

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

Paquetes y modificadores de acceso

© ADR Infor SL

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

viewnext.adrformacion.com © ADR Infor SL
VICTOR TENA PALOMARES

Indice

Competencias y Resultados de Aprendizaje desarrollados en esta unidad	3
Paquetes y modificadores de acceso	4
Objetivo	4
Paquetes	4
Características	4
Cómo utilizar paquetes	6
Visibilidad de los paquetes	8
Paquetes Estandar en Java	9
Paquete virtual	10
Creación de paquetes propios	12
Ejemplos de uso de paquetes	15
Ejecución de clases de paquetes externos	20
Modificadores de acceso	21
Tipos de modificadores	22
Ejemplos	28
Ejercicios	35
Ejercicio 1. Paquetes	35
Lo necesario para empezar	35
Recomendaciones	35
Datos a mostrar por consola	36
Ejercicio 2. Usar paquetes ya creados	37
Lo que se necesita para comenzar	37
Recomendaciones	38
Datos a mostrar por consola:	39
Recursos	40
Enlaces de Interés	40

Competencias y Resultados de Aprendizaje desarrollados en esta unidad

Competencia:

Entender el concepto de paquetes en Java y aprender su uso.

Resultados de Aprendizaje:

- Entender que son los paquetes en Java
- Aprender a manejar el concepto de paquete y a organizar las clases de un proyecto.
- Conocer los modificadores de acceso y su utilidad.
- Usar modificadores en aplicaciones Java.

Paquetes y modificadores de acceso

Objetivo

- Entender que son los paquetes en Java.
- Aprender a manejar el concepto de paquete y a organizar las clases de un proyecto.
- Conocer los modificadores de acceso y su utilidad.
- Usar modificadores en aplicaciones Java.

Paquetes

Todos **los recursos Java** contenidos en un proyecto están **identificados por su nombre**. Esto genera que varios recursos con el mismo nombre no puedan estar en el mismo proyecto.

Al crecer el número de clases crece la probabilidad de designar con el mismo nombre a dos clases diferentes.

Los paquetes resuelven el problema del conflicto entre los nombres de las clases y permiten organizar las clases en grupos dentro de un proyecto.

Vocabulario: Paquete

Un paquete en Java es un **contenedor de clases que permite agrupar** las distintas partes de un programa y que, por lo general, tienen funcionalidades con elementos comunes.

Un paquete también define la ubicación de dichas clases en un directorio de estructura jerárquica.

Un paquete **contiene un conjunto de clases relacionadas** bien por finalidad, por ámbito, por herencia o por funcionalidad.

Las clases tienen ciertos **privilegios de acceso** a los **miembros dato** y a las **funciones miembro** de otras clases **dentro de un mismo paquete**.

Características

Aplican un orden

Aplican un orden a las clases e interfaces que los componen. Esto permite buscar rápidamente las que interesen al programador.

Disminuyen el riesgo de conflicto de nombres

Disminuyen el riesgo de conflicto de nombres. Si dos clases tienen el mismo nombre, el hecho de que se encuentren en paquetes distintos evita conflictos, a no ser que se importen los dos paquetes en el mismo código.

Si ocurre esto último, puede resolverse añadiendo el **nombre cualificado** de la clase.

Escalabilidad

Un paquete puede contener subpaquetes y éstos, otros subsubpaquetes, y así sucesivamente de forma similar a la estructura de directorios de un disco duro.

Ámbito

Una clase pertenece a un paquete concreto y a su vez puede utilizar también clases contenidas en este o en otros paquetes.

Dos clases con el mismo nombre en dos paquetes distintos pueden coexistir e incluso pueden ser usadas en el mismo programa.

Nombre único

El **nombre** de una clase **debe ser único dentro del paquete** donde se define.

Los paquetes se pueden considerar como los directorios del proyecto, de hecho, **cada paquete genera un directorio** que almacenará el contenido del paquete.

Este concepto es similar a las librerías en otros lenguajes.

Nomenclatura

Los **nombres** que forman la estructura de un paquete suelen escribirse **en minúsculas**.

La separación entre paquetes a la hora de importarlos se realiza mediante el operador punto (.).

Hay muchas compañías de software que se dedican a **desarrollar comercialmente código estructurado en paquetes** que encaja con las necesidades de las empresas en múltiples sectores como la gestión de recursos humanos, la gestión de inventarios, de stocks, etc.

Cómo utilizar paquetes

Una clase se declara perteneciente a un paquete con la cláusula **package** incluida como una línea de código.

```
package nombrePackage;
```

La cláusula package debe ser **la primera sentencia del código fuente**. Por ejemplo, una clase puede comenzar así:

```
package cursojava;
```

```
public class Clase1 {
    .....
}
```

Este código implica que la clase **Clase1** pertenece al paquete **cursojava**.

La cláusula **package** es **opcional**. Si no se utiliza, las clases declaradas en el archivo fuente no pertenecen a ningún paquete concreto, sino que pertenecen a un paquete por defecto (**paquete virtual**).

La agrupación de clases en paquetes es aconsejable para mantener, bajo una ubicación común, clases relacionadas que cooperan desde algún punto de vista. Esto permite una mayor organización de los recursos del proyecto.

También resulta importante por la implicación que los paquetes 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 paquete.

package escuela:

```
public class Aula {
    public void mostrarAlumno(...){
        ....
        Alumno alumno = new Alumno();
    }
}
```

En este código definimos la clase Aula perteneciente al paquete escuela. En esta clase se usa la clase Alumno.

