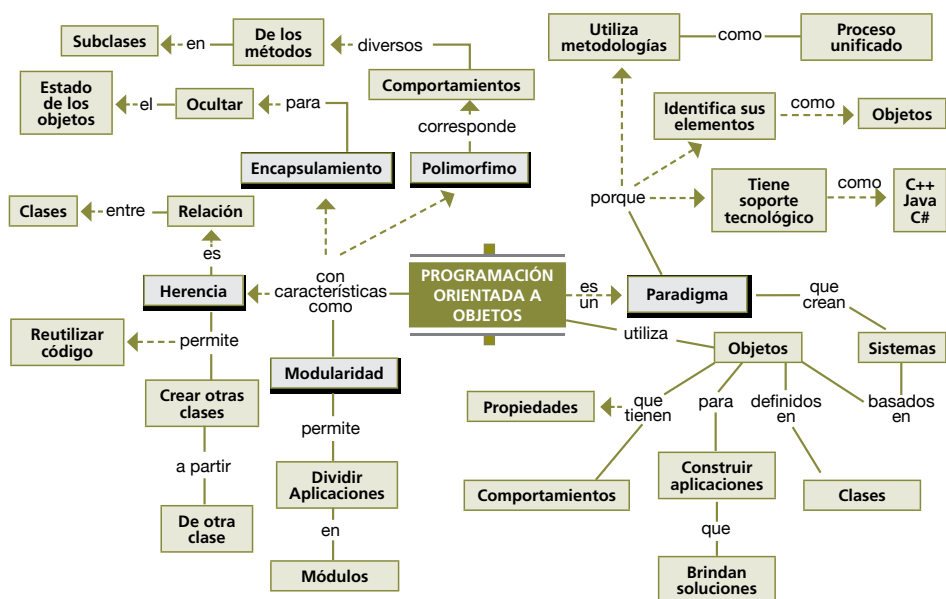
La concepción del mundo en términos de Programación Orientada a Objetos (POO) está integrada por un conjunto de entidades u objetos relacionados entre sí, estos pueden ser personas, árboles, automóviles, casas, teléfonos y computadores; en general, todo elemento que tiene atributos y que pueden ser palpables a través de los sentidos.

El ser humano tiene una apreciación de su ambiente en términos de objetos, por lo cual le resulta simple pensar de la misma manera cuando diseña un modelo para la solución de un problema, puesto que se tiene el concepto o pensamiento de objetos de manera intrínseca.

La Programación Orientada a Objetos (Object Oriented Programming), tiene como ejes centrales las clases y los objetos, donde una clase corresponde a la agrupación de datos (atributos), acciones (métodos). Una clase describe un conjunto de objetos con propiedades y comportamientos similares. La figura 1 muestra, mediante mapa mental este concepto y todo lo que lo rodea.

El buen uso de este paradigma de programación se convierte en una buena práctica que arrojaría como resultados la construcción de programas de calidad, facilita su mantenimiento y la reutilización de programas, entre otros aspectos. El tomar objetos de un problema del mundo real y extraer sus características y comportamientos, y representarlos formalmente en diagramas UML, mediante los diagramas de clases y diagramas de objetos, son de las primeras actividades inmersas en este estilo de programación.

Figura 1. Concepto de Programación Orientada a Objetos.



### 1.3 PROGRAMACIÓN ORIENTADA A OBJETOS

La Programación Orientada a Objetos (POO) es un paradigma o modelo de programación, esto indica que no es un lenguaje de programación específico, o una tecnología, sino una forma de programar donde lo que se intenta es llevar o modelar el mundo real a un conjunto de entidades u objetos que están relacionados y se comunican entre ellos como elementos constitutivos del software, dando solución a un problema en términos de programación.

En síntesis, el elemento básico de este paradigma es el objeto, el cual contiene toda la información necesaria que se abstrae del mundo del problema, los datos que describen su estado y las operaciones que pueden modificar dicho estado, determinando las capacidades del objeto.

### 1.4 PRINCIPIOS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

**Abstracción:** es la capacidad de determinar los detalles fundamentales de un objeto mientras se ignora el entorno, por ejemplo la información de un cliente en una factura, requiere el documento, el nombre, la dirección y el teléfono.

**Encapsulamiento:** es la capacidad de agrupar en una sola unidad datos y métodos. Por ejemplo para calcular el área de un triángulo se debe tener la base y la altura; o cuando se ve televisión no hay preocupación del modo como este funciona, lo único que se realiza es manipular el control y disfrutar del programa, película, entre otros ejemplos.

**Herencia:** proceso en el que un objeto adquiere las propiedades de otro a partir de una clase base, de la cual se heredan todas sus propiedades, métodos y eventos; estos, a su vez, pueden o no ser implementados y/o modificados, por ejemplo la herencia de bienes que se transmite de padre a hijo, pero el padre tiene la potestad de desheredar.

**Polimorfismo:** es una propiedad que permite enviar el mismo mensaje a objetos de diferentes clases de forma que cada uno de ellos responde al mismo mensaje de diferente modo; por ejemplo, el acto de comer para proveer nutrientes al organismo, es diferente en los siguiente animales el león (carnívoro), el canario (alpiste) y el perro (todo tipo de alimento).

**Modularidad:** es la propiedad que permite dividir una aplicación de software en unidades o partes más pequeñas, denominados módulos, donde cada una de ellas puede ser independiente de la aplicación y de los restantes módulos. Por ejemplo, el software institucional de una empresa que puede ser dividido en subprogramas como contabilidad, presupuesto, facturación entre otras.

## 1.5 VENTAJAS DEL USO DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

- » Simula el sistema del mundo real.
- » Facilita el mantenimiento del software.
- » Facilita el trabajo en equipo.
- » Modela proceso de la realidad.
- » Genera independencia e interoperabilidad de la tecnología.
- » Incrementa la reutilización del software.

## 1.6 DESVENTAJAS DEL USO DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

- » Complejidad para adaptarse.
- » Mayor cantidad de código (aunque se tiene la posibilidad de volver a ser reutilizable).

## 1.7 LENGUAJES DE PROGRAMACIÓN ORIENTADA A OBJETOS

Permite el desarrollo de aplicativos de software, que interactúan con los programas como conjuntos de objetos que se ayudan entre ellos para realizar acciones.

### Ejemplos

- » C++
- » ADA
- » Java
- » SmallTalk
- » C#
- » Php versión 5.0 en adelante

## 1.8 CLASES Y OBJETOS

Las palabras “clase” y “objeto” son utilizadas en la programación orientada a objetos, la cual planea simular u organizar datos en conjuntos modulares de elementos de información del mundo real.

### 1.8.1 Clases

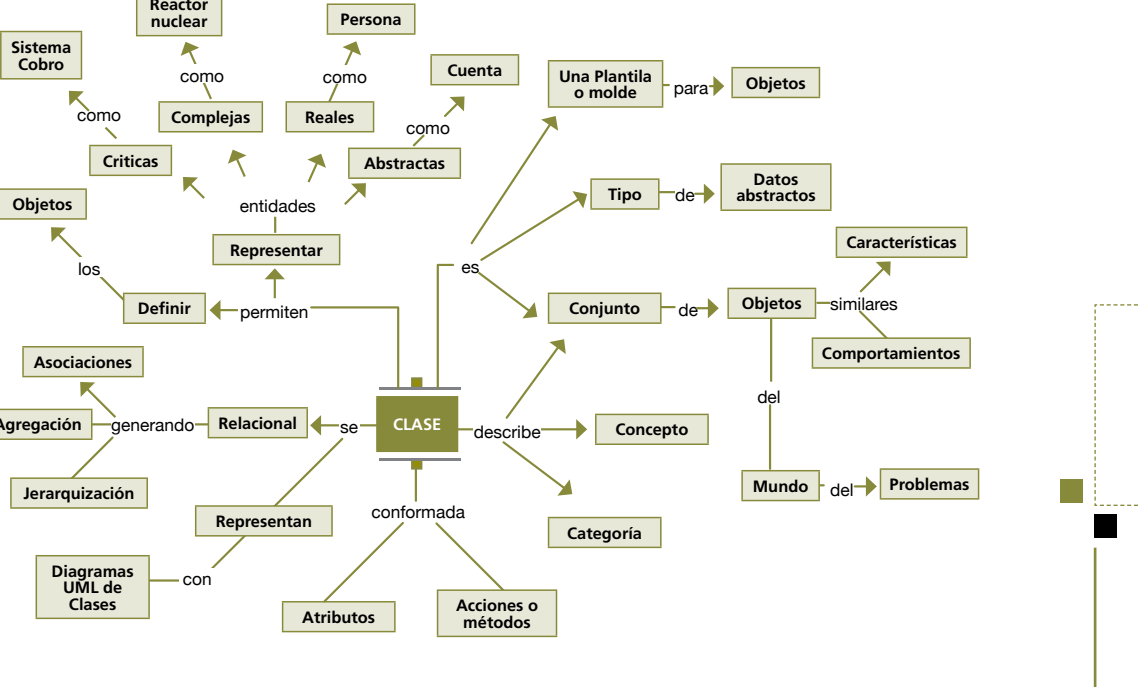
Asumida como un molde o plantilla empleada para crear objetos, lo que permite considerar que las clases agrupan a un conjunto de objetos similares.

**Tabla 1.** Ejemplos de objetos asociados a su correspondiente clase

\_\_\_\_\_

---

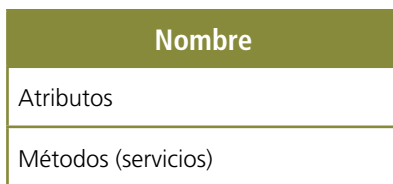
**Figura 2.** Concepto de Programación Orientada a Objetos.



### 1.8.1.1 Representación de las clases

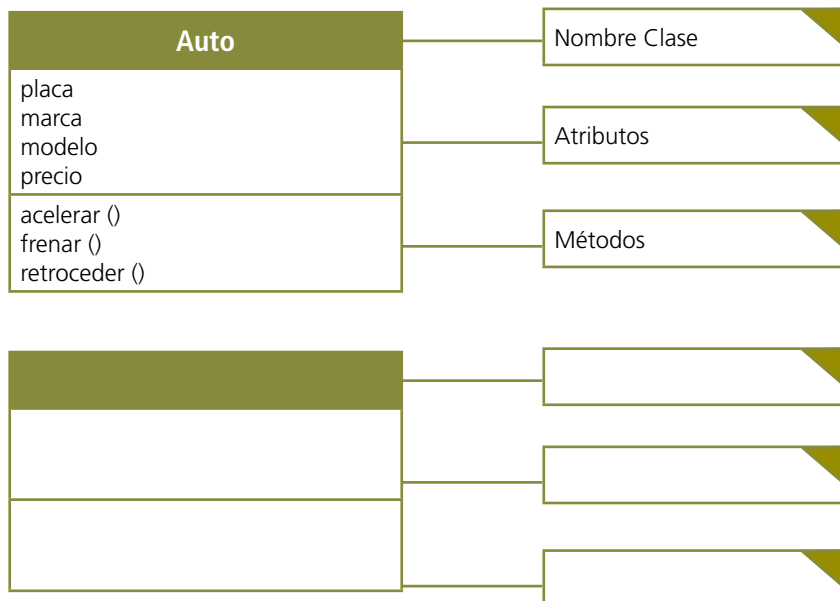
La clase encapsula la información de uno o varios objetos a través de la cual se modela el entorno en estudio. Se representa haciendo uso de los diagramas de clases de UML, conformado por un rectángulo que tiene tres divisiones, como se muestra en la figura 3.

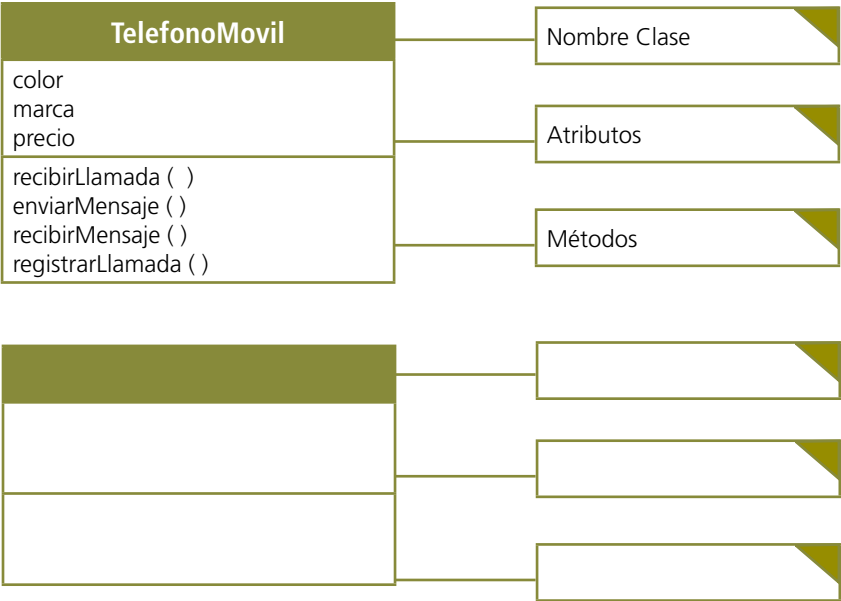
**Figura 3.** Partes del diagrama.



UML (Unified Modeling Language) es un lenguaje que permite modelar, construir y documentar los elementos que forman un sistema software orientado a objetos.

**Ejemplos:** los siguientes diagramas de clases se asocian la información correspondiente de acuerdo con el número del cuadrante, así en el primer cuadrante se registra el nombre de la clase, en el segundo cuadrante los nombres de los atributos y en el tercer cuadrante los nombres de los métodos, comportamientos o responsabilidades que realizan.





Los ejemplos anteriores representan diagramas de clases con sus correspondientes atributos y metodos, los rectángulos de los lados unidos con líneas punteadas corresponden a la documentación.

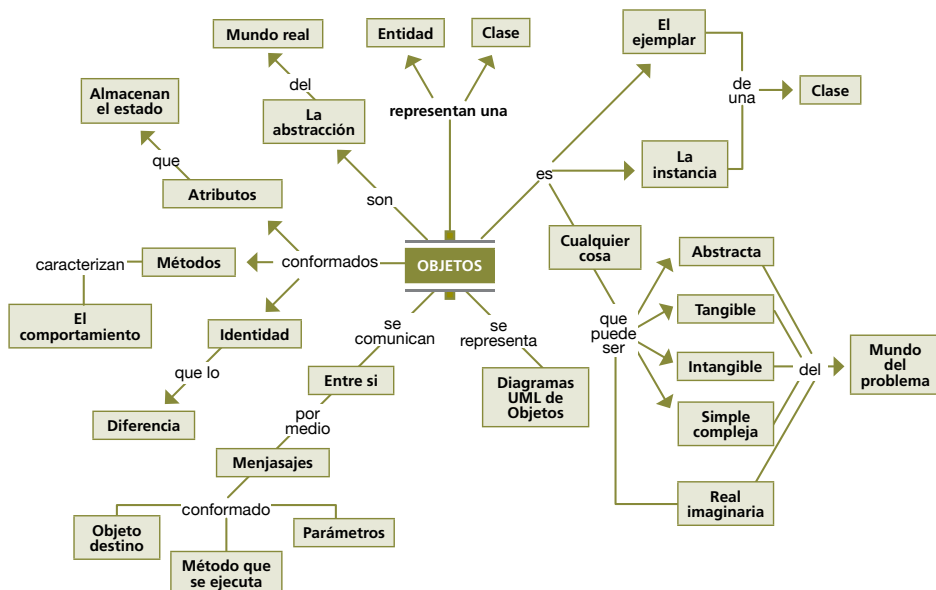
Durante del desarrollo de la temática incluyendo los capítulos siguientes, se va ampliando la información correspondiente al tema de las clases.

1.8.2 Objeto

Es cualquier cosa real (tangible o intangible) del mundo que nos rodea; por ejemplo, un auto, una casa, un árbol, un reloj, una estrella, cuenta de ahorro. Uno objeto se puede considerar como una entidad compleja provista de propiedades o características y de comportamientos o estados. La figura 4 detalla el concepto de objeto de una manera más somera.



Figura 4. Concepto de Objeto.



A un objeto lo identifican las siguientes características:

## Estado

Conformado por el conjunto de propiedades o atributos de un objeto correspondiente a los valores que pueden tomar cada uno de esos atributos.

## Ejemplo:

El objeto cliente tiene las siguientes características: documento, nombres, apellidos, dirección y teléfono.

Tabla 2. Ejemplo de objeto.

ATRIBUTO	ESTADO
Documento:	10.265.254
Nombres:	Luis Eduardo
Apellidos:	Baquero Rey
Teléfono:	300 000 00 00

Comportamiento

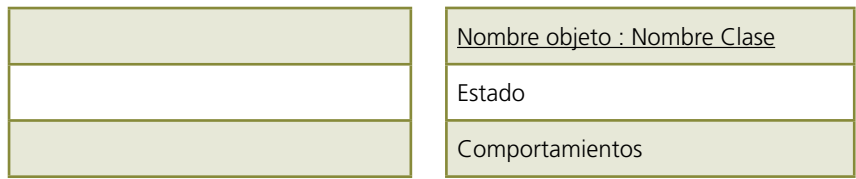
De un objeto está relacionado con la funcionalidad y determina las operaciones que este puede realizar, responder o ejecutar alguna acción ante mensajes enviados por otros objetos, por ejemplo, actualizar el teléfono de un cliente, realizar la suma de dos números.

Identidad

Corresponde a la propiedad de un objeto que le distingue de todos los demás, como es el caso del ejemplo del cliente Luis Eduardo Baquero Rey, que tiene un número de documento que lo identifica, una dirección y un teléfono.

1.8.2.1 Representación gráfica de los objetos

Los objetos se pueden representar con diagramas UML de Objeto, que es un rectángulo con tres divisiones.



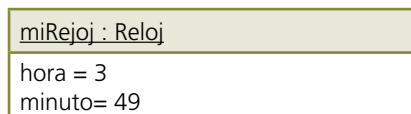
Se puede confundir los términos clase y objetos; en general, una clase es una representación abstracta de algo, mientras que un objeto es un represtación de ese algo conformado por la clase.

**Ejemplo.** Se tienen cuatro objetos correspondientes a relojes (de pared, arena, digital de pulsera). La clase Reloj tiene las características que agrupa a esa colección de objetos.

Figura 5. Ejemplos del objeto reloj.

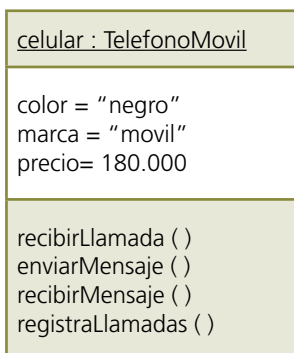
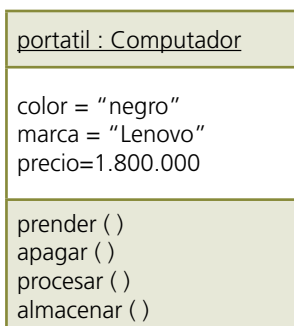


A continuación se modela la clase Reloj y un objeto reloj; mediante los diagramas correspondientes.



Para el ejemplo, un objeto miRejoj es una entidad que tiene un conjunto de propiedades o atributos, como el color, marca, forma, estilo, precio y dentro de los comportamientos están el dar la horas, minutos, adelantarse, atrasarse.

Se tiene dos ejemplos donde se modelan los objetos (computador portátil y teléfono celular) mediante el correspondiente diagrama de objetos.



Otros ejemplos de objetos son:

**Objetos físicos**

- » Barcos, aviones, perros, computadores.

**Objetos de interfaces gráficos de usuarios:**

- » Etiquetas, cajas de texto, botones

**Objetos intangibles:**

- » Cuenta de ahorros, cuenta corriente, CDT

**Ejemplos de no objetos:**

- » El amor, la felicidad, la nostalgia.

**1.8.2.2 Estructura interna de un objeto**

Los atributos de una clase definen el estado del objeto, que son los valores de los datos del objeto concreto y que lo diferencian de los demás.

**Ejemplos:**

**Tabla 3.** Ejemplo estructura interna de un Objeto.

OBJETO	ESTADO
estudiante	edad = 20 estatura 1.80
celular	color = negro marca = Nokia
computador	marca = Lenovo color = negro

Los métodos de una clase, definen el comportamiento del objeto y corresponde a las operaciones que pueden realizarse sobre los objetos de una clase.

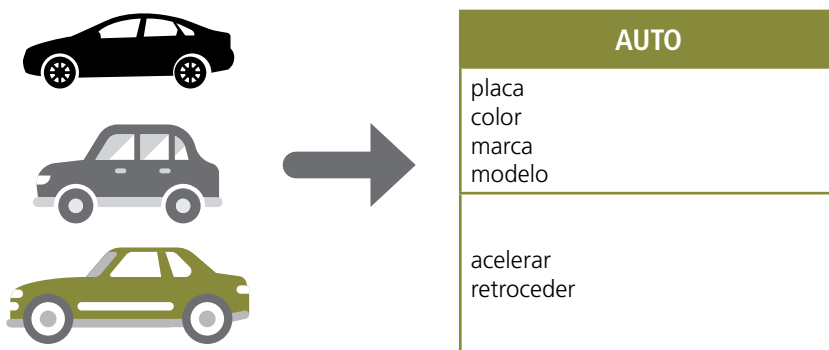
## 1.9 VISIBILIDAD EN ATRIBUTOS Y MÉTODOS

La Programación Orientada a Objetos ayuda a incrementar factores de calidad como: reutilización, facilidad del uso de las clases, entre otros. Por su parte, la comunicación y visibilidad tanto para los atributos de una clase como para los métodos puede ser: public, private, protected (público, privado, protegido).

**Público (public).** Representado por el signo más (+) indica que el método o atributo está visible dentro o fuera de la clase, es decir, es accesible desde todos lados.

**Privado (private).** Representado por el signo menos (-) indica que el atributo o método solamente será accesible dentro de la misma clase (sólo sus métodos lo pueden acceder).

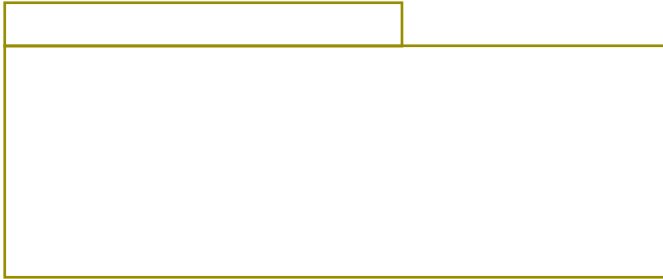
**Protegido (protected).** Representado por el símbolo numeral (#) indica que el atributo o método no es accesible desde fuera de la clase, pero se podrá acceder por métodos de la clase, además de las subclases que se deriven como es el caso de la herencia.



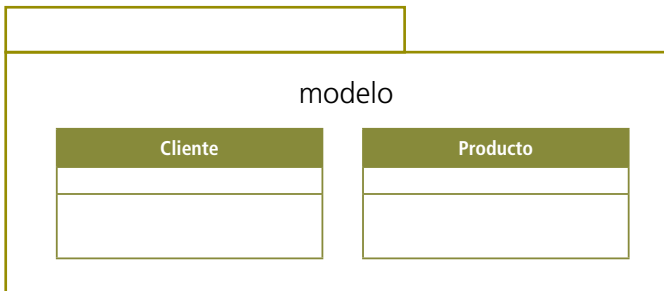
Los tres vehículos son objetos que se pueden agrupar en la clase Auto.

## 1.10 PAQUETES

Permiten agrupar los elementos de comportamientos similares que se modelan con UML, La figura 6 representa un diagrama de paquete, similar a una carpeta o folder.

**Figura 6.** Diagrama de paquete.

La figura 7 ilustra un diagrama de paquetes, conformado por dos clases comunes al modelo de un sistema.

**Figura 7.** Diagrama de paquetes con dos clases comunes.

## 1.11 LECTURAS RECOMENDADAS

- » Qué es UML.
- » Historia de UML.
- » Paradigma de Programación Orientada a Objetos.
- » Atributos, métodos y constructores.
- » Principios de la Programación Orientada a Objetos.

## 1.12 PREGUNTAS REVISIÓN DE CONCEPTOS

- » ¿Dónde utilizar UML?
- » Diferencias entre clases y objetos.
- » ¿Cuáles son los lenguajes que permiten la implementación del concepto de objetos y actualmente son fuente para el desarrollo de proyectos de software?

## 1.13 EJERCICIOS

1. Construir la clase llamada Proveedor, que está conformada por siguientes atributos:
    - » NIT.
    - » Nombre de la empresa o razón social.
    - » Nombre del representante.
    - » Primer apellido del representante.
    - » Segundo apellido del representante.
    - » Dirección empresa.
    - » Teléfono.
    - » E-mail.
- 
2. Crear una clase denominada Cubo, conformada por los siguientes atributos: ancho, alto y largo.

3. Completar la información de la siguiente tabla.

CLASE	OBJETOS
Ropa	camisa El monarca, talla 38; pantalón negro, talla 32
Mamífero	

4. Completar la información de la siguiente tabla:

CLASE	OBJETOS
Mueble	
Gato	
Docente	

## 1.14 REFERENCIAS BIBLIOGRÁFICAS

Aguilar, L. J. (2004). *Fundamentos de programación algoritmos y estructura de datos*. Tercera Edición. Ciudad: México: McGraw Hill.

Booch, G. (1996). *Análisis y diseño orientado a objetos con aplicaciones*. Ciudad: México Addison Wesley.

Deitel y Deitel. (2012). *Cómo programar en Java*. Ciudad: México Editorial Pearson (Prentice Hall).

Fowler, M., y Scott, K. UML. (1999). *Gota a gota*. Ciudad: México Editorial Pearson.

Villalobos, J., y Casallas, R. (2006). *Fundamentos de programación, aprendizaje activo basado en casos*. Ciudad: México Editorial Pearson.







# CAPÍTULO 2

**Componentes de un**

**PROGRAMA**



## 2.1 TEMÁTICA A DESARROLLAR

- » Identificadores.
- » Tipos de datos.
- » Palabras reservadas.
- » Comentarios.
- » Variables.
- » Constantes.
- » Expresiones.
- » Estructura de un programa en Java.
- » Entrada y salida de datos.
- » Operadores aritméticos.

## 2.2 INTRODUCCIÓN

En este capítulo se incluyen definiciones y conceptos relacionados con los tipos de datos en Java; se abordarán temáticas referentes a la definición de variables, su inicialización, los operadores aritméticos y las palabras reservadas.

Se indican los pasos para la creación de un programa o un proyecto, como la digitación, la compilación y la ejecución de un programa en Java, conceptos básicos de importancia dentro del desarrollo de programas y aplicaciones, a utilizarse en la programación.

Como buenas prácticas se recomienda tener en cuenta la manera de nombrar las clases, los atributos o variables, los métodos y las constantes.

## 2.3 CARACTERÍSTICAS DE UN PROGRAMA JAVA

- » El nombre de la clase debe ser igual al nombre del archivo.
- » El nombre del archivo tiene extensión java.
- » Los comentarios se colocan entre `/*` y `*/` o se inician con `//`.
- » La clase se encierra entre `{ y }`.
- » Los métodos también se encierran entre `{ y }`, como el método `main`.
- » Las sentencias (acciones) se terminan con punto y coma.
- » Los argumentos de los métodos se colocan entre paréntesis redondo ( `( )` ).
- » Se usa la indentación, sangrado o tabulación para obtener un código más legible empleando cuatro espacios como unidad de tabulación.
- » Se debe evitar las líneas de más de 80 caracteres.

## 2.4 ESTRUCTURA DE UN PROGRAMA EN JAVA

Java es un lenguaje orientado a objetos. Así que un programa en este lenguaje es una colección de clases, donde cada clase es en un archivo diferente y el nombre del archivo debe ser el mismo que el de la clase.

El programa más sencillo utiliza una clase e incluye, el método **`main()`** (es el método de inicio para la ejecución de las aplicaciones en java).

Un programa Java consta de una colección de archivos o unidades de compilación, que se describen en los apartados siguientes.

### 2.4.1 Declaración de importaciones (**`import`**)

Nombra un elemento de otro paquete que se utilizará en las declaraciones posteriores de interfaces o clases. Se puede utilizar un asterisco para incluir todos los elementos de un paquete.

**Ejemplos:**

- » `import nombrePaquete.*;`
- » `import nombrePaquete.NombreClase;`
- » `import java.util.Scanner;`

**2.4.2 Definición de clases**

Una clase consta de una declaración y un cuerpo. El cuerpo contiene campos de datos y declaraciones de métodos.

**Sintaxis:**

```
public class <NombreClase> {
    // Declaración de atributos
    <tipo> <variable> ;
    // Declaración de métodos
    <tipo> <nombreMétodo> ( <argumentos> )
    { ... }
}
```

**Ejemplo**

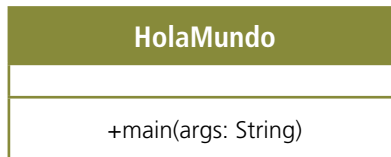
```
public class ClaseUno {
    // atributos y declaraciones de métodos
}
```

```
public class Fecha {
    // Atributos
    int dia ;
    int mes ;
    int año ;
    // Métodos
    void asignarDia ( int d ) {...}
    String darFormato ( ) {...}
}
```

## Ejemplo

Programa que imprime un mensaje **Hola Mundo, como estas**, en pantalla.

**Diagrama de clases** del programa Hola Mundo.



Un diagrama de clases en UML permite especificar la estructura de un sistema en función de las clases que lo componen.

```

public class HolaMundo {
    /** este es el método principal */
    public static void main(String[ ] args) {
        System.out.println("¿Hola Mundo, como estas?");
    }
}
  
```

**public** Es una palabra reservada que indica que la clase o el método pueden ser usados por cualquier otra clase.

**class** Es la palabra que indica que a continuación de ellas se va a declarar una clase.

Toda clase en Java que sea una aplicación debe tener un método `main()`, ya que es la entrada para la ejecución del programa.

`{ }` Permiten delimitar el alcance de la clase, determinando donde inicia y donde finaliza.

**public class HolaMundo** se define la clase `HolaMundo`, que corresponde al nombre del programa, los bloques del programa van entre llaves `{ }` que abren y cierran el bloque del programa ya sea el de la clase o el del método.

**public static void main(String [ ]args)** Es el método principal o punto de ejecución de las aplicaciones en java, esta demarcado por una llave que abre el método y otra para cerrar.

- » `public`: el método se puede usar desde fuera.
- » `static`: el método pertenece a la clase (no a los objetos de la clase).
- » `void`: indica que el método no retorna nada.
- » `String[ ] args`: es el argumento, datos que se pasan a la operación.

**`System.out.println("Hola Mundo");`** imprime el mensaje en pantalla `Hola Mundo`, se puede observar que al final de la línea hay un punto y coma ( `;` ) que forma parte de la sintaxis del programa.

- » `System`: es una clase definida que representa al computador.
- » `out`: es un objeto de la clase `System`, definida que representa la pantalla.
- » `println`: método para poner un texto en la pantalla.

Los métodos de salida `print( )` y `println( )`. Son utilizados en los programas cuya salida está basada en texto y la impresión se realiza desde la consola.

## 2.5 PRESENTACIÓN DEL PROGRAMA

- » Tabulación o sangrado: margen del texto, que se usa para remarcar la estructura del programa.
- » comentarios:
  - de documentación (`/** */`).
  - normales, hasta fin de línea (`//`).
- » Uso del punto y coma ( `;` ) para finalizar declaraciones e instrucciones.
- » Se utilizan `{ }` para definir los contenidos de clases y métodos.



## 2.6 CONSTRUCCIÓN DE UN PROGRAMA EN JAVA

La construcción de un programa en Java se realiza en varias etapas:

### Primera etapa

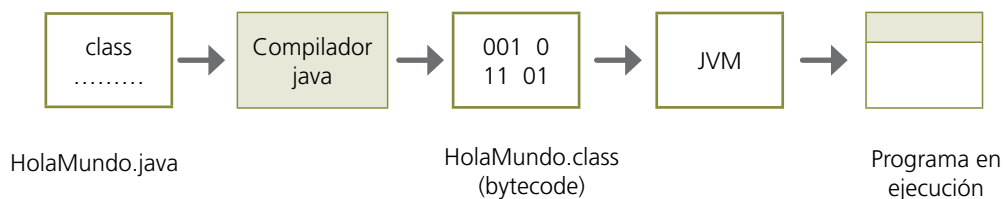
- » Creación del proyecto en cualquier entorno de desarrollo.
- » Guardarlo en el medio magnético, con el nombre del programa y la extensión java, como por ejemplo HolaMundo.java, Calculadora.java, Cuadrado.java.

### Segunda etapa

- » Compilar el programa.
- » Corregir errores de sintaxis (expresiones mal digitadas, nombres de variables diferentes a las definidas).
- » Se generan el código intermedio bytecode, al compilar el programa creando los archivos punto class por ejemplo HolaMundo.class, Calculadora.class, Cuadrado.class.

### Tercera etapa

- » En la etapa de ejecución del programa, la Máquina Virtual de Java (JVM), interpreta las instrucciones bytecode, presentado el resultado del programa en pantalla o realiza alguna tarea, calculo o proceso.



El bytecode es lenguaje nativo de cualquier implementación de la Máquina Virtual de Java. De esta forma se logra que un programa Java corra en cualquier plataforma.

## 2.7 PALABRAS RESERVADAS

Como cualquier lenguaje de programación, Java tiene un conjunto de palabras reservadas incorporada, para identificar definiciones, condiciones, ciclos entre otros, términos que no se pueden utilizar como nombres de los identificadores.

La tabla 4 se relacionan las palabras reservadas:

**Tabla 4.** Palabras reservadas en Java.

Abstract	continue	For	new	switch
Boolean	default	Goto	null	synchronized
Break	do	If	package	this
Byte	double	implements	private	threadsafe
Byvalue	else	import	protected	throw
Case	extends	instanceof	public	transient
Catch	false	Int	return	true
Char	final	interface	short	try
Class	finally	Long	static	void
Const	float	native	super	while

## 2.8 LOS IDENTIFICADORES

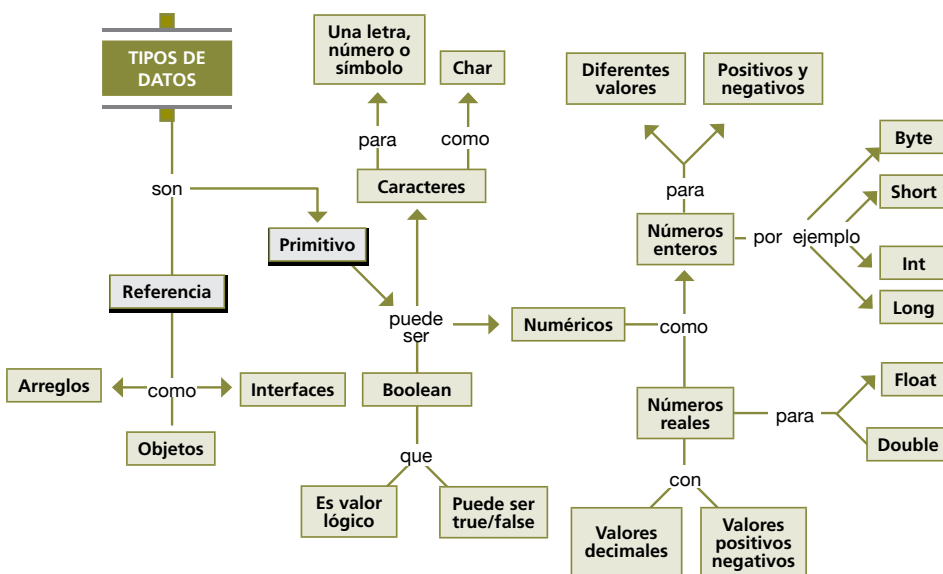
Un identificador corresponde a un nombre de variables, constantes, clases, objetos, métodos los cuales están conformados por una secuencia letras y dígitos de longitud limitada, que permite identificar los elementos de un programa en java.

Los nombres de los identificadores en java cumplen algunas reglas como el de comenzar con una letra, un signo pesos (\$), o el guión bajo (\_); no pueden iniciar con un número. Después del primer carácter, los identificadores pueden ir seguidos de cualquier combinación de letras, signo de moneda, números, guion al piso.

## 2.9 TIPOS DE DATOS

En el lenguaje de programación Java, tiene dos tipos de categorías de datos, tipos primitivos, que se caracterizan por tener un único valor, y los de tipo referencia se utilizan para referirse o acceso a un objeto. La figura 8 muestra el concepto de tipo de datos.

**Figura 8.** Concepto de Tipos de datos.



En la tabla 5 se presentan los tipos de datos de java, que se utilizan en los programas de este lenguaje de programación.

**Tabla 5.** Tipos de datos de Java.

TIPOS DE DATOS	NÚMERO DE BYTES	DESCRIPCIÓN	INICIALIZACIÓN
boolean	1	Puede contener los valores true o false.	false
byte	2	Enteros. Tamaño 8-bits. Valores entre -128 y 127. (-27 ; 27-1)	0
short	2	Enteros. Tamaño 16-bits. Entre -32768 y 32767. (-215 ; 215-1)	0
Int	8	Enteros. Tamaño 32-bits. Entre -2147483648 y 2147483647. (-231 ; 231-1)	0
long	8	Enteros. Tamaño 64-bits. Entre -9223372036854775808 y 9223372036854775807. (-263 ; 263-1)	0L
float	8	Números en coma flotante. Tamaño 32-bits.	0.0f
double	8	15 cifras decimales equivalente 4.9E-324 1.7E308	0.2d
char	2	Unicode('uxxxx')	'u000'

## 2.10 VARIABLES

Son los valores que se pueden cambiar durante la ejecución de un programa corresponde a nombres que representan un valor de cierto tipo int, char, float, double.

### Características de las variables

- » Es una unidad básica de almacenamiento de valores o datos.
- » En Java debe ser declarada antes de utilizarse.
- » Cada variable tiene un tipo y un identificador.
- » Es un nombre simbólico con un valor.
- » Las variables pueden ser iniciadas.
- » La forma para declarar las variables es: tipo de dato y nombre de la variable.

### Los nombre de las variables

- » Deben iniciar con una letra del alfabeto, guion de subrayado.
- » Los siguientes caracteres pueden ser dígitos.
- » No se puede usar palabras reservadas como nombres de las variable
- » El estándar para nombrar variables es iniciar con minúscula, para el caso cuando el nombre está compuesto por dos o más la letra inicial a partir de la segunda palabra va en mayúscula.

### Ejemplos:

`costo, iva, x, x1, x2, costoItemFacturado, nombreEmpleado.`

## 2.11 CONSTANTES

Una constante es un identificador que es similar a una variable, excepto porque no se le puede asignar un valor en tiempo de ejecución, se escriben en mayúscula sostenida.

*Son ejemplos de nombres de constantes: PI, IVA, NOMBRE*

## 2.12 COMENTARIOS

Se utilizan para hacer descripciones de código y facilitar información adicional que no es legible en el programa. Los comentarios deben contener sólo información que es relevante para la lectura y entendimiento de la aplicación.

En Java hay tres tipos de comentarios, empleados para documentar un programa o para cancelar líneas que no sean tenidas en cuenta al momento de compilar el programa.

### Comentarios para una sola línea

La pareja de caracteres `//` hay que escribirla sin dejar ningún espacio en blanco entre ellos.

#### Ejemplo:

```
// Programa que suma dos números
// x variable que almacena un dato de tipo entero
```

### Comentarios de bloque

Un comentario de bloque debe ir precedido por una línea en blanco que lo separe del resto del código; estos se usan para dar descripciones del programa utilizados o antes de cada clase, métodos, atributos. También se pueden utilizar al interior de los métodos y deben estar al mismo nivel que el código del método que describen.

#### Ejemplo:

```
/*
 * Aquí hay un comentario de bloque.
 * comentarios para varias líneas
 */
```

## Comentarios de documentación

Describen clases Java, interfaces, constructores, métodos y atributos. Cada comentario de documentación se encierra con los delimitadores de comentarios `/**...*/`

### Ejemplos:

```
/**
 * La clase que modela la información de un cliente
 */
/**
 * La clase que modela las transacciones de una cuenta corriente.
 */
```

## 2.13 DEFINICIÓN DE VARIABLES

Las variables se deben definir antes de utilizarlas o ser nombradas dentro del programa para lo cual se debe tener en cuenta el tipo de dato y nombre variable.

### Ejemplo

TIPO DE DATO	NOMBRE	VARIABLE
int	suma;	// se declara una variable entera
double	venta;	// se declara una variable double
char	letra	// se declara una variable de tipo char
boolean	respuesta	// se declara una variable de tipo booleano

Se puede observar que al final de cada sentencia se coloca un punto y coma, que forma parte de la sintaxis de java y el doble slash `//` corresponde a comentarios que ayudan a documentar el programa en cuanto a su contenido y funcionalidad.

## 2.14 DEFINICIÓN DE CONSTANTES

Se realiza anteponiéndole la palabra final al tipo de dato y una vez inicializadas no se puede cambiar su valor, Las constantes se escriben en mayúscula sostenida todo el nombre.

Ejemplos

TIPO DE DATO	NOMBRE
final double	PI = 3.1416;
final int	IVA = 16;
final int	VALOR_MAXIMO = 100;
final char	LETRA = 'S';

2.15 OPERADORES ARITMÉTICOS

Permiten la realización de las operaciones aritméticas básicas, que trabajan sobre variables numéricas, en la tabla 6 se presentan estos operadores.

Tabla 6. Operadores aritméticos.

OPERACIÓN	OPERADOR	EJEMPLO
suma	+	z=x+y; a=b+2;
resta	-	z=x-y; a=b-2;
producto	*	z=x*y; a=b*2;
división	/	z=x/y; a=b/4;
residuo o modulo de la división	%	z=x%y; a=b%2;
Incremento	++	x++;
Decremento	--	y--;



## Ejemplos

```
int a;  
int b;  
int c;  
int d;  
a = 2 + 2; // adición  
b = a * 3; // multiplicación  
c = b - 2; // substracción  
d = b / 2; // división  
e = b % 2; // resto de la división (módulo)
```

## Operadores incremento y decremento

- » El operador ++ incrementa en 1 la variable que sigue.
- » El operador - - decrementa en 1 la variable que acompaña.

## Ejemplos

```
int x = 3;  
x ++; // Se incrementa x en uno ; luego queda con el valor de 4;  
int y = 3;  
y - - // Se decrementa en uno; luego queda con el valor igual a  
dos;
```

## 2.16 LOS SEPARADORES

Admitidos en Java son:

Paréntesis **()**, utilizados para contener listas de parámetros en la definición, llamada a métodos, agrupación de expresiones, contener expresiones para control de flujo.

Llaves **{ }**, son utilizadas para definir un bloque de código, para clases, métodos ámbitos locales y para inicializar arreglos en forma automática.

Corchetes **[ ]**, Utilizados para declarar tipos de los arreglos o cuando se referencian valores de los arreglos.

Punto y coma **;** usados para separar sentencias.

Coma (,) permite separar identificadores consecutivos en una declaración de variables, también es utilizada para encadenar sentencias dentro de una sentencia de un ciclo for.

Punto (.) permite separar nombres de paquete de subpaquetes; clases de atributo, objetos de nombre del método.

## 2.17 LECTURAS RECOMENDADAS

- » Precedencia de los operadores aritméticos.
- » Tipos de datos primitivos.
- » Tipos de datos referencia.
- » Documentación de un programa.

## 2.18 PREGUNTAS DE REVISIÓN DE CONCEPTOS

- » ¿Qué tipo de datos sería recomendable para almacenar los valores de las funciones trigonométricas?
- » ¿Qué es un programa fuente?
- » ¿Qué es compilar un programa?
- » ¿Qué son errores de sintaxis?
- » ¿Qué es el API de java?

## 2.19 EJERCICIOS

- » Definir los tipos de datos para el radio, área y longitud de una circunferencia.
- » Definir cinco tipos de constantes distintas.

## 2.20 REFERENCIAS BIBLIOGRÁFICAS

Booch, G. (1996). *Análisis y diseño orientado a objetos con aplicaciones*. Ciudad: México Addison Wesley.

Deitel y Deitel. (2012). *Cómo programar en Java*. Ciudad: México Editorial Pearson (Prentice Hall).

Fowler, M., y Scott, K. UML. (1999). *Gota a gota*. Ciudad: México Editorial Pearson. Rumbaugh, J., Jacobson, I., y; Booch, G.: (2005). *The Unified Modeling Language User Guide*, San Francisco: Addison-Wesley, Reading, Mass. ISBN-13: 078-5342267976 ISBN-10: 0321267974

Villalobos, J., y Casallas, R. (2006). *Fundamentos de programación*, aprendizaje activo basado en casos. Ciudad: México Editorial Pearson.



# CAPÍTULO 3

Herramientas para el desarrollo de la

**PROGRAMACIÓN  
ORIENTADA A OBJETOS**



### 3.1 TEMÁTICA A DESARROLLAR

- » Entornos Integrados de Desarrollo (IDE).
- » Herramientas de modelamiento.
- » El lenguaje de programación Java.

### 3.2 INTRODUCCIÓN

La construcción de una aplicación en Java involucra el uso de una serie de herramientas de software, que facilitan su modelamiento, codificación, pruebas y su posterior documentación; para ello hay herramientas case y entornos integrados de desarrollo.

- » Los entornos de desarrollo integrados están diseñados para maximizar la productividad del desarrollador de aplicaciones proporcionando componentes unidos con interfaces de usuario similares. Los IDE presentan un único programa en el que se lleva a cabo todo el desarrollo y, por lo general, esta herramienta suele ofrecer muchas características para la creación, modificación, compilación, implementación y depuración de software. Los IDE brinda un marco de trabajo amigable para la mayoría de los lenguajes de programación.
- » Las herramientas de modelado UML se convierten en partes fundamentales en el desarrollo de todo sistema de información orientado a objetos, permitiendo representar los requerimientos del sistema, la funcionalidad y la interacción con otros módulos o sistemas.
- » Java como herramienta de Programación Orientada a Objetos, es utilizado para la construcción de las aplicaciones de software, donde los programas se elaboran a partir de módulos independientes, los cuales se pueden modificar o complementar.

El uso adecuado de una herramienta de modelado de software, así como de un entorno integrado de desarrollo y de un lenguaje de programación como Java, son buenas prácticas para el desarrollo de aplicaciones orientadas a objetos.

### 3.3 ENTORNOS INTEGRADOS DE DESARROLLO (IDE)

Son aplicaciones de software que integran herramientas, componen un entorno de desarrollo integrado por: un editor de texto, un compilador, un intérprete, elementos para la documentación, un depurador, un sistema de ayuda para la construcción de interfaces gráficas de usuario y un generador de control de versiones.

Un Entorno de Desarrollo Integrado (IDE), brinda una serie de opciones y funcionalidades para el desarrollo de programas o aplicaciones de software que van desde aplicaciones de escritorio (calculadora, juegos), hasta sistemas de información de tipo empresarial.

Los IDE pueden ser utilizados para un único lenguaje de programación, pero algunas de estas herramientas como Eclipse o NetBeans. Soportan más de un lenguaje de programación dentro de los que están Java, C++, Ruby, PHP entre otros, ya que incluyen plugins (programa que incrementa las funcionalidades del IDE).

Algunos de los principales y más populares IDE para Java son:

- » **BlueJ:** desarrollado como un proyecto de investigación universitaria, es libre.
- » **Eclipse:** desarrollado por la Fundación Eclipse, es libre y de código abierto.
- » **IntelliJ IDEA:** desarrollado por JetBrains, es comercial.
- » **Jbuilder:** desarrollado por Borland, es comercial pero también existe la versión gratuita.
- » **JCreator:** desarrollado por Xinox, es comercial pero también existe la versión gratuita.
- » **JDeveloper:** desarrollado por Oracle Corporation, es gratuito.
- » **NetBeans:** es gratuito y de código abierto.

#### 3.3.1 Eclipse

Entorno de desarrollo integrado de código abierto y multiplataforma (Windows, Linux), es una herramienta para la programación, desarrollo y compilación de aplicaciones en C++, Java, PHP, entre otros.

Se descarga del sitio <http://www.eclipse.org/>, seleccione la opción downloads, o digite directamente <http://www.eclipse.org/downloads/> y elegir Eclipse Classic, para 32 o 64 bits, según arquitectura del computador a trabajar.

**Figura 9.** Pantalla de carga de Eclipse LUNA.



La figura 9 corresponde a la versión LUNA de eclipse, versión 4.4 cuyo lanzamiento oficial fue el mes de junio del 2014.

### 3.3.2 NetBeans

Es otro entorno integrado de desarrollo para los programadores que les permiten digitar, compilar, depurar y ejecutar programas. Está construido en Java, pero puede servir para cualquier otro lenguaje de programación. Netbeans facilita la construcción de aplicaciones a partir de un conjunto de componentes de software llamados módulos.

La figura 10 corresponde a la pantalla de inicio de la versión 8.0.2 lanzada al mercado en noviembre del 2014.

**Figura 10.** Pantalla inicio en Netbeans.





Se descarga del sitio <http://netbeans.org/> o <http://netbeans.org/downloads/> y puede seleccionar entre las Tecnologías: Java SE, JavaFX, Java, Ruby, C/C++, PHP, All (todo el paquete completo).

### 3.3.3 BlueJ

Es un entorno integrado desarrollado de un proyecto de investigación académico de las universidades de Deakin University, Melbourne, Australia y la Universidad de Kent, en Canterbury, UK., diseñado para enseñar la programación orientada a objetos a principiantes con especial énfasis en las técnicas de visualización e interacción en un ambiente que fomenta la experimentación y exploración. Se puede descargar del sitio web <http://bluej.org/>

La figura 11 corresponde a la pantalla de inicio de la BlueJ.

**Figura 11.** Pantalla inicio de BlueJ.



### 3.3.4 GreenFoot

Es un entorno integrado desarrollado especialmente para programadores novatos. Si bien soporta el lenguaje Java en su totalidad, es especialmente útil para programar ejercicios que tengan contenido visual. En GreenFoot la interacción y visualización de objetos son los elementos fundamentales. El sitio web oficial del proyecto es <http://www.greenfoot.org/door>

La figura 12 corresponde a la pantalla de inicio de la GreeFoot.

