APRENDERAPROGRAMAR.COM

PROYECTOS JAVA. CLÁUSULAS PACKAGE E IMPORT. JERARQUÍA Y VISIBILIDAD DE CLASES. ORGANIZACIÓN EN BLUEJ. EJEMPLOS. (CU00674B)

Sección: Cursos
Categoría: Curso “Aprender programación Java desde cero”

Fecha revisión: 2029

Resumen: Entrega nº74 curso Aprender programación Java desde cero.

Autor: Alex Rodríguez

PROYECTOS JAVA EN PAQUETES (PACKAGES)

Los proyectos (en general podríamos establecer equivalencia de proyecto con programa, pero no siempre es así; proyecto es un conjunto de código que se mantiene agrupado) en Java se suelen organizar en paquetes (packages). El concepto de paquete viene siendo similar al de carpeta en Windows: un contenedor donde mantenemos cosas relacionadas entre sí.

De hecho, en entornos como Eclipse o BlueJ esto es exactamente así: al crear packages, veremos que creamos una carpeta. No obstante, la organización en packages tiene muchas más implicaciones como veremos a continuación.

La organización del proyecto será por tanto similar a la organización de archivos: en un paquete podremos tener por ejemplo clases de tipo A, en otro clases de tipo B y así sucesivamente. A su vez, un paquete puede contener subpaquetes: por ejemplo el paquete A puede contener a los subpaquetes A.1, A.2 y A.3. **Un package es una agrupación de clases afines.** Recuerda el concepto de librería existente en otros lenguajes o sistemas.

Una clase puede definirse como perteneciente a un package y puede usar otras clases definidas en ese o en otros packages. Los packages delimitan el espacio de nombres (space name). El nombre de una clase debe ser único dentro del package donde se define. Dos clases con el mismo nombre en dos packages distintos pueden coexistir e incluso pueden ser usadas en el mismo programa. Una clase se declara perteneciente a un package con la cláusula package incluida como una línea de código, cuya sintaxis es: *package nombrePackage;*. La cláusula *package* debe ser la primera sentencia del código fuente. Por ejemplo, una clase puede comenzar así:

```
package miPackage;
    . . .
    public class miClase {
. . .
```

Este código implica que la clase miClase pertenece al package miPackage. La cláusula package es opcional. Si no se utiliza, las clases declaradas en el archivo fuente no pertenecen a ningún package concreto, sino que pertenecen a un package por defecto (sin nombre). La agrupación de clases en packages es conveniente desde el punto de vista organizativo, para mantener bajo una ubicación común clases relacionadas que cooperan desde algún punto de vista. También **resulta importante por la implicación que los packages tienen en la visibilidad y acceso del código** entre distintas partes de un programa. Cuando se referencia cualquier clase dentro de otra se asume, si no se indica otra cosa, que ésta otra está declarada en el mismo package. Por ejemplo:

```
package Directivos;
    . . .
    public class JefeDeAdministracion { //Ejemplo aprenderaprogramar.com
            private Persona jefeAdmin;
      . . .
}
```

En esta declaración definimos la clase JefeDeAdministracion perteneciente al package Directivos. Esta clase usa la clase Persona. El compilador asume que Persona pertenece también al package Directivos, y tal como está hecha la definición, para que la clase Persona sea accesible (conocida) por el compilador, es necesario que esté definida en el mismo package. Si esto no fuera así, es necesario hacer accesible el espacio de nombres donde está definida la clase Persona a nuestra nueva clase. Esto se hace con la cláusula *import*. Supongamos que la clase Persona estuviera definida de esta forma:

```
package TiposBasicos;
public class Persona {
   private String nombre;
}
```

Entonces, para usar la clase Persona en nuestra clase JefeDeAdministracion deberíamos poner:

```
package Directivos;
import TiposBasicos.*;
     class JefeDeAdministracion {
          private Persona jefeAdmin;
   . . .
}
```

