Objetivos	Desarrollo y utilización de paquetes en Java. Saber qué representa la variable de entorno CLASSPATH. Clases del paquete java.io del paquete java.util

Índice

1. Paquetes....1
1.1. Nombres de los paquetes....2
1.2. Declaración de un paquete....2
1.3. La variable de entorno CLASSPATH....2
1.4. Visibilidad de los elementos fuera del paquete....4
2. Entrada/Salida: paquete java.io....5
2.1. Flujos predefinidos....5
3. Clases de utilidad: paquete java.util....5
4. Tareas....6
5. Revisión....7

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 última 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 debería 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.

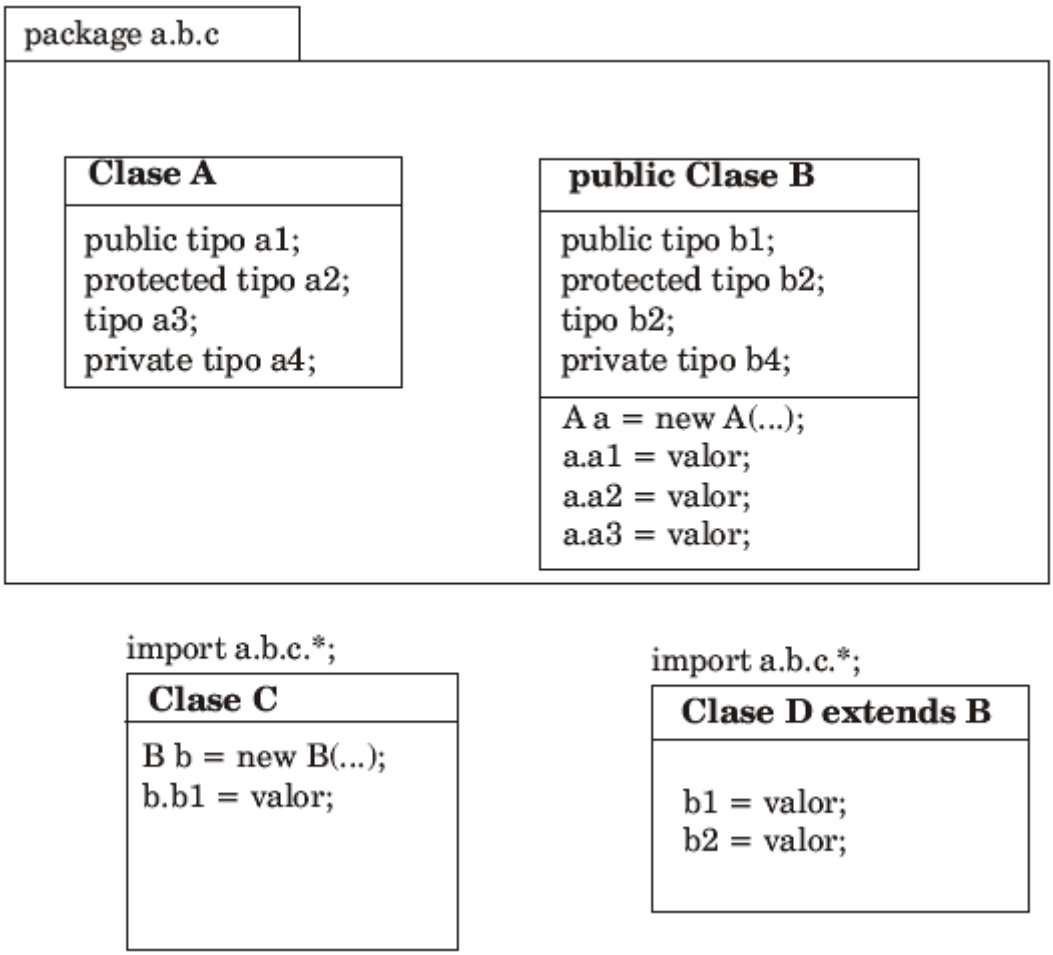
Accesibilidad de clases

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:



2. Entrada/Salida: paquete java.io

El paquete java.io proporciona una serie de clases e interfaces para soportar la entrada/salida (E/S). Los programas en Java realizan la E/S a través de flujos (streams). Un flujo es una abstracción de una fuente o sumidero de información. Un flujo está asociado a un dispositivo físico. Todos los flujos se comportan del mismo modo, a pesar de que los dispositivos físicos reales sean muy diferentes (por ejemplo se puede obtener un objeto del tipo InputStream asociado a una conexión HTTP con una determinada URL y leer la información que en este caso sería una página web, o ese mismo objeto puede estar asociado un fichero local del cual queremos leer los datos, lo que se diferencia es el dispositivo del que se lee pero a todos los efectos el flujo se comporta del mismo modo).

En Java se diferencian dos tipos de flujos: flujos de bytes y de caracteres. Los flujos de bytes se utilizan para leer o escribir datos binarios y los flujos de caracteres son apropiados para gestionar E/S basada en caracteres. Los caracteres están codificados en Unicode (2 bytes).

2.1. Flujos predefinidos

En el paquete java.lang (paquete que se importa automáticamente) existe una clase llamada System que entre otras cosas tiene tres atributos que son flujos predefinidos: in, out y err. Estos tres atributos están definidos como public y static. Esto significa que se puede acceder a ellos con el nombre de la clase y el nombre del atributo (por ser static): System.in, System.out y System.err.

System.out hace referencia a la salida estándar, que por defecto es la consola, System.in es un flujo de entrada estándar, asociado al teclado. System.err es el flujo de error estándar, que está también asociado a la consola. System.in es un objeto del tipo InputStream; System.out y System.err son objetos del tipo PrintStream. Todos estos flujos son de bytes, aunque se suelen utilizar para leer y escribir caracteres. Como se verá estos flujos se pueden envolver (wrap en inglés) con flujos basados en caracteres, pasándolo como argumento al constructor de una de estas clases de "envoltura".

Resumiendo

Si queremos realizar entrada/salida usando los flujos predefinidos hay que consultar la ayuda de InputStream y de PrintStream para ver cuales son los métodos que tienen estas clases. Además podemos buscar cuales son las clases de E/S (clases de envoltura) que reciben como parámetro del constructor objetos de alguno de estos dos tipos con el fin de mejorar la funcionalidad de aquellas.

Ejemplos:

```
BufferedReader br = new BufferedReader( new InputStreamReader( System.in) );
BufferedReader br = new BufferedReader( new FileReader ("fichero.txt"));
```

3. Clases de utilidad: paquete java.util

El paquete java.util contiene una serie de clases e interfaces que proporcionan una muy amplia funcionalidad. Estas clases se usan en otros paquetes y las podemos utilizar en nuestros programas importándolas. Entre ellas podemos citar: generación de números aleatorios (Random), manipulación de fecha y hora (TimeZone, Timer, TimerTask, Date, Calendar, · · ·), operaciones con cadenas (StringTokenizer), · · · .

Además de las anteriores el paquete contiene las colecciones que son estructuras para almacenar un grupo de objetos, por citar algunas: Arrays, ArrayList, LinkedList, Vector, Stack, HashMap, HashTable, · · · . Además de estas clases, el paquete contiene una serie de interfaces. Para ver un listado completo véase la ayuda.

4. Tareas

Realizar un paquete (colocado en el directorio c:\tmp\prac2\paquete) con nombre es.uv.lp.p2 que conste de dos clases: Detalle y Factura siendo ésta la única visible fuera del paquete.

La clase Detalle consta de

- Cuatro atributos: un int para la cantidad, un String para la descripción, un float para el importe unitario, y un float para el total.
- un constructor, con tres argumentos, que son la cantidad, la descripción y el importe unitario, y que calcula el total como el producto de la cantidad por el importe unitario.
- un método que sobre-escribe el método String toString() para dar el estado del objeto (el valor de sus atributos), en el formato:
  cantidad+"/t"+descripción+"/t"+importe unitario+"/t"+total
- un método para dar el total float darTotal()

La clase Factura consta de

- Seis atributos: un int para el numero de factura, un String para el cliente, un float para el iva, un LinkedList para el detalle, un float para el importe neto y un float para el importe total.
- un constructor al que se le pasa el nombre de un fichero que contiene los datos de la factura con el siguiente formato:
  primera linea: numerofactura,cliente,iva
  resto lineas: cantidad,descripción,importe unitario
  de tal forma que en el constructor se lea el fichero y se almacenen las lineas del detalle de la factura en el atributo del tipo LinkedList.

- un método public void mostrarFactura() que muestra todos los datos de la factura (todos los atributos de la factura con su nombre, y ademas recorra la LinkedList del detalle invocando al método toString para mostrar todas las líneas del detalle). Es importante intentar que el formato se parezca lo más posible al de una factura real.
- un método public void mostrarFacturaResumida, que mostrara la misma información que el anterior, pero omitiendo el detalle..

Realizad una clase Prueba que contenga el main y que debe residir en el directorioc:\tmp\prac2\prog
donde se creen dos facturas a partir de dos ficheros y donde se les envíen mensajes para comprobar su correcto funcionamiento.

Ayuda

Para procesar cada una de las líneas del fichero se propone utilizar la clase StringTokenizer del paquete java.util. La conversión de String a float se puede hacer utilizando el método estático parseFloat(String s) de la clase Float. Consultad la ayuda.

Cuando se necesite recorrer una LinkedList con el objetivo de mostrar el contenido, se puede obtener un objeto del tipo ListIterator y utilizar éste (con sus métodos bolean hasNext() y Object next()) para mostrar el contenido.

Para leer el fichero se propone utilizar un FileReader envuelto en un BufferedReader, ambas clases están en el paquete java.io.

5. Revisión

Paquetes	Se declara la pertenencia de una clase o conjunto de clases a un paquete comenzando el fichero con la palabra reservada package seguida del nombre del paquete. Las clases públicas son visibles fuera del paquete, si no se pone nada delante de class significa que la clase sólo puede ser vista dentro del paquete.
CLASSPATH	Es una variable de entorno que contiene las rutas en las que se buscan los paquetes.
Tareas	Se ha desarrollado un paquete y se ha visto cómo se puede utilizar (importándolo). Se han visto clases de los paquetes java.util y java.io.