Apuntes

de

Java

Tema 3:
Paquetes

TEMA 3: PAQUETES

3.1.- ¿PARA QUÉ SIRVEN LOS PAQUETES?

Para sistemas medianos o grandes, el número de clases que intervengan en un programa se hará muy grande (varios cientos). Además, a medida que se incorporan clases de terceros aumentará la posibilidad de encontrarse clases con el mismo nombre (posibilidad que aumenta cuando las clases pueden viajar por la red como es el caso de Java).

Para solucionar estos problemas, Java proporciona lo que se conocen como paquetes. Un paquete es un mecanismo de agrupación y organización de clases con tres propósitos:

- Mejorar la organización de las clases
- Reducir los problemas de colisión de nombres entre clases.
- Controlar la visibilidad de las clases, atributos y métodos definidos en él.

El propio núcleo de Java con todas sus librerías de clases hace un uso intensivo de los paquetes, clasificando todas sus clases en paquetes. Estos son algunos de los paquetes que proporciona Java:

- package java.applet
- package java.awt
- package java.beans
- package java.io
- package java.lang
- package java.sql
- package java.util

3.2.- CARACTERÍSTICAS DE LOS PAQUETES

Los paquetes son grupos relacionados de clases e interfaces que proporcionan un mecanismo conveniente para manejar un gran juego de clases e interfaces y evitar los conflictos de nombres. Además de los paquetes que ya proporciona el API de Java puedes crear tus propios paquetes y poner en ellos definiciones de clases y de interfaces utilizando la sentencia package.

Supongamos que se está implementando un grupo de clases que representan una colección de objetos gráficos como círculos, rectángulos, líneas y puntos. Si quieres que estas clases estén disponibles para otros programadores, puedes empaquetarlas en un paquete, digamos, graphics y entregar el paquete a los programadores (junto con alguna documentación de referencia como qué hacen las clases y qué métodos son públicos).

De esta forma, otros programadores pueden determinar fácilmente para qué es tu grupo de clases, cómo utilizarlos, y cómo relacionarlos unos con otros y con otras clases y paquetes. Los nombres de clases no tienen conflictos con los nombres de las clases de otros paquetes porque las clases dentro de un paquete son referenciados en términos de su paquete (técnicamente un paquete crea un nuevo espacio de nombres). Así pues, podremos tener dos clases que se llamen de la misma forma siempre que éstas estén en distinto paquete. Por el contrario, no podrá haber dos clases con el mismo nombre dentro de un mismo paquete.

Los paquetes pueden anidarse formando jerarquías. Esta posibilidad de anidamiento puede ayudar aun más en la agrupación y organización de clases, al igual que la jerarquización de directorios puede ayudarnos a organizar nuestros ficheros. Para referirse a una clase en un paquete anidado, pueden utilizarse como prefijos del nombre de la clase, los nombres de los paquetes a los que pertenece (separados mediante puntos).

Por ejemplo, para referirse a la clase Point del paquete awt dentro del paquete java, el nombre completo de la clase sería: java.awt.Point.

Toda clase debe pertenecer a un paquete. En caso de que no especifiquemos ningún paquete, la clase se convertirá en miembro del paquete por defecto de Java, que no tiene nombre.

3.2.1.- Cómo crear un paquete

Para indicar que una clase pertenece a un paquete hay que usar la cláusula package:

```
package graphics;

class Circle {
    . . .
}

class Rectangle {
    . . .
}
```

La primera línea del código anterior crea un paquete llamado graphics. Todas las clases definidas en el fichero que contiene esta sentencia son miembros del paquete. Por lo tanto, Circle y Rectangle son miembros del paquete graphics.

El nombre completo de una clase incluye como prefijo el paquete al que pertenece. Por tanto, en el ejemplo anterior el nombre completo de las clases no será Circle y Rectangle, sino graphics.Circle y graphics.Rectangle

Así, para ejecutarlas habrá que hacer:

```
c:\>java graphics.Circle
```

Y para hacer referencia a ellas desde otras clases:

```
graphics.Rectangle
```

Aunque en el ejemplo anterior se han declarado dos clases dentro de un mismo fichero y se las ha incluido a ambas en el mismo paquete. Exactamente igual se puede declarar cada clase en un fichero diferente (de hecho es lo más habitual) e incluirlas a ambas en el mismo paquete. A continuación se muestra un ejemplo:

- Este sería el código del primer fichero (Circle.java):

```
package graphics;

class Circle {
    . . .
}
```

- Y este sería el código del segundo fichero (Rectangle.java):

```
package graphics;

class Rectangle {
    . . .
}
```

Existe una correspondencia directa entre el nombre de los paquetes y los directorios en que éstos se encuentran.

Todas las clases de un paquete deberán estar en un directorio con el mismo nombre. Si el nombre del paquete tiene varias palabras (paquetes anidados) cada una de ellas será un subdirectorio.

