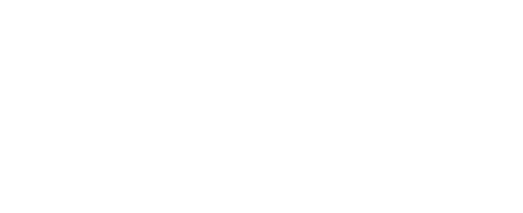
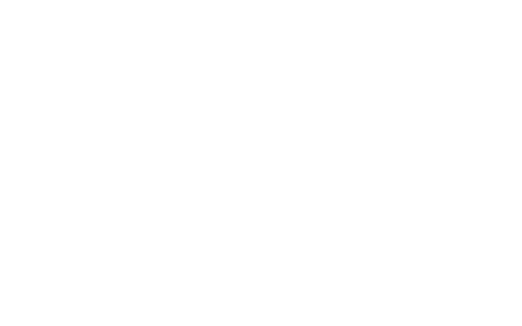
FUNDAMENTOS DE 

PROGRAMACIÓN

ORIENTADA A OBJETOS CON JAVA

Sandra Victoria Hurtado Gil

Contenido

Introducción................................................................................................................................ 5 1. Objetos y Clases................................................................................................................. 6 1.1. Objetos......................................................................................................................... 6 1.2. Clases .......................................................................................................................... 9 1.3. Terminología.............................................................................................................. 10 1.4. Representación.......................................................................................................... 10 1.5. Implementación en Java............................................................................................ 11 1.6. Ejercicio resuelto ....................................................................................................... 13 2. Introducción a Java........................................................................................................... 15 2.1. Contexto..................................................................................................................... 15 2.2. Máquina virtual de Java............................................................................................. 15 2.3. Ambientes de desarrollo............................................................................................ 16 2.4. Estructura de un programa sencillo .......................................................................... 16 2.5. Comentarios .............................................................................................................. 17 2.6. Identificadores ........................................................................................................... 17 2.7. Tipos de datos ........................................................................................................... 18 2.8. Variables.................................................................................................................... 19 2.9. Constantes................................................................................................................. 20 2.10. Expresiones y operadores ..................................................................................... 20 2.11. Expresiones válidas ............................................................................................... 22 2.12. Ejercicio resuelto.................................................................................................... 23 3. Atributos ............................................................................................................................ 24 3.1. Definición ................................................................................................................... 24 3.2. Modelado de atributos ............................................................................................... 24 3.3. Representación.......................................................................................................... 26 3.4. Implementación en Java............................................................................................ 26 3.5. Ejercicio resuelto ....................................................................................................... 27 4. Instrucciones Básicas en Java: Condicionales, Lectura y Escritura................................ 29 4.1. Instrucciones de escritura ......................................................................................... 29 4.2. Instrucciones condicionales ...................................................................................... 31 4.3. Selección simple (*if*) .................................................................................................. 32 4.4. Selección doble (*if-else*)............................................................................................ 33 4.5. Selección múltiple (*switch*)........................................................................................ 35 4.6. Instrucciones de lectura............................................................................................. 37 4.7. Ejercicio resuelto ....................................................................................................... 40 5. Métodos............................................................................................................................. 43 5.1. Definición ................................................................................................................... 43 5.2. Modelado de métodos ............................................................................................... 44

1

5.3. Representación.......................................................................................................... 45 5.4. Implementación en Java............................................................................................ 46 5.5. Ejercicio resuelto ....................................................................................................... 51

6. Referencias y Operador Punto ......................................................................................... 54 6.1. Tipos de variables en Java........................................................................................ 54 6.2. Referencias................................................................................................................ 54 6.3. Uso de los métodos con el operador punto .............................................................. 58 6.4. Valor *null*.................................................................................................................... 60 6.5. Ejercicio resuelto ....................................................................................................... 62

7. Estructuras Repetitivas en Java ....................................................................................... 64 7.1. Definición ................................................................................................................... 64 7.2. Ciclo *while*.................................................................................................................. 64 7.3. Ciclo *do-while* ............................................................................................................ 65 7.4. Ciclo *for*...................................................................................................................... 67 7.5. Instrucción *break* ....................................................................................................... 68 7.6. Ejercicio resuelto ....................................................................................................... 70

8. Referencia *this* .................................................................................................................. 72 8.1. Uso de métodos y atributos propios.......................................................................... 72 8.2. Diferenciar atributos y parámetros ............................................................................ 74 8.3. Ejercicio resuelto ....................................................................................................... 78

9. API de Java....................................................................................................................... 80 9.1. Definición ................................................................................................................... 80 9.2. Sentencia *import*........................................................................................................ 80 9.3. Búsqueda en el API................................................................................................... 81 9.4. Ejercicio resuelto ....................................................................................................... 83

10. Métodos *Set*, *Get* y Constructores................................................................................ 86 10.1. Tipos de métodos .................................................................................................. 86 10.2. Métodos “get”: Acceso o consulta de un atributo.................................................. 86 10.3. Métodos “set”: Modificación de un Atributo........................................................... 89 10.4. Constructor............................................................................................................. 91 10.5. Varios métodos constructores ............................................................................... 94 10.6. Constructor por defecto ......................................................................................... 97 10.7. Ejercicio resuelto.................................................................................................... 98

11. *ArrayList* en Java.........................................................................................................101 11.1. Uso de colecciones..............................................................................................101 11.2. Definición .............................................................................................................101 11.3. Uso de *ArrayList* ..................................................................................................102 11.4. Ciclo mejorado para recorrer listas......................................................................105 11.5. Ejercicio resuelto..................................................................................................106

2

12. Asociaciones entre Clases..........................................................................................109 12.1. Contexto y definición............................................................................................109 12.2. Representación....................................................................................................110 12.3. Implementación en Java......................................................................................111 12.4. Ejercicio resuelto..................................................................................................117

13. Paquetes y Visibilidad .................................................................................................124 13.1. Definición y representación .................................................................................124 13.2. Nombres únicos ...................................................................................................125 13.3. Paquetes en Java ................................................................................................126 13.4. Uso de clases de otros paquetes (importar) .......................................................127 13.5. Visibilidad .............................................................................................................130 13.6. Ejercicio resuelto..................................................................................................134

14. Polimorfismo por Sobrecarga .....................................................................................143 14.1. Definición .............................................................................................................143 14.2. Implementación en Java......................................................................................144 14.3. Ejercicio resuelto..................................................................................................147

15. Herencia ......................................................................................................................150 15.1. Ejemplo introductorio ...........................................................................................150 15.2. Concepto de herencia y representación..............................................................151 15.3. Implementación en Java......................................................................................155 15.4. Palabra reservada *super* .....................................................................................158 15.5. Uso de objetos con herencia ...............................................................................159 15.6. Ejercicio resuelto..................................................................................................161

16. Polimorfismo por Sobrescritura...................................................................................167 16.1. Definición .............................................................................................................167 16.2. Implementación en Java......................................................................................168 16.3. Ejercicio resuelto..................................................................................................171

17. Conversión de variables (*Cast*)...................................................................................177 17.1. Problemática métodos de clases hijas ................................................................177 17.2. *Cast*......................................................................................................................179 17.3. Operador *instanceof* ............................................................................................180 17.4. Ejercicio resuelto..................................................................................................181

18. Atributos y Métodos de Clase (*Static*) ........................................................................186 18.1. Contexto...............................................................................................................186 18.2. Atributos y métodos de clase (estáticos).............................................................186 18.3. Método *main* ........................................................................................................189 18.4. Ejercicio resuelto..................................................................................................189

19. Clases y Métodos Abstractos .....................................................................................191 19.1. Problemática instancias clase padre...................................................................191

3

19.2. Clases abstractas y su representación ...............................................................193 19.3. Métodos abstractos y su representación.............................................................194 19.4. Implementación en Java......................................................................................194 19.5. Métodos y clases final..........................................................................................199 19.6. Ejercicio resuelto..................................................................................................200

20. Interfaces.....................................................................................................................208 20.1. Un nuevo reto.......................................................................................................208 20.2. Definición .............................................................................................................208 20.3. Implementación de una interfaz ..........................................................................209 20.4. Representación....................................................................................................213 20.5. Uso de las interfaces ...........................................................................................214 20.6. Ejercicio resuelto..................................................................................................218

21. Ejemplo........................................................................................................................223 21.1. Enunciado ............................................................................................................223 21.2. Diagrama de clases .............................................................................................224 21.3. Código Java .........................................................................................................225 21.4. Ejecución del programa .......................................................................................233 Bibliografía..............................................................................................................................236

4

**Introducción**

La programación es un área de trabajo muy interesante, no solo por la alta demanda que tiene en la actualidad, sino también por su continua evolución y los constantes retos y satisfacciones que presenta a quienes trabajan en ella. La programación es mucho más que solo conocer un lenguaje técnico, pues incluye –entre otros aspectos– entender muy bien el problema que se desea resolver, elaborar modelos para representar la solución planteada y realizar pruebas para garantizar la adecuada funcionalidad del software que se desarrolle. Estos aspectos han probado ser valiosos en infinidad de campos del conocimiento, por lo que saber de programación es una habilidad valiosa para cualquier profesional, no solo para los ingenieros de sistemas o informáticos.

Existen diferentes paradigmas o estilos de programación, que tienen que ver con la forma en la cual se diseña y se implementa la solución a un problema, como por ejemplo diseñar una solución pensando principalmente en los datos y sus relaciones, o pensando en la secuencia de pasos o actividades que se deben ejecutar. Cada estilo de programación tiene sus ventajas y desventajas, y es importante conocer algunos de ellos para poder aplicarlos adecuadamente.

La programación orientada a objetos – POO – es un paradigma o estilo de programación en el cual el software se construye a partir de un conjunto de objetos que colaboran entre sí (como se detallará más adelante). Este estilo de programación ha sido uno de los más utilizados en los últimos años en el mundo del desarrollo de software, y muchos de los lenguajes de programación más populares, como Java, C#, Phyton, Ruby, entre otros, son orientados a objetos o por lo menos incluyen algunos aspectos de programación orientada a objetos.

En este libro se presentarán los principales conceptos de la programación orientada a objetos y su aplicación en el lenguaje de programación Java, para que así el lector pueda identificar las ventajas de este estilo de programación, y además pueda desarrollar una aplicación de software sencilla con este paradigma.

Para poder aprovechar mejor este contenido, se recomienda tener conocimientos básicos de lógica de programación, tales como tipos de datos, variables, expresiones y estructuras de control (condicionales y ciclos).

5

**1. Objetos y Clases**

En este capítulo se mostrarán las definiciones básicas de la programación orientada a objetos (POO): la definición de objeto y la de clase. También se mostrará la forma de representar las clases de manera gráfica, usando la notación estándar que existe para ello.

A finalizar este capítulo usted debe ser capaz de:

• Diferenciar entre objetos y clases en el contexto de la programación orientada a objetos.

• Identificar objetos y clases a partir de un enunciado dado.

• Representar una clase usando una notación gráfica estándar: UML.

**1.1. Objetos**

Un objeto es una unidad que tiene **características** propias, un **comportamiento** y una **identidad**. La identidad es aquello que hace único a cada objeto y lo diferencia de otros objetos similares.

Observe la siguiente imagen, ¿cuántos objetos se pueden identificar?



*Imagen 1-1 Objetos*

En la imagen se observan cuatro objetos, que son:

• El libro que se encuentra en la parte izquierda. Este libro tiene como **características**, entre otras, que se titula “Ingeniería de Software: Una perspectiva orientada a objetos”, que tiene como autor una persona de apellido Braude y tiene una marca de la dueña Sandra (internamente). Este libro puede ser ojeado y puede ser leído para aumentar los conocimientos de las personas. Esto conforma el **comportamiento** del libro. Aunque 

pueden existir muchos otros libros parecidos, la combinación de sus características lo hace único: ningún otro libro combina ese título, autor y dueño. Esto es lo que le da su **identidad**.

6

• El libro que se encuentra en la parte derecha de la imagen es otro objeto. Trate de identificar sus características y comportamiento. También trate de pensar en alguna característica de un libro que le permita darle identidad para diferenciarlo de otros. 

• El carro de juguete que está en la parte inferior izquierda de la imagen. Algunas de las características de este carro son: que es de color naranja, que tiene dos puestos, dos puertas y cinco llantas (contando la de repuesto). Entre su comportamiento podemos ver que puede servir como adorno, que puede ser movido hacia adelante o hacia atrás y que al moverlo puede hacerlo a diferentes velocidades, tal vez para apostar carreras con otro carro de juguete. Pero surge la pregunta: ¿Cómo diferenciar este carro de otro con las mismas características?



*Imagen 1-2 Objetos Iguales*

Si todas las características de los objetos son iguales se puede usar su posición para diferenciar uno del otro. Por ejemplo, en una tienda usted puede decir: me vende por favor el carro naranja que está a la derecha. Esto permite diferenciarlo de otro que sea igual.

• El carro que está en la parte inferior derecha es el último objeto identificado. Es otro carro, pero con características diferentes, pues 

tiene colores rojo y blanco, tiene ocho puestos, tres puertas y cuatro

llantas. Su comportamiento es similar al del primer carro, pero hay

que reconocer que su velocidad es menor a la del carro naranja y

por lo tanto perdería en una carrera.

• Un aspecto importante de los objetos es que pueden relacionarse entre sí para lograr un objetivo. Por ejemplo, se tiene la siguiente descripción: “El señor Pepe Pérez se acerca al cajero electrónico del banco MuchoDinero que hay en la esquina de la calle 51. Al llegar pasa su tarjeta débito y solicita el saldo de su cuenta de ahorros; obtiene un recibo y luego se retira. Posteriormente se acerca a este mismo cajero la señora Lola López, quien también utiliza su tarjeta y luego retira $200.000 de su cuenta corriente”.

En la anterior descripción se pueden identificar los siguientes objetos:

• El cajero electrónico de MuchoDinero que está en la calle 51

• El señor Pepe Pérez

• La señora Lola López

• La cuenta del señor Pepe Pérez

• La cuenta de la señora Lola López

Cada uno de estos objetos tiene características propias, un comportamiento y una identidad. Se puede decir, por ejemplo, que la cuenta del señor Pepe Pérez es una cuenta de ahorros, tiene un saldo y un número que está en la tarjeta (características). Además, se diferencia de la cuenta de la señora Lola López y de otras cuentas porque tiene un número único, lo que

7

le permite tener identidad. Esta cuenta presta servicios como: consultar su saldo, hacer retiros o consignaciones, lo cual es su comportamiento.

Además, estos objetos interactúan entre sí para lograr la funcionalidad que se desea.

consulta saldo 

consultarSaldo

recibosaldo

Pepe Pérez Cajero MuchoDinero calle 51

Cuenta de ahorros de Pepe

*Imagen 1-3 Objetos interactuando para lograr una funcionalidad*

En la imagen puede verse que Pepe Pérez le envía un mensaje al cajero para indicar que desea consultar el saldo y el cajero a su vez le envía un mensaje a la cuenta para saber el saldo y luego mostrar esta información.

Es importante aclarar que, en el contexto de la POO, un objeto puede ser una cosa, pero también puede ser un animal, una persona, un objeto abstracto (como la cuenta de ahorros del ejemplo anterior) o una entidad virtual (como una página web). Lo importante es que para cada objeto se puedan definir sus características, comportamientos e identidad.

***Ejercicio 1-1***

Determine cuáles de los siguientes elementos son objetos y cuáles no: • El carro Mazda negro de placas NAF-234

• El número de goles que hizo la selección Colombia en el último partido • El lapicero azul de Paola

• El color rojo de la bandera de Colombia que está en el estadio Palograndre

***Ejercicio 1-2***

Identifique los objetos que se presentan en el siguiente enunciado, definiendo para cada uno sus características y comportamiento. Trate de definir también para cada objeto qué es lo que lo diferencia de los demás (su identidad).