Con la cláusula *import TiposBasicos.*;* se hacen accesibles todas las clases declaradas en el package TiposBasicos. Si sólo se quisiera tener accesible la clase Persona se podría declarar: *import TiposBasicos.Persona;*. También es posible hacer accesibles los nombres de un package sin usar la cláusula *import* calificando completamente los nombres de aquellas clases pertenecientes a otros packages. Por ejemplo:

```
 package Directivos;
class JefeDeAdministracion { //Ejemplo aprenderaprogramar.com
   private TiposBasicos.Persona jefeAdmin;
   . . .
}
```

Sin embargo si no se usa import es necesario especificar el nombre del package cada vez que se usa el tipo Persona, lo cual puede resultar un tanto engorroso. La cláusula import en esta acepción simplemente indica al compilador dónde debe buscar clases adicionales cuando no pueda encontrarlas en el package actual y delimita los espacios de nombres y modificadores de acceso. Sin embargo, no tiene la implicación de 'importar' o copiar código fuente u objeto alguno. **En una clase puede haber tantas sentencias import como sean necesarias.** Las cláusulas import se colocan después de la cláusula package (si es que existe) y antes de las definiciones de las clases.

FORMAS DE NOMBRAR PACKAGES. JERARQUIZACIÓN Y VISIBILIDAD DE CLASES. BLUEJ.

Los packages se pueden nombrar usando nombres compuestos separados por puntos, de forma similar a como se componen las llamadas a los métodos. Por ejemplo se puede tener un package de nombre tiposBasicos.directivos. Cuando se utiliza esta estructura se habla de packages y subpackages. En el ejemplo tiposBasicos es el Package base, directivos es un subpackage y a su vez podrían existir otros subpackages de directivos. De esta forma se pueden tener los packages ordenados según una jerarquía equivalente a un sistema de archivos jerárquico. El API de Java está estructurado de esta forma, con un primer calificador (java o javax) que indica la base, un segundo calificador (awt, util, swing, etc.) que indica el grupo funcional de clases y opcionalmente subpackages en un tercer nivel, dependiendo de la amplitud del grupo. Por ejemplo en java.util.ArrayList<E>, *java* sería el package básico, *util* un subpackge y *ArrayList* una clase. Cuando se crean packages de usuario no es recomendable usar nombres de packages que empiecen por java o javax para evitar confusiones con los propios del API de Java.

Además del significado lógico descrito hasta ahora, **los packages también tienen un significado físico** al servir para almacenar los ficheros con extensión .class en el sistema de archivos del ordenador. Supongamos que definimos una clase de nombre miClase que pertenece a un package de nombre *gestorDePersonal.tiposBasicos.directivos*. Cuando la JVM vaya a cargar en memoria miClase buscará el módulo ejecutable (de nombre miClase.class) en un directorio en la ruta de acceso gestorDePersonal/tiposBasicos/directivos (esta ruta de acceso estará definida a partir del directorio raíz para la JVM). Está ruta deberá existir y estar accesible a la JVM para que encuentre las clases.

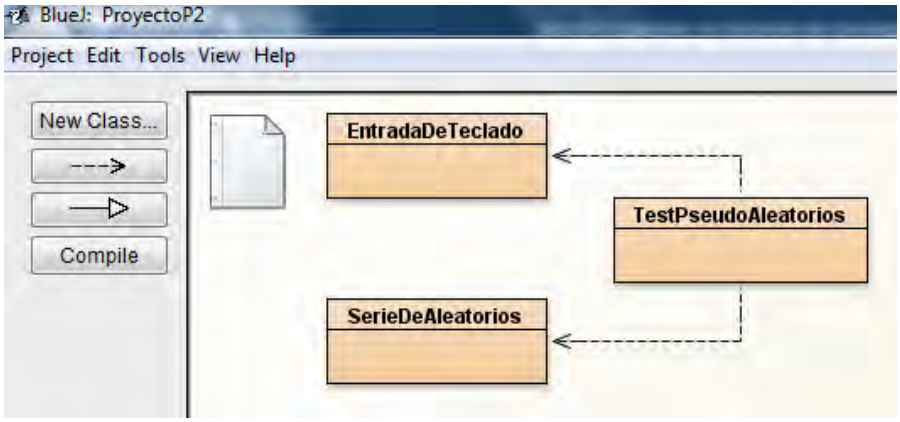
Si una clase no pertenece a ningún package (no existe cláusula *package*) se asume que pertenece a un package por defecto sin nombre, y la JVM buscará el archivo .class en el directorio que use como raíz.