Por ejemplo, las clases (ficheros .class) del paquete java.awt.image deberán almacenarse en una estructura de directorios como la siguiente: java\awt\image.

Además, esta estructura de directorios podrá ubicarse en cualquier lugar dentro del sistema de archivos. Lo único que habrá que hacer cuando un programa quiera hacer uso de las clases de este paquete es importarlo (la siguiente sección mostrará cómo) y preocuparse de que la variable CLASSPATH se encuentre apuntando al directorio donde se ha colocado la estructura de directorios java\awt\image.

Si volvemos al ejemplo anterior, los ficheros .class generados por el compilador cuando se compilen los ficheros que contienen el fuente para Circle y Rectangle deben situarse en un directorio llamado graphics en algún lugar referenciado por la variable de entorno CLASSPATH. Más adelante se explicará el funcionamiento de la variable CLASSPATH.

3.2.2.- Como importar un paquete

Cuando queramos acceder a clases que no pertenecen a nuestro mismo paquete, deberemos referirnos a ellas mediante su nombre completo. Así, si quisiésemos acceder a la clase Rectangle y Circle del ejemplo anterior o a la clase Vector del paquete java.util deberíamos hacerlo de la siguiente forma:

```
package otroPaquete;

class MiClase{
      ...
      graphics.Circle c = new graphics.Circle();
      graphics.Rectangle r = new graphics.Rectangle();
      java.util.Vector v = new java.util.Vector();
      ...

}
```

El tener que indicar constantemente el paquete al que pertenece una determinada clase puede llegar a resultar incómodo. Para evitarlo, puede usarse la palabra reservada IMPORT.

La palabra import permite indicar el paquete al que pertenecen una o más clases, de tal manera que no haya que repetirlo con cada uso de las mismas.

```
package otroPaquete;

import graphics.Circle;
import graphics.Rectangle;
import java.util.Vector;

class MiClase{
      ...
      Circle c = new Circle();
      Rectangle r = new Rectangle();
      Vector v = new Vector();
      ...

}
```

La sentencia import debe estar al principio del fichero, antes de cualquier definición de clase y después de la sentencia package (si es que la hay).

No es recomendable, pero se puede realizar la importación de varias clases con el carácter comodín *. Esto permite importar todas las clases de un paquete (no los subpaquetes). Por ejemplo:

```
package otroPaquete;

import graphics.*;
import java.util.*;
import java.awt.*;

class MiClase{
      ...
      Circle c = new Circle();
      Rectangle r = new Rectangle();
      Vector v = new Vector();
      Windows w = new Windows();
      ...

}
```

Si se intenta utilizar una clase de un paquete que no ha sido importado, el compilador mostrará un error parecido a este:

```
testing.java:4: Class Date not found in type declaration.
        Date date;
        ^
```

Sin embargo, existen ciertas excepciones como es el caso del paquete java.lang que es importado automáticamente por el JDK y que por tanto no es necesario importarlo explícitamente.

Todas las clases de un paquete son accesibles a todas las demás clases del mismo paquete. Se puede acceder a todos los atributos y métodos de una clase desde cualquier otra clase dentro del mismo paquete, excepto cuando los atributos o métodos se declaran como privados. La visibilidad de las clases se explicará con más detalle en el siguiente apartado.

3.2.3.- Visibilidad de las clases

Una clase puede ser declarada como:

- De acceso **package** (por defecto): será accesible solamente por las clases de su mismo paquete.
- De acceso **public**: será accesible por cualquier otra clase siempre que ésta, si no está en su mismo paquete, la importe.

Ejemplo:

```
package myPackage;

public class myClass{
     ...
}
```

En cada fichero de código fuente sólo podrá haber una clase con acceso public.

Algunas clases de los paquetes ofrecerán sus servicios al exterior y por tanto, deberán declararse con acceso público. Otras clase, en cambio, están pensadas únicamente para proporcionar apoyo a las clases del propio paquete y deberán ser ocultadas al exterior declarándose con acceso package.

Ejemplo:

```
package contenedores;

public class Lista{
     private Nodo raiz;
     public void add(Object dato){ ... }
     ...
}

class Nodo{
     Object dato;
```

```
        Nodo siguiente;
    }
```

3.2.4.- Colisión de nombres

¿Qué ocurre si se repite el nombre de una clase en dos paquetes importados?

```
import programacion.labinf2.*; // Hay definida una clase Vector
import java.util.*;            // Hay definida una clase Vector
...
Vector v = new Vector();       // ¿Qué Vector usar?
```

Ha ocurrido lo que se conoce como una colisión de nombre. Hemos importado dos paquetes que contienen una clase que se llama igual. En este caso, a pesar del import habría utilizar el nombre completo de la clase para deshacer la ambigüedad.

```
import programacion.labinf2.*; // Hay definida una clase Vector
import java.util.*;            // Hay definida una clase Vector
...
programación.labinf2.Vector v = new programación.labinf2.Vector();
java.util.Vector v = new java.util.Vector();
```

De esta forma queda identificado de manera univoca el vector que queremos usar en cada caso.

