PAQUETES EN JAVA (POO)

Contenido

1. ¿Qué son los paquetes en Java?

Cuando hacemos programas más grandes, es normal que el número de clases vaya creciendo. Cada vez tenemos más y más clases. Meterlas todas en el mismo directorio no suele ser buena idea. Es mejor hacer **grupos de clases**, de forma que todas las clases que traten de un determinado tema o estén relacionadas entre sí vayan juntas. Por ejemplo, si hacemos un programa de una agenda de teléfonos que los guarde en una base de datos, podemos meter todas las clases que tratan con la base de datos en un **paquete** (grupo), todas las de ventanas en otro, las de imprimir en otro, etc.

Un **paquete** de clases es un grupo de clases que agrupamos juntas porque consideramos que están relacionadas entre sí o tratan de un tema común.

Java nos ayuda a organizar las clases en **paquetes**. En cada fichero .java que hagamos, al principio indicamos a qué **paquete** pertenece la clase que hagamos en ese fichero.

Por ejemplo, si decidimos agrupar nuestros programas de pruebas en un paquete "prueba", empezando por el programa Hola Mundo de abajo, pondríamos esto en nuestro fichero HolaMundo.java.

```
package prueba;

public class HolaMundo
{
	public static void main (String [ ] args)
	{
		System.out.println ("Hola mundo");
	}
}
```

El código es exactamente igual que si no hubiéramos puesto esa línea, pero al añadir la línea "package prueba;" al principio, para que todo funcione bien Java nos obliga a organizar los directorios, compilar y ejecutar de cierta forma. Si no añadimos paquete estará en un paquete por defecto: “default package”.

2. Estructura de directorios asociada.

Si hacemos que HolaMundo.java pertenezca al paquete prueba, java nos obliga a crear un subdirectorio "prueba" y meter el HolaMundo.java ahí.

Es decir, debemos tener esto así (Unix y Windows respectivamente):

- /<directorio_usuario>/mi_proyecto_HolaMundo/prueba/HolaMundo.java
- C:\<directorio_usuario>\mi_proyecto_HolaMundo\prueba\HolaMundo.java

Las mayúsculas y minúsculas son importantes. Tal cual lo pongamos en "package", debemos poner el nombre del subdirectorio.

3. Compilar una clase que está en un paquete

Para compilar la clase que está en el paquete debemos situarnos en el directorio padre del paquete y compilar desde ahí. En Unix y Windows:

$ cd /<directorio_usuario>/mi_proyecto_HolaMundo

$ javac prueba/HolaMundo.java

C:\> cd C:\<directorio_usuario>\mi_proyecto_HolaMundo

C:\> javac prueba\HolaMundo.java

Si todo va bien, en el directorio "prueba" se nos creará un fichero HolaMundo.class.

Ejemplo:

Nos colocamos en la carpeta src del proyecto de los apuntes y hacemos:

javac main/ImportacionDePaquetes.java

De esta forma puede acceder también a las clases del otro paquete.

4. Ejecutar una clase que está en un paquete

Una vez generado el HolaMundo.class en el directorio prueba, para ejecutarlo debemos estar situados en el directorio padre de "prueba". El nombre "completo" de la clase es "paquete.clase", es decir "prueba.HolaMundo". Por ello, para ejecutar, debemos hacerlo así

$ cd /<directorio_usuario>/mi_proyecto_HolaMundo

$ java prueba.HolaMundo

Hola Mundo

C:\> cd C:\<directorio_usuario>\mi_proyecto_HolaMundo

C:\> java **prueba.HolaMundo**
Hola Mundo

Si todo va como debe, debería salir "Hola Mundo" en la pantalla

5. El classpath cuando las clases están en paquetes

Cuando las clases están en paquetes, la variable **CLASSPATH** debe ponerse de tal forma que encuentre el paquete, no la clase. Es decir, en nuestro ejemplo:

$ CLASSPATH=/<directorio_usuario>/mi_proyecto_HolaMundo
$ export CLASSPATH
$ java prueba.HolaMundo
Hola Mundo

C:\> set CLASSPATH="C:\<directorio_usuario>\mi_proyecto_HolaMundo"
C:\> java prueba.HolaMundo
Hola Mundo

O bien, si usamos la opción -cp en la línea de comandos

$ java -cp /<directorio_usuario>/mi_proyecto_HolaMundo prueba.HolaMundo

Hola Mundo

C:\> java -cp "C:\<directorio_usuario>\mi_proyecto_HolaMundo" prueba.HolaMundo

Hola Mundo

6. Paquetes anidados

Para poder organizar mejor el código, java permite hacer **subpaquetes** de los paquetes y subpaquetes de los subpaquetes y así sucesivamente. Por ejemplo, si quiero partir mis clases de prueba en pruebas básicas y pruebas avanzadas, puedo poner más niveles de paquetes separando por puntos.

package prueba.basicas;
package prueba.avanzadas;

A nivel de subdirectorios tendría que crear los subdirectorios "basicas" y "avanzadas" debajo del directorio "prueba" y meter ahí las clases que correspondan.

