Encapsulación y diseño (Programación de Ing. Telecomunicaciones)

Introducción

En una aplicación Java de cierto tamaño usaremos un gran número de clases. Para gestionarlas, las clases se suelen repartir en *paquetes*. Si hacemos la analogía con la organización de archivos o documentos en un ordenador, las clases serían los archivos concretos y los paquetes las carpetas. Podemos anidar paquetes de la misma manera que anidamos carpetas.

Cuando en un fichero .java queremos usar una clase no definida en dicho fichero, usaremos un import con la ruta desde la carpeta raíz al paquete donde se encuentra dicha clase. Los anidamientos de carpetas se expresan con ".".

Ejemplo

Supongamos las siguientes clases: Persona, en un paquete persona, y Empresa y PruebaEmpresa, ambas en un paquete empresa. Supongamos que la Empresa se compone de personas, por lo que en la clase Empresa importaremos la clase Persona. En este ejemplo tendríamos la siguiente estructura de carpetas:

```
src
|
└──proyecto
    |
    └──empresa
    |   |   Empresa.java
    |   |   PruebaEmpresa.java
    └──persona
        |   Persona.java
```

Y las siguientes líneas de código:

```
//En Empresa.java
package empresa;
import persona.Persona;
...
```

```
//En PruebaEmpresa.java
package empresa;
import persona.Persona;
import empresa.Empresa;
...
```

```
//En Persona.java
package persona;
...
```

¿Cómo podríamos compilar y ejecutar PruebaEmpresa? La idea es que para compilar un fichero .java, todas las clases de las que depende tienen que estar compiladas y visibles para el compilador. El compilador utilizará el classpath como base para buscar todos los demás paquetes. Por tanto, es clave ser ordenados.

No obstante, es un proceso complejo que se vuelve inviable en cuanto el proyecto crece en complejidad. Para esto existen herramientas como maven.

En el ejemplo anterior, estos comandos deberían ejecutarse en la carpeta raíz, *proyecto*:

```
javac -d classes src/persona/Persona.java
javac -d classes -cp classes src/Empresa/Empresa.java
java -cp classes empresa/PruebaEmpresa
```

Explicación:

- El modificador -d para compilar indica que almacenemos los ficheros .class con bytecode en la carpeta classes. Esto es recomendable para mantener el orden.
- El modificador -cp para compilar y ejecutar sirve para informar de cuál es la carpeta raíz que contiene los ficheros .class.
- En el primer comando, no necesitamos especificar el classpath porque no hay dependencias. Sólo indicamos dónde deben guardarse los ficheros de bytecode .class. Observa que se guardan de manera jerárquica, preservando la estructura de carpetas.
- En el segundo comando necesitamos especificar el classpath, que es la carpeta que hemos indicado antes con -d.
- En el tercer comando, para ejecutar, también necesitamos especificar el classpath, pues de lo contrario no encontrará los ficheros. Otra opción sería irnos a la carpeta classes y ejecutarlo desde ahí.

Modificadores

Veamos ahora los modificadores que podemos usar sobre clases, métodos y atributos. Los modificadores son palabras reservadas que permiten controlar ciertas propiedades de clases, métodos y atributos.

Modificadores de acceso

Clases

Sólo hay dos:

- **public**: la clase es accesible para otras clases.
- **default** (si no se pone nada, se aplica éste): la clase sólo es accesible para las clases del mismo paquete.

Métodos, atributos y constructores

Hay cuatro:

- **public**: el código (método o atributo) es accesible para otras clases.
- **private**: el código (método o atributo) es accesible en la propia clase.
- **default**: el código (método o atributo) es accesible para las clases del mismo paquete.
- **protected**: el código (método o atributo) es accesible para las clases del mismo paquete y para las subclases que la extiendan.

Para entender la diferencia entre los dos últimos, supongamos este ejemplo con dos paquetes:

```
proyecto
└──persona
│   │   Persona.java
└──empleado
    │   Empleado.java
```

Y supongamos que Empleado extiende a Persona. En este caso,

- Empleado podrá acceder a los atributos y métodos protected de Persona.
- Empleado no podrá acceder a los atributos y métodos default de Persona.

Otros modificadores

Clases

Hay dos modificadores:

- **final**, que indica que la clase no puede ser extendida (no se pueden definir subclases).
- **abstract**, que indica que la clase no puede ser instanciada (para crear instancias de la misma, es necesario definir primero subclases que no sean abstractas).

Métodos, atributos y constructores

Los principales son (hay otros que no veremos):

- **final**, que indica que estos atributos o métodos no pueden ser sobreescritos (overriden) por subclases.
- **static**, que indica que son atributos o métodos de la clase y no de las instancias. En el caso de métodos, sólo pueden hacer uso de atributos static.
- **abstract**, que sólo es aplicable a métodos de clases abstractas. Estos métodos no tienen cuperpo y se definen así para especificar que las subclases no abstractas tienen que proporcionar una implementación. También es importante para el funcionamiento correcto del polimorfismo.