Otra opción posible sería especificar en un import el paquete del que se desea coger la clase Vector:

```
import programacion.labinf2.*; // Hay definida una clase Vector
import java.util.*;            // Hay definida una clase Vector
import java.util.Vector;
...
programación.labinf2.Vector v = new programación.labinf2.Vector();
java.util.Vector v = new java.util.Vector();
Vector v = new Vector();       // Se usa el vector java.util.Vector
```

El import de una clase tiene prioridad sobre el import de todo un paquete.

3.3.- LA VARIABLE CLASSPATH

Una aplicación se compone de un conjunto de clases que interactúan entre sí. Sin embargo, el código (ya compilado) de todas estas clases no tiene porque estar ubicado en un mismo sitio. De hecho, lo más habitual es que las clases pertenezcan a paquetes diferentes y por tanto, sus ficheros .class se encuentren ubicados en directorios diferentes. Ante esta situación, y siempre considerando que tratamos con aplicaciones locales, cabe hacerse la siguiente pregunta: ¿cómo se localiza el código de las clases que intervienen en una aplicación?

En primer lugar, la JVM (Java Virtual Machine) añade al nombre de la clase que nosotros usemos en nuestro programa la extensión .class para formar, así, el nombre del fichero a localizar.

Una vez que se sabe el nombre del fichero a buscar, como los ficheros que contienen las clases pueden estar en cualquier lugar, es necesario de algún mecanismo que le indique al interprete Java los lugares donde puede encontrarlos. Este es el cometido de la variable de entorno CLASSPATH.

La variable CLASSPATH contiene una lista de localizaciones donde la JVM debe buscar las clases Java compiladas. Tanto el interprete de Java como el compilador de Java harán uso de esta variable para la búsqueda de ficheros de clases en la máquina local.

El formato para configurar la variable CLASSPATH en Windows y MS-DOS es el siguiente:

```
set classpath=path1;path2;...
```

Donde los "path" (path1, path2,...) son las localizaciones donde se encuentran los paquetes que contienen las clases a buscar. Como se puede observar, para configurar el CLASSPATH es necesario separar cada uno de los path mediante punto y coma

Una localización a indicar en el CLASSPATH puede ser:

- Un directorio.
- Un fichero ZIP (JAR).

En caso de que lo que se indiquen sean directorios, el directorio a indicar debe ser el directorio donde esté contenido el paquete que contiene la clase a buscar. O lo que es lo mismo, se debe indicar el directorio anterior a donde comienza la estructura de directorios correspondiente al paquete.

En caso de que lo que se indique sea un fichero JAR (formato usado por Java para agrupar y comprimir un conjunto de clases), lo único que habrá que indicar es la ruta donde se encuentra el fichero JAR, éste inclusive. Un fichero JAR es un fichero que contiene comprimidas un conjunto de clases (un conjunto de ficheros .class).

Ejemplo:

```
set classpath=c:\programacion;c:\java\lib\classes.jar
```

El orden en que se especifican los path en el CLASSPATH determina el orden en que el interprete de Java va a realizar la búsqueda de las clases. Así, el interprete de Java intentará buscar primero las clases necesarias en la ubicación indicada en el primer path, si no las encuentra allí pasará a buscarlas en el segundo path, y así sucesivamente.

En el ejemplo anterior, si lo que hubiese que buscar es la clase Mesa y ésta se encontrase dentro del paquete muebles (por tanto, sería la clase muebles.Mesa), la JVM la buscaría en el siguiente orden:

- Primero en el directorio c:\programación\ buscaría la estructura de directorios muebles\Mesa.class
- Si no lo encuentra, buscaría dentro del fichero c:\java\lib\classes.jar la estructura de directorios muebles\Mesa.class

3.3.1.- Mecanismo de funcionamiento del CLASSPATH

El mecanismo de la JVM del JDK para localizar la implementación de una clase en tiempo de ejecución es:

- Añade ".class" al nombre completo de la clase.
- Transforma los puntos en separadores de directorio "\".
- Añade la cadena resultante a cada una de las localizaciones del CLASSPATH.
- Si no se encuentra en ninguna de ellas se produce un error de ejecución.

