Introducción

Este libro tiene como finalidad servir de referencia al lector en los certificados de profesionalidad. El objetivo del libro es complementar los contenidos teóricos con la parte práctica de los mismos.

La estructura del libro es tan didáctica como la materia lo permite. Los conceptos presentados en el libro son fáciles de comprender y se acompañan de muchas fotos, ejemplos y explicaciones sencillas. En los contenidos teóricos se ha intentado exponer los conceptos de una forma simplificada, incluyendo siempre los más importantes. Así mismo, se ha procurado que los ejemplos sean lo más sencillos e intuitivos posible.

El libro trata conceptos muy interesantes, como el paradigma de la programación orientada a objetos, el lenguaje Java como un ejemplo de lenguaje orientado a objetos, HTML, CSS, JavaScript, PHP, ingeniería del software, etc.

El lector, para dominar la materia, además de manejar el libro, deberá investigar, realizar ejercicios, documentarse y ampliar conocimientos por sí mismo, puesto que este libro solamente es el empujón en la salida de una carrera ciclista, luego el alumno tendrá que pedalear y recorrer muchos kilómetros solo.

1 Introducción al paradigma orientado a objetos. Clases y objetos

La orientación a objetos es el paradigma de programación más utilizado actualmente. Es difícil programar en un lenguaje que no permita en mayor o menor medida la orientación a objetos. De hecho, muchos lenguajes clásicos se han adaptado a este paradigma para seguir estando de actualidad y no desaparecer.

En este capítulo se introduce al lector en el paradigma de la orientación a objetos y se estudiarán los conceptos de clase y objeto con ejemplos en Java, entre otros. En capítulos posteriores se profundizará más en estos conceptos, pues son la base de la POO.

1.1 EL PARADIGMA DE LA PROGRAMACIÓN ORIENTADA A OBJETOS

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 última generación permiten la programación orientada a objetos, así como la programación clásica. Por esta razón, puede entenderse la POO como una evolución de la programación clásica (programación estructurada).

1.1.1 CICLO DE DESARROLLO DEL SOFTWARE BAJO EL PARADIGMA DE ORIENTACIÓN A OBJETOS: ANÁLISIS, DISEÑO Y PROGRAMACIÓN ORIENTADA A OBJETOS

La programación orientada a objetos rompe definitivamente con el paradigma de la programación estructurada, surge el concepto de "objeto". Todos los programas se reducen a objetos que tienen una serie de atributos, y, asociados, un comportamiento o procedimientos llamados "métodos". Estos objetos, que son instancias de clases, interaccionarán unos con otros y de esa manera se diseñarán aplicaciones y programas.

Hoy en día es difícil programar sin utilizar programación orientada a objetos (POO). Esta forma de programar es más cercana al modo en que expresaríamos las cosas en la vida real que la que se da en otros tipos de programación clásicos. Los analistas y programadores deben pensar las cosas de una manera distinta para plasmar dichos conceptos en programas en términos de objetos, atributos y métodos.

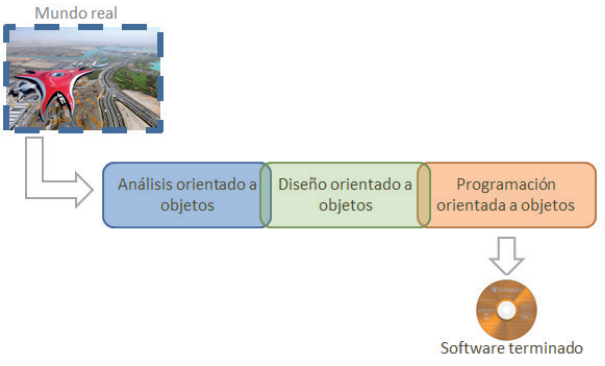

Figura 1.1. *Ciclo de desarrollo bajo el paradigma de la OO*

El primer paso en el ciclo de desarrollo orientado a objetos es el análisis orientado a objetos del mundo real. Una vez resuelta la fase de análisis se comenzará a diseñar los objetos en detalle para luego pasar a programarlos. Al contrario que en ciclos de desarrollo clásico, donde las fases tenían un comienzo y final claro, las fases definidas en el ciclo de desarrollo orientado a objetos se solapan entre sí. Una vez completadas estas tres fases el producto final es el software terminado, el cual será un compendio de todas las clases de la aplicación.

