Capítulo 2
Clases y objetos

El propósito general de este capítulo es hacer una introducción a los conceptos de clases y objetos que son el núcleo de la programación orientada a objetos en Java. Para ello, se presentan once ejercicios que cubren las nociones básicas del paradigma orientado a objetos que permitirán desarrollar programas, basados en una única clase, pero que incluyen conceptos fundamentales orientados a objetos.

- **Ejercicio 2.1. Definición de clases**

  La programación orientada a objetos proporciona varios conceptos y características para hacer que la creación y el uso de objetos sean fáciles y flexibles. Los conceptos más importantes son la clase y, sus instancias, los objetos. El primer ejercicio aborda la definición de clases y la creación de objetos.

- **Ejercicio 2.2. Definición de atributos de una clase con tipos primitivos de datos**

  Las clases se caracterizan por poseer atributos que las describen. Dichos atributos tienen asociado un cierto tipo primitivo de dato. El segundo ejercicio aplica la definición de atributos para una clase utilizando tipos primitivos de datos.

- **Ejercicio 2.3. Estado de un objeto**
  La colección de atributos con sus valores respectivos para un objeto en particular define el estado del objeto en un momento dado. La consulta del estado de un objeto puede hacerse utilizando el método *get*. A su vez, la modificación del estado del objeto puede realizarse a través del método *set*. El tercer ejercicio propone un problema para comprender el concepto de estado de un objeto.

- **Ejercicio 2.4. Definición de métodos con y sin valores de retorno**
  El segundo elemento más importante de una clase, además de sus atributos, son sus métodos. Los métodos permiten agregar diferentes niveles de comportamiento a los objetos. Estos métodos pueden tener o no un valor de retorno. El cuarto ejercicio aplica el concepto de definición de métodos con y sin valor de retorno.

- **Ejercicio 2.5. Definición de métodos con parámetros**
  Los métodos de una clase pueden contener una lista de parámetros, la cual es un conjunto de declaraciones de variables, separadas por comas, dentro de paréntesis. Estos parámetros se convierten en variables locales en el cuerpo del método. El objetivo del quinto ejercicio es comprender la definición de métodos con parámetros.

- **Ejercicio 2.6. Objetos como parámetros**
  Entre los parámetros de un método se pueden incluir, además de tipos primitivos de datos, instancias concretas de clases, es decir, objetos. El comportamiento de los objetos pasados como parámetros difiere de los tipos primitivos de datos pasados como parámetros. El sexto ejercicio hace énfasis en dichas diferencias.

- **Ejercicio 2.7. Métodos de acceso**
  Tanto los atributos como los métodos de una clase tienen un cierto ámbito de aplicación que permite que sean consultados y modificados por otras clases de acuerdo con los métodos de acceso definidos. El séptimo ejercicio pretende abarcar los diferentes métodos de acceso que se pueden asignar a los atributos y métodos de una clase.

- **Ejercicio 2.8. Asignación de objetos**
  Los objetos creados durante la ejecución de los programas pueden ser asignados a diferentes variables. Sin embargo, hay que considerar

que dichas variables no son objetos independientes, ya que comparten el mismo objeto. Para entender estos conceptos se plantea el octavo ejercicio.

- **Ejercicio 2.9. Variables locales dentro de un método**
  Cada método de una clase tiene un ámbito de ejecución delimitado tanto por sus parámetros como por las variables que se definen en el cuerpo del método. El noveno ejercicio pone en práctica la definición de variables locales en métodos.

- **Ejercicio 2.10. Sobrecarga de métodos**
  Los métodos de una clase pueden tener el mismo nombre, siempre que cada método tenga una signatura diferente, es decir, valores de retorno o lista de parámetros distintos. A esto se le llama sobrecarga de métodos; el décimo ejercicio está orientado a entender esta característica de la programación orientada a objetos.

- **Ejercicio 2.11. Sobrecarga de constructores**
  Los constructores, métodos especiales que permiten crear objetos, también se pueden sobrecargar. Por consiguiente, se pueden tener varios constructores con el mismo nombre y entre ellos se pueden invocar. El último ejercicio de este capítulo plantea un problema para afianzar dicho concepto.

Los diagramas UML utilizados en este capítulo son de clase y de objetos. Los diagramas de clase modelan la estructura estática de los programas. Los diagramas de clase de los ejercicios presentados generalmente están conformados por una o varias clases e incorporan notaciones para identificar tipos de atributos y métodos. Los diagramas de objeto identifican los objetos creados en el método *main* del programa.

Ejercicio 2.1. Definición de clases

Las clases son modelos del mundo real que capturan la estructura y comportamiento compartidos por una colección de objetos de un mismo tipo (Seidl *et al.*, 2015). Una clase está conformada por sus atributos y métodos. Una clase se define en Java como:

```
class NombreClase {
        lista de atributos
        lista de constructores
        lista de métodos
}
```

Un objeto se considera la instancia de una clase. Para crear un objeto se debe invocar a su constructor, el cual coincide con el nombre de la clase y se debe utilizar la palabra reservada *new*.

```
Clase objeto = new Clase();
```

Los constructores, además de permitir la instancia de objetos, realizan la inicialización de los atributos del objeto. Esto se logra pasando los valores de los atributos como parámetros en la invocación del constructor:

```
Clase nombreClase {
        tipo atributo;
        nombreClase(int parámetro, …) { // Constructor
            this.atributo = parámetro;
            …
        }
}
```

De otro lado, la palabra *this* se utiliza para referirse a los atributos de la clase y en particular, para diferenciar cuando los parámetros del constructor tienen el mismo nombre que los atributos.

El operador . (punto) permite acceder a los distintos atributos y métodos de una clase. El formato de la operación punto es:

```
objeto.atributo;
objeto.metódo();
```

Objetivos de aprendizaje

Al finalizar este ejercicio, el lector tendrá la capacidad para:

- Entender los conceptos: clase, objeto, constructor y la referencia *this*.
- Comprender el significado de los atributos de un objeto.
- Realizar la instanciación de objetos.

Enunciado: clase Persona

Se requiere un programa que modele el concepto de una persona. Una persona posee nombre, apellido, número de documento de identidad y año de nacimiento. La clase debe tener un constructor que inicialice los valores de sus respectivos atributos.

La clase debe incluir los siguientes métodos:

- Definir un método que imprima en pantalla los valores de los atributos del objeto.
- En un método *main* se deben crear dos personas y mostrar los valores de sus atributos en pantalla.

Instrucciones Java del ejercicio

Tabla 2.1. Instrucciones Java del ejercicio 2.1.

Instrucción	Descripción	Formato
this	Palabra clave que se puede usar dentro de un método o constructor una clase. Funciona como una referencia al objeto actual.	*this*.atributo = parámetro;
void	Palabra clave que especifica que un método no tiene un valor de retorno.	*void* nombreMétodo() { }

Solución

Clase: Persona

```
/**
 * Esta clase define objetos de tipo Persona con un nombre, apellidos,
 * número de documento de identidad y año de nacimiento.
 * @version 1.2/2020
 */
public class Persona {

    String nombre; // Atributo que identifica el nombre de una persona
    String apellidos; // Atributo que identifica los apellidos de una persona
```

```
/* Atributo que identifica el número de documento de identidad de
    una persona */
String númeroDocumentoIdentidad;
int añoNacimiento; /* Atributo que identifica el año de nacimiento
    de una persona */

/**
* Constructor de la clase Persona
* @param nombre Parámetro que define el nombre de la persona
* @param apellidos Parámetro que define los apellidos de la persona
* @param númeroDocumentoIdentidad Parámetro que define el
* número del documento de identidad de la persona
* @param añoNacimiento Parámetro que define el año de nacimiento
* de la persona
*/
Persona(String nombre, String apellidos, String númeroDocumento
    Identidad, int añoNacimiento) {
    this.nombre = nombre;
    this.apellidos = apellidos;
    this.númeroDocumentoIdentidad=númeroDocumentoIdentidad;
    this.añoNacimiento = añoNacimiento;
}

/**
* Método que imprime en pantalla los datos de una persona
*/

void imprimir() {
    System.out.println("Nombre = " + nombre);
    System.out.println("Apellidos = " + apellidos);
    System.out.println("Número de documento de identidad = " +
        númeroDocumentoIdentidad);
    System.out.println("Año de nacimiento = " + añoNacimiento);
    System.out.println();
}

/**
* Método main que crea dos personas e imprime sus datos en pantalla
*/
```

```
        public static void main(String args[]) {
            Persona p1 = new Persona("Pedro","Pérez","1053121010",1998);
            Persona p2 = new Persona("Luis","León","1053223344",2001);
            p1.imprimir();
            p2.imprimir();
        }
    }
```

Diagrama de clases

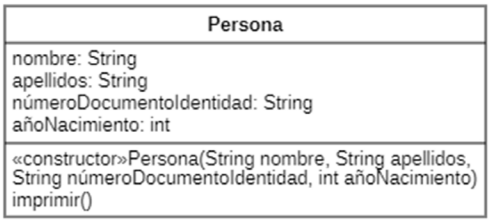

Figura 2.1. Diagrama de clases del ejercicio 2.1.

Explicación del diagrama de clases

Se ha definido una sola clase denominada Persona. El nombre de la clase se ubica en el primer compartimiento de la clase. En el segundo compartimiento se han definido los cuatro atributos junto con su tipo (nombre, apellidos y añoNacimiento de tipo *int* y númeroDocumentoIdentidad de tipo *String*). En el tercer compartimiento se han definido dos métodos, comenzando con el constructor, el cual tiene la etiqueta <<constructor>> como un estereotipo UML (brinda información adicional y personalizada) para su correcta identificación. El otro método es imprimir, el cual no contiene parámetros ni valor de retorno.

Diagrama de objetos

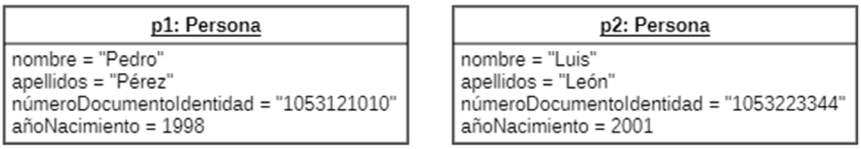

Figura 2.2. Diagrama de objetos del ejercicio 2.1.

Ejecución del programa

```
Nombre = Pedro
Apellidos = Pérez
Número de documento de identidad = 1053121010
Año de nacimiento = 1998

Nombre = Luis
Apellidos = León
Número de documento de identidad = 1053223344
Año de nacimiento = 2001
```

Figura 2.3. Ejecución del programa del ejercicio 2.1.

Ejercicios propuestos

- Agregar dos nuevos atributos a la clase Persona. Un atributo que represente el país de nacimiento de la persona (de tipo *String*) y otro que identifique el género de la persona, el cual debe representarse como un *char* con valores 'H' o 'M'.
- Modificar el constructor de la clase Persona para que inicialice estos dos nuevos atributos.
- Modificar el método imprimir de la clase Persona para que muestre en pantalla los valores de los nuevos atributos.

Ejercicio 2.2. Definición de atributos de una clase con tipos primitivos de datos

Las clases contienen una colección de atributos, que representan propiedades de los objetos. Dichos atributos tienen un tipo, el cual puede ser un tipo primitivo de dato de Java u objeto.

En la tabla 2.2 se presentan los tipos primitivos de datos en Java (Arroyo-Díaz, 2019a).

Tabla 2.2. Tipos primitivos de datos en Java

Tipo	Tamaño	Valores
byte	1 ***byte***	Valor entero entre -128 y 127
short	2 ***bytes***	Valor entero entre -32 768 y 32 767
int	4 ***bytes***	Valor entre 2 147 483 648 y 2 147 483 647
long	8 ***bytes***	Valor entre -9 223 372 036 854 775 808 y 9 223 372 036 854 775 807
float	4 ***bytes***	De -3.402823E38 a -1.401298E-45 y de 1.401298E-45 a 3.402823E38
double	8 ***bytes***	De -1.79769313486232 a -4.94065645841247E-324 y de 4.94065645841247E-324 a 1.79769313486232E308
Char	2 ***bytes***	Caracteres Unicode
boolean	1 ***bytes***	True y False

También existen los datos enumerados que representan un grupo de constantes con valores predefinidos. Se debe utilizar la palabra clave *enum* y separar las constantes con una coma. Los elementos enumerados deben estar en letras mayúsculas. El formato de los datos enumerados es:

enum variable {ELEMENTO1, ELEMENTO2, ELEMENTO3}

En este caso, la variable es un valor enumerado que puede asumir los valores ELEMENTO1, ELEMENTO2 o ELEMENTO3.

Objetivos de aprendizaje

Al finalizar este ejercicio el lector tendrá la capacidad para:

- Definir atributos de una clase con un tipo de primitivo de dato.
- Definir los valores iniciales de los atributos de una clase.
- Definir atributos de tipo enumerado.

Enunciado: clase Planeta

Se requiere un programa que modele el concepto de un planeta del sistema solar. Un planeta tiene los siguientes atributos:

- Un nombre de tipo *Sring* con valor inicial de *null*.
- Cantidad de satélites de tipo *int* con valor inicial de cero.
- Masa en kilogramos de tipo *double* con valor inicial de cero.

- Volumen en kilómetros cúbicos de tipo *double* con valor inicial de cero.
- Diámetro en kilómetros de tipo *int* con valor inicial de cero.
- Distancia media al Sol en millones de kilómetros, de tipo *int* con valor inicial de cero.
- Tipo de planeta de acuerdo con su tamaño, de tipo enumerado con los siguientes valores posibles: GASEOSO, TERRESTRE y ENANO.
- Observable a simple vista, de tipo booleano con valor inicial *false*.

La clase debe incluir los siguientes métodos:

- La clase debe tener un constructor que inicialice los valores de sus respectivos atributos.
- Definir un método que imprima en pantalla los valores de los atributos de un planeta.
- Calcular la densidad un planeta, como el cociente entre su masa y su volumen.
- Determinar si un planeta del sistema solar se considera exterior. Un planeta exterior está situado más allá del cinturón de asteroides. El cinturón de asteroides se encuentra entre 2.1 y 3.4 UA. Una unidad astronómica (UA) es la distancia entre la Tierra y el Sol= 149 597 870 Km.
- En un método *main* se deben crear dos planetas y mostrar los valores de sus atributos en pantalla. Además, se debe imprimir la densidad de cada planeta y si el planeta es un planeta exterior del sistema solar.

