

2

Programación orientada a objetos. Objetos

OBJETIVOS DEL CAPÍTULO

- / Conocer las características básicas de la programación orientada a objetos.
- / Valorar las ventajas de la programación orientada a objetos frente a la tradicional.
- / Conocer la estructura básica de las clases.
- / Reconocer las diferencias entre un objeto y una clase. Cómo se inicializan y cómo finalizan.
- / Diseñar clases sencillas.
- / Organizar clases en paquetes y utilizar los mismos.
- / Programar y diseñar métodos que acepten parámetros.

La programación orientada a objetos es un paradigma de programación totalmente diferente al método clásico de programación, el cual utiliza objetos y su comportamiento para resolver problemas y generar programas y aplicaciones informáticas.

Con la programación orientada a objetos (POO) se aumenta la modularidad de los programas y la reutilización de los mismos. Además, la POO se diferencia de la programación clásica porque utiliza técnicas nuevas como el polimorfismo, el encapsulamiento, la herencia, etc.

Generalmente, los lenguajes de programación de última generación permiten la programación orientada a objetos, así como también la programación clásica, con lo cual puede entenderse la POO como una evolución de la programación clásica.

2.1 INTRODUCCIÓN AL CONCEPTO DE OBJETO

Los programas realizados mediante el paradigma de la POO solamente tienen objetos. Todos los objetos pertenecen a una clase, por ejemplo, mi loro Felipe pertenecería a la clase *pájaro*. Todos los objetos de la clase *pájaro* se identificarán, entre otros atributos, con un nombre (en su caso Felipe), un color de plumaje (verde), una edad (en su caso 2 años) y si son domésticos o no (Felipe sí es doméstico pues lo tengo ahora en mi hombro).



Recuerda

En un símil con la costura, las clases serían los patrones y los objetos las prendas.

Clases

Las clases son los moldes de los cuales se generan los objetos. Los objetos se instancian y se generan, instancia y objeto son sinónimos. Por ejemplo, Felipe será un objeto concreto de la clase *pájaro*.



Recuerda

Cuando se escribe un programa o aplicación OO, lo que se hace es definir las clases de objetos dotándolas de estado y comportamiento y cuando se ejecute el programa se crearán los objetos, ya sea estática o dinámicamente.

Cuando se programa, las clases se escriben en ficheros ASCII con el mismo nombre que la clase y extensión .java.

Las clases tienen una estructura parecida a la siguiente:

```
[algo_1] class nombre_de_la_clase [algo_2] {  
    [Atributos]  
    [Métodos]  
}
```

Como puedes observar se ha, etiquetado con corchetes Q los elementos opcionales de la clase. También hay dos elementos opcionales etiquetados como algo_1 y algo_2 que contendrán palabras reservadas que se estudiarán en los siguientes capítulos. Es muy común ver definiciones de clases como *public class nombre_de_la_clase*. La palabra reservada *public* indica que la clase puede ser accedida por cualquier clase que necesite de su utilización. Entre los corchetes, dentro de la clase, encontraremos los atributos (una clase puede tener de cero a muchos atributos) y los métodos (una clase puede tener de cero a muchos métodos). Los métodos para la gente que haya programado en cualquier lenguaje de programación son los llamados procedimientos o funciones.



Recuerda

El nombre de la clase y del fichero que la contiene deberá de ser el mismo.

Objetos

Un objeto tiene una serie de características como son las siguientes:

- Identidad. Cada objeto es único y diferente de otro objeto. Mi loro Felipe es diferente a otros loros aunque estos sean verdes, tengan 2 años y sean también domésticos.
- Estado. El estado serán los valores de los atributos del objeto, en el caso de los objetos de la clase *pájaro* serían nombre, color, edad, doméstico, etc.
- Comportamiento. El comportamiento serían los métodos o procedimientos que realiza dicho objeto. Dependiendo del tipo o clase de objeto, estos realizarán unas operaciones u otras (cantar, volar, hablar, etc.).

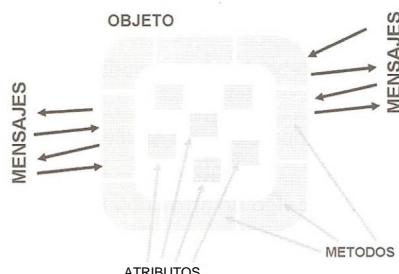


Figura 2.1. Estructura de un objeto

Mensajes

Como se dijo antes, los programas o aplicaciones orientadas a objetos están compuestos por objetos, los cuales interactúan unos con otros a través del paso de mensajes. Cuando un objeto recibe un mensaje lo que hace es ejecutar el método asociado.

Métodos

Los métodos son los procedimientos que ejecuta el objeto cuando recibe un mensaje vinculado a ese método concreto. En ocasiones este método envía mensajes a otros objetos, solicitando acciones o información.