En Manizales se celebró el año pasado el 33 Festival Internacional de Teatro, que contó con la participación de diferentes agrupaciones, para entretención de propios y visitantes. Dentro de las obras más alabadas se encuentra “Utsushi”, que es una obra de la agrupación japonesa “Sankai Juku” que representa sentimientos profundos de las personas usando solo la expresión corporal. El grupo Sankai Juku, que fue creado en 1975, está dirigido por el bailarín y coreógrafo Ushio Amagatsu. Su obra, de gran renombre internacional, fue seleccionada para ser el cierre del Festival.

Aunque se presentaron varias obras de grupos asiáticos, también se encuentran otras de países Latinoamericanos, como la obra “Maluco” del grupo “La Cuarta Colectivo Artístico” de Uruguay, que es una adaptación de una novela histórica pero reinterpretada con imaginación y humor. El grupo La Cuarta fue creado hace 13 años por Laura Pouso y Gustavo Zidan, y sus producciones han tenido gran impacto regional e internacional.

8

**1.2. Clases**

Cuando se modela un sistema usando orientación a objetos no es necesario detallar todos los objetos que hacen parte de este, sino que se identifican los diferentes grupos de objetos similares que hay. Los objetos se clasifican buscando los que representen lo mismo y compartan características y comportamiento. Esto se conoce como una clase.

Es decir, una clase es un **tipo o grupo de objetos** donde se definen las características y comportamientos que tienen en común.

Una clase también se define como el **patrón, molde o plantilla con el cual se crean objetos de un mismo tipo** para usarlos en un programa. No es posible tener objetos si no se ha definido previamente la clase para poder crear estos objetos.

En el caso de la imagen presentada en la sección anterior se identifican dos clases:

• La clase Libro, que define que los 

libros tienen como características:

título, autor y dueño. Los libros

pueden ser ojeados y leídos por las

personas.

• La clase Carro de Juguete. Esta clase

define que los carros tendrán color,

número de puestos, número de

puertas y número de llantas. También

define que los carros podrán ser

movidos hacia adelante y hacia atrás,

o usados como adornos.

Por lo general las clases aparecen como sustantivos de los cuales se habla en alguna descripción. En el ejemplo de Pepe y Lola en el cajero, las clases son: • Persona (o Cliente del banco)

• Cuenta bancaria

• Cajero electrónico

Para que Pepe Pérez pueda tener una cuenta de ahorros es necesario que primero el banco haya definido las características y los comportamientos que tendrán las cuentas. Es decir, **primero se define la clase y luego se crean los objetos** de esa clase.

Cada vez que se crea un objeto lo que se hace es darles **valores concretos** a las características, y así es posible tener objetos diferentes a partir de una misma clase.

Por ejemplo, se puede decir que todas las cuentas tienen como características: un número, un saldo y un tipo (si es de ahorro o corriente). Además, las cuentas permiten consultar el saldo, retirar y consignar. Cada objeto Cuenta tiene un valor diferente para estas características. Por ejemplo, el número de la cuenta de Pepe Pérez es 1234-56, tiene un saldo de $150.000 y es de ahorros, mientras que la cuenta de Lola López es 1235-87, tiene un saldo de $300.000 y es una cuenta corriente. En ambas cuentas se puede consultar el saldo, consignar y retirar, aunque cada una teniendo en cuenta sus valores. Por ejemplo,

9

Pepe solo puede retirar máximo $150.000 mientras que Lola puede retirar más dinero de su cuenta.

***Ejercicio 1-3***

Cuenta de Pepe número = 1234-56 saldo = $150.000 tipo = ahorros 

Cuenta de Lola número = 1235-87 saldo = $300.000 tipo = corriente

Enumere las clases que identifica en el siguiente enunciado (que es el mismo de la sección anterior):

En Manizales se celebró el año pasado el 33 Festival Internacional de Teatro, que contó con la participación de diferentes agrupaciones, para entretención de propios y visitantes. Dentro de las obras más alabadas se encuentra “Utsushi”, que es una obra de la agrupación japonesa “Sankai Juku” que representa sentimientos profundos de las personas usando solo la expresión corporal. El grupo Sankai Juku, que fue creado en 1975, está dirigido por el bailarín y coreógrafo Ushio Amagatsu. Su obra, de gran renombre internacional, fue seleccionada para ser el cierre del Festival.

Aunque se presentaron varias obras de grupos asiáticos, también se encuentran otras de países Latinoamericanos, como la obra “Maluco” del grupo “La Cuarta Colectivo Artístico” de Uruguay, que es una adaptación de una novela histórica pero reinterpretada con imaginación y humor. El grupo La Cuarta fue creado hace 13 años por Laura Pouso y Gustavo Zidan, y sus producciones han tenido gran impacto regional e internacional.

**1.3. Terminología**

En la programación orientados a objetos, los términos que se usan para definir los elementos que constituyen los objetos y las clases son:

• Los objetos son conocidos como **instancias** de las clases.

• A las características se les llama **atributos** (en inglés a veces se pueden encontrar como “*fields*” o campos). También se conocen como variables de instancia. • Los comportamientos se definen a través de un conjunto de **métodos** (también se conocen como operaciones).

**1.4. Representación**

La forma de representar una clase de manera gráfica es mediante un rectángulo dividido en tres partes. En la parte superior va el nombre de la clase, en el recuadro del medio van los atributos y en la parte inferior van los métodos.

10

Nombre

Atributos

Métodos

Esta notación se ha convertido en un estándar que conocen y aplican todos los desarrolladores de software, y que está definida por un lenguaje de modelado llamado **UML** (*Unified Modeling Language*).

Por ejemplo, para los carros de juguete el diagrama es:

**Libro**

título

autor

dueño

leer()

ojear()

El diagrama para una cuenta bancaria es:

**Cuenta**

número

saldo

tipo

consultarSaldo()

retirar()

consignar()

**1.5. Implementación en Java**

Para definir una clase en Java se utiliza la siguiente sintaxis:

| /\*\*  \* Descripción de la clase  \* @author nombreAutor  \* @version númeroVersión  \*/  **public class** NombreClase  {    } |
| --- |

Se tienen los siguientes elementos en esta definición:

11

• Descripción (comentario): Aquí se explica en qué consiste la clase. También se define el autor y el número de versión.

o Para especificar el autor se usa *@author*, seguido del nombre.

o Para especificar la versión se usa *@version*, seguido del número de versión, por ejemplo 1.0, 1.7, 2.5, etc.

• Las palabras reservadas *public class*: La palabra *public* es un modificador para indicar que la clase es pública, es decir, puede ser usada por otras clases; y la palabra *class* marca el inicio de la definición de la clase.

• El nombre de la clase. La primera letra debe ir en mayúscula.

• El bloque de código de la clase: Todo el código de la clase, es decir, los atributos y los métodos, estarán encerrados en un bloque de código que está enmarcado por llaves.

Por ejemplo, el código para definir la clase Libro (todavía sin sus atributos y métodos), es:

| /\*\*  \* Representa un libro de la biblioteca de una persona, de manera que \* la persona pueda saber los datos básicos de cada libro que posee. \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class **Libro**  {    } |
| --- |

Nota: La forma de implementar los atributos y los métodos de las clases en Java se presentará en los siguientes capítulos.

En otro ejemplo, la estructura básica para la clase Cuenta es:

| /\*\*  \* Una cuenta bancaria, es decir, un registro en  \* una entidad bancaria que tiene un saldo (cantidad de dinero) \* que se puede modificar mediante retiros o consignaciones. \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class Cuenta  {  } |
| --- |

***Ejercicio 1-4***

a) Elabore el diagrama de clases, indicando cuáles serían los atributos y los métodos para: • Un jugador de fútbol

• Un teléfono celular

• Una película

b) Escriba la definición de las clases anteriores en Java.

12

**1.6. Ejercicio resuelto**

La abuela Pata prepara deliciosos dulces en su cocina, y los nietos Hugo, Paco y Luis están siempre muy pendientes porque saben que cuando están recién hechos son deliciosos. La abuelita hace dulces de mora, guayaba y otros deliciosos sabores, pero hoy ha preparado un dulce de brevas que le ha quedado muy rico. Ella envasa el dulce en frascos de 200 gramos. En total le salen 5 frascos que va marcando con la fecha a medida que los envasa.

En cuanto se va de la cocina los tres nietos entran y consumen un poco de algunos de los frascos antes de que se dé cuenta. Pero la abuelita no es tonta, y al regresar revisa la cantidad de cada frasco y los que tienen menos de 200 gramos los aparta como regalo para sus nietos, pues sabe que les encantan esos dulces. Los otros frascos los guarda para darlos a otros miembros de la familia Pata.

En el enunciado anterior se identifican los siguientes objetos:

• La abuela Pata • El nieto Hugo • El nieto Paco • El nieto Luis

• El primer frasco de dulce • El segundo frasco de dulce • El tercer frasco de dulce • El cuarto frasco de dulce • El quinto frasco de dulce

Las clases correspondientes son dos, una para un miembro de la familia Pato y otra para los frascos de dulce. Cada frasco tiene como atributos su sabor, la cantidad (que al comienzo es 200 gramos) y la fecha. Uno de los métodos que debe prestar cada frasco de dulce es “envasar dulce” para que la abuela pueda guardar el dulce que elabora. Otro método importante es “sacar dulce”, para que todos puedan disfrutar los dulces de la abuela. Además, para que la abuela pueda saber que pasó, cada frasco de dulce debe tener un método que permita consultar su cantidad. El diagrama de clases correspondiente es:

**MiembroFamilia**

nombre

parentesco

prepararDulce() comerDulce()

**FrascoDeDulce**

sabor

cantidad

fecha

envasarDulce() sacarDulce()

consultarCantidad()

El código en Java para las clases (sin atributos y métodos, por el momento), es:

| /\*\*  \* Persona que hace parte de una familia, por ejemplo,  \* de la familia del Pato Donald.  \* @author Juan David Correa  \* @version 0.5  \*/  public class MiembroFamilia  {  } |
| --- |

13

| /\*\*  \* Un frasco con delicioso dulce, elaborado por la abuela pata. \* Por ejemplo, un frasco de dulce de guayaba.  \* @author Juan David Correa  \* @version 0.5  \*/  public class FrascoDeDulce  {  } |
| --- |

14

**2. Introducción a Java**

**Juan David Correa Granada**

En este capítulo se hace una breve presentación del lenguaje de programación Java, incluyendo la forma de definir variables de tipos primitivos y realizar operaciones básicas con ellas.

Al finalizar este capítulo usted debe ser capaz de:

• Definir variables de tipos primitivos en Java.

• Escribir expresiones matemáticas sencillas en Java.

**2.1. Contexto**

Java es un lenguaje de programación orientado a objetos, desarrollado por *Sun Microsystems*, y que hoy en día propiedad de *Oracle*. Su primera versión apareció en 1995, y se popularizó inicialmente como un lenguaje para internet, que podía correr en muchas plataformas.

Sin embargo, posteriores versiones de este lenguaje lo han convertido en un lenguaje de programación para aplicaciones empresariales, gracias a muchas de sus ventajas (dadas principalmente por la orientación a objetos) y el conjunto de librerías que ofrecen diferentes servicios.

**2.2. Máquina virtual de Java**

Los programas en Java no son compilados directamente al lenguaje de máquina, sino a un lenguaje intermedio llamado “bytecode”. Para poder ejecutar estos programas es necesario tener instalada la máquina virtual de java (MVJ o JVM – *Java Virtual Machine*). Existen máquinas virtuales para muchos sistemas operativos, y esto es lo que permite que los programas en Java sean portables.

Archivo **.**java compilar Archivo **.**class

ejecutar

Máquina Virtual de Java

Linux Window s

MacOs**…**

*Imagen 2-1 Portabilidad gracias a la máquina virtual de Java*

15

La máquina virtual, además de “convertir” las instrucciones que están en *bytecode* a las instrucciones acordes a cada sistema operativo, lleva a cabo otras funciones, como:

• Aislar los programas para que no puedan tener acceso directo al hardware del equipo. • Ejecutar funciones adicionales que facilitan la programación, por ejemplo, eliminar los elementos que han dejado de ser referenciados.

• Comprobación de algunos errores al momento de ejecución, antes de ejecutar estas operaciones en el sistema operativo.

Una de las desventajas que presenta este esquema es que los programas Java pueden ser un poco más lentos que los programas que se ejecutan directamente sobre el sistema operativo.

**2.3. Ambientes de desarrollo**

Aunque es posible desarrollar aplicaciones Java utilizando sólo las herramientas básicas, generalmente el desarrollo se hace utilizando ambientes integrados de desarrollo (IDE), que incrementan la productividad del programador.

Algunos IDE (entre una gran cantidad que existen actualmente):

- Eclipse (http://www.eclipse.org/downloads/index.php)

- NetBeans (http://www.netbeans.info/downloads/index.php)

- JBuilder (http://www.borland.com/jbuilder/)

- JDeveloper (http://www.oracle.com/technology/products/jdev/index.html)

**2.4. Estructura de un programa sencillo**

Todo programa Java está formado por clases, de las cuales por lo general hay una que se “ejecuta” (para iniciar el programa), y las demás conforman toda la estructura con la funcionalidad deseada.

Las clases que se pueden ejecutar son aquellas que tiene el método ***main***. Un programa muy sencillo, por lo tanto, está formado por una clase que tiene un método *main*, con el código que se desea ejecutar.

Para organizar las instrucciones de un programa, éstas se enmarcan en **bloques de código**. Un bloque de código está delimitado por llaves: { }, la llave que abre al comienzo del bloque, y la llave que cierra, al final. Para enmarcar todo lo que está en la clase o todo lo que está en un método se usan bloques de código con las llaves.

El “esqueleto” para un programa muy simple es:

16

| /\*\*  \* Comentario de la clase  \*/  public class NombreClase  **{**  /\*\*  \* Comentario del método  \*/  public static void main(String[] args)  **{**  // Aquí va el código del método  **}**  **}** |
| --- |

**2.5. Comentarios**

Para insertar comentarios en un programa se utiliza:

| // Para comentarios de una línea |
| --- |

| /\*  Para comentarios de varias líneas  \*/ |
| --- |

| /\*\*  Para comentarios de varias líneas, que deseen ser incluidos en la documentación que se genera automáticamente  \*/ |
| --- |

**2.6. Identificadores**

Los identificadores son los nombres que se colocan a las clases, los métodos y las variables, entre otros. Estos nombres deben ser significativos, es decir, que al leerlos se entienda lo que significan. Por ejemplo, un nombre significativo es: “colorPantalla” y no “clrPtlla”, o “numeroPersonas” y no “num”.

Para los identificadores se deben seguir las siguientes recomendaciones: • Puede contener letras, números o el símbolo de subrayado, pero sólo pueden comenzar con una letra o el símbolo de subrayado, no con números.

• No deben contener espacios en blanco.

• Java es sensible a mayúsculas y minúsculas, es decir una variable llamada “primera” será diferente a otra llamada “Primera”.

• No pueden ser palabras propias del lenguaje de programación –o palabras reservadas– como “class” o “for”.

También hay unas convenciones para los identificadores en los programas en Java, entre las cuales se tienen las siguientes:

17

• Los nombres de las clases empiezan en mayúscula. Las demás letras son minúsculas, excepto si es el comienzo de otra palabra. Ejemplo: Casa, TeatroPrincipal • Los nombres de los métodos, los atributos y las variables empiezan en minúscula. Las demás letras son minúsculas, excepto si es el comienzo de otra palabra. Ejemplo: contador, cuentaVeces

• Las constantes deben ir en mayúscula sostenida, y cada palabra se separa de la anterior por el símbolo de subrayado Ejemplo: IVA, VALOR\_TOTAL

***Ejercicio 2-1***

Para cada uno de los siguientes nombres de variables, indique si es válido y si sigue las convenciones:

• 4oPuesto

• incremento%

• estatura promedio

• valorNeto

• CapitalPais

**2.7. Tipos de datos**

En Java se habla de dos grandes grupos de tipos de datos, **los primitivos y los referenciados**. Los tipos de datos primitivos se refieren a datos que son comunes a la mayoría de los sistemas, y que se manejan como *valores,* y son los que se van a tratar en esta parte. Los tipos referenciados agrupan las clases, interfaces y los arreglos, y tienen como características que pueden ser definidos por los programadores y que no se manejan como valores sino como objetos, pero esto se estudiará en detalle más adelante.