Instrucciones Java del ejercicio

Tabla 2.3. Instrucciones Java del ejercicio 2.2.

Instrucción	Descripción	Formato
return	Finaliza la ejecución de un método y puede utilizarse para devolver el valor de un método.	*return*; *return* variable;

Solución

Clase: Planeta

```
/**
 * Esta clase define objetos de tipo Planeta con un nombre, cantidad de
 * satélites, masa, volumen, diámetro, distancia al sol, tipo de planeta y si es
 * observable o no.
 * @version 1.2/2020
 */
public class Planeta {
    // Atributo que define el nombre de un planeta
    String nombre = null;
    // Atributo que define la cantidad de satélites que tiene un planeta
    int cantidadSatélites = 0;
    // Atributo que define la masa de un planeta
    double masa = 0;
    // Atributo que define el volumen de un planeta
    double volumen = 0;
    // Atributo que define el diámetro de un planeta
    int diámetro = 0;
    // Atributo que define la distancia al sol de un planeta
    int distanciaSol = 0;
    // Atributo que define el tipo de planeta como un valor enumerado
    enum tipoPlaneta {GASEOSO, TERRESTRE, ENANO}
    // Atributo que define el tipo de planeta
    tipoPlaneta tipo;
    // Atributo que define si el planeta es observable o no
    boolean esObservable = false;

    /**
    * Constructor de la clase Planeta
    * @param nombre Parámetro que define el nombre del planeta
    * @param cantidadSatélites Parámetro que define la cantidad de
    * satélites del planeta
    * @param masa Parámetro que define la masa del planeta (en
    * kilogramos)
    * @param volumen Parámetro que define el volumen del planeta
    * (en kilómetros cúbicos)
    * @param diámetro Parámetro que define el diámetro del planeta
    * (en kilómetros)
```

```
 * @param distanciaSol Parámetro que define la distancia media del
 * planeta al sol (en kilómetros)
 * @param tipo Parámetro que define el tipo de planeta (puede ser
 * GASEOSO, TERRESTRE o ENANO)
 * @param esObservable Parámetro que define si el planeta es
 * observable o no
 */

Planeta(String nombre, int cantidadSatélites, double masa, double
    volumen, int diámetro, int distanciaSol, tipoPlaneta tipo, boolean
    esObservable) {
    this.nombre = nombre;
    this.cantidadSatélites = cantidadSatélites;
    this.masa = masa;
    this.volumen = volumen;
    this.diámetro = diámetro;
    this.distanciaSol = distanciaSol;
    this.tipo = tipo;
    this.esObservable = esObservable;
}

/**
* Método que imprime en pantalla los datos de un planeta
*/
void imprimir() {
    System.out.println("Nombre del planeta = " + nombre);
    System.out.println("Cantidad de satélites = " + cantidadSatélites);
    System.out.println("Masa del planeta = " + masa);
    System.out.println("Volumen del planeta = " + volumen);
    System.out.println("Diámetro del planeta = " + diámetro);
    System.out.println("Distancia al sol = " + distanciaSol);
    System.out.println("Tipo de planeta = " + tipo);
    System.out.println("Es observable = " + esObservable);
}

/**
* Método que calcula y devuelve la densidad de un planeta
* @return La densidad calculada del planeta
*/

double calcularDensidad() {
    return masa/volumen;
}
```

```
    /**
    * Método que determina y devuelve si un planeta es exterior o no
    * @return Valor booleano que indica si el planeta es exterior o no
    */
    boolean esPlanetaExterior(){
        float límite = (float) (149597870 * 3.4);
        /* Un planeta exterior está situado más allá del cinturón de
            asteroides */
        /* El cinturón se encuentra entre 2,1 y 3,4 UA (UA =
            149.597.870 Km) */
        if (distanciaSol > límite) {
            return true;
        } else {
            return false;
        }
    }
    /**
    * Método main que crea dos planetas, imprime sus datos en pantalla,
    * determina su densidad y si son planetas exteriores
    */
    public static void main(String args[]) {
        Planeta p1 = new Planeta("Tierra",1,5.9736E24,1.0832
            1E12,12742,150000000,tipoPlaneta.TERRESTRE,true);
        p1.imprimir();
        System.out.println("Densidad del planeta = " +
            p1.calcularDensidad());
        System.out.println("Es planeta exterior = " +
            p1.esPlanetaExterior());
        System.out.println();
        Planeta p2 = new Planeta("Júpiter",79,1.899E27,1.431
            3E15,139820,750000000,tipoPlaneta.gaseoso,true);
        p2.imprimir();
        System.out.println("Densidad del planeta = " +
            p2.calcularDensidad());
        System.out.println("Es planeta exterior = " +
            p2.esPlanetaExterior());
    }
}
```

Diagrama de clases

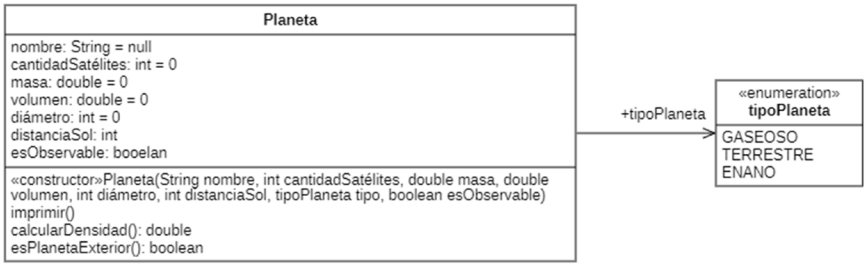

Figura 2.4. Diagrama de clases del ejercicio 2.4.

Explicación del diagrama de clases

Se ha definido una clase denominada Planeta con sus respectivos atributos que representan el nombre del planeta, cantidad de satélites que tiene, su masa, volumen, diámetro, distancia al Sol y si es observable o no. Para cada atributo, además de su tipo, se ha agregado su correspondiente valor inicial o por defecto. También se han definido sus métodos respectivos: un constructor que inicializa los valores de sus atributos, un método para imprimir los valores de sus atributos en pantalla, otro para calcular la densidad (que devuelve un valor de tipo *double*) y un último método para determinar si es un planeta exterior (que devuelve un valor booleano).

El atributo para identificar el tipo de planeta se expresa como una asociación entre la clase Planeta y la clase TipoPlaneta cuyo nombre es el nombre del atributo. En UML, las variables *enum* se identifican con el estereotipo <<enumeration>>, que representan un conjunto de valores constantes identificados como atributos en su segundo compartimiento.

Diagrama de objetos

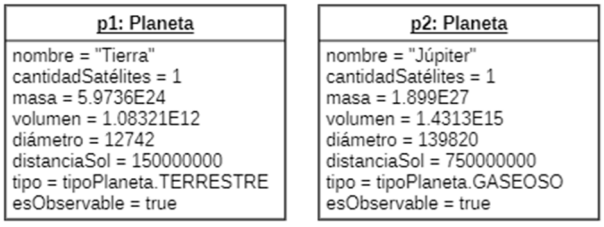

Figura 2.5. Diagrama de objetos del ejercicio 2.2.

Ejecución del programa

```
Nombre del planeta = Tierra
Cantidad de satélites = 1
Masa del planeta = 5.9736E24
Volumen del planeta = 1.08321E12
Diámetro del planeta = 12742
Distancia al sol = 150000000
Tipo de planeta = TERRESTRE
Es observable = true
Densidad del planeta = 5.514720137369484E12
Es planeta exterior = false

Nombre del planeta = Júpiter
Cantidad de satélites = 79
Masa del planeta = 1.899E27
Volumen del planeta = 1.4313E15
Diámetro del planeta = 139820
Distancia al sol = 750000000
Tipo de planeta = GASEOSO
Es observable = true
Densidad del planeta = 1.3267658771745964E12
Es planeta exterior = true
```

Figura 2.6. Ejecución del programa del ejercicio 2.2.

Ejercicios propuestos

- Agregar dos atributos a la clase Planeta. El primero debe representar el periodo orbital del planeta (en años). El segundo atributo representa el periodo de rotación (en días).
- Modificar el constructor de la clase para que inicialice los valores de estos dos nuevos atributos.
- Modificar el método imprimir para que muestre en pantalla los valores de los nuevos atributos.

Ejercicio 2.3. Estado de un objeto

Se denomina estado de un objeto al conjunto de pares *{atributo = valor de un objeto}*. El estado de un objeto puede cambiar a lo largo de su vida a medida que se vayan ejecutando sus métodos (Arroyo-Díaz, 2019a).

Las clases tienen dos tipos de métodos: *get* y *set*.

- Los métodos *get* permiten obtener el valor de un atributo de un objeto. Los métodos *get* se definen con el siguiente formato:

getNombreAtributo

- Los métodos *set* permiten asignar o cambiar el valor de un atributo de un objeto y, por lo tanto, cambian su estado. Los métodos *set* se definen con el siguiente formato:

setNombreAtributo(tipo parámetro)

Los métodos *get* y *set* tienen una estructura que se repite para cada uno de los atributos. Por ello, los entornos de desarrollo actuales permiten generar automáticamente dichos métodos.

Objetivos de aprendizaje

Al finalizar este ejercicio, el lector tendrá la capacidad para:

- Entender el concepto de estado de un objeto.
- Definir métodos *get* y *set* de una clase.

Enunciado: clase Automóvil

Se requiere un programa que modele el concepto de un automóvil. Un automóvil tiene los siguientes atributos:

- Marca: el nombre del fabricante.
- Modelo: año de fabricación.
- Motor: volumen en litros del cilindraje del motor de un automóvil.
- Tipo de combustible: valor enumerado con los posibles valores de gasolina, bioetanol, diésel, biodiésel, gas natural.
- Tipo de automóvil: valor enumerado con los posibles valores de carro de ciudad, subcompacto, compacto, familiar, ejecutivo, SUV.
- Número de puertas: cantidad de puertas.
- Cantidad de asientos: número de asientos disponibles que tiene el vehículo.
- Velocidad máxima: velocidad máxima sostenida por el vehículo en km/h.
- Color: valor enumerado con los posibles valores de blanco, negro, rojo, naranja, amarillo, verde, azul, violeta.
- Velocidad actual: velocidad del vehículo en un momento dado.

La clase debe incluir los siguientes métodos:

- Un constructor para la clase Automóvil donde se le pasen como parámetros los valores de sus atributos.
- Métodos *get* y *set* para la clase Automóvil.
- Métodos para acelerar una cierta velocidad, desacelerar y frenar (colocar la velocidad actual en cero). Es importante tener en cuenta que no se debe acelerar más allá de la velocidad máxima permitida para el automóvil. De igual manera, tampoco es posible desacelerar a una velocidad negativa. Si se cumplen estos casos, se debe mostrar por pantalla los mensajes correspondientes.
- Un método para calcular el tiempo estimado de llegada, utilizando como parámetro la distancia a recorrer en kilómetros. El tiempo estimado se calcula como el cociente entre la distancia a recorrer y la velocidad actual.
- Un método para mostrar los valores de los atributos de un Automóvil en pantalla.
- Un método *main* donde se deben crear un automóvil, colocar su velocidad actual en 100 km/h, aumentar su velocidad en 20 km/h, luego decrementar su velocidad en 50 km/h, y después frenar. Con cada cambio de velocidad, se debe mostrar en pantalla la velocidad actual.

