19. *Packages* o paquetes

Objetivos:
a) Definir el concepto de paquete
b) Interpretar el código fuente de una aplicación Java donde se utilicen paquetes
c) Construir una aplicación Java sencilla, convenientemente especificada, que incluya la declaración y utilización de un paquete

Aunque la mayoría de los programas que se han realizado hasta ahora han constado de una única clase o de unas pocas almacenadas en uno o pocos archivos, habitualmente una aplicación Java se compone de una colección de clases más o menos numerosa. Cuando los programas son relativamente grandes o se trabaja en equipo, es recomendable dividir el código en varios archivos fuente. Las razones son las siguientes:

a) el **tiempo** necesario para la compilación de todo el código cada vez que se haga una modificación mínima en una clase: si cada clase se almacena en un archivo fuente sólo será necesario la compilación de aquellos archivos que se vayan modificando y necesiten ser recompilados.
b) favorece la **eficiencia** del trabajo en equipo ya que la edición simultánea de un único archivo fuente por varios programadores es complicada: es preferible dividir el código fuente en archivos y que cada programador se responsabilice de uno o varios de ellos.
c) incrementa la **facilidad** para localizar y controlar los accesos a las clases e interfaces y evitar conflictos con identificadores.

19.1. ¿Cómo se organiza un conjunto de archivos fuente de Java?

Un paquete o *package* de Java es un conjunto de clases e interfaces relacionadas entre sí.

Por un lado, las clases e interfaces que forman parte de la plataforma Java se estructuran en varios paquetes organizados por funciones o tareas. Por ejemplo, las clases fundamentales están en `java.lang`, las clases para operaciones de entrada y salida de datos están en `java.io`, etcétera. Aprender a utilizar y familiarizarse con las clases y métodos implementados en estos paquetes representa la mayor parte del aprendizaje del lenguaje de programación Java.

Tabla 19.1. Algunos de los paquetes más importantes del API (*Application Programming Interface*) de Java

Package	Descripción del contenido
`java.lang`	Contiene las clases e interfaces más empleadas en la mayoría de los programas de Java. Es importado automáticamente por todos los programa Java: no se necesita sentencia `import` para utilizar lo declarado en este paquete.
`java.io`	Contiene clases que permiten las operaciones de entrada y salida de datos de un programa.
`java.util`	Contiene clases e interfaces de utilidades: operaciones con la fecha y la hora, generación de números aleatorios...
`java.applet`	Contiene todas las clases e interfaces necesarias para la construcción de *applets* de Java
`java.net`	Contiene clases que permite a un programa comunicarse a traves de redes (Internet o intranet)
`java.text`	Contiene clases e interfaces que permiten operaciones de números, fechas, caracteres y cadenas.

`java.awt`	Es el paquete *Abstract Windowing Toolkit*. Contiene muchas clases e interfaces necesarias para trabajar con la interfaz de usuario gráfica *clásica*.
`java.beans`	Contiene clases para facilitar a los programadores la generación de componentes de software reutilizables.

Por otro lado, cualquier programador puede introducir sus clases e interfaces en paquetes para facilitar tanto su uso en el desarrollo de un programa como su reutilización en varios de ellos. Para incluir una clase en un paquete debe incluirse la siguiente sentencia al principio del archivo fuente que contiene la clase:

```
package identificadordelPaquete;
```

Por ejemplo, para incluir la clase `Rectangulo` dentro del paquete `geometria`:

```
package geometria;
public class Rectangulo extends FiguraGeometrica {
    // Declaracion de atributos y metodos
    . . .
}
```

El alcance de la sentencia *package* es todo el archivo fuente. Si se incluyen varias clases dentro de un único archivo fuente, sólo puede declararse como pública una clase y debe tener el mismo nombre que el archivo fuente. Sólo las componentes públicas de esa clase son accesibles desde fuera del paquete.

Si no se emplea una sentencia `package`, las clases o interfaces pertenecen al paquete por defecto (sin identificador). Normalmente, este paquete por defecto sólo se emplea en aplicaciones pequeñas o temporales o al inicio de un desarrollo. En cualquier otro caso, las clases e interfaces deben pertenecer a paquetes con un identificador explícito.

Las ventajas de agrupar clases e interfaces en un paquete son las siguientes:

a) Se puede determinar más fácilmente qué clases e interfaces están relacionadas
b) Se evitan conflictos por el empleo de los mismos identificadores en diferentes clases si éstas están en paquetes distintos.
c) Puede configurarse un acceso específico dentro de las clases de un mismo paquete diferente al acceso desde otras clases fuera del paquete.

19.2. Identificación de un paquete

Es relativamente fácil que dos programadores de Java escojan el mismo identificador para dos clases diferentes. El compilador permite que haya dos clases con el mismo identificador si pertenecen a paquetes diferentes: el identificador de la clase debe ir precedida del identificador del paquete (a este identificador compuesto se le denomina identificador *cualificado*). En el ejemplo anterior: `geometria.Rectangulo`. Eso sí, existe una convención para evitar que dos programadores diferentes empleen el mismo identificador cualificado para sus paquetes:

Convención: las empresas, compañías u organizaciones deben usar el nombre de dominio de Internet inverso para identificar los nombres de los paquetes que han desarrollado. Por ejemplo: *`com.nombreEmpresa.nombrePaquete`*.