Los tipos de datos primitivos, o básicos, en Java, son:

| **Nombre** | **Descripción** | **Rango (o valores)** |
| --- | --- | --- |
| boolean | Valores lógicos o valores de verdad | true  false |
| byte | Números enteros (valores pequeños) | -128 a 127 |
| short | Números enteros (valores medianos) | -32.768 a 32.767 |
| int | Números enteros | -231 a 231-1 |
| long | Números enteros (valores grandes) | -263 a 263-1 |
| char | Caracteres (aunque Java también lo considera numérico). Cada carácter se representa entre comillas simples, por ejemplo: ‘f’, ‘%’, ‘8’ | Utiliza el formato UNICODE para la representación de cada carácter  (http://www.unicode.org/). |
| float | Para números con decimales, con precisión simple (de 6 a 7 dígitos decimales significativos) | aproximadamente  ±3.40282347x1038 |
| double | Para números con decimales, con precisión doble (15 dígitos decimales significativos) | aproximadamente  ±1. 79769313486231570x10308 |

Existe otro tipo de dato muy utilizado, para las cadenas de caracteres, que es **String**. Aunque este es un tipo referenciado, se utiliza de manera similar a los tipos de datos básicos.

18

Las cadenas de caracteres (*String*) se representan con comillas dobles, por ejemplo: “Esta es una cadena”

“texto”

“7652”

***Ejercicio 2-2***

Determine el tipo de dato de Java más adecuado para:

• La edad de una persona

• La nota final de un estudiante en una asignatura

• La marca de un vehículo

• El valor del dólar, en pesos

**2.8. Variables**

Es la unidad básica de almacenamiento en Java. Toda variable debe ser de un tipo definido, y debe ser declarada antes de ser utilizada. La sintaxis para declarar las variables es:

| tipo nombreVariable; |
| --- |

También se le puede asignar un valor inicial:

| tipo nombreVariable = valorInicial; |
| --- |

Ejemplos:

| *// Variable llamada valor, de tipo entero, sin un valor inicial:* **int valor;**  *// Variable precio, de tipo double, con valor inicial 35.6* **double precio = 35.6;** |
| --- |

Se pueden declarar varias variables del mismo tipo en la misma línea, separadas por comas. Por ejemplo:

| */\* Declara dos variables de tipo boolean:*  *una llamada resultadoUno, sin valor inicial,*  *y otra llamada resultadoDos, con valor inicial true.*  *\*/*  boolean resultadoUno, resultadoDos=true; |
| --- |

Las variables sólo son conocidas dentro del bloque de código en el cual son definidas, lo cual se denomina alcance de la variable.

Ejemplo:

19

| /\*\*  \* Ejemplo de alcance de variables  \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class Alcance  {  public static void main (String args [])  {  int valorUno = 10;  if (valorUno == 10)  {  int valorDos = 20;  System.out.print (valorUno + " " + valorDos);  }  **valorDos = 100; // error, aquí no se conoce esta variable** }  } |
| --- |

**2.9. Constantes**

El valor asignado a las constantes no puede ser modificado en el transcurso del programa.

Para declarar constantes se utiliza la misma sintaxis que para las variables, pero adicionando al comienzo la palabra reservada *final*, ejemplo:

| final double IVA = 15.3; |
| --- |

***Ejercicio 2-3***

Escriba cómo declarar, en Java, valor de la raíz cuadrada del número 2 como una constante (el valor es 1.41421356237309).

**2.10. Expresiones y operadores**

Una expresión es una combinación de operadores y operandos, que dan como resultado un valor determinado. Los operadores en Java pueden ser variables, valores literales, constantes, llamadas a métodos u otras expresiones. Ejemplos de expresiones:

| (temperatura / 2) + 3  (edad > 18) & (edad < 60) |
| --- |

A continuación, se presentan los principales operadores que se pueden usar en Java para escribir expresiones.

20

• **Asignación:**

Para asignar un valor a una variable se usa el operando =, por ejemplo:

| int cantidadAlumnos;  cantidadAlumnos = 34; |
| --- |

• **Aritméticos:**

Cuando se realiza una expresión que incluye operaciones aritméticas el resultado será del tipo de dato con mayor precisión involucrado en la expresión. Por ejemplo, si todos los operandos son enteros el resultado será entero (*¡incluso en una división!*), pero si algún operando es un número real con decimales (como un *double*) el resultado será también un número con decimales.

Operaciones básicas entre dos operandos:

+ Suma

- Resta

\* Multiplicación

/ División

% Módulo (residuo de la división)

Por ejemplo:

| int numeroImpar = 6 - 1;  int valorEntero = numeroImpar / 2; *// En valorEntero queda 2* double valorDecimal = numeroImpar / 2.0; *// En valorDecimal queda 2.5* int resultado = valorEntero \* 10 + (7 % 3); |
| --- |

Estas operaciones se pueden combinar con la asignación si uno de los operandos es el mismo donde se desea guardar el resultado. Es decir, si se tiene una operación de la forma a = a + b, se puede cambiar por: a+=b. Ejemplo:

| double gramos = 34.5;  gramos\*=3; *// Equivale a gramos = gramos \* 3;* |
| --- |

Operaciones que aplican sobre un operando:

++ Incrementa en uno el valor de la variable.

-- Decrementa en uno el valor de la variable.

Ejemplo:

| short peso = 60;  peso++; *// peso queda con valor 61* |
| --- |

***Ejercicio 2-4***

Escriba la siguiente expresión matemática como expresión válida de Java:

21

7.5�� − (12.3 + ��)

6��

• **Relacionales:**

Comparan dos operandos, y el resultado es un valor booleano.

> Mayor que

< Menor que

== Igual

!= Diferente

>= Mayor o igual

<= Menor o igual

Ejemplo:

| int edad = 20;  boolean mayorEdad = edad > 18; *//true*  boolean igualEdad = edad == 21; *// false* |
| --- |

• **Lógicos:**

Permiten realizar operaciones entre operandos de tipo booleano.

! Es la negación (es un operador unitario)

& AND

&& AND cortocircuito (si el primer operador es false no evalúa el segundo) | OR

|| OR cortocircuito (si el primer operador es true no evalúa el segundo) ^ OR excluyente

Ejemplos:

| boolean ganoCarrera = false;  boolean juegaLimpio = true;  boolean recibePremio = ganoCarrera | juegaLimpio; *//true* boolean doblePremio = ganoCarrera && juegaLimpio; *//false* |
| --- |

***Ejercicio 2-5***

Escriba la siguiente expresión lógica usando sintaxis de Java:

“Iremos de viaje si no estoy enfermo y si sales a vacaciones”.

**2.11. Expresiones válidas**

Para que una expresión sea válida debe combinar operadores y operandos que sean compatibles y siguiendo la sintaxis de los operadores. Por ejemplo, si se desea usar el

22

operador suma, deben tenerse dos operandos, como por ejemplo en 5+3. Si falta uno de los dos sería una expresión no válida:

| int suma = 8 + ; ***// expresión no válida: falta un operador*** |
| --- |

Tampoco es válido asignar valores a variables incompatibles. Por ejemplo:

| boolean pudoCalcular = 2 \* 6; ***// tipos no compatibles: // la expresión es int y la variable es boolean*** |
| --- |

***Ejercicio 2-6***

Defina si la siguiente expresión es válida o no. Justifique su respuesta:

| int resultado = 8.5 – 1.5; |
| --- |

**2.12. Ejercicio resuelto**

Se desea calcular el valor de una venta, que se calcula multiplicando el número de productos que lleva la persona, por el valor del producto. El cliente ha llevado 3 productos, y el producto tiene un valor de $20.000.

El programa en Java para hacer esto se muestra a continuación.

| /\*\*  \* Determina el valor de una venta, basado en la cantidad de productos \* y el valor del producto.  \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class Venta  {  public static void main (String args[])  {  int cantidadProductos = 3;  int valorProducto = 20000;  double valorVenta = cantidadProductos \* valorProducto;  }  } |
| --- |

23

**3. Atributos**

En este capítulo se extiende la definición de atributo y se muestra su representación gráfica en UML. También se explica la forma de implementar los atributos en el lenguaje de programación Java.

Al finalizar este capítulo usted debe ser capaz de:

• Identificar atributos de las clases en un enunciado dado.

• Representar los atributos de una clase usando una notación gráfica estándar. • Implementar una clase en Java incluyendo sus atributos.

**3.1. Definición**

En el capítulo anterior se estableció que los objetos tienen características, comportamiento e identidad. Los objetos se modelan utilizando clases que se constituyen en los elementos base de la programación orientada a objetos.

Un atributo (o variable de instancia) es la definición de una característica de los objetos de una clase. Es decir, **un atributo representa una característica** que tendrán los objetos. Por ejemplo, algunos atributos de una clase carro pueden ser el color, la placa, la marca, etc.

Para definir un atributo en una clase son indispensables dos elementos:

• El nombre: Es la denominación dada a cada atributo, normalmente es un sustantivo. Debe ser un nombre significativo.

• El tipo*:* Cada atributo debe ser de un tipo de dato definido. Estos tipos de datos dependen del lenguaje de programación seleccionado. Por ejemplo, en Java se tienen tipos de datos como *int* (número entero), *double* (número real), *boolean* (valor de verdad o booleano), *String* (cadena de caracteres), entre otros.

**3.2. Modelado de atributos**

Retomando el ejemplo del capítulo anterior: “El señor Pepe Pérez se acerca al cajero electrónico del banco MuchoDinero que hay en la esquina de la calle 51. Al llegar pasa su tarjeta débito y solicita el saldo de su cuenta de ahorros; obtiene un recibo y luego se retira. Posteriormente se acerca a este mismo cajero la señora Lola López, quien también utiliza su tarjeta y luego retira $200.000 de su cuenta corriente”. Se identificaron 3 clases:

• Persona (o Cliente del Banco)

• Cuenta bancaria

• Cajero electrónico

Para establecer los atributos de las clases se tienen en cuenta todas las fuentes de información disponibles en el proyecto, ya sea un enunciado, entrevista, documento, diagrama, cuestionario u otro que describa lo que el sistema debe permitir. El sentido común y la experiencia también son factores para tener en cuenta a la hora de establecer los atributos de una clase.

Al definir la clase Persona, por ejemplo, es posible identificar muchas características: *número de identificación*, *nombres*, *apellidos*, *edad*, *sexo*, *grupo sanguíneo*, *etnia* y *número*

24

*de hijos*. Aunque existen muchas otras características que podrían ser enunciadas, al modelar una clase el analista o desarrollador debe identificar aquellas que son **relevantes** para el **contexto** en el que se construye el sistema. Estas características relevantes serán los atributos de la clase.

Para el enunciado anterior el contexto está relacionado con el retiro de dinero de un banco en medios electrónicos, de allí que los atributos *identificación*, *nombres y apellidos* sean muy importantes, mientras que la *edad*, *grupo sanguíneo*, *etnia* y *número de hijos* no tengan la misma relevancia. Estas últimas serían muy importantes en otro contexto, por ejemplo, para un sistema médico.

Una vez identificados los atributos para modelar, se establece el tipo de dato más adecuado. Por ejemplo, para los atributos de Persona:

| **Nombre del atributo** | **Tipo de dato (en Java)** |
| --- | --- |
| Identificación | *String* o *int* |
| Nombres | *String* |
| Apellidos | *String* |

***Ejercicio 3-1***

A partir del siguiente enunciado, identifique los atributos y sus respectivos tipos de datos, para las clases CarroRecolector y Aseador:

La empresa de Aseo “Mundo Limpio” tiene varios carros de recolección de basura y también varios aseadores que se encargan de la limpieza de la ciudad. Los carros de recolección de basura pueden ser de diferentes tipos: están los carros generales, de 15 toneladas, donde se recoge la basura de los hogares; los carros industriales, de 20 toneladas, donde se recoge la basura de las industrias, y los carros especiales, de 5 toneladas, donde se recolectan los residuos peligrosos (como los residuos hospitalarios o los electrónicos).

Don Pepe, uno de los aseadores de la empresa, nos cuenta cómo últimamente se ha incrementado el uso de los carros especiales:

“*Si, señores, imagínese que debe ser por la cultura de las personas, por toda la publicidad sobre la importancia de conservar el planeta y esas cosas, que ahora hay cada vez más empresas que nos solicitan el servicio de recolección de residuos peligrosos. Quién iba a pensar, por ejemplo, que en un colegio lo necesitaran, pero resulta que sí, porque en los laboratorios de química y hasta los mismos señores del aseo manejan este tipo de residuos. Entonces la empresa ha comprado 5 carros más para este tipo de recolección, y nos parece muy bien porque así tenemos una ciudad más limpia y nosotros también más trabajito.”*

Don Pepe lleva 5 años trabajando en la empresa y ha recibido varias capacitaciones para manejar los residuos peligrosos, de lo cual se siente muy orgulloso. Los directivos de la compañía nos confirman esto, y nos dicen también que cada carro de recolección tiene todas las medidas de seguridad y sanitarias requeridas. La vida útil promedio de estos carros es de 10 años, y los que tengan más de este tiempo de uso son vendidos como chatarra.

25

**3.3. Representación**

Recordemos que las clases se representan con un rectángulo dividido en tres partes. La parte intermedia es el lugar dónde se definen los atributos. La forma de representar cada atributo en el diagrama es:

| nombre: tipo |
| --- |

Por ejemplo, para un cliente del banco, el diagrama (mostrando solo los atributos) queda:

**Cliente**

cédula: String

nombre: String

Para una cuenta bancaria el diagrama es:

**Cuenta**

número: String

saldo: double

tipo: String

consultarSaldo()

retirar()

consignar()

De acuerdo con el diagrama anterior, cada objeto que se cree de la clase Cuenta tendrá los siguientes atributos, a los que podrá darle valores:

• El número, que es de tipo *String* (puede tener caracteres como el guion). • El saldo, que es de tipo *double.*

• El tipo, que es de tipo *String* (para indicar si es corriente o de ahorros).

**3.4. Implementación en Java**

Los atributos se definen como variables, es decir:

| tipoDeDato nombre; |
| --- |

Se recomienda que el nombre del atributo esté en minúscula, con mayúscula intermedia cuando está formado por varias palabras. Los atributos deben ir **DENTRO** del bloque de código de la clase.

