Objetivos Crear y utilizar paquetes. Crear una jerarquía de clases.

Índice

1. Paquetes

Un **paquete** es una biblioteca de tipos (clases e interfaces). Veamos cuales pueden ser las diferentes utilidades que puede tener agrupar clases en un paquete:

- **Evitar conflictos de nombres**

  Cuando se diseña un programa en Java (o en general con un lenguaje orientado a objetos) se modela el dominio del problema identificando y definiendo una serie de tipos (tipo se usa en este contexto como sinónimo de clase) a los que asignamos un nombre. Cada nombre de tipo debe ser único ya que es lo que lo distingue de otros tipos. Para evitar el problema de conflicto de nombres de tipo se utilizan paquetes.

  Cada tipo tiene un nombre simple, y cada paquete tiene un nombre de paquete. El nombre del paquete seguido de un punto y el nombre del tipo constituye lo que se conoce como nombre completo.

  Por tanto los paquetes se pueden ver como una forma de evitar los conflictos de nombres en los programas escritos en Java. En lugar de intentar que el nombre simple de cada tipo sea único, nos tenemos que preocupar de que el nombre completo (incluyendo el nombre del paquete) sea único.

- **Organización**

  Otro modo de ver los paquetes es como una herramienta que ayuda a organizar los tipos que se crean. Con los paquetes organizamos los programas en grupos de tipos relacionados entre si, y organizamos los grupos jerárquicamente.

- **Biblioteca**

  Una tercera posibilidad de contemplar los paquetes es simplemente como una biblioteca de tipos. Un programa que escribimos puede hacer uso de bibliotecas desarrolladas por terceros y puestas a nuestra disposición como paquetes.

- **Implementación**

  La ultima interpretación que se puede hacer de un paquete es observarlo como una herramienta que puede ayudar a separar la interfaz de la implementación. Se pueden dar distintos privilegios

a los tipos dentro del paquete y se pueden definir tipos que solo sean accesibles por otros tipos que estén en el mismo paquete y hacer públicos solo aquellos que nos interesen.

1.1. Nombres de los paquetes

Los paquetes que importamos en nuestros programas pueden venir de diferentes fuentes, por lo tanto es importante que a la hora de asignar un nombre a un paquete este sea diferente a los nombres de los paquetes desarrollados por otros. Java recomienda poner al paquete un nombre que sea el nombre de dominio de la organización al revés. Por ejemplo si la organización es `uv.es` la recomendación es que nuestros paquetes comiencen del siguiente modo: `es.uv`

1.2. Declaración de un paquete

Una clase es colocada en un paquete introduciendo una declaración de paquete al principio del fichero fuente. Para ello utilizaremos la palabra reservada `package` seguida por el nombre del paquete y un punto y coma.

Por ejemplo la siguiente clase pertenece al paquete `es.uv.lp`

```
package es.uv.lp;

/** Clase Alumno
  * @version 1.0
  */
public class Alumno{
   private String nombre;
   private String NIP;
   private float nota;

   // Constructor(es)

   /** Metodo asignarNota sirve para asignar una nota
     * @param n nota
     */
   public void asignarNota(float n){
      nota = n;
   }
   ...
}
```

1.3. La variable de entorno CLASSPATH

La variable de entorno `CLASSPATH` define las rutas donde el compilador busca los paquetes. Cuando dentro de un determinado directorio se colocan una serie de clases y no se las declara como pertenecientes a ningún paquete, las clases son encontradas por el compilador y la máquina virtual (ejecutados ambos desde el directorio) ya que normalmente el directorio actual (.) está incluido en la variable de entorno `CLASSPATH`. Sin embargo, al importar paquetes que no se encuentren en el directorio de trabajo hay que indicar tanto al compilador como a la MVJ la ruta donde debe buscar las clases.