**Figura 12.** Pantalla inicio de GreeFoot.

### 3.3.5 IntelliJ IDEA

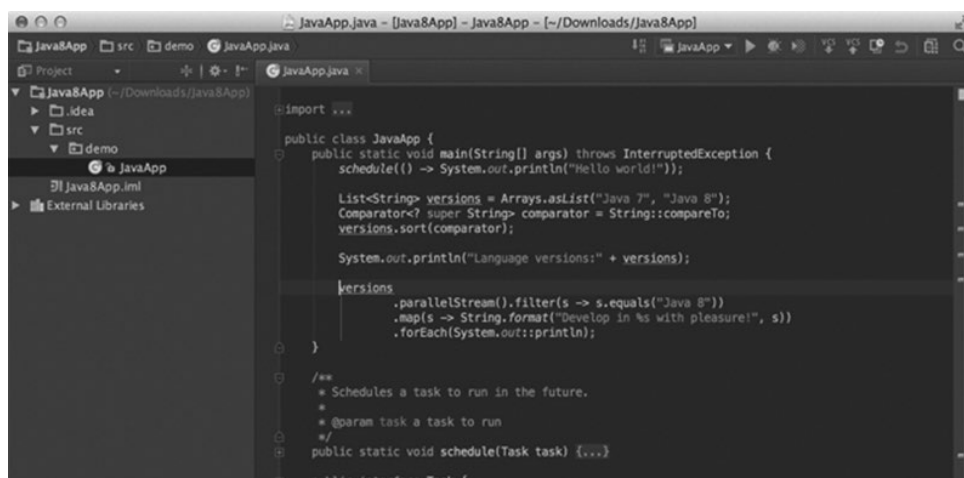
Es uno de los mejores IDE utilizado para Java, de la mano de JetBrains que siempre tubo características que no se encuentran en otros (Eclipse, NetBeans, etc) como el guardado y compilado automático, control de sugerencias y resaltado de código dentro incluso de cadenas, el mas inteligente auto-completado de código y mucho mas. La versión 15 salió el mes de noviembre de 2015 y traen varias novedades y características especiales para los desarrolladores Android. En esta publicación se citan las nuevas características y lenguajes, tecnologías frameworks soportados.

#### Características:

- » Es multiplataforma (Windows, Mac OS X, Linux).
- » Existen dos ediciones Community Edition que es de código abierto y Ultimate de pago, que se puede descargarla gratis para probarla libremente 30 días esta edición tiene más características que la Community Edition.
- » Mejor compilación, rendimiento mejorado que redujo requisitos de la memoria.
- » Soporte para Java 8, la nueva generación de la plataforma Java y con su nueva sintaxis de las expresiones lambda.
- » Actualización de las herramientas del framework Spring, con mejoras en rendimiento y asistencia.

- » Entre los lenguajes soportados están: Java, Groovy, XML, Regexp, Scala, Clojure, Ruby/JRuby, Python, PHP, entre otros
- » Sitio oficial de descarga <https://www.jetbrains.com/idea/>
- » La siguiente imagen visualiza el entorno de jetbrains, con una aplicación en Java.

Figura 13. Pantalla IntelliJ IDEA.



## 3.4 HERRAMIENTAS DE MODELAMIENTO

UML es un lenguaje visual para modelar las partes esenciales de un sistema de software. Es una herramienta útil para las etapas de diseño e implementación de aplicaciones de software y también para la documentación de un proyecto de software.

El desarrollo de aplicaciones de Software puede llegar a ser un proceso complejo y a menudo difícil que en algunos casos requiere la integración de muchos sistemas. Desde el modelado y diseño hasta el código, administración del proyecto, pruebas, despliegue, administración de cambios y más, el modelado basado en UML se ha convertido en una parte esencial para administrar esa complejidad.

### 3.4.1 StarUML

Es una herramienta para el modelamiento de software basado en los estándares UML (Unified Modeling Language) y MDA (Model Driven Architecture), el cual permite:

- » Soporte para el diseño de diagramas UML.
- » Construir los planos para una aplicación de software.
- » Generar código a partir de los diagramas y viceversa, actualmente funcionando para los lenguajes c++, c# y java.

### 3.4.2 Dia

Programa que está desarrollado para la construcción de diagramas UML, como todo producto de software libre es gratuito y de código abierto.

Dia para Linux, presenta una interfaz amigable de fácil manejo, para elaborar diferentes tipos de diagramas como, los diagramas de UML, diagramas de flujo, diagramas de red.

### 3.4.3 ArgoUML

Es una herramienta utilizada en el modelaje de sistemas, mediante la cual se realizan diseños en UML (Unified Modeling Language) llevados a cabo en el análisis y diseño de Sistemas de Software.

### 3.4.4 Rational Software Modeler

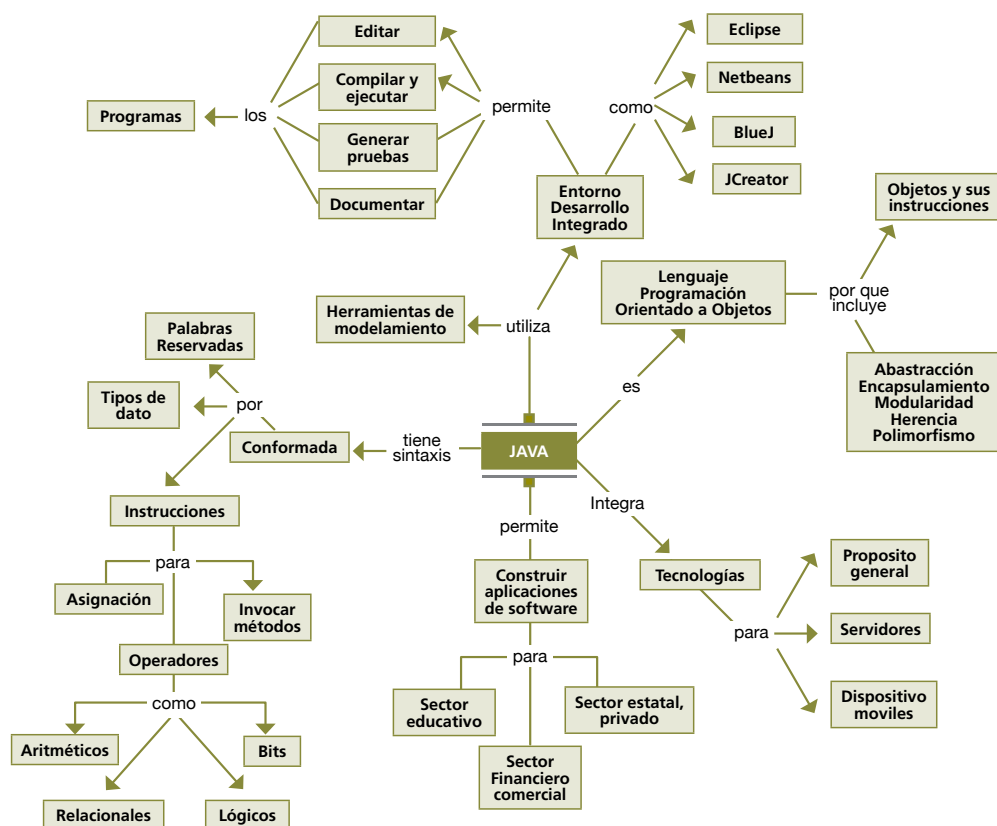
Es una herramienta de diseño y modelado visual basada en UML para la documentación y la comunicación con diferentes vistas de un sistema, debe ser adquirida la licencia para poder utilizarlo.

## 3.5 JAVA COMO LENGUAJE DE PROGRAMACIÓN

Un lenguaje de programación está construido a partir de un conjunto de expresiones, símbolos y reglas sintácticas y gramaticales que permiten escribir instrucciones o sentencias válidas, para ser ejecutadas en un computador.

Dentro de algunos lenguajes de programación orientada a objetos están C++, java, C#, php a partir versión 5.0, entre otros. Para efectos del desarrollo de proyectos y la codificación de los programas de este documento se utilizará Java como lenguaje de programación.

Figura 14. Concepto de Java.



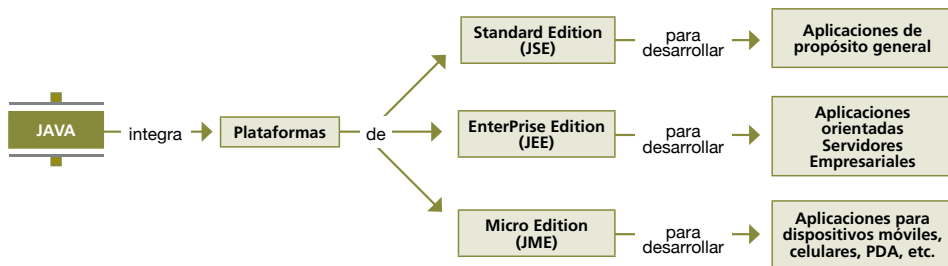
Java es un lenguaje de programación orientado a objetos, desarrollado por Sun Microsystems al iniciar la década de 1990, actualmente fue adquirido por Oracle. La tecnología Java está compuesta básicamente por dos elementos: el lenguaje Java y su plataforma, que corresponde a la máquina virtual de Java (Java Virtual Machine JVM).

La figura 14 detalla, mediante un mapa mental los fundamentos del lenguaje de programación Java y la figura 15, describe las plataformas.

Java permite desarrollo de aplicaciones o soluciones para:

- » Desarrollo de aplicaciones de software según necesidades.
- » Desarrollo empresarial.
- » Dispositivos móviles

**Figura 15.** Concepto de plataformas de Java.



Algunas de las características de Java son:

- » Una misma aplicación puede funcionar en diversos tipos de computadores y sistemas operativos: Windows, Linux, Solaris, MacOS-X, así como en otros dispositivos inteligentes.
- » Los programas Java pueden ser aplicaciones independientes.
- » Su desarrollo está impulsado por un amplio colectivo de empresas y organizaciones, integrada con la filosofía de software abierto y entorno colaborativo.

### 3.6 LECTURAS RECOMENDADAS

- » ¿Qué son los entornos de desarrollo integrados?
- » ¿Qué son Eclipse, NetBeans y BlueJ? Mencionar sus principales características
- » Importancia del modelamiento
- » Fundamentos, versiones y evolución de Java.

### 3.7 PREGUNTAS DE REVISIÓN DE CONCEPTOS

- » ¿Importancia de los IDE?
- » Diferencias entre IDE
- » ¿Qué son las variables de entorno?

### 3.8 EJERCICIOS

- » Descargar Java del sitio web oficial e instalarlo
- » Descargar los Entornos de Desarrollo Integrado:
  - Eclipse,
  - NetBeans,
  - JBlue, y
  - IntelliJ IDEA.
- » Ejecutar cada uno de los anteriores entornos integrados de desarrollo
- » Descargar las herramientas de diseño:
  - StarUML.
  - ArgoUML.
  - Dia.

### 3.9 REFERENCIAS BIBLIOGRÁFICAS

Las siguientes referencias corresponden a los sitios web donde se puede obtener información complementaria de los temas tratados en este capítulo, así como la descarga de los IDE:

- » <http://docs.oracle.com/javase/7/docs/api/>
- » <https://docs.oracle.com/javase/8/docs/api/>
- » <https://netbeans.org/>
- » <https://eclipse.org/>
- » <http://www.bluej.org/>
- » <https://www.jetbrains.com/idea/>







# CAPÍTULO 4

**Métodos Constructores,**

**SET Y GET**



## 4.1 TEMÁTICA A DESARROLLAR

- » Métodos constructores.
- » Métodos set.
- » Métodos get.
- » Modificadores.
- » Requerimientos.
- » Entrada y salida de datos.
- » Stream.
- » Clase entrada y salida de datos.

## 4.2 INTRODUCCIÓN

En este capítulo se incluye, como primer tema, el concepto de constructor el cual es un método empleado para inicializar valores en instancias de los objetos, la principal característica de los constructores es que llevan el mismo nombre de la clase y no retornan ningún tipo de dato.

El segundo tema a tratar y que se quiere resaltar, son los métodos accesores y modificadores (set/get), el método set, es utilizado para actualizar el valor de una variable y el método get para obtener el valor de la variable, son la forma de acceder a los atributos de la clase definidos como tipo privado, haciendo necesario implementar los métodos setters y getters para cada atributo de la clase definida; su implementación corresponden a una buena práctica de programación o técnicas para desarrollar y mantener software de calidad.

La temática se complementa mediante ejercicios prácticos, haciendo uso de las diferentes formas para la entrada y salida de datos estándar.

## 4.3 MÉTODOS

Corresponden a la lógica de la clase, son serie de instrucciones que realizan alguna tarea en específica; en las clases, se pueden identificar diferentes tipos de métodos:

- » Métodos inicializadores, inicializan los atributos de la clase.
- » Métodos selectores, devuelven el valor de los atributos.
- » Métodos modificadores, permiten cambiar el valor de los atributos.
- » Métodos visualizadores, imprimen el objeto, es decir, el valor de los atributos.
- » Métodos operadores, realizan cálculos y generan resultados.
- » Constructores, inicializan los atributos del objeto.

La figura 16 ilustra el concepto método.

**Figura 16.** Concepto de Métodos.



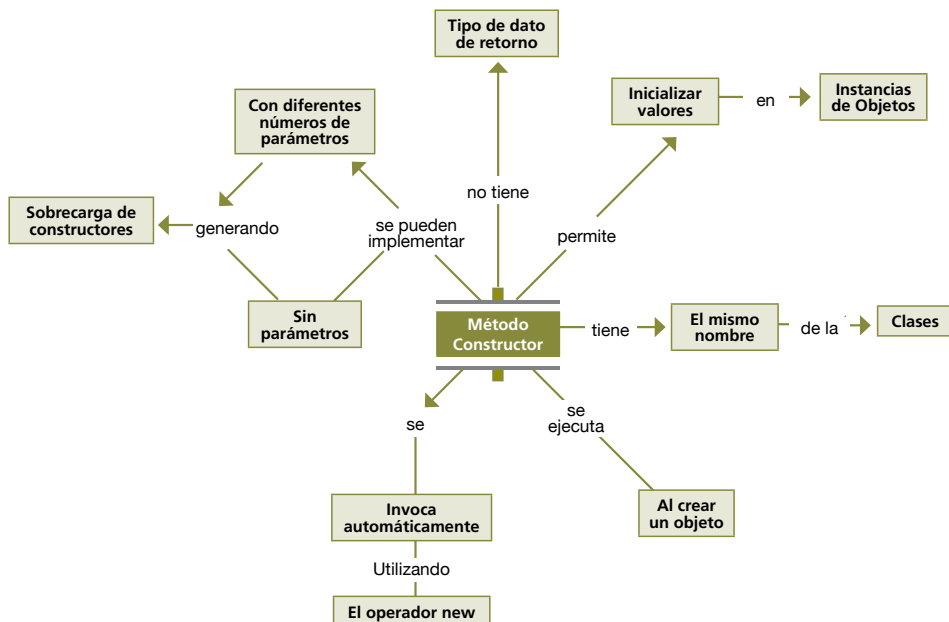
### 4.3.1 Método constructor

Es un método más de la clase que presenta las siguientes características:

- » Un constructor es un método miembro público con el mismo nombre de la clase.
- » No está indicado ningún tipo de retorno int, double, String, ni siquiera void.
- » El objetivo del constructor es reservar memoria e inicializar el valor de las variables miembro de la clase.
- » Es un método que se llama automáticamente cada vez que se crea un objeto de una clase.

La figura 17 detalla el concepto método constructor.

**Figura 17.** Concepto de Constructor



## Definición de los métodos constructores

La sintaxis para la definición de los constructores es como la de cualquier otro método, conformado por el mismo nombre de la clase, una lista de parámetros separados por coma y no puede haber ningún otro método que comparta el nombre de la clase.

```
class <NombreClase>
{
...
public <NombreClase> ( ) { }
...
}
class Calculadora
{
...
public Calculadora ( ) { }
...
}
```

Llamada a un constructor

Al constructor de una clase se le invoca en el momento de crear el objeto utilizando el operador new. La forma de uso de este operador es:

```
new <llamadaConstructor>
new unaCalculadora( );
```

Por ejemplo Calculadora unaCalculadora = new Calculadora();

new es una palabra reservada, que se utiliza para construir un objeto de una clase, permitiendo ejecutar el constructor e inicializando los campos del objeto.

Calculadora	unaCalculadora	= new	Calculadora(x,y)
Clase	Objeto		constructor

## 4.4 MODIFICADORES DE ACCESO

Los modificadores son elementos del lenguaje que se colocan delante de la definición de los atributos, métodos o clases, alteran o condicionan el significado del elemento y brindan la posibilidad de poder establecer la visibilidad o manera de dar los permisos que tendrán otros objetos para acceder a los métodos y/o atributos de la clase.

Los modificadores de acceso permiten al diseñador de una clase determinar quién puede acceder a los datos y métodos miembros de una clase. Tiene la siguiente representación sintáctica.

```
[modificadores] tipo_variable nombre;  
[modificadores] tipo_devuelto nombre_Metodo ( lista_Argumentos );
```

### Modificador public

Aplicado a un atributo o método, significa que estos pueden ser vistos y utilizados por objetos de clases siendo accesibles por cualquier referencia (objeto) que los invoque. Se representa con el signo más ( + ).

### Modificador private

Es el nivel más restrictivo de todos los modificadores, se puede utilizar en atributos como en métodos, sólo se puede acceder al elemento desde métodos de la clase, o sólo puede invocarse el método desde otro método de la clase. Se representa con el signo menos ( - ).

### this

Palabra reservada para acceder a variables de instancia de una clase, this se utiliza siempre que se quiere hacer referencia al objeto actual de la clase o cuando hay ambigüedad entre los nombres de los campos del objeto.

### Por ejemplo:

La palabra reservada this permite especificar que la variable que señala (this.x) es de la misma clase de la que se usa.



```

public class Punto {
    /**
     * atributos
     */
    private double x;
    private double y;

    /**
     * constructor con dos parametros
     * @param x
     * @param y
     */

    public Punto (double x, double y) {
        this.x = x;
        this.y = y;
    }
}

```

---

```
this. x = x;
```

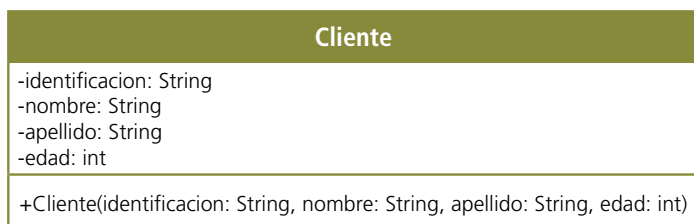
- La primera x corresponde al atributo de la clase
- La segunda x es el parámetro del método (double x).

El this, evita la ambigüedad ya que el nombre del atributo y del parámetro son el mismo ( x ).

## Ejemplos

1. La clase que modela la identificación, el nombre, el apellido, la dirección atributos correspondientes a la información del cliente.

## Diagrama de Clases



El signo “-” delante de las **propiedades** identificación, nombre y apellido y edad, indica acceso privado

El signo “+” delante de los **métodos** indica acceso público.

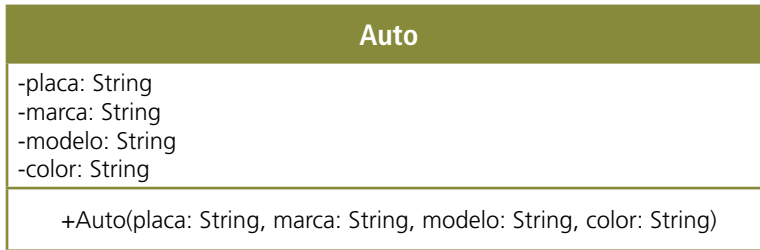
Codificación de la clase cliente, es una buena práctica en la programación orientada a objetos declarar siempre todas las variables con el identificador de acceso privado “private”, ya que esta variable solo será accesible desde la clase, con lo cual se está brindando la seguridad del programa.

```
public class Cliente {
    private String identificacion;
    private String nombre;
    private String apellido;
    private int edad;
    /**
     * constructor de la clase, con cuatro parámetros:
     * identificacion, nombre,
     * apellido, edad
     */
    public Cliente(String identificacion, String nombre,
        String apellido, int edad) {

        this.identificacion = identificacion;
        this.nombre = nombre;
        this.apellido = apellido;
        this.edad = edad;
    }
}
```

2. Clase que modela los datos de la información de un auto conformada por la placa, la marca, el modelo, el color; siendo estos los atributos del auto.

### Diagrama de clases



Codificación de la clase Auto

```
public class Auto {
    private String placa;
    private String marca;
    private String modelo;
    private String color;

    /**
     * constructor con los parámetros placa, marca, modelo,
     * color
     */
    public Auto(String placa, String marca, String modelo,
        String color) {
        this.placa = placa;
        this.marca = marca;
        this.modelo = modelo;
        this.color = color;
    }
}
```

Una clase puede definir varios constructores con distintos tipos de argumentos lo que se conoce como sobrecarga de constructores.

3. Se construye una aplicación con dos clases: La clase cliente con dos constructores el primero sin parámetro y el segundo con una lista de parámetro, separados por comas. Y la segunda clase denominada PruebaCliente, se crean dos objetos de la clase Cliente.

```

public class Cliente {
    private String identificacion;
    private String nombre;
    private String apellido;
    private int edad;
    /**
     * primer constructor de la clase los atributos se in-
    cializan con los valores
     * por defecto
     */
    public Cliente() {
        this.identificacion = "";
        this.nombre = "";
        this.apellido = "";
        this.edad = 0;
    }
    /**
     * Segundo constructor de la clase, con cuatro
     * parámetros identificacion, nombre, apellido, edad
     */
    public Cliente(String identificacion, String nombre,
String apellido,
        int edad) {
        this.identificacion = identificacion;
        this.nombre = nombre;
        this.apellido = apellido;
        this.edad = edad;
    }
}

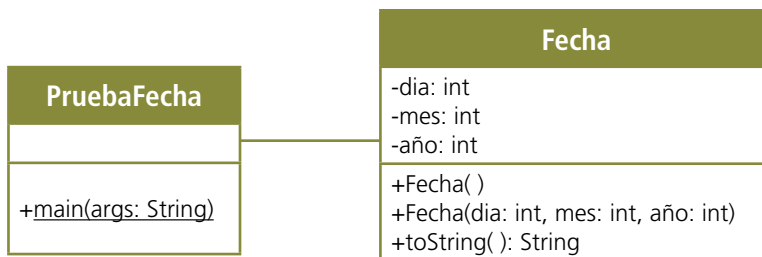
public class PruebaCliente {
    public static void main(String[] args) {
        Cliente unCliente = new Cliente();
        Cliente otroCliente = new Cliente("123456", "Alan",
        "Brito", 30);
    }
}

```

Cliente unCliente = new Cliente(), se crea el objeto unCliente y se invoca la constructor sin parámetros.

Cliente otroCliente = new Cliente("123456", "Alan ", "Brito", 30); se crea el objeto otroCliente y se invoca el constructor con parámetros, el cual inicializa los valores del objeto.

4. Aplicación fecha diseñada con dos clases. La clase fecha conformada por los atributos día, mes y año, en la que se implementan dos constructores; el primero sin parámetros (los atributos son inicializados por defecto) y el segundo con tres parámetros (día, mes, año); la segunda clase se denomina PruebaFecha.



```

public class Fecha {
    private int dia;
    private int mes;
    private int año;
    /**
     * Primer Constructor
     * Asigna los valores 30, 7 y 2015 a los atributos
     * dia, mes y año respectivamente
     */
    public Fecha( ) {
        this.dia = 30;
        this.mes = 7;
        this.año = 2015;
    }
}
  
```

```

/**
 * Segundo Constructor
 * @param dia, el dia del mes a almacenar
 * @param mes, el mes del anho a almacenar
 * @param año, el año a almacenar
 */

public Fecha(int dia, int mes, int año) {
    this.dia = dia;
    this.mes = mes;
    this.año = año;
}

/**
 * Método que retorna el objeto de la fecha
 */
public String toString() {
    return ""+dia + "/" + mes + "/" + año;
}
}

public class PruebaFecha {
    public static void main(String[] args) {
        Fecha unaFecha = new Fecha();
        Fecha miFecha = new Fecha(6,1,1995);
        System.out.println("La Fecha del objeto"
            +miFecha.toString());
    }
}

```

En la clase PruebaFecha se definen dos objetos: unaFecha y miFecha, este último con los tres parámetros, correspondientes al día, mes y año.

El objeto miFecha invoca al método toString(), el cual retorna la fecha el dato almacenado el objeto que es: 6/1/1995.

## 4.5 MÉTODOS SET Y GET

Los objetos encapsulan (estado y comportamiento), el estado definido por los datos o atributos del objeto y el comportamiento u operación del objeto definido por los métodos.

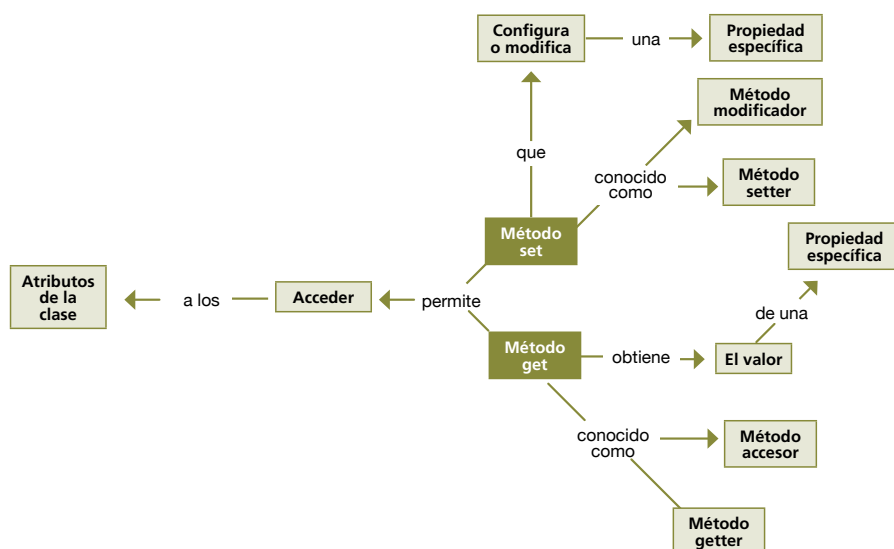
Con la finalidad de crear una puerta de entrada y de salida para modificar y obtener los atributos del objeto, se implementan los métodos set y get respectivamente para cada atributo de la clase.

Los métodos set se utilizan para inicializar o cambiar el valor de los atributos, set, al traducirlo podría corresponder a establecer, actualizar siendo método que cambia el valor del atributo, ya que este método tiene parámetro.

Los métodos get se emplean cuando se requiere obtener el valor de algún atributo, get, al traducirlo es conseguir, obtener y retorna el valor del atributo, por lo general no tiene ningún parámetro.

La figura 18 representa los conceptos relacionados con los métodos set y get.

**Figura 18.** Concepto de métodos Set y Get.



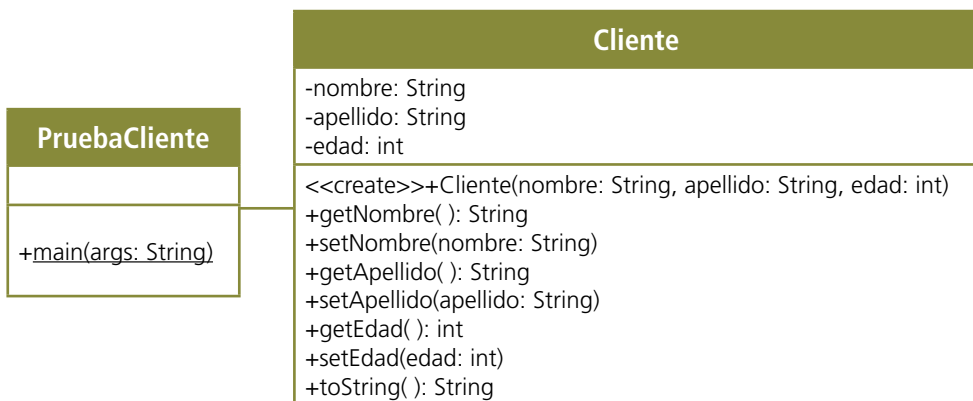
## Ejemplos

1. De la clase que modela el cliente, se ha definido el atributo nombre, para el cual se implementan los métodos: setNombre que tiene como parámetro el nuevo dato a cambiar en el atributo y getNombre retorna el valor actual del atributo nombre.

```
public class Cliente {
    private String nombre;
    public String getNombre() {
        return nombre;
    }
    public void setNombre(String nombre) {
        this.nombre = nombre;
    }
}
```

2. Aplicación conformada por la clase Cliente en la que se definen los atributos nombre, apellidos y edad, además se implementa un constructor y los métodos set y get para cada atributo. En la segunda clase denominada Prueba Cliente, se define el objeto, unCliente de la clase Cliente.

### Diagrama de clases





Para garantizar la reutilización de las clases es una buena práctica de programación crear las clases en archivos diferentes.

```
public class PruebaCliente {
    public static void main(String[] args) {
        Cliente unCliente = new Cliente("Oscar","Hernández",12);
        //impresion de los datos del objeto al ser definido
        System.out.println("Datos Cliente" +unCliente.toString());
        //se cambian los datos del objeto unCliente
        unCliente.setNombre("Mario");
        unCliente.setApellido("Cabrera");
        unCliente.setEdad(41);

        // se imprimen los datos en forma individual del
        objeto
        System.out.println("Nombre: "+unCliente.getNombre());
        System.out.println("Apellidos: "+unCliente.getApellido());
        System.out.println("Edad: "+unCliente.getEdad());

        //impresion de los datos del objeto
        System.out.println("Datos del cliente "+unCliente.toString());
    }
}
```

---

```
/**
 * Clase que modela los datos del cliente
 *
 */
public class Cliente {
    private String nombre;
    private String apellido;
    private int edad;

    /**
     * constructor de la clase, con cuatro parametros nombre,
     * apellido, edad
     */
}
```

```
public Cliente(String nombre, String apellido, int edad)
{
    this.nombre = nombre;
    this.apellido = apellido;
    this.edad = edad;
}

/**
 * metodo que retorna el nombre
 * @return nombre
 */
public String getNombre() {
    return nombre;
}

/**
 * método que actualiza o cambia el nombre
 * @param nombre
 */
public void setNombre(String nombre) {
    this.nombre = nombre;
}

/**
 * método que retorna el apellido
 * @return nombre
 */
public String getApellido() {
    return apellido;
}

/**
 * metodo que actualiza el apellido
 * @param apellido
 */
public void setApellido(String apellido) {
    this.apellido = apellido;
}

/**
 * método que retorna la edad
 * @return edad
 */
```

```
public int getEdad() {
    return edad;
}

/**
 * método que actualiza o cambia la edad
 * @param edad
 */
public void setEdad(int edad) {
    this.edad = edad;
}

/**
 * método que retorna el nombre, apellido y edad
 *
 */
public String toString() {
    return nombre+" "+apellido+" "+edad;
}
}
```

---

## 4.6 MÉTODOS DE INSTANCIAS

Un método proporciona la implementación del comportamiento dinámico de los objetos y puede cambiar el estado de los objetos que llama. También son los algoritmos de las clases, correspondientes a las operaciones que puede realizar un objeto siendo la mejor forma de crear y mantener un programa, construido por piezas o módulos denominados métodos.

Los métodos se declaran en las clases y su implementación está integrada por el tipo de retorno, el nombre y un listado de parámetros que puede expresar así:

```
tipoRetorno nombreMetodo( [lista_de_argumentos] ) {
    cuerpoMetodo
}
```

Existen diferentes tipos de métodos dentro de los que están Constructor, Accesos y modificadores (set/get).

En Java, la manera de representar las entradas y las salidas es por medio de los streams o flujos de datos, que corresponden a una conexión entre el programa y la fuente o destino de los datos, para lo cual Java tiene implementadas una serie de clases dentro de las que están: Scanner, BufferedReader, entre otras.

## 4.7 ESTRUCTURA DE UN MÉTODO

Los programas en Java se construyen mediante clases las cuales están conformadas por atributos y métodos, y a partir de estas se pueden crear los objetos.

Los métodos son los algoritmos o servicios de las clases y corresponden a la parte lógica de la programación.

Está conformado por la declaración y el cuerpo del método.

```
Declaración del metodo() {
    Cuerpo del metodo
}
```

La Declaración o firma del método está conformada por el tipo de retorno; el nombre, que debe ser una expresión verbal que indica una acción y una lista de parámetros o argumentos (opcional).

El cuerpo del método está construido por las sentencias que van a realizar una tarea específica, como por ejemplo, calcular el valor del IVA de un producto, multiplicar dos números, obtener el promedio de ventas al año, entre otros.

La sintaxis general de un método se puede definir así:

```
Tipo dato devuelto  nombre del método (lista argumentos o parámetros) {
    Bloque o conjunto de instrucciones;
}
```

**El Tipo de dato devuelto**, puede ser int, long, float, double, char, String, entre otros; dependiendo del requerimiento.

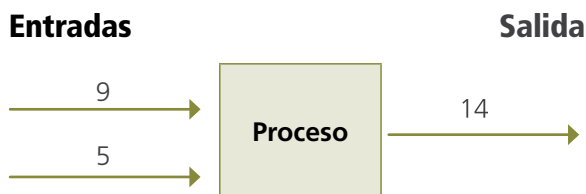
**Nombre del método**, se recomienda que sea un verbo o acción verbal como por ejemplo: sumar, obtenerSuma, calcularVentas, validarFecha. Cuando el nombre del método está conformado por nombres compuestos, el primer carácter de la segunda palabra inicia con mayúscula y así sucesivamente.

**Lista de argumentos o parámetros**, se expresa declarando el tipo de dato y el nombre de éste, separado por comas.

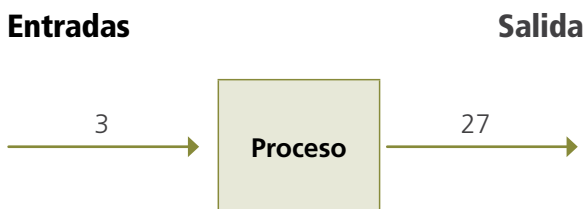
Los métodos se pueden considerar como máquinas que realizan tareas específicas, los cuales pueden tener una o más entradas y una salida.

## Ejemplos

1. Se requiere sumar dos números. Para realizar este proceso se deben ingresar dos valores, por ejemplo 9 y 5; la salida generada es 14, como se puede observar en el siguiente gráfico



2. Para calcular el cubo de un número, se necesita un valor de entrada por ejemplo 3; la salida generada es 27, como se puede observar a continuación:



### 3. Implementar un método que permita obtener el producto de dos números

Lo primero que se debe hacer para el desarrollo y construcción del Método es:

- » Realizar la lectura del enunciado del problema y comprender cuál es el requerimiento.
- » Determinar entradas y salidas.
- » A partir de las entradas, se realizan los correspondientes procesos de forma lógica para obtener el resultado.
- » Retornar resultado si es necesario.

## 4.8 PRUEBA DE ESCRITORIO

Una prueba de escritorio es la comprobación lógica de un algoritmo (método), que brinda una alternativa de solución al problema.

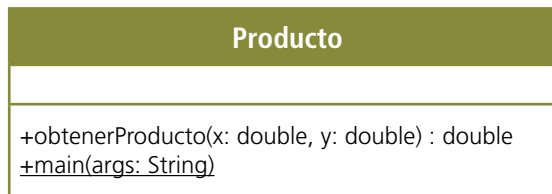
Para desarrollar la prueba de escritorio, se recomienda realizar el siguiente proceso:

- » Tomar datos específicos como entrada y seguir la secuencia indicada en el método hasta obtener un resultado.
- » Al realizar el análisis del o de los resultados de la prueba de escritorio se observan si estos son óptimos, quiere decir que el método está correcto; en caso contrario se tendrá que realizar los correspondientes ajustes.

### Prueba de escritorio Métodos obtenerProducto

NOMBRE MÉTODO	PARÁMETROS	RESULTADOS
obtenerProducto	double x, double y	Corresponde al resultado del producto de los dos números
	Supogan que x = 7; y = 4 x y 7 4	28

El programa para hallar la multiplicación de dos números e imprimir el resultado, está representado en el Diagrama de clases, que modela el producto de dos valores operados, y tiene implementados los métodos obtenerProducto() y main().



Y que codificado en Java es el siguiente:

```
public class Producto {    /*
    * Método que retorna el producto de dos números
    */
    public static double obtenerProducto(double x, double y) {
        return x * y;
    }

    /**
    * Método principal de la clase
    */
    public static void main(String[ ] args) {
        double producto=0.0;
        /*
        * Definición del objeto unaMultiplicacion de la clase
        * Producto
        */
        Producto unaMultiplicacion = new Producto( );
        /*
        * el objeto unaMultiplicacion llama al método
        * obtenerProducto
        */
        producto=unaMultiplicacion.obtenerProducto(7, 8);
        System.out.println("El producto de 7 y 8 es "+producto);
    }
}
```

El método main() es el punto de entrada de un programa en Java, siendo el primer

método que se ejecuta cuando se invoca a la máquina virtual de Java (JVM).

En el método `main()` se define el objeto `unaMultiplicacion` de la clase `Producto`, objeto que invoca al método `obtenerProducto`, el cual retorna el resultado que es almacenado en la variable `producto` para ser impreso en pantalla.

Como se puede observar el nombre de la clase (`Producto`) la primera letra esta en mayúscula, los métodos (`obtenerProducto`) y el objeto (`unaMultiplicacion`) inician con letras minúsculas, como tienen más de una palabra las demás inician con letra mayúscula.

Además es recomendable inicializar el valor de las variables por ejemplo `producto = 0.0`; porque es un dato de tipo doble porque si fuera `int` se hacia con un cero (`producto = 0`); un `String` se inicializa con comillas dobles (`" "`) `String nombre = " "`

## 4.9 MÉTODOS QUE NO RETORNAN VALOR

Depende del requerimiento o lógica con que se desarrolle el ejercicio. Cuando no es necesario algún valor devuelto, se declara con la instrucción `void`.

```
void nombre método() {
    bloque o conjunto de acciones
}
```

## 4.10 REQUERIMIENTOS

- » Son una descripción de las necesidades de un usuario para resolver un problema o alcanzar un objetivo.
- » Los requerimientos reflejan las necesidades de los clientes de un sistema que ayude a resolver algún problema.
- » Los requerimientos para un sistema de software determinan con precisión lo que hará el sistema y definen las restricciones de su operación e implementación, para satisfacer las necesidades planteadas, siendo una descripción detallada del software, la cual, puede servir como una base para diseño o implementación.

**Los requerimientos pueden ser:**



- Funcionales.
- No funcionales.

### Para la redacción de los requerimientos se debe tener en cuenta:

- Emplear el siguiente formato para expresar los requerimientos (formato Listado de requerimientos):

REQUERIMIENTO	EJEMPLO
R1 descripción	Ejemplo del R1
R2 descripción	Ejemplo del R2

- Utilizar un lenguaje de una manera consistente, que permitan diferenciar claramente entre requerimientos resaltando los aspectos importantes del requerimiento.
- Evitar en lo posible los términos técnicos.

Para la documentación de cada requerimiento se recomienda un formato como el siguiente:

Nombre	R1 nombre requerimiento 1
Descripción	Descripción puntal del requerimiento
Información de entradas	Valores de entrada
Información de salidas	Resultados o valores de salida

## 4.11 UN DESARROLLO COMPLETO

El Banco “El Ahorrador” requiere construir un programa que permita simular la apertura de las cuentas desde \$0 o con una determinada cantidad de dinero que el usuario disponga y además poder realizar las siguientes transacciones:

- » Depositar dinero.
- » Hacer retiros.
- » Obtener el saldo actual de la cuenta.

A partir del anterior enunciado determinar los requerimientos funcionales: Listado de los requerimientos del contexto planteado.

REQUERIMIENTO	EJEMPLO
R1 Apertura de cuentas con \$0	saldo = \$0
R2 Apertura de cuentas con una determinada cantidad de dinero.	saldo = \$50.000
R3 Consignar	Consignar una determinada cantidad y actualizar el valor del saldo.
R4 Retirar	Retirar una cuantía determinada de dinero y actualizar el saldo.
R5 Conocer el saldo actual	Saldo = \$15.000

Documentación de cada requerimiento:

<b>Nombre</b>	R1 apertura de cuenta con \$0
<b>Descripción</b>	Realizar la apertura de la cuenta con un valor de \$0 es una operación que se realiza una sola vez.
<b>Entradas</b>	El valor de cero.
<b>Salidas</b>	El saldo de la cuenta se inicializa con un valor de \$0.

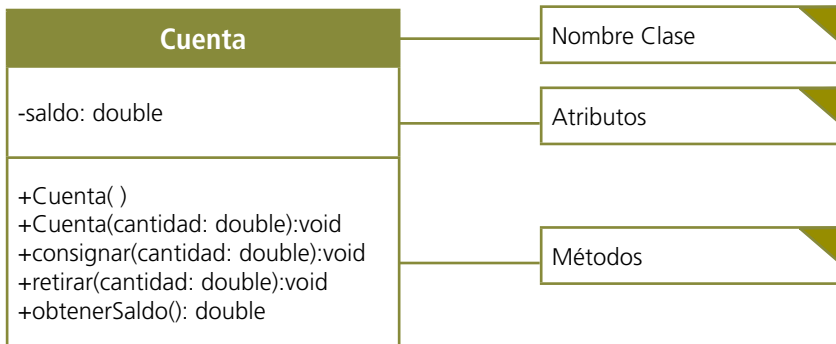
<b>Nombre</b>	R2 apertura de cuentas con una determinada cantidad de dinero.
<b>Descripción</b>	Se hace abre una cuenta con el valor determinado.
<b>Entradas</b>	Cantidad de dinero, determinada por el cliente para abrir la cuenta.
<b>Salidas</b>	El saldo de la cuenta se inicializa con el valor de la cuantía que el usuario determina.

<b>Nombre</b>	R3 Consignar.
<b>Descripción</b>	Valor que el cliente deposita en el banco e incrementa el valor del saldo.
<b>Entradas</b>	Valor consignado por el cliente.
<b>Salidas</b>	El valor del saldo incrementado con el valor de la consignación.

<b>Nombre</b>	R4 Retirar.
<b>Descripción</b>	Valor que el cliente retira del banco disminuyendo el valor del saldo.
<b>Entradas</b>	Valor del retiro.
<b>Salidas</b>	El nuevo valor del saldo disminuido con la cuantía de dinero retirada.

<b>Nombre</b>	R5 Conocer saldo actual.
<b>Descripción</b>	Valor del saldo actual.
<b>Entradas</b>	Solicitud del saldo.
<b>Salidas</b>	Impresión en pantalla del valor del saldo actual.

El siguiente gráfico corresponde al diagrama de clase la cual identificada y asociación de las características de la clase (atributos) y las operaciones que se van realizar (métodos).



## Atributos

Corresponde al saldo y es un dato de tipo double

NOMBRE DE LA CLASE	NOMBRE DEL ATRIBUTO	NOMBRE CODIFICACIÓN	OBJETIVO
Cuenta	saldo	saldo	Atributo que permite conocer el estado actual del saldo de la cuenta, y que se actualiza según sea el tipo de transacción (consignación o retiro).

## Métodos

A continuación se documenta los atributos, métodos de la clase Cuenta Dos constructores:

- » cuenta() método que permite la apertura de la cuenta con \$0; correspondientes por lo general a cuentas de nómina.

- » cuenta(double cantidad). Constructor que permite la apertura de la cuenta con un saldo o valor inicial; correspondiente a clientes del banco.
- » consignar(double cantidad); método que aumenta el valor del saldo cada vez que se realiza una consignación o en otras palabras cuando es invocado el método.
- » retirar(double cantidad); método que disminuye el valor del saldo según cantidad retirada.
- » double obtenerSaldo( ); método que retorna el valor del saldo actual existente en la cuenta.

### Documentación Métodos

#### Análisis

#### Entrada

#### Salida



- » Entrada corresponde al valor de la consignación.
- » No hay retorno de datos ya que el valor de la cantidad se suma al valor del saldo.

<b>Nombre del Método</b>	consignar.
<b>Entrada (lista de Parámetros)</b>	double cantidad.
<b>Resultado (tipo de dato de retorno)</b>	No hay retorno.
<b>Solución planteada</b>	El valor del saldo se aumenta según cantidad consignada.
<b>Firma del Método</b>	consignar(double cantidad).

### Implementación del método

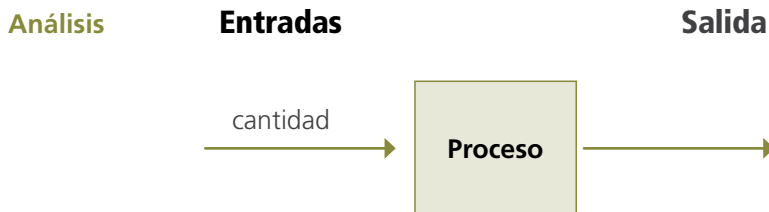
```
public void consignar(double cantidad) {
    saldo = saldo + cantidad;
}
```

Prueba de escritorio Método consignar

NOMBRE MÉTODO	PARÁMETROS	RESULTADOS
consignar	double cantidad	Suponga que se tiene un saldo = \$0
	Cantidad 100	saldo 100
	Cantidad 250	saldo 350

Método que permite realizar los retiros en la cuenta creada.

### Documentación Métodos



- » entrada corresponde al valor del retiro
- » no hay retorno de datos ya que el valor de la cantidad se resta al valor del saldo

Documentación del método retirar elaborado en la siguiente tabla.

<b>Nombre del Método</b>	retirar
<b>Entrada (lista de Parámetros)</b>	double cantidad
<b>Resultado (tipo de dato de retorno)</b>	ninguno
<b>Solución planteada</b>	El valor del saldo disminuye según valor de la transacción
<b>Firma del Método</b>	retirar(double cantidad).

### Implementación del método

```
public void retirar(double cantidad) {
    saldo = saldo - cantidad;
}
```

Prueba de escritorio Método retirar

NOMBRE MÉTODO	PARÁMETROS	RESULTADOS
retirar	double cantidad	Suponga que se tiene un saldo = \$350
	Cantidad 100	saldo 250

Implementación de la clase Cuenta en el lenguaje de programación de Java.

```
/**
 * Clase que modela la apertura de la cuenta y los movimientos de
 * la consignación y retiro de la misma
 * @author Miguel Hernández
 * @version 1.0
 */
public class Cuenta {
    private double saldo;
    /**
     * Se crea una cuenta con un saldo de $0
     */

    public Cuenta( ) {
        this.saldo=0;
    }

    /**
     * Se crea una cuenta con un saldo de acuerdo al valor de
     la apertura.
     */
    public Cuenta(double cantidad) {
        this.saldo=cantidad;
    }

    /**
     * Se incrementa el valor del saldo según valor de la
     consignación.
     * @param cantidad
     */

    public void consignar(double cantidad) {
        saldo = saldo + cantidad;
    }
    /**
     * Se disminuye el valor del saldo según valor del retiro
     * @param cantidad
     */
    public void retirar(double cantidad) {
        saldo = saldo - cantidad;
    }
}
```



```

/**
 *
 * Método que retorna el valor del saldo de la cuenta
 * @return saldo
 */
public double obtenerSaldo()
{
    return saldo;
}

```

## 4.12 ENTRADA Y SALIDA DE DATOS

Las Aplicaciones o programas desarrollados para las computadoras operan a partir de un conjunto de datos de entrada, con los cuales se realizan procesos y cálculos generando algún tipo de salida.

La forma de representar la entrada y salida de datos en Java es en base de streams (flujos de datos). Un stream es una conexión entre el programa y la fuente o destino de los datos. La información es procesada en serie, un carácter a continuación de otro, a través de esta conexión. Por ejemplo, cualquier dato que sea leído o escrito en un dispositivo de consola, de archivo, red, memoria es un stream, de los datos que pueden leerse o escribirse se encuentran: los caracteres, los dígitos, los objetos de java, los sonidos, las imágenes, entre otros.

### Flujo (Stream)



## 4.13 ENTRADA Y SALIDA ESTÁNDAR

En Java, la entrada por teclado y la salida en pantalla se establece a partir de la clase `System`. Esta clase agrupa y define métodos y objetos que tienen relación con el sistema local, para la lectura e impresión de datos.

**System.in:** Objeto de la clase `InputStream` definido para tomar datos desde la entrada estándar del sistema, que habitualmente es el teclado.

**System.out:** Objeto de la clase `PrintStream` que imprimirá los datos en la salida estándar del sistema, correspondiente a la pantalla donde el atributo `out` de la clase `System` tiene implementado el método `println` que integran el conocido `System.out.println`.

Para leer datos desde teclado se utilizan clases que se encuentran disponibles en Java dentro de las que están:

**La clase `Scanner`** permite definir objetos, para leer información desde una fuente de datos que puede ser un archivo, una cadena de caracteres, o el teclado, entre otros.

Para hacer uso de los métodos que implementa la clase `Scanner` es necesario declarar una variable (objeto), de este tipo de dato se asocia con el teclado (`System.in`).

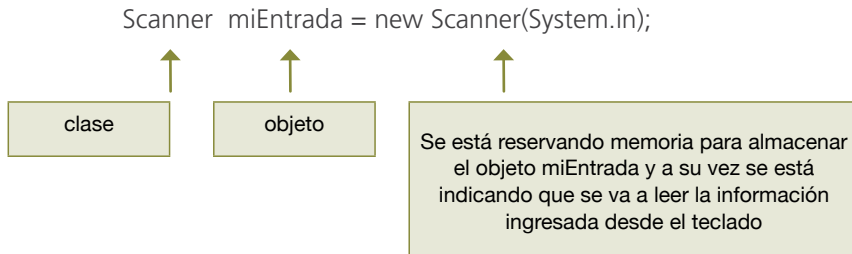
La sintaxis para definir un objeto de la clase `Scanner` es el siguiente:

```
Scanner nombre del objeto = new Scanner(System.in);
```

### Por ejemplo

- » `Scanner miEntrada = new Scanner(System.in);`
- » `Scanner miTeclado = new Scanner(System.in);`
- » `Scanner input = new Scanner(System.in);`
- » `Scanner in = new Scanner(System.in);`

## Creación de un objeto de la clase Scanner



Una vez definido el objeto, se puede invocar cualquiera de los métodos de la clase Scanner dentro de los que están:

- » `nextInt( )`: obtiene un número entero.
- » `nextDouble( )`: obtiene un número real.
- » `nextLine( )`: obtiene una cadena de caracteres.
- » `next( )`: obtiene una cadena de caracteres hasta encontrar un espacio.
- » `nextShort( )`: para leer datos de tipo short.
- » `nextFloat( )`: para datos de tipo float.
- » `nextLong( )`: para datos de tipo long.
- » `nextByte()`: para datos de tipo byte.

No existen métodos para obtener datos de tipo boolean, ni para datos de tipo char, para este último caso se puede utilizar `nextLine().charAt(0)`.

### Ejemplo

Programa que calcula el producto de dos números enteros o reales (números que pueden o no tener fracción decimal), donde los valores son ingresados por teclado.