Por ejemplo, el código de la clase Cliente, con sus atributos es:

26

| /\*\*  \* Una persona que tiene una cuenta en el banco, y por lo tanto \* se considera cliente del banco.  \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class Cliente  {  String cedula;  String nombre;  } |
| --- |

También se muestra el código de la clase Cuenta, incluyendo sus atributos:

| /\*\*  \* Una cuenta bancaria, es decir, un registro en  \* una entidad bancaria que tiene un saldo (cantidad de dinero) \* que se puede modificar mediante retiros o consignaciones. \* @author Sandra V. Hurtado  \* @version 1.2  \*/  public class Cuenta  {  String numero;  double saldo;  String tipo;  } |
| --- |

***Ejercicio 3-2***

Escriba el código Java para crear las clases Revista (en un almacén) y Paciente (en un consultorio médico), que se muestran a continuación:

**Revista**

título: String

páginas: int

precio: double

enOferta: boolean

**3.5. Ejercicio resuelto**

**Paciente**

nombre: String edad: int

sexo: char

peso: double

Del ejercicio del capítulo anterior (el de la abuela Pata que hace deliciosos dulces), se identificaron dos clases: MiembroFamilia y FrascoDeDulce. Al establecer el tipo de dato adecuado para cada atributo, el diagrama queda:

27

**MiembroFamilia**

nombre: String parentesco: String

prepararDulce() comerDulce()

**FrascoDeDulce**

sabor: String

cantidad: int

fecha: String

envasarDulce() sacarDulce()

consultarCantidad()

El código en Java para estas clases – sin tener en cuenta los métodos – sería:

| /\*\*  \* Persona que hace parte de una familia, por ejemplo,  \* de la familia del Pato Donald.  \* @author Juan David Correa  \* @version 1.0  \*/  public class MiembroFamilia  {  **String nombre;**  **String parentesco;**  } |
| --- |

| /\*\*  \* Un frasco con delicioso dulce, elaborado por la abuela pata. \* Por ejemplo, un frasco de dulce de guayaba.  \* @author Juan David Correa  \* @version 1.0  \*/  public class FrascoDeDulce  {  **String sabor;**  **int cantidad;**  **String fecha;**  } |
| --- |

28

**4. Instrucciones Básicas en Java: Condicionales, Lectura y Escritura**

En este capítulo se continúa con la introducción al lenguaje de programación Java, presentando las estructuras condicionales y unas instrucciones básicas que permiten solicitar y mostrar datos al usuario, para permitir construir programas un poco más elaborados.

Al finalizar este capítulo usted debe ser capaz de:

• Escribir un programa sencillo en Java, donde se puedan pedir y mostrar datos al usuario (enteros, reales y cadenas de caracteres).

• Describir tres instrucciones condicionales en Java: *if*, if-*else* y *switch*. • Incluir estructuras condicionales en un método en Java.

**4.1. Instrucciones de escritura**

Haciendo uso de expresiones es posible escribir programas en Java que realicen una serie de operaciones para llegar a un resultado. Sin embargo, después de hacer la operación es importante **mostrar** el resultado, pues si no se muestra ninguna información el usuario no podrá saber si el programa funcionó o no. Para esto se usan las instrucciones de escritura o de salida.

En Java existen varias formas de escribir o mostrar datos de salida al usuario, y en esta sección se mostrarán dos de ellas: una para mostrar datos por consola (solo texto) y otra para mostrar datos en pequeña ventanas llamadas ventanas de diálogo.

• **Escritura por consola**

Para mostrar algún mensaje por pantalla, la instrucción que se utiliza en Java es:

| System.out.println (MENSAJE); |
| --- |

MENSAJE es un *String* (puede ser una

variable o una cadena de texto entre

comillas).

Por ejemplo, se desea mostrar por pantalla el mensaje “¡Un saludo!”, la instrucción es:

29

| /\*\*  \* Programa para mostrar un saludo en pantalla – por consola \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class SaludoConsola  {  public static void main(String args[])  {  **System.out.println**("¡Un saludo!");  }  } |
| --- |

El resultado de ejecutar este programa es:

| ¡Un saludo! |
| --- |

Si se desea mostrar el resultado de una variable de un tipo diferente a *String* se puede usar el operador de concatenación de cadenas, que es el **+** (como el de suma, pero para cadenas es unir una cadena con otra).

Por ejemplo, retomando un ejercicio de un capítulo anterior: un programa para calcular y mostrar el valor de una venta, conociendo la cantidad de productos (que es 3) y el valor del producto (que es $20.000).

| /\*\*  \* Determina el valor de una venta, basado en la cantidad de productos \* y el valor del producto.  \* @author Sandra V. Hurtado  \* @version 1.5  \*/  public class Venta  {  public static void main (String args [])  {  int cantidadProductos = 3;  int valorProducto = 20000;  double valorVenta = cantidadProductos \* valorProducto;  System.out.println("Valor de la venta: " + valorVenta);  }  } |
| --- |

El resultado que se muestra es:

| Valor de la venta: 60000.0 |
| --- |

• **Escritura por ventana de diálogo**

De manera similar hay otro método para mostrar un mensaje, pero usando una ventana de diálogo. Este método es:

30

| JOptionPane.showMessageDialog(ventanaMayor,MENSAJE); |
| --- |

Como por el momento no se tienen más

ventanas el primer valor será *null*.

Sin embargo, para poder usar este método, se debe escribir, **antes de la clase**, la siguiente instrucción:

| import javax.swing.JOptionPane; |
| --- |

Por ejemplo, un código en Java para mostrar el mensaje “¡Un saludo!” en una ventana de diálogo, es:

| **import javax.swing.JOptionPane;**  /\*\*  \* Programa para mostrar un saludo en una ventana de diálogo \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class SaludoDialogo  {  public static void main (String args [])  {  **JOptionPane.showMessageDialog**(null, "¡Un saludo!");  }  } |
| --- |

El resultado de ejecutar este programa es:



***Ejercicio 4-1***

Escriba un programa en Java que calcule y muestre el área de un triángulo rectángulo, conociendo que su base es de 7.5 centímetros y la altura es de 6 centímetros.

**4.2. Instrucciones condicionales**

Muchas veces durante un programa es importante verificar una condición para determinar si se ejecutan o no un conjunto de instrucciones. Por ejemplo: “Mostrar un mensaje de bienvenida si la persona es mayor de edad, o un mensaje de prohibido si es menor de edad”. Para esto se debe verificar si la edad de la persona es igual o mayor a 18 años, y dependiendo del resultado se ejecutará la instrucción correspondiente.

31

Este tipo de instrucciones, que permiten **analizar una condición** para saber cuáles acciones ejecutar, se denominan estructuras de decisión o condicionales.

A continuación, se presentan brevemente tres instrucciones de decisión en el lenguaje de programación Java: selección simple (*if*), selección doble (*if-else*) y selección múltiple (*switch*).

**4.3. Selección simple (*if*)**

Esta estructura permite tomar una decisión para saber si se ejecutan un conjunto de instrucciones o no. La sintaxis es:

La condición puede ser una variable

| *boolean* o una expresión que dé como  if (condición)  resultado un valor booleano.  {  *// instrucciones*  } |
| --- |

Por ejemplo, calcular el valor de una venta, dada la cantidad de productos y considerando que el valor del producto es $20.000. Se aplica un descuento del 10 %, pero solo si el valor de la venta es superior a $50.000.

| /\*\*  \* Determina el valor de una venta, basado en la cantidad de productos, \* el valor del producto y un posible descuento.  \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class VentaConDescuento  {  /\*\*  \* Calcula el valor de una venta, a partir de la cantidad de productos. \* El valor de un producto es 20.000,  \* y se hace un descuento del 10 % si el valor es mayor a 50.000 \* @param cantidadProductos la cantidad de productos vendidos \* @return el valor de la venta, en pesos  \*/  double calcularValorVenta (int cantidadProductos)  {  final double DESCUENTO = 0.10;  int valorProducto = 20000;  double valorVenta = cantidadProductos \* valorProducto;  *//continúa* |
| --- |

32

| **if (valorVenta > 50000)**  **{**  valorVenta = valorVenta - (valorVenta \* DESCUENTO);  **}**  return valorVenta;  }  } *// fin clase VentaConDescuento* |
| --- |

Para probar esta función es necesario escribir otra clase con el método *main*. Se presenta un código simplificado para esta otra clase, solo como ilustración, pues ese tema se tratará en otros capítulos.

| /\*\*  \* Clase para usar la funcionalidad del valor de la venta  \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class PruebaValorVenta  {  public static void main (String[] args)  {  VentaConDescuento venta = new VentaConDescuento();  double valorObtenido = venta.calcularValorVenta(4);  System.out.println("Valor de la venta: " + valorObtenido); }  } |
| --- |

***Ejercicio 4-2***

Complete el siguiente fragmento de código, para que diga que los menores de 12 años y los mayores de 70 años deben ingresar con una persona responsable. Tenga en cuenta que la edad de la persona está en una variable llamada “edad”:

if ( )

|  |
| --- |

{

System.out.println("Debe ingresar con un responsable"); }

**4.4. Selección doble (*if-else*)**

Esta estructura permite tomar una decisión para seleccionar entre dos conjuntos de instrucciones, es decir, saber si se ejecutan un conjunto de instrucciones u otro. Si se cumple la condición se ejecutan las instrucciones dentro del *if*, y si no se cumple la condición se ejecutan las instrucciones dentro del *else*.

La sintaxis es:

33

| if (condición)  {  *// Instrucciones si la condición se cumple*  }  else  {  *// Instrucciones si la condición no se cumple*  } |
| --- |

Note que las instrucciones dentro del *if* y del *else* van enmarcadas por llaves: { }

Por ejemplo, calcular el valor de una venta, dada la cantidad de productos vendidos y considerando que el valor del producto es $20.000. Se aplica un descuento del 10 %, si el valor de la venta es superior a 50.000, y en caso contrario el descuento es del 5 %.

| /\*\*  \* Determina el valor de una venta, basado en la cantidad de productos, \* el valor del producto y descuentos por promociones.  \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class VentaConPromocion  {  /\*\*  \* Calcula el valor de una venta, a partir de la cantidad de productos. \* El valor de un producto es 20.000.  \* Se hace un descuento del 10 % si el valor es mayor a 50.000, \* y un descuento del 5 % si el valor es igual o inferior a 50.000 \* @param cantidadProductos la cantidad de productos vendidos \* @return el valor de la venta, en pesos  \*/  double calcularValorVenta (int cantidadProductos)  {  final double DESCUENTO\_MAYOR = 0.10;  final double DESCUENTO\_MENOR = 0.05;  int valorProducto = 20000;  double valorVenta = cantidadProductos \* valorProducto;  **if (valorVenta > 50000)**  **{**  valorVenta = valorVenta - (valorVenta \* DESCUENTO\_MAYOR);  **}**  **else**  **{**  valorVenta = valorVenta - (valorVenta \* DESCUENTO\_MENOR);  **}**  return valorVenta;  }  } |
| --- |

34

***Ejercicio 4-3***

Determine cuál mensaje se muestra por pantalla si se ejecutan las siguientes instrucciones:

| int velocidad = 60;  if (velocidad > 60 | velocidad < 20)  {  System.out.println("Velocidad no permitida");  }  else  {  System.out.println("Velocidad adecuada");  } |
| --- |

**4.5. Selección múltiple (*switch*)**

Esta estructura permite ejecutar un conjunto de instrucciones dependiendo del valor de una variable.

En este caso no se trabaja con condiciones, sino con valores específicos de una variable. Los tipos de variables que pueden usarse en el *switch* son: *byte*, *short*, *int*, *char* y *String*.

La sintaxis es:

| switch (variable)  Dependiendo del valor de  {  la variable el programa  case valorUno :  “entra” por una de las  // instrucciones  opciones.  **break**;  case valorDos :  // instrucciones  **break**;  default : // instrucciones  } |
| --- |

Si no coincide con ningún valor en

la lista “entra” a las instrucciones

que están en *default*.

Este valor es opcional.

Por ejemplo, calcular el valor de una venta, dada la cantidad de productos comprados y el tipo de cliente. El valor del producto es $20.000 y los descuentos dependen del tipo de cliente, así: para el tipo 1 no hay descuento, para el tipo 2 es un 5 %, para el 3 el descuento es 8 % y para los demás tipos es el 12 %.

35

| /\*\*  \* Determina el valor de una venta, basado en la cantidad de productos, \* el valor del producto y descuentos por tipo de cliente.  \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class VentaPorTipoCliente  {  /\*\*  \* Calcula el valor de una venta, a partir de la cantidad de productos \* y del tipo de cliente. El valor de un producto es 20.000. \* Para clientes tipo 1 no hay descuento, para tipo 2 el descuento \* es 5 %, para tipo 3 es 8 % y para los demás es 12 %. \* @param cantidadProductos la cantidad de productos vendidos \* @param tipoCliente el tipo de cliente determina el descuento, \* va de 1 a 5.  \* @return el valor de la venta, en pesos  \*/  double calcularValorVenta (int cantidadProductos, int tipoCliente) {  final double DESCUENTO\_TIPO2 = 0.05;  final double DESCUENTO\_TIPO3 = 0.08;  final double DESCUENTO\_OTROS = 0.12;  int valorProducto = 20000;  double valorVenta = cantidadProductos \* valorProducto;  switch (tipoCliente)  {  case 1:  break;  case 2:  valorVenta = valorVenta - (valorVenta \* DESCUENTO\_TIPO2);  break;  case 3:  valorVenta = valorVenta - (valorVenta \* DESCUENTO\_TIPO3);  break;  default:  valorVenta = valorVenta - (valorVenta \* DESCUENTO\_OTROS);  }  return valorVenta;  }  } |
| --- |

**Observe que en cada uno de los casos las instrucciones terminan con break. Si no se escribe esta instrucción se continuarían ejecutando todas las demás instrucciones que hay en el *switch*.**

***Ejercicio 4-4***

En un jardín infantil el cobro de la matrícula depende de la edad del niño, pues esto es lo que determina en cuál área se asigna y qué materiales utilizará. Los niños de 1 año pagan un valor de $400.000, los de 2 y 3 años pagan $500.000 y los de 4 años pagan $600.000.

36

Sin embargo, si el niño tiene un hermano que también estudia en el jardín se hace un descuento del 15 %.

Dado el anterior enunciado, escriba las instrucciones Java para determinar el valor que debe pagar un niño por su matrícula, dada su edad y la información de si tiene o no un hermano en el jardín.

**4.6. Instrucciones de lectura**

Los programas que se escriben en cualquier lenguaje de programación pueden ser mucho más flexibles si trabajan con información que ingrese el usuario. Esto permite, por ejemplo, que un programa que suma dos números no tenga que ser cambiado cada vez que el usuario desee sumar dos valores diferentes, sino que cada vez que se ejecuta lea los dos números que el usuario desea sumar y muestre el resultado.

En Java existen varias formas de leer o solicitar datos de entrada al usuario, haciendo uso de clases y métodos especializados. En esta sección se mostrarán dos formas para leer datos, uno por consola y otro mediante ventanas de diálogo.

• **Lectura por consola**

Una de las formas de recibir datos del usuario es usando la clase *Scanner*. Los pasos para usar esta clase son:

1. Se debe colocar, antes de la clase, la instrucción:

| import java.util.Scanner; |
| --- |

2. En el código se escribe, antes de hacer cualquier lectura:

| Scanner lector = new Scanner(System.in); |
| --- |

3. Cuando se desee leer un dato se usan los siguientes métodos del objeto lector, y cada uno se recibe en una variable del tipo correspondiente:

• *nextLine()* para leer un *String*

• *nextInt()* para leer un valor *int*

• *nextDouble()* para leer un valor *double*

Antes de usar estos métodos es importante mostrar un mensaje al usuario para explicarle qué información es la que debe ingresar.

Por ejemplo:

| System.out.print("Precio del producto: ");  **double precio = lector.nextDouble();** |
| --- |

A continuación, se muestra un código en Java para mostrar si una persona puede participar en una votación o no, dependiendo de su edad. Se tiene en cuenta que una persona puede votar en Colombia a partir de los 18 años.