En Java las aplicaciones son sumamente modulares, el programador creará una serie de programas compuestos por clases. Cada clase está definida en un fichero `.java` independiente que al compilarse generará un fichero `.class`. Para llevar a cabo esta transformación, el programador necesitará de un programa especial (llamado "compilador") que transformará el código Java en unos ficheros `.class` que podrán ser ejecutados en un entorno Java.

EL COMPILADOR DE JAVA

El compilador de Java (javac) viene dentro de un paquete de desarrollo denominado **JDK** (*Java Development Kit* - Kit de desarrollo Java). Si queremos desarrollar programas en Java, como mínimo deberemos instalar un JDK en la máquina donde vayamos a desarrollar

Este proceso se ve en la siguiente figura:

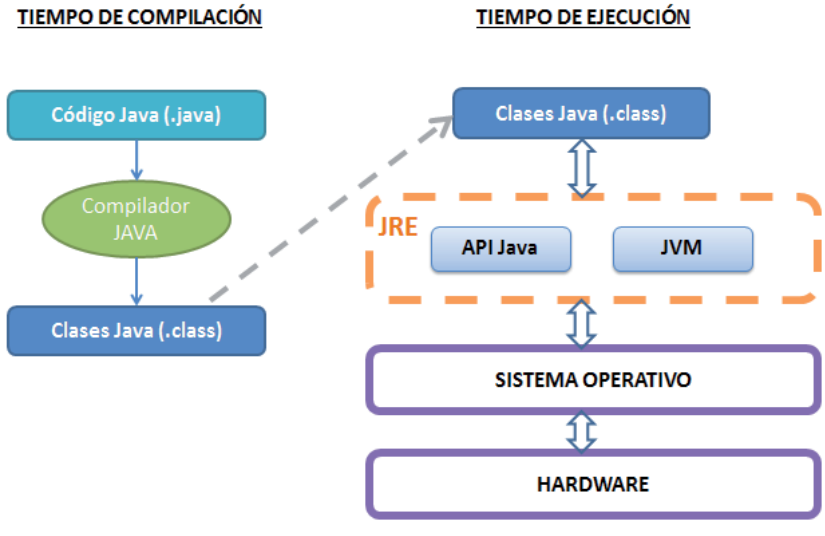

Figura 1.2. *Proceso de compilación y ejecución de un programa Java*

Una vez que el programador compila su programa, el siguiente paso es ejecutarlo. Para hacerlo, el sistema en donde se ejecute deberá tener un **JRE** (*Java Runtime Environment* - Entorno Java de ejecución), el cual contendrá una **JVM** (*Java Virtual Machine* - Máquina virtual Java). A diferencia de otros lenguajes de programación —en los que hay que compilar cada programa para cada sistema operativo—, cuando creas un programa en Java (lo compilas), este funcionará en cualquier sistema siempre y cuando tenga instalada la JVM correspondiente. Cada sistema operativo tendrá una JVM diferente.

¿DISPUESTO A PROGRAMAR?

Entonces descarga e instala el JDK y el JRE en tu máquina.

1.1.2 PROCESO DE CONSTRUCCIÓN DE SOFTWARE: MODULARIDAD. PAQUETES Y LIBRERÍAS

Cuando hablamos de modularidad entendemos por tal la capacidad que tiene una aplicación de ser dividida en varias partes más pequeñas, las cuales tienen que ser independientes unas de otras e incluso deben poder compilarse por separado.