Para compilar, en el directorio del proyecto habría que compilar poniendo todo el path hasta llegar a la clase. Lo mismo para el **CLASSPATH**.

Para ejecutar, el nombre de la clase va con todos los paquetes separados por puntos, es decir "prueba.basicas.HolaMundo" y "prueba.avanzadas.HolaMundoRemoto" y situándose en la carpeta contenedora del paquete.

Es decir:

/<directorio_usuario>/mi_proyecto_HolaMundo/prueba/basicas/HolaMundo.java
$ cd /<directorio_usuario>/mi_proyecto_HolaMundo
$ javac prueba/basicas/HolaMundo.java
$ java -cp /<directorio_usuario>/mi_proyecto_HolaMundo prueba.basicas.HolaMundo

Hola Mundo

C:\<directorio_usuario>\mi_proyecto_HolaMundo\prueba\basicas\HolaMundo.java
C:\> cd C:\<directorio_usuario>\mi_proyecto_HolaMundo
C:\> javac prueba\basicas\HolaMundo.java
C:\> java -cp C:\<directorio_usuario>\mi_proyecto_HolaMundo prueba.basicas.HolaMundo

Hola Mundo

7. Qué son los ficheros .jar

Cuando tenemos un programa grande, con varios paquetes y clases, ya sabemos cómo organizarlo, compilarlo y ejecutarlo. Sin embargo, si queremos usarlo en otra máquina o en otro proyecto, tenemos que llevarnos directorios enteros, con los ficheros que hay dentro.

Lo ideal es meter todos estos ficheros y directorios en un único fichero comprimido. Java, con su **comando jar**. Este comando está en el directorio bin de la carpeta donde tengamos java. Empaqueta todo lo que le digamos (directorios, clases, ficheros de imagen o lo que queramos) en un único fichero de extensión .jar.

Un fichero de extensión .jar es similar a los .zip de Winzip, a los .rar de Winrar o a los ficheros .tar del tar de Unix. De hecho, con Winzip o Winrar se puede ver y desempaquetar el contenido de un fichero .jar

Java nos da además otra opción, podemos ejecutar las clases del fichero .jar sin tener que desempaquetarlo. Estará listo para ejecutar.

Vamos a ver cómo hacer todo esto:

- **Cómo crear un fichero jar.**
- **Ver qué hay dentro de un jar.**
- **Modificar el contenido de un fichero jar.**
- **Ejecutar un fichero jar.**
- **El fichero de manifiesto.**

7.1 Cómo crear un fichero .jar

Para crear un fichero jar, en primer lugar, tenemos que tener todo ya perfectamente preparado y compilado, funcionando.

Si las clases de nuestro programa **no pertenecen a paquetes**, simplemente debemos meter las clases en el fichero .jar Para ello, vamos al directorio donde estén los ficheros .class y ejecutamos el siguiente comando:

```
$ cd directorio_con_los_class
$ jar cf fichero.jar fichero1.class fichero2.class fichero3.class …
```

La opción "c" indica que queremos crear un fichero.jar nuevo. Si ya existía, se machacará, así que hay que tener cuidado. La opción "f" sirve para indicar el nombre que le daremos al fichero, que va inmediatamente detrás. En nuestro caso, fichero.jar. Finalmente se pone una lista de ficheros .class (o de cualquier otro tipo) que queramos meter en nuestro jar. Se pueden usar comodines, estilo *.class para meter todos los .class de ese directorio.

Si las clases de nuestro programa **pertenecen a paquetes**, debemos meter en nuestro jar la estructura de directorios equivalente a los paquetes entera. Para ello, nos vamos al directorio padre de donde empiece nuestra estructura de paquetes. En el caso de nuestro HolaMundo con paquete, debemos meter el directorio **prueba** completo. El comando a ejecutar es este:

```
$ cd directorio_padre_de_prueba
$ jar -cf fichero.jar prueba
```

Las opciones son las mismas, pero al final en vez de las clases, hemos puesto el nombre del directorio. Esto meterá dentro del jar el directorio y todo lo que hay debajo.

Otra opción sería meter los .class, pero indicando el camino relativo para llegar a ellos:

```
$ cd directorio_padre_de_prueba
$ jar -cf fichero.jar prueba/HolaMundo.class
```

En Windows la barra va al revés...