37

| **import java.util.Scanner;**  /\*\*  \* Determina si una persona puede votar o no, dependiendo de su edad \* (En Colombia la edad para poder votar son 18 años).  \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class EdadVotarConsola  {  public static void main (String args [])  {  **Scanner lector = new Scanner(System.in);**  final int EDAD\_VOTAR = 18;  System.out.print("Escriba su edad: ");  **int edad = lector.nextInt();**  if (edad >= EDAD\_VOTAR)  {  System.out.println("Sí puede votar");  }  else  {  System.out.println("No puede votar");  }  }  } |
| --- |

Un ejemplo de ejecución de este código, donde aparece en verde lo que escribe el usuario:

| Escriba su edad: 60  Sí puede votar |
| --- |

Otro ejemplo:

| Escriba su edad: 12  No puede votar |
| --- |

• **Lectura con ventana de diálogo**

Otra de las formas de solicitar datos al usuario es usando ventanas de diálogo, para lo cual los pasos son:

1. Se debe colocar, antes de la clase, la instrucción:

| import javax.swing.JOptionPane; |
| --- |

2. Usar la siguiente instrucción cuando se desee leer un dato:

| JOptionPane.showInputDialog(MENSAJE) |
| --- |

Mensaje que se muestra al usuario para

explicarle qué dato debe ingresar.

38

Se debe asignar el valor que retorna a una variable de tipo *String*, por ejemplo:

| String nombreUsuario = JOptionPane.showInputDialog("Nombre:"); |
| --- |

3. Como el método anterior solo permite pedir una cadena (*String*), se debe usar otro método adicional para realizar la conversión en caso de necesitar leer un *int* o un *double*: • Integer.parseInt(*CADENA*) para convertir una cadena en un *int*

• Double.parseDouble(*CADENA*) para convertir una cadena en un *double* Por ejemplo:

| String valorCadena = **JOptionPane.showInputDialog**("Valor producto:"); double precio = **Double.parseDouble**(valorCadena); |
| --- |

De nuevo se mostrará el código en Java para pedir la edad de una persona y decir si puede votar o no, pero usando la ventana de diálogo para pedir los datos de entrada:

| **import javax.swing.JOptionPane;**  /\*\*  \* Determina si una persona puede votar o no, dependiendo de su edad. \* (En Colombia la edad para poder votar son 18 años).  \* Muestra la información y solicita los datos por ventanas de diálogo \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class EdadVotarDialogo  {  public static void main (String args [])  {  **String cadenaEdad = JOptionPane.showInputDialog(**  **"Escriba su edad");**  final int EDAD\_VOTAR = 18;  **int edad = Integer.parseInt(cadenaEdad);**  if (edad >= EDAD\_VOTAR)  {  JOptionPane.showMessageDialog(null,"Sí puede votar");  }  else  {  JOptionPane.showMessageDialog(null, "No puede votar");  }  }  } |
| --- |

Un ejemplo al ejecutar este código:

39



En caso de querer solicitar un *char*, tanto por consola como con ventanas de diálogo, se debe leer una cadena (*String*), y luego obtener el primer carácter de la cadena, con la instrucción:

*CADENA*.charAt(0)

Por ejemplo:

| String texto = "verdadero";  char primerCaracter = texto.charAt(0); |
| --- |

En este caso, en la variable “primerCaracter” queda la letra v.

***Ejercicio 4-5***

Modifique la clase Venta, mostrada en este capítulo, para que **la cantidad de productos** vendidos y **el valor del producto** se pidan al usuario como entradas, y que al final se muestre el valor de la venta. Realice la lectura de datos primero por consola y luego mediante ventanas de diálogo.

**4.7. Ejercicio resuelto**

Una persona llega a una entidad financiera para enviar una cantidad de dinero a otra persona. Para poder hacer esto le preguntan si desea realizar una consignación, una transferencia o un giro. Si realiza una consignación cobran el 2 % de la cantidad que desea enviar, si realiza una transferencia cobran el 4 % y si realiza un giro cobran el 3 %. El programa debe solicitar la cantidad de dinero que se desea enviar y la forma de envío, y debe mostrar cuál es el valor que se le cobra a la persona para hacer este envío.

Usando instrucciones de lectura y escritura por consola:

40

| import java.util.Scanner;  /\*\*  \* Cálculo del cobro a una persona para enviar una cantidad de dinero, \* dependiendo de la forma de envío.  \* Consignación (c) es 2%, transferencia (t) es 4% y giro (g) es 3% \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class EnvioDineroConsola  {  public static void main(String[] args)  {  Scanner lector = new Scanner(System.in);  System.out.print("Candidad de dinero que desea enviar: "); double cantidadDinero = lector.nextDouble();  lector.nextLine(); //Para "limpiar" el lector  System.out.print(  "Forma de envío: consignación, transferencia o giro ");  String cadenaForma = lector.nextLine();  char formaEnvio = cadenaForma.charAt(0);  double valorCobro = 0;  switch (formaEnvio)  {  case 'c': valorCobro = cantidadDinero \* 0.02;  break;  case 't': valorCobro = cantidadDinero \* 0.04;  break;  case 'g': valorCobro = cantidadDinero \* 0.03;  break;  }  System.out.println("El valor que cobrará es: "+ valorCobro);  lector.close(); //Para "cerrar" apropiadamente la consola  }  } |
| --- |

Usando ventanas de diálogo:

41

| import javax.swing.JOptionPane;  /\*\*  \* Cálculo del cobro a una persona para enviar una cantidad de dinero, \* dependiendo de la forma de envío.  \* Consignación (c) es 2%, transferencia (t) es 4% y giro (g) es 3% \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class EnvioDineroDialogo  {  public static void main(String[] args)  {  String cadenaCantidad = JOptionPane.showInputDialog(  "Cantidad de dinero que desea enviar ");  double cantidadDinero = Double.parseDouble(cadenaCantidad);  String cadenaForma = JOptionPane.showInputDialog(  "Forma de envío: consignación, transferencia o giro ");  char formaEnvio = cadenaForma.charAt(0);  double valorCobro = 0;  switch (formaEnvio)  {  case 'c': valorCobro = cantidadDinero \* 0.02;  break;  case 't': valorCobro = cantidadDinero \* 0.04;  break;  case 'g': valorCobro = cantidadDinero \* 0.03;  break;  }  JOptionPane.showMessageDialog(null,  "El valor que cobrará es: "+ valorCobro);  }  } |
| --- |

42

**5. Métodos**

En este capítulo se extiende la definición de método y se muestra su representación gráfica en UML y la forma de implementarlo en el lenguaje de programación Java.

Al finalizar este capítulo usted debe ser capaz de:

• Representar los métodos de una clase usando una notación gráfica estándar. • Implementar una clase sencilla en Java incluyendo sus métodos.

**5.1. Definición**

Como se ha explicado previamente, los objetos, además de tener características e identidad, tienen comportamiento. El comportamiento se refiere a las acciones que puede hacer un objeto, o las acciones que se pueden hacer con él, incluyendo responder “preguntas” sobre sus características.

Cuando se modela un grupo de objetos mediante una clase, el comportamiento común que se desee representar de estos objetos se establece en los métodos. Por ejemplo, en un carro (automóvil) sus métodos son: encender, acelerar, frenar, cambiar una llanta, venderlo, etc.

Un método, por lo tanto, representa **una acción, comportamiento o servicio de los objetos**. También puede entenderse un método como una pequeña función o procedimiento que actúa sobre el objeto y sus atributos.

Es importante tener claro que un método no es un programa completo, solo una parte pequeña de un programa, y que está relacionado con el objeto sobre el cual se ejecuta. Un programa completo está formado por varios objetos que hacen uso de sus métodos para cumplir alguna funcionalidad.

Para definir un método, los elementos son:

• El nombre: Por lo general es un verbo que explica la acción que se puede realizar. • Los parámetros: Corresponden a información adicional que se envía al objeto para que pueda realizar la acción. No siempre es necesario enviar información adicional al objeto para usar un método, depende de lo que se desee pedir. Por ejemplo, si a una persona se le pregunta el nombre, no es necesario darle información adicional, porque cada persona tiene como atributo su nombre. Pero si a una persona se le pide que pague una factura, se le debe dar la factura y el dinero, porque estos no son atributos de la persona.

Un método puede tener cero, uno o más parámetros.

• El tipo de retorno: Corresponde al tipo de información que se espera que devuelva (o “retorne”) un objeto cuando se llame al método. No siempre los métodos retornan algo, por ejemplo, si a una persona se le pregunta el nombre, se espera que retorne una cadena de caracteres con el nombre, pero si a una persona se le pide que entre a una oficina solo se espera que realice la acción y no que retorne ninguna información.

Un método solo puede tener **un** valor de retorno (o ninguno). No es posible que un método retorne varias cosas. Si se necesitan varios resultados, se debe dividir el método y hacer un método diferente para cada resultado.

43

**5.2. Modelado de métodos**

Para establecer los métodos de las clases se tienen en cuenta todas las fuentes de información disponibles en el proyecto para determinar qué acciones realiza cada objeto o que acciones se pueden realizar con ese objeto en el programa. Los métodos por lo general se relacionan con los atributos que tiene cada objeto, permitiendo consultarlos, modificarlos o hacer operaciones con ellos.

Es importante tener en cuenta que en la programación orientada a objetos cada método no debe realizar muchas acciones, solo debe realizar una función concreta.

Al igual que con los atributos, solo se deben establecer los métodos que sean **relevantes** para el sistema que se esté modelando. Por ejemplo, en un almacén donde vendan carros, no son importantes los métodos acelerar o frenar del carro; sino otros métodos como definir el precio del carro, venderlo o asignarle la placa (si es nuevo).

En un ejemplo previo: “El señor Pepe Pérez se acerca al cajero electrónico del banco MuchoDinero que hay en la esquina de la calle 51. Al llegar pasa su tarjeta débito y *solicita el saldo de su cuenta* de ahorros; obtiene un recibo y luego se retira. Posteriormente se acerca a este mismo cajero la señora Lola López, quien también utiliza su tarjeta y luego *retira $200.000 de su cuenta* corriente”.

Se han dejado en cursiva las acciones que se realizan con cada cuenta, que son: consultar saldo y retirar dinero. Por lo tanto, estas dos acciones o servicios se identifican como métodos. También, considerando el contexto, que es una entidad bancaria, se definirá el método *consignar*.

Para cada método se definirá si necesita que le envíen información adicional para poder realizar su función (los parámetros) y si devolverá o retornará algún valor, y de qué tipo sería este valor. Por ejemplo, para la cuenta bancaria:

| **Nombre del**  **Método** | **Parámetros y su tipo de dato (en Java)** | **Lo que retorna y su tipo de dato (en Java)** |
| --- | --- | --- |
| Consultar saldo | No requiere (pues el saldo es un atributo de la clase). | El saldo: *double* |
| Retirar | La cantidad de dinero que se desea retirar: *double*. | Una indicación de si se pudo retirar o no: *boolean* |
| Consignar | La cantidad de dinero que se consignará: *double*. | No retorna nada. |

***Ejercicio 5-1***

A partir del siguiente enunciado, identifique los métodos – con sus parámetros y retorno – que tendría la clase Colectivo.

En la ciudad de Lejanías tienen un servicio de transporte muy exclusivo, que funciona solo con colectivos. Cada colectivo tiene capacidad para 8 personas únicamente, y durante todo el recorrido se suben y bajan personas. Por supuesto, al final del recorrido se deben bajar todas las personas que vayan en el colectivo. Se debe tener en cuenta que no es posible que se suban más personas que la capacidad dada por el colectivo. En algunos puntos del recorrido se tienen puntos de control donde un supervisor verifica la cantidad de personas que tiene el colectivo, garantizando así que se cumplan con las normas establecidas.

44

**5.3. Representación**

En el tercer compartimiento del rectángulo que representa a una clase van los métodos, es decir, los servicios que tendrán los objetos de esa clase.

Por ejemplo, para la clase Cuenta, el diagrama es:

**Cuenta**

número: String

saldo: double

tipo: String

consultarSaldo(): double

consignar(cantidad: double): void retirar(cantidad: double): boolean

La forma de representar cada método es:

Métodos detallados

| nombre (parámetros): retorno |
| --- |

Un **parámetro** es una variable que contiene información adicional (diferente a la que ya tiene el objeto), que necesita recibir el método para poder cumplir con el servicio que le están pidiendo. Un parámetro es un valor de “entrada” para el método.

Cada parámetro tiene un nombre y un tipo de dato, así:

| nombre: tipo |
| --- |

Por ejemplo, en la clase Cuenta los parámetros de cada método son:

• consultarSaldo**()**:double → no tiene parámetros. Para una cuenta poder consultar su saldo no necesita nada, pues esa información ya la tiene cada objeto internamente como atributo y por eso no recibe parámetros. Se colocan los paréntesis sin nada dentro.

• retirar(**cantidad:double**):boolean → tiene el parámetro “cantidad”. Para poder retirar de una cuenta se debe saber cuánta cantidad de dinero se desea retirar. Esta cantidad puede ser un número con decimales, por lo que se selecciona el tipo de dato *double*.

• consignar(**cantidad:double**):void → tiene el parámetro “cantidad”. Para poder consignar se necesita saber qué cantidad de dinero se desea consignar. Esa cantidad es un valor de tipo *double*.

Además, dependiendo del servicio que se le pida al objeto, él **retornará** algún valor o no. Por ejemplo, cuando usted dice “por favor abran la ventana”, no espera que le den nada, solo que se realice la acción. Por otra parte, cuando dice “por favor me presta una regla”, usted espera que le den algo (una regla).

45

Con los métodos sucede algo similar: todos realizan acciones, pero algunos retornarán algo y otros no retornarán nada. En Java, si el método retorna algún valor se debe definir de qué tipo es, y si no retorna nada se escribe ***void***.

En el caso de la clase Cuenta:

• consultarSaldo():**double** → Se espera que indique cuál es el saldo, y por lo tanto hay un valor de retorno. Como el saldo es de tipo *double*, el tipo de retorno de este método también es *double*.

• retirar(cantidad:double):**boolean** → Cada cuenta realiza el retiro de dinero internamente, pero algunas veces no podrá hacerlo porque no hay saldo suficiente. Por ese motivo se espera saber si se pudo realizar el retiro o no. Es decir, además de cambiar internamente el saldo del objeto cuenta, se espera un valor de verdad como respuesta (true o false), lo cual es el tipo *boolean*.

• consignar(cantidad:double):**void** → No se espera que retorne nada. Simplemente se desea que se consigne el valor en la cuenta, es decir, que se aumente su saldo, pero esto es algo que se hará internamente, no hay que esperar respuesta. En este caso se define como tipo de retorno “void”.

***Ejercicio 5-2***

a) Defina para los siguientes métodos si necesitan parámetros o no, y de qué tipo serían: • El método “ladrar” de un perro

• El método “sumar” de una calculadora

• El método “cocinar” de un horno microondas

• El método “encender” de un computador

b) Defina para los siguientes métodos si retornan algo o no, y de qué tipo sería lo que retorna:

• El método “sumar” de una calculadora

• El método “encender” de un computador

• El método “cambiarTamaño” de un cuadrado

• El método “consultarNombre” de una persona

**5.4. Implementación en Java**

Después de tener identificados todos los métodos en el diagrama de clases, se puede elaborar el código correspondiente en un lenguaje de programación como Java.

Los métodos en el lenguaje Java tienen dos partes:

• Un encabezado: tiene la información general del método, que es igual a la que está en el diagrama, solo que escrito con la notación del lenguaje de programación. • Un cuerpo: es donde se escribe el código, que se ejecutará cada vez que llamen al método. Es decir, es el “corazón” del método, donde se define su comportamiento, las acciones que realiza con el objeto y sus atributos.

A continuación, se presentará la forma de definir los métodos de una clase usando el lenguaje de programación Java.

46

• **El encabezado**

El encabezado del método es donde se indica su nombre, los parámetros que recibe y el tipo de dato que retorna. La forma de definirlo es:

| /\*\*  \* Descripción del método  \* @param nombreParámetro explicación  \* @return explicación  \*/  tipoRetorno nombreMétodo(parámetros)  {  } |
| --- |

• Lo primero que debe ir es el comentario que explica en qué consiste el método, los parámetros que recibe y lo que retorna.

o Para cada parámetro se usa un *@param*, seguido del nombre del parámetro y luego una explicación del mismo.

o Para lo que retorna se usa *@return*, seguido de una explicación de lo que retorna.