Solución

Clase: Automóvil

```
/**
 * Esta clase define objetos de tipo Automóvil con una marca, modelo,
 * motor, tipo de combustible, tipo de automóvil, número de puertas,
 * cantidad de asientos, velocidad máxima, color y velocidad actual.
 * @version 1.2/2020
 */
public class Automóvil {
    // Atributo que define la marca de un automóvil
```

```
String marca;
// Atributo que define el modelo de un automóvil
int modelo;
// Atributo que define el motor de un automóvil
int motor;
// Tipo de combustible como un valor enumerado
enum tipoCom {GASOLINA, BIOETANOL, DIESEL, BIODISESEL,
    GAS_NATURAL}
// Atributo que define el tipo de combustible
tipoCom tipoCombustible;
// Tipo de automóvil como un valor enumerado
enum tipoA {CIUDAD, SUBCOMPACTO, COMPACTO, FAMILIAR,
    EJECUTIVO, SUV}
// Atributo que define el tipo de automóvil
tipoA tipoAutomóvil;
// Atributo que define el número de puertas de un automóvil
int númeroPuertas;
// Atributo que define la cantidad de asientos de un automóvil
int cantidadAsientos;
// Atributo que define la velocidad máxima de un automóvil
int velocidadMáxima;
// Color del automóvil como un valor enumerado
enum tipoColor {BLANCO, NEGRO, ROJO, NARANJA,
    AMARILLO, VERDE, AZUL, VIOLETA}
// Atributo que define el color de un automóvil
tipoColor color;
// Atributo que define la velocidad de un automóvil
int velocidadActual = 0;

/**
* Constructor de la clase Automóvil
* @param marca Parámetro que define la marca de un automóvil
* @param modelo Parámetro que define el modelo (año de
*fabricación) de un automóvil
* @param motor Parámetro que define el volumen del cilindraje del
* motor (puede ser gasolina, bioetanol, diésel, biodiesel o gas natural)
* @param tipoAutomóvil Parámetro que define el tipo de automóvil
* (puede ser Carro de ciudad, Subcompacto, Compacto, Familiar,
* Ejecutivo o SUV)
* @param númeroPuertas Parámetro que define el número de
* puertas de un automóvil
```

```
* @param cantidadAsientos Parámetro que define la cantidad de
* asientos que tiene el automóvil
* @param velocidadMáxima Parámetro que define la velocidad
* máxima permitida al automóvil
* @param color Parámetro que define el color del automóvil (puede
* ser Blanco, Negro, Rojo, Naranja, Amarillo, Verde, Azul o Violeta)
*/

Automóvil(String marca, int modelo, int motor, tipoCom
    tipoCombustible, tipoA tipoAutomóvil, int númeroPuertas, int
    cantidadAsientos, int velocidadMáxima, tipoColor color) {
    this.marca = marca;
    this.modelo = modelo;
    this.motor = motor;
    this.tipoCombustible = tipoCombustible;
    this.tipoAutomóvil = tipoAutomóvil;
    this.númeroPuertas = númeroPuertas;
    this.cantidadAsientos = cantidadAsientos;
    this.velocidadMáxima = velocidadMáxima;
    this.color = color;
}

/**
* Método que devuelve la marca de un automóvil
* @return La marca de un automóvil
*/

String getMarca() {
    return marca;
}

/**
* Método que devuelve el modelo de un automóvil
* @return El modelo de un automóvil
*/
int getModelo() {
    return modelo;
}
```

```
    /**
    * Método que devuelve el volumen en litros del cilindraje del motor
    * de un automóvil
    * @return El volumen en litros del cilindraje del motor de un
    * automóvil
    */
    int getMotor() {
        return motor;
    }

    /**
    * Método que devuelve el tipo de combustible utilizado por el motor
    * de un automóvil
    * @return El tipo de combustible utilizado por el motor de un
    * automóvil
    */
    tipoCom getTipoCombustible() {
        return tipoCombustible;
    }

    /**
    * Método que devuelve el tipo de automóvil
    * @return El tipo de automóvil
    */
    tipoA getTipoAutomóvil() {
        return tipoAutomóvil;
    }

    /**
    * Método que devuelve el número de puertas de un automóvil
    * @return El número de puertas que tiene un automóvil
    */
    int getNúmeroPuertas() {
        return númeroPuertas;
}

    /**
    * Método que devuelve la cantidad de asientos de un automóvil
    * @return La cantidad de asientos que tiene un automóvil
    */
    int getCantidadAsientos() {
        return cantidadAsientos;
    }
```

```
/**
* Método que devuelve la velocidad máxima de un automóvil
* @return La velocidad máxima de un automóvil
*/
int getVelocidadMáxima() {
    return velocidadMáxima;
}

/**
* Método que devuelve el color de un automóvil
* @return El color de un automóvil
*/
tipoColor getColor() {
    return color;
}

/**
* Método que devuelve la velocidad actual de un automóvil
* @return La velocidad actual de un automóvil
*/
int getVelocidadActual() {
    return velocidadActual;
}

/**
* Método que establece la marca de un automóvil
* @param marca Parámetro que define la marca de un automóvil
*/
void setMarca(String marca) {
    this.marca = marca;
}

/**
* Método que establece el modelo de un automóvil
* @param modelo Parámetro que define el modelo de un automóvil
*/
void setModelo(int modelo) {
    this.modelo = modelo;
}

/**
* Método que establece el volumen en litros del motor de un automóvil
* @param motor Parámetro que define el volumen en litros del
* motor de un automóvil
*/
```

```
void setMotor(int motor) {
    this.motor = motor;
}

/**
* Método que establece el tipo de combustible de un automóvil
* @param tipoCombustible Parámetro que define el tipo de
* combustible de un automóvil
*/
void setTipoCombustible(tipoCom tipoCombustible) {
    this.tipoCombustible = tipoCombustible;
}

/**
* Método que establece el tipo de automóvil
* @param tipoAutomóvil Parámetro que define el tipo de automóvil
*/
void setTipoAutomóvil(tipoA tipoAutomóvil) {
    this.tipoAutomóvil = tipoAutomóvil;
}

/**
* Método que establece el número de puertas de un automóvil
* @param númeroPuertas Parámetro que define el número de
* puertas de un automóvil
*/
void setNúmeroPuertas(int númeroPuertas) {
    this.númeroPuertas = númeroPuertas;
}

/**
* Método que establece la cantidad de asientos de un automóvil
* @param cantidadAsientos Parámetro que define la cantidad de
* asientos de un automóvil
*/
void setCantidadAsientos(int cantidadAsientos) {
    this.cantidadAsientos = cantidadAsientos;
}

/**
* Método que establece la velocidad máxima de un automóvil
* @param velocidadMáxima Parámetro que define la velocidad
* máxima de un automóvil
```

```
*/
void setVelocidadMáxima(int velocidadMáxima) {
    this.velocidadMáxima = velocidadMáxima;
}

/**
* Método que establece el color de un automóvil
* @param color Parámetro que define el color de un automóvil
*/
void setColor(tipoColor color) {
    this.color = color;
}

/**
* Método que establece la velocidad de un automóvil
* @param velocidadActual Parámetro que define la velocidad actual
* de un automóvil
*/
void setVelocidadActual(int velocidadActual) {
    this.velocidadActual = velocidadActual;
}

/**
* Método que incrementa la velocidad de un automóvil
* @param incrementoVelocidad Parámetro que define la cantidad a
* incrementar en la velocidad actual de un automóvil
*/
void acelerar(int incrementoVelocidad) {
    if (velocidadActual + incrementoVelocidad < velocidadMáxima) {
        /* Si el incremento de velocidad no supera la velocidad
            máxima */
        velocidadActual = velocidadActual + incrementoVelocidad;
    } else { /* De otra manera no se puede incrementar la velocidad y
        se genera mensaje */
    System.out.println("No se puede incrementar a una velocidad
        superior a la máxima del automóvil.");
    }
}

/**
* Método que decrementa la velocidad de un automóvil
* @param marca Parámetro que define la cantidad a decrementar en
* la velocidad actual de un automovil
*/
```

```
    void desacelerar(int decrementoVelocidad) {
        /* La velocidad actual no se puede decrementar alcanzando un
           valor negativo */
        if ((velocidadActual - decrementoVelocidad) > 0) {
            velocidadActual = velocidadActual - decrementoVelocidad;
        } else { /* De otra manera no se puede decrementar la velocidad y
           se genera mensaje */
            System.out.println("No se puede decrementar a una velocidad
               negativa.");
        }
    }

    /**
    * Método que coloca la velocidad actual de un automóvil en cero
    */
    void frenar() {
        velocidadActual = 0;
    }

    /**
    * Método que calcula el tiempo que tarda un automóvil en recorrer
    * cierta distancia
    * @param distancia Parámetro que define la distancia a recorrer por
    * el automóvil (en kilómetros)
    */
    double calcularTiempoLlegada(int distancia) {
        return distancia/velocidadActual;
    }

    /**
    * Método que imprime en pantalla los valores de los atributos de un
        automóvil
    */
    void imprimir() {
        System.out.println("Marca = " + marca);
        System.out.println("Modelo = " + modelo);
        System.out.println("Motor = " + motor);
        System.out.println("Tipo de combustible = " + tipoCombustible);
        System.out.println("Tipo de automóvil = " + tipoAutomóvil);
        System.out.println("Número de puertas = " + númeroPuertas);
```

```
        System.out.println("Cantidad de asientos = " +
            cantidadAsientos);
        System.out.println("Velocida máxima = " + velocidadMáxima);
        System.out.println("Color = " + color);
    }

    /**
    * Método main que crea un automóvil, imprime sus datos en
    * pantalla y realiza varios cambios en su velocidad
    */
    public static void main(String args[]) {
        Automóvil auto1 = new
Automóvil("Ford",2018,3,tipoCom.DIESEL,tipoA.EJECUTIVO,5,6,250,
            tipoColor.NEGRO);
        auto1.imprimir();
        auto1.setVelocidadActual(100);
        System.out.println("Velocidad actual = " + auto1.
            velocidadActual);
        auto1.acelerar(20);
        System.out.println("Velocidad actual = " + auto1.
            velocidadActual);
        auto1.desacelerar(50);
        System.out.println("Velocidad actual = " + auto1.
            velocidadActual);
        auto1.frenar();
        System.out.println("Velocidad actual = " + auto1.
            velocidadActual);
        auto1.desacelerar(20);
    }
}
```

Diagrama de clases

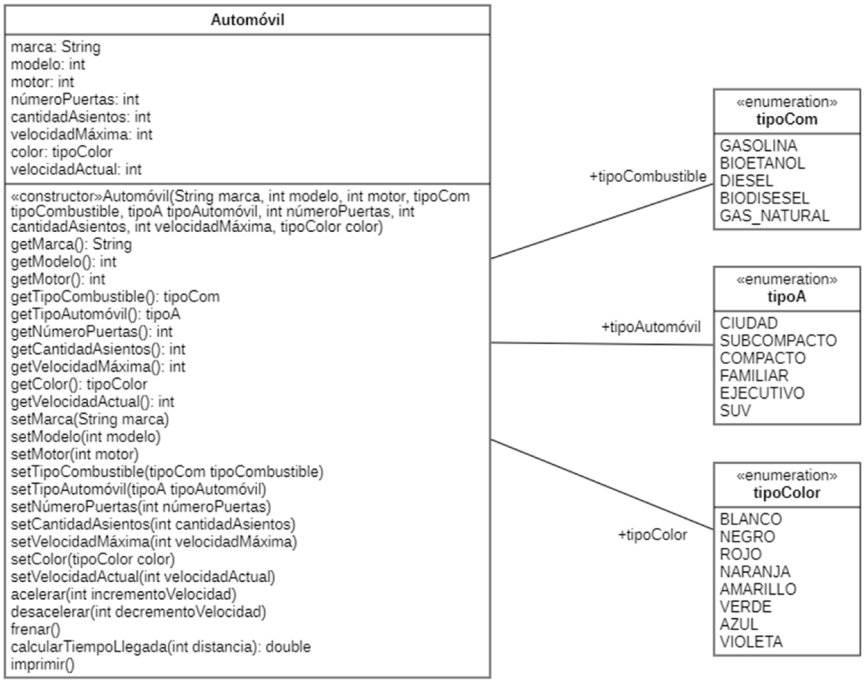

Figura 2.7. Diagrama de clases del ejercicio 2.3.

Explicación del diagrama de clases

Se ha definido una clase denominada Automóvil con sus respectivos atributos y métodos. El compartimiento de métodos incluye los métodos *get* y *set* para cada atributo de la clase. Los atributos de tipo enumerado se modelan como asociaciones entre la clase Automóvil y diferentes clases independientes con el estereotipo <<enumeration>>, cada una de ellas con su respectiva lista de constantes definidas en el compartimiento de atributos de la clase.

La clase Automóvil posee atributos como: marca (de tipo *String*), modelo (de tipo *int*), motor (de tipo *int*), tipo de combustible (valor enumerado que se representa con la asociación a la clase tipoCom de tipo <<enumaration>>), tipo de automóvil (valor enumerado que se representa con la asociación a la clase tipoA de tipo <<enumaration>>), número de puertas (de tipo *int*), cantidad de asientos (de tipo *int*), número de asientos

disponibles que tiene el vehículo (de tipo *int*), velocidad máxima (de tipo *int*), color (valor enumerado que se representa con la asociación a la clase tipoColor de tipo <<enumaration>>) y velocidad actual (de tipo *int*).

La clase Automóvil cuenta además de los métodos *get* y *set*, con un constructor para inicializar los valores de todos sus atributos y métodos para acelerar y desacelerar a cierta velocidad, frenar el automóvil, calcular tiempo de llegada para recorrer una cierta distancia y para imprimir los valores de los atributos en pantalla.

Diagrama de objetos

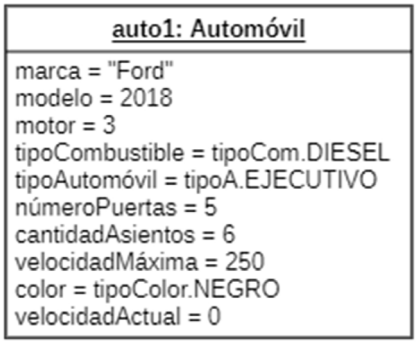

Figura 2.8. Diagrama de objetos del ejercicio 2.3.

Ejecución del programa

```
Marca = Ford
Modelo = 2018
Motor = 3
Tipo de combustible = DIESEL
Tipo de automóvil = EJECUTIVO
Número de puertas = 5
Cantidad de asientos = 6
Velocida máxima = 250
Color = NEGRO
Velocidad actual = 100
Velocidad actual = 120
Velocidad actual = 70
Velocidad actual = 0
No se puede decrementar a una velocidad negativa.
```

Figura 2.9. Ejecución del programa del ejercicio 2.3.

Ejercicios propuestos

- Agregar a la clase Automóvil, un atributo para determinar si el vehículo es automático o no. Agregar los métodos *get* y *set* para dicho atributo. Modificar el constructor para inicializar dicho atributo.
- Modificar el método acelerar para que si la velocidad máxima se sobrepase se genere una multa. Dicha multa se puede incrementar cada vez que el vehículo intenta superar la velocidad máxima permitida.
- Agregar un método para determinar si un vehículo tiene multas y otro método para determinar el valor total de multas de un vehículo.

Ejercicio 2.4. Definición de métodos con y sin valores de retorno

Los métodos de una clase con y sin valores de retorno tienen el siguiente formato, el cual se denomina signatura del método (Deitel y Deitel, 2017):

tipoRetorno nombreMétodo(tipo parámetro1, tipo parámetro2, …)

- **tipoRetorno**: representa el tipo de retorno del método, el cual puede ser un tipo de dato primitivo, un objeto, o si no devuelve nada debe colocarse la palabra reservada *void*.
- **nombreMétodo**: representa el nombre del método.
- **Parámetros**: es un listado de parámetros separados por comas, que se utilizan en el cuerpo del método para cumplir la finalidad del método. Para cada parámetro se debe colocar previamente su tipo de dato. Un método puede no tener parámetros.

Objetivos de aprendizaje

Al finalizar este ejercicio, el lector tendrá la capacidad para:

- Definir métodos con y sin valores de retorno en una clase.
- Definir un programa conformado por varias clases y una clase de prueba que contenga el método *main*.

Enunciado: clases sobre figuras geométricas

Se requiere un programa que modele varias figuras geométricas: el círculo, el rectángulo, el cuadrado y el triángulo rectángulo.

- El círculo tiene como atributo su radio en centímetros.
- El rectángulo, su base y altura en centímetros.
- El cuadrado, la longitud de sus lados en centímetros.
- El triángulo, su base y altura en centímetros.

Se requieren métodos para determinar el área y el perímetro de cada figura geométrica. Además, para el triángulo rectángulo se requiere:

- Un método que calcule la hipotenusa del rectángulo.
- Un método para determinar qué tipo de triángulo es:

  Equilátero: todos sus lados son iguales.

  Isósceles: tiene dos lados iguales.

  Escaleno: todos sus lados son diferentes.

Se debe desarrollar una clase de prueba con un método *main* para crear las cuatro figuras y probar los métodos respectivos.

Instrucciones Java del ejercicio

Tabla 2.4. Instrucciones Java del ejercicio 2.4.