7.2 Ver qué hay dentro de un fichero .jar

Para comprobar si nuestro jar está bien hecho, podemos ver su contenido. El comando es este

```
$ jar tf fichero.jar
```

La opción "t" indica que queremos un listado del fichero.jar. La opción "f" es igual que antes. Esto nos dará un listado de los class (y demás ficheros) que hay dentro, indicando en qué directorio están. Deberíamos comprobar en ese listado que están todas las clases que necesitamos y la estructura de directorios concuerda con la de paquetes.

7.3 Modificar un fichero .jar

Para cambiar un fichero dentro de un jar o añadirle uno nuevo, la opción del comando jar es "u". Si el fichero existe dentro del jar, lo reemplaza. Si no existe, lo añade.

Por ejemplo, si hacemos un cambio en nuestro HolaMundo.class con paquete y lo recompilamos, podemos reemplazarlo así en el jar

```
$ jar uf fichero.jar prueba/HolaMundo.class
```

7.4 Cómo ejecutar un jar

Para ejecutar un jar, simplemente debemos poner el fichero jar en el CLASSPATH. Ojo, hay que poner el fichero.jar, no el directorio en el que está el fichero.jar. Este suele ser un error habitual al empezar, pensar que basta con poner el directorio donde está el jar.

El path para indicar la ubicación del fichero puede ser absoluto o relativo al directorio en el que ejecutemos el comando java. El comando para ejecutar una clase dentro de un jar, en nuestro caso del HolaMundo con paquete, suponiendo que estamos en el directorio en el que está el fichero.jar, sería este:

```
$ java -cp ./fichero.jar prueba.HolaMundo
```

Hola Mundo

Simplemente, en la opción -cp del CLASSPATH hemos puesto el fichero.jar con su PATH relativo. Detrás hemos puesto el nombre de la clase completo, con su paquete delante. Nuevamente, en Windows la barra de directorio va al revés.

7.5 El fichero Manifiesto

Ejecutar así tiene una pega. Además de acordarse de poner la opción -cp, hay que saber el nombre de la clase que contiene el método main(). Además, si nuestro programa es muy grande, tendremos varios jar, tanto nuestros como otros que nos bajemos de internet o de donde sea. La opción -cp también puede ser pesada de poner en ocasiones.

Una opción rápida que a todos se nos ocurre es crearse un pequeño fichero de script/comandos en el que se ponga esta orden. Puede ser un fichero .bat de Windows o un script de Unix. Este fichero debe acompañar al fichero.jar y suponiendo que estén en el mismo directorio, su contenido puede ser este:

java -cp ./fichero.jar prueba.HolaMundo

Para ejecutarlo, se ejecuta como un fichero normal de comandos/script. Si el fichero se llama ejecuta.sh o ejecuta.bat, según sea Unix o Windows:

$ ./ejecuta.sh

Hola Mundo

C:\> ejecuta.bat

Hola Mundo

Sin embargo, java nos ofrece otra posibilidad de forma que no tengamos que hacer este fichero. Simplemente, en un fichero de texto metemos una línea en la que se ponga cuál es la clase principal. Este fichero se conoce como fichero de manifiesto y su contenido puede ser este:

Main-Class: prueba.HolaMundo

Cuando construimos el jar, debemos incluir este fichero de una forma especial. Por ejemplo, si el fichero lo llamamos manifiesto.txt y lo ponemos en el directorio donde vamos a construir el jar, el comando para hacerlo sería este:

$ jar cmf manifiesto.txt fichero.jar prueba/HolaMundo.class

En Windows, nuevamente, la barra al revés. Al comando de crear jar le hemos añadido la opción "m" para indicar que vamos a añadir un fichero de manifiesto. Hemos añadido además el fichero manifiesto.txt. El orden de las opciones "mf" es importante. El fichero de manifiesto y el fichero.jar se esperan en el mismo orden que pongamos las opciones. En el ejemplo, como hemos puesto primero la opción "m", debemos poner manifiesto.txt delante de fichero.jar. El resto de ficheros son los que queremos empaquetar.

Una vez construido, se ejecuta fácilmente. Basta con poner:

$ java -jar fichero.jar

La opción "-jar" indica que se va a ejecutar el fichero.jar que se ponga a continuación haciendo caso de su fichero de manifiesto. Como este fichero de manifiesto dice que la clase principal es prueba.HolaMundo, será esta la que se ejecute.

De esta forma nos basta con entregar el jar y listo. El comando para arrancarlo es sencillo.

Es más, en Windows, si lo configuramos para que los ficheros jar se abran con java y la opción -jar, bastará con hacer doble click sobre ellos para que se ejecuten.