• Luego va la línea de encabezado del método, también llamada la interfaz o la firma del método, así:

o Primero va el tipo de retorno (como *int*, *double* o *void*).

o Luego sigue el nombre del método, con la primera letra en minúscula. o Luego van los parámetros entre paréntesis.

▪ Si no recibe parámetros se colocan los paréntesis vacíos.

▪ Si van varios parámetros se separan por comas.

▪ Para cada parámetro va primero el tipo de dato y luego el nombre.

Para el ejemplo:

a) El encabezado del método consultarSaldo de la clase Cuenta, que aparece en el diagrama como “consultarSaldo(): double”, es:

| /\*\*  \* Consulta el saldo actual de la cuenta (el dinero que tiene). \* @return el valor del saldo, en pesos.  \*/  double consultarSaldo() |
| --- |

b) El encabezado del método “retirar(cantidad:double): boolean” es:

| /\*\*  \* Retira o saca una cantidad de dinero de la cuenta,  \* si hay saldo suficiente.  \* @param cantidad la cantidad que se desea retirar, en pesos. \* @return si se pudo hacer el retiro porque tenía dinero  \* suficiente o no (*true* o *false*)  \*/  boolean retirar(double cantidad) |
| --- |

47

c) El encabezado del método “consignar(cantidad:double): void” es:

| /\*\*  \* Consigna o adiciona una cantidad de dinero en la cuenta, \* lo cual incrementa el saldo.  \* @param cantidad la cantidad de dinero que se desea  \* consignar, en pesos  \*/  void consignar(double cantidad) |
| --- |

• **El cuerpo**

El cuerpo de cada método tiene las instrucciones Java para hacer las acciones que se requieren. El cuerpo de los métodos siempre va enmarcado o encerrado por llaves; esto permite diferenciar los diferentes métodos que tenga una clase.

Por ejemplo, para el método “consignar”, lo que se hace es sumar la cantidad que se recibe al saldo de la cuenta, así:

| /\*\*  \* Consigna o adiciona una cantidad de dinero en la cuenta, \* lo cual incrementa el saldo.  \* @param cantidad la cantidad de dinero que se desea  \* consignar, en pesos  \*/  void consignar(double cantidad)  **{**  **saldo = saldo + cantidad;**  **}** |
| --- |

Cuando un método retorna algún valor, la instrucción para hacerlo es:

| return **loQueRetorna**; |
| --- |

Puede ser un valor, una variable o una

expresión, que corresponda al tipo de dato

que se indicó como retorno del método.

Por ejemplo, para el método “retirar”, debe mirar si el saldo alcanza para retirar la cantidad solicitada. Si es así se resta del saldo y se retorna un valor verdadero (*true*), y si no alcanza el saldo se retorna un valor falso (*false*).

48

| /\*\*  \* Retira o saca una cantidad de dinero de la cuenta,  \* si hay saldo suficiente.  \* @param cantidad la cantidad que se desea retirar, en pesos. \* @return si se pudo hacer el retiro porque tenía dinero  \* suficiente o no (true o false)  \*/  boolean retirar(double cantidad)  {  if (saldo >= cantidad)  {  saldo = saldo - cantidad;  **return true;**  }  else  {  **return false;**  }  } |
| --- |

| Es importante tener en cuenta:  • Si un método tiene en su encabezado algún tipo de retorno, siempre debe tener una instrucción *return*.  • La instrucción *return* termina la ejecución del método, incluso si está dentro de un ciclo. |
| --- |

Por ejemplo, para el método “consultarSaldo” no es necesario hacer ninguna operación, solo se debe retornar el saldo actual, por lo tanto, el método queda:

| /\*\*  \* Consulta el saldo actual de la cuenta (el dinero que tiene). \* @return el valor del saldo, en pesos.  \*/  double consultarSaldo()  {  return saldo;  } |
| --- |

A continuación, se presenta el código completo de la clase Cuenta, como referencia:

49

/\*\*

\* Una cuenta bancaria, es decir, un registro en

\* una entidad bancaria que tiene un saldo (cantidad de dinero) \* que se puede modificar mediante retiros o consignaciones. \* @author Sandra V. Hurtado

\* @version 1.5

\*/

public class Cuenta

{

String numero;

double saldo;

String tipo;

/\*\*

\* Consulta el saldo actual de la cuenta (el dinero que tiene). \* @return el valor del saldo, en pesos.

\*/

double consultarSaldo()

{

return saldo;

}

/\*\*

\* Retira o saca una cantidad de dinero de la cuenta, \* si hay saldo suficiente.

\* @param cantidad la cantidad que se desea retirar, en pesos. \* @return si se pudo hacer el retiro porque tenía dinero \* suficiente o no (true o false)

\*/

boolean retirar(double cantidad)

{

if (saldo >= cantidad)

{

saldo = saldo - cantidad;

return true;

}

else

{

return false;

}

}

/\*\*

\* Consigna o adiciona una cantidad de dinero en la cuenta, \* lo cual incrementa el saldo.

\* @param cantidad la cantidad de dinero que se desea \* consignar, en pesos

\*/

void consignar(double cantidad)

{

saldo = saldo + cantidad;

}

} *// fin clase Cuenta*

50

***Ejercicio 5-3***

Escriba el código Java para las clases que se presentan en el siguiente diagrama, incluyendo el código de los métodos. Tenga en cuenta que el IMC (índice de masa corporal) es el peso dividido el cuadrado de la estatura (peso / estatura2).

**Rectángulo**

base: double

altura: double

calcularÁrea(): double

calcularPerímetro(): double **5.5. Ejercicio resuelto**

**Báscula**

calcularIMC(peso: double, estatura: double): double

La abuela Pata prepara deliciosos dulces en su cocina, de diferentes sabores. Ella envasa cada dulce en frascos de 200 gramos, que marca con la fecha para garantizar siempre su frescura. Su familia disfruta mucho estos dulces y por supuesto cuando pueden sacan dulce de los frascos para disfrutarlos.

El diagrama de clases, sin detallar los métodos, es:

**FrascoDeDulce**

sabor: String

cantidad: int

fecha: String

envasarDulce()

sacarDulce()

consultarCantidad()

Se analiza cada método para determinar si necesita o no parámetros y si retorna o no algún valor.

• envasarDulce:

o Parámetros: Necesita saber la cantidad de dulce que se desea envasar y también la fecha en la cual se envasó, por lo tanto, necesita dos parámetros. El primer parámetro se llamará “cantidadEnvasar” y será un valor entero (*int*), que representa los gramos de dulce que se desean envasar en el frasco. El segundo parámetro se llamará “fechaEnvasado” y será una cadena de texto (*String*).

o Retorno: Como los frascos tienen un límite de 200 gramos este método devolverá una indicación de si se pudo envasar la cantidad deseada o no (en caso de ser superior a 200 gramos). Esto corresponde a un valor de verdad (*true* o *false*), que es de tipo *boolean*.

• sacarDulce:

o Parámetros: Necesita saber la cantidad de dulce que se desea sacar, para así disminuir la cantidad que hay en el frasco. Por lo tanto, necesita un

51

parámetro que sea la cantidad de dulce que se desea sacar, en gramos. Este parámetro se llamará “cantidadSacar” y será un valor entero (*int*).

o Retorno: Hay que tener en cuenta que no siempre se alcanza a sacar todo lo que se desea porque a veces hay menos cantidad de lo que se cree. Por eso este método debe indicar cuánta cantidad efectivamente se pudo sacar. Es decir, este método retornará un valor entero (*int*).

• consultarCantidad:

o Parámetros: No necesita recibir ningún valor como parámetro, porque la información que necesita, que es la cantidad que hay en el frasco, ya está como atributo.

o Retorno: Este método debe retornar un valor, porque lo que se necesita es precisamente saber la cantidad de dulce que hay en el frasco. Por lo tanto, debe retornar un valor entero (*int*).

Después de este análisis el diagrama de clases con la información detallada de los métodos queda:

**FrascoDeDulce**

sabor: String

cantidad: int

fecha: String

envasarDulce(cantidadEnvasar: int, fechaEnvasado: String): boolean

sacarDulce(cantidadSacar: int): int

consultarCantidad(): int

El código correspondiente a esta clase, incluyendo los métodos, es:

| /\*\*  \* Un frasco con delicioso dulce, elaborado por la abuela pata. \* Por ejemplo, un frasco de dulce de guayaba.  \* @author Sandra V. Hurtado  \* @version 1.5  \*/  public class FrascoDeDulce  {  String sabor;  int cantidad;  String fecha;  /\*\*  \* Envasar o insertar dulce en el frasco, hasta máximo 200 gramos. \* @param cantidadEnvasar la cantidad, en gramos,  \* que se desea envasar de dulce  \* @param fechaEnvasado fecha en la cual se envasa el dulce \* @return una indicación (true/false) de si se pudo envasar o no \* toda la cantidad deseada o solo 200 gramos  \*/  *//continúa* |
| --- |

52

| boolean envasarDulce(int cantidadEnvasar, String fechaEnvasado) {  fecha = fechaEnvasado;  if (cantidadEnvasar > 200)  {  cantidad = 200;  return false;  }  else  {  cantidad = cantidadEnvasar;  return true;  }  }  /\*\*  \* Saca cierta cantidad del dulce del frasco, si hay suficiente \* en el frasco; pero si hay menos solo se puede sacar lo que queda. \* @param cantidadSacar la cantidad, en gramos,  \* que se desea sacar de dulce  \* @return la cantidad de dulce que se pudo sacar, en gramos \*/  int sacarDulce(int cantidadSacar)  {  int cantidadSacada = 0;  if (cantidad >= cantidadSacar)  {  cantidad = cantidad - cantidadSacar;  cantidadSacada = cantidadSacar;  }  else  {  cantidadSacada = cantidad;  cantidad = 0;  }  return cantidadSacada;  }  /\*\*  \* Consulta la cantidad de dulce del frasco  \* @return la cantidad de dulce que hay en el frasco, en gramos \*/  int consultarCantidad()  {  return cantidad;  }  } *// fin clase FrascoDeDulce* |
| --- |

53

**6. Referencias y Operador Punto**

En este capítulo se explicará el concepto de referencia en el lenguaje de programación Java, que es la forma en la cual se trabaja con los objetos. Se mostrará la diferencia entre los valores y las referencias; se explicará el uso de los métodos en los objetos con el operador punto y se presentará el valor especial *null*.

A finalizar este capítulo usted debe ser capaz de:

• Explicar la diferencia entre un valor y una referencia en Java.

• Usar el operador punto para usar un servicio de un objeto, es decir, llamar uno de sus métodos.

**6.1. Tipos de variables en Java**

En un programa orientado a objetos lo primero que debe hacerse es definir las clases que se necesitarán. Sin embargo, después de crear las clases, y para que el programa logre los objetivos deseados, es muy importante crear objetos de esas clases y trabajar con sus valores y métodos.

Para poder crear y utilizar los objetos en un lenguaje de programación como Java, es necesario tener variables. Java tiene dos grandes grupos de tipos de datos para poder crear variables:

• Los primitivos (o básicos): corresponden a valores simples y son: *boolean*, *byte*, *short*, *int*, *long*, *float*, *double* y *char*.

• Los referenciados (u objeto): corresponden a las clases, y es el tema que se presentará en este capítulo.

**6.2. Referencias**

Cuando se trabaja con tipos primitivos las variables **contienen** un dato o valor, pero cuando se trabaja con objetos las variables **referencian** el objeto, es decir, no lo contienen, sino que permiten llegar al objeto para usarlo; es como un alias o un apodo con el cual conocemos a una persona.

Por ejemplo, se definirán dos variables: una primitiva y otra referenciada, cada una con un valor inicial.

| int ancho = 20;  String frase = new String("¡Viva aprender!"); |
| --- |

Una representación de cómo queda la memoria puede ser:

La variable “ancho”

ancho frase

contiene el valor.La variable “frase”

20 “¡Viva 

aprender!”

referencia al objeto.

54

Las variables de tipo primitivo se usan directamente en las expresiones, mientras que las variables referenciadas por lo general no se usan directamente, sino que a través de ellas se pueden llamar los métodos del objeto. Por ejemplo:

| int dobleAncho = ancho \* 2; |
| --- |

Se usa directamente

| String palabraInicial = frase.substring(1,5); |
| --- |

Permite usar el método “*substring*”

Al igual que las variables de tipo primitivo, las variables referenciadas pueden declararse antes de usarlas, pero en ese caso todavía no están referenciando ningún objeto, de la misma forma que una variable primitiva que no se ha inicializado no tiene ningún valor. Por ejemplo:

| int edad;  String descripcion; |
| --- |

La variable “edad”:

• Es de tipo *int*, es decir, un tipo primitivo.

• Por el momento no tiene ningún valor.

La variable “descripción”:

• Es de tipo *String*, es decir, un tipo referenciado (*String* es una clase). • Por el momento no referencia a ningún objeto.

edad descripcion

?

?

Para poder usar estas variables es necesario asignarles algún valor o referencia. A una variable referenciada se le puede asignar un objeto que ya existe, o uno nuevo, usando la instrucción *new*.

Cuando a una variable primitiva se le asigna otra variable, lo que se hace es copiar el valor, mientras que cuando a una variable referenciada se le asigna otra variable, lo que se hace es que referencia **el mismo objeto** (NO crea una copia).

Por ejemplo:

| int anhos = 25;  edad = anhos;  String texto = new String("Esta es una cadena de texto");  descripcion = texto; |
| --- |

anhos descripcion 

edad text

25

25

“Esta es una 

cadena de texto”

55

En otro ejemplo, se tiene la clase Cuenta: **Cuenta**

número: String

saldo: double

tipo: String

consultarSaldo(): double

consignar(cantidad: double): void

retirar(cantidad: double): boolean

Para crear un objeto de esta clase en Java, la instrucción es:

| Cuenta cuentaJuan = **new Cuenta();** |
| --- |

Lo que se está haciendo es: creando un objeto de la clase Cuenta, que es **referenciado** por la variable “cuentaJuan”.

Una forma de representar esto es:

cuentaJuan

número = “” 

saldo = 0.0

tipo = “”

Esto también se puede escribir en dos pasos, así:

**La variable** 

(que referencia)

**El objeto**

| Cuenta cuentaJuan; |
| --- |

En este primer paso se declara la variable. Es una variable llamada “cuentaJuan” de tipo Cuenta, es decir, para trabajar con objetos de esa clase, pero actualmente no está referenciando nada.

| cuentaJuan = new Cuenta(); |
| --- |

En este segundo paso se crea el objeto Cuenta (con ***new***) y se asigna para que sea referenciado por la variable “cuentaJuan”.

Ahora, se define otra variable de tipo Cuenta, y se le asigna el objeto que se acabó de crear, así:

| Cuenta cuentaBancaria = cuentaJuan; |
| --- |

Una representación de cómo queda la memoria puede ser:

56

cuentaBancaria cuentaJuan

número = “” 

saldo = 0.0

tipo = “”

NO se ha creado un nuevo objeto. La variable “cuentaBancaria” referencia **el mismo objeto** anterior.

Las dos variables están referenciando al mismo objeto.

Ahora se crea un nuevo objeto:

| Cuenta cuentaLola = new Cuenta(); |
| --- |

Lo cual se representa en memoria:

cuentaBancaria cuentaJuan

número = “” 

saldo = 0.0

tipo = “”

***Ejercicio 6-1***

******cuentaLola

número = “” saldo = 0.0 tipo = “” 

Se tienen **dos** objetos Cuenta.

Se tiene la siguiente clase y unas instrucciones en Java:

**Gato**

maullar(veces: int, volumen: String): void

| Gato gatito;  Gato consentido = new Gato(); |
| --- |

A partir del código anterior responda las siguientes preguntas:

• ¿Cuántos objetos de la clase Gato se crean?