Por poner un ejemplo, vamos a suponer que colocamos el paquete bajo el directorio `c:\tmp\prac2\paquete`. La clase que se ha puesto como ejemplo en el apartado anterior deberá estar colocada en una ruta:

`c:\tmp\prac2\paquete\es\uv\lp\Alumno.class`

Queremos definir una clase `Clase` que utiliza la clase `Alumno`. Supongamos que esta clase se coloca en el directorio `c:\tmp\prac2\prog`.

```
import es.uv.lp.Alumno;

public class Clase{
   private Alumno[] als;
```

```
    private Alumno[] delegados;
    private String aula;

    public void matricularAlumno(Alumno al){
        //Codigo para añadir el alumno al array
    }
    ...
}
```

Si estamos en el directorio `c:\tmp\prac2\prog` la variable de entorno `CLASSPATH` debe contener la ruta `c:\tmp\prac2\paquete` ya que el compilador toma todas la rutas de esta variable como base para buscar los paquetes, formando en este caso concreto (al encontrar el `import`) la ruta (tras sustituir los puntos por \) `c:\tmp\prac2\paquete\es\uv\lp\` que es donde se encuentra la clase `Alumno`.

Finalmente, indicar que también es posible usar dentro del programa el nombre completo (incluyendo el paquete) para referirnos a la clase (este nombre es conocido como *qualified name*). Por tanto el siguiente código es perfectamente válido, aunque es más complicado de leer (y de escribir).

```
public class Clase{
    private es.uv.lp.Alumno[] als;
    private es.uv.lp.Alumno[] delegados;
    private String aula;

    public void matricularAlumno(es.uv.lp.Alumno al){
        //Codigo para añadir el alumno al array
    }
    ...
}
```

Para poder compilar correctamente la clase `Clase` desde su directorio `c:\tmp\prac2\prog` debemos de escribir en la linea de ordenes:

```
c:\tmp\prac2\prog>set CLASSPATH=c:\tmp\prac2\paquete;.
```

y compilar normalmente

```
c:\tmp\prac2\prog>javac Clase.java
```

otra posibilidad es pasar la ruta como una opción al compilador:

```
c:\tmp\prac2\prog>javac -classpath c:\tmp\prac2\paquete;. Clase.java
```

Resumiendo

Para definir que una clase pertenece a un paquete hay que usar la palabra reservada `package` *seguida por el nombre que va se le va a dar al paquete. Para importarlo hay que usar la palabra* `import` *seguida del nombre del paquete. Los puntos que aparecen en el nombre del paquete se corresponden con una ruta en nuestra estructura de directorios. Para hacer visible el paquete a las utilidades (*`javac` y `java`*) hay que poner a partir de qué directorios se deben buscar para esto usamos la variable de entorno* `CLASSPATH` *o la opción* `-classpath` *pasada a las utilidades.*

1.4. Visibilidad de los elementos fuera del paquete.

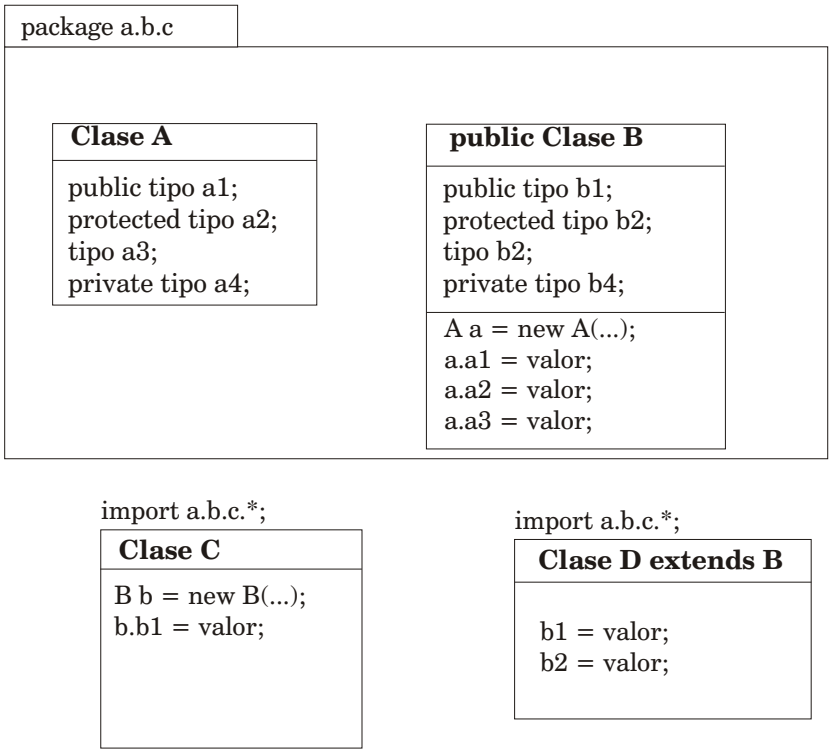
Accesibilidad de clases

Visible para	*public*	*package*
Otra clase del mismo paquete	*si*	*si*
Una clase de otro paquete	*si*	*no*