Siguiendo el mismo modelo del programa anterior, lo que se de realizar es:

- » La lectura de los datos se define un objeto de la clase Scanner.
- » `Scanner input = new Scanner(System.in);`
- » Se definen las variables, `double multiplicando` y `double multiplicador`.
- » Se realiza la lectura de los datos, `multiplicando=input.nextDouble()`.

Para utilizar la clase Scanner se debe digitar al inicio del programa; **import** java.util.Scanner; que significa importar la clase del paquete java útil.

## Ejemplo

El siguiente código de java corresponde a una clase que modela operaciones básicas de suma, resta, producto y división.

```
/**
 * Clase que modela las operaciones aritméticas básicas
 * @author (Miguel Hernández Bejarano)
 * @version 1.0
 */
public class Calculadora {
    private double x;
    private double y;

    public Calculadora() {
        this.x = 0;
        this.y = 0;
    }

    public Calculadora(double x, double y) {
        this.x = x;
        this.y = y;
    }

    public double getX() {
        return x;
    }

    public void setX(double x) {
        this.x = x;
    }
}
```

```
public double getY() {
    return y;
}

public void setY(double y) {
    this.y = y;
}

public double obtenerSuma(){
    return x+y;
}
public double obtenerResta(){
    return x-y;
}

public double obtenerProducto(){
    return x*y;
}

public double obtenerDivision(){
    return x/y;
}
}
```

---

La sentencia return se utiliza para finalizar el método y retornar el resultado del correspondiente proceso.

```
import java.util.Scanner;

public class PruebaCalculadora {

    //se define el objeto cassio
    private Calculadora cassio;

    public PruebaCalculadora() {
        //se instancia el objeto cassio de la clase operación
        this.cassio=new Calculadora();
    }
}
```

```

public void obtenerOperaciones(){
    double a;
    double b;
    Scanner lector = new Scanner(System.in);
    double resultado;
    //se leen los datos por teclado
    System.out.print("Digite Numero : ");
    a= lector.nextDouble();
    System.out.print("Digite Numero : ");
    b= lector.nextDouble();
    //actualizacion del estado del objeto cassio
    cassio.setX(a);
    cassio.setY(b);

    resultado=cassio.obtenerSuma();
    System.out.println("Resultado Suma : "+resultado);
    resultado=cassio.obtenerResta();
    System.out.println("Resultado Resta : "+resultado);
    resultado=cassio.obtenerProducto();
    System.out.println("Resultado Producto : "+resultado);
    resultado=cassio.obtenerDivision();
    System.out.println("Resultado División : "+resultado);

}

public static void main(String[] args) {

    PruebaCalculadora unaCalculadora =new PruebaCalcu
    ladora();
    unaCalculadora.obtenerOperaciones();

}
}

```

## La clase `BufferedReader`

Es una clase derivada que permite la lectura o introducción de datos por medio del teclado, para utilizar esta clase se debe importar los paquetes:

- » `BufferedReader`.
- » `InputStreamReader`.
- » `IOException`.

### Definición de un objeto de la clase `BufferedReader`:

```
BufferedReader br = new BufferedReader (new InputStreamReader  
(System.in));
```

- `br` es el nombre del objeto.
- La clase `InputStreamReader` establece el paso del flujo de bytes a flujos de caracteres.
- El uso del `BufferedReader` requiere forzosamente del manejo de excepciones por ello también es necesario importar la clase `IOException`. (el tema de excepciones será abordado en un próximo curso).

## Ejemplo

### Definición de un objeto

```
BufferedReader br = new BufferedReader (new InputStreamReader  
(System.in));
```

### Ingreso de datos

Lectura de datos por teclado y almacenado en una variable.

```
String valor = br.readLine();
```

La lectura de datos por teclado corresponde a cadenas de texto, por lo tanto `valor` es de tipo `String`, por ello la asignación de la lectura se realiza de manera directa a la variable.

Cuando se requiere leer valores numéricos (enteros o con fracción decimal), se debe realizar una “conversión” de la cadena leída a número, por ejemplo si nosotros introducimos por teclado 123, el programa lo tomará como texto y no como número, por lo que si queremos hacer operaciones con esos números debemos convertir la lectura a entero con el método “parse” y almacenarla en una variable del mismo tipo, por ejemplo:

```
int dato;
dato = Integer.parseInt(br.readLine());
```

La estructura para realizar el proceso de entrada de datos por teclado es

```
BufferedReader teclado = new BufferedReader(new InputStreamReader(System.in));
```

## IOException

### Ejemplo

Programa que a partir del radio, calcula el diámetro, longitud y área de la circunferencia. Con el objeto de facilitar su implementación y mantenimiento, las aplicaciones se dividen en módulos, para este ejercicio se diseñan cuatro métodos, tres para cumplir con los requerimientos; de obtener el diámetro, obtener la longitud, obtener área y el método main de la clase.

Método para calcular la longitud de la circunferencia, a partir del radio.

<b>Nombre del Método</b>	obtenerLongitud
<b>Entrada (lista de Parámetros)</b>	ninguno
<b>Resultado (tipo de dato de retorno)</b>	double
<b>Solución planteada</b>	Se multiplica el valor del radio por 2; por la constante PI y se retorna el resultado
<b>Firma del Método</b>	obtenerLongitud( int radio )



Entradas

Salida

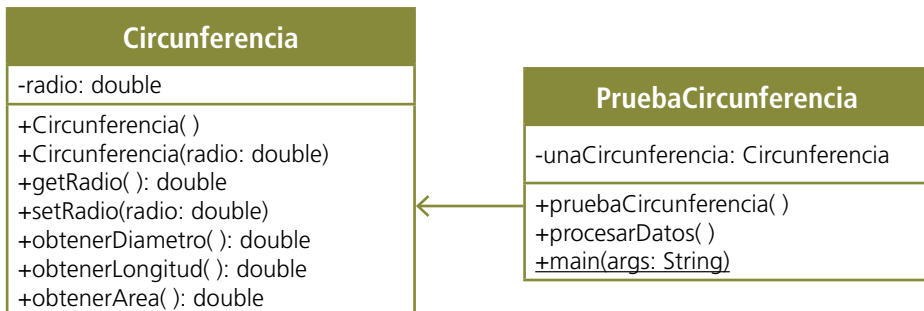


```

public double obtenerLongitud() {
    return 2 * Math.PI * radio;
}
  
```

- » Math es una clase de java y
- » PI, corresponde al valor de la constante de pi

### Diagrama de clases



```

public class Circunferencia {
    private double radio;

    public Circunferencia() {
        this.radio = radio;
    }

    public Circunferencia(double radio) {
        this.radio = radio;
    }

    public double getRadio() {
        return radio;
    }
}
  
```

```

    public void setRadio(double radio) {
        this.radio = radio;
    }

    /**
     * método que calcula y devuelve el diámetro
     */
    public double obtenerDiametro() {
        return 2 * radio;
    }

    /**
     * método que calcula y retorna longitud de la circunferencia
     */
    public double obtenerLongitud() {
        return 2* Math.PI * radio;
    }

    /**
     * método que calcula y retorna el area de la circunferencia
     */

    public double obtenerArea() {
        return Math.PI * Math.pow(radio,2) ;
    }

}

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;

public class PruebaCircunferencia {
    /**
     * Definición del objeto de la clase Circunferencia
     */
    private Circunferencia unaCircunferencia;

    public PruebaCircunferencia() {
        this.unaCircunferencia = new Circunferencia();
    }
}

```

```

/**
 * Metodo que realiza la lectura e impresion de los re
sultados
 * @throws IOException
 */

public void procesarDatos() throws IOException {
    double radio = 0;
    String numero = "";
    BufferedReader teclado = new BufferedReader(new
InputStreamReader(System.in));
        System.out.println("Digite el valor del Radio:");
        numero = teclado.readLine();
        radio = Double.parseDouble(numero);
        unaCircunferencia.setRadio(radio);
    System.out.println("Valor Radio: "+ unaCircunferen
cia.getRadio()+ "\nDiametro "+unaCircunferencia.
obtenerDiametro()+
"\nLongitud Circunferencia:
"+unaCircunferencia.obtenerLongitud()+
"\nÁrea Circunferencia: "+ unaCircunferencia.obte
nerArea());
}

public static void main(String[] args) throws IOException
{
    PruebaCircunferencia prueba = new PruebaCircunfe
rencia();
    prueba.procesarDatos();
}
}

```

## La clase Math

Representa la librería matemática de Java y tiene una serie de métodos definidos, que permiten obtener el cálculo de las operaciones que se usan cotidianamente como: raíz cuadrada, valor absoluto, logaritmo, potencias, funciones trigonométricas, entre otros.

## Métodos

- » `Math.abs( x )` para `int`, `long`, `float` y `double`.
- » `Math.sin( double )` Calcula el seno de un dato tipo `double`.
- » `Math.cos( double )` Calcula el coseno de un `double`.
- » `Math.tan( double )` Calcula la tangente.
- » `Math.pow( a,b )` Eleva el número `a` a la potencia `b`.
- » `Math.round( x )` para `double` y `float`.
- » `Math.random()` devuelve un `double` aleatorio.
- » `Math.log( double )` Devuelve el logaritmo de base `e` del valor.
- » `Math.sqrt( double )` Devuelve el redondeo de la raíz cuadrada del valor.

## Constantes

- » `Math.E` para la base exponencial.
- » `Math.PI` para `PI`.

## Ejemplo

Método que a partir de la base y el exponente, retorna el resultado de la potencia.

```
public double obtenerLongitud(int base, int exponente) {  
    return Math.PI * Math.pow(base,expo) ;  
}
```

Suponga que la base es 2 y el exponente 5 el valor retornado es 32.

## 4.14 LECTURAS RECOMENDADAS

- » Métodos.
- » Parámetros.
- » Métodos de la clase.
- » Método del objeto.
- » Sentencia return.
- » Sentencia void.
- » Clase String.
- » Sobrecarga de constructores.
- » Sentencias this y return.
- » Encapsulamiento.

## 4.15 PREGUNTAS DE REVISIÓN DE CONCEPTOS

- » ¿Qué Diferencia hay entre los métodos set y un constructor?
- » ¿Cuál es la importancia de implementar los métodos set y get?
- » ¿Qué es un constructor por defecto?
- » Importancia de los métodos
- » ¿Cuántas entradas puede tener un método?
- » ¿Cuántos valores puede retornar un método?
- » ¿Cómo realizar la lectura de un dato de tipo char?

## 4.16 EJERCICIOS

- » Elaborar diagrama de clases y la codificación del constructor, los métodos set y get; para cada uno de los siguientes enunciados:
  - Se requiere modelar la información de un libro conformada, por el título, editorial, número de páginas y el autor.
  - Modelar la información correspondiente al teléfono, conformada por la referencia, costo, marca, modelo.
- » Construir la clase Docente, la cual estará integrada por los siguientes atributos:

VISIBILIDAD	NOMBRE ATRIBUTO	TIPO
private	nombre	String
private	documento	long
private	teléfono	String
private	Email	String

- » Crear uno o más constructores para la clase.
- » Crear los métodos get y set para los distintos atributos de la clase.
- » Elaborar una clase que implemente los métodos para obtener, la raíz cuadrada, el logaritmo en base 10, logaritmo en base e.
- » Diseñar una clase que provea los métodos para las funciones trigonométricas de seno, coseno y tangente; expresados en radianes.

## 4.17 REFERENCIAS BIBLIOGRÁFICAS

Aguilar, L. J. (2004). *Fundamentos de programación algoritmos y estructura de datos*. Tercera Edición. Ciudad: México: McGraw Hill.

Booch, G. (1996). *Análisis y diseño orientado a objetos con aplicaciones*. Ciudad: México Addison Wesley.

Deitel y Deitel. (2012). *Cómo programar en Java*. Ciudad: México Editorial Pearson (Prentice Hall).

Villalobos, J., y Casallas, R. (2006). *Fundamentos de programación, aprendizaje activo basado en casos*. Ciudad: México Editorial Pearson.



# CAPÍTULO 5

## CONDICIONALES





## 5.1 TEMÁTICA A DESARROLLAR

- » Estructuras de control.
- » Condicionales.
- » Operadores relacionales.
- » Operadores lógicos.
- » if – else.
- » switch-case.
- » Operador condicional ( ? ).

## 5.2 INTRODUCCIÓN

Las estructuras de control condicionales permiten realizar la comparación de un atributo o variable con otro u otros valores, y con base en el resultado de esa evaluación se siguen determinadas acciones dentro del programa.

Las instrucciones que definen los condicionales en Java son, el if – else, el switch - case y el operador condicional ( ? o alternativa).

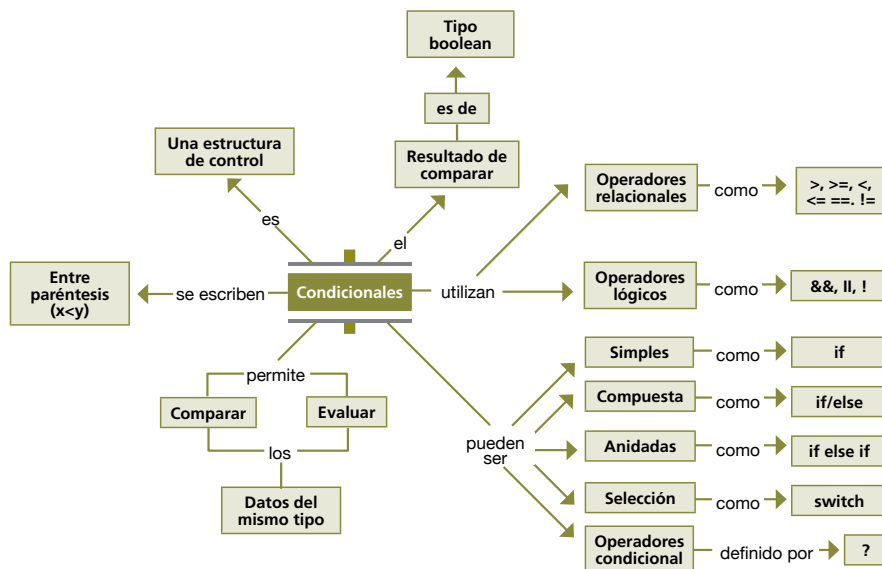
La sentencia if evalúa la expresión de la condición en caso de ser verdadera ejecuta las instrucciones que están dentro de su bloque de if.

Estructura de selección if - else permite especificar una acción a realizar cuando la condición es verdadera y otra distinta cuando la condición es falsa. Es decir ejecuta el bloque if si la condición se cumple, en caso contrario se ejecuta el bloque de instrucciones del else.

El operador condicional ( ? ) evalúa una expresión booleana y determina cuál de las acciones se debe ejecutar en base al resultado de la comparación.

La siguiente figura amplía el concepto de las condicionales en Java, incluyendo el uso de operadores:

Figura 19. Concepto de las Estructuras condicionales.



En el desarrollo del capítulo y a través de ejemplos concretos se utilizan buenas prácticas de programación que van desde el modelado de la situación planteada a la solución en Java, incluyendo estilos programación dentro del código fuente como la indentación, el uso de los constructores, la identificación de los métodos, la declaración y nombre de las variables locales, el uso obligatorio de los corchetes, los espacios después de una palabra clave y antes del corchete, entre otros aspectos.

### 5.3 ESTRUCTURAS DE CONTROL CONDICIONALES

Son elementos del lenguaje de programación que permiten tomar decisiones en un programa, las estructuras de selección permiten tomar distintos cursos de acción de acuerdo a una condición, que viene dada por una expresión lógica. Las condiciones hacen uso de los operadores relacionales y/u operadores lógicos.

#### Ejemplo:

- »  $(X \geq Y)$
- »  $(X \geq Y) \ \&\& \ (X \geq Z)$
- »  $(A \neq 0) \ || \ (A \neq 5)$

### 5.4 OPERADORES RELACIONALES

Son símbolos que se usan para establecer una relación para evaluar dos o más expresiones y/o valores (combinación de operadores y variables) donde el resultado de la comparación puede ser verdadero o falso.

La tabla 7 presenta los operadores relacionales utilizados en Java:

Tabla 7. Operadores relacionales.

OPERADOR	NOMBRE	UTILIZACIÓN	RESULTADO
>	Mayor que	$X > Y$	verdadero si X es mayor que Y
>=	Mayor o igual a	$X \geq Y$	verdadero si X es mayor o igual que Y
<	Menor que	$X < Y$	verdadero si X es menor que Y
<=	Menor o igual a	$X \leq Y$	verdadero si X es menor o igual que Y
==	Igual a	$X == Y$	verdadero si X es igual a Y
!=	Diferente de	$X != Y$	verdadero si X es distinto de Y

Ejemplo:

```
» 7 > 2    /* true */
» 3 >= 5   /* false */
» 14 == 14  /* true */
» 32 != 32  /* false */
```

## 5.5 OPERADORES LÓGICOS

Permiten construir expresiones lógicas, Java tiene los operadores lógicos definidos Y, O, NO, los argumentos y resultados de estos operadores son de tipo booleano. En la tabla 8 se detallan estos operadores y se complementa con un ejemplo.

**Tabla 8.** Operadores lógicos.

OPERADOR	NOMBRE	DESCRIPCIÓN	EJEMPLO	RETORNA
&&	AND (y)	Conjunción	<code>c &gt; 7 &amp;&amp; c &lt;= 10</code>	Retorna true si ambos operandos son verdaderos (true).
	OR (o)	Disyunción	<code>x == 1    y == 2</code>	Retorna true si alguno de los operandos es verdadero (true).
!	NOT (no)	Negación	<code>! x</code>	Niega el operando que se le pasa.

Estos se conocen con el nombre de conjunción, disyunción y negación. En la tabla 9 se presentan los resultados de aplicar los operadores lógicos para los posibles valores de entrada.

**Tabla 9.** Resultado operadores lógicos.

X	Y	X && Y	X    Y	!X
false	false	false	false	true
false	true	false	true	true
true	false	false	true	false
true	true	true	true	false

## Ejemplo

- » `(true || false) && true // true`
- » `(false || false) && (!true) //false`
- » `!true && !false //false`
- » `!(4 > 8) || (9 > 0) && (4 == 2) //false`

Los operadores relacionales y lógicos son fundamentales para implementar las condiciones de las sentencias de control.

## 5.6 TIPOS DE CONDICIONALES

### 5.6.1 Condicionales simple

Las sentencias condicionales permiten comparar el valor de una variable con otra, con valores numéricos y con base en el resultado de esta comparación, se sigue un curso de acción dentro del programa. La sentencia que permite llevar a cabo la ejecución condicional es la palabra reservada `if`.

Si al evaluar la expresión el resultado es verdadero entonces se ejecuta(n) las sentencias u operación(es) del bloque del `if`.

La notación para esta sentencia es la siguiente:

```
if ( Expresion boolean ) {
    instrucciones;
}
```

### Por ejemplo

- » `If ( x > y)` compara si `x` es mayor que `y`.
- » `If ( a == b)` se evalúa si la variable `a` es igual a la variable `b`.
- » `If ( x != VALOR_MAXIMO)` se evalúa si `x` es diferente al valor de la constante.

## Evaluar intervalos

La expresión del condicional puede utilizar los operadores relacionales y lógicos.

### Ejemplos:

1. Para evaluar si un valor se encuentra entre un intervalo cerrado y el otro abierto (1.0,5.0] La condición sería:

```
if (a>1.0 && a<=5.0)
```

» Al evaluar si el intervalo no se encuentre entre 6 y 10

```
if (a<=6.0 || a>10)
```

» Negación ( ! ) del valor de un intervalo

```
if (!(a>1 && a<=10))
```

Suponga que a vale 8, la expresión interna de comparar es verdadera ya que 8 es mayor que 1 y menor que 10; al negar esta expresión queda falsa.

Los condicionales le permiten validar datos.

### Ejemplos:

1. En la operación de la división, el divisor no puede ser cero se evalúa si este es igual a cero, para lo cual se imprime el mensaje: error dividir por cero, en caso contrario, se ejecuta la sentencia `cociente = dividendo / divisor`;

```
if ( divisor == 0 ) {  
    System.out.println("error dividir por cero");  
} else {  
    cociente = dividendo / divisor;  
}
```

2. Otra forma de realizar el anterior proceso se da al cambiar el operador de la condición en vez de igual, se utiliza el operador diferente ( `divisor != 0` ) al ser verdadera la condición se ejecuta la sentencia `cociente = dividendo / divisor`; en caso contrario se imprime el mensaje error dividir por cero.

```

if ( divisor != 0 ) {
    cociente = dividendo / divisor;
} else {
    System.out.println("error dividir por cero");
}

```

---

3. Para calcular la raíz cuadrada de un número este no puede ser menor que cero, se evalúa si el número es menor que cero, en la sentencia `if`, al ser verdadero se imprime el mensaje raíz imaginaria, en caso contrario se ejecuta la sentencia, `raiz = Math.sqrt( numero );`

```

if ( numero < 0 ) {
    System.out.println("raíz imaginaria");
} else {
    raiz = Math.sqrt( numero );
}

```

Si la variable `numero` es mayor e igual que cero, se ejecuta la sentencia, `raiz = Math.sqrt( numero );` en caso contrario se imprime el mensaje raíz imaginaria.

```

if ( numero >= 0 ) {
    raiz = Math.sqrt( numero );
} else {
    System.out.println("raíz imaginaria");
}

```

---

4. Programa que permite calcular el descuento de 5%, para compras superiores a \$100.000.

La aplicación se modela a partir de las clases `Compra`, que tiene como responsabilidad la lectura del valor de la compra e impresión de los valores de la compra, el descuento y el total a pagar.

La clase `Descuento`, permite modelar los cálculos del descuento y el total a pagar.

En Java, se recomienda definir constantes en vez de utilizar valores dentro de los programas como por ejemplo.

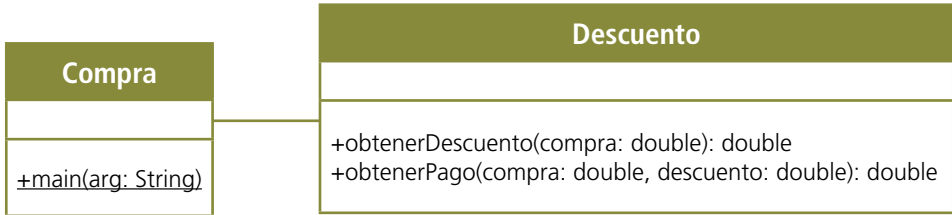
```

final double VALOR_COMPRA_DESCUENTO = 100000;
final double VALOR_DESCUENTO = 0.05;

```



## Diagrama de clases



La entrada de datos del método se da a partir de la compra y la salida es el descuento que depende del monto de la compra.

## Entradas

## Salida



Si la compra es menor que cien mil no tiene descuento, en caso contrario se calcula el descuento, como apoyo para la implementación del método, se utiliza la tabla 10.

**Tabla 10.** Apoyo implementación del método.

Nombre del Método	obtenerDescuento
Entrada (lista de Parámetros)	ninguno
Resultado (tipo de dato de retorno)	double valorCompra
Solución del problema	Se evalúa si la compra es mayor o igual que cien mil, para calcular el descuento
Firma del Método	obtenerDescuento ( )

De la anterior tabla se puede construir la cabecera del método:

Tipo retorno	Nombre del Método	Lista parámetros
double	obtenerDescuento	( )

```
if ( compra >= VALOR_COMPRA_DESCUENTO) {
    descuento = compra * VALOR_DESCUENTO;
}
```

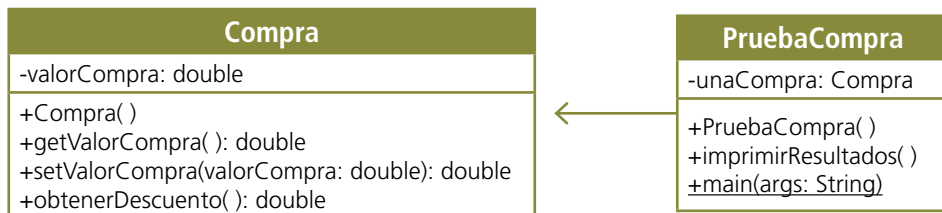
Las variables se inicializan cuando se definen:

```
double descuento = 0.0;
```

La comparación `compra >= VALOR_COMPRA_DESCUENTO` es una expresión booleana.

La sentencia `descuento = compra * VALOR_DESCUENTO;` se ejecuta solamente si la expresión booleana es verdadera.

### Diagrama de Clases



Codificación en Java del modelado del problema, a partir del diagrama de clases:

```
public class Compra {
    private double valorCompra;

    public Compra() {
        this.valorCompra = 0;
    }

    public double getValorCompra() {
        return valorCompra;
    }
}
```

```
public void setValorCompra(double valorCompra) {
    this.valorCompra = valorCompra;
}

public double obtenerDescuento() {
    double descuento = 0.0;
    final double VALOR_COMPRA_DESCUENTO = 100000;
    final double VALOR_DESCUENTO = 0.05;
    if ( valorCompra >= VALOR_COMPRA_DESCUENTO) {
        descuento = valorCompra * VALOR_DESCUENTO;
    }
    return descuento;
}
}
```

### 5.6.2 Condicionales compuestas

Están conformadas por un `if` y un `else` que permiten optar entre dos opciones o alternativas posibles en función del cumplimiento o no de una determinada condición.

La sintaxis de una sentencia condicional `if else` es la siguiente:

```
if (expresion_booleana) {
    sentencia_de_verdadero;
}
else {
    sentencia_de_falso;
}
```

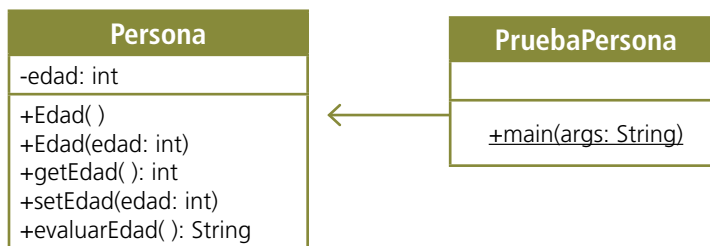
### Ejemplos

1. Aplicación que permite determinar a partir de la edad de una persona, si es mayor o menor de edad.

Se construyen dos clases, en la clase Persona se implementa el método que evalúa si es mayor o menor de edad y en clase PruebaPersona se lee la edad de la persona.

```
edad = lea.nextInt();
```

## Diagrama de Clases



```

if ( edad >= 18 ) {
    resultado = "Es una persona mayor de Edad";
} else {
    resultado = "Es una persona menor de edad";
}
  
```

- » Si la condición ( edad >= 18 ) es verdadera se ejecuta la sentencia; resultado = "Es una persona mayor de edad"; en caso contrario se ejecuta la sentencia resultado = "Es una persona menor de edad";
- » Ambas sentencias (if o else) no se ejecutaran al mismo tiempo, esto quiere decir que son excluyentes.

Codificación en Java del modelado del problema, a partir del diagrama de clases:

```
/**
 * Clase que modela la edad de la persona
 */
public class Persona {
    private int edad;
    public Persona() {
        this.edad = 0;
    }

    public Persona(int edad) {
        if (edad > 0){
            this.edad = edad;
        } else {
            this.edad = 0;
        }
    }

    public int getEdad() {
        return edad;
    }

    public void setEdad(int edad) {
        this.edad = edad;
    }

    public String evaluarEdad () {
        String resultado = "";
        //Este es un condicional que compara números
        if ( edad >= 18 ) {
            resultado = "Es una persona mayor de Edad";
        }else{
            resultado = "Es una persona menor de edad";
        }
        return resultado;
    }
}
```

```
import java.util.Scanner;

public class PruebaPersona {
    public static void main (String arg[]) {
        Scanner lea = new Scanner(System.in);
        Persona unaPersona = new Persona();
        int edad = 0;
        System.out.print("Digite edad de la persona ");
        edad = lea.nextInt();
        unaPersona.setEdad(edad);
        System.out.println(unaPersona.evaluarEdad()+" tiene
        "+unaPersona.getEdad()+" años");
    }
}
```

### 5.6.3 Condicionales anidadas

Consisten en que dentro de una sentencia if, se pueden colocar otra sentencia if, en ocasiones se requiere anexar un if a continuación de otro if, o un if - else. En otras palabras un condicional anidado es un bloque condicional dentro de otro como lo señala la siguiente notación:

```
if (condición1) {
    instrucción1;
} if (condición2) {
    instrucción2;
} if (condición3) {
    instrucción3;
}
...
else {
    instrucciones del else
}
}
```

- » Dentro del bloque de sentencias del if (o de los bloques de if-else) puede encontrarse otra sentencia if.
- » Las llaves ({} ) son necesarias para agrupar las sentencias múltiples.
- » Es conveniente minimizar el número de anidaciones.
- » Se debe revisar el código de las condiciones para evitar errores

La siguiente notación muestra la forma general de su empleabilidad:

```
if (condición1) {  
    instrucción1;  
} else if (condición2) {  
    Instrucción2;  
} else if (condición3) {  
    Instrucción3;  
}  
...  
else {  
    instrucciones del else  
}  
}
```

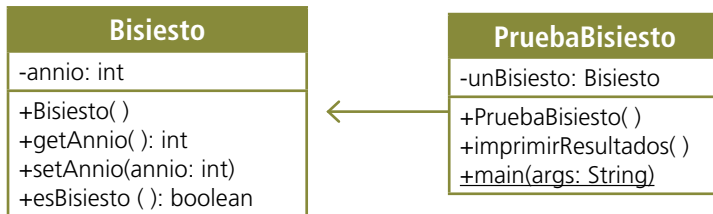
## Ejemplos

1. Programa que determina si un año es o no bisiesto.

Aplicación diseñada con las clases Bisiesto que implementa el método para evaluar el año y la clase prueba bisiesto que tiene como responsabilidad la lectura del año e impresión del si es o no un año bisiesto, en esta clase se define: Un objeto unBisiesto de la clase Bisiesto: Bisiesto unBisiesto = **new** Bisiesto(); El objeto unBisiesto invoca al método esBisiesto (unBisiesto.esBisiesto( )); instrucción incluidas dentro del condicional (**if** ( unBisiesto.esBisiesto( ))); ya que el método retorna un valor booleano de true o false.

Son bisiestos los años múltiplos de 4; que no múltiplos de 100 y que sean múltiplos de 400.

## Diagrama de Clases



Codificación en Java del modelado del problema si un año es bisiesto a partir del diagrama de clases:

```

public class Bisiesto {
    private int año;

    public Bisiesto() {
        this.año = 2015;
    }

    public int getAño() {
        return año;
    }

    public void setAño(int año) {
        this.año = año;
    }

    public boolean esBisiesto( ) {
        boolean bisiesto = false;
        if (año % 4 == 0) {
            if ((año % 100 != 0) || (año % 400 == 0)) {
                bisiesto=true;
            } else {
                bisiesto=false;
            }
        } else {
            bisiesto=false;
        }
        return bisiesto;
    }
}
  
```



La clase PruebaBisiesto queda así:

```
import java.util.Scanner;

public class PruebaBisiesto {
    private Bisiesto unBisiesto;

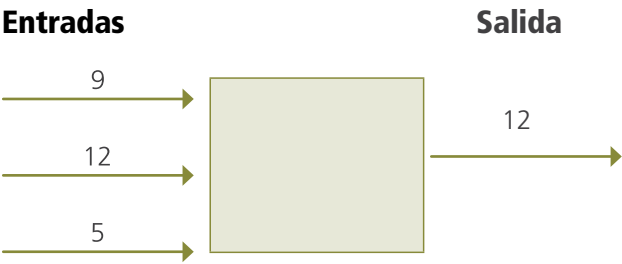
    public PruebaBisiesto() {
        this.unBisiesto = new Bisiesto();
    }

    public void imprimirResultados() {
        int aa = 0;
        System.out.print("Digite el año: ");
        Scanner lea = new Scanner(System.in);
        aa = lea.nextInt();
        unBisiesto.setAnio(aa);
        if(unBisiesto.esBisiesto()){
            System.out.println(unBisiesto.getAnio()+ "
            es un año bisiesto ");
        } else {
            System.out.println(unBisiesto.getAnio()+ "
            No es un año bisiesto ");
        }
    }

    public static void main(String[] args) {
        PruebaBisiesto unaPrueba = new PruebaBisiesto();
        unaPrueba.imprimirResultados();
    }
}
```

2. Método que evalúa tres números enteros diferentes y retorna el mayor de estos.

Etapas de análisis, son tres datos de entrada y una salida; por ejemplo ingresa el 9, 12, 5; la salida es 12, que corresponde al número mayor.



Para que el ejercicio sea más general y pueda ser aplicado a diferentes valores se utilizan tres variables *x*, *y*, *z* de tipo entero que corresponden a las entradas de datos.

Como apoyo a la parte de análisis se completa en la tabla 11.

Tabla 11. Apoyo parte de análisis.

Nombre del Método	obtenerMayor
Entrada (lista de Parámetros)	ninguno
Resultado (tipo de dato de retorno)	Int
Planteamiento del problema	Comparar las variables para determinar el mayor <ul style="list-style-type: none"><li>• Si <math>(x &gt; y)</math> y <math>(x &gt; z)</math>, el mayor es <i>x</i></li><li>• Si <math>(y &gt; x)</math> y <math>(y &gt; z)</math>, el mayor es <i>y</i></li><li>• Si <math>(z &gt; x)</math> y <math>(z &gt; y)</math>, el mayor es <i>z</i></li></ul>

La firma del método sería:

Nombre del Método  
obtenerMayor

Lista parámetros  
( )

La implementación del método sería:

```
public int obtnerMayor( ) {
    int mayor = 0;
    if ( x >= y && x >= z ) {
        mayor = x;
    } else if( y >= x && y >= z ) {
        mayor = y;
    } if( z >= x && z >= y ) {
        mayor=z;
    }
    return mayor;
}
```

En la prueba de escritorio de este método se realiza el seguimiento para determinar si el método es correcto, o sino para realizar los ajustes correspondientes.

Suponiendo que  $x = 12$ ;  $y = 9$ ;  $z = 5$ , la primera condición a evaluar es si  $x$  es mayor que  $y$ , como es verdadero, evalúa si  $x$  es mayor que  $z$ ; esto lo realiza la sentencia **if** ( $x \geq y \ \&\& \ x \geq z$ ) como  $x$  es el número mayor, entonces la variable `mayor` almacena el número 12, tal como se indica a continuación:

X	Y	Z	MAYOR
12	9	5	12

Si  $x = 9$ ;  $y = 12$ ;  $z = 5$ , evalúa **if** ( $x \geq y \ \&\& \ x \geq z$ ) si  $x$  es mayor que  $y$ , como es falso no evalúa si  $x$  es mayor que  $z$ ; continua con la siguiente comparación, **else if** ( $y \geq x \ \&\& \ y \geq z$ ) y es el mayor que  $x$ ;  $y$  es mayor que  $z$ , entonces la variable `y` tiene almacenado el número mayor que es 12.

X	Y	Z	MAYOR
12	9	5	12

La tabla completa sería:

X	Y	Z	MAYOR
12	9	5	12
9	12	5	12

```

/**Sea x,y,z variables enteras las cuales se van
 * comparando para ir devolviendo el numero mayor en cada
 * caso de la instrucción condicional
 */
public int obtnerMayor( ) {
    if( x >= y )
        if( x >= z )
            return x;
        else
            return z;
    else if( y >= z )
        return y;
    else
        return z;
}

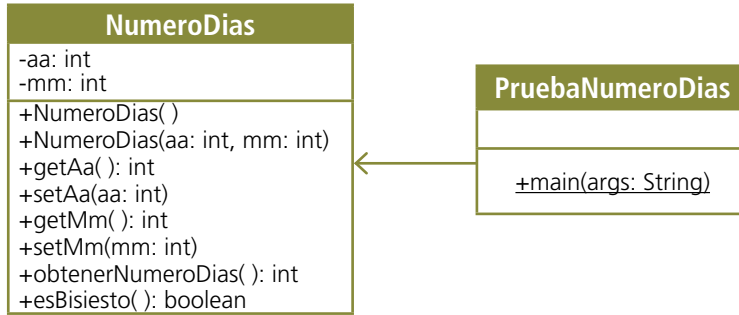
```

- Programa que determina el número de días del mes teniendo en cuenta si el año es bisiesto, a partir del número del mes y del año.

Aplicación diseñada con dos clases: La clase NumeroDias, tiene la responsabilidad de evaluar el número de días y determinar si el año es bisiesto, para lo cual se implementaron dos métodos y la clase PruebaNumeroDias, que tiene la responsabilidad de leer el número del mes y año e imprimir el número de días.

- » `obtenerNumeroDias()`, método evalúa el numero del mes si este es 1, 3, 5, 7, 8, 10, 12; retorna 31; si no, compara si el mes es 2 invocando al método `esBisiesto`, en caso de ser verdadero el retorno en número de días es 29 y 28 en caso contrario; para los demás meses 4, 6, 9, 11; retorna 30.
- » El metodo `esBisiesto(int aa)`, que tiene como parámetro el año, el cual lo evalúa, en el condicional `if ((aa % 4 == 0) && ((aa % 100 != 0) || (aa % 400 == 0)))` en caso de ser verdadero retorna true, de lo contrario retorna false.

## Diagrama de Clases



Codificación en Java del modelado del problema, a partir del diagrama de clases:

```

/**
 * Clase que modela el número de días del mes y
 * evalúa si el año es bisiesto
 */

public class NumeroDias {
    private int aa;
    private int mm;

    public NumeroDias( ) {
        this.aa = 2015;
        this.mm = 4;
    }

    public NumeroDias(int aa, int mm) {
        this.aa = aa;
        this.mm = mm;
    }
}
  
```

```
public int getAa() {
    return aa;
}

public void setAa(int aa) {
    this.aa = aa;
}

public int getMm() {
    return mm;
}

public void setMm(int mm) {
    this.mm = mm;
}

public int obtenerNumeroDias( ){
    int numeroDias = 0;
    if ((mm==1) || (mm==3) || (mm==5) || (mm==7) || (mm==8) ||
        (mm==10) || (mm==12)) {
        numeroDias = 31;
    } else if (mm==2){
        if ( esBisiesto()) {
            numeroDias = 29;
        }else{
            numeroDias = 28;
        }
    } else {
        numeroDias = 30;
    }
    return numeroDias;
}
```

```

/**
 * Metodo que evalua si el año es bisiestto
 * @return true si es año bisiestto; false en caso contrario
 */

private boolean esBisiesto(){
    boolean bisiestto = false;
    if ((aa % 4 == 0) && ((aa % 100 != 0) || (aa % 400 == 0))) {
        bisiestto=true;
    } else {
        bisiestto=false;
    }
    return bisiestto;
}

}

/**
 * Clase que modela la lectura del año y mes e imprime el número
 * de días del mes
 */
import java.util.Scanner;

public class PruebaNumeroDias {
    public static void main(String[] args) {
        //definición objeto unBisiesto de la clase Bisiesto
        Scanner lea = new Scanner(System.in);
        NumeroDias numeroDias = new NumeroDias();
        int aa = 0;
        int mm = 0;
        System.out.print("Digite el año ");
        aa = lea.nextInt();
        System.out.print("Digite el mes ");
        mm = lea.nextInt();
        numeroDias.setAa(aa);
        numeroDias.setMm(mm);
        System.out.println("El Mes "+numeroDias.getMm()+", del año
        "+numeroDias.getAa()+" es de "
        +numeroDias.obtenerNumeroDias()+" días ");
        System.exit(0);
    }
}

```

## 5.7 SELECCIÓN MÚLTIPLE

Cuando se requiere comparar una variable con una serie de datos diferentes, es recomendable utilizar la sentencia `switch`, en cual se indican los posibles valores que puede tomar la variable y las sentencias que se tienen que ejecutar cuando el valor de la variable coincide con alguno de estos.

La estructura `switch case`, permite resumir los `if-else-if` o `if` anidados

La sintaxis de la instrucción `switch`, `case`. En java es:

<pre>switch (expression o variable de control) {     case exp 1:         sentencia 1;         sentencia 2;         break;      case exp 2:     case exp N:         sentencia N;         break;     default:         sentencia D; }</pre>	<pre>switch ( expresión ) {     case valor1:         break;     case valor2:         break;     case valorN:         break;     default: } }</pre>
--	--

La instrucción `switch` toma el valor de la variable que se le pasa como argumento, el cual se compara con cada uno de los valores literales de las sentencias `case`, como se explica a continuación.

- » La palabra reservada `switch`, debe ir en el encabezado del bloque.
- » La variable de control, puede ser de tipo: `byte`, `char`, `short` o `int`.
- » Los paréntesis que contienen la variable de control son obligatorios.
- » La palabra reservada `case`, ejecutará las sentencias correspondientes, con base en el valor de la variable de control, que deberá de evaluarse.



- » La cláusula default es opcional, para el caso que el dato de la variable no coincide con ningún valor, entonces se ejecutan las sentencias por default cuando la haya.
- » La sentencia break al final de cada case pasa el control al final de la sentencia **switch**; de esta forma, cada vez que se ejecute un case todos los enunciados case restantes son ignorados y termina la operación del **switch**.

## El funcionamiento del **switch**

- » Se compara la expresión o variable de control con el primer valor, si coincide, se ejecutan las instrucciones ubicadas bajo ese dato y todas las siguientes que se encuentren hasta hallar un **break**.
- » Si no coincide, se compara con el segundo valor, y así sucesivamente.
- » Al no coincidir con ningún valor, se ejecutan las instrucciones que haya en la parte default, si existe.
- » Después de un **break**, la instrucción **switch** termina y continúa con la siguiente instrucción.

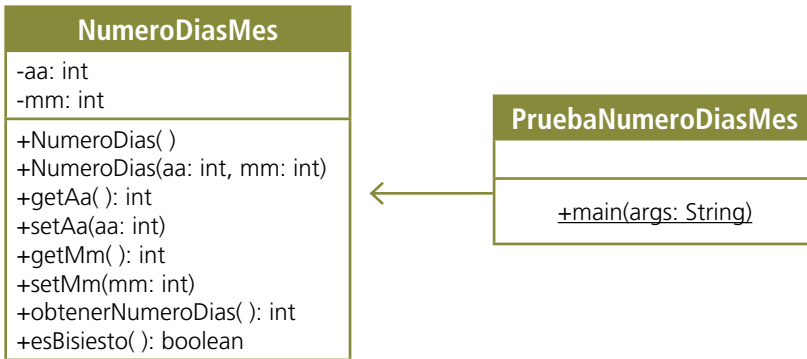
## Ejemplos

Programa que determina el número de días del mes, teniendo en cuenta si el año es bisiesto, haciendo uso de la sentencia switch.

Aplicación diseñada con dos clases: PruebaNumeroDiasMes que tiene la responsabilidad de leer el número del mes y año e imprimir el número de días.

La clase NumeroDiasMes, tiene la responsabilidad de evaluar el número de días y determinar si el año es bisiesto, para lo cual se implementaron dos métodos:

- » obtenerNumeroDias( )
- » El metodo **esBisiesto( )**



- » Si la expresión retorna la etiqueta 1, 3, 5, 7, 8, 10, 12 se ejecuta la sentencia `numeroDias = 31;`
- » Si la expresión retorna la etiqueta 2 invoca la método `esBisiesto()`;
- » Si la expresión retorna la etiqueta 4, 6, 9, 11 se ejecuta la sentencia `numeroDias = 30;`
- » Cuando ninguno de los casos corresponde, se ejecutara la sentencia del default `System.out.println("Mes no contemplado")`.

Codificación en Java del modelado del problema, a partir del diagrama de clases.

```

/**
 * Clase que modela el numero de dias del mes y
 * si un año es bisiesto
 *
 */
public class NumeroDiasMes {

```

```

    private int aa;
    private int mm;

```

Implementación de los métodos set y get.

```

    public NumeroDiasMes(int aa, int mm) {
        this.aa = aa;
        this.mm = mm;
    }

```

```
public int obtenerNumeroDias(){
    int numeroDias = 0;
    switch(mes) {
        case 1: case 3: case 5: case 7: case 10: case 12:
        case 8:
            numeroDias = 31;
            break;
        case 2:
            if ( esBisiesto(año)) {
                numeroDias = 29;
            }else{
                numeroDias = 28;
            }
            break;
        case 4: case 6: case 9: case 11:
            numeroDias = 30;
            break;
        default:
            System.out.println("Mes no contemplado");
    }
    return numeroDias;
}
```

---

```
/**
 * Metodo que evalua si el año es bisiesto
 * @return true si es año bisiesto; false en caso contrario
 */

private boolean esBisiesto(){
    boolean bisiesto = false;
    if ((aa % 4 == 0) && ((aa % 100 != 0) || (aa % 400 == 0))) {
        bisiesto=true;
    } else {
        bisiesto=false;
    }
    return bisiesto;
}
}
```

```

/**
 * Clase que modela la lectura del año y mes e impresión del numero
 * de días del mes
 */
import java.util.Scanner;
public class PruebaNumeroDiasMes {
    public static void main(String[] args) {
        //definición objeto numeroDias de la clase NumeroDiasMes
        Scanner lea = new Scanner(System.in);
        NumeroDiasMes numeroDias = new NumeroDiasMes ();
        int aa = 0;
        int mm = 0;
        System.out.print("Digite el año ");
        aa = lea.nextInt();
        System.out.print("Digite el mes ");
        mm = lea.nextInt();
        numeroDias.setAa(aa);
        numeroDias.setMm(mm);
        System.out.println("El Mes "+numeroDias.getMm()+", del año
        "+numeroDias.getAa()+” es de “
        +numeroDias.obtenerNumeroDias()+” días “);
        System.exit(0);
    }
}

```

## 5.8 OPERADOR CONDICIONAL (?)

En Java el operador (?) denominado alternativa es un operador ternario, conformado por tres operandos, el cual permite bifurcar el flujo del programa a partir del valor de una variable boolean:

**expresión1 ? expresión2 : expresión3;** cuando el resultado de evaluar la expresión lógica es verdadero, retorna el valor de la segunda expresión y en caso contrario, devuelve el valor de la tercera expresión.

**resultado = (expresión lógica ? expresión2 : expresión3);**

El operador condicional `?` es similar a un `if else`, por ejemplo:

Si `x` es igual a cero, entonces `a` es igual a 1; en caso contrario `a` igual a 2

Si `x` es igual a cero, entonces `a` es igual a 1; en caso contrario `a` igual a 2

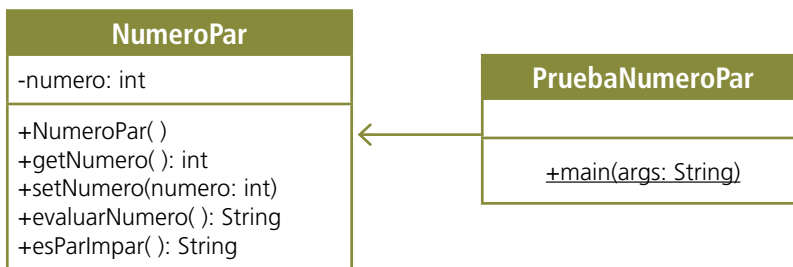
```
if( x == 0 ) {
    a = 1;
} else {
    a = 2;
}
```

```
a = (x == 0) ? 1 : 2;
```

## Ejemplos:

1. Programa que evalúa si un número es par o impar utilizando el operador condicional (`?`).

## Diagrama de clases



```
/**
 * Clase que modela un número evaluando si es par o impar
 *
 */
```

```
public class NumeroPar {
    private int numero;

    public NumeroPar() {
        this.numero = 0;
    }

    public int getNumero() {
        return numero;
    }

    public void setNumero(int numero) {
        this.numero = numero;
    }
}
```

```
/**
 * Metodo que evalua si el número es par o impar
 * utilizando del operador alternativa
 * @return
 */
```

```
public String evaluarNumero()
{
    String respuesta=((numero % 2 == 0)?"par":"impar");
    return respuesta;
}
```

```
/**
 * Metodo que evalua si el número es par o impar
 * utilizando la sentencias if-else
 * @return
 */

public String esParImpar( )
{
    String respuesta = "";
    if(numero % 2 == 0){
        respuesta = "par";

    } else {
        respuesta = "impar";
    }
    return respuesta;
}

}

import java.util.Scanner;

/**
 * Clase que modela lectura e impresion de un numero
 */

public class PruebaNumeroPar {
    public static void main(String[] args) {
        //definición de objetos
        Scanner lea = new Scanner(System.in);
        NumeroPar numero = new NumeroPar();
        //definición de variable
        int n = 0;

        System.out.print("Digite un numero ");
        n = lea.nextInt();
        numero.setNumero(n);
        System.out.println("El numero "+numero.getNumero()+", es
        "+numero.evaluarNumero());
    }

}
```

## 5.9 PREGUNTAS DE REVISIÓN DE CONCEPTOS

- » Importancia de los condicionales en java.
- » Semejanzas entre los condicionales anidados y la sentencia switch.
- » Importancia de realizar pruebas de escritorio a los programas.
- » ¿Qué ventaja tiene modelar una aplicación en diferentes clases?

## 5.10 LECTURAS RECOMENDADAS

- » Condicionales.
- » Jerarquía de los operadores relacionales y lógicos.