Instrucción	Descripción	Formato
Math.PI	Un valor *double* que representa la variable matemática pi = 3.1416, la relación entre el radio de la circunferencia y su diámetro.	Math.PI
Math.pow	Método que devuelve el valor del primer argumento elevado a la potencia del segundo argumento.	*Math.pow*(argumento1, argumento2);
&&	Operador lógico AND. Devuelve solamente un valor verdadero cuando las dos expresiones son verdaderas, en caso contrario devuelve falso.	(expresión) && (expresión2)

Solución

Clase: Círculo

```
/**
 * Esta clase define objetos de tipo Círculo con su radio como atributo.
 * @version 1.2/2020
 */
public class Círculo {
    int radio; // Atributo que define el radio de un círculo

    /**
    * Constructor de la clase Círculo
    * @param radio Parámetro que define el radio de un círculo
    */
    Círculo(int radio) {
        this.radio = radio;
    }

    /**
    * Método que calcula y devuelve el área de un círculo como pi
    * multiplicado por el radio al cuadrado
    * @return Área de un círculo
    */
    double calcularArea() {
        return Math.PI*Math.pow(radio,2);
    }

    /**
    * Método que calcula y devuelve el perímetro de un círculo como la
    * multiplicación de pi por el radio por 2
    * @return Perímetro de un círculo
    */
    double calcularPerímetro() {
        return 2*Math.PI*radio;
    }
}
```

Clase: Rectángulo

```
/**
* Esta clase define objetos de tipo Rectángulo con una base y una
* altura como atributos.
* @version 1.2/2020
*/
public class Rectángulo {
    int base; // Atributo que define la base de un rectángulo
    int altura; // Atributo que define la altura de un rectángulo

    /**
    * Constructor de la clase Rectangulo
    * @param base Parámetro que define la base de un rectángulo
    * @param altura Parámetro que define la altura de un rectángulo
    */
    Rectángulo(int base, int altura) {
        this.base = base;
        this.altura = altura;
    }

    /**
    * Método que calcula y devuelve el área de un rectángulo como la
    * multiplicación de la base por la altura
    * @return Área de un rectángulo
    */
    double calcularArea() {
        return base * altura;
    }

    /**
    * Método que calcula y devuelve el perímetro de un rectángulo
    * como (2 * base) + (2 * altura)
    * @return Perímetro de un rectángulo
    */
    double calcularPerímetro() {
        return (2 * base) + (2 * altura);
    }
}
```

Clase: Cuadrado

```
/**
 * Esta clase define objetos de tipo Cuadrado con un lado como atributo.
 * @version 1.2/2020
 */
public class Cuadrado {
    int lado; // Atributo que define el lado de un cuadrado

    /**
    * Constructor de la clase Cuadrado
    * @param lado Parámetro que define la longitud de la base de un
    * cuadrado
    */
    public Cuadrado(int lado) {
        this.lado = lado;
    }

    /**
    * Método que calcula y devuelve el área de un cuadrado como el
    * lado elevado al cuadrado
    * @return Área de un Cuadrado
    */
    double calcularArea() {
        return lado*lado;
    }

    /**
    * Método que calcula y devuelve el perímetro de un cuadrado como
    * cuatro veces su lado
    * @return Perímetro de un cuadrado
    */
    double calcularPerímetro() {
        return (4*lado);
    }
}
```

Clase: TriánguloRectángulo

```
/**
* Esta clase define objetos de tipo Triángulo Rectángulo con una
* base y una altura como atributos.
* @version 1.2/2020
*/
public class TriánguloRectángulo {
    int base; // Atributo que define la base de un triángulo rectángulo
    int altura; // Atributo que define la altura de un triángulo rectángulo

    /**
    * Constructor de la clase TriánguloRectángulo
    * @param base Parámetro que define la base de un triángulo
    * rectángulo
    * @param altura Parámetro que define la altura de un triángulo
    * rectángulo
    */
    public TriánguloRectángulo(int base, int altura) {
        this.base = base;
        this.altura = altura;
    }

    /**
    * Método que calcula y devuelve el área de un triángulo rectángulo
    * como la base multiplicada por la altura sobre 2
    * @return Área de un triángulo rectángulo
    */
    double calcularArea() {
        return (base * altura / 2);
    }

    /**
    * Método que calcula y devuelve el perímetro de un triángulo
    * rectángulo como la suma de la base, la altura y la hipotenusa
    * @return Perímetro de un triángulo rectángulo
    */
    double calcularPerímetro() {
        return (base + altura + calcularHipotenusa()); /* Invoca al
            método calcular hipotenusa */
    }
```

```
    /**
    * Método que calcula y devuelve la hipotenusa de un triángulo
    * rectángulo utilizando el teorema de Pitágoras
    * @return Hipotenusa de un triángulo rectángulo
    */
    double calcularHipotenusa() {
        return Math.pow(base*base + altura*altura, 0.5);
    }

    /**
    * Método que determina si un triángulo es:
    * - Equilatero: si sus tres lados son iguales
    * - Escaleno: si sus tres lados son todos diferentes
    * - Escaleno: si dos de sus lados son iguales y el otro es diferente de
    * los demás
    */
    void determinarTipoTriángulo() {
        if ((base == altura) && (base == calcularHipotenusa()) && (altura
            == calcularHipotenusa()))
            System.out.println("Es un triángulo equilátero"); /* Todos sus
                lados son iguales */
        else if ((base != altura) && (base != calcularHipotenusa()) &&
            (altura != calcularHipotenusa()))
            System.out.println("Es un triángulo escaleno"); /* Todos sus
                lados son diferentes */
        else
            System.out.println("Es un triángulo isósceles"); /* De otra
                manera, es isósceles */
        }
}
```

Clase: PruebaFiguras

```
/**
 * Esta clase prueba diferentes acciones realizadas en diversas figuras
 * geométricas.
 * @version 1.2/2020
 */
public class PruebaFiguras {
```

```
    /**
    * Método main que crea un círculo, un rectángulo, un cuadrado y
    * un triángulo rectángulo. Para cada uno de estas figuras geométricas,
    * se calcula su área y perímetro.
    */
    public static void main(String args[]) {
        Círculo figura1 = new Círculo(2);
        Rectángulo figura2 = new Rectángulo(1,2);
        Cuadrado figura3 = new Cuadrado(3);
        TriánguloRectángulo figura4 = new TriánguloRectángulo(3,5);

        System.out.println("El área del círculo es = " + figura1.
            calcularArea());
        System.out.println("El perímetro del círculo es = " + figura1.
            calcularPerímetro());
        System.out.println();
        System.out.println("El área del rectángulo es = " + figura2.
            calcularArea());
        System.out.println("El perímetro del rectángulo es = " + figura2.
            calcularPerímetro());
        System.out.println();
        System.out.println("El área del cuadrado es = " + figura3.
            calcularArea());
        System.out.println("El perímetro del cuadrado es = " + figura3.
            calcularPerímetro());
        System.out.println();
        System.out.println("El área del triángulo es = " + figura4.
            calcularArea());
        System.out.println("El perímetro del triángulo es = " + figura4.
            calcularPerímetro());
        figura4.determinarTipoTriángulo();
    }
}
```

Diagrama de clases

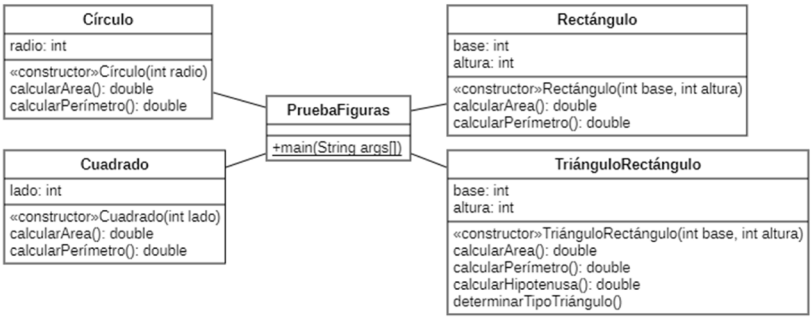

Figura 2.10. Diagrama de clases del ejercicio 2.4.

Explicación del diagrama de clases

Se ha definido una clase para cada figura geométrica (Círculo, Cuadrado, Rectángulo y TriánguloRectángulo). Cada clase contiene sus atributos respectivos. La clase Círculo tiene su radio como atributo (de tipo *int*). La clase Cuadrado tiene la longitud de sus lados como atributo (de tipo *int*). La clase Rectángulo tiene como atributos su base y altura (ambos de tipo *int*). La clase TriánguloRectángulo, su base y altura (ambos de tipo *int*). La hipotenusa no es atributo, en este caso será un valor que será calculado por medio de un método.

Todas las clases poseen los mismos métodos: un constructor y métodos para calcular el área y calcular el perímetro que devuelven valores de tipo *double*. La clase TriánguloRectángulo tiene dos métodos adicionales para calcular el valor de la hipotenusa y para determinar qué tipo de triángulo es (de acuerdo con la longitud de sus lados).

La clase PruebaFiguras es una clase que no tiene significado en el dominio conceptual del problema. Como su nombre lo indica se ha definido para definir un único método *main* que sirva como punto de entrada para la ejecución del programa. El método *main* es un método estático cuya representación en UML es subrayando la signatura del método.

Diagrama de objetos

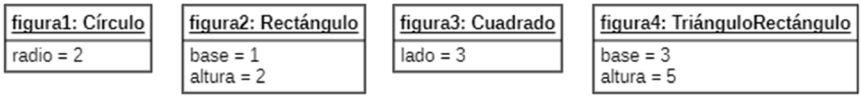

Figura 2.11. Diagrama de objetos del ejercicio 2.4.

Ejecución del programa

```
El área del círculo es = 12.566370614359172
El perímetro del círculo es = 12.566370614359172

El área del rectángulo es = 2.0
El perímetro del rectángulo es = 6.0

El área del cuadrado es = 9.0
El perímetro del cuadrado es = 12.0

El área del triángulo es = 7.0
El perímetro del triángulo es = 13.8309518948453
Es un triángulo escaleno
```

Figura 2.12. Ejecución del programa del ejercicio 2.4.

Ejercicios propuestos

- Agregar una nueva clase denominada Rombo. Definir los métodos para calcular el área y el perímetro de esta nueva figura geométrica.
- Agregar una nueva clase denominada Trapecio. Definir los métodos para calcular el área y el perímetro de esta nueva figura geométrica.

Ejercicio 2.5. Definición de métodos con parámetros

Los métodos de una clase pueden tener parámetros, los cuales son datos que se requieren para la ejecución del método. Un método puede no contener ningún parámetro, un solo parámetro o muchos parámetros.

Se recomienda que la cantidad de parámetros sea la adecuada para el correcto funcionamiento del método, es decir, que no tenga demasiados parámetros.

Objetivo de aprendizaje

Al finalizar este ejercicio, el lector tendrá la capacidad para definir métodos con parámetros de entrada.

Enunciado: clase CuentaBancaria

Se requiere un programa que modele una cuenta bancaria que posee los siguientes atributos:

- Nombres del titular.
- Apellidos del titular.
- Número de la cuenta bancaria.
- Tipo de cuenta: puede ser una cuenta de ahorros o una cuenta corriente.
- Saldo de la cuenta.

Se debe definir un constructor que inicialice los atributos de la clase. Cuando se crea una cuenta bancaria, su saldo inicial tiene un valor de cero.

En una determinada cuenta bancaria se puede:

- Imprimir por pantalla los valores de los atributos de una cuenta bancaria.
- Consultar el saldo de una cuenta bancaria.
- Consignar un determinado valor en la cuenta bancaria, actualizando el saldo correspondiente.
- Retirar un determinado valor de la cuenta bancaria, actualizando el saldo correspondiente. Es necesario tener en cuenta que no se puede realizar el retiro si el valor solicitado supera el saldo actual de la cuenta.

Solución

Clase: CuentaBancaria

```
/**
 * Esta clase define objetos que representan una cuenta bancaria que
 * tienen un nombre y apellidos del titular, un número de cuenta, un
 * tipo de cuenta (ahorros o corriente) y un saldo.
 * @version 1.2/2020
 */
public class CuentaBancaria {
    // Atributo que define los nombres del titular de la cuenta bancaria
    String nombresTitular;
    // Atributo que define los apellidos del titular de la cuenta bancaria
    String apellidosTitular;
    // Atributo que define el número de la cuenta bancaria
    int númeroCuenta;
    // Tipo de cuenta como un valor enumerado
    enum tipo {AHORROS, CORRIENTE}
    // Atributo que define el tipo de cuenta bancaria
    tipo tipoCuenta;
    /* Atributo que define el saldo de la cuenta bancaria con valor inicial
       cero */
    float saldo = 0;

    /**
    * Constructor de la clase CuentaBancaria
    * @param nombresTitular Parámetro que define los nombres del
    * titular de una cuenta bancaria
    * @param apellidosTitular Parámetro que define los apellidos del
    * titular de una cuenta bancaria
    * @param numeroCuenta Parámetro que define el número de una
    * cuenta bancaria
    * @param tipoCuenta Parámetro que define el tipo de una cuenta
    * bancaria (puede ser ahorros o corriente)
    */
```

```
CuentaBancaria(String nombresTitular, String apellidosTitular, int
    numeroCuenta, tipo tipoCuenta) {
    /* Tener en cuenta que no se pasa como parámetro el saldo ya
        que inicialmente es cero. */
    this.nombresTitular = nombresTitular;
    this.apellidosTitular = apellidosTitular;
    this.númeroCuenta = numeroCuenta;
    this.tipoCuenta = tipoCuenta;
}

/**
* Método que imprime en pantalla los datos de una cuenta bancaria
*/
void imprimir() {
    System.out.println("Nombres del titular = " + nombresTitular);
    System.out.println("Apellidos del titular = " + apellidosTitular);
    System.out.println("Número de cuenta = " + númeroCuenta);
    System.out.println("Tipo de cuenta = " + tipoCuenta);
    System.out.println("Saldo = " + saldo);
}

/**
* Método que imprime en pantalla el saldo actual de una cuenta
* bancaria
*/
void consultarSaldo() {
    System.out.println("El saldo actual es = " + saldo);
}

/**
* Método que actualiza y devuelve el saldo de una cuenta bancaria a
* partir de un valor a consignar
* @param valor Parámetro que define el valor a consignar en la
* cuenta bancaria. El valor debe ser mayor que cero
* @return Valor booleano que indica si el valor a consignar es válido
* o no
*/
```

```
boolean consignar(int valor) {
    // El valor a consignar debe ser mayor que cero
    if (valor > 0) {
        saldo = saldo + valor; /* Se actualiza el saldo de la cuenta con
            el valor consignado */
        System.out.println("Se ha consignado $" + valor + " en la
            cuenta. El nuevo saldo es $" + saldo);
        return true;
    } else {
        System.out.println("El valor a consignar debe ser mayor que
            cero.");
        return false;
    }
}

/**
* Método que actualiza y devuelve el saldo de una cuenta bancaria a
* partir de un valor a retirar
* @param valor Parámetro que define el valor a retirar en la cuenta
* bancaria. El valor debe ser mayor que cero y el saldo de la cuenta
debe quedar con un valor positivo o igual a cero
* @return Valor booleano que indica si el valor a retirar es válido o no
*/
boolean retirar(int valor) {
    /* El valor debe ser mayor que cero y no debe superar el saldo
        actual */
    if ((valor > 0) && (valor <= saldo)) {
        saldo = saldo - valor; /* Se actualiza el saldo de la cuenta con
            el valor retirado */
        System.out.println("Se ha retirado $" + valor + " en la cuenta.
            El nuevo saldo es $" + saldo);
        return true;
    } else {
        System.out.println("El valor a retirar debe ser menor que el
            saldo actual.");
        return false;
    }
}
```

```
        /**
        * Método main que crea una cuenta bancaria sobre las cuales se
        * realizan las operaciones de consignar y retirar
        */
        public static void main(String args[]) {
            CuentaBancaria cuenta = new CuentaBancaria("Pedro","Pérez",
            123456789,tipo.AHORROS);
            cuenta.imprimir();
            cuenta.consignar(200000);
            cuenta.consignar(300000);
            cuenta.retirar(400000);
        }
    }
```