Ejemplo:

Supongamos que la JVM necesita cargar la clase ESIDE.ejemplos.ejemplo1 y que el CLASSPATH está definido como:

```
c:\>set CLASSPATH=c:\programas;c:\temp
```

La JVM sabe que, según el nombre completo de la clase, debe encontrarla en un fichero que se encuentre en la dirección relativa ESIDE\ejemplos\ejemplo1.class

El problema es que no conoce exactamente a partir de qué directorio comenzar a buscar esa estructura de directorios. Lo que hace la JVM es buscar el fichero a partir de las localizaciones que aparecen en el CLASSPATH. Por tanto, lo buscará en:

- c:\programas\ESIDE\ejemplos\ejemplo1.class
- c:\temp\ESIDE\ejemplos\ejemplo1.class

Si la clase no se encuentra en ninguno de estos dos sitios se producirá un error.

3.4.- EJEMPLOS DE CONFIGURACIÓN DEL CLASSPATH

Ejemplo 1:

Supongamos que tenemos una clase Alumno que se encuentra dentro del paquete universidad.eside. El nombre completo de la clase será, por tanto, universidad.eside.Alumno.

Este sería su código:

```
package universidad.eside;

public class Alumno{
     ...
}
```

Según lo explicado anteriormente, el fichero que contiene el código compilado de esta clase se llamará Alumno.class y deberá estar colocado dentro de la estructura de directorios universidad\eside.

Supongamos ahora que estos directorios se encuentran a su vez dentro del directorio c:\programacion. La estructura completa de directorios sería la siguiente:

```
c:\programacion\universidad\eside\Alumno.class
```

Según lo anterior, ¿como habría que configurar el CLASSPATH si quisiésemos usar en nuestro programa la clase Alumno?.

Solución:

El CLASSPATH siempre habrá que configurarlo para que apunte al directorio anterior donde comienza la estructura de directorios correspondiente al paquete. En este caso la estructura de directorios correspondiente al paquete es universidad\eside y por tanto, habrá que colocar el CLASSPATH para que apunte al directorio anterior que es c:\programación. Por tanto, el CLASSPATH se configuraría de la siguiente manera:

```
set classpath=c:\programacion
```

Ejemplo 2:

Supongamos ahora que además de la clase Alumno tenemos otra clase Profesor que no se encuentra ubicada dentro de ningún paquete (en realidad, y como ya sabemos, se encontrará ubicada dentro del paquete por defecto de Java). Este podría ser su código:

```
public class Profesor{
     ...
}
```

Supongamos que el fichero resultado de compilar la clase Profesor (Profesor.class) lo colocamos en el directorio c:\programación\labinf2

Y supongamos ahora que queremos construir un programa que use estas dos clases. Este podría ser el código del programa:

```
package cualquierPaquete;

import universidad.eside.Alumno;    // También válido universidad.eside.*;
import Profesor;

import java.lang.String;  // No es necesario, el paquete java.lang.* siempre
                          // está accesible accesible

public class CualquierClase{

     public static void main(String[] args){
          // Usamos las clases Profesor y Alumno
     }

}
```

Según los supuestos anteriores, ¿Cómo deberíamos configurar el CLASSPATH?

Solución:

La regla a seguir es siempre la misma: El CLASSPATH siempre habrá que configurarlo para que apunte al directorio anterior donde comienza la estructura de directorios correspondiente al paquete. En este caso la clase Profesor no está dentro de ningún paquete luego el directorio anterior donde comienza la estructura de directorios del paquete será el propio directorio donde se encuentra la clase Profesor (c:\programación\labinf2). Aplicando la misma regla para la clase Alumno, el CLASSPATH habría que configurarlo de la siguiente manera:

```
set classpath=c:\programacion;c:\programacion\labinf2
```

3.5.- EJERCICIO

Diseño de una jerarquía de clases para las figuras geométricas que se ubique dentro del paquete figura.

- Comienza creando una clase Figura, que debe tener como atributos área y perímetro. También un método area() y otro perimetro() que devuelven, como sus nombres indican, el área y el perímetro. Todo esto en el fichero Figura.java.

- Crea después tres clases hijas Rectangulo, Cuadrado, y Circulo, en tres ficheros independientes, formando parte del paquete figura. Estas clases tienen que definir sus constructores para que, recibiendo la longitud de dos lados, la de un lado, y la del radio (respectivamente), se calcule de forma automática el área y el perímetro.

- Cada clase tiene que tener un atributo de clase areaTotal que almacene el total de área de todas las figuras creadas de esa clase, de modo que el atributo de clase areaTotal de la clase Figura sume las áreas totales de todas las instancias de sus subclases (no se debe instanciar la clase Figura tal cual).

- Crear una nueva clase TestFigura (que no esté en el mismo paquete), en cuyo método main haga lo siguiente: inicializa un círculo de radio 2, un cuadrado de lado 3, y un rectángulo de lados 4 y 5, visualizando su área según se crean, y que visualice finalmente el área total. Esta nueva clase debe utilizar el paquete figura, para poder realizar estas operaciones.