Java —mediante los paquetes— permite modularizar las aplicaciones en varios trozos, los cuales están bien definidos y a la postre pueden servir para reutilizar código.

1.1.2.1 La modularidad en Java: los paquetes

RECUERDA

Librería o paquete son conceptos similares.

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, estará 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.

1.1.2.2 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 `awt` (fíjate en el asterisco). 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`, lo cual muchas veces no es aconsejable puesto que el código se hace muy voluminoso:

```
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, se desaconseja esta última opción.

1.1.2.3 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 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 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.

1.1.3 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 este caso, *Felipe*), un color de plumaje (verde), una edad (en este caso, dos años) y si son domésticos o no (*Felipe* sí es doméstico: lo tengo ahora en mi hombro).

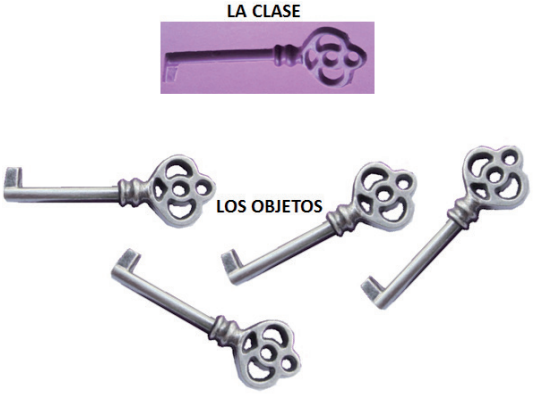

Figura 1.3. *Símil visual de lo que serían una clase y varios objetos de dicha clase*

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, con lo cual "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 han etiquetado con corchetes (`[ ]`) 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 cero o muchos atributos) y los métodos (una clase puede tener cero o muchos métodos). Los métodos –para la gente que haya programado en cualquier lenguaje de programación– son los llamados "procedimientos" o "funciones".

RECUERDA

En Java, el nombre de la clase y el del fichero que la contiene deberá ser el mismo.

- **Objetos**. Un objeto tiene las siguientes características:
  - **Identidad. Cada objeto es único y diferente de otro objeto. Mi loro Felipe es diferente a otros loros, aunque estos sean verdes, tengan dos 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.).**

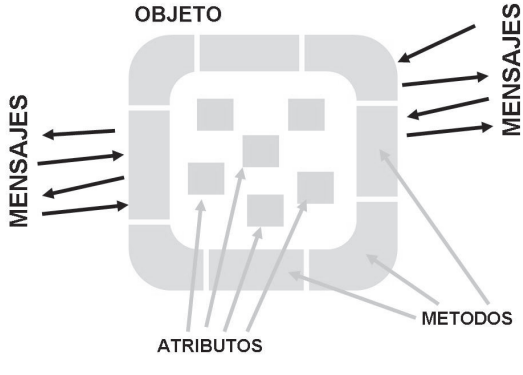

Figura 1.4. *Estructura de un objeto*

- **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, la información se procesa para luego destruirse y liberar la memoria que estaban ocupando.

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 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: basta con reemplazar el objeto por otro similar, o bien reprogramarlo.

- **Ocultación de información**. En la POO se ocultan los detalles de implementación y lo que prima es la interfaz.

1.2 CLASES Y OBJETOS

En la programación orientada a objetos, las clases permiten a los programadores abstraer el problema que se busca resolver ocultando los datos y la manera en la que estos se manejan para llegar a la solución (se oculta la implementación). En los siguientes apartados se estudiarán en profundidad los conceptos de clase y objeto.

1.2.1 LAS CLASES. PROPIEDADES Y MÉTODOS

En un programa orientado a objetos es impensable que desde el mismo programa se acceda directamente a las variables internas de una clase si no es a través de métodos **getters** y **setters** (por ejemplo `getEdad()` o `setEdad()`).

IMPORTANTE

La abstracción es importante en el análisis y diseño de aplicaciones orientadas a objetos. La finalidad del A&D es crear un conjunto de clases que resuelvan el problema que se está abordando.

Por lo tanto, en la definición de nuestras clases deberemos cuidar lo siguiente:

- No se deberá tener acceso directo a la estructura interna de las clases. El acceso a los atributos será a través de *getters* y *setters*.
- En el supuesto de que haya que modificar el código sin modificar la interfaz con otras clases o programas, esto debería poder hacerse sin que tenga ninguna repercusión con otras clases o programas. Se busca que las clases tengan un alto grado de cohesión (independencia).

En Java hay varios niveles de acceso a los miembros de una clase:

- **`public`** (acceso público).
- **`protected`** (acceso protegido).
- **`private`** (acceso privado).
- **`no especificado`** (acceso en su paquete).

Cuando especificamos el nivel de acceso a un atributo o método de una clase, lo que estamos especificando es el nivel de accesibilidad que va a tener ese atributo o método, que puede ir desde el acceso más restrictivo (`private`) al menos restrictivo (`public`).

RECUERDA

Una subclase es una clase que hereda ciertas características de la clase padre, aunque puede añadir algunas propias. Las subclases se estudiarán en profundidad más adelante.

Dependiendo de la finalidad de la clase, utilizaremos un tipo de acceso u otro.

- **Acceso público** (`public`). Un miembro público puede ser accedido desde cualquier otra clase o subclase que necesite utilizarlo. Una interfaz de una clase estará compuesta por todos los miembros públicos de la misma.
- **Acceso privado** (`private`). Un miembro privado puede ser accedido solamente desde los métodos internos de su propia clase. Otro acceso será denegado.
- **Acceso protegido** (`protected`). El acceso a estos miembros es igual que el acceso privado. No obstante, para las subclases o clases del mismo paquete (*package*) a la que pertenece la clase, se considerarán estos miembros como públicos.
- **Acceso no especificado** (`paquete`). Los miembros no etiquetados podrán ser accedidos por cualquier clase perteneciente al mismo paquete.

CONSEJO

Para un mayor control de acceso se recomienda etiquetar los miembros de una clase como `public`, `private` y `protected`.

A modo de resumen se especificarán los niveles de acceso vistos anteriormente en la siguiente tabla:

Modificador de acceso	`public`	`protected`	`private`	Sin especificar (acceso paquete)
¿El método o atributo es accesible desde la propia clase?	SÍ	SÍ	SÍ	SÍ
¿El método o atributo es accesible desde otras clases en el mismo paquete?	SÍ	SÍ	NO	SÍ
¿El método o atributo es accesible desde una subclase en el mismo paquete?	SÍ	SÍ	NO	SÍ
¿El método o atributo es accesible desde subclases en otros paquetes?	SÍ	(*)	NO	NO
¿El método o atributo es accesible desde otras clases en otros paquetes?	SÍ	NO	NO	NO

(*) Este caso no se suele dar con frecuencia. Se podría acceder al atributo o método desde objetos de la subclase, pero no así por objetos de la superclase.

Tabla 1.1. *Modificadores de acceso en Java*

Cuando creamos una clase en Java es posible definir la relación que esa clase tiene con otras clases o la relación que tendrá esa clase con las clases de su mismo paquete.

CONSEJO

Una clase definida como pública puede ser utilizada por las clases de su paquete y otros paquetes mientras que una clase no definida como pública solamente podrá ser utilizada por las clases de su propio paquete.

Clase pública	Clase NO definida como pública
`public class miClase { ..... }`	`class miClase { ..... }`
Puede ser utilizada por cualquier clase.	Puede ser utilizada SOLO por clases de su propio paquete.

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 conocer la clase `test` son los métodos públicos para utilizar la clase.

1.2.2 OBJETOS: ESTADO, COMPORTAMIENTO E IDENTIDAD

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`).

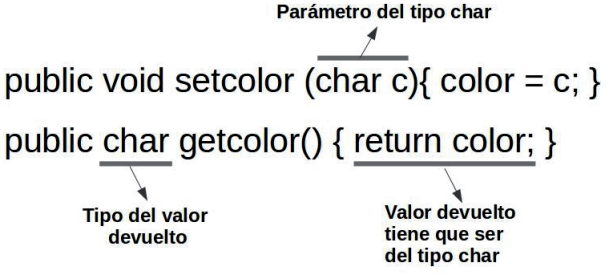

Figura 1.5. *Parámetros y valores devueltos*

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`.

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++, el 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.

RECUERDA

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 (existe más de un constructor en la clase). Eso quiere decir que es posible crear objetos de la clase `pájaro` de distintas formas.

1.2.3 OBJETOS COMO INSTANCIAS DE CLASE. LA INSTANCIA ACTUAL THIS

Como ya se ha dicho, los programas realizados mediante el paradigma de la POO solamente tienen objetos, y todos y cada uno de estos objetos pertenecen a una clase concreta. Además Java mantiene un puntero a la instancia actual, al que llama **`this`**.

Java, al igual que C++, proporciona una referencia al objeto con el que se está trabajando. La referencia `this` no es ni más ni menos que el objeto que está ejecutando el método. En los ejemplos que hemos estado utilizando, en muchas ocasiones se obviaba esta referencia puesto que se sobreentiende que el objeto está invocando al método. En algunas ocasiones nos va a servir para resolver ambigüedades o para devolver referencias al propio objeto. En el siguiente ejemplo se ve claramente el uso del `this`.

OBSERVA

En el siguiente código vas a poder apreciar que la referencia `this` en ocasiones se puede omitir. Observa también cómo se devuelve una referencia al propio objeto en los métodos `incrementarAncho()` e `incrementarAlto()`.

