

Capítulo 1

APECTOS BÁSICOS DE JAVA

Cuando hablamos de Java no debemos pensar únicamente en un lenguaje de programación. La plataforma Java proporciona, además de un lenguaje de programación, todo un conjunto de especificaciones, tecnologías y librerías de clases, mediante los cuales se pueden crear diferentes tipos de programas informáticos capaces de ser ejecutados en una amplia variedad de sistemas operativos.

Esta independencia del sistema operativo ha contribuido a que numerosos fabricantes software hayan apostado por esta tecnología, desarrollando productos y herramientas que facilitan la labor de programador y ayudan a aumentar la potencia y rendimiento de las aplicaciones creadas con Java.

Así pues no es de extrañar la enorme penetración que Java tiene en el mundo del software, siendo la principal opción para numerosas Empresas y programadores independientes a la hora de acometer sus desarrollos.

1.1. LA MÁQUINA VIRTUAL JAVA

“Escribe una vez y ejecuta en cualquier parte”, este es el lema que lanzaron los creadores de Java cuando lanzaron la primera versión de la plataforma y, básicamente, se refiere al hecho de poder crear y compilar un programa en Java en una determinada máquina y poderlo ejecutar después en diferentes sistemas operativos, sin necesidad de sucesivas recompilaciones del código.

Y todo esto es posible gracias a la existencia de un entorno de ejecución, conocido como **Máquina Virtual Java o JVM**, capaz de traducir en tiempo de ejecución el código compilado a código ejecutable nativo para un sistema operativo particular.

Esto supone la necesidad de que en el equipo donde se va a ejecutar el programa exista una versión de JVM para ese sistema operativo particular. Esto actualmente no supone ningún problema, ya que la mayoría de los sistemas operativos modernos disponen de JVM que además se incluye como una propia extensión del sistema.

1.2. EDICIONES JAVA

Java, en lo que a lenguaje de programación se refiere, existe solamente uno, sin embargo, el rápido aumento del número de clases para trabajar con Java que iban apareciendo con cada nueva versión de la plataforma, obligó a Sun organizar ésta en tres niveles o ediciones y distribuir entre ellas las distintas librerías disponibles, enfocándose cada una de estas ediciones en un tipo de desarrollo concreto:

- **Java Standar Edition (Java SE).** Digamos que esta edición proporciona lo básico para trabajar en Java, incluyendo:
 - Herramientas para la compilación y ejecución de programas.
 - Runtime de Java. O lo que es lo mismo, la Máquina Virtual de Java y todas aquellas clases de uso general necesarias para la construcción y ejecución de cualquier tipo de programa Java.
 - Como añadido, Java SE también contiene clases para la construcción de interfaces gráficas de usuario y para acceso a bases de datos.
- **Java Enterprise Edition (Java EE).** Esta edición incluye todas las clases necesarias para la construcción de aplicaciones Internet/intranet con Java.
- **Java Micro Edition (Java ME).** Proporciona todo el soporte necesario para la creación de aplicaciones que puedan ser ejecutadas en dispositivos de pequeño tamaño, tales como teléfonos móviles o agendas electrónicas. En este sentido, Java ME incluye:
 - Una versión reducida de la Máquina Virtual Java para este tipo de dispositivos (KVM).
 - Librerías de clases específicas para este tipo de aplicaciones.

Estas ediciones pueden descargarse libremente desde el sitio Web de Java Sun: <http://java.sun.com>.

1.3. ESTRUCTURA DE UN PROGRAMA JAVA

A la hora de escribir una aplicación en Java, debemos de tener en cuenta que todo programa escrito en este lenguaje se estructura en clases. Aún es pronto para intentar comprender el concepto de clase, de momento

bastará indicar que una clase es un conjunto de métodos (funciones) que se encargan de realizar un tareas que tienen algún tipo de relación entre ellas.

Por ejemplo, una clase llamada `GestionEmpleados` podría estar compuesta de métodos que realizaran operaciones con los empleados de una empresa, como altas y bajas de empleados, búsquedas a partir de un identificador, etc.

Además de métodos, las clases pueden incluir datos (llamados campos o atributos) que sean compartidos por todos los métodos de la clase.

Para definir una clase en Java se utiliza la sintaxis:

```
class NombreClase
{
    //contenido de la clase
}
```

NombreClase sería el nombre que le queremos asignar a nuestra clase. Las reglas que debe cumplir un nombre de clase son muy básicas: no puede coincidir con ninguna palabra reservada del lenguaje y entre los caracteres que lo forman no se pueden incluir espacios ni signos de puntuación. Por otro lado, suele seguirse el convenio de que la primera letra de cada palabra que compone el nombre se escriba en mayúsculas, mientras que el resto lo harán en minúsculas.