Diagrama de clases

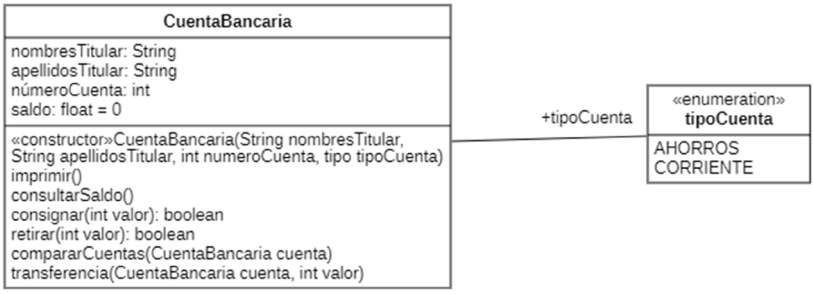

Figura 2.13. Diagrama de clases del ejercicio 2.5.

Explicación del diagrama de clases

Se ha definido una clase para modelar una cuenta bancaria. La clase cuenta con los atributos que representan los nombres del titular (de tipo *String*), los apellidos del titular (de tipo *String*), su número de cuenta (de tipo *int*) y el saldo de la cuenta con un valor inicial de cero (de tipo *float*). Se han definido los métodos de la cuenta bancaria: su constructor, un método para imprimir los datos en pantalla y consultar saldo (ambos no retornan ningún valor) y consignar y retirar que tienen un parámetro (el valor a consignar y retirar respectivamente). Los métodos consignar y retirar devuelven un valor booleano que determina si se pudo realizar la operación o no.

La clase CuentaBancaria tiene un atributo denominado tipoCuenta, que es un valor enumerado. En UML, el valor enumerado se puede representar como una clase con el estereotipo <<enumeration>> con sus valores constantes localizados como atributos en la clase.

Diagrama de objetos

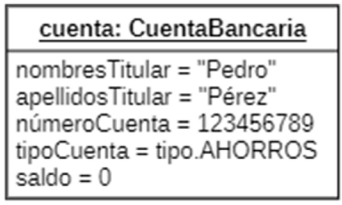

Figura 2.14. Diagrama de objetos del ejercicio 2.5.

Ejecución del programa

```
Nombres del titular = Pedro
Apellidos del titular = Pérez
Número de cuenta = 123456789
Tipo de cuenta = AHORROS
Saldo = 0.0
Se ha consignado $200000 en la cuenta. El nuevo saldo es $200000.0
Se ha consignado $300000 en la cuenta. El nuevo saldo es $500000.0
Se ha retirado $400000 en la cuenta. El nuevo saldo es $100000.0
```

Figura 2.15. Ejecución del programa del ejercicio 2.5.

Ejercicios propuestos

- Agregar a la clase CuentaBancaria, un atributo que represente el porcentaje de interés mensual aplicado a la cuenta.
- Agregar un método que calcule un nuevo saldo aplicando la tasa de interés correspondiente.

Ejercicio 2.6. Objetos como parámetros

Los métodos de una clase también pueden recibir un listado de objetos como parámetros. Es interesante que los cambios que se realicen a dichos

objetos se mantienen en los métodos que invocaron el método que utilizó el objeto como parámetro.

Por lo tanto, el paso de objetos como parámetros de un método es como el paso de parámetros por referencia; las variables mantienen una referencia al objeto. Mientras que, cuando se pasa un objeto como parámetro, se está realizando una copia de la referencia al objeto (Deitel y Deitel, 2017).

Objetivos de aprendizaje

Al finalizar este ejercicio, el lector tendrá la capacidad para:

- Definir métodos que utilizan objetos como parámetros de entrada.
- Definir métodos que invocan a otros métodos definidos anteriormente.

Enunciado: clase CuentaBancaria

Se requiere modificar el programa de la cuenta bancaria (ejercicio 2.5) para que realice las siguientes actividades:

- Comparar saldos entre cuentas bancarias. La cuenta para comparar es un objeto que se envía como parámetro del método. El método devuelve un valor booleano de verdadero si la cuenta actual es mayor o igual a la cuenta que se pasó como parámetro.
- Transferir dinero de una cuenta bancaria a otra. El método debe recibir como parámetro la cuenta de destino y el valor a transferir. El saldo de la cuenta actual debe disminuir el valor a transferir y el saldo de la cuenta destino debe aumentar. El método debe reutilizar el método retirar para evaluar si la cantidad a transferir se encuentra en la cuenta de origen.

Solución

Clase: CuentaBancaria (continuación)

```
/**
* Método que compara los saldos de dos cuentas bancarias y
* muestra el resultado en pantalla
* @param cuenta Parámetro que define otra cuenta bancaria con la
* cual se va a comparar la cuenta bancaria actual
*/
void compararCuentas(CuentaBancaria cuenta) {
    /* Determina si el saldo de la cuenta actual es mayor o igual que
        el saldo de la otra cuenta */
    if (saldo >= cuenta.saldo) {
        System.out.println("El saldo de la cuenta actual es mayor o
            igual al saldo de la cuenta pasada como parámetro.");
    } else {
        System.out.println("El saldo de la cuenta actual es menor al
            saldo de la cuenta pasada como parámetro.");
    }
}

/**
* Método que transfiere un valor de una cuenta a otra
* @param cuenta Parámetro que define otra cuenta bancaria a la
* cual se le va a transferir un valor
* @param valor Parámetro que define el valor a transferir
*/
void transferencia(CuentaBancaria cuenta, int valor) {
    if (retirar(valor)) { // Si se puede retirar el valor de la cuenta actual
        cuenta.consignar(valor); /* Se realiza la consignación en la
            otra cuenta */
    }
}

/**
* Método main que crea dos cuentas bancarias sobre las cuales se
* realizan las operaciones de consignar, comparar, transferir y
* consultar saldos
*/
```

```
    public static void main(String args[]) {
        CuentaBancaria cuenta1 = new
            CuentaBancaria("Pedro","Pérez",123456789,tipo.
            AHORROS);
        CuentaBancaria cuenta2 = new
            CuentaBancaria("Pablo","Pinzón",44556677,tipo.
            AHORROS);
        cuenta1.consignar(200000);
        cuenta2.consignar(100000);
        cuenta1.compararCuentas(cuenta2);
        cuenta1.transferencia(cuenta2,50000);
        cuenta1.consultarSaldo();
        cuenta2.consultarSaldo();
    }
```

Diagrama de clases

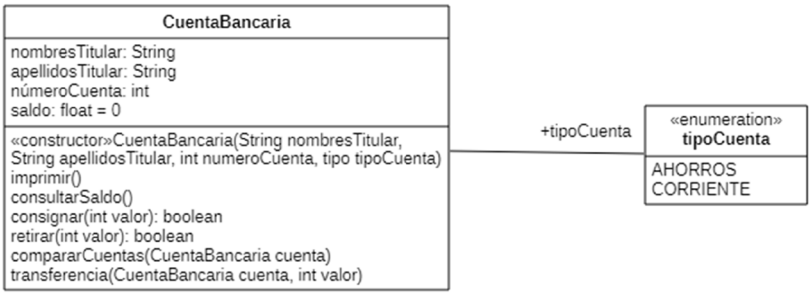

Figura 2.16. Diagrama de clases del ejercicio 2.6.

Explicación del diagrama de clases

Comparando este diagrama con el del ejercicio anterior, se han agregado dos nuevos métodos a la clase CuentaBancaria:

- Comparar cuentas, que tiene como parámetro un objeto de tipo CuentaBancaria.
- Transferencia, cuyos parámetros son un objeto de tipo CuentaBancaria y el valor a transferir a dicha cuenta.

Ambos métodos no tienen valor de retorno. La clase CuentaBancaria tiene un atributo denominado tipoCuenta, un valor enumerado. En UML, el

valor enumerado se puede representar como una clase con el estereotipo <<enumeration>> con sus valores constantes localizados como atributos en la clase.

Diagrama de objetos

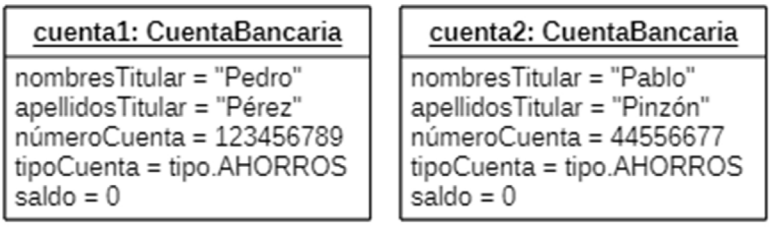

Figura 2.17. Diagrama de objetos del ejercicio 2.6.

Ejecución del programa

```
Se ha consignado $200000 en la cuenta. El nuevo saldo es $200000.0
Se ha consignado $100000 en la cuenta. El nuevo saldo es $100000.0
El saldo de la cuenta actual es mayor o igual al saldo de la cuenta pasada como parámetro.
Se ha retirado $50000 en la cuenta. El nuevo saldo es $150000.0
Se ha consignado $50000 en la cuenta. El nuevo saldo es $150000.0
El saldo actual es = 150000.0
El saldo actual es = 150000.0
```

Figura 2.18. Ejecución del programa del ejercicio 2.6.

Ejercicios propuestos

- Agregar un atributo a la clase CuentaBancaria, que determine si la cuenta está activa (de tipo *boolean*). Una cuenta está activa si tiene un saldo positivo. No se pueden realizar consignaciones a la cuenta si está inactiva. Si al retirar dinero, el saldo de la cuenta es cero, la cuenta pasa a considerarse inactiva.

Ejercicio 2.7. Métodos de acceso

Java posee modificadores de acceso para restringir el alcance de una clase, constructores, atributos, métodos y variables.

De acuerdo con Reyes y Stepp (2016), los modificadores de acceso disponibles en Java son:

- **Por defecto**: cuando no se especifica el modificador de acceso se dice que está predeterminado. Los elementos que no se declaran utilizando modificadores de acceso son accesibles solo dentro del mismo paquete.
- **Privado**: el modificador de acceso privado se especifica usando la palabra clave *private*. Los elementos declarados como privados son accesibles solo dentro de la clase en la que se declaran. Cualquier otra clase del mismo paquete no podrá acceder a estos elementos.
- **Protegido**: el modificador de acceso protegido se especifica mediante la palabra clave *protected*. Los elementos declarados como protegidos son accesibles dentro del mismo paquete o subclases en paquetes diferentes.
- **Público**: el modificador de acceso público se especifica mediante la palabra clave *public*. El modificador de acceso público tiene el alcance más amplio entre todos los demás modificadores de acceso. Los elementos que se declaran públicos son accesibles desde cualquier lugar del programa.

Objetivo de aprendizaje

Al finalizar este ejercicio, el lector tendrá la capacidad para definir métodos de acceso para clases, atributos, constructores y métodos de un programa.

Enunciado: clase Película

Realizar un programa en Java que defina una clase Película con los siguientes atributos privados:

- Nombre: es el nombre de la película y es de tipo *String*.
- Director: representa el nombre del director de la película y es de tipo *String*.
- Género: es el género de la película, un tipo enumerado con los siguientes valores: ACCIÓN, COMEDIA, DRAMA y SUSPENSO.
- Duración: duración de la película en minutos, esta es de tipo *int*.
- Año: representa el año de realización de la película.
- Calificación: es la valoración de la película por parte de sus usuarios y es de tipo *double*.

Se debe definir un constructor público para la clase, que recibe como parámetros los valores de todos sus atributos. También se deben definir los siguientes métodos:

- Métodos *get* y *set* para cada atributo y con los derechos de acceso privados para los *set* y públicos para los *get*.
- Un método imprimir público que muestre en pantalla los valores de los atributos.
- Un método privado *boolean esPeliculaEpica()*, el cual devuelve un valor verdadero si la duración de la película es mayor o igual a tres horas, en caso contrario devuelve falso.
- Un método privado *String calcularValoración()*, el cual según la tabla 2.5 devuelve una valoración subjetiva.

Tabla 2.5. Valoración de las películas

Calificación	Valoración	Calificación	Valoración
[0, 2]	Muy mala	(7, 8]	Buena
(2, 5]	Mala	(8, 10]	Excelente
(5, 7]	Regular		

- El método privado *boolean esSimilar()* compara la película actual con otra película que se le pasa como parámetro. Si ambas películas son del mismo género y tienen la misma valoración, devuelve verdadero; en caso contrario, devuelve falso.
- Un método *main* que construya dos películas; determinar si son películas épicas; calcular su respectiva valoración y determinar si son similares. Las dos películas están en la tabla 2.6.
- Mostrar los resultados de la ejecución del método *main*.

Tabla 2.6. Objetos películas

Objeto	Nombre	Director	Género	Duración	Año	Calif.
película1	Gandhi	Richard Attenborough	DRAMA	191	1982	8.1
película2	Thor	Kenneth Branagh	ACCIÓN	115	2011	7.0

Solución

Clase: Película

```
/**
 * Esta clase define objetos que representan una Película. Una película
 * tiene un nombre, un director, un tipo, una duración, un año de
 * estreno y una calificación realizada por los usuarios.
 * @version 1.2/2020
 */
public class Película {
    // Atributo que define el nombre de la película
    private String nombre;
    // Atributo que define el director de la película
    private String director;
    // Tipo de película como un valor enumerado
    private enum tipo {ACCIÓN, COMEDIA, DRAMA, SUSPENSO};
    // Atributo que define el tipo de película
    tipo género;
    // Atributo que define la duración de la película
    private int duración;
    // Atributo que define el año de estreno de la película
    private int año;
    // Atributo que define la calificación de la película por el público
    private double calificación;

    /**
    * Constructor de la clase Película
    * @param nombre Parámetro que define el nombre o título de la
    * película
    * @param director Parámetro que define el nombre completo del
    * director de la película
    * @param género Parámetro que define el género de la película
    * @param duración Parámetro que define la duración de una
    * película (en minutos)
    * @param año Parámetro que define el año de estreno de la película
    * @param calificación Parámetro que define la calificación de la
    * pelicula realizada por el público
    */
```

```
public Película(String nombre, String director, tipo género, int dura-
    ción, int año, double calificación) {
    this.nombre = nombre;
    this.director = director;
    this.género = género;
    this.duración = duración;
    this.año = año;
    this.calificación = calificación;
}

/**
* Método que devuelve el nombre de una película
* @return El nombre de una película
*/
public String getNombre() {
    return nombre;
}

/**
* Método que establece el nombre de una película
* @param nombre Parámetro que define el nombre de una película
*/
private void setNombre(String nombre) {
    this.nombre = nombre;
}

/**
* Método que devuelve el director de una película
* @return El director de una película
*/
public String getDirector() {
    return director;
}

/**
* Método que establece el director de una película
* @param director Parámetro que define el director de una película
*/
private void setDirector(String director) {
    this.director = director;
}
```

```
/**
* Método que devuelve el género de una película
* @return El género de una película
*/
public tipo getGénero() {
    return género;
}

/**
* Método que establece el género de una película
* @param género Parámetro que define el género de una película
*/
private void setGénero(tipo género) {
    this.género = género;
}

/**
* Método que devuelve la duración de una película
* @return La duración de una película
*/
public int getDuración() {
    return duración;
}

/**
* Método que establece la duración de una película
* @param duración Parámetro que define la duración de una película
*/
private void setDuración(int duración) {
    this.duración = duración;
}

/**
* Método que devuelve el año de una película
* @return El año de estreno de una película
*/
public int getAño() {
    return año;
}

/**
* Método que establece el año de estreno de una película
* @param año Parámetro que define el año de una película
*/
```

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

    /**
    * Método que devuelve la calificación de una película
    * @return La calificación de una película
    */
    public double getCalificación() {
        return año;
    }

    /**
    * Método que establece la calificación de una película
    * @param calificación Parámetro que define la calificación de una
        película
    */
    private void setCalificación(double calificación) {
        this.calificación = calificación;
    }

    /**
    * Método que imprime en pantalla los datos de una película
    */
    public void imprimir() {
        System.out.println("Nombre: " + nombre);
        System.out.println("Director: " + director);
        System.out.println("Género: " + género);
        System.out.println("Duración: " + duración);
        System.out.println("Año: " + año);
        System.out.println("Calificación: " + calificación);
    }

    /**
    * Método que determina si una película se puede considerar como épica
    * @return Valor booleano que determina si una película es épica o no
    */
    private boolean esPelículaEpica() {
        /* Una película se considera épica si tiene una duración igual o
            superior a 180 minutos */
        if (duración >= 180) {
            return true;
        } else {
```

```
            return false;
        }
    }

    /**
    * Método que determina la valoración cualitativa de una película
    * @return Valoración de una película
    */
    private String calcularValoración() {
        if (calificación >= 0 && calificación <= 2) { /* Entre [0, 2] se con-
            sidera "Muy mala" */
            return "Muy mala";
        } else if (calificación > 2 && calificación <= 5) { /* Entre (2, 5] se
            considera "Mala" */
            return "Mala";
        } else if (calificación > 5 && calificación <= 7) { /* Entre (5,7] se
            considera "Regular" */
            return "Regular";
        } else if (calificación > 7 && calificación <= 8) { /* Entre (7,8] se
            considera "Buena" */
            return "Buena";
        } else if (calificación > 8 && calificación <= 10){ /* Entre (8,10] se
            considera "Excelente" */
            return "Excelente";
        } else {
            return "No tiene asignada una valoración válida";
        }
    }

    /**
    * Método que determina si una película es similar a otra
    * @return Valor booleano que determina si una película es similar a
    * otra
    */
    private boolean esSimilar(Película película) {
        /* Dos películas son similares si ambas son del mismo género y si
            tienen la misma valoración */
        if (película.género == género && película.calcularValoración() ==
            calcularValoración()) {
            return true;
        } else {
            return false;
        }
```

```
    }

    /**
    * Método main que crea dos películas, imprime sus datos en
    * pantalla, determina si son épicas y si son similares
    */
    public static void main(String ags[]) {
        Película película1 = new Película("Gandhi", "Richard Attenborough",
            tipo.DRAMA,191,1982,8.3);
        Película película2 = new Película("Thor", "Kenneth Branagh",
            tipo.ACCIÓN, 115,2011,7.0);
        película1.imprimir();
        System.out.println();
        película2.imprimir();
        System.out.println();
        System.out.println("La película " + película1.getNombre() + " es
            épica: " + película1.esPelículaEpica());
        System.out.println("La película " + película2.getNombre() + " es
            épica: " + película2.esPelículaEpica());
        System.out.println("La película " + película1.getNombre() + " y
            la película " + película2.getNombre() + " son similares = " +
                película1.esSimilar(película2));
    }
}
```

Diagrama de clases

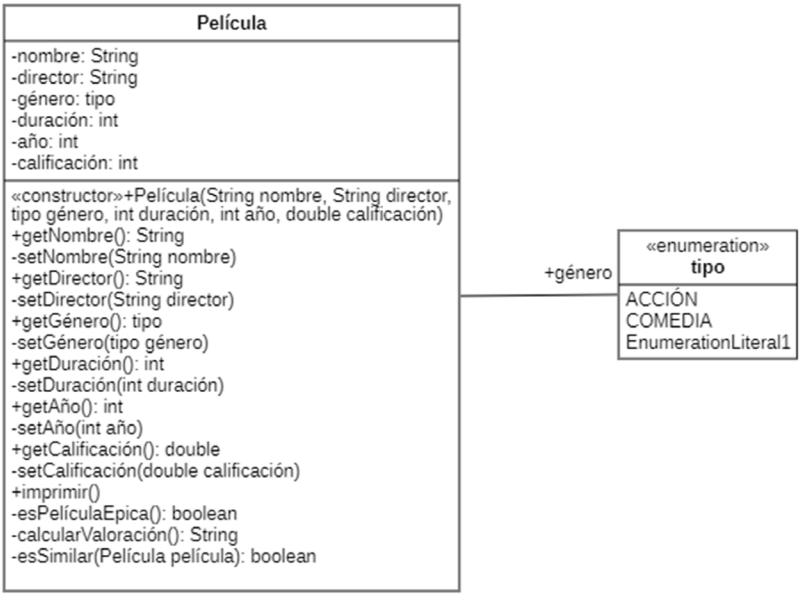

Figura 2.19. Diagrama de clases del ejercicio 2.7.

Explicación del diagrama de clases

Se ha definido una clase Película con los atributos nombre, director, género, duración, año y calificación, cada uno de ellos con su respectivo tipo de dato. Todos los atributos de la clase son privados, lo cual en UML se identifica con el símbolo (-), este símbolo antecede el nombre del atributo.

El constructor de la clase y los métodos *get* de cada atributo tienen acceso público representados con el símbolo (+), que antecede el nombre del método, en UML. Así mismo, métodos privados, con el símbolo (-) antes del nombre del método, los métodos *set* de cada atributo y los métodos para determinar si es una película épica, para calcular la valoración de la película y para determinar si una película es similar a otra.

Los métodos públicos podrán ser accedidos por objetos diferentes a Película, mientras que los métodos privados solo serán accedidos por objetos de la misma clase. Como los atributos son privados, la única forma de cambiar valores a sus atributos es por objetos de la misma clase y por medio del método *set*.

La consulta de los atributos por medio de los métodos *get* es pública y puede hacerse por objetos diferentes a Película.

Para representar el género de la película, el cual es un atributo enumerado, se ha definido una asociación denominada género con una clase tipo *enumeration*. Este tipo de clases se representan en UML con el estereotipo <<*enumeration*>> y con su lista de constantes como atributos en el segundo compartimiento de la clase.

Diagrama de objetos

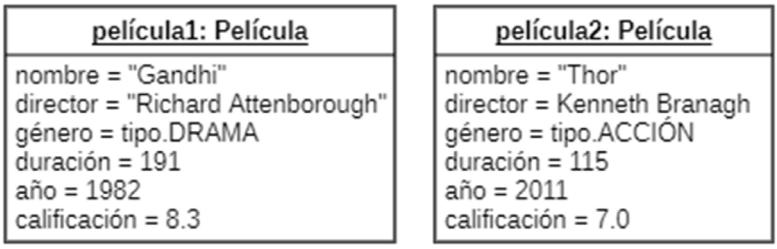

Figura 2.20. Diagrama de objetos del ejercicio 2.7.

Ejecución del programa

```
Nombre: Gandhi
Director: Richard Attenborough
Género: DRAMA
Duración: 191
Año: 1982
Calificación: 8.3

Nombre: Thor
Director: Kenneth Branagh
Género: ACCIÓN
Duración: 115
Año: 2011
Calificación: 7.0

La película Gandhi es épica: true
La película Thor es épica: false
La película Gandhi y la película Thor son similares = false
```

Figura 2.21. Ejecución del programa del ejercicio 2.7.

Ejercicios propuestos

Definir un método privado denominado *imprimirCartel* que muestra en pantalla los datos de la película en el siguiente formato:

-------- Título ----------

****

Año

Género1, Género2, Género3

Director

La valoración se debe convertir a una cantidad determinada de asteriscos (*).

Una película puede tener hasta tres géneros. Los géneros se muestran separados por comas.

Ejercicio 2.8. Asignación de objetos

En Java, cuando se asigna un objeto a otro de la misma clase, ambos están referenciando al mismo objeto (Schildt, 2018). En una sentencia de asignación como:

```
Object o1, o2;
o1 = new Object();
o2 = o1;
```

Ambos objetos o1 y o2 están referenciando el mismo objeto, no se crea un objeto duplicado, ni se asigna memoria extra. Los cambios que se realicen a un objeto se actualizan de inmediato en el otro.

Objetivos de aprendizaje

Al finalizar este ejercicio, el lector tendrá la capacidad para:

- Determinar el contenido de un objeto que ha pasado por varias instrucciones de asignación y se ha cambiado su contenido o referencia a otros objetos.
- Cambiar el estado de un objeto utilizando métodos *set*.

Enunciado: clase Avión

Se tiene un código que modela una clase Avión que posee dos atributos: nombre del fabricante del avión (tipo *String*) y número de motores del avión (tipo *int*). La clase tiene un constructor y métodos *get* y *set* para cada atributo. Además, tiene los siguientes métodos adicionales:

- Un método denominado imprimirFabricante(), que muestra en pantalla el nombre del fabricante de un avión.
- Un método denominado cambiarFabricante(Avión a), que recibe como parámetro un objeto de tipo Avión y le cambia el valor de su atributo fabricante a un valor predefinido "Loockhead".

En el método *main* se crean dos aviones: a1 (fabricante "Airbus" con cuatro motores) y a2 (fabricante "Lookheed" con cuatro motores). Luego, los datos de cada avión se imprimen por pantalla. Después, se realizan llamadas a los métodos setFabricante y cambiarFabricante, los cuales cambiarán los valores de sus atributos. ¿Cuál es el resultado final de la ejecución del método *main*? Determinar lo que se imprime en pantalla.

Solución

Clase: Avión

```
/**
 * Esta clase define objetos de tipo Avión con un fabricante y número de
 * motores como atributos.
 * @version 1.2/2020
 */
public class Avión {
    private String fabricante; /* Atributo que define el nombre del fabri-
        cante del avión */
    private int númeroMotores; /* Atributo que define el número de mo-
        tores del avión */

    /**
    * Constructor de la clase Avión
    * @param fabricante Parámetro que define el nombre del fabricante
    * de un avión
    * @param númeroMotores Parámetro que define el número de
    * motores que tiene un avión
```

```
*/
private Avión(String fabricante, int númeroMotores) {
    this.fabricante = fabricante;
    this.númeroMotores = númeroMotores;
}

/**
* Método que devuelve el nombre del fabricante de un avión
* @return El nombre del fabricante de un avión
*/
public String getFabricante() {
    return fabricante;
}

/**
* Método que establece el nombre de un fabricande de un avión
* @param fabricante Parámetro que define el nombre del fabricante
* de un avión
*/
private void setFabricante(String fabricante) {
    this.fabricante = fabricante;
}

/**
* Método que cambia el fabricante de un avión pasado como
* parámetro por el valor "Lockheed"
* @param a Parámetro que define un avión
*/
private void cambiarFabricante(Avión a) {
    a.setFabricante("Lockheed");
}

/**
* Método que devuelve el número de motores de un avión
* @return El número de motores de un avión
*/
public int getNúmeroMotores() {
    return númeroMotores;
}
```

```
    /**
    * Método que establece el número de motores de un avión
    * @param númeroMotores Parámetro que define el número de
    * motores de un avión
    */
    private void setNúmeroMotores(int númeroMotores) {
        this.númeroMotores = númeroMotores;
    }

    /**
    * Método que imprime en pantalla el fabricante de un avión
    */
    public void imprimirFabricante() {
        System.out.println("El fabricante del avión es: " + fabricante);
    }