```
class rectangulo
{
    private int ancho = 0;
    private int alto = 0;
    rectangulo(int an, int al){
           ancho = an; //se puede omitir el this
           this.alto = al;
    }
    public int getAncho(){return this.ancho;}
    public int getAlto(){return alto;} //se puede omitir el this
    public rectangulo incrementarAncho(){
           ancho++; //se puede omitir el this
           return this;
    }
    public rectangulo incrementarAlto(){
           this.alto++;
           return this;
    }
}
```

1.2.4 USO DE MÉTODOS ESTÁTICOS Y DINÁMICOS

En este apartado se va a ver cuándo un método o atributo debe 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 cómo 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, los métodos `nuevopajaro()`, `muestrapajaros()` o el `método main()` tiene 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);
           p1.muestrapajaros();
           p2.muestrapajaros();
    }
}
```

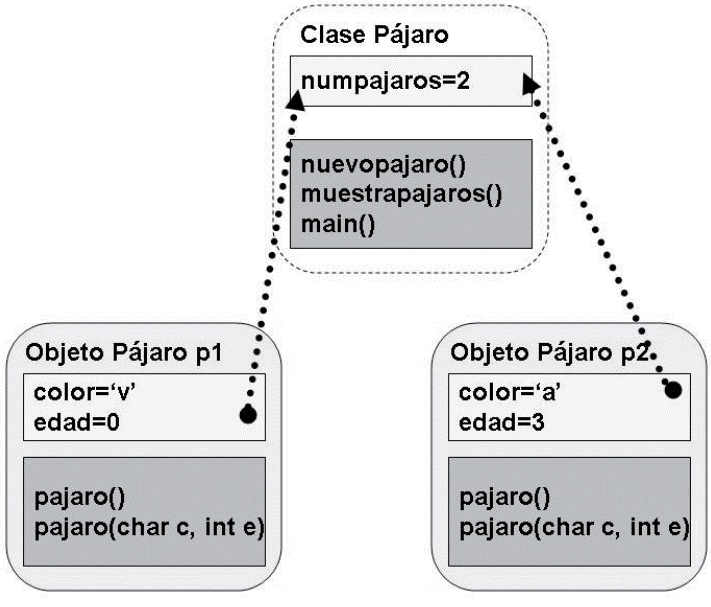

Figura 1.6. *Clase pájaro y objetos de dicha clase*

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`.