1.4. EL MÉTODO MAIN.

Todo programa Java, independientemente del número de clases que lo formen, debe contener en alguna de sus clases un método, conocido como método *main* o principal, que constituirá el punto de inicio de la aplicación. Este método, **será invocado por la Máquina Virtual de Java cuando le demos la orden de ejecutar el programa**; dicho método tendrá que tener un formato específico, formato definido por la especificación de Sun.

El formato del método `main` será el siguiente:

```
public static void main (String [] args)
{
    //contenido del método
}
```

Poco a poco iremos explicando el significado de cada una de las palabras clave que aparecen en la definición del método. De momento quedémonos con el nombre de éste (`main`) y con el tipo de devolución (`void`)

que significa que el método no devolverá ningún resultado al punto de llamada, algo lógico en este caso pues quien hace la llamada al método es la JVM y ésta no debe esperar ningún resultado tras su ejecución.

1.5. PROGRAMA BÁSICO EN JAVA

Llegados a este punto, vamos a crear nuestro primer programa en Java, consistente en una sencilla aplicación que muestre en la consola de comandos (antiguamente conocida como ventana MS-DOS) un mensaje de bienvenida.

Para esta aplicación solo necesitaremos crear una única clase con un único método, el método *main()*, en el que codificaremos la instrucción encargada de mostrar el mensaje. He aquí el código de nuestra clase:

```
public class Ejemplo
{
    public static void main (String[] args)
    {
        System.out.println("Mi primer programa Java");
    }
}
```

Aunque tendremos oportunidad de analizar más adelante los elementos que componen la instrucción anterior, es importante comentar que el envío del mensaje a la consola es realizado a través de un método (`println`) de una de las clases incorporadas en el Java estándar. Esto viene a demostrar una de las características de la programación en Java, y es que además de las instrucciones propias de cualquier lenguaje de programación, **la realización de cualquier programa requiere el uso de diferentes clases incluidas en la plataforma para conseguir la funcionalidad requerida.**

1.6. COMPILACIÓN Y EJECUCIÓN DE PROGRAMAS JAVA

Una vez escrito el código fuente del programa, es necesario realizar dos operaciones para poder ver los resultados:

- **Compilación.** Operación consistente en transformar el código fuente en un código intermedio, conocido como bytecode, que no está ligado a ningún sistema operativo concreto, sino a un entorno de ejecución conocido como Máquina Virtual Java. Esto, como ya hemos comentado antes, permite la portabilidad del código entre distintas plataformas. Los bytecodes son almacenados en archivos `.class`, tal y como se ilustra en la figura 1.

Como explicaremos más adelante, la compilación de un programa Java se realiza clase a clase a través de la utilidad **javac.exe** proporcionada por el software de desarrollo para Java de Sun (SDK), utilidad ésta que debe invocada a través de la consola de comandos.

- **Ejecución.** La ejecución, que no tiene porqué llevarse a cabo en la misma máquina donde se ha realizado la compilación, consiste en traducir los bytecodes a código máquina nativo y realizar la ejecución del mismo. Esta operación es realizada por un intérprete incorporado en la JVM y que es invocado a través del comando **java.exe** proporcionado por el SDK.