8. Paquetes de Java

El lenguaje Java proporciona una serie de paquetes que incluyen ventanas, utilidades, un sistema de entrada/salida general, herramientas y comunicaciones. En la versión actual del JDK, algunos de los paquetes Java que se incluyen son los que se muestran a continuación, que no es una lista exhaustiva, sino para que el lector pueda tener una idea aproximada de lo que contienen los paquetes más importantes que proporciona el JDK. Posteriormente, en el desarrollo de otros apartados del Tutorial, se introducirán otros paquetes que también forman parte del JDK y que, incorporan características a Java que hacen de él un lenguaje mucho más potente y versátil, como son los paquetes Java2D o Swing.

java.applet

Este paquete contiene clases diseñadas para usar con applets. Hay la clase Applet y tres interfaces: AppletContext, AppletStub y AudioClip.

java.awt

El paquete Abstract Windowing Toolkit (awt) contiene clases para generar widgets y componentes GUI (Interfaz Gráfico de Usuario), de manipulación de imágenes, impresión, fuentes de caracteres, cursores, etc.. Incluye las clases Button, Checkbox, Choice, Component, Graphics, Menu, Panel, TextArea, TextField...

javax.swing

Lo mismo que el anterior, pero una versión mejorada.

java.io

El paquete de entrada/salida contiene las clases de acceso a ficheros, de filtrado de información, serialización de objetos, etc.: FileInputStream, FileOutputStream, FileReader, FileWriter. También contiene los interfaces que facilitan la utilización de las clases: DataInput, DataOutput, Externalizable, FileFilter, FilenameFilter, ObjectInput, ObjectOutput, Serializable...

java.lang

Este paquete incluye las clases del lenguaje Java propiamente dicho: Object, Thread, Exception, System, Integer, Float, Math, String, Package, Process, Runtime, etc.

java.net

Este paquete da soporte a las conexiones del protocolo TCP/IP y, además, incluye las clases Socket, URL y URLConnection.

java.sql

Este paquete incluye todos los interfaces que dan acceso a Bases de Datos a través de JDBC, Java DataBase Connectivity, como son: Array, Blob, Connection, Driver, Ref, ResultSet, SQLData, SQLInput, SQLOutput, Statement, Struct; y algunas clases específicas: Date, DriveManager, Time, Types...

java.util

Este paquete es una miscelánea de clases útiles para muchas cosas en programación: estructuras de datos, fechas, horas, internacionalización,etc. Se incluyen, entre otras, Date (fecha), Dictionary (diccionario), List (lista), Map (mapa), Random (números aleatorios) y Stack (pila FIFO). Dentro de este paquete, hay tres paquetes muy interesantes: java.util.jar, que proporciona clases para leer y crear ficheros JAR; java.util.mime, que proporciona clases para manipular tipos MIME, Multipurpose Internet Mail Extension (RFC 2045, RFC 2046) y java.util.zip, que proporciona clases para comprimir, descomprimir, calcular checksums de datos, etc. con los formatos estándar ZIP y GZIP.

9. Cómo importar clases entre distintos proyectos en Eclipse

Importar clases dentro de un mismo proyecto es trivial, solo tenemos que hacer un **import elPaquete.laClase**, y ya podremos crear objetos de las clases que se encuentran en el paquete que acabamos de importar.

Sin embargo, cuando intentamos importar entre distintos proyectos, veremos que no podemos hacerlo con este método, de hecho, no es trivial el cómo hacerlo.

Se puede hacer de dos formas, la primera:

9.1 Añadir a las fuentes de un proyecto, las fuentes de otro

1. Pinchamos sobre el proyecto al que queremos importar paquetes de otros proyectos, y con click-derecho accedemos a las propiedades (Properties) y ahí accedemos a “Java Build Path”.
2. En esta ventana accedemos a la pestaña “Source” y pinchamos en “Link Source”.
3. Aquí añadiremos el directorio de fuentes de otro proyecto que queremos importar. En “Linked Folder Location” ponemos la ruta de la carpeta src del otro paquete. En “Folder Name” nos pondrá src por defecto, pero este nombre ya existirá asi que lo cambiamos por el nombre que queramos.
4. Ya está hecho, ahora nos saldrá dentro del proyecto otra carpeta que apunta a la carpeta que indicamos y podremos importar dicho paquete para crear clases de este.

Otra solución que se suele utilizar más en grandes proyectos:

9.2 Añadir a las librerías de un proyecto, bytecode de otro

La solución es generar un fichero .jar con el bytecode de nuestro proyecto, con la evidente pega de tener que recompilar y crear el jar si cambiamos algo en el proyecto que estamos intentando utilizar en otro.

Lo podemos hacer con los siguientes pasos:

1. Para empezar, tendremos que compilar el proyecto que quieres usar en otro u otros.
2. Ahora nos dirigimos al directorio bin y añadimos todo el contenido en un fichero .zip
3. Renombramos el .zip a .jar
4. Solo queda incluir la librería .jar como uno de los jars de tu proyecto.

Otra opción diferente de las mencionadas sería simplemente añadir la carpeta donde tenemos las clases compiladas, sin comprimir, aunque en proyectos grandes es preferible comprimir.