1.2.5 LOS PAQUETES

Los conceptos de paquete, librería y `CLASSPATH` se han visto en apartados anteriores. Sobre los paquetes hay que tener en cuenta las siguientes ideas:

- Un paquete es un conjunto de clases relacionadas entre sí.
- Un paquete puede contener a su vez subpaquetes.

- Java mantiene su biblioteca de clases en una estructura jerárquica.
- Cuando nos referimos a una clase de un paquete (salvo que se haya importado el paquete) hay que referirse a la misma especificando el paquete (y subpaquete si es necesario) al que pertenece (por ejemplo: `java.io.File`).
- Los paquetes permiten reducir los conflictos con los nombres puesto que dos clases que se llaman igual, si pertenecen a paquetes distintos, no deberían dar problemas.
- Los paquetes permiten proteger ciertas clases no públicas al acceso desde fuera del mismo.

Para la **creación de un paquete** muy sencillo vamos a seguir los siguientes pasos:

El objetivo es crear un paquete con dos clases y llamar a dichas clases desde un programa aparte.

1 Lo primero que hay que hacer es crear en el directorio donde estamos compilando los programas un subdirectorio con nombre; por ejemplo, `Utilidades`. En este subdirectorio vamos a tener varios paquetes (serán subpaquetes del paquete `utilidades`). El primero de ellos se va a llamar `educación` y vamos a tener dentro de él dos clases ya compiladas llamadas `saludar` y `despedirse` (`saludar.class` y `despedirse.class`).

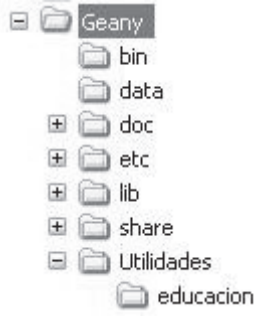

Figura 1.7. *Estructura de directorios de los paquetes que se van a crear*

Cada subpaquete estará situado en un subdirectorio aparte del subdirectorio `Utilidades`. Como se puede observar en la imagen anterior, se está utilizando el compilador Geany. En este directorio (`Geany`) se guardan los programas y las clases, y es aquí donde crearemos estos subdirectorios.

2 Las clases que se crearán son las siguientes:

```
package Utilidades.educacion;
import java.io.*;
public class saludar{
    public void saludo(){
          System.out.println("Hola");
    }
}
//** Fin código****
//** Inicio código****
package Utilidades.educacion;
import java.io.*;
public class despedirse{
```

```
    public void despedida(){
           System.out.println("Adios");
    }
}
```

```
package Utilidades.educacion;
```

Nótese que, con la sentencia anterior, ambas clases indican que pertenecen al paquete `educación`.

3 Una vez que he hecho eso, el siguiente paso será importar el paquete con la sentencia `import`.

```
import Utilidades.educacion.*;
public class test {
public static void main(String[] args) {
                        saludar s=new saludar();
                        despedirse d=new despedirse();
                        s.saludo();
                        d.despedida();
}
}
```

En el programa anterior se crearán dos objetos, uno de cada clase (`saludar` y `despedirse`) y se hace una llamada a un método de cada clase para verificar que el paquete funciona correctamente. Si se han realizado estos pasos correctamente, la compilación no debería dar ningún error.

1.2.6 CONCEPTO DE "PROGRAMA" EN EL PARADIGMA ORIENTADO A OBJETOS

DEFINICIÓN DE PROGRAMA

Un programa es una serie de órdenes o instrucciones ordenadas con una finalidad concreta que realizan una función determinada.