    /**
    * Método main que crea dos aviones y modifica sus fabricantes
    */
    public static void main(String args[]) {
        Avión a1 = new Avión("Airbus",4);
        Avión a2;
        Avión a3 = new Avión("Boeing",4);
        a2 = a1;
        a1.imprimirFabricante();
        a2.imprimirFabricante();
        a1.setFabricante("Douglas");
        a1.imprimirFabricante();
        a2.imprimirFabricante();
        a1.cambiarFabricante(a2);
        a2.imprimirFabricante();
    }
}
```

Diagrama de clases

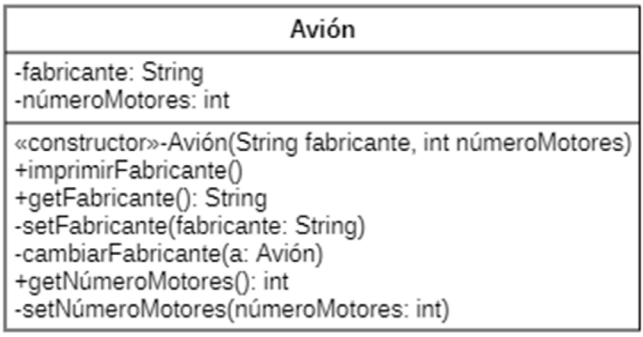

Figura 2.22. Diagrama de clases del ejercicio 2.8.

Explicación del diagrama de clases

Se ha definido una clase Avión con dos atributos que representan el fabricante del avión (de tipo *String*) y el número de motores que tiene (de tipo *int*). Ambos atributos son privados, tienen el símbolo (-) precediendo el nombre del atributo.

La clase Avión tiene un constructor que inicializa los atributos del avión. Además, tiene los métodos *get* y *set* de cada atributo. Los métodos *get* son públicos y los *set*, privados. El método cambiarFabricante es privado y recibe como parámetro un objeto de tipo Avión.

Diagrama de objetos

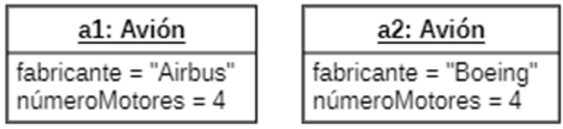

Figura 2.23. Diagrama de objetos del ejercicio 2.8.

Ejecución del programa

```
El fabricante del avión es: Airbus
El fabricante del avión es: Airbus
El fabricante del avión es: Douglas
El fabricante del avión es: Douglas
El fabricante del avión es: Lockheed
```

Figura 2.24. Ejecución del programa del ejercicio 2.8.

El primer avión a1 tiene como fabricante "Airbus", por ello imprime en pantalla dicho valor en la primera línea. Al segundo avión a2 se le asignó el contenido de a1, por ello imprime también "Airbus" en la segunda línea. Luego, se cambia el fabricante de a1, pasa de "Airbus" a "Douglas", por ello imprime "Douglas" en la tercera línea. Al cambiar el contenido de a1, también cambia en forma automática el contenido de a2, por ello imprime "Douglas" en la cuarta línea. Por último, al invocar el método cambiarFabricante se cambia el valor del objeto pasado como parámetro a2 y se imprime "Lockheed" en la quinta línea.

Ejercicios propuestos

- En un método *main* se deben crear dos objetos Avión y asignar el primer objeto al segundo. Luego, mostrar en pantalla los valores de los atributos de los dos objetos. Después, asignar el valor "*Stealth*" al atributo fabricante del segundo objeto. Por último, es necesario imprimir en pantalla el valor del atributo fabricante del primer objeto ¿Qué se mostrará en pantalla?

Ejercicio 2.9. Variables locales dentro de un método

En Java, las variables locales se declaran dentro de un método. Su visibilidad está solo dentro del método donde se define. Ningún código fuera de un método puede ver las variables locales dentro de otro método. No es necesario declarar una variable local con un modificador de visibilidad (Deitel y Deitel, 2017).

La vida útil de una variable local cubre desde su declaración hasta el retorno del método. Las variables locales se crean en la pila de llamadas cuando se ingresa al método y se destruyen cuando se sale del método.

Las constantes en Java son variables cuyos valores no pueden cambiar, una vez hayan sido definidas. Las constantes se declaran por medio de instrucciones con el siguiente formato:

static final tipoDato nombreConstante = valor;

Objetivos de aprendizaje

Al finalizar este ejercicio, el lector tendrá la capacidad para:

- Definir variables locales en un método.
- Definir constantes en una clase.

Enunciado: clase ConversorMetros

Realizar un programa en Java que permita realizar las siguientes conversiones de unidades de longitud:

- Metros a centímetros.
- Metros a milímetros.
- Metros a pulgadas.
- Metros a pies.
- Metros a yardas.

Instrucciones Java del ejercicio

Tabla 2.7. Instrucciones Java del ejercicio 2.9.

Instrucción	Descripción	Formato
final	Palabra clave para definir que una vez que a una variable se le ha asignado un valor, siempre contendrá el mismo valor (permanece constante). Se recomienda que los nombres de las constantes se escriban en mayúscula.	*final tipo* VARIABLE = valor;

Solución

Clase: ConversorMetros

```
/**
 * Esta clase define objetos de tipo ConversorMetros el cual permite
 * realizar conversiones entre diferentes unidades de medición de longitud.
 * @version 1.2/2020
 */
public class ConversorMetros {
    /* Atributo que define la cantidad de metros a convertir a diferentes
       unidades de longitud */
    double metros;
    final int FACTOR_METROS_CM = 100; /* Factor de conversión de
       metros a centímetros */
    final int FACTOR_METROS_MILIM = 1000; /* Factor de conversión
       de metros a milímetros */
    final double FACTOR_METROS_PULGADAS = 39.37; /* Factor de
       conversión de metros a pulgadas */
    final double FACTOR_METROS_PIES = 3.28; /* Factor de
       conversión de metros a pies */
    final double FACTOR_METROS_YARDAS = 1.09361; /* Factor de
       conversión de metros a yardas */

    /**
    * Constructor de la clase ConversorMetros
    * @param metros Parámetro que define la cantidad de metros a
    * convertir a otras unidades de longitud
    */
    public ConversorMetros(double metros) {
        this.metros = metros;
    }

    /**
    * Método que convierte metros a centímetros
    * @return Resultado de la conversión de metros a centímetros
    */
    public double convertirMetrosToCentímetros() {
        double centímetros;
        centímetros = FACTOR_METROS_CM * metros;
        return centímetros;
    }
```

```
/**
* Método que convierte metros a milímetros
* @return Resultado de la conversión de metros a milimetros
*/
public double convertirMetrosToMilímetros() {
    double milímetros;
    milímetros = FACTOR_METROS_MILIM * metros;
    return milímetros;
}

/**
* Método que convierte metros a pulgadas
* @return Resultado de la conversión de metros a pulgadas
*/
public double convertirMetrosToPulgadas() {
    double pulgadas;
    pulgadas = FACTOR_METROS_PULGADAS * metros;
    return pulgadas;
}

/**
* Método que convierte metros a pies
* @return Resultado de la conversión de metros a pies
*/
public double convertirMetrosToPies() {
    double pies;
    pies = FACTOR_METROS_PIES * metros;
    return pies;
}

/**
* Método que convierte metros a yardas
* @return Resultado de la conversión de metros a yardas
*/
public double convertirMetrosToYardas() {
    double yardas;
    yardas = FACTOR_METROS_YARDAS * metros;
    return yardas;
}
```

```
    /**
    * Método main que define una cierta cantidad de metros y los
    * convierte a diferentes unidades de longitud
    */
    public static void main (String args[]) {
        ConversorMetros conversor = new ConversorMetros(3.5);
        System.out.println("Metros = " + conversor.metros);
        System.out.println("Metros a centímetros = "  + conversor.convertirMetrosToCentímetros());
        System.out.println("Metros a milímetros = "  + conversor.convertirMetrosToMilímetros());
        System.out.println("Metros a pulgadas = "  + conversor.convertirMetrosToPulgadas());
        System.out.println("Metros a pies = "  + conversor.convertirMetrosToPies());
        System.out.println("Metros a yardas = "  + conversor.convertirMetrosToYardas());
    }
}
```

Diagrama de clases

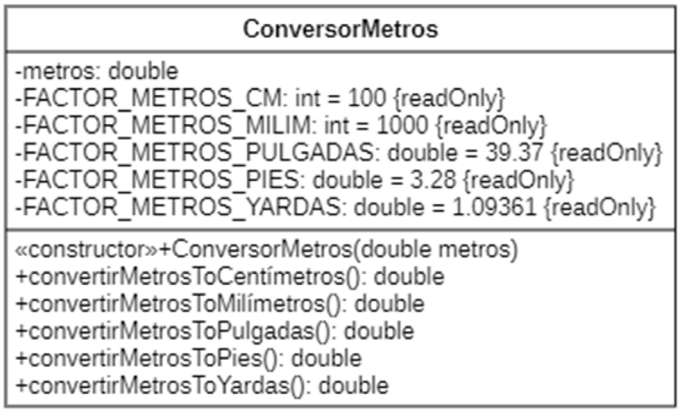

Figura 2.25. Diagrama de clases del ejercicio 2.9.

Explicación del diagrama de clases

Se ha definido una clase denominada ConversorMetros. Tiene un atributo denominado metros (de tipo *double*) para definir la cantidad de metros a

convertir a diferentes unidades de longitud. También se han definido como atributos un conjunto de constantes que se identifican con la etiqueta UML *{readOnly}*. Se recomienda que las constantes se escriban en mayúsculas y si el identificador es una palabra compuesta, cada palabra se separa con el símbolo: _.

La clase tiene un constructor para inicializar la cantidad de metros a convertir y un conjunto de cinco métodos públicos (identificados con el símbolo +) para realizar diferentes conversiones, cada uno con su valor de retorno correspondiente (de tipo *double*). Los métodos permiten convertir un valor en metros a centímetros, milímetros, pulgadas, pies y yardas.

Diagrama de objetos

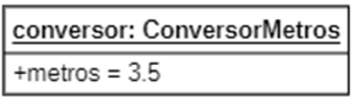

Figura 2.26. Diagrama de objetos del ejercicio 2.9.

Ejecución del programa

```
Metros = 3.5
Metros a centímetros = 350.0
Metros a milímetros = 3500.0
Metros a pulgadas = 137.795
Metros a pies = 11.479999999999999
Metros a yardas = 3.827635
```

Figura 2.27. Ejecución del programa del ejercicio 2.9.

Ejercicios propuestos

- Hacer clases similares para realizar conversiones de unidades de medición como:
  - Medidas de superficie o área: convertir áreas (1 área= 100 m^2) a: hectáreas (1 hectárea= 10 000 m^2); kilómetros cuadrados (1 kilómetro cuadrado= 1 000 000 m^2); fanegas (1 fanega = 6460 m^2) y acres (1 acre= 4046.85 m^2).
  - Medidas de volumen: convertir litros a: galones (1 galón=4,41 litros); pintas (1 pinta= 0.46 litros); barriles (1 barril= 158.99

litros), metros cúbicos (1 m^3= 1000 litros) y hectolitros (1 hectolitro= 100 litros).

Ejercicio 2.10. Sobrecarga de métodos

La sobrecarga de métodos se refiere a que una clase puede poseer múltiples métodos con el mismo nombre (Samoylov, 2019). Es requisito que cada método sobrecargado tenga diferente signatura. Por lo tanto, los métodos sobrecargados realizan la "misma acción", pero teniendo en cuenta:

- Diferentes cantidades de parámetros.

void método(int paramétro1, int paramétro2, int paramétro3)
void método(int paramétro1, int paramétro2)

- Diferentes tipos de parámetros.

void método(int paramétro1, int paramétro2, int paramétro3)
void método(double paramétro1, double paramétro2, double paramétro3)

- Diferente orden de parámetros.

void método(int paramétro1, double paramétro2, float paramétro3)
void método(float paramétro1, double paramétro2, int paramétro3)

- Diferente valor de retorno.

void método(int paramétro1, double paramétro2, float paramétro3)
String método(int paramétro1, double paramétro2, float paramétro3)

Objetivo de aprendizaje

Al finalizar este ejercicio, el lector tendrá la capacidad para definir métodos sobrecargados en una clase.

Enunciado: clase Pedido

Realizar un programa en Java que permita calcular el pedido que realiza un cliente en un restaurante.

Los pedidos de un restaurante están conformados por las siguientes partes:

- Un primer plato.
- Un segundo plato.
- Una bebida.
- Un postre.

Cada uno de dichas partes tiene un nombre y un valor. Se requiere definir métodos sobrecargados para calcular el valor del pedido dependiendo si el cliente solicita:

- Un primer plato y una bebida.
- Un primer plato, un segundo plato y una bebida.
- Un primer plato, un segundo plato, una bebida y un postre.

Implementar un método *main* que utilice los tres métodos sobrecargados en tres diferentes pedidos.

Solución

Clase: Pedido

```
/**
 * Esta clase define objetos de tipo Pedido de un restaurante que consta
 * de diferentes platos y que tiene un determinado valor.
 * @version 1.2/2020
 */
public class Pedido {

    /**
     * Método que tiene como parámetros los elementos que conforman
     * un pedido con un primer plato, su bebida y sus costos
     * correspondientes. El método calcula el costo total del pedido y
     * muestra en pantalla los datos del pedido y su costo total.
     * @param primerPlato Parámetro que define el nombre del primer
     * plato que conforma el pedido
     * @param costoPrimerPlato Parámetro que define el costo del
     * primer plato que conforma el pedido
     * @param bebida Parámetro que define el nombre de la bebida que
     * conforma el pedido
```

```
* @param costoBebida Parámetro que define el costo de la bebida
* que conforma el pedido
*/
public void calcularPedido(String primerPlato, double costoPrimer
    Plato, String bebida, double costoBebida) {
    double total = costoPrimerPlato + costoBebida;
    System.out.println("El costo de " + primerPlato + " y " + bebida +
        " es = $" + total);
}