19.3. Acceso a los componentes de un paquete

Como se ha comentado anteriormente, sólo los componentes públicos de un paquete son accesibles desde fuera del paquete desde el que se han definido. Para emplear un componente público de un paquete debe hacerse una de las siguientes cosas, cada una de las cuales es adecuada a una situación distinta:

a) Referenciarlo mediante su identificador **cualificado**. Por ejemplo:

```
geometria.Rectangulo r1 = new geometria.Rectangulo();
```

Esta opción puede ser adecuada si el identificador sólo se emplea una o muy pocas veces.

b) Importar el componente del paquete, incluyendo una sentencia `import` al principio del archivo fuente, antes de cualquier declaración de clase o interfaz y después de la sentencia `package`, si existe.

```
import geometria.Rectangulo;
...
// Puede hacerse referencia directamente a la clase rectangulo
geometria.Rectangulo r1 = new geometria.Rectangulo();
```

Esta opción es adecuada si sólo se emplean uno o pocos componentes del paquete `geometria`.

c) Importar el paquete completo, empleando la sentencia `import` con el identificador del paquete seguido de un punto y un asterisco:

```
import geometria.*;
...
// Puede hacerse referencia directamente a cualquier
// componente del paquete geometria
geometria.Rectangulo r1 = new geometria.Rectangulo();
```

Por defecto, la plataforma Java importa automáticamente tres paquetes completos:

- El paquete por defecto (sin identificador explícito).
- El paquete `java.lang` que contienen las clases de Java más corrientes, como `Object` y `Math`.
- El paquete actual de trabajo.

19.4. Modificadores de acceso a los componentes

Los programadores incluyen sus clases e interfaces dentro de paquetes por dos razones. La primera es por motivos organizativos y la segunda es por la configuración de acceso a los componentes.

Existen cuatro tipos de modificadores de acceso en Java: `public`, `private`, `protected` y `package`. Los tres primeros se han de definir explícitamente mediante la palabra reservada en la declaración del atributo o del método correspondiente, mientras que la cuarta es el acceso por defecto si no se indica otro explícitamente.

En la Tabla 19.2 se resume el tipo de acceso correspondiente a cada uno de estos modificadores.

Tabla 19.2. Modificadores de acceso en Java

Modificador	Tipo de acceso
`public`	Acceso desde cualquier método de cualquier clase
`package`	Acceso desde cualquier método de una clase perteneciente al mismo paquete (mismo directorio)
`protected`	Acceso desde cualquier método de una clase perteneciente al mismo paquete (mismo directorio) y desde las clases descendientes (subclases)
`private`	Acceso exclusivo desde los métodos de la clase correspondiente

19.5. Recomendaciones para la gestión de archivos de Java

Aunque la especificación del lenguaje Java no obliga a ello, se suele emplear el siguiente procedimiento para construir un sistema jerárquico que facilite la gestión y uso de los archivos fuente y compilados (*bytecodes*):

a) Almacenar el código fuente de una clase o interfaz en un archivo de texto con el mismo identificador que la clase o interfaz y extensión `.java`. Por ejemplo: `Rectangulo.java`.

b) Poner el archivo en un directorio con el mismo nombre que el paquete al que pertenezca la clase o la interfaz. Por ejemplo: `geometria/Rectangulo.java`. El identificador *cualificado* del componente del paquete (`geometria.Rectangulo`) y la vía de acceso al archivo son ahora paralelos o equivalentes.

c) Como se ha comentado anteriormente, por convención, una organización emplea su nombre de dominio a la inversa para identificar los *packages* que desarrolla. Por ejemplo, la empresa ficticia Novedades García y Arranz con nombre de dominio `garciayarranz.com` añadiría `com.garciayarranz` al principio de los identificadores de sus *packages*. Así, si la empresa García y Arranz tuviera un paquete `geometria` que contuviera un archivo fuente `Rectangulo.java`, debería incluirse en una secuencia de directorios de la siguiente forma:

```
package com.garciayarranz.geometria;

public class Rectangulo {
    . . .
}
```

d) Al compilar un archivo fuente, el compilador genera un archivo de *bytecodes* distinto para cada clase e interfaz declarada en el archivo fuente. El nombre del archivo es el identificador de la clase o interfaz y la extensión es .class. Los archivos con extensión suelen organizarse en una estructura específica de directorios que refleja el nombre del paquete y diferente de la estructura utilizada para organizar los archivos fuente. Por ejemplo:

Archivo de *bytescodes*: `clases/com/garciayarranz/geometria/Rectangulo.class`
Archivo fuente: `fuentes/com/garciayarranz/geometria/Rectangulo.java`

e) El directorio raíz de la estructura de directorios debe incluirse la vía del acceso a las clases del ordenador. Esta vía de acceso se define mediante la variable de entorno CLASSPATH.

Cuando el compilador o el intérprete Java necesitan localizar una clase en un paquete, buscan en cada uno de los directorios asignados a CLASSPATH para ver si el árbol de directorios que contiene el paquete se encuentra ahí.

En un sistema Windows 95/98 la vía de acceso se define añadiendo la siguiente linea al archivo autoexec.bat: `set CLASSPATH = c:\viadeacceso\clases`. Si la variable de entorno CLASSPATH ya existe, se añade un punto y coma y `c:\viadeacceso\clases` a la vía de acesso ya existente. Es necesario rearrancar el ordenador para que el cambio haga efecto.

En un sistema Windows NT, XP o posteriores, la vía de acceso se establece a través del menú Inicio/Configuración/Panel de Control/Sistema/Entorno.

En un sistema UNIX, la vía de acceso se establece de diferentes formas dependiendo de la plataforma específica que se utilice.

f) Esta estructura facilita que tanto el compilador como el intérprete de Java busquen y encuentren las clases en el directorio apropiado. Si se desea los archivos pueden encontrarse dentro de archivos comprimidos en formato ZIP.