Todos estamos familiarizados con la ejecución de programas (editores de textos, navegadores, juegos, reproductores de música o películas, etc.). Por regla general, cuando queremos ejecutar un programa se lo indicamos al sistema haciendo doble clic sobre él, e incluso algunos usuarios más avanzados ejecutan comandos desde un intérprete de comandos o consola. Si alguna vez has tenido la curiosidad de abrir un programa con un bloc de notas o editor de texto te habrás dado cuenta de que aparece algo horrible en el editor: una serie de símbolos ininteligibles (por los humanos). Eso es porque los programas están en binario, que es el lenguaje que entienden las máquinas. Entonces te preguntarás: si al final de este libro seré capaz de escribir programas, ¿podré entender esos códigos? La respuesta es no. En este libro vamos a aprender un lenguaje de programación para escribir programas de manera entendible por los humanos, que luego traduciremos al lenguaje máquina entendible por los ordenadores mediante otros programas llamados "intérpretes" o "compiladores".

En la siguiente figura se verá todo esto de modo más gráfico:

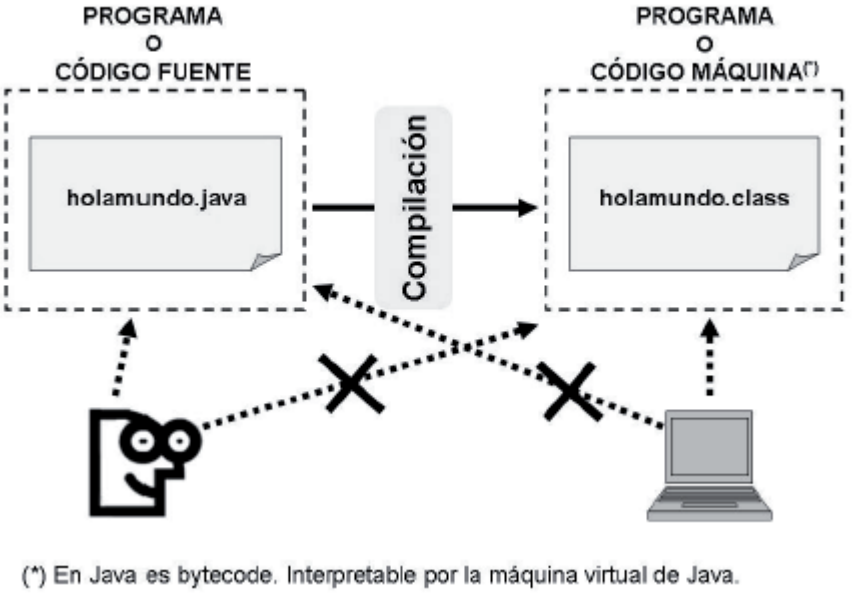

Figura 1.8. *Proceso de compilación en Java*

Como se puede observar, el código fuente es el que escribe el programador, que luego lo compila a código máquina. Compilar equivale a transformar el programa inteligible por el programador al programa inteligible por la máquina. El código fuente o programa fuente está escrito en un lenguaje de programación y el compilador es un programa que se encarga de transformar el código fuente en código máquina.

Los compiladores son programas específicos para un lenguaje de programación, los cuales transforman el programa fuente en un programa directa o indirectamente ejecutable por la máquina destino. No es posible compilar un programa escrito en lenguaje Java con un compilador de C porque este no lo entendería.

El lenguaje máquina que genera Java es un lenguaje intermedio interpretable por una máquina virtual instalada en el ordenador donde se va a ejecutar. Una máquina virtual es una máquina ficticia que traduce las instrucciones máquina ficticias a instrucciones para la máquina real. La ventaja de la misma es que los programas se pueden ejecutar en cualquier tipo de hardware siempre y cuando tenga instalada la máquina virtual correspondiente (la JVM). Los programas no van a cambiar, lo que cambiará es la máquina virtual dependiendo del hardware (no será igual la máquina virtual de un *smartphone* que la de un PC).