## 5.11 EJERCICIOS

Construir las correspondientes sentencias de condicionales que satisfacen los siguientes requerimientos:

1. Hallar el mayor de dos números enteros diferentes.

---

2. Comparar si una cantidad es superior a \$150.000.

---



3. Evaluar si un número entero positivo es múltiplo de tres.
- 

4. Determinar si un número es mayor que 20
- 

5. A partir del siguiente método:

```
public void obtenerResultado( ) {  
    int a = 15;  
    int b = 25;  
    int c = 40;  
    boolean r=false;  
    boolean s=false;  
    boolean t=false;  
    boolean u=false;  
  
    r = a + b > c;  
    s = a - b < c;  
    t = a - b == c;  
    u = a * b != c;  
    System.out.println(r + " " + s + " " + t + " " + u);  
}
```

Determinar (se sugiere realizar el seguimiento a papel) cuáles valores quedan almacenados en:

r \_\_\_\_\_

s \_\_\_\_\_

t \_\_\_\_\_

u \_\_\_\_\_

6. Suponga que:      a = 15;      b = 18;      c = 20

```
public void probar(int a, int b, int c) {
    boolean r=false;
    r = ((a > b)|| (a < c)) && ((a == c) || (a >= b));

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

Cuál será el resultado de r \_\_\_\_\_

7. Suponga que:  $a = 15$ ;  $b = 18$ ;  $c = 20$ ;  $d = 7$ ;

```
public void probar(int a, int b, int c, int d) {
    boolean r=false;
    r = ((a >= b) || (a < d)) && ((a >= d) && (c > d));
    System.out.println(r);
}
```

Cuál será el resultado de  $r$  \_\_\_\_\_

8. Suponga que:  $a = 15$ ;  $b = 18$ ;  $c = 20$ ;

```
public void probar(int a, int b, int c) {
    boolean r=false;
    r = !(a == c) && (c > b);
    System.out.println(r);
}
```

Cuál será el resultado de  $r$  que se imprime \_\_\_\_\_

9. Suponga que:  $a = 4$ ;  $x = 9$ ;

```
public void probar(int a, int x) {
    boolean b;
    b = (++a > ((x - a) - 1)) || (++a != 4);
    System.out.println(b);
}
```

Cuál será el resultado de  $b$ : \_\_\_\_\_

Cuál será el resultado de  $a$ : \_\_\_\_\_

Cuál será el resultado de  $x$ : \_\_\_\_\_

10. Para comparación de cadenas de la siguiente información

```
String moneda = "dollar";  
String mondenaInternacional = " Dollar"
```

Cuál será el resultado de la comparación

`moneda.equals(monedaInterancional)` \_\_\_\_\_

`moneda.equalsIgnoreCase(monedaInterancional)` \_\_\_\_\_

`moneda.compareTo(monedaInterancional)` \_\_\_\_\_

`moneda.equals("dólar")` \_\_\_\_\_

`"dollar".equalsIgnoreCase("Dollar")` \_\_\_\_\_

`moneda.compareTo("Dollar")` \_\_\_\_\_

11. Implementar un método que permita determinar si un número es múltiplo de 3.
12. Diseñar el método que permita determinar si dos números son iguales o diferentes.
13. Elaborar una clase que implemente los métodos para las operaciones básicas: suma, resta, producto y división.
14. Diseñar una clase que provea los métodos para determinar el mayor de dos números enteros diferentes y el número menor.
15. Implementar un método para obtener el menor de tres números.
16. Construir una aplicación en Java, que a partir de la obtención de una fecha cualquiera (dd/mm/aaaa), imprima la fecha del día siguiente.

## 5.12 REFERENCIAS BIBLIOGRÁFICAS

Aguilar, L. J. (2004). *Fundamentos de programación algoritmos y estructura de datos*. Tercera Edición. Ciudad: México: McGraw Hill.

Booch, G. (1996). *Análisis y diseño orientado a objetos con aplicaciones*. Ciudad: México Addison Wesley.

Deitel y Deitel. (2012). *Cómo programar en Java*. Ciudad: México Editorial Pearson (Prentice Hall).

Fowler, M., y Scott, K. UML. (1999). *Gota a gota*. Ciudad: México Editorial Pearson.

Villalobos, J., y Casallas, R. (2006). *Fundamentos de programación, aprendizaje activo basado en casos*. Ciudad: México Editorial Pearson.





# CAPÍTULO 6



**CADENAS**





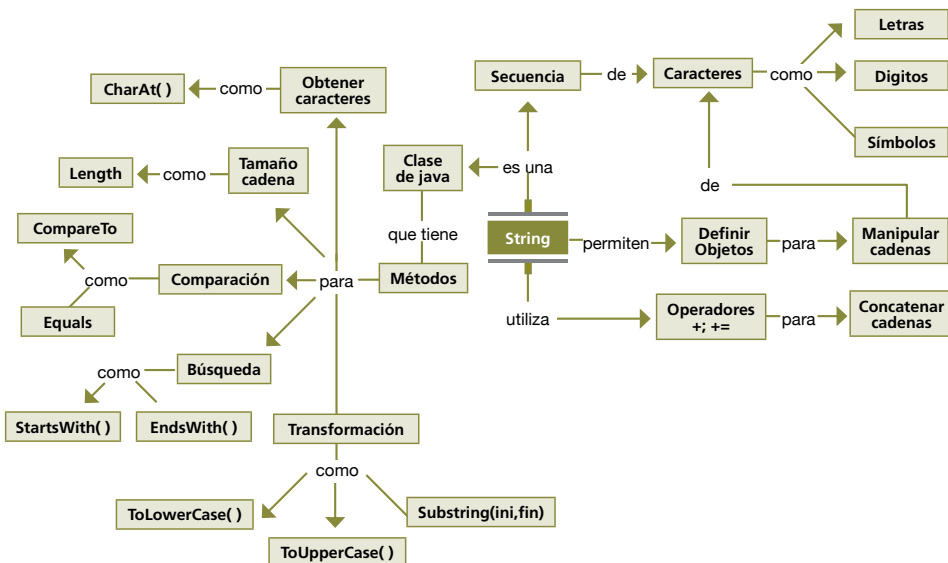
## 6.1 TEMÁTICA A DESARROLLAR

- » String.
- » Métodos asociados a la clase String.

## 6.2 INTRODUCCIÓN

Un String es una secuencia finita de caracteres, que pueden ser letras, dígitos, símbolos especiales; en Java existen varias clases para manipular cadenas de caracteres, una de estas es la clase String definida en la API de Java. Esta clase proporciona algunos métodos dentro de los que permiten convertir el objeto cadena en caracteres, convertir números en cadenas, buscar palabras en una cadenas, crear substrings, cambiar la cadena de caracteres mayúsculas a minúsculas o viceversa, obtener la longitud de la cadena, comparar cadenas, entre otros. La figura 20 detalla el concepto de la clase String.

Figura 20. Concepto de la clase String.



## 6.3 CLASE STRING

Un String en Java representa una cadena de caracteres no modificable. Así por ejemplo la instrucción `String str = "Universidad";` define el String `str` y cada una de las letras que lo integran (U, n, i, v, e, r, s, i, d, a, d) esta palabra no se puede modificar (inmutable).

El API de java proporciona, para la clase String los constructores y una amplia gama de métodos, para el manejo de cadenas, la cadena debe estar entre comillas dobles.

En la tabla 12 se contemplan algunos ejemplos de cadenas.

**En la tabla 12 se contemplan algunos ejemplos de cadenas.**

**Tabla 12.** Ejemplo de cadenas.

SINTAXIS	EJEMPLOS
<b>Etapas de declaración:</b> <code>String identificador;</code>	<code>String nombre;</code> <code>String dato;</code>
<b>Asignación:</b> <code>identificador = " mensaje ";</code>	<code>String nombre = "Carlos";</code> <code>String dato = "123456";</code>

Una cadena es una agrupación del dato primitivo carácter o char. En el lenguaje Java una cadena es una instancia de la clase String y ofrece la siguiente funcionalidad:

- » Inicialización.
- » Asignación.
- » Acceso.
- » Comparación.
- » Búsqueda.
- » Verificación.
- » Conversión a tipos.

Algunos métodos básicos incluidos en la clase String se citan y se describe su funcionalidad en la tabla 13.

**Tabla 13.** Métodos básicos.

MÉTODO	FUNCIONALIDAD
String() String(String str)	Dos de los constructores incorporados en la clase String permiten crear una cadena vacía y una cadena a partir de otra existente.
char charAt(int index)	Este método devuelve el carácter presente en la cadena en la posición dada por el parámetro.
int length()	Este método devuelve la longitud de la cadena.
boolean equals(String str)	Este método permite responder si la cadena enviada como parámetro es igual a la cadena actual.
String toLowerCase()	Este método retorna la cadena convertida en letras minúsculas.
String toUpperCase()	Este método retorna la cadena convertida a letras mayúsculas.

La concatenación, es el proceso de unir dos o más cadenas. Por ejemplo, se definen dos objetos str1, str2; de la clase String y se utiliza el operador + para concatenar cadenas de caracteres o strings:

```
String str1 = "Hola"
String str2 = " Mundo"
String salida = str1+str2
```

El objeto salida almacena "Hola Mundo".

Otra forma de realizar la concatenación de cadenas es utilizando el operador += para unir el nombre con el apellido:

```
String nombre="Juan Carlos ";
nombre+="Bustamante";
System.out.println(nombre);
```

Algunos métodos asociados a la clase String:

**int length( )**, Método que retorna la longitud de una cadena (String), incluyendo espacios en blanco. La longitud siempre es una unidad mayor que el índice asociado al último carácter de la String.

### Ejemplo

```
String nombre = "Ricardo Quintero";  
int longitud = nombre.length();
```

El valor de la variable longitud es 16, correspondiente al número de caracteres de la cadena nombre.

**boolean equals(Object obj)**, método que compara el String con el objeto especificado. El resultado es true si y solo si el argumento es no nulo y es un objeto String que contiene la misma secuencia de caracteres. Para comparar dos cadenas, se tiene:

```
String cadena1="Telecomunicaciones";  
String cadena2=new String("Telecomunicaciones");  
if(cadena1.equals(cadena1)){  
    System.out.println("cadenas de igual contenido");  
} else {  
    System.out.println("Cadenas Diferentes");  
}
```

Rutinas que realizan la comparación de una cadena que conforma una clave de acceso, utilizando el método equals( ):

```
String password = "ABC123";  
if ( password.equals("abc123") {  
    System.out.println("Clave correcta");  
} else {  
    System.out.println("Clave errada");  
}
```

Se imprime clave errada, ya que hay diferencias entre las minúsculas y mayúsculas

**boolean equalsIgnoreCase(String str)** método que evalúa si dos String tienen los mismos caracteres y en el mismo orden sin tener en cuenta las mayúsculas. Si es así devuelve true y si no false.

Para la comparación de una cadena que conforma una clave de acceso, utilizando el método `equalsIgnoreCase()`:

```
String password = "abc123";
if password.equalsIgnoreCase("ABC123") {
    System.out.println("Clave correcta ");
} else {
    System.out.println("Clave Errada ");
}
```

Se imprime clave correcta, ya que el método `equalsIgnoreCase()`, ignora la diferencias entre las minúsculas y mayúsculas

**boolean startsWith(String str)**, método que retorna true si el String sobre la que se aplica inicia con el argumento; false si esto no ocurre.

El siguiente fragmento de código evalúa, si la cadena inicia con los caracteres ip:

```
String parametro = "ip = 192.168.100.2";
if (parametro.startsWith("ip")) {
    System.out.println("La direccion "+parametro);
} else {
    System.out.println("El parámetro no es una ip");
}
```

**boolean endsWith(String str)** método que retorna true si el String termina en los caracteres indicados, false si esto no ocurre.

Líneas de código que evalúa, si la cadena termina con .co:

```
String url = "http://www.ulibertadores.edu.co";
if (url.endsWith(".co")) {
    System.out.println("Pagina colombiana");
} else {
    System.out.println("Pagina extranjera");
}
```

**char charAt(int indice)**, método que retorna el carácter en la posición indicada por índice. El rango de índice va de 0 a `length() - 1`. `cadena.charAt(i)`.

## Ejemplos

1. Ubicar un carácter mediante un índice.

```
String nombre = "Miguel Hernández";  
char c = nombre.charAt(0);
```

Se imprime la M, correspondiente al primer carácter.

2. Impresión de todos los caracteres de una cadena.

```
for ( int i = 0; i < cadena.length;i++ ) {  
    System.out.println(" "+cadena[i]);  
}
```

**String toLowerCase()**, método que convierte un cadena a letras minúsculas.

## Ejemplos

1. Se almacena una cadena de caracteres en el objeto cadena1 y se imprime en letras mayúsculas:

```
String cadena1="Telecomunicaciones";  
System.out.println(cadena.toLowerCase());
```

La impresión es TELECOMUNICACIONES.

2. Se almacena en el objeto titulo una cadena de caracteres y se realiza la conversión a mayúsculas, asignando el resultado a la cadena mayúscula.

```
String titulo = "Fundación Universitaria";  
String mayuscula = titulo.toUpperCase();
```

El resultado es:

```
mayuscula = "FUNDACIÓN UNIVERSITARIA";
```

**String toUpperCase()** Método que convierte una cadena a letras mayúsculas.

## Ejemplos

1. Se almacena una cadena de caracteres en el objeto `cadena1` y se imprime en letras minúsculas:

```
String cadena1="Telecomunicaciones";
System.out.println(cadena1.toUpperCase());
```

La impresión es telecomunicaciones.

2. Se almacena en el objeto `titulo` una cadena de caracteres y se realiza la conversión a mayúsculas, asignando el resultado a la cadena minúscula:

```
String titulo = "Fundación Universitaria";
String minuscula = titulo.toLowerCase();
minuscula = "Fundación Universitaria";
```

El resultado es:

```
minuscula = "Fundación Universitaria";
```

**`String substring( int inicial, int fin )`** Método que retorna una parte o subcadena (substring) de un string dado.

## Ejemplos

1. Se extrae una subcadena a partir de la posición inicial y una final:

```
String nombre = "Oscar Alejandro";
String subCadena = nombre.substring(6,14);
```

La variable `subCadena` almacena "Alejandro", ya que corresponde al sexto carácter y va hasta el 14

2. Se extrae de un substring especificando la posición de comienzo y la del final.

```
String str = "El lenguaje Java";
String subStr=str.substring(3, 11);
```

La **variable** `subStr` almacena la subcadena "lenguaje"

**`String trim( )`** método que retorna la cadena (String) que se le pasa al argumento, pero sin espacios en blanco al principio ni al final. No elimina los espacios en blanco situados entre las palabras.



## Ejemplo

```
String autores = " Baquero & Hernández ";
String resaltar = "*" + autores.trim() + "*";
```

\*Baquero & Hernández\* elimina el espacio al inicio y le agrega asterisco, al inicio y final de la cadena.

**String replace (char anteriorChar, char nuevoChar):** método que permite cambiar el carácter asociado al primer argumento por el segundo argumentos, de la cadena (String) del método, generando una nueva cadena. La cadena (String) sobre la que se aplica el método replace no cambia, simplemente se crea otra cadena en base a la String sobre la que se aplica el método, tal como se puede observar en los líneas de código presentadas a continuación.

```
// Cadena sobre la que realizaremos la sustitución
String cadena1 = "Institucion Universitaria Los Libertadores";

// Cadena en la que almacenaremos el resultado
String cadena2 = null;

cadena2 = cadena1.replace("Institucion", "Fundación");

System.out.println(cadena2);
// Fundación universitaria los Libertadores

System.out.println(cadena1);
/* Al ejecutar el método
replace no afecta a la cadena original */
```

## 6.4 OBTENER CADENAS DESDE LAS PRIMITIVAS

La clase **String** tiene métodos, implementados para transformar valores de otros tipos de datos a su representación como cadena. Todas estas funciones tienen el nombre de `valueOf`, estando el método sobrecargado para todos los tipos de datos básicos.

## Ejemplos

Rutina que convierten datos numéricos a cadenas (String):

```
String tres = String.valueOf(3);
String dosPuntoUno = String.valueOf(2.1);
final double PI = 3.141592;
String piString = String.valueOf(PI);
```

## 6.5 OBTENER PRIMITIVAS DESDE LAS CADENAS

Para esto se utilizan los métodos de las clases Integer y Float.

```
String dato = "2011";
int alfaInteger = Integer.parseInt(dato);
String beta = "20.11";
float betaFloat = Float.parseFloat(beta);
```

**compareTo(String str)** método que realiza una comparación entre un String (en este caso) y otro objeto de tipo String, retornando un número entero menor que 0 si la cadena argumento es lexicográficamente mayor que la cadena a comparar, 0 si son iguales y será mayor que 0 si es menor.

### Ejemplos

```
String str1 = "a";
String str2 = "b";

System.out.println(str1.compareTo(str2)); // -1
System.out.println(str2.compareTo(str1)); // 1  str2 > str1

str2 = "aa";

System.out.println(str1.compareTo(str2)); // -1
System.out.println(str2.compareTo(str1)); // 1

str2 = "c";

System.out.println(str1.compareTo(str2)); // -2
System.out.println(str2.compareTo(str1)); // 2

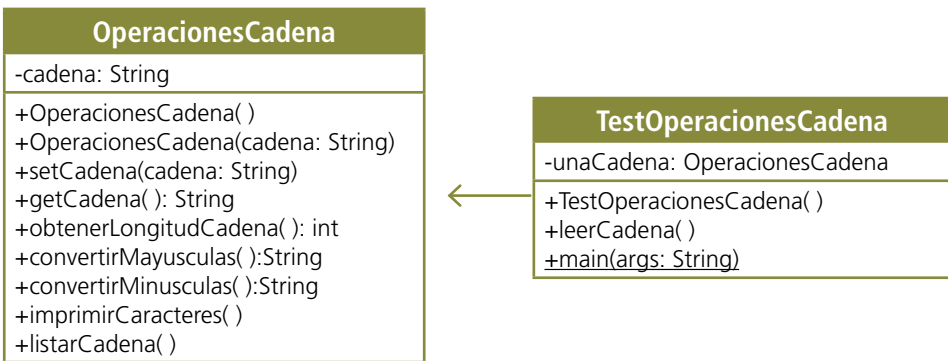
str1 = "c";

System.out.println(str1.compareTo(str2)); // 0
System.out.println(str2.compareTo(str1)); // 0  cadena iguales
```

## Programas de ejemplo

1. Aplicación, que a partir de una cadena ingresada por teclado imprime: los caracteres que la conforman; el tamaño de la cadena, y la conversión a mayúsculas y minúsculas.

El siguiente diagrama de clases define unas operaciones básicas de Strings.



```

/**
 * clase que modela las operaciones de una cadena como:
 * tamaño de la cadena, convertir a mayúsculas y minúsculas.
 * @author Ing. Miguel Hernández Bejarano
 * @version 1.0
 */

public class OperacionesCadena {
    /**
     * atributo de la clases
     */
    private String cadena;

    /**
     * constructor que inicializa la cadena con un espacio en
     blanco
     */
    public OperacionesCadena( ) {
        this.cadena = " ";
    }
}
  
```

```

/**
 * Constructor que permite crear cadenas
 * @param cadena
 */
public OperacionesCadena(String cadena) {
    this.cadena = cadena;
}

/**
 * Método que actualiza el contenido de la cadena
 * @param cadena
 */
public void setCadena(String cadena) {
    this.cadena = cadena;
}

/**
 * Método que retorna el valor de la cadena
 * @return cadena
 */
public String getCadena( ) {
    return cadena;
}

/**
 * Método que retorna el tamaño o longitud de la cadena
 * @return tamaño cadena
 */
public int obtenerLongitudCadena( ){
    return cadena.length();
}

/**
 * Método que convierte y retorna la cadena en mayúsculas
 * @return cadena en mayúsculas
 */
public String convertirMayusculas( ) {
    return cadena.toUpperCase();
}

/**
 * Método que convierte y retorna la cadena en minúsculas
 * @return cadena en minúsculas
 */

```

```

    public String convertirMinusculas( ) {
        return cadena.toLowerCase();
    }
    /**
     * Método que imprime los caracteres que conforma la cadena
     */
    public void imprimirCaracteresCadena( ){
        char c=' ';
        System.out.println("\nImpresion de la Cadena por
        caracteres");
        for(int i=0;i<cadena.length();i++){
            c=cadena.charAt(i);
            System.out.println(i+" "+c);
        }
        System.out.println(" ");
    }

    /**
     * Método que imprime la cadena
     */
    public void imprimirCadena( ){
        System.out.println(" "+cadena);
    }
}

import javax.util.Scanner;
/**
 * clase que modela la lectura de la cadena e
 * impresión de los resultados
 * @version 1.0
 */
public class Cadena {
    /**
     * atributos se define objeto de la clase OperacionesCadena
     */
    private OperacionesCadena operacion;

    /**
     * constructor de la clase cadena
     */
    public Cadena(){
        this.cadena=" ";
        this.operacion= new OperacionesCadena();
    }
}

```

```

/**
 * método en el que se realiza la lectura de la cadena
 * e impresión de los resultado de operaciones con la cadena
 */
public void leerCadena(){
    String cadena;
    String resultados = " ";
    int tam = 0;
        Scanner in = new Scanner(System.in);
    //se lee la cadeena por teclado
    cadena=in.nextLine();
    System.out.println("Digite Cadena ");
    System.out.println("Impresión de la cadena ");
    operacion.imprimirCaracteres(cadena);
    tam=operacion.obtenerlongitudCadena(cadena);
    System.out.println(" \nLa Longitud de la cadena: "
    +tam+" caracteres ");
    System.out.println(" \nLa cadena texto en mayúsculas:"
    +operacion.convertirMayusculas(cadena));
    System.out.println("La cadena texto en minúsculas: "
    +operacion.convertirMinusculas(cadena));
}

/**
 * metodo principal, para iniciar la ejecución del proyecto
 * @param args
 */
public static void main(String[] args) {
    //definición del objeto str de la clase Cadena
    Cadena str=new Cadena();
    str.leerCadena();
}
}

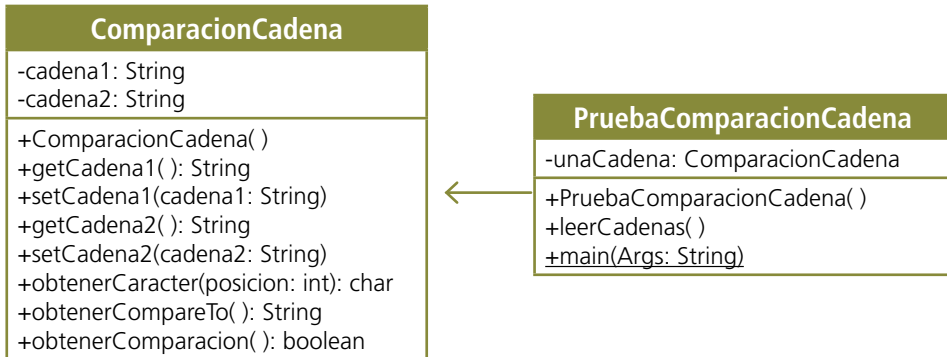
```

- 
- » Crear un proyecto en Eclipse llamado cadena, copiar la clase OperacionesCadena y digitalizar la clase TestOperacionesCadena, compilar y correr la aplicación.

```
import java.util.Scanner;
/**
 * clase que modela las operaciones la lectura de la cadena e
 * impresion de los resultados de las operaciones de la cadena
 * @ File Name : TestOperacionesCadena.java
 * @author : Miguel Hernández
 * @version 1.0
 */
public class TestOperacionesCadena {
    private OperacionesCadena unaCadena;
    public TestOperacionesCadena() {
        this.unaCadena= new OperacionesCadena();
    }
    public void leerCadena(String cadena){
        unaCadena.setCadena(cadena);
        System.out.println( "\nLa Longitud de la cadena: "
            +unaCadena.obtenerLongitudCadena()+" caracteres ");
        System.out.println( "\nLa cadena texto en mayusculas: "
            +unaCadena.convertirMayusculas());
        System.out.println( "La cadena texto en minusculas: " +
            unaCadena.convertirMinusculas());
        unaCadena.imprimirCaracteresCadena();
        unaCadena.imprimirCadena();
    }

    public static void main(String[] args) {
        Scanner in = new Scanner(System.in);
        TestOperacionesCadena prueba = new TestOperacionesCadena();
        String cadena=" ";
        //se lee la cadeena por teclado
        System.out.println ("Digite Cadena ");
        cadena=in.nextLine();
        prueba.leerCadena(cadena);
    }
}
```

2. Aplicación que, a partir de dos cadenas ingresadas por teclado, efectúa la comparación de las dos cadenas utilizando los métodos `compareTo()`, y `equals()`, así como la impresión del segundo carácter de la cadena2.



```

public class ComparacionCadena {

    private String cadena1;
    private String cadena2;

    public ComparacionCadena( ) {
        this.cadena1 = "Hola";
        this.cadena2 = "ola";
    }

    public String getCadena1() {
        return cadena1;
    }

    public void setCadena1(String cadena1) {
        this.cadena1 = cadena1;
    }

    public String getCadena2() {
        return cadena2;
    }

    public void setCadena2(String cadena2) {
        this.cadena2 = cadena2;
    }
}
  
```



```
/**
 * metodo que retorna el caracter de la segunda posición de
 * la cadena
 */
public char obtenerCaracter(int posicion){
    char caracter=cadena2.charAt(posicion);
    return caracter;
}

/**
 * metodo que compara dos cadenas
 * @return retorna cero si son iguales
 */

public String obtenerCompareTo(){
    int res=cadena1.compareTo(cadena2);
    String respuesta=" ";
    if (res==0)
        respuesta="Las cadenas son iguales";
    else if (res>0)
        respuesta="La cadena mayor es "+cadena1;
    else
        respuesta="La cadena mayor es "+cadena2;
    return respuesta;
}

/**
 * Metodos que compara dos cadenas
 * @return true si las dos cadenas son iguales o false en caso
 * contrario
 */
public boolean obtenerComparacion(){
    if (cadena1.equals(cadena2)){
        return true;
    }
    else{
        return false;
    }
}
}
```

```

import java.util.Scanner;

public class PruebaComparacionCadena {

    private ComparacionCadena unaCadena;

    /**
     * constructor
     */
    public PruebaComparacionCadena() {
        this.unaCadena = new ComparacionCadena();
    }

    /**
     * método que realiza la lectura de las cadenas
     * e impresión de los resultados
     */

    public void leerCadenas(){
        String cadena1;
        String cadena2;
        Scanner lea = new Scanner(System.in);
        char caracter = ' ';
        boolean cadenasIguales = false;
        String comparaCadenas = "";
        String cambioLetras = "";
        String segundaComparacion = "";

        //se lee la primera cadena
        System.out.println("Digite Primera Cadena ");
        cadena1=lea.nextLine();
        System.out.println("Digite Segunda Cadena ");
        //se lee la segunda cadena
        cadena2=lea.nextLine();

        this.unaCadena.setCadena1(cadena1);
        this.unaCadena.setCadena2(cadena2);

        //el objeto operacion accesa al metodo metodoCharAt
        //que retorna el caracter de una determinada posición
    }
}

```

```

        caracter=unaCadena.obtenerCaracter(1);

        comparaCadenas= unaCadena.obtenerCompareTo() ;

        cadenasIguales=unaCadena.obtenerComparacion();
        if(cadenasIguales==true){
            segundaComparacion="Cadenas Iguales ";
        }
        else{
            segundaComparacion="Cadenas diferentes ";
        }

        //impresion de resultados
        System.out.println("Primera Cadena:
        "+unaCadena.getCadena1()+"\n"+
        "Segunda Cadena:  "+unaCadena.getCadena2()+"\n"+
        "Caracter de la posición dos de la Cadena1:"+caracter+"\n"+
        "Comparacion de Cadenas metodo compareTo:
        "+comparaCadenas+"\n"+
        "Segunda Comparacion de Cadenas con el metodo equals():
        "+segundaComparacion+"\n");
    }

    /**
     * método principal donde se ejecuta el proyecto
     * @param Args
     */
    public static void main(String Args []){
        PruebaComparacionCadena prueba = new PruebaComparacion
        Cadena();
        prueba.leerCadenas();
    }
}

```

## 6.6 LECTURAS RECOMENDADAS

- » Clase StringBuffer.
- » Clase StringReader.
- » Clase Character.
- » Clase StringTokenizer.

## 6.7 PREGUNTAS DE REVISIÓN DE CONCEPTOS

- » Establecer cuáles son las semejanzas y deferencias entre los métodos equals y compareTo.
- » Describa las ventajas que se tienen sobre la utilización de cadenas.
- » Determinar otro tipo de aplicación de las cadenas.

## 6.8 EJERCICIOS

Crear una clase que incluya los elementos incorporados a partir del siguiente String definido: String cadena = "Universidad".

U	n	i	v	e	r	s	i	d	a	d
---	---	---	---	---	---	---	---	---	---	---

Completar la sentencia de los métodos según el requerimiento planteado.

EjCadena
-cadena: String
+EjCadena()
+imprimirPrimerElementoCadena()
+imprimirTercerElementoCadena()
+imprimirUltimoElementoCadena()
+imprimirPosicionesImpares()
+imprimirElementoMitad()
+ imprimirUltimoElemento()
+ imprimirCadena()

Imprimir el primer dato de la cadena

```
public void imprimirPrimerElementoCadena ( ) {  
}
```

Imprimir el tercer elemento de la cadena

```
public void imprimirTercerElementoCadena ( ) {  
}
```

Imprimir el último dato de la cadena

```
public void imprimirUltimoElementoCadena ( ) {  
}
```

Imprimir todos los elementos de las posiciones impares de la cadena (sugerencia tenga en cuenta que este método debe incluir un ciclo).

```
public void imprimirPosicionesImpares ( ) {  
}
```

Imprimir el dato de la mitad de la cadena.

```
public void imprimirElementoMitad ( ) {  
}
```

Imprimir los caracteres que hay en la cadena de la última posición a la primera.

```
public void imprimirUltimoElemento ( ) {  
}
```

Imprimir todos los caracteres de la cadena, cadena (sugerencia tenga en cuenta que este método debe incluir un ciclo).

```
public void imprimirCadena ( ) {  
}
```

## 6.9 REFERENCIAS BIBLIOGRÁFICAS

Aguilar, L. J. (2004). *Fundamentos de programación algoritmos y estructura de datos*. Tercera Edición. Ciudad: México: McGraw Hill.

Barnes D., Kölling M. (2007), *Programación orientada a objetos con Java*. Ciudad Madrid, Prentice Hall.

Bertrand M. (1999). *Construcción de software orientado a objetos* (2ª edición) Ciudad: Madrid, Prentice Hall.

Booch, G. (1996). *Análisis y diseño orientado a objetos con aplicaciones*. Ciudad: México Addison Wesley.

Deitel y Deitel. (2012). *Cómo programar en Java*. Ciudad: México Editorial Pearson (Prentice Hall).

Fowler, M., y Scott, K. UML. (1999). *Gota a gota*. Ciudad: México Editorial Pearson.

Graig, L. (2003). *UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado* (2ª edición) Ciudad: México Prentice Hall.

Rumbaugh, J., Jacobson, I., y; Booch, G.: (2005). *The Unified Modeling Language User Guide*, San Francisco: Addison-Wesley, Reading, Mass. ISBN-13: 078-5342267976 ISBN-10: 0321267974

Villalobos, J., y Casallas, R. (2006). *Fundamentos de programación, aprendizaje activo basado en casos*. Ciudad: México Editorial Pearson.





# CAPÍTULO 7

**CICLOS**





## 7.1 TEMÁTICA A DESARROLLAR

- » Estructuras de control repetitivas.

## 7.2 INTRODUCCIÓN

Los ciclos se encuentran entre las estructuras más comunes y utilizadas en programación, permitiendo que una o más instrucciones se repitan sin tener que volver a digitar el mismo código.

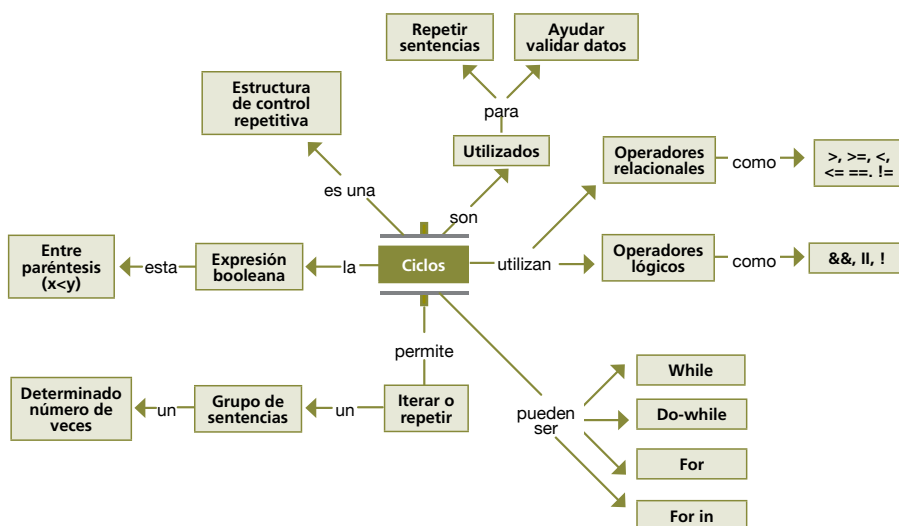
Los ciclos o iteraciones en Java son:

- » Ciclo while, se utiliza para ejecutar una serie de instrucciones (o una sola) mientras se cumpla una condición, cuando esta se deje de cumplir, las instrucciones que están dentro del ciclo dejarán de ejecutarse.
- » Ciclo do-while es bastante similar al ciclo while, con la única diferencia en que el ciclo while si no se cumple la condición desde la entrada no realizarán la instrucciones, pero en el caso del do-while siempre se ejecutan las instrucciones de su bloque como mínimo una vez.
- » Ciclo for, permite ejecutar un conjunto de sentencias un número determinado de veces fijado al principio del ciclo.
- » Ciclo for mejorado (for-each), es la estructura iterativa ideal para recorrer colecciones de objetos o de cualquier otro tipo de valor desde el inicio hasta el final.

El uso de una u otra de estas estructuras de control depende de las preferencias del programador y en muy pocos casos, del enunciado del problema; teniendo como presente, que unas pueden tener mejor ventaja que otra.

La figura 21 del Concepto de Ciclos complementa el concepto de las iteraciones en Java.

Figura 21. Concepto de Ciclos.



### 7.3 ESTRUCTURAS DE CONTROL REPETITIVAS O CICLOS

Son estructuras iterativas que permiten repetir una o varias sentencias un número determinado de veces, utilizadas cuando se hace necesario iterar el mismo conjunto de instrucciones una cantidad específica de veces, determinada por una expresión booleana en programa o método.

La estructura de un ciclo comprende:

- » Una variable o más variables de control o variables iniciales.
- » Una condición booleana de parada o fin del ciclo.
- » Un incremento o decremento de la variable de control.

Los tipos de ciclos que se pueden utilizar en un programa Java son:

- » While.
- » do while.
- » for.
- » foreach o for mejorado.

### 7.3.1 Estructura de repetición **while** (mientras)

Estructura repetitiva utilizada en programación que permite especificar que un programa o método debe repetir un bloque de código, mientras se cumpla la condición, cuando deje de cumplir la condición, las iteraciones terminan y se ejecutará la primera sentencia que esta inmediatamente después de la estructura repetitiva while.

Sintaxis general de esta instrucción **while** es:

```
while (expresiónBooleana) {
    Instrucción1;
}
```

Se recomienda utilizar las llaves { } cuando es una sola sentencia, para el caso en que se tengan más de una instrucción es obligatorio.

```
while (condición) {
    Instrucción1;
    Instrucción2;
    . . .
    InstrucciónN
}
```

```
while (condición) {
    Instrucción1;
}
```

La estructura repetitiva while evalúa la condición o expresión booleana para determinar si itera el conjunto de instrucciones que están entre las llaves, mientras la condición sea verdadera se repiten el bloque de instrucciones que se encuentre entre los paréntesis cursivos.

Para el funcionamiento del ciclo while, se plantea la siguiente situación: se requiere generar la impresión de los cinco primeros números naturales.

Acción	Proceso	Impresión del número
Inicialización	numero = 1;	
impresión	System.out.println(n);	1
incremento	numero ++;	
Impresión	System.out.println(n);	2
incremento	numero ++;	
Impresión	System.out.println(n);	3
incremento	numero ++;	
Impresión	System.out.println(n);	4
incremento	numero ++;	
impresión	System.out.println(n);	5
incremento	numero ++;	

Como se puede observar, las instrucciones que se repiten son la impresión y el incremento de la variable numero, para este caso estas serían las instrucciones que se incluirían dentro del ciclo while.

```

inicializacion;

while(condicion) {
    instrucciones a repetir;
    incremento o
    decremento
}

```

```

Inicialización int numero = 1;
Condición o expresión booleana, numero <= 5
expresiones a repetir; System.out.println(n);
incremento    numero++;

```

```
while(condición o expresión booleana) {
```



Número de interacciones

```
}
```

Lo ideal es utilizar el ciclo while cuando se conoce previamente el número de iteraciones.

```
//inicialiación
numero = 1;
while (numero <= 5) { //condición numero <= 5
    //instrucciones que se repiten
    System.out.println(""+numero); //impresión
    numero++; //incremento
}
```

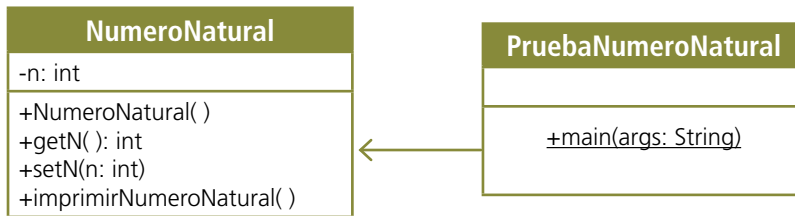
El comportamiento de la anterior rutina es:

- » Mientras la condición (**numero** <= 5), se cumpla se realizarán las sentencias definidas entre el bloque de las llaves { ... }: `System.out.println(""+numero);` y `numero++;`.
- » Cada vez que se termina de ejecutar el bloque de instrucciones se vuelve a evaluar la condición, si es verdadera, nuevamente se vuelven a realizar el bloque de instrucciones que están entre las llaves.
- » Se termina la ejecución cuando la condición pasa a ser falsa, que para el caso es cuando la variable número es mayor que cinco.

## Ejemplos

1. Programa que imprime los diez primeros números naturales.

### Diagrama de clases del ejemplo planteado.



Codificación en Java del modelado del problema, a partir del diagrama de clases:

```

public class NumeroNatural {
    private int n;

    public NumeroNatural(int n) {
        this.n = n;
    }
    public int getN() {
        return n;
    }

    public void setN(int n) {
        this.n = n;
    }

    public void imprimirNumerosNaturales() {
        int x = 1;
        /**
         * En el ciclo while se evalúa x mientras que x <= n
         * se imprime el valor de x e incrementa en uno
         */
    }
}
  
```

```

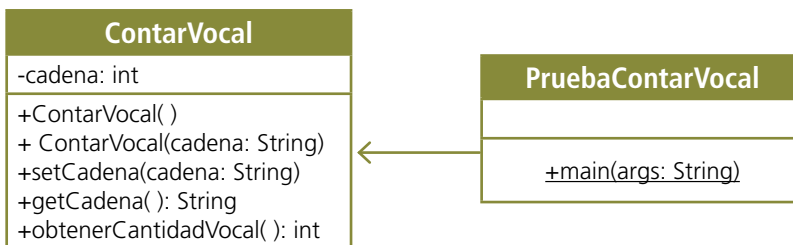
        while (x <= n) {
            System.out.println(" "+ x);
            x ++;
        }
    }
}

import java.util.Scanner;
public class PruebaNumeroNatural {
    public static void main(String[] args) {
        Scanner lea = new Scanner(System.in);
        int numero = 0;
        System.out.print("Digite la cantidad de números
naturales a imprimir: ");
        numero = lea.nextInt();
        NumeroNatural unNumero = new NumeroNatural(numero);
        unNumero.imprimirNumerosNaturales();
    }
}

```

2. Programa que cuenta el número de veces que aparece la vocal e en una cadena determinada.

## Diagrama de clases.





Codificación en Java del modelado del enunciado, a partir del diagrama de clases:

```
public class ContarVocal {
    /**
     * atributo de la clases
     */
    private String cadena;

    /**
     * constructor que inicializa la cadena con un espacio en blanco
     */
    public ContarVocal( ) {
        this.cadena = " ";
    }

    /**
     * Método que actualiza el contenido de la cadena
     * @param cadena
     */
    public void setCadena(String cadena) {
        this.cadena = cadena;
    }

    /**
     * Método que retorna el valor de la cadena
     * @return cadena
     */
    public String getCadena( ) {
        return cadena;
    }

    /**
     * Método que cuenta el número de apariciones de la vocal e
     */
    public int obtenerCantidadVocal( ) {
        int i = 0;
        int cantidad = 0;
        while (i < cadena.length()) {
            if(cadena.charAt(i)=='e') {
                cantidad ++;
            }
            i++;
        }
    }
}
```

```

        }
        return cantidad;
    }
}

import java.util.Scanner;

public class PruebaContarVocal {
    public static void main(String[] args) {
        ContarVocal cuentaVocal = new ContarVocal();
        String cadena="";
        int cantidad = 0;
        Scanner lea = new Scanner(System.in);
        System.out.print ("Digite Cadena: ");
        cadena = lea.nextLine();
        cuentaVocal.setCadena(cadena);
        cantidad = cuentaVocal.obtenerCantidadVocal();
        System.out.println("La vocal e esta presente "+ cantidad
+ "
        veces"+ " en la cadena: " + cuentaVocal.getCadena());
    }
}

```

## Sentencias break y continue

Estas sentencias cambian el flujo de control especialmente en los ciclos o estructuras repetitivas.

**break** permite que un programa se salga de los ciclos while, for, do...while y de la sentencia de selección switch en el cual se encuentra.

La sentencia break permite salir de las estructuras de control anidadas, pasando el control del programa a la instrucción que se encuentra luego de la llave de cierre del bloque.

### Ejemplo:

```
while ( expresiónBoolean) {  
    while ( expresiónBoolean) {  
        if( condicion ) {  
            break;  
        }  
    }  
}
```

**continue:** procede con la siguiente iteración (repetición) del ciclo while, for o do-while, saltándose las sentencias que se encuentran luego de la sentencia continue.

### Ejemplo

```
while ( expressionBoolean) {  
    if(expresiónBoolean ) {  
        continue;  
    }  
}
```

## 7.3.2 Estructura de repetición do-while

El ciclo do-while ejecuta una instrucción o un bloque de instrucciones que están entre las llaves { }, mientras que la expresión booleana especificada en el while se evalúe como verdadera. La sintaxis general es:

<pre>do {     &lt;instrucciones&gt; } while(&lt;condición&gt;);</pre>	<pre>do {     instrucciones a repetir;     incremento o decremento } while (condicionBooleana);</pre>
---	---

La diferencia del significado de do-while respecto a while es que en el do... while, primero ejecuta las <instrucciones> y luego evalúa la <condición> o expresión booleana y de esta forma podrá seguir iterando las instrucciones que están dentro del ciclo.

```
int numero = 1;  ← Inicialización variable de control
do{
```

```
    System.out.println(""+numero);  ← Incremento variable de control
    numero++;                        ← condición
} while (numero < 11); ←
```

```
int numero = 1;
do{
```



Número de Iteraciones

```
    } while (numero < 11);
```

## Ejemplos

1. Aplicación que genera la impresión de los n primeros números naturales, utilizando el ciclo do while.

```
public class NumeroNatural {
    private int n;

    public NumeroNatural(int n) {
        this.n = n;
    }

    public void setN(int n) {
        this.n = n;
    }

    public int getN() {
        return n;
    }
}
```

```

        public void imprimirNumeros() {
            int x = 1;
            do {
                System.out.print(" "+x);
                x++;
            } while (x <= n);
        }
    }

import java.util.Scanner;
public class PruebaNumeroNatural {
    public static void main(String[ ] args) {
        int cantidad = 0;
        Scanner lea = new Scanner(System.in);
        System.out.print("Cantidad de numeros naturales a imprimir: ");
        cantidad = lea.nextInt();
        NumeroNatural numero = new NumeroNatural(cantidad);
        numero.imprimirNumeros();
    }
}

```

2. Programa que cuenta el número de veces que aparece la vocal e en una cadena determinada, haciendo uso del ciclo do while.

```

public class ContarVocal {

    /**
     * atributo de la clases
     */
    private String cadena;

    /**
     * constructor que inicializa la cadena con un espacio en blanco
     */
}

```

```

public ContarVocal( ) {
    this.cadena = "";
}

/**
 * Método que actualiza el contenido de la cadena
 * @param cadena
 */
public void setCadena(String cadena) {
    this.cadena = cadena;
}

/**
 * Método que retorna el valor de la cadena
 * @return cadena
 */
public String getCadena( ) {
    return cadena;
}

/**
 * Método que cuenta el número de apariciones de la vocal e
 */

public int obtenerCantidadVocal( ) {
    int i = 0;
    int cantidad = 0;
    do {
        if(cadena.charAt(i)=='e') {
            cantidad ++;
        }
        i++;
    } while (i < cadena.length());
    return cantidad;
}
}

```

La clase PruebaContarVocal es similar al ejemplo anterior, nótese la clase ContarVocal los métodos constructor, set y get son idénticos.

### 7.3.3 Estructura de repetición for

Utilizada para repetir una instrucción o un conjunto de instrucciones, hasta que una condición se cumpla. La sintaxis de la sentencia for es:

```
for (instrucción; condición; instrucción) {  
    instrucciones  
}  
  
for (inicialización; condición; incremento) {  
    instrucciones  
}  
  
for (inicialización variable ; expresión Booleana; incremento o decremento) {  
    Instrucción (es) o acciones;  
}
```

**Inicialización:** Es una instrucción que permite dar valor inicial a la variable de control del ciclo, por ejemplo `int n = 1`.

**Expresión booleana:** Es una condición que se evalúa antes de realizar el bloque de instrucciones en el ciclo. Esta condición determina cuando se termina la ejecución del ciclo, por ejemplo `n <= 10`.

**Incremento:** Es una instrucción usada para incrementar o decrementar el valor de las variables de control, por ejemplo `n ++`; `n - -`.

---

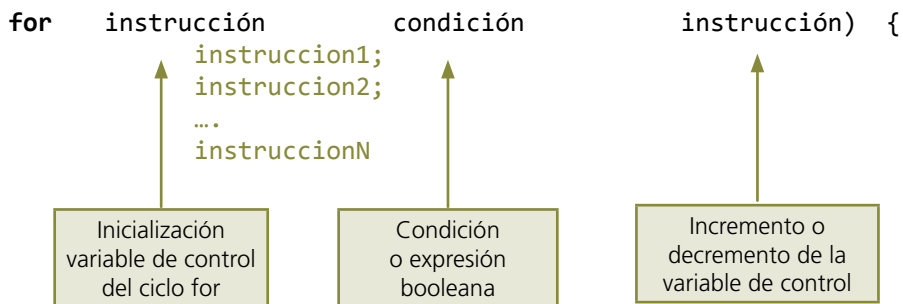
### Ejemplos

```
for (int n = 1; n <= 10; n ++)  
for (int x = 10; n >= 1; n - -)
```

## Funcionamiento del ciclo for

La estructura repetitiva del ciclo for está conformada por tres partes y un grupo de instrucciones que están entre las llaves { }.

```
for ( instrucción ; condición ; instrucción) {
    instruccion1;
    instruccion2;
    ...
    instruccionn;
}
```



Impresión de los 10 primeros números naturales.

```
for (int n = 1;      n <= 10;      n ++ ) {

    //instruccion
    System.out.println(""+n);

}
```

The diagram illustrates the three components of the example `for` loop, each in a box with an arrow pointing to its corresponding part in the code above:

- Inicialización de la variable int n = 1;** (Initialization of the variable `int n = 1;`) points to `(int n = 1;`.
- Condición n <= 10** (Condition `n <= 10`) points to `n <= 10;`.
- Incremento n++;** (Increment `n++;`) points to `n ++ )`.



```
for (int n = 1; n <= 10; n++) {
```



```
}
```

Ejecución de las instrucciones o bloque de instrucciones que están dentro de las llaves.

- » La inicialización (`int n = 1;`), se realiza una única vez, antes de la primera iteración.
- » La condición (`n <= 10;`), se revisa antes de comenzar una iteración.
- » El incremento o decremento de la variable de control (`n++`), se realiza cada vez que termina una iteración.

## Ejemplos

1. Programa que imprime los diez primeros números naturales, utilizando el ciclo for.

```
/**
 * Clase que implementa la impresión de los n primeros números naturales
 * utilizando el ciclo for
 */
```

```
public class NumeroNatural {

    private int n;

    public NumeroNatural(int n) {
        this.n = n;
    }
    public int getN() {
        return n;
    }
}
```

```

public void setN(int n) {
    this.n = n;
}

public void imprimirNumerosNaturales()
{
    /**
     * En el ciclo for se evalúa x mientras que x <= n
     * se imprime el valor de x e incrementa en uno
     */
    for (int x = 1; x <= n; x++) {
        System.out.println(" " + x);
    }
}
}

```

La clase PruebaNumeroNatural es idéntica al ejemplo anterior, nótese la clase NumeroNatural los métodos constructor, set y get son idénticos.

2. Programa que cuenta el número de veces que aparece la vocal e en una cadena determinada, haciendo uso del ciclo for.