• ¿Cuántas variables se están definiendo? ¿De qué tipo son?

• Elabore el dibujo que muestre cómo quedarían las variables y los objetos en memoria.

• Escriba la instrucción en Java para crear un objeto Gato y que sea referenciado por la variable gatito.

57

Volviendo al ejemplo con la clase Cuenta:

| cuentaUno  cuentaBanco  /*\* Código donde se crea un objeto Cuenta,*  *que es referenciado por dos variables*  *\*/*  Cuenta cuentaUno = new Cuenta();  número = “”  Cuenta cuentaBanco = cuentaUno;  saldo = 0.0  tipo = “” |
| --- |

| /*\* Ahora se consigna en la cuenta,*  cuentaUno  cuentaBanco  *usando la variable cuentaUno*  *\*/*  cuentaUno.consignar(200000);  número = “”  saldo =200000.0  tipo = “” |
| --- |

| */\* Se muestra el saldo,*  *usando la variable cuentaBanco*  *\*/*  System.out.println("Saldo: $"+ cuentaBanco.consultarSaldo()); |
| --- |

El resultado es:

| Saldo: $200000.0 |
| --- |

Aunque se usen diferentes variables se está trabajando con el mismo objeto.

**6.3. Uso de los métodos con el operador punto**

Después de crear los objetos, se pueden usar los servicios que ofrecen, es decir, **llamar** a los métodos para realizar alguna operación.

Siguiendo con el ejemplo de la clase Cuenta, se observa que tiene el método “consultarSaldo”. Usando este método es posible saber el saldo o cantidad de dinero de una cuenta, pero antes por lo general se consigna o se retira algún valor, para lo cual se pueden usar los métodos “consignar” y “retirar”.

Por ejemplo, para obtener y mostrar el saldo de una cuenta en la cual se consigna $150.000 y se retiran $50.000 y $20.000, el código en Java es:

58

| Cuenta cuenta = new Cuenta();  cuenta**.consignar(150000);**  boolean pudoPrimerRetiro = cuenta**.retirar(50000);**  boolean pudoSegundoRetiro = cuenta**.retirar(20000);**  double saldoFinal = cuenta**.consultarSaldo();**  System.out.println("El saldo es: " + saldoFinal); |
| --- |

Como se puede ver en el ejemplo anterior, la forma de usar (o llamar) los métodos de un objeto es con el operador punto (**.**).

Primero va la variable, sigue el punto y luego el método que se desee utilizar.

La sintaxis es:

objeto método ( )

Se debe tener en cuenta:

• Si el método retorna algo es adecuado guardar (asignar) lo que retorna en una variable, como en el ejemplo del saldo:

| double saldoFinal = cuenta.consultarSaldo(); |
| --- |

• Si el método no retorna nada (*void*) no se necesita ninguna variable. Como en el ejemplo de consignar:

| cuenta.consignar(150000); |
| --- |

• Si el método recibe parámetros, se deben enviar **los valores** para cada parámetro, en el orden adecuado y de acuerdo con los tipos de datos definidos.

Por ejemplo, se desea usar el método “maullar” que está definido en la clase Gato.

El método está definido así:

| maullar(veces:int, volumen:String):void |
| --- |

Es decir, para usar el método “maullar” se deben enviar dos parámetros: un entero (las veces que maullará) y una cadena (el volumen del maullido).

Por lo tanto, las instrucciones en Java para crear un objeto Gato y llamar a su método maullar, indicándole que maúlle 3 veces a volumen bajo serían:

| Gato mascota = new Gato();  mascota.**maullar(3,"bajo");** |
| --- |

***Ejercicio 6-2***

Escriba las instrucciones en Java para crear un objeto Gato y decirle que maúlle una vez muy alto.

59

***Ejercicio 6-3***

Elabore un dibujo que represente cómo quedarían las variables y los objetos en el siguiente código:

| String mensaje1 = "Buenos días";  String mensaje2 = mensaje1;  String mensaje3 = "¿Cómo está?"; |
| --- |

**6.4. Valor *null***

Cuando se definen variables dentro de un método es recomendable asignarles un valor inicial. Esto ayuda a evitar errores en la ejecución del programa.

Por ejemplo, el siguiente código es erróneo:

| int suma;  suma = suma + 1; //error |
| --- |

Al tratar de ejecutarlo, como la variable “suma” no tiene un valor inicial, no se puede ejecutar la segunda instrucción.

Lo mismo sucede para las **variables de tipo objeto o referenciadas**: Si no se le asigna ningún valor queda con “basura”, lo cual puede llevar a errores. Por este motivo, en Java se definió un valor especial para indicar que una variable de este tipo no está referenciando a ningún objeto: este valor es ***null***.

Por ejemplo, se tiene el siguiente código:

| Cuenta cuenta1 = new Cuenta();  Cuenta cuenta2 = null;  Cuenta cuenta3; |
| --- |

En el anterior código se tienen tres variables de tipo Cuenta, de las cuales la primera (cuenta1) está referenciando un objeto, la segunda (cuenta2) tiene valor *null* y la última (cuenta3) no tiene ningún valor definido, así:

cuenta1 cuenta2 cuenta3

**?**

**null** número = “” 

saldo = 0.0

tipo = “”

Posteriormente se desea saber la cantidad de dinero (o el saldo) que tiene cada cuenta – lo cual solo funcionará para la variable que está referenciando un objeto.

Es decir, en el siguiente código solo es correcta la primera instrucción, las dos siguientes generan errores al compilar o al ejecutar el programa, porque las variables no están referenciando ningún objeto:

60

| System.out.println(cuenta1.consultarSaldo());  System.out.println(cuenta2.consultarSaldo()); **//ERROR** System.out.println(cuenta3.consultarSaldo()); **//ERROR** |
| --- |

Para evitar estos errores se modificará el código. De esta manera, antes de preguntar por el saldo, se validará si la variable está referenciando un objeto, lo cual se hace preguntando si es diferente de ***null***, así:

a) Para cuenta1:

| if (**cuenta1 != null**)  {  System.out.println(cuenta1.consultarSaldo());  } |
| --- |

Este código funciona apropiadamente y muestra el saldo que tiene la cuenta. b) Para cuenta2:

| if (**cuenta2 != null**)  {  System.out.println(cuenta2.consultarSaldo());  } |
| --- |

Este código también funciona apropiadamente, y como la variable está con valor *null*, no entra al *if*. No se muestra el mensaje, pero no sale error y el programa continúa normalmente.

c) Para cuenta3:

| if (cuenta3 != null)  {  System.out.println(cuenta3.consultarSaldo());  } |
| --- |

Este código **genera un error**. Como la variable cuenta3 no tiene un valor definido –ni siquiera *null*– no se puede usar.

En resumen, con respecto al valor *null*:

• Se usa el valor *null* para inicializar una variable de tipo objeto (referenciada) cuando todavía no tiene un objeto asociado.

• Al comparar una variable de tipo objeto con el valor *null* se puede saber si tiene asociado un objeto o no.

• Si trata de usar una variable referenciada que no tiene ningún valor –ni siquiera null– se generará un error.

• Se pueden usar las variables que tengan valor *null,* pero solo para validar si están referenciando o no a un objeto.

• **Solo se pueden usar los métodos en variables que referencien un objeto**. No se pueden usar en variables que no tengan valor o que sean *null* porque se generaría un error.

61

**6.5. Ejercicio resuelto**

Considerando que se tiene la clase FrascoDeDulce (el diagrama se muestra a continuación), escriba el código en Java para crear un frasco, tratar de envasar 250 gamos de dulce, y mostrar si fue posible envasar los 250 gramos o solo 200. Luego intentar sacar 150 gramos y 60 gramos y mostrar en cada caso la cantidad de dulce que efectivamente se sacó y la cantidad de dulce que quedó en el frasco.

**FrascoDeDulce**

sabor: String

cantidad: int

fecha: String

envasarDulce(cantidadEnvasar: int, fechaEnvasado: String): boolean

sacarDulce(cantidadSacar: int): int

consultarCantidad(): int

| /\*\*  \* Creación de un objeto FrascoDeDulce, usando sus métodos  \* para envasar, sacar dulce y consultar la cantidad.  \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class PruebaDulces  {  public static void main (String args [])  {  *// Crear el frasco de dulce*  FrascoDeDulce frasco = **new FrascoDeDulce();**  *// Tratar de envasar 250 gramos*  boolean pudoEnvasar = **frasco.**envasarDulce(250,"10/09/2017"); if (pudoEnvasar)  {  System.out.println("Se envasaron 250 gramos");  }  else  {  System.out.println("Se envasaron solo 200 gramos");  }  *// Sacar 150 gramos*  int cantidadSacada = **frasco.**sacarDulce(150);  int cantidadFinal = **frasco.**consultarCantidad();  System.out.println("Del frasco se sacaron "+  cantidadSacada + " gramos, y quedan: "+ cantidadFinal); *//continúa* |
| --- |

62

| *// Sacar 60 gramos*  cantidadSacada = **frasco.**sacarDulce(60);  cantidadFinal = **frasco.**consultarCantidad();  System.out.println("Del frasco se sacaron "+  cantidadSacada + " gramos, y quedan: "+ cantidadFinal); }  } *// fin clase PruebaDulces* |
| --- |

El resultado de ejecutar este programa es:

| Se envasaron solo 200 gramos  Del frasco se sacaron 150 gramos y quedan 50  Del frasco se sacaron 50 gramos y quedan 0 |
| --- |

63

**7. Estructuras Repetitivas en Java**

En este capítulo se presenta la última parte de la introducción al lenguaje de programación Java, presentando las estructuras repetitivas o ciclos.

Al finalizar este capítulo usted debe ser capaz de:

• Describir tres instrucciones repetitivas en Java: *while*, *do-while* y *for*. • Incluir estructuras repetitivas en un método en Java.

**7.1. Definición**

Cuando se elaboran programas a veces se identifica un conjunto de instrucciones que se repiten. Por ejemplo, si se desea calcular el peso promedio de un grupo de 20 estudiantes, hay que preguntar 20 veces por el peso (por cada estudiante) e ir sumando para poder hacer al final el cálculo del promedio, que es la suma de los pesos dividido 20. Las instrucciones que se repiten en este caso son preguntar el peso e ir sumando. No es necesario escribirlas 20 veces, sino que se puede escribir una sola vez y enmarcarlas en una estructura repetitiva para indicar al programa que se ejecutan varias veces.

Las estructuras repetitivas, por lo tanto, permiten repetir un conjunto de instrucciones dependiendo de una condición. Mientras la condición sea verdadera las instrucciones se seguirán ejecutando. El ciclo se detiene cuando la condición se vuelva falsa.

En Java se tienen tres estructuras para repeticiones: *while*, *do-while* y *for*, las cuales se presentarán a continuación.

**7.2. Ciclo *while***

La sintaxis es:

Si la condición es verdadera **entra** (o repite) el ciclo, para

| ejecutar las instrucciones.  while (condición)  {  *// instrucciones*  } |
| --- |

Por ejemplo, se tiene la clase Sopa:

**Sopa**

cantidad: double

sacar(cantidadSacar: double): void

getCantidad(): double

El método “sacar” es para sacar una cantidad de sopa, en onzas.

El método “getCantidad” es para consultar la cantidad de sopa que hay en el momento.

Se desea repartir sopa para un almuerzo. Cada plato es de 20 onzas y se desea saber cuántos platos se alcanzan a repartir con una olla de sopa.

64

Para esto se tendrá otra clase, llamada ReparticiónSopa:

**ReparticiónSopa**

calcularPlatosSopa(): int

En el método “calcularPlatosSopa” se creará un objeto Sopa y se sacarán 20 onzas cada vez. Se continúa sacando sopa mientras aún alcance para un plato, es decir, mientras haya más de 20 onzas de sopa en el frasco.

Cuando haya menos de 20 onzas de sopa se termina de repartir, y se debe decir cuántas veces se pudo sacar sopa, es decir, cuántos platos se sacaron.

| /\*\*  \* Reparten varios platos de sopa, usando un ciclo.  \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class ReparticionSopa  {  /\*\*  \* Calcula cuántos platos se pueden sacar de una sopa  \* @return los platos de 20 onzas que se sacan de una sopa \*/  int calcularPlatosSopa()  {  Sopa ollaSopa = new Sopa();  int platos = 0;  *// Sigue mientras haya cantidad suficiente para otro plato* **while (ollaSopa.getCantidad() >= 20)**  **{**  ollaSopa.sacar(20);  platos++;  **}**  return platos;  }  } |
| --- |

**7.3. Ciclo *do-while***

Este ciclo es muy parecido al ciclo *while*, y su única diferencia es que la condición no se evalúa al principio sino al final del ciclo.

Como la condición se evalúa al final, este ciclo garantiza que las instrucciones se ejecutan por lo menos una vez.

La sintaxis es:

65

| do  {  *// instrucciones*  } while (condición);Si la condición es verdadera |
| --- |

**regresa** al ciclo ejecutando las

instrucciones.

A diferencia de otras estructuras de control en Java – condicionales y repetitivas-, en este caso la instrucción termina con punto y coma. Esto se debe a que la condición va al final del bloque de instrucciones, y no al comienzo como todas las demás.

Por ejemplo, se tiene la clase Cajero, que representa cajeros electrónicos:

**Cajero**

cantidadDinero: int

getCantidadDinero(): int

retirarDinero(cantidad: int): boolean

El método “getCantidadDinero” es para saber la cantidad de dinero que tiene el cajero. El método “retirarDinero” es para retirar alguna cantidad del cajero. Retorna true si tenía dinero suficiente para retirar, y false si no pudo retirar porque no tenía dinero suficiente.

Se desea hacer una prueba con un cajero electrónico, para ir retirando de a 200.000 cada vez, mientras se pueda retirar, y al final debe mostrar cuánto dinero quedó en el cajero. Para esto se tendrá la clase DiagnósticoCajero, con el método *main*.

**DiagnósticoCajero**

+main(args: String[]): void

El código de esta clase es:

| /\*\*  \* Realiza diagnósticos o pruebas a un cajero electrónico,  \* retirando dinero varias veces, mientras todavía tenga suficiente. \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class DiagnosticoCajero  {  public static void main(String[] args)  {  Cajero cajeroPrueba = new Cajero();  boolean pudoRetirar;  *//continúa* |
| --- |

66

*// retira 200000 del cajero y sigue si pudo retirar*

**do**

**{**

pudoRetirar = cajeroPrueba.retirarDinero(200000);

**} while (pudoRetirar);**

int dineroRestante = cajeroPrueba.getCantidadDinero();

System.out.println("Dinero restante: "+ dineroRestante);

}

} *// fin clase DiagnosticoCajero*

**7.4. Ciclo *for***

Este ciclo permite incluir, en el encabezado, una instrucción de inicialización y una de iteración. Es decir, en el *for* –además de la condición– se puede incluir una instrucción que se hace **antes** de comenzar el ciclo (una sola vez) y una que se hace al final de **cada repetición**.

La sintaxis es:

Instrucción que se hace **una**

**sola vez,** antes de comenzar

Instrucción que se hace al finalizar **cada repetición**.

| for (incialización**;** condición**;** iteración) {  *// instrucciones*  } |
| --- |

Si la condición es

verdadera sigue en el ciclo

(como el *while*).

Por ejemplo, se tiene la clase Televisor:

**Televisor**

pulgadas: int

setPulgadas(pulgadasTV: int): void

calcularConsumo(): double

El método “setPulgadas” es para definir el tamaño, en pulgadas, que tendrá el televisor. El método “calcularConsumo” es para saber cuál es el consumo, en vatios, del televisor.

Se desea un programa en Java para crear 10 televisores de diferentes tamaños, empezando en 31 pulgadas, e incrementando en una pulgada cada vez hasta 40 pulgadas. De cada

67

televisor se debe mostrar cuál es el consumo, y al final se debe mostrar cuál es el consumo total de todos los televisores.