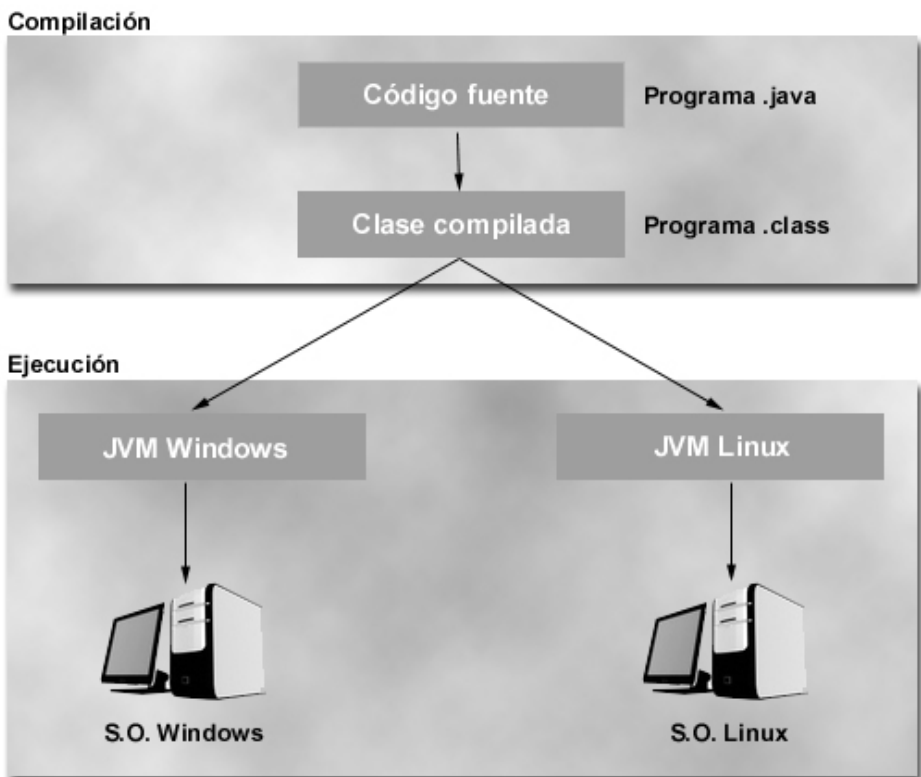


Figura 1

Durante la fase de desarrollo de los programas, los programadores realizan constantemente las tareas de compilación y posterior ejecución del código para probar su funcionamiento. Es por ello que, en vez de utilizar directamente las utilidades *javac* y *java* desde la línea de comandos, se haga uso de algún entorno de desarrollo integrado (IDE) que facilite la realización de estas tareas, abstrayendo al programador de los detalles asociados al empleo directo de estas utilidades.

Existen una amplia variedad de IDEs para Java a disposición de los programadores; unos de libre distribución y otros con licencia. Uno de los más populares, y que será el que utilicemos nosotros en el curso es NetBeans. NetBeans es un potente entorno de desarrollo de libre distribución que podemos encontrar en la página www.netbeans.org. Con este entorno podremos desarrollar cualquier aplicación Java, tanto estándar como enterprise, de una manera sencilla.

1.7. EDITAR, COMPILAR Y EJECUTAR PROGRAMAS

En este capítulo explicaremos como compilar y ejecutar programas en Java, tomando como ejemplo el programa de bienvenida que hemos presentado anteriormente.

Estas operaciones pueden ser llevadas a cabo mediante las herramientas del SDK de Java o utilizando un entorno de desarrollo que las lleve incorporadas. Vamos a estudiar el empleo de ambas opciones.

1.7.1. *Utilización de las herramientas del SDK*

Aunque en la gran mayoría de las ocasiones se utilizarán IDE's para la construcción de programas Java, es importante conocer como se realizan estas operaciones a un más bajo nivel. Por ello, comenzaremos analizando la forma de trabajar de los comandos básicos del SDK.

Estos comandos podemos encontrarlos en la carpeta \bin del directorio de instalación del SDK. Si aun no lo tenemos instalado podemos descargarlo de forma gratuita de la página Web de Java: <http://java.sun.com>. Aquí, dentro de la zona de descargas para la edición estándar, elegiremos la última de las versiones existentes de esta plataforma, actualmente la versión JDK 1.6 (comercialmente llamada Java 6).

Tras la instalación del SDK se crearán una serie de directorios en la ubicación que le hemos indicado, entre ellos el directorio \bin donde están los comandos **javac.exe** y **java.exe** (ver figura 2).

Antes de poder utilizar estos comandos, será necesario configurar en nuestra máquina ciertas variables del sistema.

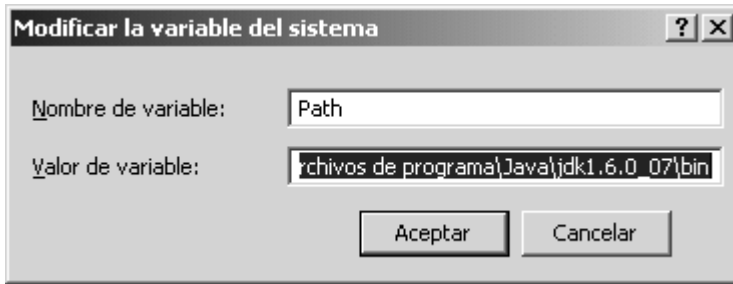


Figura 3

El motivo por el que se incluye esta dirección en la variable PATH es para permitir que puedan invocarse los comandos de compilación y ejecución de programas **desde cualquier directorio del disco**.

La variable de entorno CLASSPATH

De cara a que se puedan localizar las clases utilizadas por un programa, tanto a la hora de compilarlo como a la hora de ejecutarlo, la variable de entorno CLASSPATH debe contener las direcciones de las carpetas donde se encuentran los archivos .class utilizados por nuestro programa, incluido el propio .class del programa que se va a ejecutar.

Normalmente, las direcciones de las clases del JDK no es necesario incluirlas en el CLASSPATH ya que implícitamente el compilador sabe dónde encontrarlas. Como normalmente se suele ejecutar un programa desde el mismo directorio en el que se encuentra, se suele incluir el valor "." en la variable CLASSPATH para hacer referencia al directorio actual.