LOS COMPILADORES E INTÉRPRETES

A diferencia de los compiladores, los intérpretes leen línea a línea el código fuente y lo ejecutan. Este proceso es muy lento y requiere tener cargado en memoria el intérprete. La ventaja de los intérpretes es que la depuración y corrección de errores del programa es mucho más sencilla que con los compiladores.

Java es uno de los lenguajes más utilizados en la actualidad. Es un lenguaje de propósito general y su éxito radica en que es el lenguaje de Internet. *Applets*, *servlets*, páginas JSP o JavaScript utilizan Java como lenguaje de programación.

El éxito de Java reside en que es un lenguaje multiplataforma. Java utiliza una máquina virtual en el sistema destino y por lo tanto no hace falta recompilar de nuevo las aplicaciones para cada sistema operativo. Java, por lo tanto, es un lenguaje interpretado que para mayor eficiencia utiliza un código intermedio (*bytecode*). Este código intermedio o *bytecode* es independiente de la arquitectura y por lo tanto puede ser ejecutado en cualquier sistema.

CONCEPTO DE PROGRAMA EN LA POO

Un programa en POO se organiza en clases. Dichas clases tienen datos más operaciones que se hacen sobre dichos datos (métodos). Cuando se ejecuta un programa se crearán varios **objetos**, cada uno de los cuales será instancia de una clase, y se pasarán **mensajes** entre ellos, realizando de esta manera las tareas que demanda el usuario.

1.3 TEST DE CONOCIMIENTOS

1 ¿Cuál de las siguientes afirmaciones es falsa?:
- **a)** Con la programación orientada a objetos (POO) se aumenta la modularidad.
- **b)** Con Java no se puede realizar programación clásica (programación estructurada).
- **c)** El constructor del objeto es un procedimiento llamado automáticamente cuando se crea un objeto de esa clase.
- **d)** El `CLASSPATH` sirve para localizar clases creadas por el usuario o terceras personas.

2 ¿Cuál de las siguientes afirmaciones es falsa?:
- **a)** Un paquete es un conjunto de clases relacionadas entre sí.
- **b)** Un método definido como `protected` puede ser accedido desde la misma clase pero no desde subclases en el mismo paquete.
- **c)** En Java no existen variables o funciones globales como en otros lenguajes de programación.
- **d)** El compilador de Java viene dentro de un paquete de desarrollo denominado JDK.

3 ¿Cuál de las siguientes afirmaciones es falsa?:
- **a)** Las clases son los moldes de los cuales se generan los objetos.
- **b)** Las clases se escriben en un lenguaje llamado *bytecode*.
- **c)** En Java NO hay destructores como en C++.
- **d)** El polimorfismo, el encapsulamiento y la herencia son técnicas nuevas de la POO.

4 ¿Cuál de las siguientes afirmaciones es falsa?:

a) El estado de un objeto variará dependiendo de los valores de los atributos del objeto.

b) Una clase definida como `pública` puede ser utilizada por las clases de su paquete y otros paquetes.

c) El compilador de Java viene dentro de un paquete de desarrollo denominado JRE.

d) Un *package* es un conjunto de clases relacionadas entre sí.

5 ¿Cuál de las siguientes afirmaciones es falsa?:

a) Los métodos son los procedimientos que ejecuta el objeto cuando recibe un mensaje.

b) El destructor se ejecutará automáticamente siempre que se destruye un objeto de dicha clase.

c) Un método no definido como `private` no es accesible desde otras clases en el mismo paquete.

d) Java proporciona una referencia al objeto con el que se está trabajando que se llama `this`.

6 ¿Cuál de las siguientes afirmaciones es falsa?:

a) Un paquete puede contener a su vez subpaquetes.

b) Para realizar programas en Java, necesitamos un JDK y un JRE en la máquina de desarrollo.

c) 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.

d) Un compilador puede compilar programas realizados en varios lenguajes de programación.