**Recuerda**

En un programa OO primeramente se crean los objetos y entre ellos se envían mensajes procesándose la información para luego destruirse y liberar la memoria que estaban ocupando.

A FONDO**LOS OBJETOS**

Los objetos del mundo real tienen dos características principales:

- **Estado.**
- **Comportamiento.**

Los loros, por ejemplo, tienen un estado que puede ser el color del plumaje, el nombre, si hablan o no, etc. Y también un comportamiento (hablar, piar, comer, etc.).

Algunos objetos son más complejos que otros, por ejemplo, mi consola es mucho más compleja que mi linterna de espeleología. Mi linterna tiene solo dos estados (encendida y apagada) y la interfaz es sumamente sencilla (apagar y encender).

Al programar una aplicación en Java deberemos de modelar estos estados y comportamientos en clases y objetos.

La programación orientada a objetos proporciona los siguientes beneficios:

- **Modularidad.** El código fuente de un objeto puede mantenerse y reescribirse sin que ello implique la reprogramación del código de otros objetos de la aplicación.
- **Reutilización de código.** Es muy sencillo utilizar clases y objetos de terceras personas. La ventaja de esto es que no tenemos que conocer los detalles de su implementación interna sino solamente su interfaz.
- **Facilidad de testeo y reprogramación.** Si tenemos un objeto que está dando problemas en una aplicación no tenemos que reescribir el código de toda la aplicación, sino que tenemos que reemplazar el objeto por otro similar, o bien reprogramarlo.
- **Ocultación de información.** En la POO se ocultan los detalles de implementación y lo que priva es la interfaz.

2.2

CARACTERÍSTICAS DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

A continuación, se citan algunas de las características fundamentales de la programación orientada a objetos:

- Abstracción. Según la RAE abstraer es “separar por medio de una operación intelectual las cualidades de un objeto para considerarlas aisladamente o para considerar el mismo objeto en su pura esencia o noción”. Cuando se programa orientado a objetos lo que se hace es abstraer las características de los objetos que van a tomar parte del programa y crear las clases con sus atributos y sus métodos.
- Encapsulamiento. El encapsulamiento es una de las propiedades fundamentales de la programación orientada a objetos. Cuando se programa orientado a objetos, los objetos se ven según su comportamiento externo. Por ejemplo, en la clase *pájaro* se le puede enviar un mensaje para que cante y el objeto *pájaro* ejecutará su método cantar. Lo más interesante de todo esto es que el programador no tiene por qué saber cómo funciona internamente el método cantar, simplemente lo ejecuta.



Recuerda

En la POO las clases son vistas como una caja negra. Los programadores no tienen por qué saber nada de los datos que almacenan y el interior de los métodos que permiten manipularlos, solamente tienen que conocer su interfaz.

- Herencia. Todas las clases se estructuran formando jerarquías de clases.

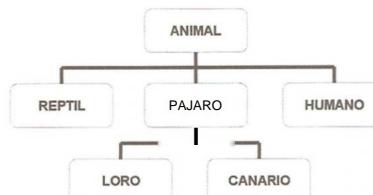


Figura 2.2. Jerarquía de clases

Las clases pueden tener superclases (la clase *pájaro* tiene la superclase *animal*) y subclases (la clase *pájaro* tiene las subclases *loro* y *canario*). También existe la posibilidad que una clase herede de varias superclases.



Importante

En Java una clase SOLO puede tener UNA superclase. Este hecho se denomina herencia simple. En C++ se permite la herencia múltiple. Java puede simular la herencia múltiple utilizando interfaces.

Cuando una clase hereda de una superclase obtiene los métodos y las propiedades de dicha superclase. Además, la funcionalidad propia de la misma clase se combinará con la heredada de la superclase.

- Polimorfismo. El polimorfismo permite crear varias formas del mismo método, de tal manera que un mismo método ofrezca comportamientos diferentes.

2.3 PROPIEDADES Y MÉTODOS DE LOS OBJETOS



Recuerda

Datos, propiedades o atributos son sinónimos y referencian a las variables de una clase.

En una clase se agrupan datos (variables) y métodos (funciones). Todas las variables o funciones creadas en Java deben pertenecer a una clase, con lo cual no existen variables o funciones globales como en otros lenguajes de programación.



Recuerda

En una clase Java los atributos pueden ser tipos primitivos (*char*, *int*, *boolean*, etc.) o bien pueden ser objetos de otra clase. Por ejemplo, un objeto de la clase *coche* puede tener un objeto de la clase *motor*.

En la siguiente clase se puede observar perfectamente los atributos de la clase y los métodos:

```
class pajaro
{
    //*** atributos o propiedades ***
    private char color; //propiedad o atributo color
    private int edad; //propiedad o atributo edad
    //*** métodos de la clase ***
    public void setedad(int e){edad = e;}
    public void printedad(){System.out.println(edad);}
    public void setcolor(char c){color=c;}
    public void printcolor(){
        switch(color){
            //Los pájaros son verdes, amarillos, grises, negros o blancos
            //No existen pájaros de otros colores
            case 'v': System.out.println("verde");break;
            case 'a': System.out.println("amarillo");break;
            case 'g': System.out.println("gris");break;
        }
    }
}
```

```
        case 'n': System.out.println("negro");break;
        case 'b': System.out.println("blanco");break;
        default: System.out.println("color no establecido");
    }
}

class test
{
    public static void main(String[] args) {
        pajaro p;
        p=new pajaro();
        p.setedad(5);
        p.printedad();
    }
}
```

Como se puede ver en el código anterior existen dos clases diferentes (*pájaro* y *test*), las cuales residirán en dos ficheros diferentes (pajaro.java y test.java). En la clase *test* se crea un objeto de la clase *pájaro* y se llama a los métodos para actualizar la edad y mostrarla por pantalla. En ningún momento la clase *test* puede acceder a los métodos o atributos *private* de la clase *pájaro* (esto es gracias a la abstracción), ni falta que le hace porque lo único que debe de conocer la clase *test* son los métodos públicos para utilizar la clase.

2.4

PROGRAMACIÓN DE LA CONSOLA: ENTRADA Y SALIDA DE INFORMACIÓN

Java permite la entrada por teclado y la salida de información a través de la pantalla mediante la clase *System* del paquete *java.lang*. La clase *System* contiene, entre otros tres objetos los cuales están asociados a tres flujos estándar que se abren cuando se ejecuta el programa y se cierran cuando éste finaliza. Estos objetos *static* son los siguientes:

- *System.out*. Referencia a la salida estándar (pantalla).
- *System.in*. Referencia a la entrada estándar (teclado). El objeto *System.in* pertenece a la clase *InputStream*. Un ejemplo de utilización es el siguiente:

```
char c;
try {
c = (char) System.in.read() ;
}catch(Exception e) {
e.printStackTrace() ;
}
```

**Recuerda**

El método **void printStackTrace()** indica el método donde se lanzó la excepción.

En el ejemplo anterior se llama al método *read()* del objeto *System.in* que es el método básico de lectura de un carácter por teclado. Generalmente, lo que se hace es leer una línea completa desde teclado, no un carácter nada más. Por lo tanto hay que utilizar el siguiente código:

```
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader buff = new BufferedReader(isr);
String ln = buff.readLine();
```

En este código se crea un objeto del tipo *InputStreamReader* al cual se le pasa como parámetro en el constructor el objeto *System.in* (que es un *InputStream*). Es una clase derivada de *Reader* y acepta en el constructor como parámetro un *InputStream*. En la siguiente línea, el objeto sigue//va a pedir un *reader* al crearse y aprovechamos para pasarle como parámetro el *InputStreamReader* (*isr*) que hemos creado anteriormente.

**Truco**

Otra forma de leer datos desde teclado es el siguiente:

```
String sdato = System.console().readLine(); //lectura de un string.
dato = Integer.parseInt(sdato); //transformación del string en entero.
```

- *System.err*. Referencia a la salida de error estándar que, al igual que la salida estándar, es la pantalla. Utilizado para mostrar mensajes de error. Al igual que *System.out* tiene los métodos *print()* y *println()*. Un ejemplo de utilización de este objeto es el siguiente:

```
int a=10, b=0, c;
try {
    c=a/b;
}
catch (ArithmetricException e) {
    System.err.println("Error: "+e.getMessage());
    return;
}
System.out.println("Resultado: "+c) ;
```

2.5 PARÁMETROS Y VALORES DEVUELTOS

Los métodos pueden permitir que se los llame especificando una serie de valores. A estos valores se les denomina parámetros. Los parámetros pueden tener un tipo básico (*char*, *int*, *boolean*, etc.) o bien ser un objeto. Además, los métodos (salvo el constructor) pueden retornar un valor o no (en ese caso se pone **void**).

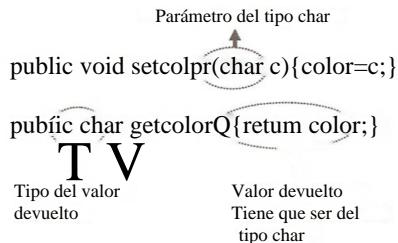


Figura 2.3. Parámetros y valores devueltos por un método

En la figura anterior se pueden ver dos métodos para la clase *pájaro*, *setcolor* la cual admite un parámetro y *getcolor* la cual devuelve un valor de tipo *char*.

2.6 CONSTRUCTORES Y DESTRUCTORES DE OBJETOS

En Java existen unos métodos especiales que son los constructores y destructores del objeto. Estos métodos son opcionales, es decir, no es obligatorio programarlos salvo que se necesiten.

- El constructor del objeto es un procedimiento llamado automáticamente cuando se crea un objeto de esa clase. Si el programador no los declara, Java generará uno por defecto. La función del constructor es inicializar el objeto.
- El destructor, por el contrario, se ejecutará automáticamente siempre que se destruye un objeto de dicha clase. Los destructores, generalmente, se utilizan para liberar recursos y cerrar flujos abiertos (realiza una limpieza final). Los destructores no reciben parámetros, al contrario que los constructores, la sobrecarga no está permitida. En C++ destructor se denomina igual que el constructor nada más que se le coloca delante el símbolo ~. En Java, la destrucción de objetos sigue otra filosofía distinta a la de otros lenguajes de programación. El sistema de destrucción de objetos ya se verá más adelante en profundidad.



Importante: Destructores en Java

En Java NO hay destructores como en C++.

En el siguiente código se verán los constructores para la clase *pájaro*:

```
class pajaro
{
    //*** atributos o propiedades ****
    private char color; //propiedad o atributo color
    private int edad; //propiedad o atributo edad
    //*** métodos de la clase ****
    pajaro(){color = 'v'; edad = 0;} //constructor de la clase pájaro
    pajaro(char c, int e){color = c; edad = e;} // constructor de la clase pájaro
    /* Aquí irán los demás métodos de la clase */
    public static void main(String[] args) { //método main
        pajaro p1,p2;
        p1=new pajaro();
        p2=new pajaro('a',3);
    }
}
```

Como se puede ver, el constructor de la clase *pájaro* está sobrecargado. Es posible crear objetos de la clase *pájaro* de distintas formas.

2.7 USO DE MÉTODOS ESTÁTICOS Y DINÁMICOS

En este apartado se va a ver cuándo un método o atributo debe de ser estático y cuándo no. Cuando un método o atributo se define como *static* quiere decir que se va a crear para esa clase solo una instancia de ese método o atributo. En el siguiente caso se ve como se ha creado un atributo *numpajaros* que contará el número de pájaros que se van generando. Si ese atributo no fuera estático sería imposible contar los pájaros, puesto que en cada instancia del objeto se crearía una variable *numpajaros*. De la misma manera, el método *nuevopajaro()*, *muestrapajaros()* o el método *main()* tienen sentido que sean estáticos.

```
class pajaro
{
    //*** atributos o propiedades ****
    private static int numpajaros=0;
    private char color; //propiedad o atributo color
    private int edad; //propiedad o atributo edad
    //*** métodos de la clase ****
    static void nuevopajaro(){numpajaros++;}
    pajaro () {color = 'v'; edad = 0; nuevopajaro();}
    pajaro(char c, int e){color = c; edad = e; nuevopajaro();}
    static void muestrapajaros(){System.out.println(numpajaros);}
    public static void main(String[] args) {
        pajaro p1,p2;
        p1=new pajaro();
        p2=new pajaro('a', 3);
```

```

    pl.muéstrapajaros();
    p2.muestrapajaros();
}

}

```

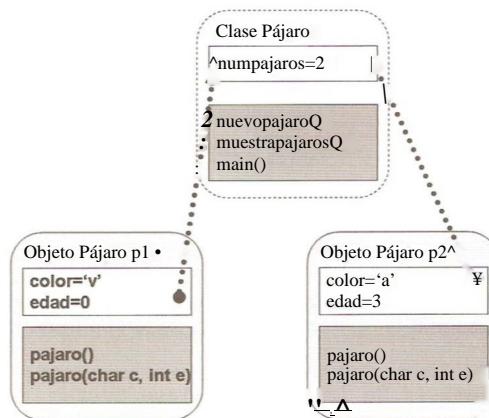


Figura 2.4. Clase pájaro y objetos de dicha clase

Como se puede ver en la figura anterior, el atributo `numpajaros` y los métodos `nuevopajaro()`, `muestrapajaros()` y `main()` se comparten por todos los objetos creados de la clase `pájaro`.

2.8 LIBRERÍAS DE OBJETOS (PAQUETES)



Recuerda

Librería o paquete es lo mismo.

Un paquete o package es un conjunto de clases relacionadas entre sí, las cuales están ordenadas de forma arbitraria. Las clases que forman parte de un paquete no derivan todas ellas de una misma superclase. Por ejemplo, el paquete `java.io` agrupa las clases que permiten a un programa realizar la entrada y salida de información. Un paquete también puede contener a otros paquetes. Con el uso de paquetes se evitan conflictos, como llamar dos clases con el mismo nombre (si existen, estarán cada una en paquetes diferentes). Al estar en el mismo paquete, las clases de dichos paquetes tendrán un acceso privilegiado a los miembros de dato y métodos de otras clases del mismo paquete.

**Recuerda**

Gracias a los paquetes es posible organizar las clases en grupos. Las clases de un mismo paquete están relacionadas entre sí.

A FONDO**LA SENTENCIA IMPORT**

Imaginemos que queremos utilizar una clase contenida en algún paquete, la forma de utilizar dicha clase es generalmente utilizando la sentencia import. Podemos importar una clase individual, como por ejemplo:

```
import java.lang.System; // Se importa la clase System
```

O bien podemos importar todas las clases de un paquete:

```
import java.awt.*;
```

En este caso podremos utilizar todas las clases del paquete. Un ejemplo de esto sería el siguiente:

```
import java.awt.*;
```

```
Frame fr = new Frame( "Panel ejemplo" );
```

También es posible utilizar la clase sin utilizar la sentencia import:

```
java.awt.Frame fr = new java.awt.Frame( "Panel ejemplo" );
```

Generalmente, cuando se van a crear varios objetos de una o varias clases de un paquete no se utiliza esta última opción porque hay que escribir mucho código.

**Un applet**

Un *applet* es una aplicación Java que se ejecuta en la ventana de un navegador. Los *applets* se ejecutan en la máquina del cliente y nunca en el servidor.

Tabla 2.1. Librerías Java

Paquete o librería	Descripción
java.io	Librería de Entrada/Salida. Permite la comunicación del programa con ficheros y periféricos.
java.lang	Paquete con clases esenciales de Java. No hace falta ejecutar la sentencia import para utilizar sus clases. Librería por defecto.
java.útil	Librería con clases de utilidad general para el programador.
java.applet	Librería para desarrollar <i>applets</i> .
java.awt	Librerías con componentes para el desarrollo de interfaces de usuario.
java.swing	Librerías con componentes para el desarrollo de interfaces de usuario. Similar al paquete awt.
java.net	En combinación con la librería java.io, va a permitir crear aplicaciones que realicen comunicaciones con la red local e Internet.
java.math	Librería con todo tipo de utilidades matemáticas.
java.sql	Librería especializada en el manejo y comunicación con bases de datos.
java.security	Librería que implementa mecanismos de seguridad.
java.rmi	Paquete que permite el acceso a objetos situados en otros equipos (objetos remotos).
java.beans	Librería que permite la creación y manejo de componentes <i>javabeans</i> .

**Un javabean**

Un *javabean* es un componente reutilizable que encapsula varios objetos en uno solo. *Bean* es una vaina en inglés y, como su nombre indica, permite tener un único objeto en vez de varios más simples.

2.8.1 LOCALIZACIÓN DE LIBRERÍAS

Para encontrar una clase u otro recurso Java necesita dos cosas:

- El nombre del paquete.
- Las rutas donde están situados los paquetes y las clases (*path* de búsqueda más conocido como CLASSPATH).

El CLASSPATH sirve para localizar clases creadas por el usuario o terceras personas que no son parte de la plataforma Java. Establecer el valor de esta variable es necesario cuando se va a utilizar una clase que no está en

el mismo directorio (o subdirectorios) de la clase donde se está trabajando o no está en ningún lugar definido por el mecanismo de extensiones.

Una alternativa a establecer el valor de la variable de entorno CLASSPATH es utilizar las opciones —cp o -classpath (es lo mismo) al ejecutar Java (java, javac, javah o jdb).

Imaginemos que tenemos un paquete utilidades.proyecto y dentro de él una clase que se llama programa.class. Esta clase se encuentra en la siguiente ruta:

```
/java/Misclases/utilidades/proyecto
```

Para ejecutar la clase deberíamos de ejecutar el siguiente comando:

```
% java -classpath /java/Misclases utilidades.proyecto.programa
```

Otra opción es establecer la variable de entorno CLASSPATH

```
CLASSPATH = /java/Misclases:path2:path3:...  
export CLASSPATH
```

Y luego ejecutar el siguiente comando:

```
% java utilidades.proyecto.programa
```

Java ya se encargaría de buscar la clase en el directorio especificado por la variable CLASSPATH.

**Importante**

Los valores de la variable de entorno CLASSPATH están separados por dos puntos en Linux/Unix y por punto y coma en sistemas Windows®.



RESUMEN DEL CAPÍTULO

En este tema se va a abordar las características básicas de la programación orientada a objetos. El objetivo del mismo es comenzar a trabajar con la orientación a objetos mediante las características más básicas de la misma. El alumno deberá de poner interés en el aprendizaje de estos conceptos que serán desarrollados más en profundidad en los capítulos 4 y 5 de este libro. Una vez trabajado el tema, el alumno será capaz de resolver problemas básicos utilizando clases con métodos sencillos.



EJERCICIOS RESUELTOS

- 1. Realiza una clase *Temperatura*, la cual convierta grados Celsius a Farenheit y viceversa. Para ello crea dos métodos *double celsiusToFarenheit(double)* y *double farenheitToCelsius(double)*.

En la construcción ten en cuenta las siguientes fórmulas:

- Farenheit a Celsius $C = (F - 32)/1,8$
- Celsius a Farenheit $F = (1,8)C + 32$

Solución:

```
class Temperatura {
    public static double celsiusToFarenheit(double temp)
    {
        return (1.8)*temp + 32;
    }
    public static double farenheitToCelsius(double temp)
    {
        return (temp - 32)/1.8;
    }
    public static void main(String[] args) {
        System.out.println (" 0 grados Celsius son "+celsiusToFarenheit(0)+" Grados Farenheit");
        System.out.println("15 grados Celsius son "+celsiusToFarenheit(15)+" Grados Farenheit");
        System.out.println("20 grados Celsius son "+celsiusToFarenheit(20)+" Grados Farenheit");
        System.out.println("0 grados Farenheit son "+farenheitToCelsius(0)+" Grados Celsius");
        System.out.println("40 grados Farenheit son "+farenheitToCelsius(45)+" Grados Celsius");
        System.out.println("70 grados Farenheit son " + farenheitToCelsius(70)+"Grados Celsius");
    }
}
```

El programa anterior dará la siguiente salida:

0 grados Celsius son 32.0 grados Farenheit.

15 grados Celsius son 59.0 grados Farenheit.

20 grados Celsius son 68.0 grados Farenheit.

0 grados Farenheit son -17.777777777778 grados Celsius.

40 grados Farenheit son 7.22222222222222 grados Celsius.

70 grados Farenheit son 21.11111111111111 grados Celsius.

- 2. Tenemos la siguiente clase *coche*:

```
class coche {  
    private int velocidad;  
    coche (){velocidad=0;}  
}
```

Añade a la clase *coche* los siguientes métodos:

int getVelocidad(). Este método devuelve la velocidad actual.

void acelera(int mas). Este método actualiza la velocidad a mas kilómetros más.

void frenar(int menos). Este método actualiza la velocidad a menos kilómetros menos.

Solución:

```
class coche {  
    private int velocidad;  
    coche (){velocidad=0;}  
    public int getVelocidad() {return velocidad;}  
    public void acelera(int mas){velocidad+=mas;}  
    public void frenar(int menos){velocidad-=menos;}  
}
```

- 3. Averigua si son verdaderas o falsas las siguientes afirmaciones:

1. Las clases que forman parte de un paquete deben derivar todas ellas de una misma superclase.
2. En una clase Java los atributos pueden ser tipos primitivos o bien pueden ser objetos de otra clase.
3. Cuando se escribe un programa o aplicación orientada a objetos, lo que se hace es definir las clases de objetos dotándolas de estado y comportamiento y cuando se ejecute el programa se crearán los objetos, ya sea estática o dinámicamente.
4. La ocultación de información significa que al proporcionar los ficheros .class, el programador no tiene por qué proporcionar los ficheros .java donde reside todo el código.
5. Un paquete o *package* es un conjunto de clases relacionadas entre sí, las cuales están ordenadas de forma arbitraria.
6. Un *javabeans* es un componente reutilizable que encapsula varios objetos en uno solo.
7. El método *void printStackTrace()* indica el método donde se lanzó la excepción.
8. La abstracción en POO es una de las propiedades fundamentales de la misma mediante la cual los objetos se ven según su comportamiento externo.

9. El polimorfismo permite crear varias formas del mismo método, de tal manera que un mismo método ofrezca comportamientos idénticos pero con distinta forma.
 10. Las clases se escriben en ficheros ASCII. El nombre del fichero puede ser cualquiera, pero lo importante es que la extensión sea .java.
 11. En un programa orientado a objetos primeramente se crean los objetos y entre ellos se envían mensajes procesándose la información, para luego destruirse y liberar la memoria que estaban ocupando.
- 4. ¿Está correctamente definida la siguiente clase? ¿Compilará o habrá que modificarla para poder generar el fichero .class?

```
class pajaro {  
    public void setEdad(int e){edad = e;}  
    public void printEdad(){System.out.println(edad);}  
    public void setColor(char c){color=c;}  
    private char color;  
    private int edad;  
}
```

Solución:

Realmente la clase compila y funcionará sin problemas. No obstante, los atributos se suelen colocar por convenio en la parte superior del cuerpo de la clase y los métodos en la parte inferior. No es común encontrarse los atributos al final.

- 5. La siguiente clase tiene problemas de compilación:

```
public class satélite {  
    private double meridiano;  
    private double paralelo;  
    private double distancia_tierra;  
    satélite (double m,double p,double d){  
        meridiano=m;  
        paralelo=p;  
        distancia_tierra=d;  
    }  
    satélite (){  
        meridiano=paralelo=distancia_tierra=0;  
    }  
    public void setPosicion(double m,double p,double d){  
        meridiano=m;  
        paralelo=p;  
        distancia_tierra=d;  
    }  
    public void printPosicion(){
```

```

        System.out.println(<El satélite se encuentra en el paralelo "+ paralelo+
Meridiano "+meridiano+ " a una distancia de la tierra de "+distancia_tierra*
"Kilómetros");
    }
}

```

Averigua los problemas y corrígelos.

Solución:

La solución se encuentra al final de los ejercicios propuestos.

- 6. Crea una clase *rebajas* con un método *descubrePorcentaje()* que descubra el descuento aplicado en un producto. El método recibe el precio original del producto y el rebajado y haya el porcentaje.

Solución:

```

public class rebajas {
    public static double descubrePorcentaje(double original, double actual) {
        return(original-actual)*100/original;
    }
    public static void main(String[] args) {
        System.out.println(descubrePorcentaje(100,79));
        System.out.println(descubrePorcentaje(100,50));
    }
}

```

1

EJERCICIOS PROPUESTOS

- 1. Realiza una clase *finanzas* que convierta dólares a euros y viceversa. Codifica los métodos *dolaresToEuro()* y *eurosToDolares()*. Prueba que dicha clase funciona correctamente haciendo conversiones entre euros y dólares. La clase tiene que tener:
 - Un constructor *finanzas()* por defecto el cual establecerá el cambio Dólar-Euro en 1.36.
 - Un constructor *finanzas(double)*, el cual permitirá configurar el cambio dólar-euro.
- 2. Realiza una clase *minumero* que proporcione el doble, triple y cuádruple de un número proporcionado en su constructor (realiza un método para doble, otro para triple y otro para cuádruple). Haz que la clase tenga un método *main* y comprueba los distintos métodos.

■ 3. Realiza una clase *número* que almacene un número entero y tenga las siguientes características:

- Constructor por defecto que inicializa a 0 el número interno.
- Constructor que inicializa el número interno.
- Método añade que permite sumarle un número al valor interno.
- Método resta que resta un número al valor interno.
- Método getValor. Devuelve el valor interno.
- Método getDoble. Devuelve el doble del valor interno.
- Método getTriple. Devuelve el triple del valor interno.
- Método setNúmero. Inicializa de nuevo el valor interno.

■ 4. Empareja cada paquete con su correspondiente descripción:

Tabla 2.2. Tabla ejercicio 4

Paquete o librería	Descripción
java.security	Paquete con clases esenciales de Java. No hace falta ejecutar la sentencia import para utilizar sus clases. Librería por defecto.
java.rmi	Librería especializada en el manejo y comunicación con bases de datos.
java.beans	Librería con clases de utilidad general para el programador,
java.applet	Librería para desarrollar <i>applets</i> .
java.math	Librería que implementa mecanismos de seguridad.
java.sql	Paquete que permite el acceso a objetos situados en otros equipos (objetos remotos).
java.net	Librería de Entrada/Salida. Permite la comunicación del programa con ficheros y periféricos.
java.awt	Librerías con componentes para el desarrollo de interfaces de usuario.
java.swing	Librería que permite la creación y manejo de componentes <i>javabeans</i> .
java.io	Librerías con componentes para el desarrollo de interfaces de usuario. Similar al paquete awt.
java.lang	En combinación con la librería java.io, va a permitir crear aplicaciones que realicen comunicaciones con la red local e Internet.
java, útil	Librería con todo tipo de utilidades matemáticas.

- 5. Modificación del ejercicio resuelto número 5.
 - Modifica la clase *satélite* y añádele los siguientes métodos:
 - Método *void variaAltura(double desplazamiento)*. Este método acepta un parámetro que será positivo o negativo dependiendo de si el satélite tiene que alejarse o acercarse a La Tierra.
 - Método *boolean enOrbitaQ*. Este método devolverá *false* si el satélite está en tierra y *true* en caso contrario.
 - Método *void variaPosicion(double variap, double variam)*. Este método permite modificar los atributos de posición (meridiano y paralelo) mediante los parámetros variap y variam. Estos parámetros serán valores positivos o negativos relativos que harán al satélite modificar su posición.
- 6. (Ejercicio de dificultad alta) Crea la clase *peso*, la cual tendrá las siguientes características:
 - Deberá tener un atributo donde se almacene el peso de un objeto en kilogramos.
 - En el constructor se le pasará el peso y la medida en la que se ha tomado ('Lb' para libras, 'Li' para lingotes, 'Oz' para onzas, 'P' para peniques, 'K' para kilos, 'G' para gramos y 'Q' para quintales).
 - Deberá de tener los siguientes métodos:
 - *getLibras*. Devuelve el peso en libras.
 - *getLingotes*. Devuelve el peso en lingotes.
 - *getPeso*. Devuelve el peso en la medida que se pase como parámetro ('Lb' para libras, 'Li' para lingotes, 'Oz' para onzas, 'P' para peniques, 'K' para kilos, 'G' para gramos y 'Q' para quintales).
 - Para la realización del ejercicio toma como referencia los siguientes datos:
 - 1 Libra = 16 onzas = 453 gramos.
 - 1 Lingote = 32,17 libras = 14,59 kg.
 - 1 Onza = 0,0625 libras = 28,35 gramos.
 - 1 Penique = 0,05 onzas = 1,55 gramos.
 - 1 Quintal = 100 libras = 43,3 kg.
 - Crea además un método *main* para testear y verificar los métodos de esta clase.
- 7. Crea una clase con un método *millasAMetros()* que toma como parámetro de entrada un valor en millas marinas y las convierte a metros.
 - Una vez tengas este método escribe otro *millasAKilometros()* que realice la misma conversión, pero esta vez exprese el resultado en kilómetros.
- Nota: 1 milla marina equivale a 1852 metros.
- 8. Crea la clase *coche* con dos constructores. Uno no toma parámetros y el otro sí. Los dos constructores inicializarán los atributos marca y modelo de la clase. Crea dos objetos (cada objeto llama a un constructor distinto) y verifica que todo funciona correctamente.
- 9. Implementa una clase *consumo*, la cual forma parte de la centralita electrónica de un coche y tiene las siguientes características:
 - Atributos:

- kms. Kilómetros recorridos por el coche,
- litros. Litros de combustible consumido,
- vmed. Velocidad media,
- pgas. Precio de la gasolina.

■ Métodos:

- getTiempo. Indicará el tiempo empleado en realizar el viaje.
- consumoMedio. Consumo medio del vehículo (en litros cada 100 kilómetros).
- consumoEuros. Consumo medio del vehículo (en euros cada 100 kilómetros).

No olvides crear un constructor para la clase que establezca el valor de los atributos. Elige el tipo de datos más apropiado para cada atributo.

■ 10. Para la clase anterior implementa los siguientes métodos, los cuales podrán modificar los valores de los atributos de la clase:

- setKms
- setLitros
- setVmed
- setPgas

■ 11. (Ejercicio de dificultad alta) El restaurante mexicano de Israel cuya especialidad son las papas con chocos nos pide diseñar un método con el que se pueda saber cuántos clientes pueden atender con la materia prima que tienen en el almacén. El método recibe la cantidad de papas y chocos en kilos y devuelve el número de clientes que puede atender el restaurante teniendo en cuenta que por cada tres personas, Israel utiliza un kilo de papas y medio de chocos.

■ 12. Modifica el programa anterior creando una clase que permita almacenar los kilos de papas y chocos del restaurante. Implementa los siguientes métodos:

- *public void addChocos(int x)*. Añade x kilos de chocos a los ya existentes.
- *public void addPapas(int x)*. Añade x kilos de papas a los ya existentes.
- *public int getComensales()*. Devuelve el número de clientes que puede atender el restaurante (este es el método anterior).
- *public void showChocos()*. Muestra por pantalla los kilos de chocos que hay en el almacén.
- *public void showPapas()*. Añade Muestra por pantalla los kilos de papas que hay en el almacén.

Solución al ejercicio resuelto número 3:

- | | | |
|--------------|--------------|---------------|
| 1. Falsa | 5. Verdadera | 9. Falsa |
| 2. Verdadera | 6. Verdadera | 10. Falsa |
| 3. Verdadera | 7. Verdadera | 11. Verdadera |
| 4. Falsa | 8. Falsa | |

Solución al ejercicio resuelto número 5:

```
public class satélite { //class se escribe con doble s
    private double meridiano;
    private double paralelo; //falta el ; del final
    private double distancia_tierra;
    satélite (double m,double p,double d){
        meridiano=m;
        paralelo=p;
        distancia_tierra=d;
    }
    satélite (){ //cuidado con los acentos
        meridiano=paralelo=distancia_tierra=0;
    }
    public void setPosicion(double m,double p,double d){
        //los parámetros siempre separados por comas
        meridiano=m;
        paralelo=p;
        distancia_tierra=d; //variable mal escrita
    }
    public void printPosicion(){
        System.out.println(«El satélite se encuentra en el paralelo «+
        « Meridiano «+meridiano+» a una distancia de la tierra de «+distancia.
        Kilómetros»);
        // forma correcta de mostrar el estado del satélite
    }
}
```

paralelo+
_tierra+»