1.7.1.2. Compilación de una clase Java

Una vez configuradas las variables de entorno, procedemos a codificar el programa, para lo cual utilizaremos un editor de texto cualquiera. El resultado final lo almacenaremos como un archivo de texto con extensión .java y nombre igual al de la clase (Ejemplo.java). Es importante destacar en este punto que **el nombre de un archivo de código fuente Java deberá tener el mismo nombre que la clase definida como pública (public)**.

Seguidamente, podemos proceder a su compilación, la cual generará como resultado un archivo .class por cada clase incluida en el código fuente. En este caso, se generará un único archivo llamado Ejemplo.class.

La compilación se llevará a cabo mediante el comando javac.exe del JDK. Para ello, nos situaremos con la consola de comandos en el directorio

donde se encuentra el archivo .java, en nuestro caso donde hayamos colocado el programa Ejemplo.java, y teclearemos:

```
javac Ejemplo.java
```

Si el código no presenta ningún error de sintaxis la compilación se hará correctamente y no se generará ningún mensaje de aviso, volviéndose a mostrar en pantalla la línea de comandos.

Empaquetado de una clase Java

Normalmente, las clases Java se estructuran en paquetes. Aunque en temas posteriores volveremos a hablar de los paquetes, es necesario saber que el empaquetado de las clases facilita la organización de las mismas, agrupando en un mismo paquete clases que realicen alguna función similar o estén relacionadas de alguna manera.

Hay que pensar que los paquetes son para las clases lo que las carpetas son para los archivos de un disco. De hecho, físicamente, los paquetes no dejan de ser eso, carpetas o directorios donde se almacenan los archivos .class.

De cara a conseguir distintos niveles organizativos, un paquete puede dividirse a su vez en subpaquetes y cada uno de estos contener nuevos subpaquetes, así hasta conseguir los niveles de anidación que consideremos necesarios.

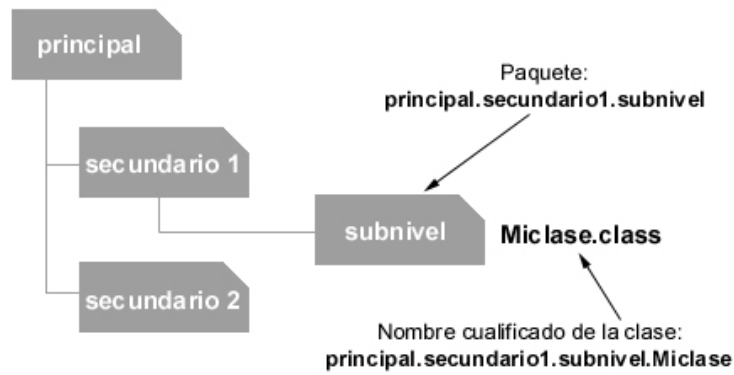


Figura 4

En la figura 4 vemos un ejemplo de organización de paquetes, donde una determinada clase llamada `Miclase.class` se encuentra dentro de un paquete en un tercer nivel de anidación.

Cuando las clase se encuentran dentro de paquetes, el nombre identificativo de la clase, denominado **nombre cualificado**, es el formado por el nombre de la misma precedido por los distintos paquetes en que se encuentra. Ya veremos más adelante las implicaciones que tiene en un programa de cara a la utilización de las mismas el tener clases incluidas dentro de paquetes.

A la hora de definir la clase, si queremos que esta forme parte de un determinado paquete será necesario indicarlo mediante la sentencia **package**, sentencia que debe ser la primera en aparecer dentro del archivo .java. En el caso de la clase de ejemplo que aparece en la figura 4, la definición de la misma sería:

```
package principal.secundariol.subnivel;
public class MiClase{
:
}
```

Para compilar una clase que está definida dentro de un paquete, se debería utilizar el siguiente formato del comando javac desde el directorio en el que se encuentra el archivo .java (directorio actual):

```
javac -d . NombreClase.java
```

En el ejemplo anterior:

```
javac -d . MiClase.java
```

Donde el atributo “d” del comando indica que la clase será incluida en el directorio indicado en la sentencia package, mientras que el punto “.” indicaría que la ruta de ese directorio comienza desde el directorio actual. **Si dicho directorio no existiera en el momento de la compilación, se crearía al ejecutar el comando anterior.** En cualquier caso, la situación final sería la indicada en la figura 4.

1.7.1.3. Ejecución de una clase Java

Como resultado de la compilación, se habrá generado en el mismo directorio en el que se encuentra el archivo de código fuente .java un archivo bytecode llamado Ejemplo.class. Este archivo, tal y como hemos comentado al principio, puede ser transportado a otro equipo para proceder a su ejecución. Dicha ejecución es llevada a cabo también desde la consola de comandos a través del programa java.exe, seguido del nombre de la clase que contiene al método *main()*. Si seguimos en el directorio donde se encuentra el archivo Ejemplo.class teclearemos lo siguiente para realizar su ejecución: