Prácticas de Lenguajes, Tecnologías y Paradigmas de programación. Curso 2020-21

Parte I: Java

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Práctica I – Material adicional

Polimorfismo en Java: herencia y sobrecarga

Índice

1. Paquetes y clases en Java

Cuando los programas son relativamente grandes o se trabaja en equipo, es recomendable dividir el código en partes o paquetes para ahorrar tiempo de compilación ante modificaciones, favorecer la eficiencia del trabajo en equipo y controlar el acceso a las clases e interfaces evitando conflictos con identificadores.

Los *paquetes* son directorios que contienen ficheros con clases precompiladas (`.class`) siguiendo una jerarquía diferente a la de la herencia. Normalmente los paquetes incluyen clases siguiendo un criterio de cohesión funcional. Por ejemplo, el paquete `java.awt.geom` contiene clases para definir operaciones sobre objetos relacionados con la geometría en dos dimensiones.

Las clases pueden importarse individualmente o por paquetes añadiendo el comodín `*` al final de la instrucción para indicar que se desea tener acceso a todas las clases del paquete. Las importaciones se explicitan al principio del fichero. Por ejemplo:

```
import java.applet.Applet;
import java.awt.geom.*;
```

donde los puntos separan subpaquetes, como se verá más adelante. Esta notación también nos permite controlar el espacio de nombres diferenciando las clases entre sí cuando tienen el mismo nombre. Así, por ejemplo, en Java existen tres clases predefinidas con el mismo nombre `Timer`, cada una en un paquete distinto y con una funcionalidad totalmente distinta. Las tres pueden usarse en el mismo programa importándolas al principio del fichero:

```
import java.util.Timer;
import javax.management.timer.Timer;
import javax.swing.Timer;
```

y definiendo variables utilizando la ruta para llegar al directorio donde se encuentra la clase:

```
java.util.Timer t1 = new java.util.Timer();
javax.management.timer.Timer t2 = new javax.management.timer.Timer();
javax.swing.Timer t3 = new javax.swing.Timer();
```

También pueden realizarse importaciones estáticas para evitar los nombres de las clases en la invocación de métodos estáticos, referencias a constantes estáticas, etc. Por ejemplo:

```
import static java.lang.Math.*;
...
double r = cos(PI * theta);
```

En un fichero se puede definir más de una clase pero sólo una puede ser pública. En este caso, el nombre del fichero debe coincidir con el de la clase

pública (con la extensión `.java`). El resto de las clases del fichero sólo serán visibles dentro del paquete. Es usual que los ficheros contengan una sola clase, pero a veces se declaran más clases cuando sólo se usan en las clases definidas en ese mismo fichero. Para utilizar paquetes en Java, se necesita generar una estructura de directorios que tenga la misma jerarquía que las librerías que se crean. Cuando se quiere que una clase pertenezca a una librería, el nombre del paquete al que pertenece la clase ha de especificarse en la primera línea del fichero con la sintaxis

```
package paquete1.paquete2. ... .paqueteN;
```

La ruta de directorios descrita con la notación punto, describe el camino relativo desde el directorio donde se guarda todo el proyecto hasta el directorio donde se guardará la clase.

Por ejemplo, si tenemos un proyecto en el directorio `lineales`, y se desea que las clases definidas en el fichero `Pila.java` formen parte de un paquete `modelos`, el cual es un subpaquete del paquete `librerias`, se escribiría la instrucción `package librerias.modelos;` en la primera línea del fichero. Esto equivale a decir que las clases definidas en el fichero `Pila.java` están disponibles en el directorio `/.../lineales/librerias/modelos`.

Existe otro tema que concierne a los paquetes y la ejecución de algunos de los comandos como el de compilación (`javac`), ejecución (`java`) y generación de documentación (`javadoc`). Por defecto, los comandos buscan las librerías a partir del directorio donde está instalado el JDK y del directorio donde se ejecutan. Cuando se necesitan otros directorios, los comandos suponen la existencia de la variable de entorno `CLASSPATH`, en la que se define una secuencia de caminos hasta los directorios, a partir de los cuales, los comandos buscan los paquetes.

2. Modificadores. Visibilidad

La accesibilidad a las clases no depende solo del paquete en el que se encuentren, sino también de otros factores. Vamos a revisar brevemente y de forma genérica la visibilidad de las clases y de sus componentes según la definición que hagamos de ellas.

Definición de una clase

Sintaxis:

modifAcceso modifClase `class` NomClase
[`extends` NomClase]
[`implements` listaInterfaces]

donde

modifAcceso indica desde dónde se puede acceder al uso de la clase.

`public` la clase es visible desde cualquier otra clase sin importar el paquete en el que esté.

sin especificar accesible sólo a las clases del mismo paquete.

`private` sólo es visible en la clase en la que se define[1].

modifClase afecta a las clases derivadas de la clase.

`abstract` para las clases abstractas.

`final` evita que la clase pueda ser derivada.

Herencia tipos de clases de las que se deriva.

`extends` se utiliza para indicar que la clase hereda de NomClase, en Java sólo se permite heredar de una única clase padre. En caso de no incluir la cláusula extends, se asumirá que se está heredando directamente de la clase java.lang.Object

`implements` indica que esta clase es de los tipos de interfaz indicados por listaInterfaces, pudiendo existir tantos como queramos separados por comas.

Definición de variables

Sintaxis:

[**modifVisibilidad**] [**modifAtributo**] tipo nomVariable;

donde

modifVisibilidad delimitan el acceso desde el exterior de la clase.

public accesible desde cualquier clase.

private solo es accesible desde la clase donde se define.

protected accesible en las clases del mismo paquete en el que se define así como en todas sus clases derivadas, incluso cuando pertenecen a otro paquete distinto.

sin especificar accesible desde cualquier clase del mismo paquete.

modifAtributos características especiales.

static la variable no forma parte de los objetos sino que es común a todos los objetos de la clase.

final el primer valor que recibe la variable es inalterable.

transient excluye el atributo de la serialización [2] aunque la clase implemente la interfaz `Serializable`.

volatile informa de que el atributo es accesible de forma asíncrona por dos hilos impidiendo que el compilador altere el orden de las instrucciones.

Definición de métodos de una clase

Sintaxis:

[**modifVisibilidad**] [**modifFunción**] tipo nomFunción (listaParámetros)

donde

modifVisibilidad mismas normas que los atributos.

[1] Se pueden definir clases dentro de otras clases y se las conoce como *clases internas*.

[2] Aplanar un objeto consiste en obtener su información en forma de cadena de caracteres. El método `toString` de Java, es una herramienta con la que realizar un aplanamiento definido por el programador. La serialización en Java es un aplanamiento con una sintaxis predefinida. Suele utilizarse para transportar objetos a otros dispositivos (disco,...) o para transmitirlo a través de la red.

modifFunción puede tener los siguientes valores:

static es de la clase, no se aplica a los objetos. Cuando se invoca un método estático desde otra clase, se precede con el nombre de la clase en la que se define, seguido de un punto.

sin especificar es un método dinámico.

final el método no se puede sobreescribir en una clase derivada.

abstract se delega la implementación a una clase derivada.

native escrito en código nativo[3] resultante de alguna compilación.

synchronized ejecutado en exclusividad por un hilo [4]. Se usa para el control de la concurrencia.

[3] Código nativo es aquel que es ejecutable directamente por un procesador y se puede obtener compilando un lenguaje de alto nivel.

[4] Si varios hilos se están ejecutando concurrentemente e intentan ejecutar al mismo tiempo un método dinámico sobre el mismo objeto, si el método está marcado como synchronized, sólo uno se ejecuta y los demás esperan a que acabe. Si el método es estático, sólo se ejecuta un método estático de la clase a la vez. Uno de los hilos que espera tomará el relevo en la ejecución siguiendo una política de asignación de la exclusividad.