```

public class ContarVocal {
    /** atributo de la clases */
    private String cadena;
    /**constructor que inicializa la cadena con un espacio en blanco */
    public ContarVocal( ) {
        this.cadena = " ";
    }
    /**
     * Método que actualiza el contenido de la cadena */
    public void setCadena(String cadena) {
        this.cadena = cadena;
    }
}

```

```

/**
 * Método que retorna el valor de la cadena */
public String getCadena( ) {
    return cadena;
}
/** Método que cuenta el numero de apariciones de la vocal e
 */
public int obtenerCantidadVocal( ) {
    int cantidad = 0;
    for (int i = 0; i < cadena.length(); i++) {
        if(cadena.charAt(i)=='e') {
            cantidad ++;
        }
    }
    return cantidad;
}
}

```

La clase PruebaContarVocal es idéntica al ejemplo anterior, nótese la clase ContarVocal los métodos constructor, set y get son iguales a la del ejemplo del ciclo while.

## 7.4 COMPARACIÓN DE LOS CICLOS WHILE, DO-WHILE, FOR

Los tres ciclos son estructuras repetitivas, como se puede observar en el ejemplo del programa que imprime los diez primeros números naturales.

### Ciclo for

```

public class NumeroNaturalFor {
    public void imprimirNumero() {
        //inicialiación; condición; incremento
        for(int numero = 1; numero <= 10; numero++) {
            //instrucción que se repite
            System.out.println(numero); //impresión
        }
    }
}

```

## Ciclo while

```
public class NumeroNaturalWhile{
    public void imprimirNumero ()
    {
        int numero = 1;
        do{

            System.out.println(""+numero);
            numero++;
        } while (numero <= 10);
    }
}
```

## Ciclo do-while

```
public class NumeroNaturalDoWhile{
    public void imprimirNumero ()
    {
        int numero = 1;
        do{
            System.out.println(""+numero);
            numero++;
        } while (numero <= 10);
    }
}
```

## Diagrama de Clases

Se puede utilizar el mismo nombre de la clase, ya que los nombres del método son diferentes.

Los métodos imprimen los diez primeros números naturales 1, 2, 3, ..., 10.

Todos tienen:

- » Inicialización (numero = 1)
- » Condición (numero <=10)
- » Incremento (numero ++)

En el ciclo while se evalúa la condición y después se ejecutan las instrucciones de impresión e incremento.

En el ciclo do while se ejecutan las instrucciones de impresión e incremento, y después se evalúa la condición.

En el ciclo for, inicialización del numero en 1; se evalúa la condición; se ejecutan las instrucciones de impresión y después realiza el incremento.

## 7.5 CICLO FOR EACH (FOR MEJORADO)

Es un ciclo repetitivo que no utiliza un contador. Su uso es indicado para colecciones de objetos, como por ejemplo: arreglos, matrices, listas, entre otras estructuras, que no se incluyen en este texto.

### Ejemplo

Programa que imprime los días de la semana

```
public class DiaSemana {
    public static void imprimirDiasSemana( ) {
        String semana [ ] = {"Lunes", "Martes", "Miércoles",
                              "Jueves", "Viernes", "Sábado", "Domingo"};
        for (String dia: semana){
            System.out.print(" "+dia);
        }
    }

    public static void main(String[] args) {

        imprimirDiasSemana();
    }
}
```

## 7.6 USO DE LOS CICLOS REPETITIVOS

Determinar cuándo utilizar uno u otro ciclo repetitivo es un proceso complejo, porque existen equivalencias entre los ciclos, lo que hay son recomendaciones y nociones generales de cuando usar alguno de ellos:

- » Un ciclo for se puede utilizar cuando se conoce cuántas iteraciones son necesarias realizar y sobre todo cuando la variable que hace las veces de contador no debe ser modificada.
- » El ciclo while, cuando la variable de control de ciclo debe incrementarse/decrementarse según una o varias condiciones.
- » Un ciclo do...while, cuando necesita ejecutar las sentencias del ciclo repetitivo, por lo menos una vez.
- » El for each para realizar tareas de consulta de datos y no para asignación de valores.

## 7.7 LECTURAS RECOMENDADAS

- » Estructuras repetitivas o ciclos.
- » Establecer las principales diferencias entre las siguientes versiones del ciclo for:

```
inicializadores;
for( ; condiciones; incrementos) {
instrucciones;
}
```

```
inicializadores;
for( ; condiciones; ) {
instrucciones;
incrementos;
}
```

```
inicializadores;
for( ; ; ) {
// debe tener un break para romper el ciclo
instrucciones;
incrementos;
}
```

## 7.8 PREGUNTAS Y EJERCICIOS DE REVISIÓN DE CONCEPTOS

- » Importancia de los ciclos en java.
- » Realizar cuadro comparativo con las semejanzas y deferencias de los ciclos.
- » Cómo identificar que se debe aplicar un ciclo para resolver un problema.

## 7.9 EJERCICIOS

- » Establezca la diferencia entre contador y acumulador.
- » Aplicación que genera la impresión de los 10 primeros números pares.
- » Implementar un método que imprima los n primeros múltiplos del 7.
- » Diseñar el método que imprima la sumatoria de los n primeros números impares.
- » Generar la impresión de cualquier tabla de multiplicar, donde el multiplicador vaya hasta 12.
- » Implementar una aplicación que permita contar el número de vocales de una cadena.
- » Las claves de acceso son la barrera más común, para evitar el acceso no autorizado a un sistema, son un recurso informático de gran importancia para el acceso a una aplicación, para realizar transacciones en un cajero electrónico. Se requiere de una aplicación que implemente una clave de acceso, para identificar si la clave digitada corresponde con la clave que se tiene con antelación almacenada en un atributo.
- » Construir una aplicación que permita determinar el número de apariciones de las vocales, incluyendo mayúsculas y minúsculas que existen en una cadena de caracteres, haciendo uso del método constructor con parámetros.

## 7.10 REFERENCIAS BIBLIOGRÁFICAS

Aguilar, L. J. (2004). *Fundamentos de programación algoritmos y estructura de datos*. Tercera Edición. Ciudad: México: McGraw Hill.

Barnes D., Kölling M. (2007), *Programación orientada a objetos con Java*. Ciudad Madrid, Prentice Hall.

Bertrand M. (1999). *Construcción de software orientado a objetos* (2ª edición) Ciudad: Madrid, Prentice Hall.

Booch, G. (1996). *Análisis y diseño orientado a objetos con aplicaciones*. Ciudad: México Addison Wesley.

Deitel y Deitel. (2012). *Cómo programar en Java*. Ciudad: México Editorial Pearson (Prentice Hall).

Fowler, M., y Scott, K. UML. (1999). *Gota a gota*. Ciudad: México Editorial Pearson.

Graig, L. (2003). *UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado* (2ª edición) Ciudad: México Prentice Hall.

Rumbaugh, J., Jacobson, I., y; Booch, G.: (2005). *The Unified Modeling Language User Guide*, San Francisco: Addison-Wesley, Reading, Mass. ISBN-13: 078-5342267976 ISBN-10: 0321267974

Villalobos, J., y Casallas, R. (2006). *Fundamentos de programación, aprendizaje activo basado en casos*. Ciudad: México Editorial Pearson.







# CAPÍTULO 8

**Relaciones entre**

**CLASES**



## 8.1 TEMÁTICA A DESARROLLAR

- » Cardinalidad.
- » Asociación.
- » Agregación.
- » Composición.
- » Generalización.
- » Dependencia.

## 8.2 INTRODUCCIÓN

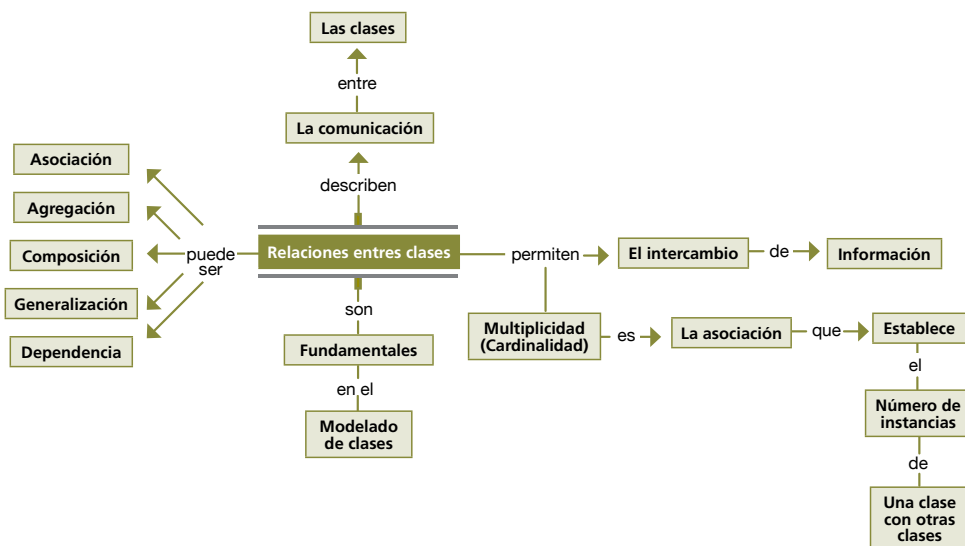
En una aplicación de software, lo más normal es que se involucren dos o más clases, estas clases de alguna manera deben relacionarse entre sí; en la construcción de un proyecto de software están presentes los diagrama de clases los cuales representan la estructura estática de un sistema en términos de clases y de relaciones entre estas.

En el modelado orientado a objetos las relaciones entre clases más importante son:

- » **Asociación:** es una conexión entre clases, que implica la existencia de una relación estructural entre objetos de esas clases.
- » **Generalización:** es una relación entre una clase más general y una más específica o especializada.
- » **Dependencia:** es una relación de uso entre clases.

El mapa mental de la figura 22 describe un poco más pormenorizado el concepto de relación entre clases.

**Figura 22.** Concepto de relaciones entre Clases.



Este capítulo pretende introducir hacia el tema de modelado de clases que serán aplicados en las temáticas de los capítulos subsiguientes.

## 8.3 ELEMENTOS ENTRE LAS RELACIONES DE CLASES

### 8.3.1 Cardinalidad

La cardinalidad hace referencia la cantidad de objetos que participan en la relación determinada. Existen tres tipos de multiplicidad o cardinalidad a saber:

- Uno a Uno.
- Uno a Muchos.
- Muchos a Muchos.

La tabla 14 muestra las diferentes notaciones para la lectura de los tipos de cardinalidad.

**Tabla 14.** Tipos de cardinalidad.

NOTACIÓN	LECTURA
1	Exactamente Uno
*	Muchos
0...1	Cero a uno
0...*	Cero a muchos
1...*	Uno a Muchos (al menos uno)
M...N	De M hasta N (enteros naturales)

La cardinalidad es una restricción a una asociación, que limita el número de objetos relacionados de una clase como, por ejemplo:

- Con un número fijo: 1.
- Con un intervalo de valores: 2..5.
- Con un rango en el cual uno de los extremos es un asterisco. Significa que es un intervalo abierto. Por ejemplo, 2..\* significa 2 o más.
- Con una combinación de elementos como los anteriores separados por comas: 1, 3..5, 7, 15..\*.
- Con un asterisco (\*) En este caso indica que puede tomar cualquier valor (cero o más).

## Ejemplos:

1. La universidad tiene un rector y el rector dirige la universidad (relación uno a uno).



2. La universidad tiene muchos estudiantes.



## 8.3.2 ASOCIACIÓN

Una asociación representa una relación entre clases y es el mecanismo que permite a los objetos comunicarse entre sí. Describe la conexión entre diferentes clases (la conexión entre los objetos reales se denomina conexión de objetos o enlace).

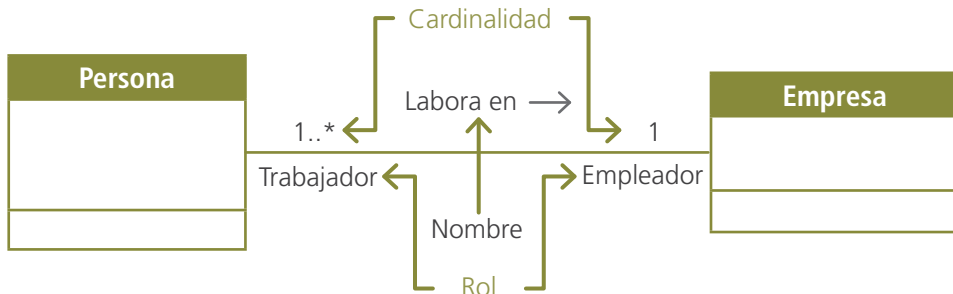
Las asociaciones pueden ser unidireccionales o bidireccionales (indicando si los dos objetos participantes en la relación pueden intercambiar mensajes entre sí, o es únicamente uno de ellos el que recibe información del otro). Cada extremo de la asociación también tiene un valor de cardinalidad, que indica cuántos objetos de ese lado de la asociación están relacionados con un objeto del extremo contrario.

En UML, las asociaciones se representan por medio de líneas que conectan las clases participantes en la relación, y también pueden mostrar el papel y la multiplicidad de cada uno de los participantes. La multiplicidad se muestra como un rango [mínimo-máximo] de valores no negativos, con un asterisco (\*) representando el infinito en el lado máximo.

Las asociaciones tienen algunos elementos o atributos, que están presentes, como lo son:

- » **Nombre:** describe de una forma más específica la naturaleza de la relación entre los objetos. No suele ser necesario, salvo que haya varias asociaciones, por ejemplo, 'trabaja en' o 'dirige' para una asociación entre empleados y departamentos.
- » **Roles:** define el nombre de un objeto desde el otro. Por ejemplo, trabajador y empleador para una relación entre varias Personas y Empresa.

- » **Cardinalidad:** permite definir el número de objetos de ambos tipos. Por ejemplo en una empresa laboran muchos trabajadores.

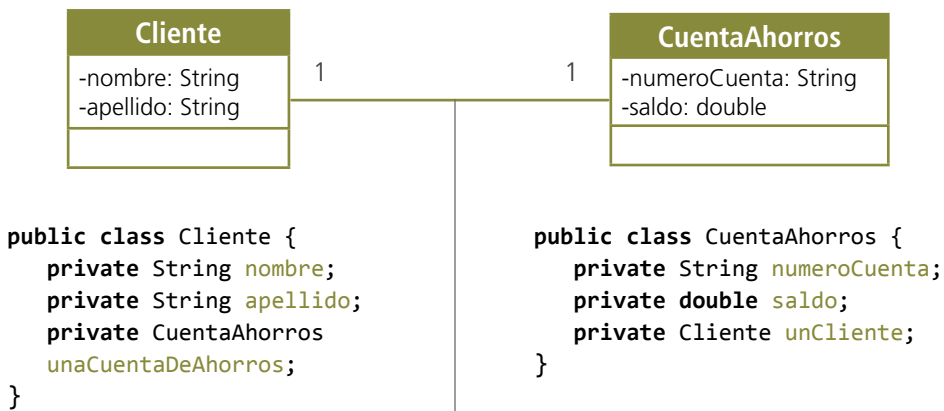


En programación orientada a objetos, el comportamiento de un sistema se define en términos de interacciones entre objetos; es decir, intercambios de mensajes. Enviar un mensaje habitualmente resulta en la invocación de una operación en el receptor. Las asociaciones son necesarias para la comunicación, ya que los mensajes son enviados a través de las asociaciones; sin asociaciones, los objetos quedarían aislados e incapaces de interactuar.

## Ejemplo

Un cliente tiene una cuenta de ahorros. Asociación bidireccional con cardinalidad

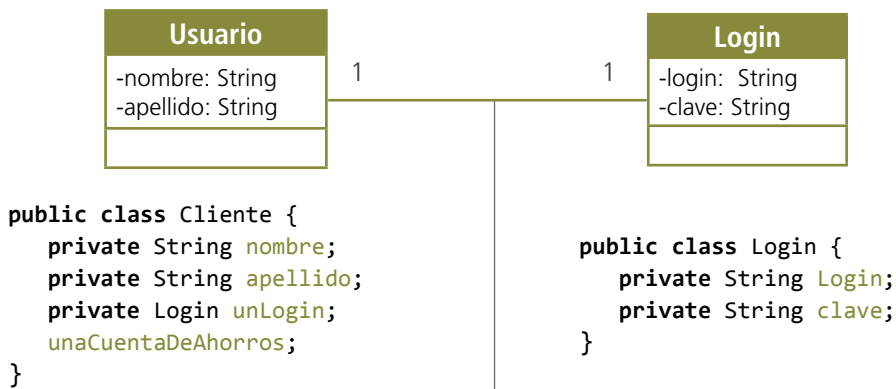
1. 1-1.





## 2. Asociación direccional con cardinalidad 1-1.

Relación direccional que indica que el Usuario tiene un Login. Conformado por un login y una clave.



### 8.3.3 Agregación

Es una relación del tipo todo-partes, donde los objetos de una clase están compuestos por objetos de otra clase. Por ejemplo, un automóvil está compuesto por motor, ruedas, puertas, entre otros.

La representación en UML el símbolo de agregación es un diamante colocado en el extremo en el que está la clase que representa el "todo".



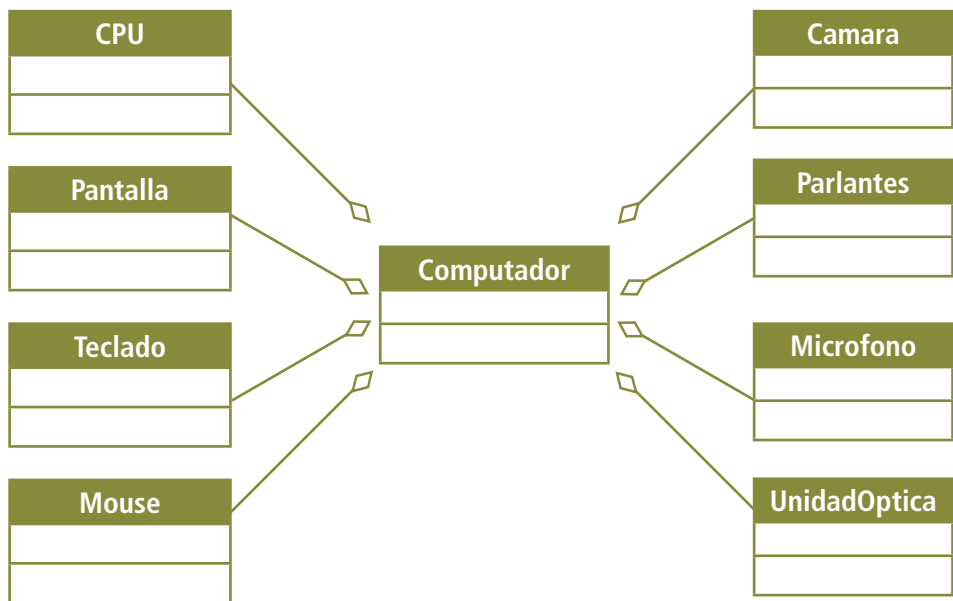
La ClaseB está conformada por objetos de tipo A (uno-muchos).

### Ejemplos

- Un automóvil tiene llantas.
- Un teléfono celular tiene un display o pantalla.
- Se tienen las clases cliente y empresa donde una empresa reúne a varios clientes.



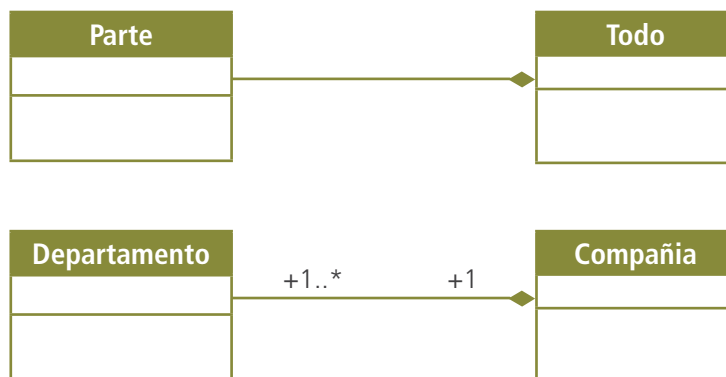
- Un computador es el resultado de agregación, ya que está integrado por una serie de dispositivos, como la Unidad Central de Proceso (CPU), un monitor o pantalla un teclado, un mouse, una cámara, dos parlantes, micrófono, unidad óptica (CD, DVD, Blu Ray), entre otros. En otras palabras se puede aplicar una prueba a un objeto de la clase computador ya que está conformado o tiene una CPU, un teclado, un mouse, una cámara, dos parlantes, un micrófono unidad óptica. El modelo de clases sería el siguiente:



### 8.3.4 La composición

La composición es un tipo especial de agregación, que denota una fuerte dependencia de la Clase "Todo", a la Clase "Parte". Se grafica con una flecha rombo de diamante relleno, desde la parte hacia el todo.

## Ejemplo



### La composición se caracteriza porque:

- Es dependencia fuerte ya que el elemento dependiente desaparece al destruirse el que lo contiene y, si es de cardinalidad 1. Es creado al mismo tiempo.
- Hay una pertinencia fuerte. Se puede decir que el objeto contenido es parte constructiva y vital del que lo contiene.

## 8.4 LA RELACIÓN DE ESPECIALIZACIÓN/GENERALIZACIÓN

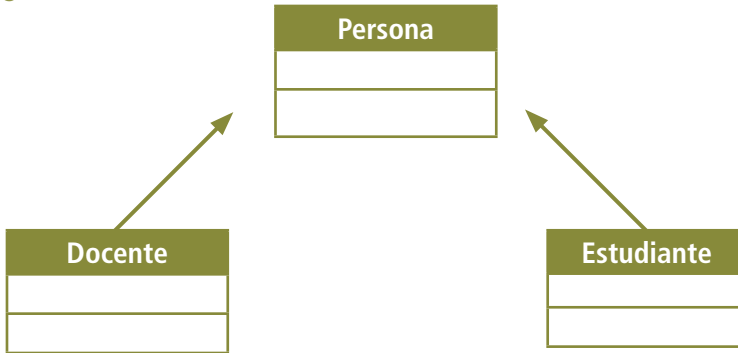
Esta relación indica que una clase (clase derivada) hereda los métodos y atributos especificados por una clase (clase base), por lo cual una clase derivada además de tener sus propios métodos y atributos, podrá acceder a las características y atributos visibles de su clase base (public y protected).

### Características

- » La generalización es la relación entre una clase (superclase) y una o más versiones especializadas (subclases).
- » Los atributos y métodos comunes a un grupo de subclases se asocian a la superclase y son compartidos por las subclases.
- » Se dice que cada subclase hereda las características de la superclase, aunque también puede agregar, es decir, incluir sus propios atributos y operaciones específicas.

- » La notación UML para la generalización es una flecha que conecta la superclase con sus subclases, apuntando a la superclase.
- » Junto a la punta de flecha en el diagrama pueden colocarse un discriminador, un atributo de tipo enumeración que indica la propiedad de un objeto que se abstrae en una relación de generalización particular (visualizado en el ejemplo).

### Ejemplo



En este diseño de clases se especifica que las clases Estudiante y Docente heredan de la clase Persona; es decir, Estudiante y Docente podrán acceder a las características de Persona. También puede tener su respectiva diferenciación, ya que un Estudiante puede obtener sus notas previa evaluación realizada por parte de un Profesor. Otra forma de identificar la relación desde la lectura es: todo estudiante es una persona y todo docente es una persona.

## 8.5 DEPENDENCIA

Cuando objetos de una clase utilizan objetos de otra clase, existe una relación de dependencia entre las respectivas clases.

En UML esta relación se representa en el diagrama de clases con una flecha y trazo discontinuo en el sentido de las clases, cuyos objetos usan los de otra clase, dependen de la especificación de la clase usada. Si cambia la especificación, habrá que hacer cambios en las clases que la usan.



### Del diagrama de clases se puede observar que:

- » La ClaseA usa a la ClaseB.
- » La ClaseA depende de la ClaseB.
- » Dada la dependencia, todo cambio en la ClaseB podrá afectar a la ClaseA.
- » La ClaseA conoce la existencia de la ClaseB pero la ClaseB desconoce que existe la ClaseA.

Las dependencias generalmente se utilizan para indicar que una clase utiliza a otra como argumento en alguna operación o sus objetos utilizan alguna de las operaciones de la otra clase.

### Ejemplo

Se tiene la clase Documento la cual incluye al atributo texto y la clase ImpresoraLaser se encarga de imprimir los Documentos:



## 8.6 LECTURAS RECOMENDADAS

- » Grado de la asociación.
- » Técnicas comunes del modelamiento.

## 8.7 PREGUNTAS DE EVALUACIÓN

- » ¿Qué es el modelado de colaboraciones simples?
- » ¿Qué es la ingeniería directa?
- » ¿Qué es la ingeniería inversa?

## 8.8 EJERCICIOS

### Dependencia

1. Completar la información de la siguiente tabla.

DESCRIPCIÓN	NOTACIÓN
Un vuelo necesita una reservación.	
Un vuelo tiene muchas reservaciones.	
Un vuelo tiene un horario.	
Un vuelo tiene un origen.	
Un vuelo pertenece a una aerolínea.	
Un pasajero compra un ticket.	
Un pasajero compra muchos tickets.	
Un pasajero tiene un pasaporte.	
Un equipo de fútbol tiene muchos jugadores.	
Una persona tiene una cédula.	
Una empresa tiene cero o más empleados.	

2. Plantear tres ejemplos para cada uno de los siguientes enunciados y los correspondientes diagramas de clases.
  - Número fijo:  $m$  ( $m$  denota el número).
  - Describiendo una multiplicidad de "uno-uno".
  - Describiendo una multiplicidad de "uno-muchos".

## Agregación

1. Completar la información de la siguiente tabla que resume algunas diferencias entre agregación y composición.

CARACTERÍSTICA	AGREGACIÓN	COMPOSICIÓN
Representación	Rombo sin color	Rombo relleno
Puede tener varias asociaciones que comparte los componentes	Sí	
Destrucción de los componentes al destruir el compuesto		
Cardinalidad a nivel del compuesto		

2. Construir un diagrama de clases en donde se identifique que un auto está integrado por ruedas, caja de cambios, puerta y motor.
3. Modelar en un diagrama de clases un Computador conformado por la CPU, el teclado, el monitor y el mouse.
4. Construir una relación de clases que permita modelar la relación: dada por la frase “la universidad tiene varias facultades”.
5. Elaborar el diagrama de clases de la relación que existe entre un texto o libro y sus páginas.

## Asociación

1. Construir el diagrama de clases de la asociación Juego Participante, donde el juego:
  - a. Tiene dos participantes.
  - b. Tiene cuatro participantes.

2. Construir el diagrama de clases y el código en java para una relación en donde un cliente puede tener asociadas muchas órdenes de compra y una orden de compra solo puede tener asociado un cliente.
3. Modelar la relación Empresa-Empleado a partir de diagramas de clases donde un empleado puede trabajar en varias empresas (multi-empleo) y, por supuesto, que una empresa puede estar formada por más de un empleado.
4. Consultar los tipos de asociación que existen y navegabilidad.
5. Elaborar el diagrama de clases donde un gerente dirige un departamento, y un departamento tiene varios empleados y varios empleados trabajan en varios proyectos.

## Agregación

1. Construir el modelo de clases en donde se tienen las clases Agenda y Contacto, donde una Agenda agrupa varios Contactos.
2. Elaborar un diagrama de clases que modele la relación entre un avión y sus dos alas.
3. Diseñe un diagrama de clases que permita modelar las relaciones dadas entre la universidad, cursos, estudiantes y maestros que describen una universidad con muchos estudiantes, estos asisten a diferentes cursos, que son orientados por diferentes maestros.

## Generalización

1. Se requiere construir el modelo de diagrama de clases para el siguiente enunciado: Una colección de autos, conformada por autos deportivos, ambulancias, camiones. Todas estas clases podrían derivar de una clase Automóvil, de manera que se aprovechase la definición común de parte de la implementación. Cada clase añadiría los detalles relativos a esa clase particular. Así la clase ambulancia añadiría el método encenderSirena(), o la clase camión podría tener la propiedad booleana estaCargado.



2. Una compañía editorial produce una serie libros impresos como audio-libros en discos compactos. Se requiere construir el modelo de clases conformado por:
  - » Una clase denominada Publicación que registra el título del libro y el precio de una publicación. A partir de esta clase, derive dos clases:
    - La clase Libro a la cual le agregue el número de páginas (numero entero) y
    - La clase CD, a la cual le agregue el tiempo de reproducción en minutos (numérico real).

Cada una de las clases debe tener propiedades para acceder a sus respectivos datos. Elabore un diagrama de clases UML indicando las relaciones de herencia y codifique un sistema mediante el cual se generen instancias de las clases Libro y CD, donde el usuario capture sus datos y se inserten en los respectivos objetos.

## Dependencia

1. Diseñar el diagrama de clases que modele la relación de las entidades Curso y PlanificacionCurso. Donde la clase PlanificaciónCurso implementa los métodos adicionarCurso y cancelarCurso los cuales reciben como parámetro un objeto de la clase Curso y se aplica el concepto de dependencia.
2. Codificar en java el anterior modelo de clases.

## 8.9 REFERENCIAS BIBLIOGRÁFICAS

Weitzenfeld, A. (2005). *Ingeniería de software orientada a objetos con UML, Java en Internet*. Ciudad: México, Thomson.

Booch, G. (1996). *Análisis y diseño orientado a objetos con aplicaciones*. Ciudad: México Addison Wesley.





# CAPÍTULO 9

**Abstracción y**

**ENCAPSULAMIENTO**



## 9.1 TEMÁTICA A DESARROLLAR

- » Abstracción.
- » Encapsulamiento.

## 9.2 INTRODUCCIÓN

La Programación Orientada a Objetos (POO) es un paradigma de programación que usa objetos y sus interacciones para el diseño y construcción de aplicaciones de software por computador. Es un paradigma, ya que existen metodologías de soporte como OpenUP, Scrum, XP, elementos que lo representan como los objetos y herramientas tecnológicas que lo soportan como C++, Java, C#, entre otras.

Los pilares de la programación orientada a objetos son abstracción, encapsulamiento y polimorfismo. Son los elementos más importantes que deben tener los objetos de software, para cumplir con el paradigma de orientación a objetos.

## 9.3 ABSTRACCIÓN

La abstracción se puede considerar como la capacidad de integrar las propiedades y comportamientos necesarios para la correcta representación del objeto dentro del sistema. Es decir no se tienen en cuenta aspectos del sistema en estudio que no son de interés, con el objeto de concentrarse en aquellos que son relevantes para modelar el sistema.

Dentro de las propiedades o características esenciales se encuentran los atributos (o datos) y los comportamientos que corresponden a los métodos. Al modelar, se piensa en objetos y es necesario tomar las características y propiedades de un elemento real y llevarlo a un objeto. La abstracción es base fundamental para el diseño de las aplicaciones de software.

Java proporciona un amplio conjunto de clases útiles para desarrollar aplicaciones, como es el caso de las colecciones, que encuentran definidas en el paquete `java.util` y las cuales representan un grupo de objetos, que se pueden emplear en diferentes aplicaciones. Para utilizarlas en un programa fuente Java se deben importar haciendo uso de `import`.

Para las estructuras de datos el programador puede contar con una serie de recursos de software y en tal caso, se tiene el tipo de dato abstracto (TAD), el TAD Pila, el TAD Cola y otras estructuras de uso común en una aplicación con manejo dinámico de memoria.

Las API de Java brindan un conjunto de clases útiles para realizar toda clase de tareas necesarias dentro de un programa. La API proporciona una interfaz de Java con un conjunto de métodos y clases de utilidad que se puede utilizar para escribir aplicaciones en Java (java.lang, java.io, java.sql); esto es otra forma de abstracción.

La abstracción es un principio de la programación orientada a objetos y una es una herramienta que permite tratar la complejidad y el entorno del problema. En general, una aplicación de software es una descripción abstracta de una actividad o procedimiento que sucede en el mundo real.

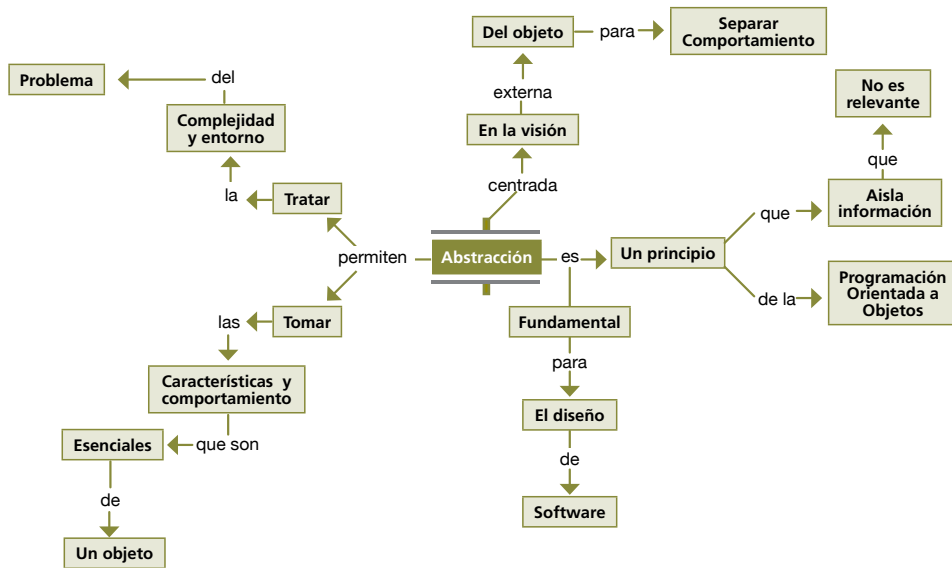
## Ejemplos

- » Las diferentes escalas en mapas.
- » Un automóvil es una entidad del que se identifican características como el color, marca, el modelo, el precio, la placa, entre otros. Y algunos de sus métodos o funciones típicas de esta entidad son el girar a la derecha, girar a la izquierda, frenar, encender, retroceder, avanzar.

La gerencia de un parqueadero requiere un sistema para controlar los vehículos que ingresan a sus instalaciones. En este caso, dentro las características de la clase Vehículo están: la marca, el modelo, el propietario: número de documento, nombres y apellidos, dirección, teléfono y correo.

La figura 23 del Concepto de Abstracción complementa este principio de la Programación Orientada a Objetos.

Figura 23. Concepto de Abstracción.



## 9.4 ENCAPSULAMIENTO

El encapsulamiento significa reunir a todos los elementos bajo un mismo nombre, esto es atributos y operaciones que pueden considerarse pertenecientes a una misma entidad.

El encapsulamiento corresponde a la posibilidad que tiene un objeto de ocultar los datos y los métodos que le son propios y que serán solo accesibles dentro del propio objeto.

La abstracción y el encapsulamiento están relacionados ya que la abstracción propende por determinar las características y comportamientos de un objeto y el encapsulamiento se ocupa en la implementación establecida por el comportamiento del objeto.

En java, el encapsulamiento garantiza que los usuarios de un objeto no pueden modificar el estado del objeto sin usar interfaz (correspondiente al conjunto de métodos que son accesibles a otros objetos). Una ventaja del encapsulamiento es que



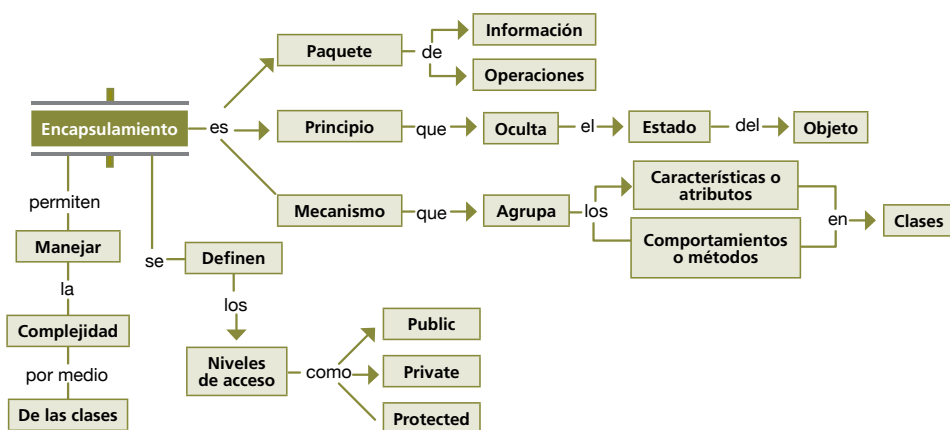
si un módulo cambia internamente sin modificar su interfaz, el cambio no supondrá ninguna modificación del sistema.

El encapsulamiento en un sistema orientado a objeto se representa en cada clase u objeto, definiendo sus atributos y métodos con los siguientes modos de acceso:

- » public representado con el signo más (+), tanto para atributos como métodos los cuales son accesibles dentro y fuera de la clase pudiendo ser llamados por cualquier clase.
- » private representado con el signo menos (-), ya sean atributos o métodos los cuales solo son accesibles dentro de la misma clase.
- » protected representado con el signo numeral (#) para los atributos o los métodos que son accesibles dentro de la implementación de la misma clase y sus clases derivadas o subclasses (clases hijas).
- » Por default, sucede cuando no se especifica ningún modificador de acceso se utiliza el nivel de acceso por defecto, que consiste en que el elemento puede ser accedido sólo desde las clases que pertenezcan al mismo paquete.

La figura 24 del Concepto de Encapsulamiento complementa este principio de la Programación Orientada a Objetos.

**Figura 24.** Concepto de Encapsulamiento.



## Ejemplos

1. El reproductor de DVD muestra las acciones que se pueden realizar en el tablero de mando o el control remoto, de modo que play, pause, stop, review, zoom, entre otros; el usuario utiliza el DVD por lo que ofrece y no ingresa en el interior para hacer nuevas acciones o realizar alguna de ellas.
2. La clase Persona encapsula los atributos del nombre, la edad, el método constructor y los métodos set y get.

```
class Persona {  
    private String nombre;  
    private int edad;  
  
    public Persona(String nombre, int edad) {  
        super();  
        this.nombre = nombre;  
        this.edad = edad;  
    }  
  
    public String getNombre() {  
        return nombre;  
    }  
  
    public void setNombre(String nombre) {  
        this.nombre = nombre;  
    }  
  
    public int getEdad() {  
        return edad;  
    }  
  
    public void setEdad(int edad) {  
        this.edad = edad;  
    }  
}
```

### Persona

```
-nombre: String
-edad: int

+Persona(nombre: String, edad: int)
+getNombre( ): String
+setNombre(nombre: String)
+getEdad( ): int
+setEdad(edad: int)
```

La tabla 15 establece un comparativo entre los conceptos de encapsulamiento y abstracción.

**Tabla 15.** Comparación entre abstracción y encapsulamiento.

ABSTRACCIÓN	ENCAPSULAMIENTO
La abstracción resalta los aspectos importantes de un objeto.	Exige que sus características y métodos estén bien definidos y no se confundan con los de otros.
No se incluyen detalles sobre la implementación de las operaciones.	Oculto los detalles internos de un objeto.
A través de la abstracción conseguimos extraer las cualidades principales sin detenernos en los detalles.	Especifica los datos y comportamiento lógico que está oculto en un objeto.
Es una simplificación o modelo de un concepto complejo, proceso o un objeto del mundo real.	Permite la visibilidad de atributos y/o métodos, en una determinada clase, a partir de los modificadores de acceso private, protected y public.
Permite simplificar la comprensión del mundo del problema. Es la forma general de ver a un objeto, sin conocer en su composición interior u otros componentes.	La encapsulación en programación significa agrupar a todos los componentes de un objeto en uno solo.
Permite resaltar las características y comportamientos relevantes de un objeto.	Corresponde a agrupar a todos los componentes de un objeto en uno solo.

## 9.5 LECTURAS RECOMENDADAS

- » Paradigma de Programación Orientada a Objetos.
- » Principios de la Programación Orientada a Objetos.

## 9.6 PREGUNTAS DE REVISIÓN DE CONCEPTOS

- » ¿Cómo identificar que se debe aplicar abstracción para resolver un problema?
- » ¿Cómo identificar que se debe aplicar encapsulamiento para resolver un problema?

## 9.7 EJERCICIOS

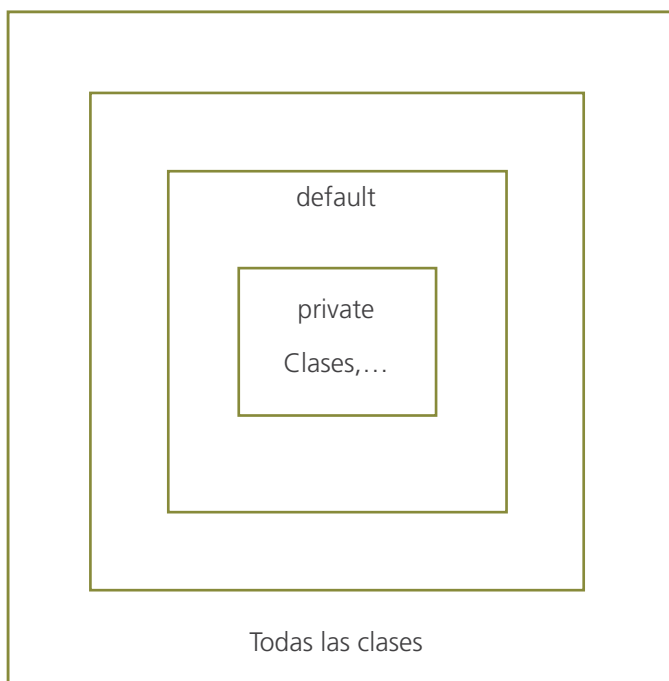
### Abstracción

- » Determinar las características que se pueden abstraer de los estudiantes de la universidad.
- » Obtener las características de los computadores del laboratorio y sus comportamientos.
- » Obtener las características de los pupitres de un salón y sus comportamientos.
- » Liste siete elementos que usted emplea normalmente sin conocer como es el funcionamiento interno.

### Encapsulamiento

1. Identificar y nombrar las características y operaciones de un celular.
2. Reconocer y nombrar las características y operaciones de un libro.
3. Modelar los atributos y operaciones de un computador.

- » Completar la información de la siguiente gráfica en base a los modificadores de acceso y que elementos involucran:



- » Completar la información de la siguiente tabla.

MODIFICADORES DE ACCESO				
	LA MISMA CLASE	OTRA CLASE DEL MISMO PAQUETE	SUBCLASE DE OTRO PAQUETE	OTRA CLASE DE OTRO PAQUETE
public				
protected				No
default			No	
private		No		

## 9.8 REFERENCIAS BIBLIOGRÁFICAS

Barnes D., Kölling M. (2007), *Programación orientada a objetos con Java*. Ciudad Madrid, Prentice Hall.

Booch, G. (1996). *Análisis y diseño orientado a objetos con aplicaciones*. Ciudad: México Addison Wesley.

Deitel y Deitel. (2012). *Cómo programar en Java*. Ciudad: México Editorial Pearson (Prentice Hall).

Fowler, M., y Scott, K. UML. (1999). *Gota a gota*. Ciudad: México Editorial Pearson.

Villalobos, J., y Casallas, R. (2006). *Fundamentos de programación, aprendizaje activo basado en casos*. Ciudad: México Editorial Pearson





# CAPÍTULO 10



**HERENCIA**





## 10.1 TEMÁTICA A DESARROLLAR

- » Herencia.
- » Superclase.
- » Subclase.
- » Protected.
- » Extends.
- » Super.

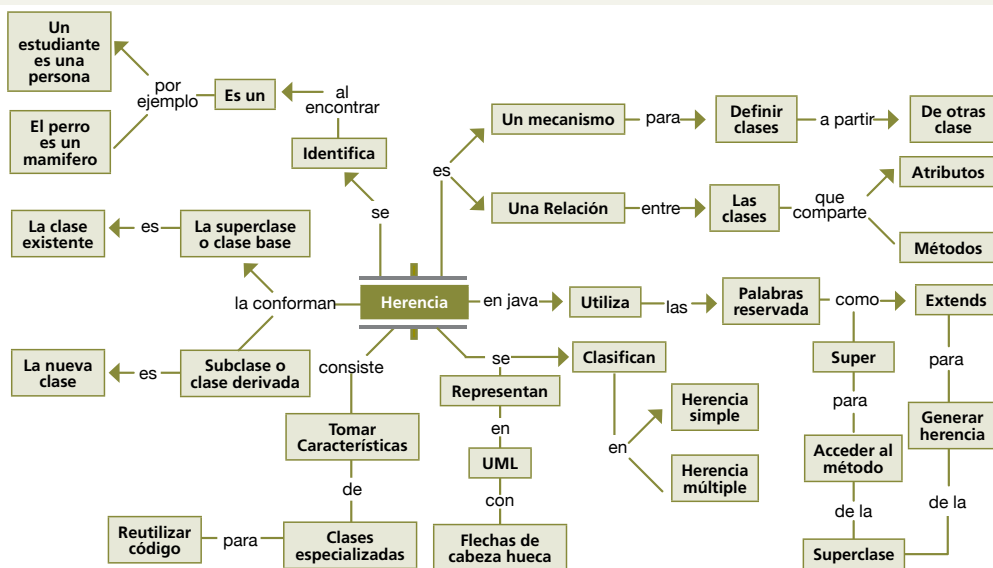
## 10.2 INTRODUCCIÓN

La Programación Orientada a Objetos (POO) es una forma especial de programar, más cercana a como se expresan o representan las cosas en la vida real que otros tipos de programación; es un paradigma que integra un conjunto de teorías, estándares, modelos y métodos que permiten modelar el dominio de un problema o proyecto e implementarlo en un lenguaje de programación para brindar la solución a ese requerimiento.

El paradigma orientado a objetos, tiene unas características que lo definen y lo diferencian de los demás paradigmas, como lo son la abstracción, encapsulamiento, modularidad, herencia y polimorfismo entre otras, en este capítulo se aborda el concepto de herencia, donde la idea básica es poder crear clases basadas en clases ya existentes. Cuando se hereda de una clase existente, se está reutilizando código (métodos y atributos). En otras palabras, la herencia permite que los sistemas orientados a objetos puedan definir clases en término de otras clases.

En el mapa mental de la figura 25 se complementa el concepto de Herencia.

**Figura 25.** Concepto de Herencia.



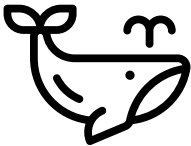

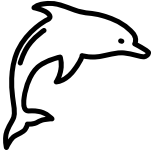
### 10.3 HERENCIA

Es uno de los pilares fundamentales de la programación orientada a objetos, mecanismo que permite definir nuevas clases a partir de otras ya definidas, la idea básica es poder crear clases basadas en clases ya existentes, de forma que heredan las propiedades de la clase base.

La herencia permite implementar las relaciones en las cuales un objeto pertenece a una clase que es hija de otra clase, lo que significa que, la clase hija hereda de su clase padre, la estructura y el comportamiento de esta.

#### Ejemplo: Animales mamíferos

##### Características comunes

BALLENA	PERRO	DELFIN
		
color tamaño peso edad	color tamaño peso edad	color tamaño peso edad

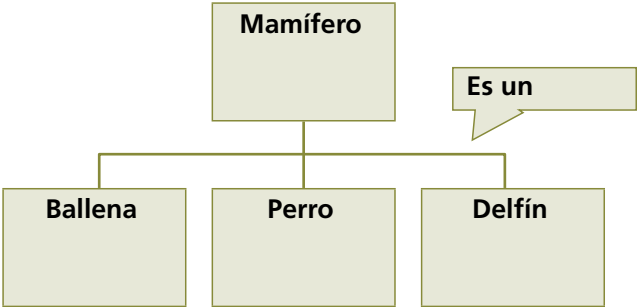
Las características comunes y propias de cada uno pueden ser el color, tamaño, peso, edad y se pueden agrupar en una entidad que puede ser Mamífero.

Una ballena **es un** Mamífero

Un perro **en un** Mamífero




Un delfín **es un** Mamífero

Un perro, delfín y una ballena son objetos diferentes, pero tienen una cosa en común: los tres son mamíferos. Así como lo son los delfines y las ballenas, aunque se muevan en un entorno diferente, entonces un perro es un mamífero, un mamífero es un animal, un animal es un ser vivo, entre otros. Lo cual permite establecer la relación **es un**.



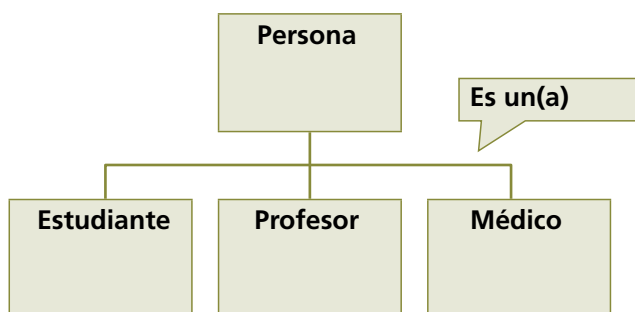
## Ejemplo: Diferentes personas

### Características comunes

ESTUDIANTE	PROFESOR	MÉDICO
		
nombre apellido edad sexo	nombre apellido edad sexo	nombre apellido edad sexo





Se puede observar que se hay características comunes y propias de cada uno (estudiante, profesor y médico). Los atributos comunes permiten agrupar en una sola entidad denominada Persona.

- » Un estudiante **es una** Persona
- » Un profesor **es una** Persona
- » Un médico **es una** Persona

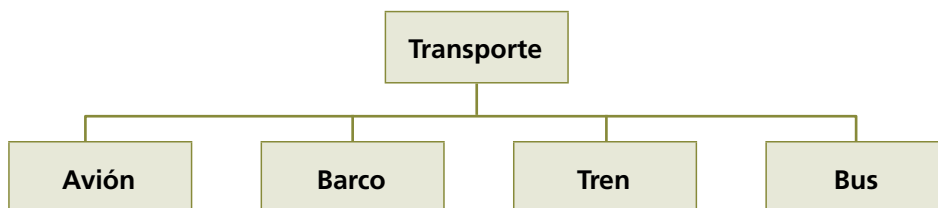


La herencia propende por utilizar una clase ya creada como la clase mamífero para tomar las características (color, tamaño, peso, edad), en clases más especializadas o derivadas, para el caso, perro, ballena, delfín, y a partir de la clase base reutilizar el código que sea común con la clase base y solamente definir nuevos métodos o redefinir algunos de los existentes.

## Ejemplo: medios de transporte

BUS	TREN	AVIÓN	BARCO
			
marca modelo color velocidad	marca modelo color velocidad	marca modelo color velocidad	marca modelo color velocidad

- » Avión **es un** Transporte
- » Barco **es un** Transporte
- » Tren **es un** Transporte
- » Bus **es un** Transporte



Cuando se declara una clase, se definen las características o variables y los métodos o comportamientos que pueden tener los objetos que pueden ser creados a partir de la clase.

Por ejemplo, a partir de la clase Persona, se define nombre, apellido y el género, con sus respectivos constructores y métodos de acceso y modificación de las variables de instancia.

Persona
-nombre: String -apellido: String -genero: char
+Persona( ) +Persona(nombre: String, apellido: String, genero: char) +getNombre(): String +setNombre(nombre: String) +getApellido(): String +setApellido(apellido: String) +getGenero(): char +setGenero(genero: char) +toString(): String

Al definir la clase Estudiante, el cual tiene un nombre, un apellido y género, no es necesario empezar desde cero, se puede decir que la clase Estudiante sería una clase que hereda de la clase Persona. Solamente a la clase Estudiante se anexarían atributo matrícula y los métodos como los constructores y los métodos de acceso y modificación que sean solo específicos al estudiante. Esto significa que cuando se hereda de una clase existente, además de reutilizar atributos, también se puede agregar nuevos atributos y métodos para cumplir con la nueva clase.

Los atributos comunes a todas las clases Persona y estudiante se define en la clase Base. Los métodos comunes en la clase Padre y el atributo matrícula en la clase Estudiante, que es la clase Hija.

Estudiante
-matricula: long
+Estudiante( ) +Estudiante(matricula: long) +setMatricula(matricula: long) +getMatricula( ): long +toString( ): String

Las relaciones de herencia se establecen entre clases, puesto que, cubren a todos los objetos que se crearán como instancias de esas clases. La clase Padre se denomina superclase, es decir, la clase de la cual heredan o se derivan otras clases y las clases hijas, o las que heredan se denominan subclases.

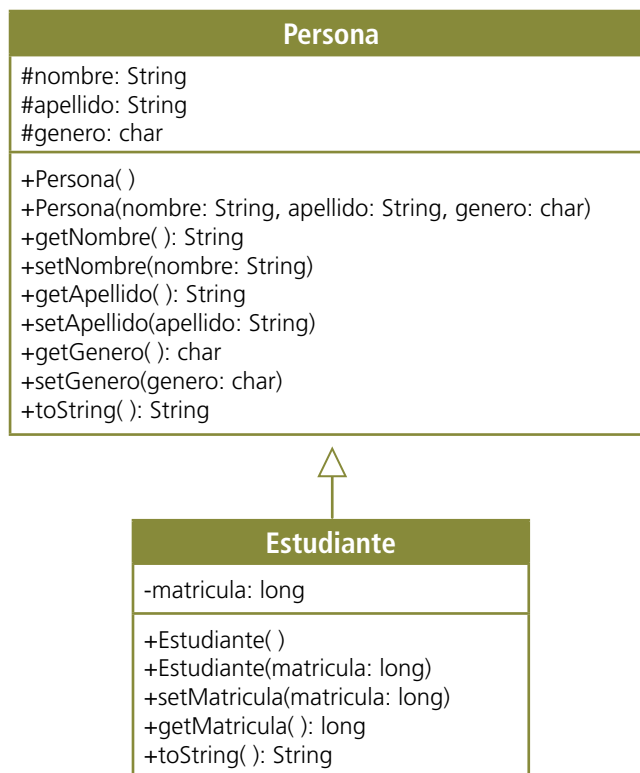
Para el ejemplo la clase Persona es la superclase, la cual agrupa las clases con propiedades y comportamientos comunes y la clase Estudiante es la clase derivada o clase hija.

## 10.4 REPRESENTACIÓN DE LA HERENCIA

En diagramas UML de clases se hace con una flecha de cabeza hueca.



El diagrama de clases sería el siguiente:





Java solo maneja herencia simple, en el ejemplo la subclase Estudiante, comparte la estructura y el comportamiento de una sola superclase Persona, en otras palabras, una subclase puede heredar datos y métodos de una única clase.

El símbolo # que se antecede a los atributos nombre, apellido y género corresponde al modificador `protected`, que puede ser utilizado para un atributo o un método, indicando que es accesible en la clase en donde se define y en cualquiera de sus subclases en otras palabras al heredar si se puede usar desde la clase derivada. Para indicar que una clase deriva de otra, heredando sus propiedades (métodos y atributos), en java se usa la palabra **extends**. Codificando las clases del ejemplo sería:

---

### La superclase, clase base o clase Padre:

```
public class Persona {  
    protected String nombre;  
    . . . . .  
}
```

---

### La subclase, clase derivada o clase hija:

```
public class Estudiante extends Persona{  
    private long matricula;  
    . . . . .  
}
```

---

La palabra reservada `extends`, permite en Java definir la herencia, entonces la sentencia: **public class Estudiante extends Persona**, se está diciendo que la clase Estudiante hereda de la clase Persona.

Cuando se implementan clases que utilizarán la herencia, se emplea la palabra **protected** en lugar de `private`; para definir atributos o métodos donde es usada, será privada para cualquier clase externa, pero podrán ser utilizados como si fuera pública para cualquier clase derivada o clase hija.

La herencia se identifica al encontrar la relación **es-un** entre la nueva clase y la ya existente.

## Ejemplos

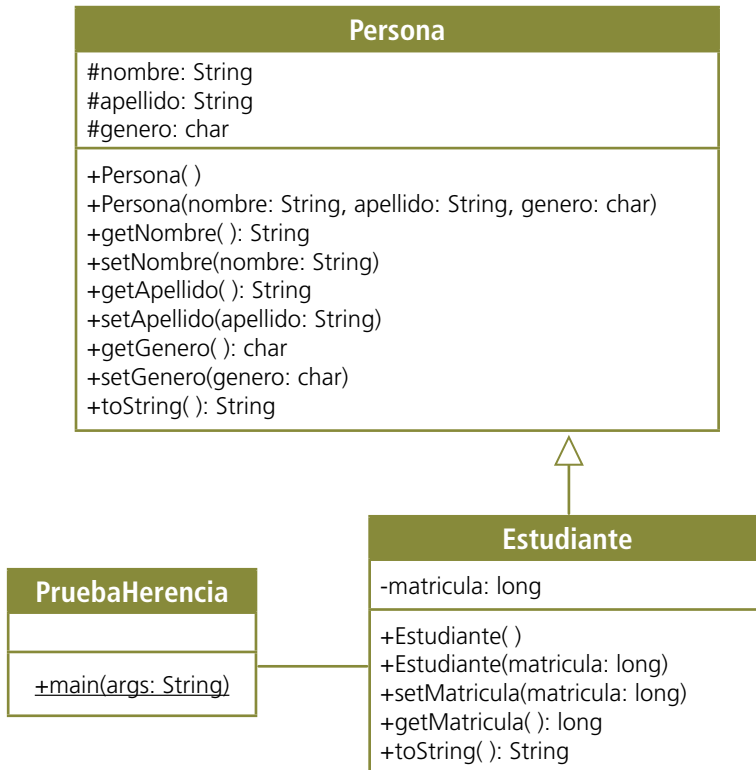
- » Un estudiante **es una** Persona.
- » Una acelga **es una** Verdura, que es un tipo de Vegetal.

- » Un DVD **es una** unidad de almacenamiento, que a su vez es un tipo de componente de un computador.

Al decir que un elemento es un tipo de otro, se establece una especialización:

- » Donde elemento tiene las características generales compartidas y otras propias.
- » La herencia es una manera de representar las características que se reciben del nivel más general.

Codificación de las clases Persona y Estudiante del ejemplo



En las clases **Persona** y **Estudiante** se agruparon los elementos que pueden considerarse pertenecientes a esas entidades, al mismo nivel de abstracción, propiedad de la programación orientada a objetos denominada encapsulamiento.