**FábricaTelevisores**

+main(args: String[]): void

El código de esta clase queda:

| /\*\*  \* Fábrica que construyen televisores de diferentes pulgadas \* y calcula y muestra su consumo, en vatios.  \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class FabricaTelevisores  {  public static void main(String[] args)  {  double consumoTotal = 0;  *// comienza con pulgadas en 31, y*  *// mientras sea menor o igual a 40 sigue construyendo televisores. // En cada iteración incrementa en 1 las pulgadas*  for (int pulgadas=31; pulgadas <= 40; pulgadas++)  {  Televisor tele = new Televisor();  tele.setPulgadas(pulgadas);  double consumoTv = tele.calcularConsumo();  consumoTotal += consumoTv;  System.out.println("Consumo televisor es:"+consumoTv); }  System.out.println("Total consumo: "+ consumoTotal); }  } |
| --- |

**7.5. Instrucción *break***

Esta instrucción permite terminar un ciclo sin importar si se cumple la condición o no. Cuando se encuentra esta instrucción el programa sale del ciclo y continúa con las instrucciones que hay después del ciclo.

Por ejemplo, se desea modificar el programa de los televisores para que el ciclo se detenga si algún televisor tiene un consumo mayor a 50. En este caso se debe mostrar de cuántas pulgadas es el televisor con este alto consumo.

68

| /\*\*  \* Fábrica que construyen televisores de diferentes pulgadas \* y calcula y muestra su consumo, en vatios.  \* Se desea un máximo consumo de 50 vatios, si algún televisor \* supera esta cantidad se detiene la producción.  \* @author Sandra V. Hurtado  \* @version 2.0  \*/  public class FabricaTelevisores  {  public static void main(String[] args)  {  double consumoTotal = 0;  for (int pulgadas=30; pulgadas <= 40; pulgadas++)  {  Televisor tele = new Televisor();  tele.setTamaño(pulgadas);  double consumoTv = tele.calcularConsumo();  consumoTotal += consumoTv;  System.out.println("Consumo televisor es:"+consumoTv);  if (consumoTv > 50)  {  System.out.println("Pulgadas alto consumo: "+pulgadas); *// Termina el ciclo.*  **break;**  }  }  System.out.println("Total consumo: "+ consumoTotal);  }  } |
| --- |

***Ejercicio 7-1***

Implemente lo siguiente con un ciclo *for*:

| int edad = 0;  while (edad < 5)  {  System.out.println("Feliz cumpleaños # "+ edad);  edad = edad + 1;  } |
| --- |

***Ejercicio 7-2***

Se tiene la clase Cuadrado:

69

**Cuadrado**

lado: int

setLado(valorLado: int): void

calcularÁrea(): int

El método “setLado” es para definir el tamaño, en centímetros, que tendrá el cuadrado. El método “calcularÁrea” es para calcular el área del cuadrado.

Escriba un programa en Java que permita crear 11 cuadrados de diferentes lados, comenzado en un cuadrado de 25 centímetros de lado e incrementando cada vez en cinco centímetros. Al final del programa se debe mostrar cuál es el área total, es decir, la suma de las áreas de todos los cuadrados.

**7.6. Ejercicio resuelto**

Considerando que se tiene la clase FrascoDeDulce (el diagrama se muestra a continuación), escriba el código en Java para pedir al usuario cuántos frascos de dulce desea envasar, y luego se debe pedir la cantidad (en gramos) de dulce que se envasará en cada frasco. Al finalizar de envasar los frascos se debe decir cuántos gamos totales de dulce se envasaron.

**FrascoDeDulce**

sabor: String

cantidad: int

fecha: String

envasarDulce(cantidadEnvasar: int, fechaEnvasado: String): boolean

sacarDulce(cantidadSacar: int): int

consultarCantidad(): int

Para resolver el ejercicio, se creará otra clase, llamada EnvasaFrascosDulce, que tendrá el método *main*.

**EnvasaFrascosDulce**

+main(args: String[]): void

Dentro de este método se realizará lo propuesto en el enunciado, como se observa a continuación.

70

| import javax.swing.JOptionPane;  /\*\*  \* Programa para envasar varios dulces, solicitando de cada uno \* los gramos que se envasarán y mostrando la cantidad total al final. \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class EnvasaFrascosDulce  {  public static void main (String args [])  {  *// Variable para guardar el total envasado (en gramos)*  int cantidadTotalEnvasada = 0;  *// Pregunta cuántos frascos desea envasar*  String cadenaFrascos = JOptionPane.showInputDialog(  "Cantidad de frascos que desea envasar ");  int cantidadFrascos = Integer.parseInt(cadenaFrascos);  *// Ciclo para envasar los frascos*  **for (int i=0; i<cantidadFrascos; i++)**  {  FrascoDeDulce frascoDulce = new FrascoDeDulce();  String cadenaGramos = JOptionPane.showInputDialog(  "Cantidad de dulce (en gramos) que desea envasar: ");  int cantidadGramos = Integer.parseInt(cadenaGramos);  frascoDulce.envasarDulce(cantidadGramos, "17/07/2017");  *//Va sumando la cantidad de cada frasco*  cantidadTotalEnvasada += frascoDulce.consultarCantidad(); }  JOptionPane.showMessageDialog(null, "Se envasaron en total "+ cantidadTotalEnvasada + " gramos de dulce");  }  } *// fin clase EnvasaFrascosDulce* |
| --- |

71

**8. Referencia *this***

En este capítulo se presentará la palabra reservada *this* en Java, para poder hacer referencia a los atributos y métodos del mismo objeto con el cual se está trabajando. Esta referencia ayuda a evitar confusiones cuando se tienen algunas variables con el mismo nombre, y por lo tanto ayuda a que los programas sean más fáciles de entender.

Al finalizar este capítulo usted debe ser capaz de:

• Usar la referencia *this* para hacer uso de atributos y métodos del objeto dentro de la misma clase.

**8.1. Uso de métodos y atributos propios**

Recordemos:

• Después de crear los objetos, se pueden usar los servicios que ofrecen, tanto los atributos como los métodos.

• La forma de usar o llamar los atributos o métodos de un objeto es con el operador punto (**.**). Primero va la variable, sigue el punto y luego el atributo o método que se desee utilizar.

Lo anterior aplica para métodos que son de otros objetos, pero ¿qué pasa si se desea usar un método del mismo objeto?

Por ejemplo, en una empresa que vende teléfonos celulares se tiene la clase Celular:

**Celular**

marca: String

tipo: char

setMarca(marca: String): void

setTipo(tipo: char): void

calcularValor(): double

calcularIva(): double

El método “calcularValor” está definido de la siguiente forma:

| /\*\*  \* Calcula el valor del celular, así:  \* Los de tipo 'c' valen 50.000, los de tipo 'b' 130.0000  \* y los de de tipo 'a' 250.000  \* @return el valor del celular  \*/  *//continúa* |
| --- |

72

| double calcularValor()  {  double valor = 0;  switch (tipo)  {  case 'a': valor = 250.000;  break;  case 'b': valor = 130.000;  break;  case 'c': valor = 50.000;  break;  }  return valor;  } |
| --- |

El IVA es el 16 % del valor del celular, es decir, para poder calcular el IVA se necesita saber cuál es el valor del celular.

Por lo tanto, para no repetir código, dentro el método “calcularIva” se llamará al método “calcularValor” y así se obtendrá el valor del celular. Es decir, se usará un método del mismo objeto.

Cuando se usa un **método que es del mismo objeto**, se tienen dos opciones para usarlo:

a) Usar la palabra ***this*** para representar el objeto (como diciendo “**este** objeto”, en el que se está trabajando”), así:

**this** elMétodo( )

O,

b) Llamar el método directamente, ya que se está dentro de la misma clase, sin usar un objeto ni el punto. Es decir:

elMétodo( )

Por lo tanto, el método “calcularIva” puede quedar de dos formas:

a) Usando *this*:

| /\*\*  \* Calcula el IVA, que es el 16% del valor del celular.  \* @return el valor del IVA para el celular  \*/  double calcularIva ()  {  double valorCelular = **this.calcularValor();**  double iva = valorCelular \* 0.16;  return iva;  } |
| --- |

73

b) Sin usar *this*:

| /\*\*  \* Calcula el IVA, que es el 16% del valor del celular.  \* @return el valor del IVA para el celular  \*/  double calcularIva ()  {  double valorCelular = **calcularValor();**  double iva = valorCelular \* 0.16;  return iva;  } |
| --- |

En general, siempre que dentro de una clase se desee hacer referencia a sus propios elementos (atributos y métodos) se puede usar la variable de referencia *this*. El uso de la variable de referencia *this* da mayor claridad en los programas.

| Esta variable representa siempre la instancia o el objeto “actual” dentro de la misma clase. |
| --- |

Por ejemplo, en el primer código que se presentó en este capítulo, del método “calcularValor”, se puede escribir de la siguiente forma:

| /\*\*  \* Calcular el valor del celular, así:  \* Los de tipo 'c' valen 50.000, los de tipo 'b' 130.0000  \* y los de de tipo 'a' 250.000  \* @return el valor del celular  \*/  double calcularValor()  {  double valor = 0;  switch (**this.tipo**)  {  case 'a': valor = 250.000;  break;  case 'b': valor = 130.000;  break;  case 'c': valor = 50.000;  break;  }  return valor;  } |
| --- |

**8.2. Diferenciar atributos y parámetros**

Una de las principales utilidades de la variable de referencia *this* es que permite diferenciar entre atributos de la clase y parámetros de un método.

En Java es válido que un parámetro se llame igual que un atributo, y precisamente para evitar confusiones se usa *this*.

74

Por ejemplo, continuando con el ejemplo de la clase Celular, el método “setMarca” recibe un parámetro llamado “marca”, que es igual al nombre del atributo. El código, aparentemente, sería:

| /\*\*  \* Establecer o modificar la marca del celular  \* @param marca la marca del celular  \*/  void setMarca(String marca)  {  **Este código es incorrecto**.  **marca = marca;**  } |
| --- |

El código anterior es incorrecto, porque Java identifica la variable como la de menor alcance, es decir, que AMBAS las toma como el parámetro. Por lo tanto, el código anterior no hace nada, solo asigna el valor del parámetro a él mismo, y NO se cambia la marca del celular.

Para evitar ese error, y lograr que el método efectivamente asigne el nuevo valor de la marca al celular, se usa *this* en el atributo, como se observa a continuación:

| /\*\*  \* Establecer o modificar la marca del celular  \* @param marca la marca del celular  \*/  void setMarca(String marca)  {  **this.marca** = marca;  } |
| --- |

De esta manera, el valor o la referencia del parámetro es asignado al atributo, es decir, se cambia la marca del celular, que es lo que se desea en este método.

De manera similar, en el método “setTipo” de la clase Celular se recibe un parámetro llamado tipo, que es el mismo nombre del atributo. Por lo tanto, para evitar confusiones, se debe usar la referencia *this*. El código queda:

75

| /\*\*  \* Definir el tipo del celular, el cual puede ser: a, b o c \* Si no se ingresa un tipo válido se define como tipo c.  \* @param tipo el nuevo tipo del celular  \*/  void setTipo(char tipo)  {  if (tipo == 'a' || tipo == 'b' || tipo == 'c')  {  Este es el parámetro  **this.tipo** = tipo;  }  else  Este es el atributo  {  **this.tipo** = 'c';  }  } |
| --- |

De esta forma, en los métodos se pueden usar parámetros que tengan el mismo nombre que los atributos de la clase, pero se deben diferenciar internamente con el *this*.

El código completo de la clase Celular es:

| /\*\*  \* Aparato de comunicación, entretenimiento y productividad, conocido \* como celular o móvil.  \* @author Sandra V. Hurtado  \* @version 1.0  \*/  public class Celular  {  String marca;  char tipo;  /\*\*  \* Establecer o modificar la marca del celular  \* @param marca la marca del celular  \*/  void setMarca(String marca)  {  this.marca = marca;  }  *//continúa* |
| --- |

76

| /\*\*  \* Definir el tipo del celular, el cual puede ser: a, b o c \* Si no se ingresa un tipo válido se define como tipo c. \* @param tipo el nuevo tipo del celular  \*/  void setTipo(char tipo)  {  if (tipo == 'a' || tipo == 'b' || tipo == 'c')  {  this.tipo = tipo;  }  else  {  this.tipo = 'c';  }  }  /\*\*  \* Calcular el valor del celular, así:  \* Los de tipo 'c' valen 50.000, los de tipo 'b' 130.0000 \* y los de de tipo 'a' 250.000  \* @return el valor del celular  \*/  double calcularValor()  {  double valor = 0;  switch (this.tipo)  {  case 'a': valor = 250.000;  break;  case 'b': valor = 130.000;  break;  case 'c': valor = 50.000;  break;  }  return valor;  }  /\*\*  \* Calcula el IVA, que es el 16% del valor del celular. \* @return el valor del IVA para el celular  \*/  double calcularIva ()  {  double valorCelular = calcularValor();  double iva = valorCelular \* 0.16;  return iva;  }  } *// fin clase Celular* |
| --- |

77

***Ejercicio 8-1***

Escriba el código Java para la siguiente clase, teniendo en cuenta que el valor de un lote es de $30.000 por metro cuadrado. Use la referencia *this* donde sea pertinente.

**Lote**

largo: double

ancho: double

setLargo(largo: double): void

setAncho(ancho: double): void

calcularÁrea(): double

calcularValor(): double

**8.3. Ejercicio resuelto**

Se tiene la clase FrascoDeDulce, donde se han cambiado el nombre de un parámetro:

**FrascoDeDulce**

sabor: String

cantidad: int

fecha: String

envasarDulce(cantidadEnvasar: int, fecha: String): boolean

sacarDulce(cantidadSacar: int): int

consultarCantidad(): int

Se muestra el código de esta clase FrascoDeDulce, haciendo uso de la referencia *this* en los métodos.

| /\*\*  \* Un frasco con delicioso dulce, elaborado por la abuela pata. \* Por ejemplo, un frasco de dulce de guayaba.  \* @author Sandra V. Hurtado  \* @version 2.0  \*/  public class FrascoDeDulce  {  String sabor;  int cantidad;  String fecha;  *//continúa* |
| --- |

78

/\*\*

\* Envasar o insertar dulce en el frasco, hasta máximo 200 gramos. \* @param cantidadEnvasar la cantidad, en gramos, que se desea envasar \* @param fecha la fecha en la cual se envasa el dulce \* @return una indicación (true/false) de si se pudo envasar o no \* toda la cantidad deseada o solo 200 gramos \*/

boolean envasarDulce(int cantidadEnvasar, String fecha) {

**this.fecha** = fecha;

if (cantidadEnvasar > 200)

{

**this**.cantidad = 200;

return false;

}

else

{

**this**.cantidad = cantidadEnvasar;

return true;

}

}

/\*\*

\* Saca cierta cantidad del dulce del frasco, si hay suficiente \* en el frasco; pero si hay menos solo se puede sacar lo que queda. \* @param cantidadSacar la cantidad, en gramos, que se desea sacar \* @return la cantidad de dulce que se pudo sacar, en gramos \*/

int sacarDulce(int cantidadSacar)

{

int cantidadSacada = 0;

if (**this**.cantidad >= cantidadSacar)

{

**this**.cantidad = **this**.cantidad - cantidadSacar; cantidadSacada = cantidadSacar;

}

else

{

cantidadSacada = this.cantidad;

**this**.cantidad = 0;

}

return cantidadSacada;

}

/\*\*

\* Consulta la cantidad de dulce del frasco

\* @return la cantidad de dulce que hay en el frasco, en gramos \*/

int consultarCantidad()

{

return **this**.cantidad;

}

} *// fin clase FrascoDeDulce*

79