Accesibilidad de miembros:

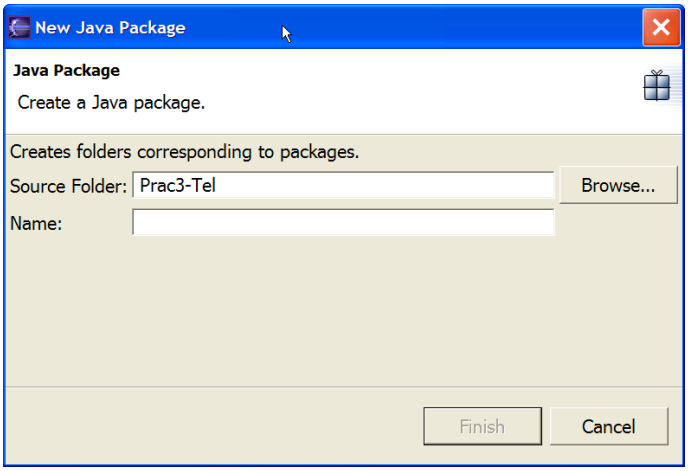
Accesible a	*public*	*protected*	*package*	*private*
La misma clase	*si*	*si*	*si*	*si*
Clase en el mismo paquete	*si*	*si*	*si*	*no*
Subclase en otro paquete	*si*	*si*	*no*	*no*
No subclase en otro paquete	*si*	*no*	*no*	*no*

Esto se muestra gráficamente en la siguiente figura:



1.5. Paquetes y Eclipse

Desde Eclipse se puede crear un paquete dentro de un proyecto que se haya declarado del tipo `Java`. Para ello, una vez creado el proyecto, pulsando con el botón derecho del ratón sobre el nombre del proyecto en la vista `Package Explorer` aparece un menú contextual en el que podemos seleccionar ***New*** $\longrightarrow$ ***Package***. Pulsando sobre esta opción aparece una ventana similar a la que se muestra a continuación donde se puede introducir el nombre del paquete:



Una vez dado el nombre del paquete (que lo único que hace es crear la estructura de directorios a partir del nombre que se ha proporcionado) podemos crear clases dentro del paquete pulsando con el botón derecho sobre él y seleccionando ***New*** $\longrightarrow$ ***Class***.

Si hacemos esto vemos que automáticamente se pone como primera linea del fichero con la clase

```
package nombre;
```

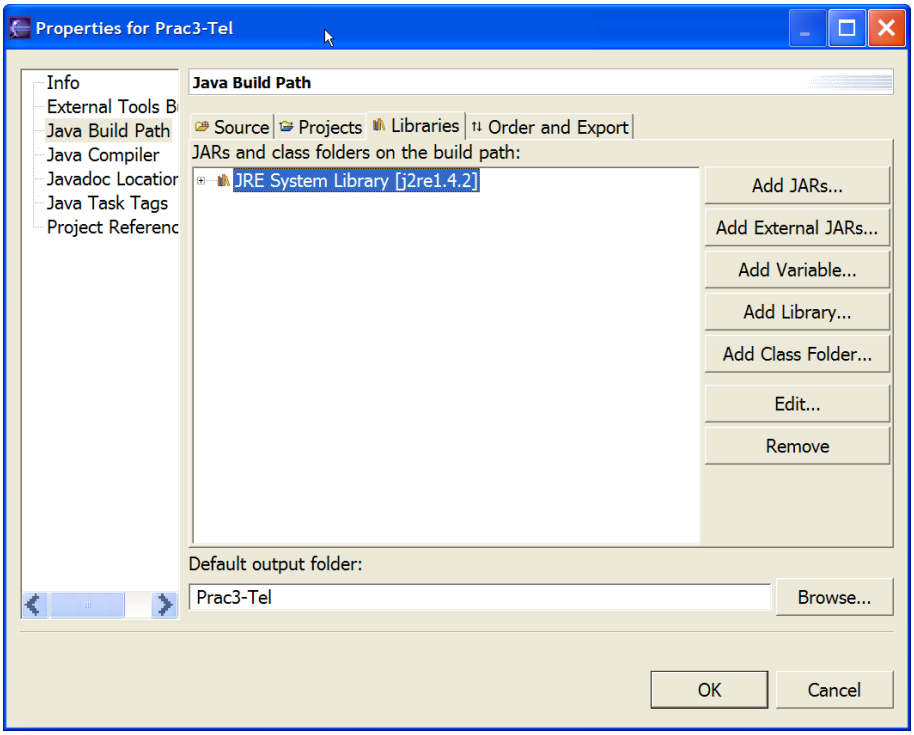
donde `nombre` es el nombre del paquete que hemos introducirlo al crearlo.