/**
* Método sobrecargado que tiene como parámetros los elementos
* que conforman un pedido con un primer plato, un segundo plato,
* su bebida y sus costos correspondientes. El método calcula el costo
* total del pedido y muestra en pantalla los datos del pedido y su
* costo total.
* @param primerPlato Parámetro que define el nombre del primer
* plato que conforma el pedido
* @param costoPrimerPlato Parámetro que define el costo del
* primer plato que conforma el pedido
* @param segundoPlato Parámetro que define el nombre del
* segundo plato que conforma el pedido
* @param costoSegundoPlato Parámetro que define el costo del
* segundo plato que conforma el pedido
* @param bebida Parámetro que define el nombre de la bebida que
* conforma el pedido
* @param costoBebida Parámetro que define el costo de la bebida
* que conforma el pedido
*/
public void calcularPedido(String primerPlato, double
    costoPrimerPlato, String segundoPlato, double
    costoSegundoPlato, String bebida, double costoBebida) {
    double total = costoPrimerPlato + costoSegundoPlato +
        costoBebida;
    System.out.println("El costo de " + primerPlato + " + " +
        segundoPlato + " + " + bebida + " es = $" + total);
}
```

```
public void calcularPedido(String primerPlato, double
    costoPrimerPlato, String segundoPlato, double
    costoSegundoPlato, String bebida, double costoBebida) {
    double total = costoPrimerPlato + costoSegundoPlato +
        costoBebida;
    System.out.println("El costo de " + primerPlato + " + " +
        segundoPlato + " + " + bebida + " es = $" + total);
}

/**
* Método sobrecargado que tiene como parámetros los elementos
* que conforman un pedido con un primer plato, segundo plato, su
* bebida, postre y sus costos correspondientes. El método calcula el
* costo total del pedido y muestra en pantalla los datos del pedido y
* su costo total.
* @param primerPlato Parámetro que define el nombre del primer
* plato que conforma el pedido
* @param costoPrimerPlato Parámetro que define el costo del
* primer plato que conforma el pedido
* @param segundoPlato Parámetro que define el nombre del
* segundo plato que conforma el pedido
* @param costoSegundoPlato Parámetro que define el costo del
* segundo plato que conforma el pedido
* @param postre Parámetro que define el nombre del postre que
* conforma el pedido
* @param costoPostre Parámetro que define el costo del postre que
* conforma el pedido
* @param bebida Parámetro que define el nombre de la bebida que
* conforma el pedido
* @param costoBebida Parámetro que define el costo de la bebida
* que conforma el pedido
*/
public void calcularPedido(String primerPlato, double
    costoPrimerPlato, String segundoPlato, double
    costoSegundoPlato, String postre, double costoPostre, String
    bebida, double costoBebida) {
    double total = costoPrimerPlato + costoSegundoPlato +
        costoBebida + costoPostre;
```

```
            System.out.println("El costo de " + primerPlato + " + " +
                segundoPlato + " + " + bebida + " + " +
                    postre + " es = $" + total);
        }
        /**
        * Método main que crea tres diferentes tipos de pedidos en el
        * restaurante.
        */
        public static void main (String args[]) {
            Pedido pedido1 = new Pedido();
            pedido1.calcularPedido("Sancocho", 5000, "Gaseosa", 2000);
            Pedido pedido2 = new Pedido();
            pedido2.calcularPedido("Crema de verduras", 5000,
                "Churrasco", 6000, "Gaseosa", 2000);
            Pedido pedido3 = new Pedido();
            pedido3.calcularPedido("Crema de espinacas",
                5000,"Salmón",10000,"Tiramisú",5000,"Gaseosa",2000);
        }
    }
```

Diagrama de clases

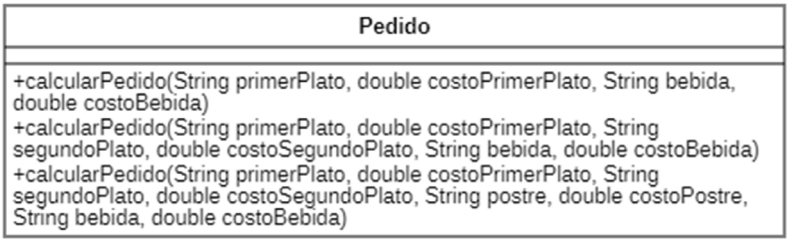

Figura 2.28. Diagrama de clases del ejercicio 2.10.

Explicación del diagrama de clases

Se ha definido una clase denominada Pedido, la cual tiene tres métodos públicos (identificados con el símbolo +) sobrecargados. Al estar sobrecargados, los métodos tienen el mismo nombre, pero diferente signatura. En este caso, los métodos tienen diferente cantidad de parámetros. La clase no tiene atributos.

Diagrama de objetos

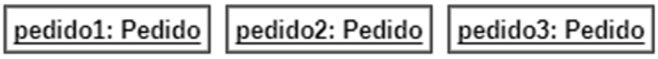

Figura 2.29. Diagrama de objetos del ejercicio 2.10.

Ejecución del programa

```
El costo de Sancocho y Gaseosa es = $7000.0
El costo de Crema de verduras + Churrasco + Gaseosa es = $13000.0
El costo de Crema de espinacas + Salmón + Gaseosa + Tiramisú es = $22000.0
```

Figura 2.30. Ejecución del programa del ejercicio 2.10.

Ejercicios propuestos

- Definir una clase denominada Suma, la cual tiene varios métodos sumar sobrecargados:
  - Un método sumar que obtiene la suma de dos valores enteros pasados como parámetros.
  - Un método sumar que obtiene la suma de tres valores enteros pasados como parámetros.
  - Un método sumar que obtiene la suma de dos valores *double* pasados como parámetros.
  - Un método sumar que obtiene la suma de tres valores *double* pasados como parámetros.

Ejercicio 2.11. Sobrecarga de constructores

A veces es necesario inicializar un objeto de diferentes formas. Para ello, se realiza la sobrecarga de constructores. Los constructores sobrecargados deben tener diferente número o tipo de parámetros. Cada constructor realiza una tarea diferente (Liang, 2017).

Se puede utilizar la palabra reservada *this* para llamar a un constructor desde otro. La llamada *this* debe ser la primera línea de dicho constructor.

Objetivos de aprendizaje

Al finalizar este ejercicio, el lector tendrá la capacidad para:

- Definir constructores sobrecargados.
- Definir constructores que llamen a otros constructores.

Enunciado: clase ArtículoCientífico

Realizar un programa en Java que permita modelar un artículo científico. Los artículos científicos contienen los siguientes metadatos: nombre del artículo, autor, palabras claves, nombre de la publicación, año y resumen.

Se deben definir tres constructores sobrecargados:

- El primero inicializa un artículo científico con solo su título y autor.
- El segundo constructor, un artículo científico con su nombre, autor, palabras claves, nombre de la publicación y año. Debe invocar al primer constructor.
- El tercer constructor, un artículo científico con su nombre, autor, palabras claves, nombre de la publicación, año y resumen. Debe invocar al segundo constructor.

Se requiere un método que imprima los atributos de un artículo en pantalla. Realizar un método *main* que utilice el tercer constructor para instanciar un artículo científico e imprima los valores de sus atributos en pantalla.

Instrucciones Java del ejercicio

Tabla 2.8. Instrucciones Java del ejercicio 2.11.

Instrucción	Descripción	Formato
length	Variable final que permite obtener el tamaño del *array*. No confundir con *length*() con paréntesis, que se aplica a objetos *String*.	nombreArray.length
++	Operador de incremento utilizado para incrementar el valor en 1. Hay dos tipos: preincremento (el valor se incrementa primero y luego se calcula el resultado) y posincremento (el valor se usa por primera vez para calcular el resultado y luego se incrementa).	Preincremento: ++variable Posincremento: variable++

Solución

Clase: ArtículoCientífico

```
/**
 * Esta clase define objetos de tipo ArtículoCientífico con un título,
 * autor, tres palabras clave, año de publicación y un resumen.
 * @version 1.2/2020
 */
public class ArtículoCientífico {
    String título; // Atributo que define el título de un artículo científico
    String autor; // Atributo que define el autor de un artículo científico
    // Atributo que define las palabras clave de un artículo científico
    String[] palabrasClaves = new String[3];
    String publicación; /* Atributo que define la publicación que incluye
        el artículo científico */
    int año; /* Atributo que define el año de publicación de un artículo
        científico */
    String resumen; /* Atributo que define el resumen de un artículo
        científico */

    /**
    * Constructor de la clase ArtículoCientífico
    * @param título Parámetro que define el título del artículo científico
    * @param autor Parámetro que define el autor del artículo científico
    */
    public ArtículoCientífico(String título, String autor) {
        this.título = título;
        this.autor = autor;
    }

    /**
    * Constructor sobrecargado de la clase ArtículoCientífico
    * @param título Parámetro que define el título del artículo científico
    * @param autor Parámetro que define el autor del artículo científico
    * @param palabrasClaves Parámetro que define un listado de
    * palabras clave para el artículo científico
    * @param publicación Parámetro que define el nombre de la
    * publicación a la que pertenece el artículo científico
    * @param año Parámetro que define el año de publicación del
    * artículo científico
    */
```

```
public ArtículoCientífico(String título, String autor, String[] palabras-
    Claves, String publicación, int año) {
    this(título, autor); // Invoca al constructor sobrecargado
    this.palabrasClaves = palabrasClaves;
    this.publicación = publicación;
    this.año = año;
}

/**
* Constructor sobrecargado de la clase ArtículoCientífico
* @param título Parámetro que define el título del artículo científico
* @param autor Parámetro que define el autor del artículo científico
* @param palabrasClaves Parámetro que define un listado de
* palabras clave para el artículo científico
* @param publicación Parámetro que define el nombre de la
* publicación a la que pertenece el artículo científico
* @param año Parámetro que define el año de publicación del
* artículo científico
* @param resumen Parámetro que define el resumen del artículo
* científico
*/
public ArtículoCientífico(String título, String autor, String[] palabras-
    Claves, String publicación, int año, String resumen) {
    this(título, autor, palabrasClaves, publicación, año); /* Invoca al
        constructor sobrecargado */
    this.resumen = resumen;
}

/**
* Método que imprime en pantalla los datos de un artículo científico
*/
public void imprimir() {
    System.out.println("Título del artículo = " + título);
    System.out.println("Autor del artículo = " + autor);
    System.out.println("Palabras clave = ");
    // Recorre el array para imprimir cada palabra clave
    for (int i = 0; i < palabrasClaves.length; i++) {
        System.out.println(palabrasClaves[i]);
    }
    System.out.println("Publicación = " + publicación);
    System.out.println("Año = " + año);
    System.out.println("Resumen = " + resumen);
}
```

```
    /**
     * Método main que instancia un artículo científico y muestra sus
     * datos en pantalla
     */
    public static void main (String args[]) {
        String[] palabras = {"Física","Espacio","Tiempo"};
        ArtículoCientífico artículo = new ArtículoCientífico("La teoría es-
            pecial de la relatividad", "Albert Einstein",palabras, "Anales de
            Física", 1913, "Las leyes de la física son las mismas en todos
            los sistemas de referencia inerciales.");
        artículo.imprimir();
    }
}
```

Diagrama de clases

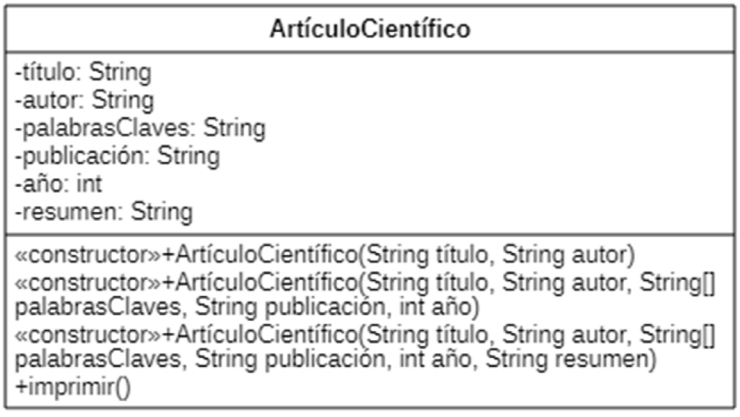

Figura 2.31. Diagrama de clases del ejercicio 2.11.

Explicación del diagrama de clases

Se ha definido una clase denominada ArtículoCientífico, la cual tiene los atributos privados (identificados con el símbolo -): título, autor, palabras clave, publicación, año y resumen. La clase tiene tres constructores públicos sobrecargados (con el mismo nombre y diferente signatura) y un método imprimir. Todos los constructores y el método imprimir son públicos (identificados con el símbolo +).

Diagrama de objetos

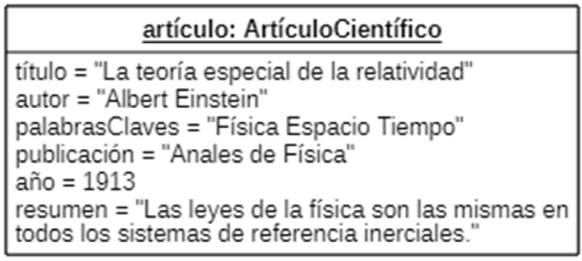

Figura 2.32. Diagrama de objetos del ejercicio 2.11.

Ejecución del programa

```
Título del artículo = La teoría especial de la relatividad
Autor del artículo = Albert Einstein
Palabras clave =
Física
Espacio
Tiempo
Publicación = Anales de Física
Año = 1913
Resumen = Las leyes de la física son las mismas en todos los sistemas de referencia inerciales.
```

Figura 2.33. Ejecución del programa del ejercicio 2.11.

Ejercicios propuestos

- Definir una clase Empleado que tiene como atributos: identificador, nombre, apellidos y edad del empleado. La clase contiene dos constructores:
  - El primer constructor no tiene parámetros e inicializa los atributos del objeto con los siguientes valores: identificador del empleado con el valor 100, el nombre con "Nuevo empleado", apellidos con "Nuevo empleado" y edad del empleado con 18.
  - El segundo constructor asigna valores a los atributos de acuerdo con los valores pasados como parámetros.
- Definir una clase Caja que tiene como atributos la longitud de su base, anchura y altura. La clase contiene tres constructores:
  - El primer constructor asigna valores a los atributos de acuerdo con los valores pasados como parámetros.

- El segundo constructor inicializa todos los atributos de una caja con valores de cero.
- El tercer constructor recibe un parámetro de longitud y les asigna dicho valor a todos sus atributos.
- Definir un nuevo atributo que represente el tipo de caja y un nuevo constructor que reciba como parámetros los valores de los cuatro atributos. Este constructor debe invocar al primero.