Para que una clase pueda ser usada fuera del package donde se definió debe ser declarada con el modificador de acceso *public*. Por ejemplo:

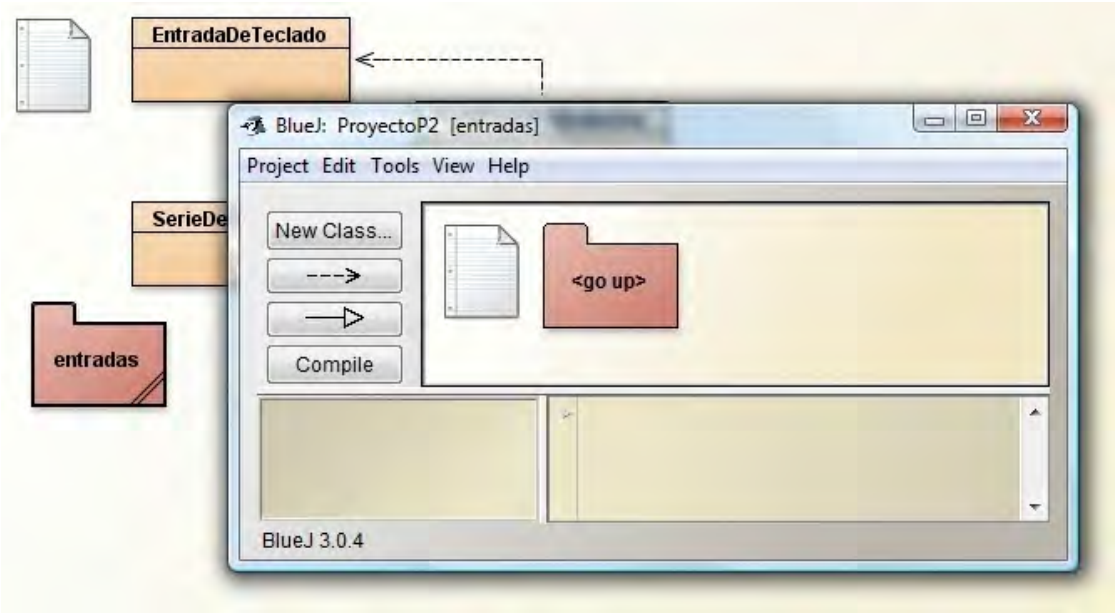
```
package Directivos;
public class JefeDeAdministracion {
                private Persona jefeAdmin;
        . . .
    }
```

Si una clase no se declara public sólo puede ser usada por clases que pertenezcan al mismo package.

Consideremos un proyecto en BlueJ. Hasta ahora lo hemos visto como algo así:



Aquí veíamos las clases y disponíamos del botón New Class… para ir añadiendo clases. ¿En qué package están estas clases? En el package por defecto (sin nombre), asociado al directorio raíz donde hayamos guardado el proyecto. Supongamos ahora que nuestra organización fuera que la clase EntradaPorTeclado perteneciera a un package denominado entradas.usuarios, es decir, que la clase perteneciera al subpaquete "usuarios" del paquete "entradas". Para crear los paquetes pulsamos en el menú Edit -> New package, y escribimos el nombre del package base, esto es, *entradas*. Nos aparece un icono "entradas" y haciendo doble click sobre él se nos abre la ventana del package creado:



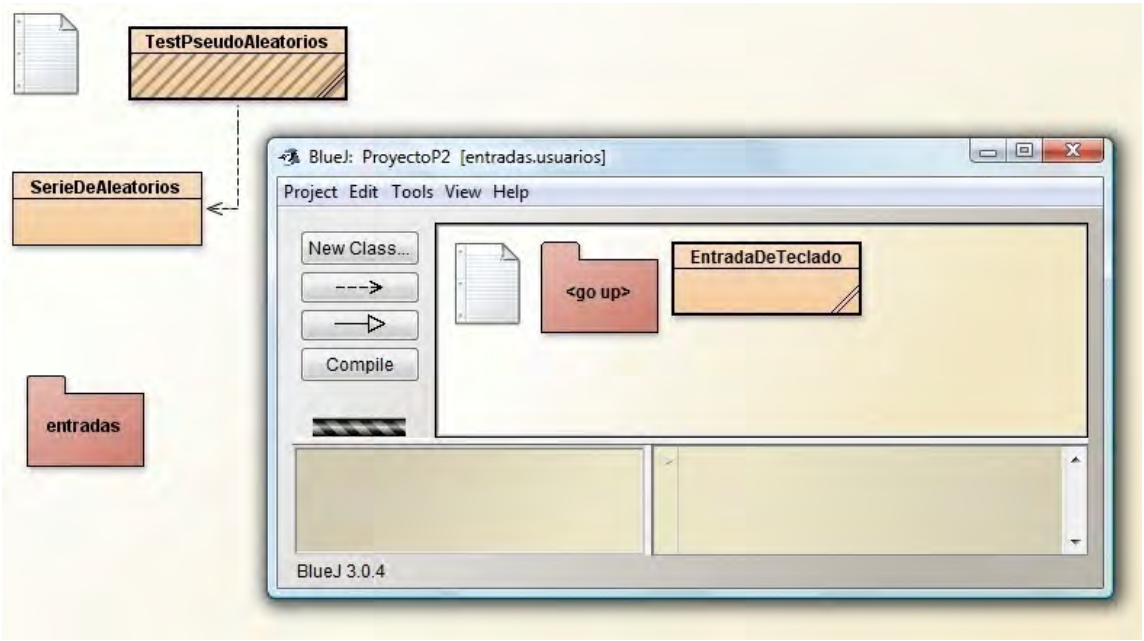
Donde el icono tipo carpeta "go up" nos sirve para subir de nivel en la jerarquía de packages. Situados en la ventana del package *entradas* repetiremos la operación creando el package *usuarios*. Ahora desde el package *entradas* podemos hacer dos cosas: subir de nivel hacia la raíz, o profundizar nivel hacia *usuarios*. Si comprobamos nuestro sistema de archivos, veremos cómo se han ido creando carpetas anidadas que de momento sólo contienen un archivo package.BlueJ. Además, si entramos en el package *usuarios* y vemos lo que nos indica la ventana de BlueJ, aparece [entradas.usuarios], es decir, por defecto nos está estableciendo la ruta de packages con notación de puntos (muy al estilo de Java). En el nivel raíz, borremos ahora el package *entradas*. Nos aparecerá una advertencia indicándonos que si borramos el package eliminaremos de forma permanente el directorio asociado al package incluyendo todos los contenidos de ese directorio (esto tiene su peligro pues podríamos borrar carpetas accidentalmente). Aceptemos y vayamos al sistema de archivos. Veremos cómo las carpetas correspondientes a packages y subpackages han desaparecido.

Existe una forma abreviada de definir estructuras de packages. Cuando empezamos un proyecto es posible que ya hayamos pensado cómo vamos a organizarlo. Podemos definir de forma directa una estructura usando, desde el nivel raíz, la creación de un package con la notación de puntos. BlueJ creará de forma automática las carpetas y subcarpetas necesarias. Por ejemplo, lo mismo que hicimos antes en dos pasos (primero crear el package *entradas* y luego crear el package *usuario*), podemos hacerlo dándole a Edit -> New package, y escribiendo la jerarquía o ruta de packages: en nuestro caso escribimos *entradas.usuarios*. De esta manera, BlueJ nos crea automáticamente los packages anidados y las carpetas y subcarpetas correspondientes.

Veamos un ejemplo de aplicación de lo expuesto hasta ahora. Para nuestro programa TestPseudoAleatorios (que lo teníamos funcionando correctamente en el directorio raíz) vamos a suponer que nos llevamos la clase EntradaDeTeclado a el package *entradas.usuarios*. Para ello creamos una clase de igual nombre dentro del package, copiamos el código de la clase que estaba en la raíz y luego eliminamos la clase de la raíz. Al crear la clase dentro del package automáticamente nos aparece una línea de cabecera: *package entradas.usuarios;*. Esta línea es la que **codifica a qué package pertenece la clase.** A continuación irá el código tal y como lo habíamos definido:

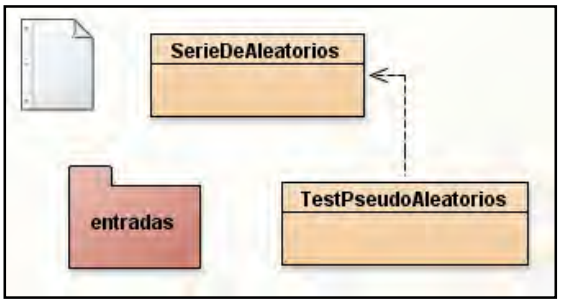
```
package entradas.usuarios; //Define a qué package pertenece la clase
import java.util.Scanner; //Importación de la clase Scanner desde la biblioteca Java
 // Ejemplo aprenderaprogramar.com
public class EntradaDeTeclado {  // Definimos la clase EntradaDeTeclado
   private String entradaTeclado; //Variable de instancia (campo) del método

   public EntradaDeTeclado ()  { //Constructor
      entradaTeclado=""; } //Cierre del constructor

   public void pedirEntrada ()  {  //Método de la clase
      Scanner entradaEscaner = new Scanner (System.in);
      entradaTeclado = entradaEscaner.nextLine ();
   } //Cierre del método pedirEntrada

   public String getEntrada () { return entradaTeclado; } //Cierre del método getEntrada
} //Cierre de la clase
```

La visualización de la estructura del proyecto es similar a esta:



Al haber eliminado una clase (que hemos puesto en un package) hay que compilar de nuevo. Y al compilar nos salta el error: *cannot find symbol – class entradaPorTeclado*. ¿Qué ocurre? Que el que la clase se encuentre en un package **afecta a su visibilidad.** En este caso la clase TestPseudoAleatorios no puede "ver" a la clase EntradaDeTeclado. Vamos a solucionarlo con una importación a la clase

TestPseudoAleatorios de la clase EntradaDeTeclado con esta sintaxis: *import entradas.usuarios.EntradaDeTeclado;*. También podríamos haber usado *import entradas.usuarios.*;*. En el esquema de clases de BlueJ vemos esto:



TestPseudoAleatorios no aparece conectado con una flecha de relación de uso con el package *entradas*. ¿Por qué? Porque la flecha se reserva para relaciones de uso directo de una clase a otra dentro de un mismo paquete. Cuando se trata de usos a través de *import* no nos aparece de forma explícita la representación de dicha relación.

Una última cuestión: ni los paquetes superiores reconocen las clases de los subpaquetes ni al revés: aquí, en principio, nadie conoce a nadie. Por tanto también hemos de importar las clases o paquetes incluso desde paquetes que están jerárquicamente en la misma línea y por encima de otros paquetes.

EJERCICIO

Crea un proyecto Java con la siguiente estructura:

a) Un package denominado formas dentro del cual existan los siguientes packages: formas1dimension, formas2dimensiones y formas3dimensiones.

b) Dentro del package formas1dimension deben existir las clases Recta y Curva.

c) Dentro del package formas2dimensiones deben existir las clases Triangulo, Cuadrilatero,Elipse, Parabola e Hiperbola.

d) Dentro del package formas3dimensiones deben existir las clases Cilindro, Cono y Esfera.

Nota: crea las clases sin rellenar su código, simplemente para poder visualizar que están dentro de los packages adecuados.

Visualiza el resultado y comprueba que las clases están agrupadas de forma adecuada.

Puedes comprobar si tu código es correcto consultando en los foros aprenderaprogramar.com.

Próxima entrega: CU00675B

Acceso al curso completo en aprenderaprogramar.com -- > Cursos, o en la dirección siguiente:

http://www.aprenderaprogramar.com/index.php?option=com_content&view=category&id=68&Itemid=188