Si necesitamos importar clases de un paquete que alguien ha desarrollado y que está en un formato `jar` (comprimido) hay que indicarle a Eclipse que lo utilice. Eclipse necesitará acceder a este fichero en dos momentos:

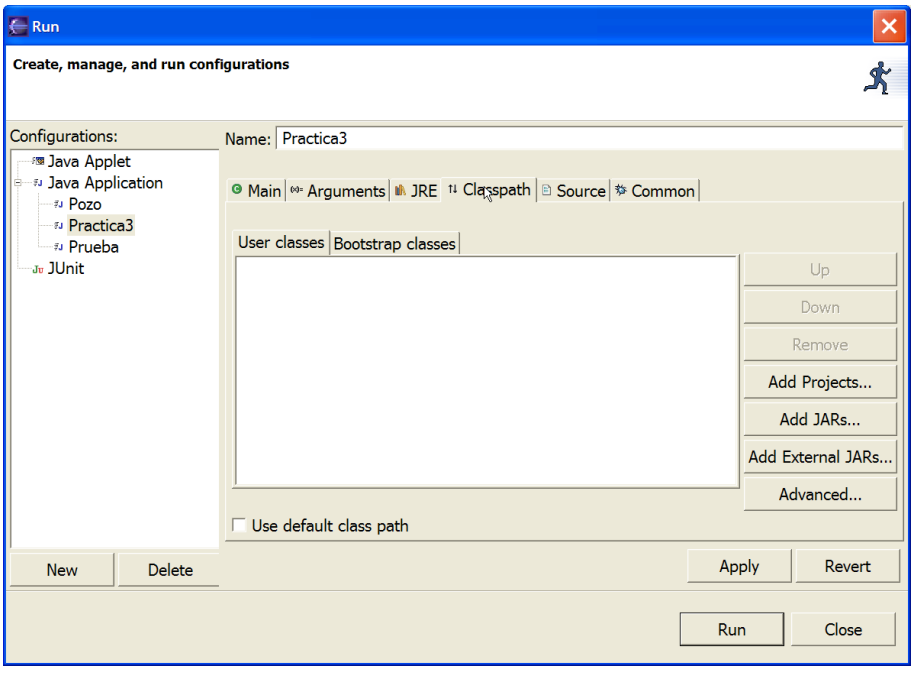
1. en el momento de la compilación para comprobar que se están utilizando de forma adecuada las clases declaradas y,
2. en el momento de la ejecución.

Para indicarle que busque clases en un determinado fichero a la hora de **compilar** se sigue la siguiente secuencia en el menú ***Project*** $\longrightarrow$ ***Properties***. Aparece una ventana en la que pulsando en el panel de la izquierda sobre ***Java Build Path*** y seleccionando en el panel de la derecha la pestaña ***Libraries*** se puede indicar qué fichero `jar` debe considerar pulsando sobre

- ***Add jar...*** si el fichero `jar` está importado en el proyecto o,
- ***Add external jar..*** si no se ha importado el fichero `jar` al proyecto.