```
public class Persona {
    protected String nombre;
    protected String apellido;
    protected char genero;

    public Persona() {
        this.nombre = "";
        this.apellido = "";
        this.genero = ' ';
    }

    public Persona(String nombre, String apellido, char genero) {
        this.nombre = nombre;
        this.apellido = apellido;
        this.genero = genero;
    }

    public String getNombre() {
        return nombre;
    }

    public void setNombre(String nombre) {
        this.nombre = nombre;
    }

    public String getApellido() {
        return apellido;
    }

    public void setApellido(String apellido) {
        this.apellido = apellido;
    }

    public char getGenero() {
        return genero;
    }

    public void setGenero(char genero) {
        this.genero = genero;
    }

    @Override
    public String toString() {
        return nombre+" "+apellido;
    }
}
```

```
public class Estudiante extends Persona{
    private long matricula;

    public Estudiante() {
        this.matricula=0;
    }

    public Estudiante(long matricula) {
        this.matricula = matricula;
    }

    public void setMatricula(long matricula) {
        this.matricula = matricula;
    }

    public long getMatricula() {
        return matricula;
    }

    @Override
    public String toString() {
        return ""+matricula;
    }
}

public class PruebaHerencia {
    public static void main(String[] args) {
        Estudiante unEstudiante = new Estudiante();
        unEstudiante.setNombre("Alan");
        unEstudiante.setApellido("Brito");
        unEstudiante.setGenero('M');
        unEstudiante.setMatricula(12345);
        System.out.println("\nEstudiante");
        System.out.println("Matricula: " +
            unEstudiante.getMatricula());
        System.out.println("Nombre: " +
            unEstudiante.getNombre());
        System.out.println("Apellido: " +
            unEstudiante.getApellido());
        System.out.println("Genero: " +
            unEstudiante.getGenero());
    }
}
```

Al heredar de una clase base, se hereda tanto los atributos como los métodos, mientras que los constructores son utilizados, pero no heredados.

## 10.5 SENTENCIA SUPER

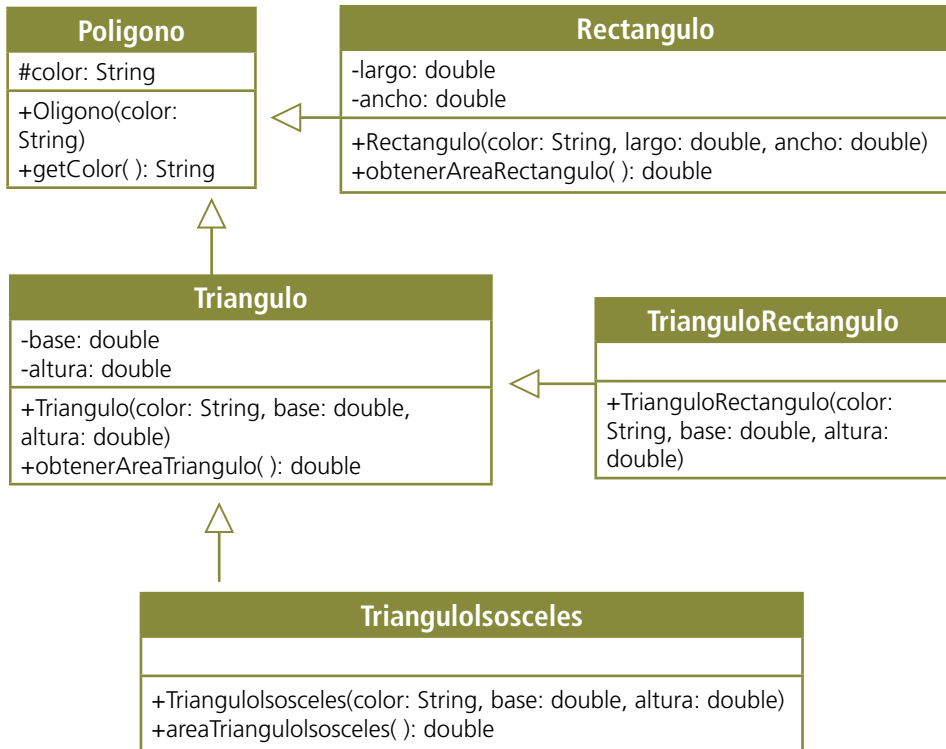
- » La palabra reservada `super` sirve para indicar que una variable o un método es de la superclase, `super()` llama al constructor de la clase de la que se hereda (**extends**). Es de tener en cuenta que todas las clases heredan en última instancia a la clase **Object**.
- » El uso de `super` es opcional pero, en caso de utilizarse, ha de ser obligatoriamente la primera sentencia del constructor. Si no se utiliza, el compilador inserta la sentencia **`super()`** al comienzo de cada método constructor de la subclase.
- » `super` permite acceder, desde la subclase, a los métodos y las variables de la superclase.

### Ejemplo

Programa que calcula el área del rectángulo, de los triángulos rectángulo e isósceles, a partir de la base y la altura e imprimiendo el color de cada figura geométrica.

El modelamiento de las clases se realiza a partir de:

- Un rectángulo **es un** polígono,
- Un triángulo **es un** polígono
- Un triángulo rectángulo **es un** triángulo
- Un triángulo isósceles **es un** triángulo



- Superclase Poligono con las Subclases: Triángulo y Rectángulo.
- SuperClase Triangulo con las subclases: Triangulolsosceles y TrianguloRectangulo.

```

class Poligono {
    protected String color;
    public Poligono(String color ) {
        this.color=color;
    }
    public String getColor() {
        return color;
    }
}
  
```

```
class Rectangulo extends Poligono {
    private double largo;
    private double ancho;
    public Rectangulo(String color, double largo, double ancho) {
        super(color);
        this.largo=largo;
        this.ancho=ancho;
    }
    public double obtenerAreaRectangulo(){
        return largo*ancho;
    }
}

class Triangulo extends Poligono {
    private double base;
    private double altura;
    public Triangulo(String color, double base, double altura) {
        super(color);
        this.base=base;
        this.altura=altura;
    }

    public double obtenerAreaTriangulo(){
        return base*altura/2;
    }
}

public class TrianguloIsosceles extends Triangulo {
    public TrianguloIsosceles(String color, double base,
    double altura){
        super(color,base,altura);
    }
    public double areaTrianguloIsosceles(){
        double calculoArea= super.obtenerAreaTriangulo();
        return calculoArea;
    }
}

public class TrianguloRectangulo extends Triangulo {
    public TrianguloRectangulo(String color, double base
    double altura){
        super(color,base,altura);
    }
}
```

```

public class Principal {
    /**
     * metodo Principal, entrada de la aplicación en java
     * @param args
     */
    public static void main(String[] args) {
        double superficie;
        //definición objeto rectangulo
        Rectangulo rectangulo=new Rectangulo("Azul",5,4);
        superficie=rectangulo.obtenerAreaRectangulo();
        System.out.println("Este rectángulo es de color
        "+rectangulo.getColor()+" de base = 5 y altura = 4 ");
        System.out.println("El área es: "+superficie);
        //definición objeto trianguloIsosceles
        TrianguloIsosceles trianguloIsosceles=new
        TrianguloIsosceles("Verde",6,8);
        superficie=trianguloIsosceles.areaTrianguloIsosceles();
        System.out.println("Este triangulo isósceles es de color
        "+trianguloIsosceles.getColor()+" de base = 6 y altura = 8 ");
        System.out.println("El área es: "+superficie);
        //definición objeto trianguloRectangulo
        TrianguloRectangulo trianguloRectangulo=new
        TrianguloRectangulo("Rojo ",7,5);
        superficie=trianguloRectangulo.obtenerAreaTriangulo();
        System.out.println("Este triangulo Rectangulo es de color
        "+trianguloRectangulo.getColor()+" de base =
        7 y altura = 5 ");
        System.out.println("El área es: "+superficie);
    }
}

```

## 10.6 LECTURAS RECOMENDADAS

- » Herencia por extensión.
- » Herencia por restricción.
- » Herencia múltiple.



## 10.7 PREGUNTAS DE REVISIÓN DE CONCEPTOS

- » Objetivos de la herencia.
- » Como identificar que se debe aplicar herencia para resolver un problema
- » Tipos de herencia.
- » Tipos de herencia en Java.
- » A que corresponde la clase Object en Java.

## 10.8 EJERCICIOS

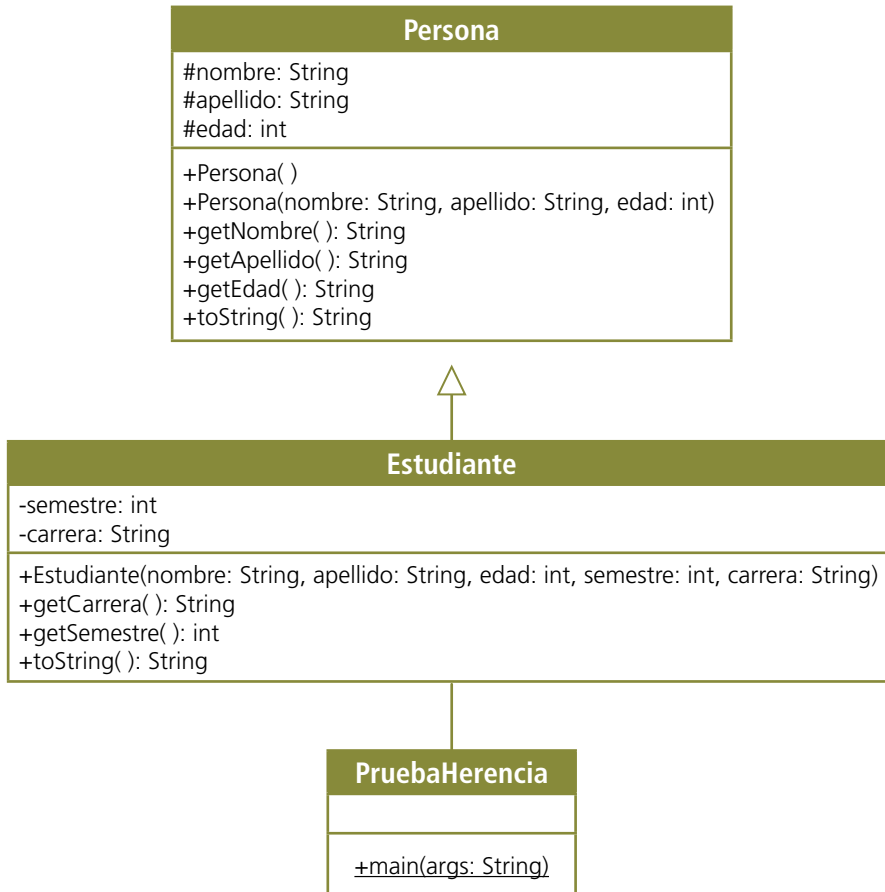
1. Crear el diagrama de clase para los siguientes requerimientos:

- Implementar la clase base denominada Animal, con sus atributos y métodos particulares, por ejemplo, nombre.
- Crear clases heredadas de Animal, como por ejemplo: Mamíferos, Reptiles, entre otros.
- Crear subclases de las creadas anteriormente, como Perro, León, Delfín.

2. Crear el diagrama de clase para el siguiente requerimiento:

Las impresoras láser, de burbuja, 3D y de matriz de punto son subclases de la superclase Impresora. Los atributos generales de una impresora son la marca, la referencia, la velocidad, el precio y la resolución, mientras que sus operaciones son imprimir y forma de alimentar el papel. La clase Impresora es una generalización de las clases: ImpresoraLaser, ImpresoraBurbuja, Impresora3D e ImpresoraMatrizPunto.

3. Codificar en Java que a partir del siguiente diagrama de clases:



4. Para cada uno de los siguientes enunciados construir el diagrama de clases en UML y codificar en java el diagrama de clases diseñado:
- En la universidad El Templo del Saber existen estudiantes, docentes, directivos, personal de mantenimiento, todos estos tienen, diferentes propiedades, comunes como la identificación, nombre, apellidos, género, edad. Pero existen algunas características propias para el caso de los estudiantes, el semestre, entre otros. Y para los docentes, directivos y demás personal el sueldo. Estas son las propiedades que se tendrán en cuenta para modelar el sistema.

- b. El banco El Ahorrador Feliz, requiere la construcción de un sistema que le permita controlar las cuentas de sus clientes y para ello las clasifica en dos: Cuentas de ahorros y Cuentas corriente. Todas las cuentas del Banco tienen los siguientes datos:
- Número de cuenta (dato numérico).
  - Nombre del cliente (cadena).
  - Saldo (dato numérico con fracción decimal).

Con cada cuenta se pueden realizar las siguientes operaciones:

- Consultar datos: a través de sus propiedades.
- Depositar: incrementa el saldo con la cantidad de dinero que se deposita.
- Retirar: antes de hacer el retiro, se debe verificar la suficiencia de saldo y en caso de aprobarlo, se disminuye el saldo.

Las cuentas de ahorros presentan las siguientes características:

- Porcentaje de interés mensual.
- Método para depositar los intereses el primer día de cada mes.
- Solamente se puede retirar un monto menor o igual al saldo.

Las cuentas corrientes presentan las siguientes características:

- Cobro del 4x1000, por mil en cada transacción
- Se puede sobregirar hasta en \$300.000, cuando el saldo no es suficiente para el retiro.

## 10.9 REFERENCIAS BIBLIOGRÁFICAS

Aguilar, L. J. (2004). *Fundamentos de programación algoritmos y estructura de datos*. Tercera Edición. Ciudad: México: McGraw Hill.

Barnes D., Kölling M. (2007), *Programación orientada a objetos con Java*. Ciudad Madrid, Prentice Hall.

Bertrand M. (1999). *Construcción de software orientado a objetos* (2ª edición) Ciudad: Madrid, Prentice Hall.

Booch, G. (1996). *Análisis y diseño orientado a objetos con aplicaciones*. Ciudad: México Addison Wesley.

Deitel y Deitel. (2012). *Cómo programar en Java*. Ciudad: México Editorial Pearson (Prentice Hall).

Fowler, M., y Scott, K. UML. (1999). *Gota a gota*. Ciudad: México Editorial Pearson.

Graig, L. (2003). UML y Patrones: *Una introducción al análisis y diseño orientado a objetos y al proceso unificado* (2ª edición) Ciudad: México Prentice Hall.

Rumbaugh, J., Jacobson, I., y; Booch, G.: (2005). *The Unified Modeling Language User Guide*, San Francisco: Addison-Wesley, Reading, Mass. ISBN-13: 078-5342267976 ISBN-10: 0321267974

Villalobos, J., y Casallas, R. (2006). *Fundamentos de programación, aprendizaje activo basado en casos*. Ciudad: México Editorial Pearson.





# CAPÍTULO 11

## **POLIMORFISMO**



## 11.1 TEMÁTICA A DESARROLLAR

- » Polimorfismo.
- » Polimorfismo de subtipado.
- » Sobrecarga de métodos.
- » Sobreescribir métodos.

## 11.2 INTRODUCCIÓN

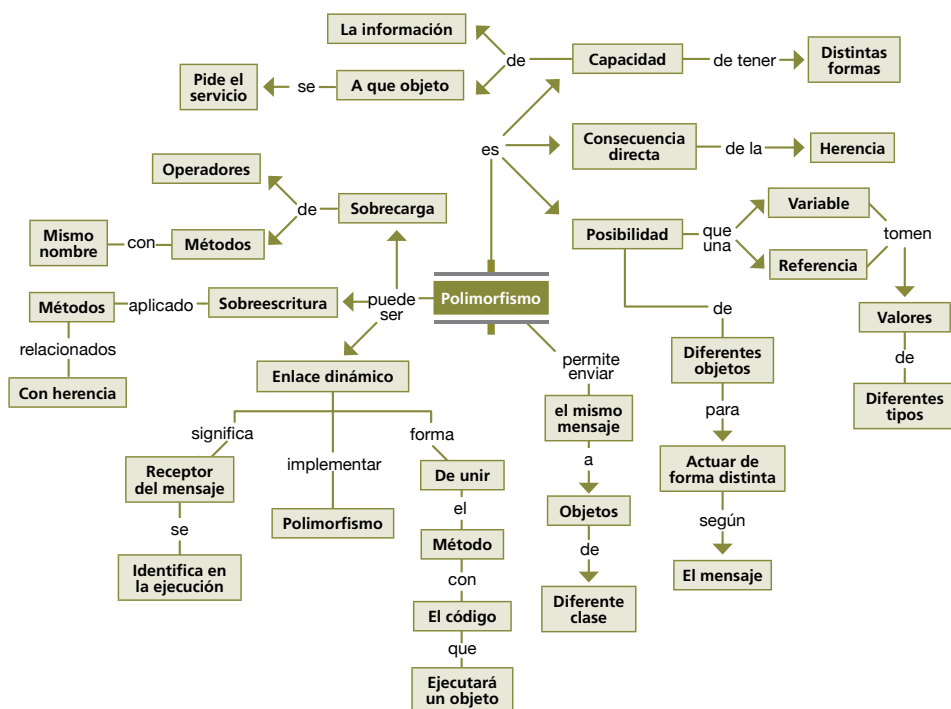
El término polimorfismo hace referencia a que una característica de una clase puede tomar varias formas; y en Programación Orientada a Objetos es una propiedad que hace referencia que más de un objeto puede crearse usando la misma clase base, por ejemplo, se tiene un teléfono base, con la capacidad de timbrar y tener un auricular; para luego obtener un teléfono digital, inalámbrico, con botones para marcar; y posteriormente un teléfono celular, entonces polimorfismo es la capacidad del código de un programa para ser utilizado con diferentes tipos de datos u objetos.

El polimorfismo se puede establecer mediante la sobrecarga que se refiere al uso del mismo identificador u operador en distintos contextos y con distintos significados. Java permite la sobrecarga de métodos que es una de las formas en las cuales se implementa el polimorfismo, esto quiere que en una misma clase puedan tener métodos con el mismo nombre.

La herencia y el polimorfismo están estrechamente ligados. La herencia implica polimorfismo el cual genera el polimorfismo de subtipado que es la capacidad para redefinir un método en clases que se hereda de una clase base se llama especialización. Por lo tanto, se puede llamar un método de objeto sin tener que conocer su tipo intrínseco: esto es polimorfismo de subtipado, por ejemplo, en un juego de ajedrez con los objetos rey, reina, alfil, caballo, torre y peón, cada uno heredando el objeto pieza. El método movimiento podría ser uno solo sin tener que verse conectado con cada tipo de pieza del ajedrez en particular. La figura 26 detalla el concepto de Polimorfismo.



Figura 26. Concepto de Polimorfismo.



### 11.3 POLIMORFISMO

Es la propiedad que indica que un elemento puede tomar diferentes formas. Para la programación orientada a objetos, el elemento puede ser un atributo, un método o un operador.

Por ejemplo, en Java un caso muy común es el del operador más '+', el cual se puede utilizar para sumar números y para concatenar cadenas, ya que el operador está sobrecargado de tal manera que cuando se utiliza Java reconoce el tipo de operación suma o concatenación respectivamente.

Para el caso de los métodos, una clase de polimorfismo que se denomina sobrecarga, específicamente sobrecarga de métodos, se define como el uso de varios métodos con el mismo nombre y cuyo contenido es diferente.

## 11.4 SOBRECARGA DE MÉTODOS

La sobrecarga consiste en implementar dos o más métodos que compartan el mismo nombre dentro de la misma clase, pero con parámetros diferentes. Los métodos se diferencian por su firma.

Java permite la sobrecarga de métodos e identifica entre métodos con diferentes firmas, evitando la necesidad de crear métodos diferentes que en esencia hacen lo mismo. También hace posible que los métodos se comporten de manera distinta basados en los argumentos que reciben.

### Ejemplos

1. Sobrecarga del método actualizarDato.

```
public class Actualizacion {  
    void actualizarDato(int x) {  
  
    }  
  
    void actualizarDato (String x) {  
  
    }  
  
    void actualizarDato (long x) {  
  
    }  
  
    void actualizarDato (float x) {  
  
    }  
  
    void actualizarDato (double x) {  
  
    }  
}
```

Se puede observar que el tipo de parámetro, es diferente en cada método implementado, como por ejemplo: int, long, String, float, double.

---

2. Sobrecarga del método obtenerSuma el cual cambia el número y tipo de parámetros:

```
public class Suma{
    /** Métodos sobrecargados */

    double obtenerSuma (String x, String y) {
        ...
    }

    double obtenerSuma(double x, double y) {
        ...
    }

    double obtenerSuma (double x, double y, double z) {
        ...
    }
}
```

En la clase suma, se definen tres métodos con el mismo nombre obtenerSuma, pero con listas de argumentos diferente. La diferencia está en el número y el tipo de argumentos en cada método.

---

## Programas de ejemplo

Sobrecarga de métodos para calcular producto, el cual cambia el número y tipo de parámetros.

```
/**
 * Clase que modela la sobrecarga de métodos por el número
 * de parametros de estos o el tipo de dato que recibe.
 * @version 1.0
 */
```

```

public class SobreCargaProducto {
    public int obtenerProducto(int x, int y){
        return x*y;
    }
    public int obtenerProducto(int x, int y, int z){
        return x*y*z;
    }
    public int obtenerProducto(String str1,String str2){
        int x=convertirEntero(str1);
        int y=convertirEntero(str2);
        return x*y;
    }
    private int convertirEntero(String numero){
        return Integer.parseInt(numero);
    }
}

/**
 * clase que modela los datos para la sobrecarga de los métodos
 * @version 1.0
 */
public class MainSobrecarga {
    public static void main (String []args){
        int a = 2;
        int b = 4;
        int c = 5;
        String str1 = "8";
        String str2 = "7";
        SobreCargaProducto sp= new SobreCargaProducto();
        System.out.println("Método con dos parámetros: "+
            a+" x "+b+" = "+sp.obtenerProducto(a,b));
        System.out.println("Método con tres parámetros: "+
            a+" x "+b+" x "+c+" = "+sp.obtenerProducto(a,b,c));
        System.out.println("Método con parámetros de tipo String: "+
            str1+" x "+str2+" = "+sp.obtenerProducto(str1,str2));
    }
}

```

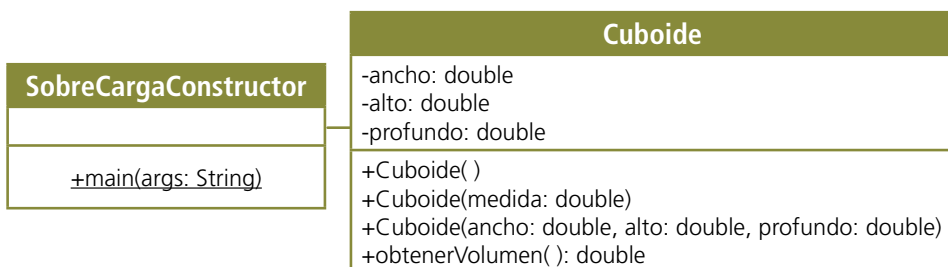
## 11.5 SOBRECARGA DE CONSTRUCTORES

Una clase puede tener más de un constructor y los constructores se diferencian por la cantidad, el tipo y el orden de los parámetros.

### Ejemplo

Sobrecarga del constructor de la clase Cuboide, la cual está conformada por tres constructores, con un número de diferentes parámetros:

### Diagrama de clases



La implementación del diagrama de clases en lenguaje Java es la siguiente:

```
public class SobreCargaConstructor {
    public static void main(String[] args) {
        //se definen tres instancias de la clase Cubo
        Cuboide cubo1 = new Cuboide();
        Cuboide cubo2 = new Cuboide(7);
        Cuboide cubo3 = new Cuboide(10, 20, 15);
        //se define el atributo volumen de tipo double
        double volumen = 0.0;
        volumen = cubo1.volumen();
        System.out.println("El volumen de cubo1 es " + volumen);
        volumen = cubo2.volumen();
        System.out.println("El volumen de cubo2 es " + volumen);
        volumen = cubo3.volumen();
        System.out.println("El volumen de cubo3 es " + volumen);
    }
}
```

```

/**
 * clase que modela la sobrecarga de constructores para calcular el
 * volumen del cuboide
 */
public class Cuboide {
    private double ancho;
    private double alto;
    private double profundo;

    /**
     * Constructor que inicializa valor de los atributos por defecto
     * las dimensiones del cubo no están definidas
     */
    public Cuboide() {
        this.ancho = 0;
        this.alto = 0;
        this.profundo = 0;
    }

    /**
     * Constructor con un parámetro
     * @param medida
     */
    public Cuboide(double medida) {
        this.ancho = medida;
        this.alto = medida;
        this.profundo = medida;
    }
}

```

```

/**
 * Constructor con tres parametro
 * @param ancho
 * @param alto
 * @param profundo
 */
public Cuboide(double ancho, double alto, double profundo) {
    this.ancho = ancho;
    this.alto = alto;
    this.profundo = profundo;
}

```

```
/**
 * metodo que retorna el volumen del cuboide
 * @return volumen
 */
public double obtenerVolumen() {
    return ancho * alto * profundo;
}
```

Al sobrecargar los constructores, de la clase Cuboide se crean tres objetos (cubo1, cubo2, cubo3) con diferentes inicializaciones:

- » Cuboide cubo1 = new Cuboide();
- » Cuboide cubo2 = new Cuboide(7);
- » Cuboide cubo3 = new Cuboide(10, 20, 15);

Que al ejecutar el programa imprimen:

El volumen de cubo1 es: 0.0.

El volumen de cubo2 es: 343.0.

El volumen de cubo es: 3000.0.

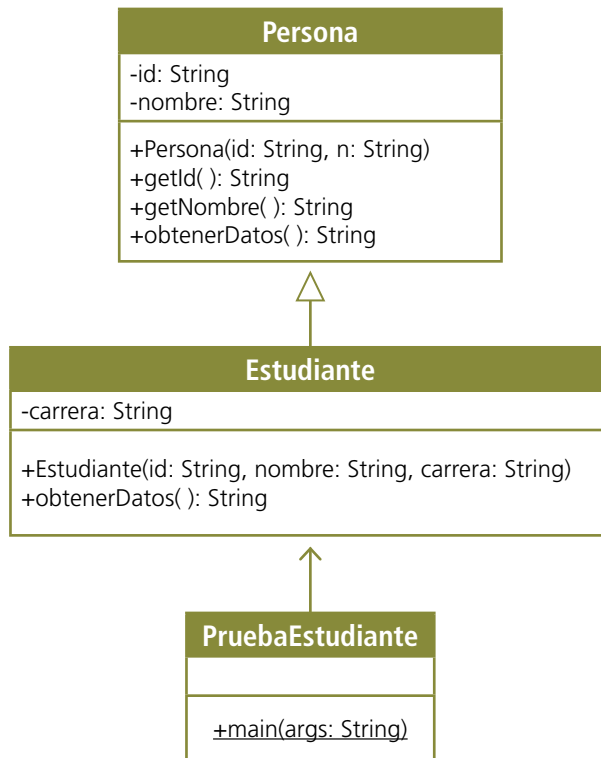
## 11.6 SOBRESCRITURA DE MÉTODOS

Cuando una subclase implementa un método que fue definido en la superclase, con el mismo nombre, la misma cantidad y tipos de parámetros. En este caso se presenta la sobrescritura de métodos. Consiste en la implementación de métodos previamente definidos en la superclase, con igual firma.

Al heredar los métodos de una clase se tiene la opción de sobrescribirlos para darles una funcionalidad distinta a la que tenían. Es decir se toma el método de la clase padre en la clase hija y se realizan ajustes al código que se encuentra dentro de las llaves que definen el cuerpo del método según sean los requerimientos y a esto se le llama sobrescritura.

## Ejemplos

1. Ejercicio conformado por las clases, Persona, Estudiante, que implementa el método obtenerDatos, método que es heredado de la clase Persona y sobre escrito en la clase Estudiante.



```

public class Persona {
    private String id;
    private String nombre;
    public Persona(String id, String nombre){
        this.id = id;
        this.nombre = nombre;
    }
    public String getId(){
        return id;
    }
}
  
```



```

        public String getNombre(){
            return nombre;
        }
        public String obtenerDatos(){
            return id + " " + nombre;
        }
    }

    public class Estudiante extends Persona {
        private String carrera;
        public Estudiante(String id, String nombre, String carrera){
            super(id,nombre);
            this.carrera = carrera;
        }
        public String obtenerDatos(){
            return getId() + getNombre() + carrera;
        }
    }

    public class PruebaEstudiante {
        public static void main(String[] args) {
            Persona unEstudiante = new Estudiante("10020","Carlos
            Rodriguez","Ingenieria de Sistemas");
            System.out.println( unEstudiante.obtenerDatos());
        }
    }

```

2. Cada clase de java hereda de la clase Object, y la clase Object tiene varios métodos dentro de los que está el método toString().

```

public class Persona{
    private String nombre;
    public Persona(String n){
        this.nombre = n;
    }
    //aqui se sobreescribe el metodo
    @Override
    public String toString(){
        return nombre;
    }
}

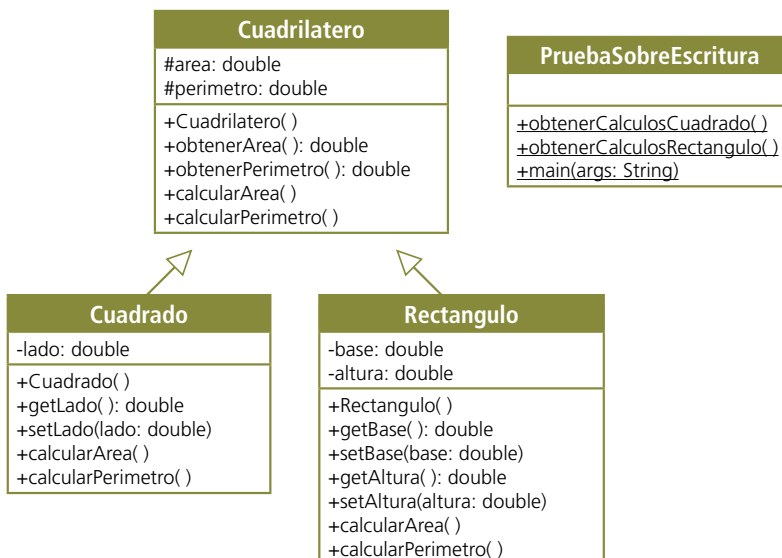
```

La clase `Persona` sobrescribe el método `toString` indicando que devuelve el nombre de la persona:

```
public class Prueba {
    public static void main(String[] args) {
        Persona p = new Persona("Mercedes");
        System.out.println("Nombre a partir del método
        "+p.toString());
        System.out.println("Nombre a partir del objeto "+p);
    }
}
```

Se crea objeto `p` de la clase `Persona`, al imprimir el objeto, por medio de la sentencia `System.out.println("Nombre a partir del objeto "+p);` se envía, como parámetro un objeto que llama al método `toString` e imprime "Mercedes", el proceso es el mismo cuando se invoca con el objeto `p.toString()`.

3. A partir de la relación de herencia de la clase base `Cuadrilatero` que definen los métodos `calcularArea` y `calcularPerimetro`; las subclases `Cuadrado` y `Rectángulo` que redefinen estos métodos ya que el cálculo del área y el perímetro de cada de estas dos figuras geométricas es diferente.



```
public class Cuadrilatero {
    protected double area;
    protected double perimetro;

    public Cuadrilatero() {
        this.area=0;
        this.perimetro=0;
    }
    public double obtenerArea() {
        return area;
    }
    public double obtenerPerimetro() {
        return perimetro;
    }
    public void calcularArea(){
    public void calcularPerimetro(){

}

public class Cuadrado extends Cuadrilatero{
    private double lado;

    public Cuadrado() {
        this.lado = 0;
    }
    public double getLado() {
        return lado;
    }
    public void setLado(double lado) {
        this.lado = lado;
    }

    public void calcularArea() {
        this.area = lado * lado;
    }
    public void calcularPerimetro() {
        this.perimetro = 4 * lado;
    }
}
```

```

public class Rectangulo extends Cuadrilatero{
    private double base;
    private double altura;

    public Rectangulo() {
        this.base = 0;
        this.altura = 0;
    }
    public double getBase() {
        return base;
    }

    public void setBase(double base) {
        this.base = base;
    }
    public double getAltura() {
        return altura;
    }
    public void setAltura(double altura) {
        this.altura = altura;
    }
    public void calcularArea() {
        this.area = base * altura;
    }

    public void calcularPerimetro() {
        this.perimetro = (2 * base)+(2 * altura);
    }
}

```

```

public class PruebaSobreEscritura {
    public void obtenerCalculosCuadrado(){
        System.out.println("\nEl Cuadrado\n");
        Cuadrado unCuadrado = new Cuadrado();
        unCuadrado.setLado(3);
        unCuadrado.calcularPerimetro();
        unCuadrado.calcularArea();
        unCuadrado.calcularPerimetro();
        unCuadrado.calcularArea();
    }
}

```

```
System.out.println("Lado "+unCuadrado.getLado());
System.out.println("Perímetro del cuadrado:
"+unCuadrado.obtenerPerimetro());
System.out.println("Área del cuadrado:
"+unCuadrado.obtenerArea());
}
```

```
public void obtenerCalculosRectangulo(){
    System.out.println("\nEl Rectangulo\n");
    Rectangulo unRectangulo = new Rectangulo();
    unRectangulo.setBase(3);
    unRectangulo.setAltura(4);
    unRectangulo.calcularPerimetro();
    unRectangulo.calcularArea();
    System.out.println("Base: "+unRectangulo.getBase());
    System.out.println("Altura: "+unRectangulo.getAltura());
    System.out.println("Perímetro del rectángulo:
"+unRectangulo.obtenerPerimetro());
    System.out.println("Área del rectángulo:
"+unRectangulo.obtenerArea());
}
```

```
public static void main(String[] args) {
    //se define el objeto pse de la clase
    PruebaSobreEscritura
    PruebaSobreEscritura pse = new PruebaSobreEscritura();
    //el objeto pse invoca los métodos
    pse.obtenerCalculosCuadrado();
    pse.obtenerCalculosRectangulo();
}
}
```

## 11.7 ENLACE DINÁMICO

Un mensaje con una determinada firma puede enviarse a cualquier objeto que lo incluya en su interfaz. Pero como diferentes objetos pueden implementar un mismo método de manera distinta, el resultado del mensaje no depende en sí del mismo mensaje, sino de su receptor.

El enlace dinámico significa que el receptor del mensaje se determina durante la ejecución del programa.

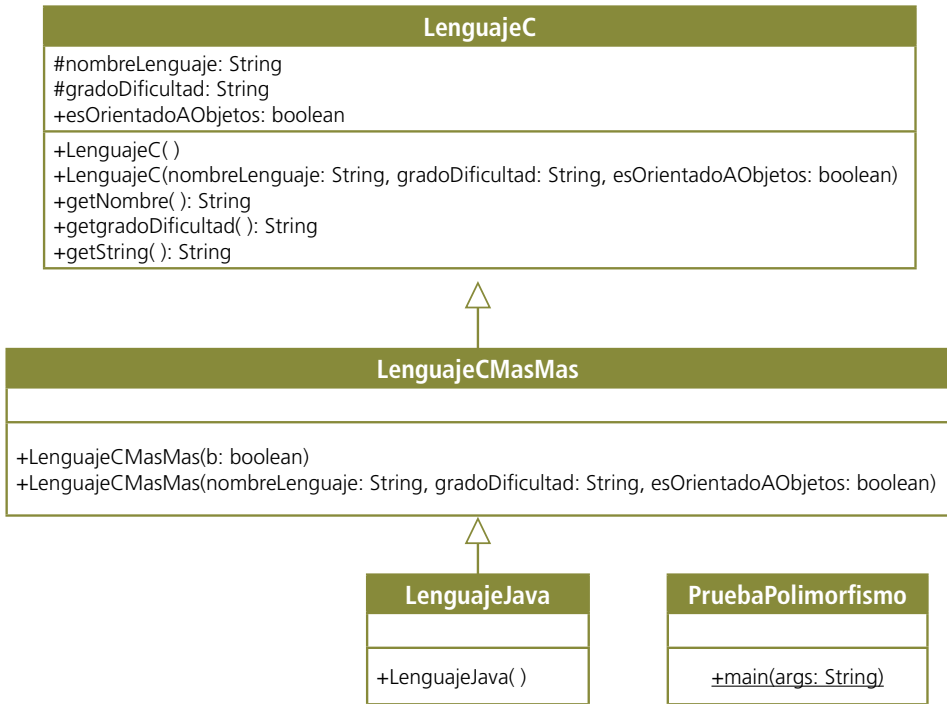
Polimorfismo y enlace dinámico no son exactamente lo mismo: el enlace dinámico es el mecanismo que hace posible implementar el polimorfismo.

Los métodos polimórficos se utilizan cuando adoptan diferentes formas de implementación según el tipo de objetos, cumpliendo el mismo objetivo. Por ejemplo, lanzar puede ser implementado para una objeto balón de: fútbol, baloncesto, voleibol; mover una cualquier ficha de un tablero de ajedrez.

### Ejemplos

El lenguaje C es uno de los lenguajes de programación más populares que existen hoy en día. El lenguaje C++ Es un lenguaje derivado del C, pero mucho más complejo, ya que incluye la programación orientada al objeto; y el lenguaje de Programación Java, es un lenguaje orientado a objetos, con una sintaxis similar a C++, pero algunos conceptos de diseño diferentes.

- » Lenguaje C, es un Lenguaje de Programación, que tiene características generales como un nombre, grado de dificultad y no es orientado a objetos.
- » C++ es un Lenguaje de Programación Orientada a Objetos.
- » C++ es una derivación de Lenguaje C.
- » Java es un Lenguaje de Programación Orientada a Objetos.



```

public class LenguajeC {
    protected String nombreLenguaje;
    protected String gradoDificultad;
    public boolean esOrientadoAObjetos;

    //sobrecarga de constructores
    /**
     * constructor vacio
     */
    public LenguajeC(){
        this.nombreLenguaje="C";
        this.gradoDificultad = "BAJO";
        this.esOrientadoAObjetos=false;
    }

    public LenguajeC(String nombreLenguaje, String gradoDificultad,
        boolean esOrientadoAObjetos){
        this.nombreLenguaje=nombreLenguaje;
    }
}

```

```

        this.gradoDificultad = gradoDificultad;
        this.esOrientadoAObjetos = esOrientadoAObjetos;
    }
    public String getNombre(){
        return nombreLenguaje;
    }
    public String getgradoDificultad(){
        return gradoDificultad;
    }

    public String getString(){
        String s = nombreLenguaje;
        if(!esOrientadoAObjetos){
            s+= " No";
        }
        s += " es un Lenguaje de Programación Orientado a Objetos ";
        s+="\ny tiene un grado de dificultad de "+ gradoDificultad;
        return s;
    }
}

public class LenguajeCMasMas extends LenguajeC{
    //sobrecarga de constructores
    public LenguajeCMasMas(boolean b){
        //se inicializan atributos de la superclase LenguajeC
        super("C++","ALTO",true);
    }

    public LenguajeCMasMas(String nombreLenguaje, String
        gradoDificultad, boolean
        esOrientadoAObjetos){
        super(nombreLenguaje, gradoDificultad,
        esOrientadoAObjetos);
    }
}

public class LenguajeJava extends LenguajeCMasMas {
    public LenguajeJava(){
        //true, porque se indica que es Orientado a Objetos
        super("Java","MEDIO",true);
    }
}

```



```
public class PruebaPolimorfismo {  
    public static void main(String args[]){  
        LenguajeC unLenguaje;  
        unLenguaje = new LenguajeC();  
        System.out.println("\n"+unLenguaje.getString());  
        unLenguaje = new LenguajeCMasMas(true);  
        System.out.println("\n"+unLenguaje.getString());  
        unLenguaje = new LenguajeJava();  
        System.out.println("\n"+unLenguaje.getString());  
    }  
}
```

Aunque los objetos se accedan con referencias de superclase, el método que se ejecuta es el de la clase real del objeto.

## 11.8 LECTURAS RECOMENDADAS

- » Polimorfismo.
- » Sobrecarga.
- » Enlace dinámico.

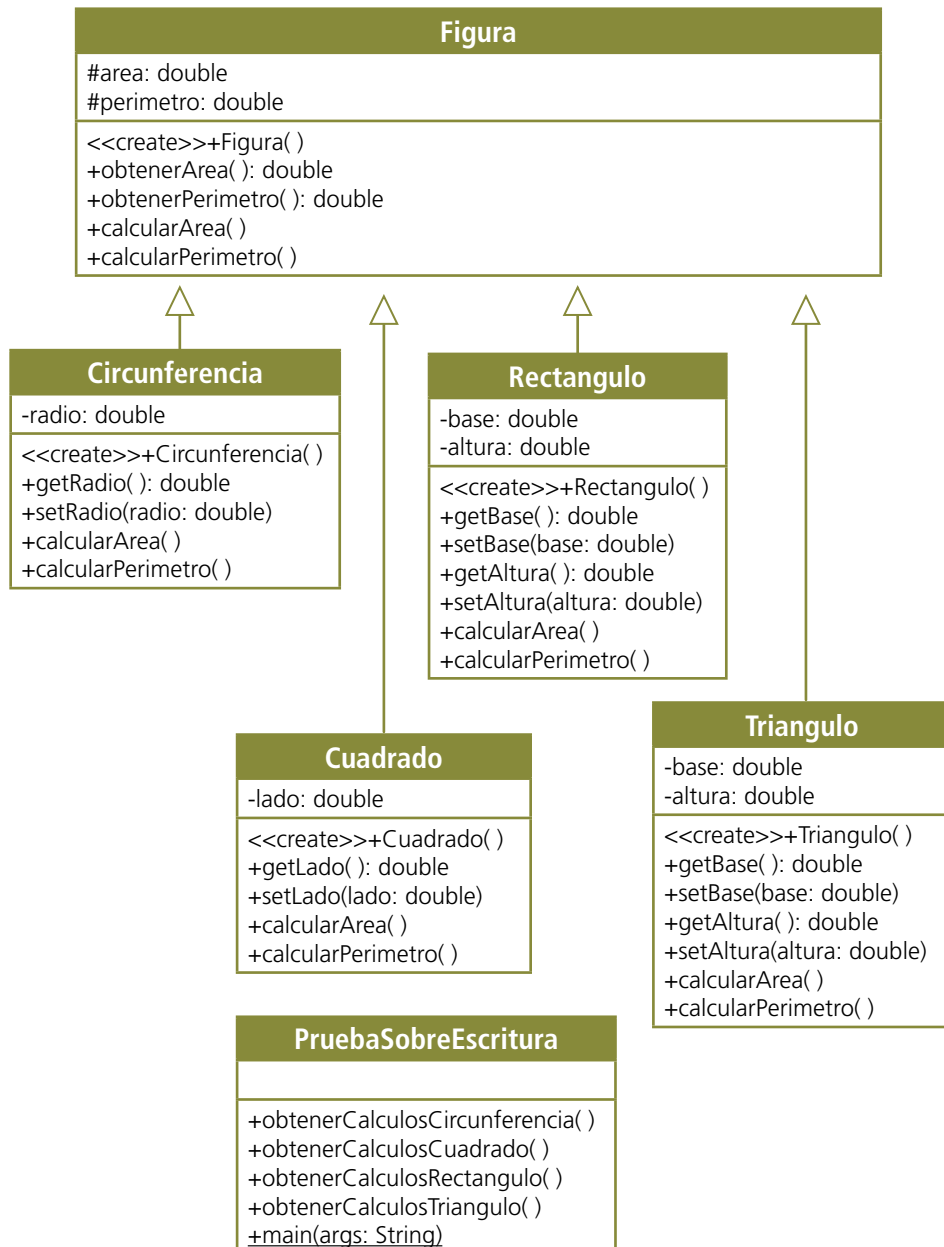
## 11.9 PREGUNTAS DE REVISIÓN DE CONCEPTOS

- » Objetivos del polimorfismo.
- » Como identificar que se debe aplicar polimorfismo para resolver un problema.

## 11.10 EJERCICIOS

1. Construir la aplicación en Java a partir de la relación de herencia de la superclase **Figura** que define los métodos **calcularArea** y **calcularPerimetro**, las subclases **Circunferencia**, **Cuadrado**, **Triángulo** y **Rectangulo**; redefinen estos métodos ya que el cálculo del área y el perímetro de cada de estas figuras geométricas es diferente.

## Diagrama de clases



2. Construir una aplicación en Java que aplique los conceptos de sobrecarga de constructores para inicializar las dimensiones de un rectángulo, teniendo en cuenta los siguientes indicaciones:
    - a. El primer constructor de Rectangulo inicializa un nuevo Rectangulo con valores por defecto (por ejemplo altura = 0.0, ancho = 0.0).
    - b. El segundo constructor inicializa el nuevo Rectangulo con la altura y ancho a partir de datos ingresados por teclado.
- 

3. Se tiene la información del estudiante conformado por el código, nombre, apellido y edad. Se requiere construir una aplicación en la cual se implementa la sobrecarga de los métodos constructores.
- 

4. Consultar el método toString( ) de Java y aplicarlo para el siguiente ejercicio:

La empresa El Viajero ha solicitado la creación de un sistema donde quiere tener representados a todos sus productos en clases de Java, también solicitó que fuera realizado de tal manera (aplicando la abstracción) que si es necesario agregar un nuevo producto no fuera un impacto muy grande para el sistema. La empresa se especializa en todo tipo de transportes y maquinarias, actualmente tiene en sus vitrinas los siguientes productos:

- » Buses.
- » Camiones.
- » Helicópteros.
- » Aviones.

## 11.11 REFERENCIAS BIBLIOGRÁFICAS

Bertrand M. (1999). *Construcción de software orientado a objetos* (2ª edición) Ciudad: Madrid, Prentice Hall.

Booch, G. (1996). *Análisis y diseño orientado a objetos con aplicaciones*. Ciudad: México Addison Wesley.

Deitel y Deitel. (2012). *Cómo programar en Java*. Ciudad: México Editorial Pearson (Prentice Hall).

Fowler, M., y Scott, K. UML. (1999). *Gota a gota*. Ciudad: México Editorial Pearson.

Graig, L. (2003). *UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado* (2ª edición) Ciudad: México Prentice Hall.

Rumbaugh, J., Jacobson, I., y; Booch, G.: (2005). *The Unified Modeling Language User Guide*, San Francisco: Addison-Wesley, Reading, Mass. ISBN-13: 078-5342267976 ISBN-10: 0321267974

Villalobos, J., y Casallas, R. (2006). *Fundamentos de programación, aprendizaje activo basado en casos*. Ciudad: México Editorial Pearson.





# CAPÍTULO 12

**Interface y**

**CLASES ABSTRACTAS**



## 12.1 TEMÁTICA A DESARROLLAR

- » Interface.
- » Clases abstractas.

## 12.2 INTRODUCCIÓN

En este capítulo se abordan los temas de interfaz y clase abstracta, donde una interfaz es una plantilla para la construcción de clases la cual está conformada por la declaración de métodos sin implementar, que especifican un protocolo de comportamiento para una o varias clases. Lo anterior permite organizar la programación y a su vez obliga a que ciertas clases utilicen los mismos métodos (nombres y parámetros) y establecen relaciones entre clases que no estén relacionadas.

En Java una clase puede implementar una o varias interfaces, en ese caso, la clase debe proporcionar la declaración y definición de todos los métodos de cada una de las interfaces o bien declararse como clase abstract. Por otro lado, una interfaz puede emplearse también para declarar constantes que luego puedan ser utilizadas por otras clases.

Las clases abstractas son un tipo de clases que permiten crear métodos vacíos, es decir, no tienen cuerpo de declaración, no tienen las llaves { } ni código dentro de ellos y deben estar precedidos por la palabra clave **abstract**. Si una clase contiene uno o más métodos abstractos, esta clase debe ser abstracta.



## 12.3 INTERFACE

La declaración de una interfaz es similar a una clase, aunque emplea la palabra reservada **interface** en lugar de **class** y no incluye ni la declaración de variables de instancia ni la implementación del cuerpo de los métodos (solo las cabeceras). La sintaxis de declaración de una interfaz es la siguiente:

```
public interface IdentificadorInterfaz {
    // Cuerpo de la interfaz . . .
}
```



### Ejemplos

```
public interface TransporteAereo {
    public void despegar() ;
    public void aterrizar() ;
}
```

Cualquier clase que declare que implementa una determinada interface, debe comprometerse a implementar todos y cada uno de los métodos que esa interfaz define. Esa clase, además, pasará a pertenecer a un nuevo tipo de datos extra que es el tipo de la interface que implementa.

```
public class Viaje implements TransporteAereo {
    public void despegar( ) {
        System.out.println("despegar");
    }
    public void aterrizar(){
        System.out.println("aterrizar");
    }
}
```

En este ejemplo se define la interface ISaludo que tiene un atributo y dos métodos y la clase Herencia que implementa los métodos de la interface.

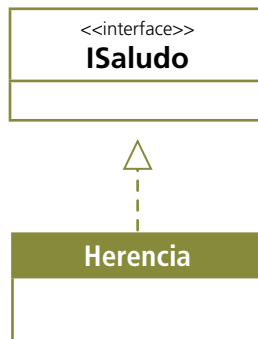
```

public interface ISaludo {
    public static final String nombre = "Hola soy una Interface";
    public abstract void saludar();
    public abstract String getNombre();
}

public class Herencia implements ISaludo{
    public Herencia (){
        saludar();
    }
    @Override
    public String getNombre(){
        return nombre;
    }
    @Override
    public void saludar(){
        System.out.println("Hola "+getNombre());
    }

    public static void main(String... args){
        Herencia hereda = new Herencia();
    }
}

```



Las interfaces actúan, por tanto, como tipos de clases. Se pueden declarar variables que pertenezcan al tipo de la interfaz, se pueden declarar métodos cuyos argumentos sean del tipo de la interface.

Una clase puede implementar tantas interfaces como desee, pudiendo, por tanto, pertenecer a tantos tipos de datos diferentes como interfaces implemente. En este sentido, las interfaces suplen la carencia de herencia múltiple de Java (y además corrigen o evitan todos los inconvenientes que del uso de esta se derivan).

Un interface es una lista de métodos donde solo están las cabeceras de métodos, sin implementación, que define un comportamiento específico para un conjunto de objetos. Algunas de sus características son:

- » La Interfaz es **public** o **default**. Tal como se observa a continuación:

```
public interface IdentificadorInterfaz {  
    // Cuerpo de la interfaz . . .  
}
```

- » Los atributos de la interface son **public static final**, así como se presenta el siguiente ejemplo:

```
public interface PuntosCardinales {  
    public static final int NORTE = 1;  
    public static final int SUR = 2;  
    public static final int ORIENTE = 3;  
    public static final int OCCIDENTE = 4;  
  
}
```

Una interfaz puede contener identificadores de variables constantes; estas variables constantes se heredan por cualquier clase que implemente la interface. El siguiente código muestra un ejemplo:

```
public interface IdentificadorInterfaz {  
    public static final int VALOR_MAXIMO = 10;  
}
```

- » Todos los métodos de una interface son métodos **public abstract**. Y las firmas de métodos sin implementación, como se observa en las siguientes líneas de código:

```
public interface Figura {  
  
    public abstract double calcularArea( );  
    public abstract double calcularPerimetro( );  
  
}
```

- » Permite simular herencia múltiple.

```
public interface IUno {
    public abstract void imprimirMensaje1();
    public abstract void imprimirMensaje2();
}

public interface IDos {
    public abstract void imprimirMensaje3();
    public abstract void imprimirMensaje4();
}

public class PruebaInterfaces implements IUno, IDos {

    @Override
    public void imprimirMensaje1() {
        System.out.println("Mensaje 1");
    }

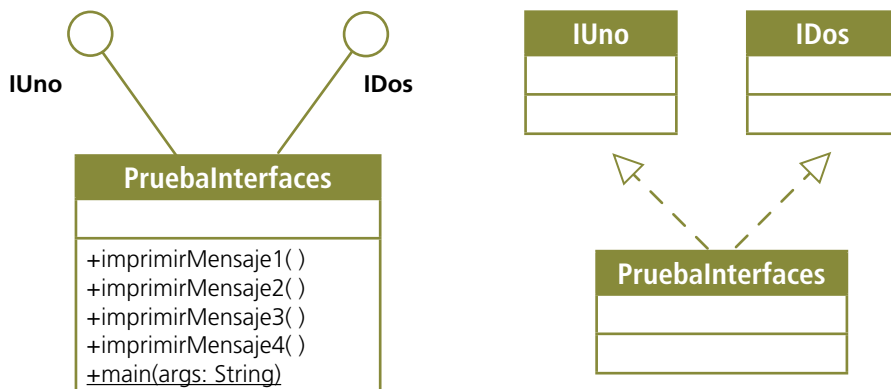
    @Override
    public void imprimirMensaje2() {
        System.out.println("Mensaje 2");
    }

    @Override
    public void imprimirMensaje3() {
        System.out.println("Mensaje 3");
    }

    @Override
    public void imprimirMensaje4() {
        System.out.println("Mensaje 4");
    }

    public static void main(String[] args) {
        PruebaInterfaces prueba = new PruebaInterfaces();
        prueba.imprimirMensaje1();
        prueba.imprimirMensaje2();
        prueba.imprimirMensaje3();
        prueba.imprimirMensaje4();
    }
}
```

## Diagrama de clases.



## El uso de las interfaces se puede aplicar en:

- » La revelación de la interface de la programación de un objeto (funcionalidad del objeto) sin presentar su implementación (encapsulado).
- » La implementación puede cambiar sin afectar el llamado de la interface, que no necesita la implementación en Java.
- » La implementación de métodos de similares comportamientos en clases sin relacionar.
- » El modelado de herencia múltiple, imponiendo conjuntos múltiples de comportamientos a la clase.

## 12.4 CLASES ABSTRACTAS

Las Clases abstractas se declaran con la palabra reservada **abstract**. Pueden definir métodos y variables, pero no se pueden usar directamente para crear objetos usando **new**. Algunas de las características son:

- » A partir de una clase abstracta se tiene la posibilidad de declarar clases, las cuales definen los métodos sin tener que implementarlos.
- » Las subclases de una clase abstracta debe sobrescribir los métodos abstractos definidos.
- » Una clase abstracta puede incluir métodos implementados y métodos abstractos.

Es necesario crear una clase que extienda la clase abstracta e implemente todos los métodos que en la clase abstracta hayan sido declarados abstract. Por ejemplo:

```
public abstract class ClaseAbstracta {
    public abstract void metodoUno();
    public String toString() {
        return "Clase Abstracta";
    }
}
```

El **metodoUno()**, es un método abstracto que incluye la implementación (no tiene código), sólo se define la firma del método (nombre y parámetros).

El método **toString()**, en cambio, no es un método abstracto y sí incluye implementación.

```
class SubClase extends ClaseAbstracta {
    public void metodoUno() {
        System.out.println("Metodo uno");
    }
}
```

La clase derivada SubClase implementa la abstracción, pues deriva de ClaseAbstracta e implementa metodoUno().

## Ejemplo

En la ClaseAbstracta, se define el método abstracto despedir que es implementado en la clase HerenciaAbstracta.

```
public abstract class ClaseAbstracta{
    public String nombre;
    public void saludar(){
        System.out.println("Hola "+nombre);
    }
    public abstract void despedir();
}
public class HerenciaAbstracta extends ClaseAbstracta{
    public HerenciaAbstracta(){
        obtenerNombre("Estudiante");
        saludar();
        despedir();
    }
}
```

```

    public void obtenerNombre(String nombre){
        super.nombre = nombre;
    }
    @Override
    public void despedir() {
        System.out.println("Hasta pronto "+super.nombre);
    }
    public static void main(String... args){
        HerenciaAbstracta ha = new HerenciaAbstracta();
    }
}

```

## 12.5 PREGUNTAS DE REVISIÓN DE CONCEPTOS

- » ¿Cuáles son las similitudes y las diferencias entre una clase abstracta y una interface?
- » ¿Cuándo se recomienda utilizar una interfase y una clase abstracta?
- » Cuando una clase implementa varias interfaces, ¿se puede considerar que es herencia múltiple?

## 12.6 EJERCICIOS

1. Construir una aplicación con el respectivo diagrama de clases en UML y la codificación Java de una aplicación que permita dar respuesta a los siguientes requerimientos:

Que permita calcular el área y el perímetro para las siguientes figuras geométricas: (para lo cual se recomienda implementar la Interfaz IFigura).

```

public interface IFigura {
    public double  calcularArea( );
    public double  calcularPerimetro( );
}

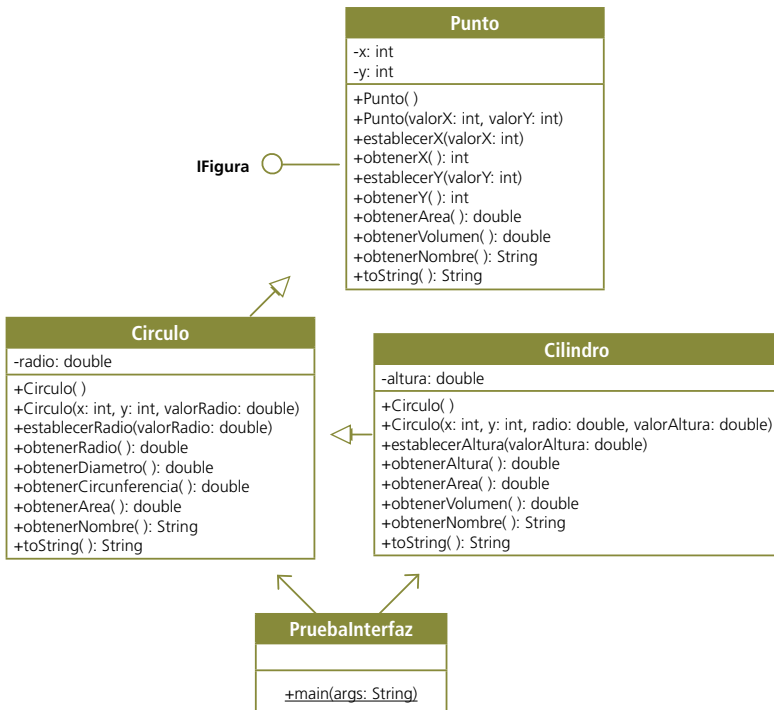
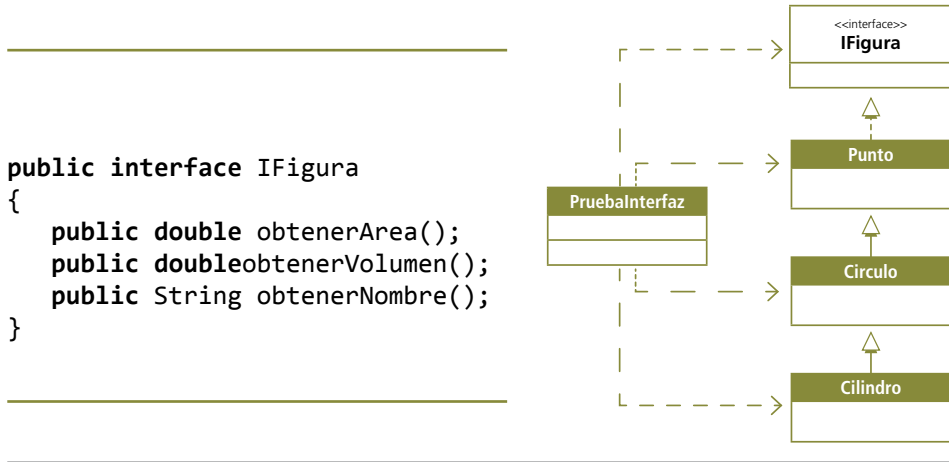
```

Las clases que implementa esta interfaz son:

- » La circunferencia.
- » El rectángulo.
- » El triángulo equilátero.

2. Construir una aplicación a partir del siguiente diagrama de clases en UML realizando la codificación Java de una aplicación que permita dar respuesta a los siguientes requerimientos:

Que permita calcular el área, el volumen y obtener el nombre de la figura para lo cual se recomienda implementar la Interfaz IFigura.





3. Construir una aplicación a partir del siguiente diagrama de clases en UML realizando la codificación Java de una aplicación que permita dar respuesta a los siguientes requerimientos:

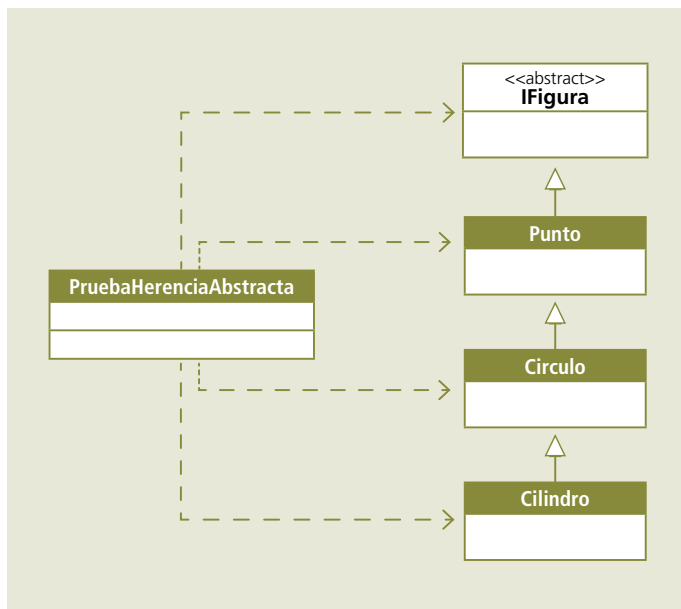
Que permita calcular el área, el volumen y obtener el nombre de la figura para lo cual se recomienda implementar la clase abstracta Figura

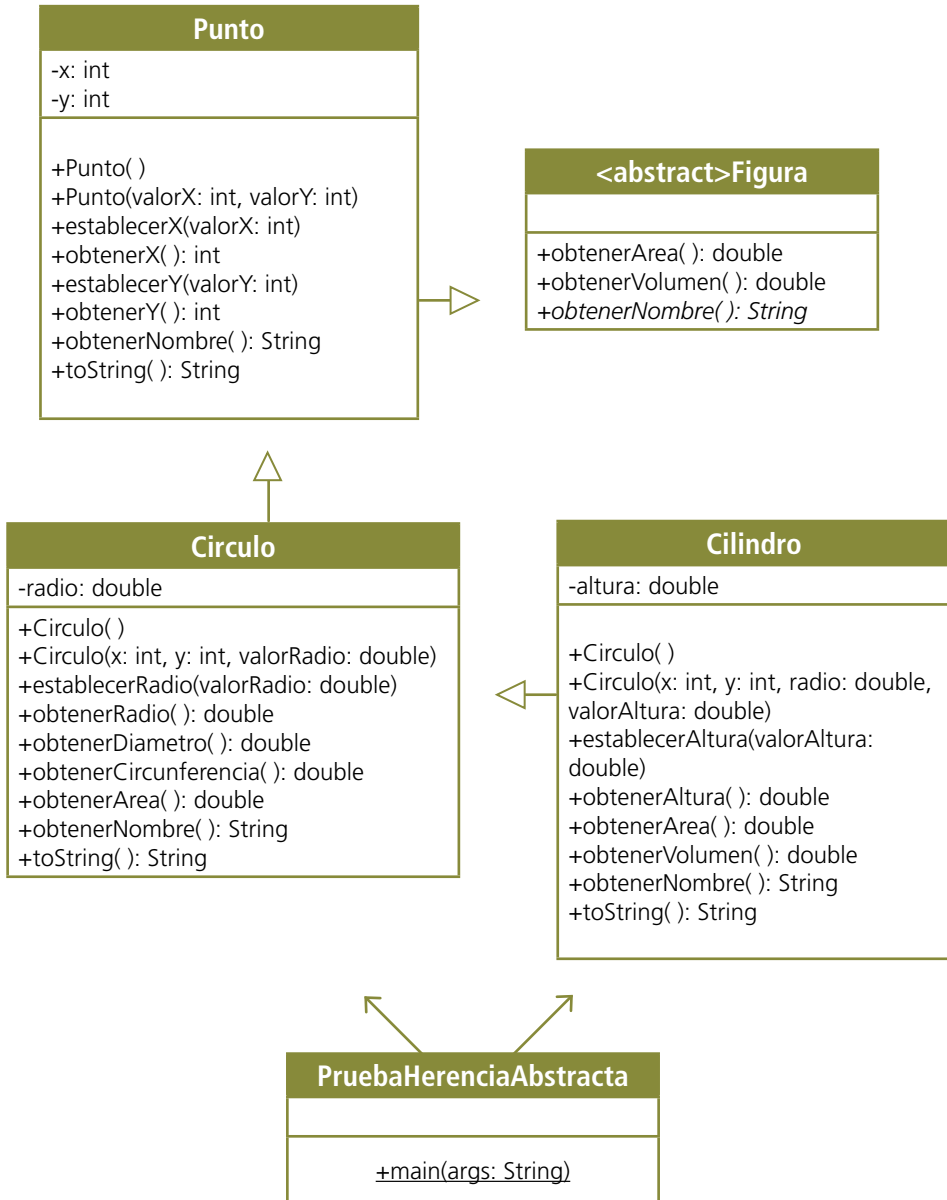
```
public abstract class Figura {

    // retorna el área de la figura; 0.0 de manera predeterminada
    public double obtenerArea() {
        return 0.0;
    }

    // retorna el volumen de la figura; 0.0 de forma predeterminada
    public double obtenerVolumen() {
        return 0.0;
    }

    // método abstracto, sobrescrito por las subclases
    public abstract String obtenerNombre();
}
```





## 12.7 REFERENCIAS BIBLIOGRÁFICAS

Booch, G. (1996). *Análisis y diseño orientado a objetos con aplicaciones*. Ciudad: México Addison Wesley.

Deitel y Deitel. (2012). *Cómo programar en Java*. Ciudad: México Editorial Pearson (Prentice Hall).

Fowler, M., y Scott, K. UML. (1999). *Gota a gota*. Ciudad: México Editorial Pearson.

Graig, L. (2003). *UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado* (2ª edición) Ciudad: México Prentice Hall.

Rumbaugh, J., Jacobson, I., y; Booch, G.: (2005). *The Unified Modeling Language User Guide*, San Francisco: Addison-Wesley, Reading, Mass. ISBN-13: 078-5342267976 ISBN-10: 0321267974

Villalobos, J., y Casallas, R. (2006). *Fundamentos de programación, aprendizaje activo basado en casos*. Ciudad: México Editorial Pearson.



# CAPÍTULO 13

**Modelado Orientado a Objetos y**

**APLICACIONES DE SOFTWARE**



## 13.1 TEMÁTICA A DESARROLLAR

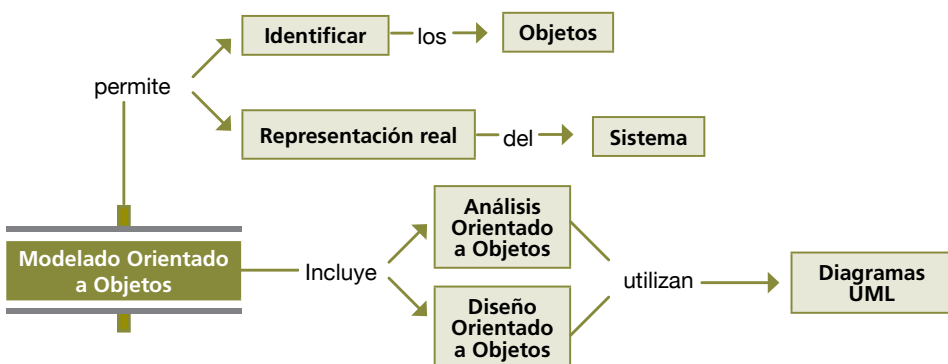
- » Análisis Orientado a Objetos.
- » Diseño Orientado a Objetos.
- » Diagramas UML.
- » Aplicaciones de Software.

## 13.2 INTRODUCCIÓN

El enfoque orientado a objetos brinda una serie de ventajas como la familiaridad con los conceptos, facilitando de esta forma la comunicación con el usuario o cliente, para representar sistemas sin determinar las características de implementación en el computador, permitiendo pasar directamente el dominio del problema al modelo del sistema, en un proceso de modelación más natural similar como el hombre lo visualiza.

Este capítulo esboza los parámetros generales que se deben tener en cuenta para el buen desarrollo de una aplicación de software con enfoque orientado a objetos, empezando con el análisis, luego los requerimientos y los elementos relacionados con el diseño de la misma, para terminar con un ejemplo concreto como aplicación de software en Java.

**Figura 27.** Concepto de Modelado Orientado a Objetos.



### 13.3 ANÁLISIS ORIENTADO A OBJETOS

Es un método de análisis que examina los requisitos desde la perspectiva de las clases y objetos, que se encuentran en el enunciado o términos del dominio del problema, tomando como punto de referencia entidades tangibles halladas en el problema, tales entidades varían dependiendo de los diversos casos prácticos, pero en todos los casos son elementos reales que forman parte del problema de forma directa.

El Análisis Orientado a Objetos (AOO) se basa en conceptos de objetos y atributos, el todo y las partes, clases y miembros. Este enfoque pretende conseguir modelos que se ajusten mejor al problema real, a partir de conocer el dominio del problema.

Los documentos que deben tenerse o desarrollarse durante el análisis:

- » Especificación de requisitos o requerimientos (definición y especificación).
- » Escenarios y subescenarios.
- » Prototipos y su evaluación.
- » Diagramas de casos de uso.

La especificación de requisitos o requerimientos comprende las tareas relacionadas con la determinación de las necesidades o de las condiciones a satisfacer para un software nuevo o modificado, tomando en cuenta los diversos requerimientos de los clientes. El propósito es hacer que los mismos alcancen un estado óptimo antes de obtener la fase de diseño en el proyecto. Los requerimientos deben ser medibles, comprobables, sin ambigüedades o contradicciones.

Para la especificación de requerimientos se debe:

- » Identificar las necesidades del usuario (del negocio).
- » Describir los objetivos de la aplicación.
- » Definir características y funciones generales de la aplicación.
- » El equipo de Ingeniería de sistemas y los clientes deben establecer en conjunto las metas y objetivos de la aplicación.

## 13.4 ESCENARIOS

Es una descripción parcial y concreta del comportamiento de un sistema en una determinada situación. Los escenarios son historias de usuarios y sus actividades. Los escenarios destacan objetivos sugeridos por la apariencia y comportamiento del sistema; refiriéndose a lo que los usuarios quieren hacer con el sistema; que procedimientos se usan, cuáles no se usan, se realizan o no satisfactoriamente y que interpretaciones hacen de lo que les sucede.

### Elementos de los escenarios:

- » Configuración (setting). Por ejemplo, una oficina con un empleado enfrente del computador trabajando en una hoja electrónica. Es importante concretar que la persona es el auxiliar contable y que los objetos de trabajo son balances y presupuestos.
- » Agentes o actores. El auxiliar contable es el único agente en el escenario que se describe, pero es normal en las actividades humanas participen más de un actor, cada uno con sus objetivos.
- » Cada escenario implica por lo menos un agente y al menos un objetivo. En el caso de que haya más de un actor y objetivo, se define un agente como actor principal.

## 13.5 PROTOTIPO

Un prototipo puede ser un esquema en papel de una pantalla o pantallas, una imagen electrónica, una simulación en video de una tarea, una representación de una red de computadores, entre otras; de hecho un prototipo puede ser cualquier cosa, desde una representación basada en papel hasta un artefacto o pieza de software.

Son características de los prototipos:

- » Una representación de aquellos aspectos del software que serán visibles para el cliente (por ejemplo, la configuración de la interfaz con el usuario y el formato de los despliegues de salida).
- » El prototipo es evaluado por el cliente para una retroalimentación; gracias a esta se refinan los requisitos del software que se desarrollará.



- » El prototipo permite brindar una representación limitada de un diseño a los usuarios para interactuar con él. Es una herramienta útil para hacer participar al usuario en el desarrollo y poder evaluar el producto ya en las primeras fases del desarrollo.

## 13.6 DIAGRAMAS DE CASOS DE USO

Los casos de uso son una técnica para capturar información de cómo un sistema o negocio funciona actualmente, o de cómo se requiere que se desempeñe. Cada caso de uso proporciona uno o más escenarios que indican cómo debería interactuar el sistema con el usuario o con otro sistema para obtener un objetivo específico.

El modelo de casos de uso especifica la funcionalidad que el sistema debe ofrecer y está conformado por los siguientes elementos:

- » Actores.
- » Casos de uso.
- » Relaciones.

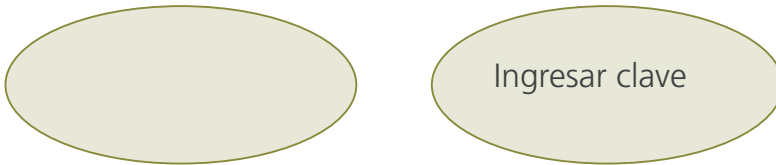
---

**Actores:** modelan diferentes roles que pueden representar los usuarios, máquinas, sensores, entre otros; que interactúan con el sistema a desarrollar.



Se denota por una figura humana plana.

**Casos de uso:** representan todo lo que el usuario puede realizar con el sistema, se denota mediante un ovalo, con la acción correspondiente.



**Relaciones:** permiten asociar actores y casos de uso. Las relaciones que se pueden dar en un diagrama de casos de uso son:

- » **Asociación.** Es el tipo de relación más básica que indica la invocación, desde un actor o caso de uso a otra operación (caso de uso). Esta relación se representa con una línea simple.



- » **Dependencia o instanciación.** Es una forma muy particular de relación entre clases, en la cual una clase depende de otra, es decir, se instancia (se crea). Esta relación se representa con una flecha punteada.



- » **Generalización.** Este tipo de relación es uno de los más utilizados, cumple una doble función dependiendo de su estereotipo, que puede ser de Uso (<<uses>>) o de Herencia (<<extends>>).



- » **Extends.** Se utiliza cuando un caso de uso es similar a otro en sus características.
- » **Uses.** Utilizado cuando se tiene un conjunto de características que son similares en más de un caso de uso y no se desea mantener copiada la descripción de la característica.

## Ejemplos

Se requiere construir un sistema de información que permita modelar las actividades que realiza un usuario con los procesos que hace en su correo electrónico: enviar y recibir mensajes electrónicos y actualizar direcciones de contactos, entre otras; teniendo en cuenta que se debe ingresar al sistema de correo con el login y el password.

Actor

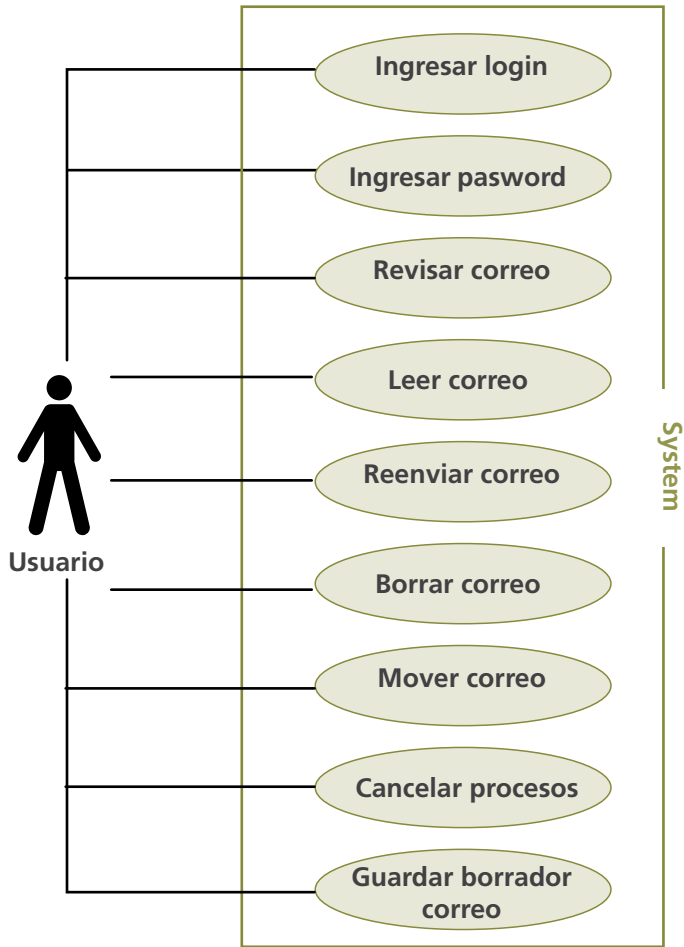


Usuario

Las principales actividades que realiza el usuario en su cuenta de correo son:

1. Ingresar login y password,
2. Revisar correos pendientes,
3. Leer correo,
4. Responder correo,
5. Reenviar comunicación a un determinado grupo,
6. Cancelar envío de correo,
7. Guardar correo en borrador, y
8. Borrar correos

El diagrama del caso de uso estará conformado, por el actor que en este caso es el usuario y los casos de uso corresponden a las actividades realizadas en la cuenta de correo, un cuadro que encierra los casos de uso que corresponde al sistema.



Son características de los diagramas de casos de uso:

- » Describen qué es lo que debe hacer el sistema, pero no como.
- » No son diseñados para representar el diseño y no puede describir los elementos internos de un sistema.
- » Sirven para facilitar la comunicación con los futuros usuarios del sistema y con el cliente, y resultan especialmente útiles para determinar las características necesarias que tendrá el sistema.

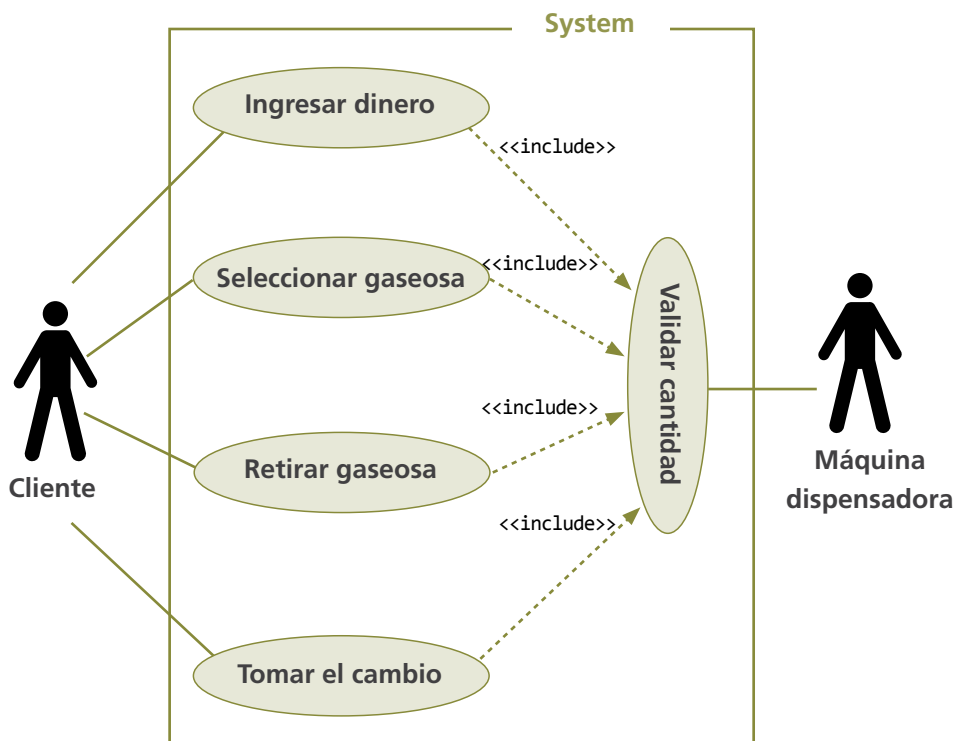
## Ejemplo 2

La administradora de la cafetería de la universidad requiere de una aplicación que le permita automatizar el proceso de compra de las gaseosas de una determinada marca para una máquina expendedora, donde cada gaseosa tiene un costo de \$2.500 y para que la persona interesada pueda adquirir el producto debe ingresarle el valor en monedas, cuyas denominaciones pueden ser de \$100, \$200, \$500; a continuación el cliente podrá seleccionar la gaseosa donde todas tienen costos iguales disponibles en la máquina, además de eso el sistema debe permitir retirar el producto cuando el valor ingresado sea mayor o igual a \$2.500, el retiro de cambio se realiza para los casos en que haya ingresado un valor superior al costo del producto.

**El actor:** es el cliente.

El nombre del caso de uso corresponde con el objetivo del actor, que es normalmente el que comienza el caso de uso, como por ejemplo ingresar dinero.

El diagrama de casos de uso de la máquina expendedora de gaseosas sería:



Para la documentación de los casos de uso se debe tener en cuenta:

- » **Nombre del caso de uso:** se expresa mediante el uso de un verbo en infinitivo, por ejemplo consultar información del cliente, imprimir documento, generar reporte, crear usuario.
- » **Entradas:** datos de entrada proporcionados por el actor.
- » **Resultados:** parcial o final según sea el requerimiento.
- » **Descripción:** resumen que expresa de manera puntual, el requerimiento o la funcionalidad del sistema, y el resultado que se espera que el sistema genere una vez ejecutada su función.
- » **Precondición condiciones:** describen en qué situación se debe encontrar el sistema y su entorno para poder comenzar el caso de uso.
- » **Postcondición:** condiciones que describen en qué situación debe quedar el sistema y su entorno una vez que el caso de uso haya finalizado con éxito.
- » **Flujo normal:** secuencia de interacciones entre los actores y el sistema que lleva a la finalización con éxito del caso de uso.
- » **Flujo alternativo o excepciones:** situaciones eventuales y su tratamiento, que pueden darse durante la secuencia normal.

Los diagramas de casos de uso describen las relaciones y las dependencias entre un grupo de casos de uso y los actores participantes en el proceso. El siguiente formato presenta uno de los ítems de cómo podría ser una de las formas de representar la documentación de los casos de uso:

<b>Nombre:</b>	
<b>Descripción:</b>	
<b>Entradas:</b>	
<b>Resultados:</b>	
<b>Actores:</b>	
<b>Precondiciones:</b>	
<b>Flujo normal:</b>	
<b>Actor</b>	<b>Sistema</b>
<b>Flujo alternativo:</b>	
<b>Pos condiciones:</b>	

Según las necesidades, los casos de uso pueden especificarse con distinto detalle. Como un requisito funcional clásico: se especifica el servicio que debe proporcionar el sistema a los usuarios en texto libre. Por ejemplo:

<b>Nombre:</b>	R1 – ingresar Monedas a la máquina
<b>Descripción:</b>	Se requiere registrar el valor de las monedas depositadas en la máquina por parte del cliente
<b>Entradas:</b>	
	Moneda 100, Moneda 200, Moneda 500
<b>Resultados:</b>	
	Total valor dinero depositado con base a las monedas que han sido ingresadas
<b>Precondiciones:</b>	
	No existe valor procesado previo
<b>Flujo normal:</b>	
	<ol style="list-style-type: none"> <li>1. El usuario pulsa sobre el botón correspondiente al valor de la moneda.</li> <li>2. El sistema muestra en una caja de texto el total del valor ingresado.</li> <li>3. El sistema comprueba el valor de las monedas ingresadas y va generando el total acumulado.</li> <li>4. El sistema deshabilita opciones cuando se ha ingresado el valor del producto o un valor superior</li> </ol>
<b>Flujo alternativo:</b>	
	<p>El usuario no tiene el dinero suficiente.</p> <p>El sistema no entrega el producto hasta completar el valor de este, verificar el total ingresado, y le avisa al usuario de ello permitiéndole que los corrija</p>
<b>Pos condiciones:</b>	
	El valor total ha sido registrado en el sistema



<b>Nombre:</b>	R2 – Retirar gaseosa
<b>Descripción:</b>	Permitir reclamar producto
<b>Entradas:</b>	
	Solicitud producto
<b>Resultados:</b>	
	Total valor dinero depositado en base a las monedas que han sido ingresadas
<b>Precondiciones:</b>	
	El usuario ingreso monedas
<b>Flujo normal:</b>	
	<ol style="list-style-type: none"> <li>1. El usuario pulsa sobre el botón correspondiente solicitar producto</li> <li>2. El sistema verifica total</li> <li>3. El usuario toma el producto</li> </ol>
<b>Flujo alternativo:</b>	
	El sistema comprueba el total del dinero, si el valor es menor le avisa al usuario de ello permitiéndole que complete la cantidad
<b>Pos condiciones:</b>	
	<p>El usuario recibe el producto</p> <p>El usuario recibe el mensaje de error</p>

## 13.7 DISEÑO ORIENTADO A OBJETOS

Es una fase de la metodología orientada a objetos para el desarrollo de Software. Su uso induce a los programadores a pensar en términos de objetos, en vez de procedimientos cuando planifican su código. Un objeto agrupa datos encapsulados y procedimientos para representar una entidad.

El diseño orientado a objetos es la disciplina que define los objetos y sus interacciones para resolver un problema de negocio que fue identificado y documentado durante el análisis orientado a objetos.

- » **Tarjetas CRC** (CRC: Clase, Responsabilidades, Colaboradores). Es una herramienta que por su sencillez permite centrarse en la importancia del objeto y sus operaciones en el sistema.

Como una extensión informal a UML, la técnica de las tarjetas CRC se puede usar para guiar el sistema a través de análisis guiados por la responsabilidad. Las clases se examinan, se filtran y se refinan en base a sus responsabilidades con respecto al sistema, y las clases con las que necesitan colaborar para completar sus responsabilidades.

Una tarjeta CRC, es una tarjeta indexada de 3 x 5 cm. conformada por:

- » El nombre de la clase y su descripción.
- » La responsabilidad de la clase que incluye:
  - Conocimiento interno de la clase.
  - Servicios brindados por la clase.
- » Los colaboradores para las responsabilidades. Un colaborador es una clase cuyos servicios son necesarios para una responsabilidad.

NOMBRE DE LA ABSTRACCIÓN (CLASE)	
Responsabilidades	Colaboradores
(operaciones y atributos)	(relaciones de las clases) Elementos con los que va a interactuar la abstracción.

- » Típicamente los sustantivos de los casos de uso o de los requerimientos del proyecto son útiles para encontrar candidatos a clases, correspondientes al dominio del problema.
- » Los verbos de los casos de uso son candidatos a responsabilidades.

## Enunciado técnico

La **administradora** de la cafetería de la universidad, requiere de una **aplicación**, que le **permita automatizar** el proceso de **compra** de las **gaseosas** de una determinada marca para una **máquina** expendedora, donde cada gaseosa tiene un **costo** de \$2.500, y para que la **persona** interesada pueda adquirir el **producto** debe ingresarle el valor en **monedas** cuyas **denominaciones** pueden ser de \$100, \$200, \$500. A continuación el **cliente** podrá seleccionar la **gaseosa** donde todas tienen costos iguales **disponibles** en la **máquina**, además de eso el sistema debe permitir retirar el producto cuando el valor ingresado sea **mayor o igual** a \$2.500, la **entrega** de **cambio** se realiza para los casos en que haya ingresado un valor superior al costo del producto.

La máquina presentará mensajes de error cuando no haya existencias del producto solicitado, falta ingresar más monedas para completar el valor de la gaseosa, se intenta retirar cambio cuando se ingresa el valor completo del producto.

## Lista de palabras clave:

- » Administradora.
- » Aplicación.
- » Permita automatizar.
- » Compra.
- » Gaseosas.
- » Productos.
- » Costo.
- » Persona.
- » Usuario.
- » Producto.
- » Cambio.
- » Máquina.
- » Costo.
- » Retirar producto.
- » Entregar cambio.
- » Ingresado.
- » Disponible.
- » Mensajes de error.

## Tarjetas CRC

NOMBRE DE LA CLASE: USUARIO	
Responsabilidades	Colaboradores
<ul style="list-style-type: none"> <li>• Ingresar monedas</li> <li>• Seleccionar producto</li> <li>• Retirar productos</li> <li>• Tomar el producto</li> </ul>	<ul style="list-style-type: none"> <li>• Máquina</li> <li>• Monedas</li> </ul>

## NOMBRE DE LA CLASE: PRODUCTO

Responsabilidades	Colaboradores
<ul style="list-style-type: none"> <li>Mantener número de productos existentes.</li> </ul>	<ul style="list-style-type: none"> <li>Máquina</li> <li>Usuario</li> </ul>

## NOMBRE DE LA CLASE: MONEDA

Responsabilidades	Colaboradores
<ul style="list-style-type: none"> <li>Contabilizar cantidad de monedas por denominación y sus correspondientes totales.</li> <li>Mostrar total de dinero ingresado</li> </ul>	<ul style="list-style-type: none"> <li>Máquina</li> </ul>

## NOMBRE DE LA CLASE: MÁQUINA

Responsabilidades	Colaboradores
<ul style="list-style-type: none"> <li>Entregar producto</li> <li>Entregar Cambio</li> <li>Presentar total dinero</li> <li>Presentar total cambio</li> <li>Presentar Mensaje de Error</li> </ul>	<ul style="list-style-type: none"> <li>Usuario</li> <li>Monedas</li> <li>Producto</li> </ul>

Listado tentativo de Clases:

- » Denominaciones.
- » Producto.
- » Máquina.
- » Usuario.

### 13.8 DIAGRAMAS UML (UNIFIED MODELING LANGUAGE)

UML es un lenguaje que permite modelar construir y documentar los elementos que forman un sistema software orientado a objetos. Se ha convertido en el estándar de la industria del software, debido a que ha sido impulsado por los autores de los tres métodos más usados de orientación a objetos: Grady Booch, Ivar Jacobson y Jim Rumbaugh.

El lenguaje UML nace de la necesidad de un lenguaje preciso de especificación que permitiera la comunicación entre los que diseñan el software y de los que lo construyen.

Dentro de los tipos de diagramas a utilizar están:

#### Vista estructural:

- » **Diagrama de clases:** muestra las clases y sus asociaciones.
- » **Diagrama de objetos:** corresponde a una instancia particular del Diagrama de clases.

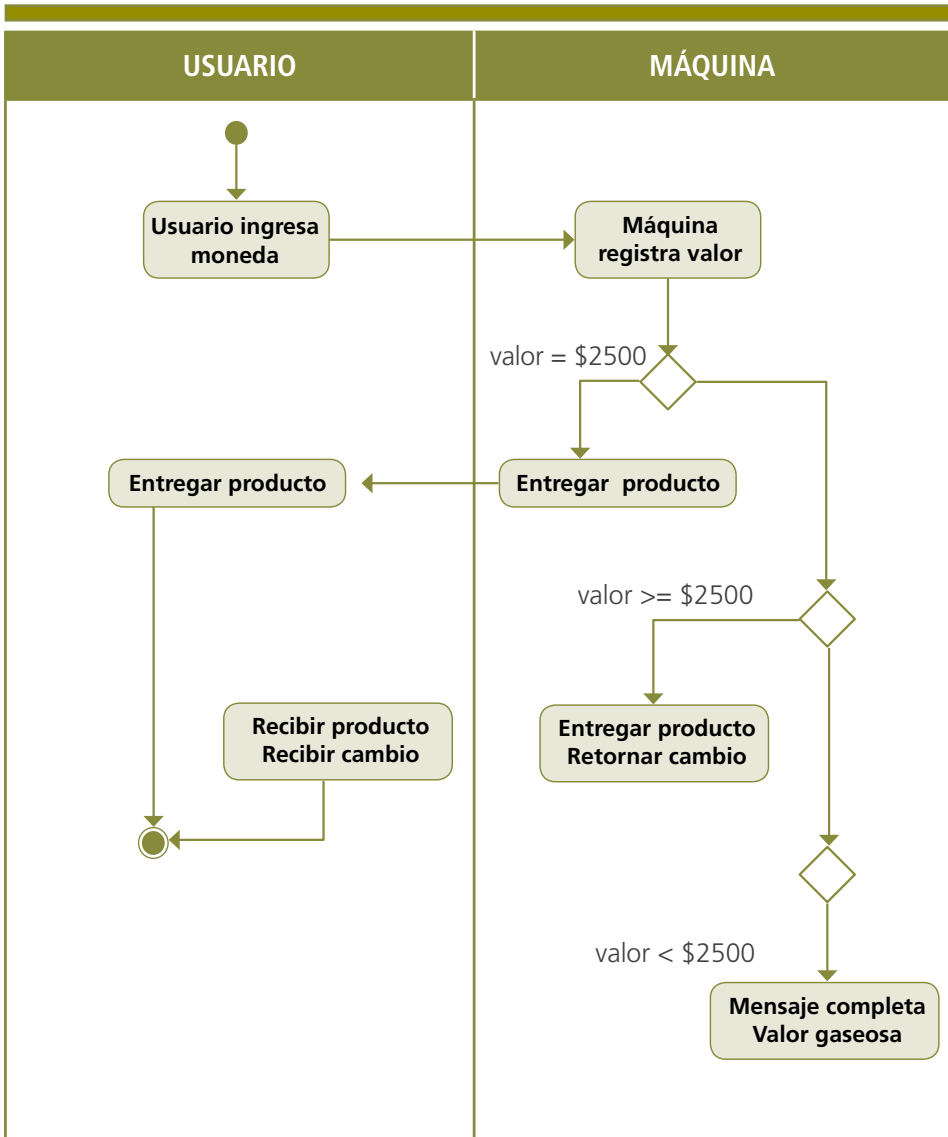
#### Vista de usuario

- » Diagrama de casos de uso

#### Vista de comportamiento

- » Diagrama de actividades

Diagrama de actividades para la máquina de gaseosas del enunciado anterior:



## 13.9 APLICACIONES DE SOFTWARE

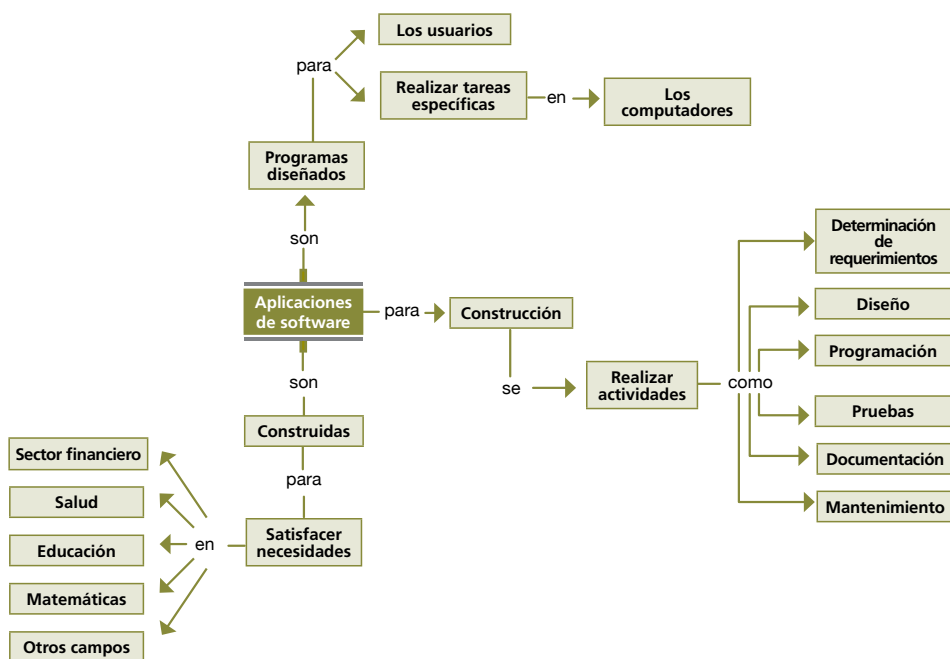
El desarrollo de una aplicación de software en Java implica una serie de actividades que a continuación se relacionan:

- » La identificación de los requerimientos del software a partir de las necesidades que se deben satisfacer del usuario o del negocio.
- » Diseño de la aplicación de software esta conformad componentes de una aplicación (entidades del negocio), el cual describe en general el cómo se construirá una aplicación de software. Para ello se hace uso diagramas UML de clases, despliegue, secuencia, entre otros.
- » Programación, construcción de la aplicación de software en un lenguaje de programación a partir de los diagramas elaborados en la etapa de diseño,
- » Pruebas. Hacen referencia a la comprobación de las necesidades que satisface el software esto es que responda correctamente a los requerimientos planteados en la especificación del problema.
- » Documentación, corresponde a la documentación que es propia desarrollo del software dentro de los que están diferentes diagramas UML, utilizados, manuales usuario sistema, entre otros.
- » Mantenimiento etapa en la que se realizan ajustes y mejoras a la aplicación de software construida.

El mapa mental de la figura 28 amplía el concepto de una aplicación de software.



Figura 28. Concepto de Aplicaciones de software.



## Ejemplo

El Centro Médico El Buen Samaritano, cuya misión es velar por el buen estado nutricional de sus pacientes, requiere la implementación de un aplicativo de software para que su personal médico pueda calcular el Índice de Masa Corporal (IMC) de cada paciente que es atendido. Para lo cual se apoya en la siguiente tabla desarrollada por la Organización Mundial de la Salud:

CLASIFICACIÓN	CLASE	IMC (KG/M2)
Bajo peso		<18,50
	Delgadez severa	<16,00
	Delgadez moderada	16,00 - 16,99
	Delgadez ligera	17,00 - 18,49
Rango normal		18,50 - 24,99
Sobrepeso		≥25,00
	Pre-Obeso	25,00 - 29,99
Obeso		≥30,00
	Obeso Clase I	30,00 - 34,99
	Obeso Clase II	35,00 - 39,99
	Obeso Clase III	≥40,00

**Fuente:** Organización Mundial de la Salud.

Lo recomendado para un estado nutricional bueno, es aquel valor personal que se encuentre dentro del rango específico como normal, en valores que van de 18,50 a 24,99. La Organización Mundial de la Salud (OMS) define el sobrepeso cuando el IMC es mayor o igual a 25 kg/m<sup>2</sup> y obesidad cuando el IMC es mayor o igual a 30 kg/m<sup>2</sup>.

Los valores registrados en la anterior tabla son independientes de la edad y son para ambos sexos.

El IMC de una persona se define de una relación entre el peso y la estatura aplicando la siguiente formula:

Índice de Masa Corporal (IMC, Kg/ m<sup>2</sup>) = Peso(Kg) / Estatura(m<sup>2</sup>)

A partir de la anterior información se requiere la construcción e implementación de una aplicación de software que permitan calcular:

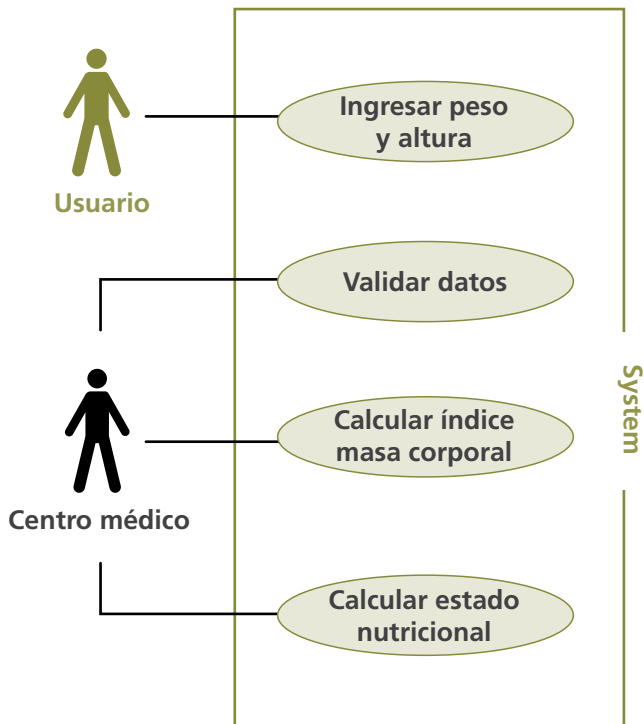
- » El Índice de Masa Corporal expresado en
  - kilogramos y metros<sup>2</sup>
  - Libras y pulgadas (unidades de medida)
- » El estado nutricional de una persona

## Actividades propuestas

- a. Determinar requerimientos.
- b. Elaborar diagrama de casos de uso y documentarlos.
- c. Elaborar listado depurado sustantivos y verbos del enunciado del ejercicio.
- d. Construir las tarjetas CRC.
- e. Diseñar Diagrama de clases.
- f. Construir aplicación (codificación según diagramas de clases).
- g. Generar la documentación (javadoc) del programa.

**a. Definición de requerimientos funcionales:**

- » Ingresar información del paciente.
- » Validar datos.
- » Calcular el índice de masa corporal.
- » Determinar el estado nutricional.

**b. Diagrama de casos de uso.**

## Documentación casos de uso

<b>Nombre:</b> (R1) Ingresar peso y estatura	
<b>Descripción:</b> Ingreso de datos correspondientes al peso y estatura como elementos para calcular el índice de masa corporal.	
<b>Entradas:</b>	
Se dan a partir de las medidas del peso y la estatura del paciente, correspondiente a datos numéricos positivos.	
<b>Resultados:</b>	
Registro de los datos correspondientes al peso y la estatura en el sistema.	
<b>Actores</b>	
Usuario y el centro médico.	
<b>Precondiciones:</b>	
Ninguna.	
<b>Flujo normal:</b>	
Actor	Sistema
1 Usuario ingresa el peso y la estatura.	2. Sistema visualiza peso y estatura. 3. Sistema registra datos ingresados.
<b>Flujo alternativo:</b>	
1.1 Usuario ingresa datos con valores negativos. 1.2 Usuario ingresa datos con formato errado.	
<b>Pos condiciones:</b>	
Sistema registra los datos ingresados.	

Documentación casos de uso

<b>Nombre:</b> (R2) calcular Índice de Masa Corporal (IMC)	
<b>Descripción:</b> Se realiza proceso para obtener el IMC a partir del peso y la masa	
<b>Entradas:</b>	
Tomados del (R1) Ingresar peso y estatura.	
<b>Resultados:</b>	
Obtener el cálculo del IMC.	
<b>Actores</b>	
Sistema (centro médico).	
<b>Precondiciones:</b>	
(R1)	
<b>Flujo normal:</b>	
Actor	Sistema
	1. Sistema calcula Índice de Masa Corporal. 2. Visualización del resultado del Índice de Masa Corporal.
<b>Flujo alternativo:</b>	
ninguno	
<b>Pos condiciones:</b>	
Sistema tiene registrado el Índice de Masa Corporal	

<b>Nombre:</b> (R3) estado nutricional del paciente	
<b>Descripción:</b> Se realiza proceso para obtener el estado nutricional del paciente del cálculo del IMC.	
<b>Entradas:</b>	
Tomados del (R2) IMC calculado.	
<b>Resultados:</b>	
Obtener el estado nutricional del paciente.	
<b>Actores</b>	
Sistema (centro médico).	
<b>Precondiciones:</b>	
(R2)	
<b>Flujo normal:</b>	
Actor	Sistema
	1. Sistema determina el estado nutricional del paciente. 2. Visualización del resultado del estado nutricional del paciente.
<b>Flujo alternativo:</b>	
ninguno	
<b>Pos condiciones:</b>	
Sistema presenta el estado nutricional del paciente.	

**Nota:** desarrollar los otros casos de uso planteados en el ejercicio.

### c. Enunciado del problema subrayando palabras clave.

El **Centro Médico** El Buen Samaritano, cuya misión es velar por el buen **estado nutricional** de sus **pacientes**, requiere la implementación de un **aplicativo** de software para que su **personal médico** pueda calcular el **Índice de Masa Corporal (IMC)**, de cada paciente que es atendido.

Para lo cual se apoya en la siguiente tabla desarrollada por La Organización Mundial de la Salud.

Lo recomendado para un estado nutricional bueno, es aquel valor personal se encuentre dentro del **rango** específico como **normal**, en valores que van de 18,50 - 24,99. La Organización Mundial de la Salud (OMS) define el sobrepeso cuando el IMC es mayor o igual a 25 kg/m<sup>2</sup> y obesidad cuando el IMC es mayor o igual a 30 kg/m<sup>2</sup>.

Índice de Masa Corporal (IMC, kg/m<sup>2</sup>) = Peso (kg) / Estatura (m<sup>2</sup>)

A partir de la anterior información se requiere la construcción e implementación de una aplicación de software que permitan calcular:

- » El Índice de Masa Corporal expresado en:
  - **(kilogramos y metros<sup>2</sup>)**.
  - **Libras y pulgadas** (unidades de medida).
- » El **estado nutricional** de una persona.

Listado de palabras clave del enunciado.

Centro médico, paciente, médico, estado, aplicativo, índice de masa corporal, rango, normal, kilogramos y metros, libras y pulgadas



Listado de posibles clases del sistema

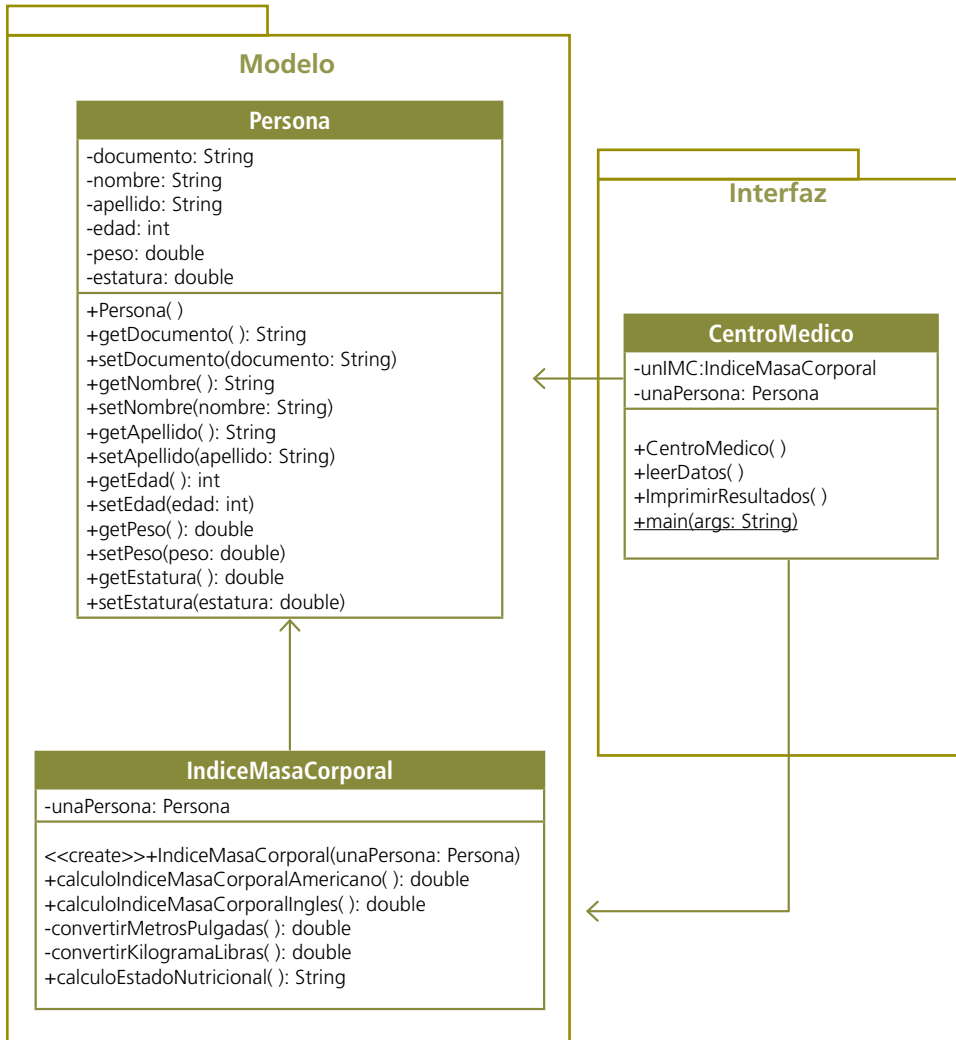
- » Índice de masa corporal.
- » Centro médico.

#### d. Elaboración de las Tarjetas CRC

Nombre de la clase:	IndiceMasaCorporal
RESPONSABILIDAD	COLABORACIÓN
Registrar información del peso y estatura del paciente.	Centro medico
Calcular el Índice de Masa Corporal del paciente.	
Hallar estado nutricional del paciente.	

Nombre de la clase:	CentroMedico
RESPONSABILIDAD	COLABORACIÓN
Lectura y visualización de información.	IndiceMasaCorporal
Validar datos ingreso de datos numéricos	

e. Diagrama de clases.



## Documentación métodos de la clase Índice Masa Corporal

Método para calcular el Índice Masa Corporal expresado en kilogramos y metros (Sistema Americano).

### Análisis.

**Entrada:** corresponde a los atributos de la clase de la masa y la estatura que fueron ingresados

**Salida:** es el resultado de aplicar la formula

Índice de Masa Corporal (IMC, kg/ m<sup>2</sup>) = Peso (kg) / Estatura (m<sup>2</sup>).

Nombre del método	calcularIndiceMasaCorporalAmericano
Entrada (lista de parámetros)	No tiene
Resultado (tipo de dato de retorno)	double Índice de masa corporal
Solución planteada	Cálculo a partir de aplicar la fórmula IMC = peso / estatura*estatura;
Firma del método	calcularIndiceMasaCorporalAmericano ( )

```
public double calcularIndiceMasaCorporalAmericano( ) {
    return peso / estatura*estatura;
}
```

Pruebas de escritorio.

Supongamos las siguientes medidas:

PESO	ESTATURA	RETORNO
60kg	1.72	20.28
78Kg	1.70	26.98

## Documentación Métodos de la clase Índice Masa Corporal

Método que calcularEstadoNutricional

### Análisis.

**Entrada:** corresponde a los atributos de la clase de la masa y la estatura que fueron ingresados

**Salida:** es el resultado de aplicar la formula

Índice de Masa Corporal ( IMC, Kg/ m<sup>2</sup>) = Peso(Kg) / Estatura(m<sup>2</sup>)

<b>Nombre del método</b>	calcularEstadoNutricional
<b>Entrada (lista de parámetros)</b>	No tiene
<b>Resultado (tipo de dato de retorno)</b>	double Índice de masa corporal
<b>Solución planteada</b>	<p>A partir del índice de masa corporal se evalúa si el peso está entre un determinado rango</p> <p>si (imc &lt; 16)  si (imc &gt;=16 &amp;&amp; imc &lt; 17)  si (imc &gt;=17 &amp;&amp; imc &lt; 18.5)  si (imc &gt;=18.5 &amp;&amp; imc &lt; 25)  si (imc &gt;=25 &amp;&amp; imc &lt; 30)  si(imc &gt;=30 &amp;&amp; imc &lt; 35)  si (imc &gt;=35 &amp;&amp; imc &lt; 40)  si (imc &gt;=40 )</p>
<b>Firma del método</b>	calcularEstadoNutricional( )

```

public String calcularEstadoNutricional( ) {
    double imc = calcularIndiceMasaCorporalAmericano();
    String estado = "";
    if (imc < 16) {
        estado = "Delgadez severa ";
    } else if (imc >=16 && imc < 17) {
        estado = "Delgadez moderada";
    } else if (imc >=17 && imc < 18.5) {
        estado = "Delgadez ligera";
    } else if (imc >=18.5 && imc < 25) {
        estado = "Peso Normal";
    } else if (imc >=25 && imc < 30) {
        estado = "Pre-Obeso";
    } else if (imc >=30 && imc < 35) {
        estado = "Obeso Clase I";
    } else if (imc >=35 && imc < 40) {
        estado = "Obeso Clase II";
    } else if (imc >=40 ) {
        estado = "Obeso Clase III";
    }
    return estado;
}

```

Pruebas de escritorio.

Supongamos las siguientes medidas:

PESO	ESTATURA	IMC	RETORNO
60kg	1.72	20.28	20.28
78Kg	1.70	26.98	26.98

## f. Codificación de las clases `IndiceMasaCorporal` y `CentroMedico`

```
/**
 * clase que modela la información de la persona.
 * @ Project : indice de masa corporal
 * @ File Name : IndiceMasaCorporal
 * @author : Miguel Hernández
 * @version 1.0
 */

public class Persona {
    private String documento;
    private String nombre;
    private String apellido;
    private int edad;
    private double peso;
    private double estatura;

    public Persona() {
        this.documento = "123";
        this.nombre = "Luis Eduardo";
        this.apellido = "Baquero Rey";
        this.edad = 25;
        this.peso = 85;
        this.estatura = 190;
    }

    public Persona(String documento, String nombre, String
        apellido,
        int edad, double peso, double estatura) {
        this.documento = documento;
        this.nombre = nombre;
        this.apellido = apellido;
        this.edad = edad;
        this.peso = peso;
        this.estatura = estatura;
    }
}
```

```
public String getDocumento() {
    return documento;
}

public void setDocumento(String documento) {
    this.documento = documento;
}

public String getNombre() {
    return nombre;
}

public void setNombre(String nombre) {
    this.nombre = nombre;
}

public String getApellido() {
    return apellido;
}

public void setApellido(String apellido) {
    this.apellido = apellido;
}

public int getEdad() {
    return edad;
}

public void setEdad(int edad) {
    this.edad = edad;
}

public double getPeso() {
    return peso;
}

public void setPeso(double peso) {
    this.peso = peso;
}

public double getEstatura() {
    return estatura;
}
```

```

        public void setEstatura(double estatura) {
            this.estatura = estatura;
        }
    }

/**
 * clase que modela las operaciones para calcular
 * indice de masa corporal y el estado nutricional
 * @ Project : indice de masa corporal
 * @ File Name : IndiceMasaCorporal
 * @author : mhernandezb lebaqueror
 * @version 1.0
 */
public class IndiceMasaCorporal {

    private Persona unaPersona;

    public IndiceMasaCorporal(Persona unaPersona) {
        this.unaPersona = unaPersona;
    }

    public double calcularIndiceMasaCorporalAmericano()
    {
        return unaPersona.getPeso()/ (
            unaPersona.getEstatura()*unaPersona.getEstatura());
    }

    public double calcularIndiceMasaCorporalIngles()
    {
        return convertirMetrosPulgadas ( ) / (
            convertirKilogramasLibras( ) *
            convertirKilogramasLibras( ) );
    }

    private double convertirMetrosPulgadas ( ) {
        final double CONVERSION = 39.37;
        return unaPersona.getEstatura() * CONVERSION;
    }
}

```



```

private double convertirKilogramaLibras( ) {
    final double CONVERSION = 2.2046;
    return unaPersona.getPeso() * CONVERSION;
}

public String calcularEstadoNutricional( ) {
    /*
    * Clasificación Clase IMC (Kg/m2)
    *      Bajo Peso <18,50
    *
    *      Delgadez severa <16,00
    *      Delgadez moderada 16,00 - 16,99
    *      Delgadez ligera 17,00 - 18,49
    *      Rango normal 18,50 - 24,99
    *      Sobrepeso >=25,00
    *      Pre-Obeso 25,00 - 29,99
    *      Obeso >=30,00
    *      Obeso Clase I 30,00 - 34,99
    *      Obeso Clase II 35,00 - 39,99
    *      Obeso Clase III >=40,00
    */
    double imc = calcularIndiceMasaCorporalAmericano();
    String estado = "";
    if (imc < 16) {
        estado = "Delgadez severa ";
    } else if (imc >=16 && imc <17) {
        estado = "Delgadez moderada";
    } else if (imc >=17 && imc <18.5) {
        estado = "Delgadez ligera";
    } else if (imc >=18.5 && imc <25) {
        estado = "Peso Normal";
    } else if (imc >=25 && imc <30) {
        estado = "Pre-Obeso";
    } else if (imc >=30 && imc <35) {
        estado = "Obeso Clase I";
    } else if (imc >=35 && imc <40) {
        estado = "Obeso Clase II";
    } else if (imc >=40 ) {
        estado = "Obeso Clase III";
    }
    return estado;
}
}

```

```

/**
 * clase que modela los procesos de lectura e impresión
 * para calcular indice de masa corporal y el estado nutricional
 * @ Project : indice de masa corporal
 * @ File Name : CentroMedico
 * @author : mhernandezb lebaqueror
 * @version 1.0
 */
import java.text.DecimalFormat;
import java.util.Scanner;

public class CentroMedico {
    private IndiceMasaCorporal unIMC;
    private Persona unaPersona;

    public CentroMedico() {
        this.unaPersona = new Persona();
    }

    public void leerDatos() {
        String documento = "";
        String nombre = "";
        String apellido = "";
        int edad = 0;
        double peso = 0.0;
        double estatura = 0.0;
        Scanner lea = new Scanner(System.in);
        System.out.print("Documento ");
        documento = lea.nextLine();
        System.out.print("Nombre ");
        nombre = lea.nextLine();
        System.out.print("Apellido ");
        apellido = lea.nextLine();
        System.out.print("Edad ");
        edad = lea.nextInt();
        System.out.print("Estatura ");
        estatura = lea.nextDouble();
        System.out.print("Peso ");
        peso = lea.nextDouble();
        //cargar los datos al objeto
    }
}

```

```
        unaPersona.setDocumento(documento);
        unaPersona.setNombre(nombre);
        unaPersona.setApellido(apellido);
        unaPersona.setEdad(edad);
        unaPersona.setEstatura(estatura);
        unaPersona.setPeso(peso);
        this.unIMC = new IndiceMasaCorporal(unaPersona);
    }

    public void ImprimirResultados( ) {
        //formato decimal de dos cifras
        DecimalFormat formatoDecimal = new
        DecimalFormat("####.##");
        System.out.println("\nEl estado de la Persona\n");
        System.out.println("Documento "+unaPersona.getDocu
        mento());
        System.out.println("Nombre "+unaPersona.getNombre());
        System.out.println("Apellido "+unaPersona.getApelli
        do());
        System.out.println("Edad "+unaPersona.getEdad());
        System.out.println("Estatura "+unaPersona.getEstatu
        ra());
        System.out.println("Peso "+unaPersona.getPeso());

        System.out.println("Indice de Masa Corporal en Kg/M2
        "+formatoDecimal.format(unIMC.calcularIndiceMasaCor
        poralAmericano()));
        System.out.println("Indice de Masa Corporal lbs/pulg2
        "+formatoDecimal.format(unIMC.calcularIndiceMasaCorpo
        ralIngles()));
        System.out.println("Estado Nutricional
        "+unIMC.calcularEstadoNutricional());
    }

    public static void main(String[] args) {
        CentroMedico unCentroMedio = new CentroMedico();
        unCentroMedio.leerDatos();
        unCentroMedio.ImprimirResultados();
    }
}
```

- g. Mediante la utilización como eclipse o netbeans, generar la respectiva documentación con la utilidad de javadoc.
- 

### 13.10 LECTURAS RECOMENDADAS

- » UML.
- » Diagramas de UML.
- » Principios del modelado.

### 13.11 PREGUNTAS DE REVISIÓN DE CONCEPTOS

- » Importancia del modelamiento de una aplicación de software.
- » Qué es el modelado orientado a objetos.
- » Utilidad de documentar una aplicación de software.

### 13.12 EJERCICIOS

Para cada uno de los siguientes enunciados realizar las siguientes actividades.

- a. Determinar requerimientos.
- b. Elaborar diagrama de casos de uso y documentarlos.
- c. Elaborar listado depurado sustantivos y verbos del enunciado del ejercicio.
- d. Construir las tarjetas CRC.
- e. Diseñar Diagrama de clases.
- f. Construir aplicación (codificación según diagramas de clases).
- g. Generar la documentación (javadoc) del programa.

1. El centro médico El Buen Samaritano cuya misión es velar por la salud y bienestar de sus pacientes en la actualidad dentro de su políticas para brindar un mejor servicio, necesita la implementación de un sistema automatizado que tomando como base los resultados obtenidos en el laboratorio de análisis clínicos, donde un médico determina si una persona tiene anemia o no, lo cual depende de su nivel de hemoglobina en la sangre, de su edad y de su género. Si el nivel de hemoglobina que tiene una persona está en el rango que le corresponde, este determina su resultado positivo y en caso contrario negativo. La información en la que el médico se basa para obtener el resultado es la siguiente:

EDAD (MESES)	NIVEL HEMOGLOBINA	GÉNERO
0 - 1	13 - 26 g%	Femenino o masculino
$1 < x \leq 6$	10 - 18 g%	Femenino o masculino
$6 < x \leq 12$	11 - 15 g%	Femenino o masculino

EDAD (MESES)	NIVEL HEMOGLOBINA	GÉNERO
$1 < x \leq 5$	11.5 - 15 g%	Femenino o masculino
$5 < x \leq 10$	12.6 - 15.5 g%	Femenino o masculino
$10 < x \leq 15$	13 - 15.5 g%	Femenino o masculino
Mujeres $\geq 15$	12 - 16 g%	Femenino
Hombres $\geq 15$	14 - 18 g%	masculino

2. Desarrollar un aplicación que permita realizar el modelado de la pantalla de un reloj digital e imprima las horas y minutos separadas por dos punto, por ejemplo, 02:35; 12:40; 1:53 p. m.; las 11:59 a. m. es un minuto antes de las doce 00:00.

La aplicación debe permitir adelantar o atrasar los minutos, las horas y fijar una hora inicial en el reloj; e imprimir además de las horas, los minutos y el meridiano (a. m., p. m.).

3. El departamento de ciencias básicas requiere una aplicación que permita al usuario seleccionar una de las siguientes figuras geométricas: La circunferencia, el cuadrado, el rectángulo, el rombo, el trapecio y el romboide para luego calcular su área y perímetro.

La aplicación deberá:

- » Permitir leer los datos por consola según sea el tipo de figura y solicitar únicamente los datos necesarios.
- » Antes de imprimir el resultado se debe verificar si los datos ingresados son valores positivos o de lo contrario volver a capturarlos.
- » Imprimir los resultados de la figura seleccionada

### 13.14 REFERENCIAS BIBLIOGRÁFICAS

Aguilar, L. J. (2004). *Fundamentos de programación algoritmos y estructura de datos*. Tercera Edición. Ciudad: México: McGraw Hill.

Barnes D., Kölling M. (2007), *Programación orientada a objetos con Java*. Ciudad Madrid, Prentice Hall.

Bertrand M. (1999). *Construcción de software orientado a objetos* (2ª edición) Ciudad: Madrid, Prentice Hall.

Booch, G. (1996). *Análisis y diseño orientado a objetos con aplicaciones*. Ciudad: México Addison Wesley.

Deitel y Deitel. (2012). *Cómo programar en Java*. Ciudad: México Editorial Pearson (Prentice Hall).

Fowler, M., y Scott, K. UML. (1999). *Gota a gota*. Ciudad: México Editorial Pearson.

Graig, L. (2003). *UML y Patrones: Una introducción al análisis y diseño orientado a objetos y al proceso unificado* (2ª edición) Ciudad: México Prentice Hall.

Rumbaugh, J., Jacobson, I., y; Booch, G.: (2005). *The Unified Modeling Language User Guide*, San Francisco: Addison-Wesley, Reading, Mass. ISBN-13: 078-5342267976 ISBN-10: 0321267974

Villalobos, J., y Casallas, R. (2006). *Fundamentos de programación, aprendizaje activo basado en casos*. Ciudad: México Editorial Pearson.





# CAPÍTULO 14

## DOCUMENTACIÓN





## 14.1 TEMÁTICA A DESARROLLAR

- » JavaDoc.
- » Soporte Javadoc para los IDE.
- » API Java 8.

## 14.2 INTRODUCCIÓN

La documentación de Java es muy completa para cada versión del JDK, esa documentación se puede consultar en línea o tenerla en un archivo aparte, previamente descargado. El API presenta los packages, clases e interfaces con descripciones bastante detalladas de las variables y métodos, así como sobre las relaciones jerárquicas.

La documentación de Java está escrita en HTML y se explora con un navegador web como Microsoft Internet Explorer, Mozilla FireFox, Google Chrome, Opera o Safari.

## 14.3 JAVADOC

Es una herramienta para generar documentación para una aplicación en Java a partir de comentarios, Javadoc realiza algunos comentarios, de los que exige una sintaxis especial, deben comenzar por `"/**"` y terminar por `"*/"`, incluyendo una descripción y algunas etiquetas especiales que se insertan en el código fuente de una aplicación en Java. La documentación se genera en formato HTML la cual permite ser consultada por medio de un navegador.

El paquete de desarrollo de Java incluye la herramienta javadoc con la cual se genera un conjunto de páginas web, a partir de los archivos de código correspondientes a los diferentes programas que conforman la aplicación. Esta herramienta toma en consideración algunos comentarios para generar una documentación estructurada de las clases atributos y métodos.

Es de notar que javadoc ayuda a la comprensión de la arquitectura de la solución, se dice que javadoc se centra en la interfaz (API - Application Programming Interface) de las clases y paquetes de Java.

## La herramienta Javadoc

La sintaxis con la que tiene que ser invocada la herramienta es: javadoc [opciones] Clase

Aunque existe una gran variedad de opciones, estas son algunas de las que nos pueden resultar útiles en el desarrollo de las prácticas:

- javadoc -public Clase: presenta sólo las clases y los miembros públicos.
- javadoc -protected Clase: presenta las clases y los miembros protegidos.
- (protected) y públicos.
- javadoc -package Clase: presenta las clases y los miembros con acceso de paquete, los protegidos y los públicos.
- javadoc -private Clase: presenta todas las clases y todos los miembros, incluyendo los privados.

## Las etiquetas

Aparte de los comentarios propios, la utilidad javadoc proporciona una serie de etiquetas para completar la información que se necesite dar de una determinada clase o método.

La herramienta Javadoc escanea las etiquetas cuando están incluidas dentro del programa en un comentario Javadoc.

Estas etiquetas para documentación permiten generar una API en HTML a partir del código fuente con los comentarios. Las etiquetas comienzan con el símbolo arroba (@) y son sensibles a mayúsculas y minúsculas.

Una etiqueta se debe ubicar al inicio de una línea, o al menos sólo precedida por espacio(s) y asterisco(s), o será tratada como texto normal por el compilador.

### Hay dos tipos de etiquetas:

- **Etiquetas de bloque:** solo se pueden utilizar en la sección de etiquetas que sigue a la descripción principal. Son de la forma: @etiqueta
- **Etiquetas inline:** se pueden utilizar tanto en la descripción principal como en la sección de etiquetas. Son de la forma: {@tag}, es decir, se escriben entre los símbolos de llaves.

## Ejemplos.

### De algunas etiquetas:

#### @author

Muestra el autor de la clase en el argumento nombre. Un comentario de este tipo puede tener más de un autor en cuyo caso se puede usar tantas etiquetas de este tipo como autores hayan colaborado en la creación del código fuente o bien se pueden ponerlos a todos en una sola etiqueta. En este último caso, Javadoc inserta una (,) y un espacio entre los diferentes nombres.

### Ejemplos

```
/**
 * @author (your name)
 */
/**
 * @author Luis Eduardo Baquero, Miguel Hernández Bejaran
 */
```

#### @deprecated

Esta etiqueta marca un identificador como desactualizado, es decir, inadecuado para continuar con su uso. Es conveniente asegurarse de que la entidad desactualizada continúa funcionando de forma que no se dañe el código existente que no ha sido actualizado. La desactualización ayuda a animar a los desarrolladores de código a actualizarse a la última versión, preservando la integridad del código existente.

### Ejemplos

**@Deprecated** Este método se eliminará en próximas versiones

```
public double obtenerVolumen() {
    return 0.0;
}
```

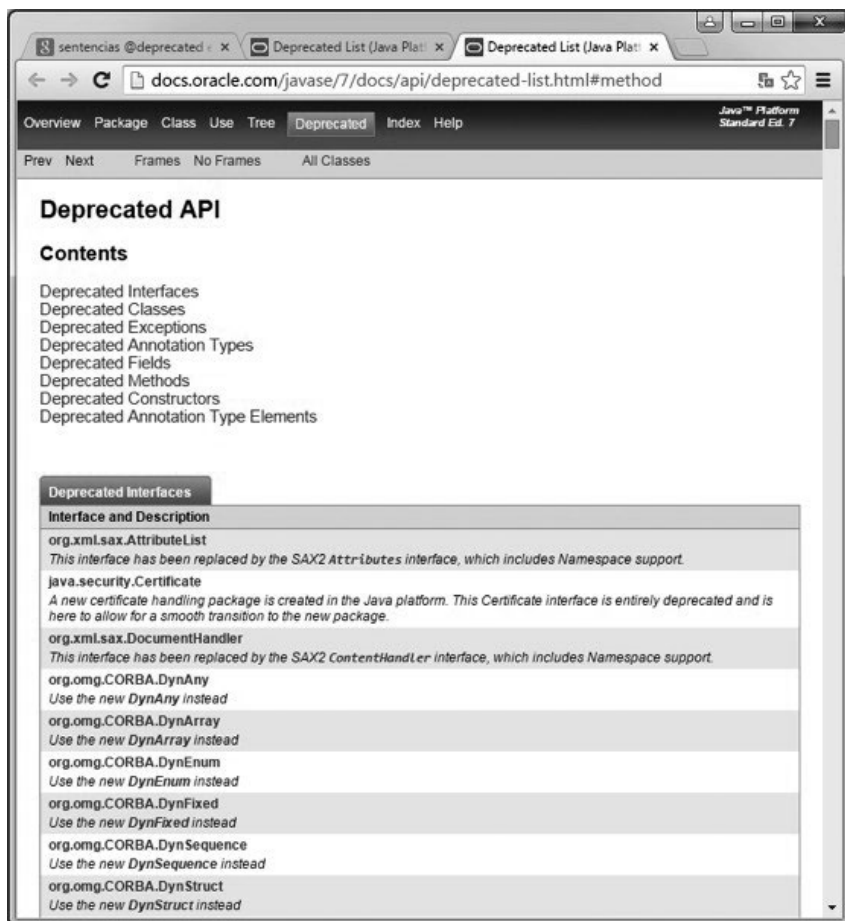
**@Deprecated:** para marcar un método como desaprobado, no debe ser utilizado en la programación ya que de seguro puede generar problemas, aunque esté disponible.

## @Deprecated

```
public static void metodoEstatico(){
    System.out.println("Estatico Desaprobado");
}
```

Para obtener mayor información al respecto esta el API de java, tal como se presenta en la siguiente imagen del sitio oficial

<http://docs.oracle.com/javase/7/docs/api/deprecated-list.html#method>



### **{@link}**

Se debe crear un enlace con el texto “etiqueta” que apunta a la documentación del paquete, la clase o el miembro especificado.

### **Ejemplo**

```
/** Método que invoca a {@link #obtenerLectura(String)}. */

public void processLineByLine() {
    obtenerLectura(scanner.nextLine());
}

protected void obtenerLectura(String aLine){
    Scanner scanner = new Scanner(aLine);
}
```

### **@param**

Anexa un parámetro en la sección Parámetros. La descripción se puede escribir en más de una línea. Esta etiqueta solo es válida en los comentarios para documentación de métodos y constructores.

### **Ejemplos**

```
/**
 * constructor con dos parametros
 * @param x
 * @param y
 */

public Punto (double x, double y) {
    this.x = x;
    this.y = y;
}

/**
 * metodo que actualiza el apellido
 * @param apellido
 */
public void setApellido(String apellido) {
    this.apellido = apellido;
}
```

## @return

Agrega una cabecera con un enlace que apunta a una referencia. Un comentario para documentación puede contener más de una de estas etiquetas; todos los enlaces de este tipo se agrupan bajo la misma cabecera.

### Ejemplos

```
/**
 * Método que retorna el objeto de la fecha
 */
public String toString() {
    return ""+dia + "/" + mes + "/" + año;
}

/**
 * método que retorna el apellido
 * @return nombre
 */
public String getApellido() {
    return apellido;
}
```

## @version

Anexa en la cabecera de la documentación generada con la versión de esta clase. Por versión, normalmente se hace referencia a la versión de la aplicación de software.

### Ejemplo

```
/**
 * @version 1.0
 */
/**
 * @version 24 de julio de 2015
 */
```

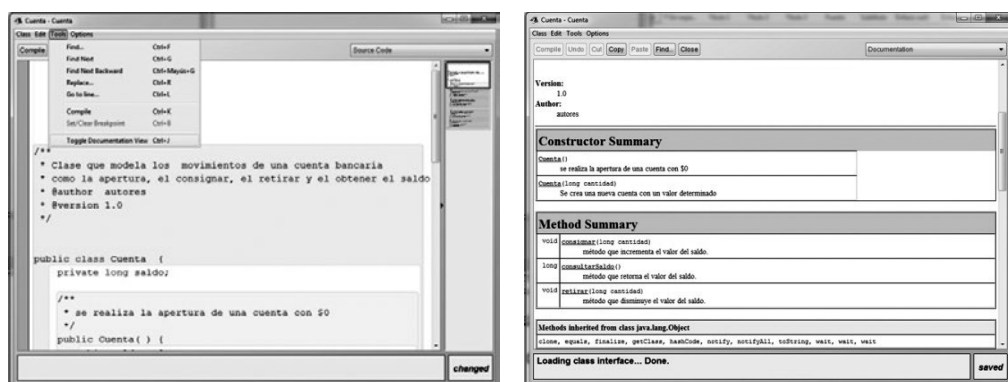
## 14.4 JAVADOC EN LOS IDE

### BlueJ

Soporte de BlueJ para javadoc. Si un proyecto ha sido comentado haciendo uso de Javadoc, BlueJ ofrece utilidades para generar la documentación HTML completa. Para lo cual en la ventana principal se debe seleccionar la opción Tools/Project Documentation del menú, luego se generará la documentación y se visualiza en la ventana de un navegador.

En el editor de BlueJ se puede pasar de la vista del código fuente de una clase a la vista de su documentación cambiando la opción Source Code a Documentation en la parte derecha de la ventana del menú Tools del editor. Esta opción ofrece una vista previa y rápida de la documentación pero no contendrá referencias a la documentación de las superclases o de las clases que se usan. La figura 28 presenta la documentación en BlueJ

Figura 28. Documentacion en BlueJ.



Documentación en java

Documentación java doc

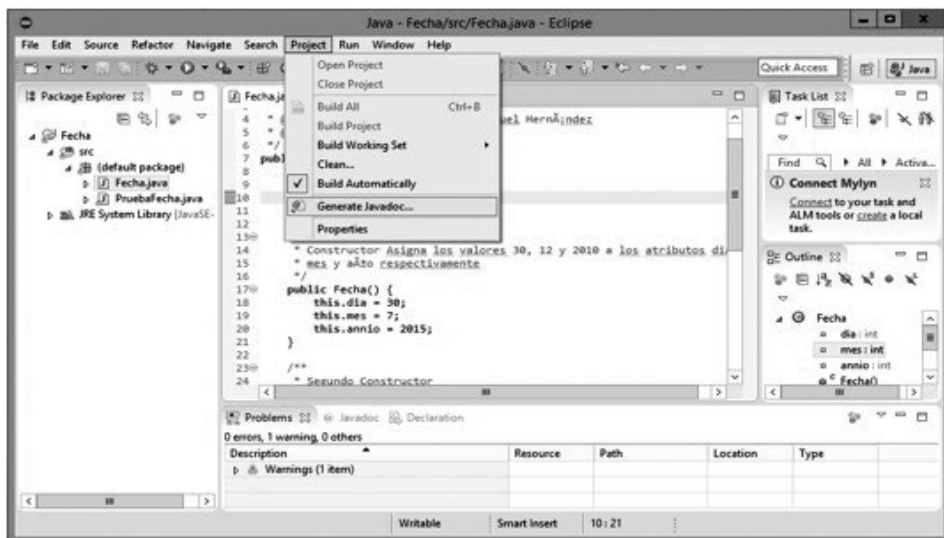
### Soporte de Javadoc para Eclipse

Eclipse permite, de una forma muy sencilla generar, automáticamente, la documentación del propio proyecto.

Antes de crear los archivos de documentación, es necesario configurar la herramienta JavaDoc que Eclipse debe utilizar. Para ello basta con escribir la ubicación del ejecutable javadoc en la opción Window-Preferences-Java-Javadoc accesible desde el menú principal

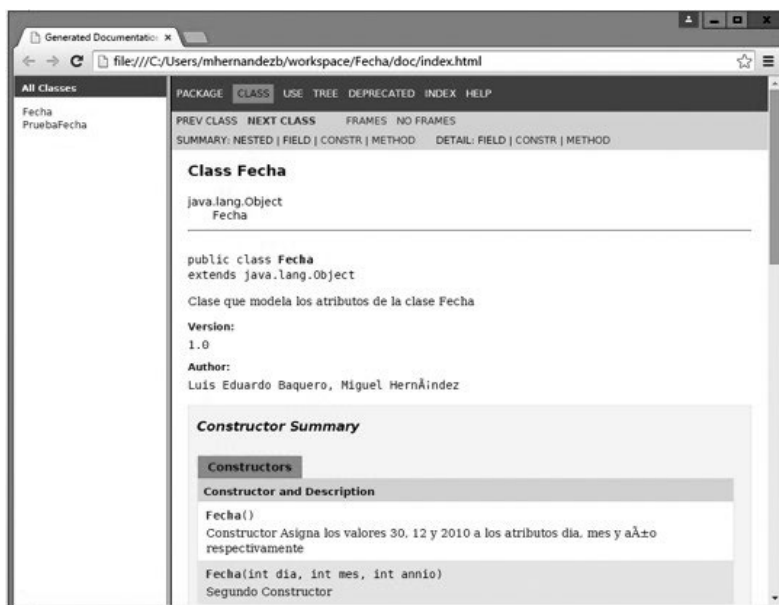


Figura 29. Generación Javadoc en Eclipse.



En la figura 30 se visualiza la documentación Javadoc, generada en eclipse.

Figura 30. Documentación Javadoc en Eclipse.



## Soporte de Javadoc para Netbeans

NetBeans dispone de todo un entorno para crear documentación javadoc a partir de Tools-Javadoc manager se llama al administrador de javadocs, el cual es capaz de montar filesystems que contienen documentos javadoc y gestionar su uso. Lo normal es que exista una carpeta llamada docs, dentro de la raíz del paquete que es en la cual se genera la documentación.

En la carpeta de documentación el archivo index.html será el índice general. Mediante el administrador de archivos se debe montar esta carpeta (Mount) para que pueda ser buscada.

**Figura 31.** Generación Javadoc en Netbeans.

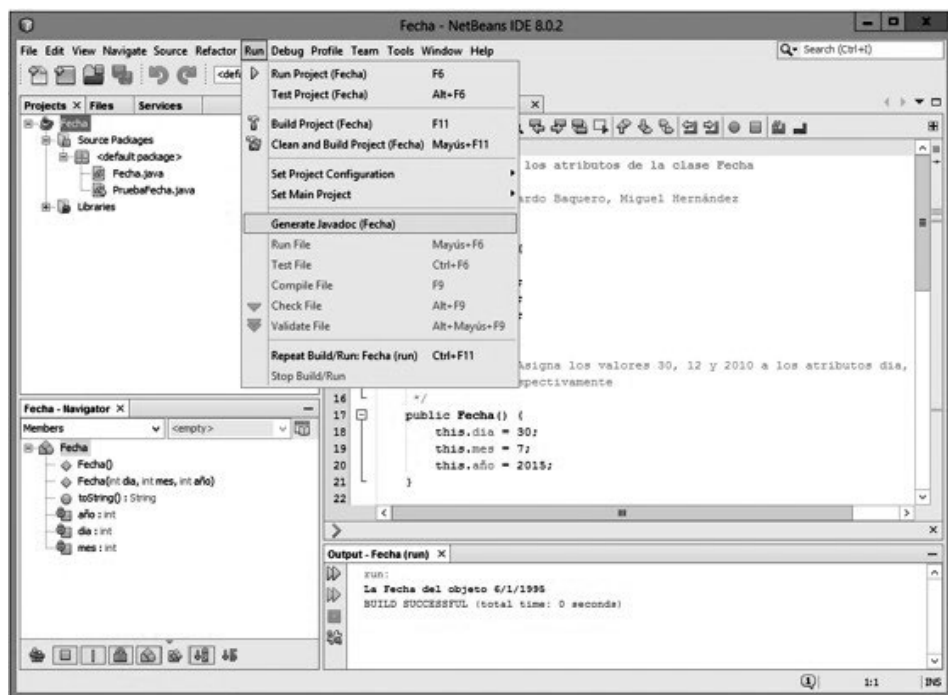
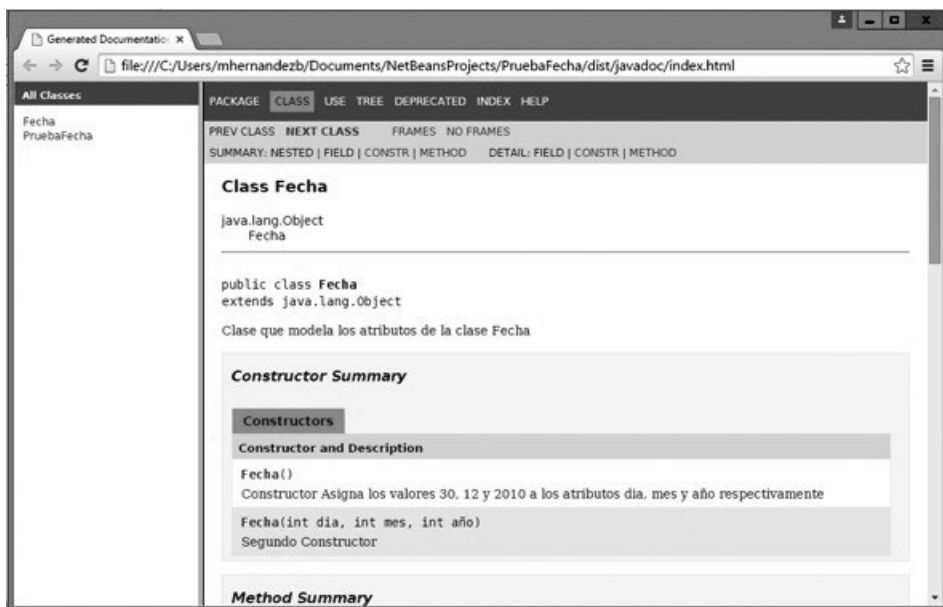


Figura 32. Documentación Javadoc en Netbeans.



## 14.5 API DE JAVA

Es una herramienta que integra y provee de un conjunto de clases utilitarias para la construcción de aplicaciones de software en Java.

El API, incluye las clases de la versión correspondiente (6, 7, 8) con sus correspondientes métodos constructores y de más métodos asociados a cada clase.

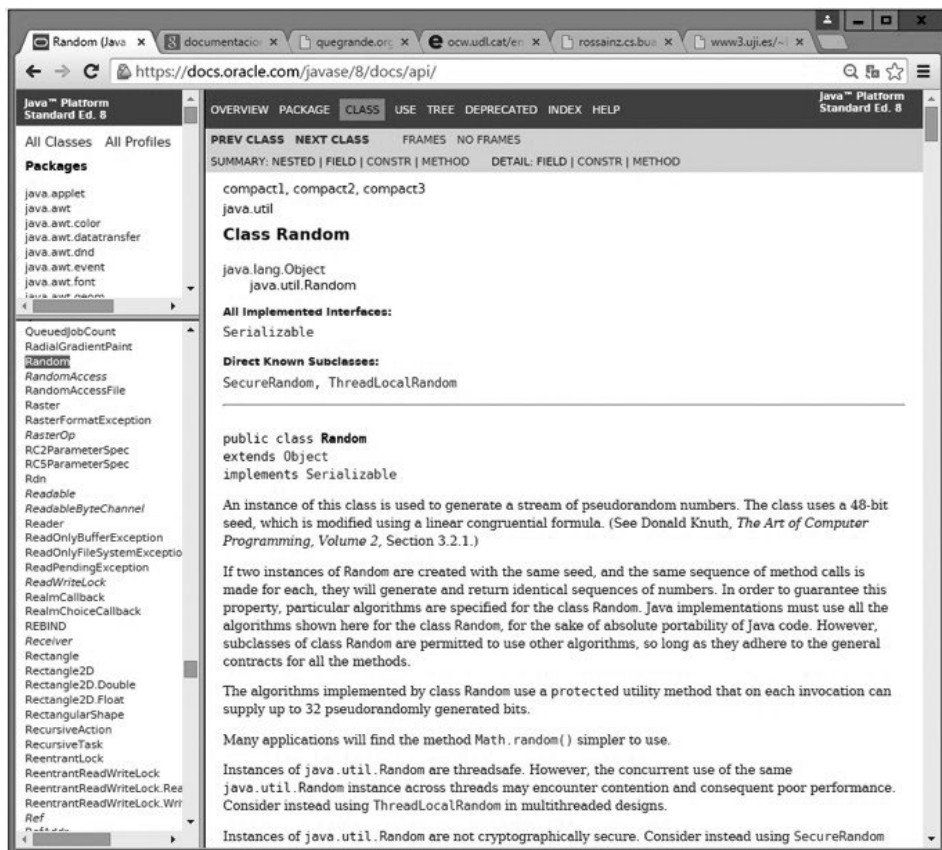
El API esta organizado en paquetes lógicos, donde cada paquete contiene un conjunto de clases relacionadas semánticamente, como por ejemplo:

- » java.io Entrada y Salida de Datos.
- » java.util Estructuras de datos implementadas para que sean usadas.

El API de Java es una herramienta de apoyo de consulta, para los desarrolladores de aplicaciones en Java.

En la figura 33, se presenta en el API de Java 8, la clase Random.

Figura 33. API Java 8.



## 14.6 LECTURAS RECOMENDADAS

- » Revisar la URL <https://docs.oracle.com/javase/8/docs/api/>
- » Convenciones de código para el lenguaje de programación.

## 14.7 PREGUNTAS DE REVISIÓN DE CONCEPTOS

- » ¿Por qué documentar una aplicación en Java?
- » ¿Importancia de la documentación de un proyecto de Java?

## 14.8 EJERCICIOS

1. Realice una descripción breve de cada una de las siguientes etiquetas, dentro las mas utilizadas para documentar una aplicación de software con javadoc.

ETIQUETA	DESCRIPCIÓN
@author	
@deprecated	
@param	
@return	
@see	
@since	
@throws	
@Version	

2. Utilizando javadoc, documente la aplicación de software del capítulo anterior, denominada Centro Médico El Buen Samaritano.

## 14.9 REFERENCIAS BIBLIOGRÁFICAS

<https://docs.oracle.com/javase/8/docs/api/>





**LOS LIBERTADORES**  
FUNDACIÓN UNIVERSITARIA

Bogotá D.C., 2017

Este libro es el resultado de varios años de experiencias donde se retroalimentan carencias y éxitos en busca de buenas prácticas de la programación orientada a objetos (POO); para tal efecto se utiliza Java como lenguaje de programación de alto nivel. Está diseñado para cualquier estudiante que desea hacer inmersión en la programación de computadores, independientemente del área o disciplina del conocimiento, pero eso sí, cambiando el paradigma de la programación estructurada al de los objetos como en la vida real.

Se divide de catorce capítulos de la siguiente manera: relación general de la temática a desarrollar, una introducción al tema central donde se complementa con un mapa mental, el desarrollo de la temática respectiva incluyendo ejemplos. Se recomiendan unas lecturas que posibilitan ampliar el tema, unas preguntas de revisión de conceptos, ejercicios propuestos y finalmente se propone una bibliografía complementaria.



**LOS LIBERTADORES**  
FUNDACIÓN UNIVERSITARIA

ISBN 000-0000-000