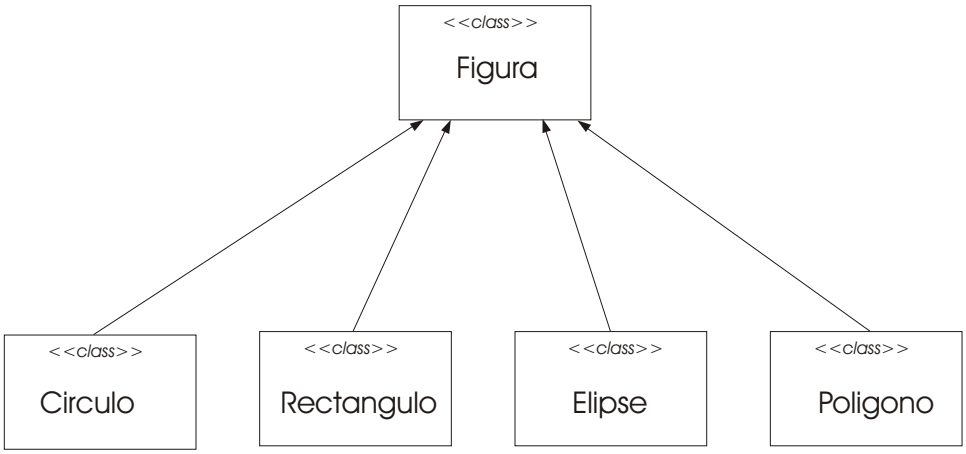


Para indicarle que clases en un determinado fichero a la hora de **ejecutar** se sigue la siguiente secuencia en el menú ***Run*** $\longrightarrow$ ***Run ...***. En la ventana que aparece y tras crear la configuración para ejecutar (dar un nombre, indicar el proyecto y la clase que contiene el `main`), pulsando sobre la pestaña `Classpath` y quitando la opción de ***Use default class path*** se puede seleccionar el fichero `jar` igual que en el caso anterior.



2. Tareas

Realizar la siguiente jerarquía de clases en un paquete llamado `es.uv.tel.lp.p3`



donde la flecha indica una relación de herencia.

La clase `Figura` debe seguir la siguiente especificación:

```
public class Figura{
   protected int posx;
   protected int posy;
   protected int velocidadx;
   protected int velocidady;
   protected Color color;

   // posicion inicial de la figura en (px,py)
   // velocidad de la figura en (velx,vely)
   // Color de la figura en Color (de java.awt)
   public Figura(int px, int py, int velx, int vely, Color c){}

   // En este caso el cuerpo de este metodo esta vacio ya que no sabemos como
   // pintar una Figura hasta no especificar que tipo de figura
   public void pinta(Graphics g){}


   // Tiempo esta en milisegundos y la velocidad es en puntos/segundo
   // hay que realizar una conversion
   public void mueve(int tiempo){}

   // Devuelve posx
   public int getX(){}

   // Devuelve posy
   public int getY(){}

   // Devuelve velocidadx
   public int getVX(){}

   // Devuelve velocidady
   public int getVY(){}

   // Establece una nueva velocidadx
   public void cambiaVX(int vx){}

   // Establece una nueva velocidady
   public void cambiaVY(int vy){}
}
```

Todas las clases deben sobreescribir el método `pinta(Graphics g)` de `Figura` ya que cada figura concreta debe especificar cómo se pinta. Además cada figura tiene sus propios atributos (debido a la geometría de cada figura). Para ver cuales son los atributos que debe tener cada clase se debe consultar en la clase `Graphics` del paquete `java.awt` los métodos que ofrece para pintar las figuras propuestas.

Se proporciona el paquete `es.uv.tel.lp.mov` (en el fichero `movimiento.jar` que contiene una clase `Ventana` que ofrece un método `anadirFigura(Figura fig)` para ir añadiendo figuras a la ventana. Cada vez que se añade a la ventana una figura ésta se pone en movimiento según se haya especificado en el método `mueve(...)` que tienen todas las figuras.

El método `mueve(...)` de `Figura` determina que las figuras sigan un movimiento con velocidad uniforme:

$$x(t+1) = x(t) + v_x \Delta t$$
$$y(t+1) = y(t) + v_y \Delta t$$

donde Δt es el intervalo de tiempo (en segundos). Este intervalo de tiempo lo recibe mediante las llamadas que se realizarán desde `Ventana` y coincide con el tiempo que se pasa al constructor de `Ventana` salvo que este último está en milisegundos. Este es el movimiento que adquieren todas las figuras por heredar de `Figura`. Aunque es posible que alguna figura pueda ocultar el método de la clase padre con el fin de definir su propia forma de movimiento.

Material adicional a este enunciado:

- El paquete movimiento.jar.
- La documentación (API) del paquete APImovimiento.zip.