El compilador asume que Alumno pertenece también al paquete escuela y buscará esta clase dentro de este paquete.

Para que el compilador pueda encontrar la clase Alumno, esta debe crearse dentro del paquete escuela y para ello también contendrá como primera línea la sentencia:

```
package escuela;
```

Para poder utilizar clases que no forman parte del mismo paquete que la clase que estamos creando se debe utilizar una cláusula llamada **import**. Esta cláusula permitirá el acceso a clases contenidas en diferentes paquetes.

Suponemos que la clase Alumno tiene este código:

```
package personas;

public class Alumno{
    ....
    public void Alumno(){
        ....
    }
}
```

Ahora ya no podremos utilizar esta clase dentro de Aula como hacíamos antes ya que no pertenecen al mismo paquete. Para poder utilizarla necesitaremos hacer uso de la sentencia **import**:

```
package escuela;
import personas.Alumno;

public class Aula{
    Alumno alumno = new Alumno();
    ....
}
```

Las cláusulas import se colocan después de la cláusula package (si es que existe) y antes de las definiciones de la clase.

El carácter * permite importar en una sola sentencia todas las clases contenidas en un paquete. En nuestro ejemplo anterior podríamos sustituir la línea

```
import personas.Alumno;
```

por

```
import personas.*;
```

También podemos utilizar clases sin necesidad de utilizar la sentencia `import`, para ello debemos utilizar el **nombre cualificado** de la clase, esto es, añadir el nombre del paquete al nombre de la clase a utilizar:

```
package escuela;

public class Aula{
    personas.Alumno alumno = new personas.Alumno();
    ....
}
```

La combinación entre la cláusula `import` y el uso del nombre cualificado de la clase permite la coexistencia en un proyecto de clases con el mismo nombre y diferente paquete.

Para poder utilizar dos clases con el mismo nombre dentro de otra deberemos utilizar el siguiente código:

Suponemos que tenemos dos clases llamadas **Alumno** dentro de nuestro proyecto, una pertenece al paquete **jerarquía** y la otra al paquete **personas**. Para poder utilizar las dos en la clase `Aula` deberemos utilizar el siguiente código:

```
package escuela;
import personas.Alumno;

public class Aula{
    Alumno alumno = new Alumno(); /* Esta sentencia se refiere a la clase alumno del paquete personas*/
    jerarquia.Alumno alumno = new jerarquia.Alumno(); /* Esta sentencia se refiere a la clase alumno del paquete jerarquia*/
    ....
}
```

Visibilidad de los paquetes

Por defecto, el programador de una clase tiene acceso a las clases del paquete virtual y a las del paquete `java.lang` de la API. Para acceder a clases de cualquier otro paquete existen dos opciones:

El programador puede referirse a ellas escribiendo toda la ruta del paquete al que pertenecen.

```
java.io.InputStreamReader isr = new java.io.InputStreamReader(System.in);
java.io.BufferedReader br = new java.io.BufferedReader(isr);
```

También se puede importar las clases que se van a utilizar en el programa utilizando la palabra reservada **import** seguida del **nombre del paquete** que quiere importar. Además, la línea que contiene al `import` debe escribirse en primer lugar y antes de la declaración de la clase, aunque después de la declaración del paquete externo al que se desea que pertenezca la clase.


```
import java.io.InputStreamReader;
import java.io.BufferedReader;
import java.io.IOException;
.....
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
```

Lo que mas se emplea dentro de esta segunda opción es el asterisco (*), que **permite acceder a todas las clases públicas de primer nivel del paquete** haciendo uso únicamente de aquéllas que se van a utilizar en el código. Esto ralentiza un poco la compilación, aunque suele ser la opción más empleada:

```
import java.io.*;
.....
InputStreamReader isr = new InputStreamReader(System.in);
BufferedReader br = new BufferedReader(isr);
```

si se quiere instanciar la clase `ActionEvent` que pertenece a `java.awt.event`, no sería correcto hacer esto:

```
import java.awt.*;
...
ActionEvent ae = new ActionEvent(..);
```

sino que debería importarse `java.awt.event`. Es decir,

```
import java.awt.event.*;
```

`java.awt.event` representa el paquete `java` que contiene al subpaquete `awt` que contiene al subpaquete `event`.

Paquetes Estandar en Java

La estructura de Java contiene una serie de paquetes ya creados que aportan funcionalidad básica al lenguaje. Estos paquetes están organizados y permiten el uso de clases creadas anteriormente.

Estos son algunos de los paquetes más importantes:

Paquete	Funcionalidad
<code>java.applet</code>	Contiene las clases necesarias para crear applets. Se ejecutan en la ventana del navegador.
<code>java.awt</code>	Contiene clases que permiten crear aplicaciones con GUI independientes de la plataforma.
<code>java.io</code>	Contiene clases que definen distintos flujos de datos de Entrada/Salida
<code>java.lang</code>	Contiene clases esenciales y se importa sin la necesidad de la sentencia <code>import</code> .
<code>java.net</code>	Se usa en combinación con las clases del paquete <code>java.io</code> para leer y escribir datos en la red.
<code>java.util</code>	Contiene otras clases útiles que ayudan a realizar funciones básicas.

Paquete virtual

Toda clase que, explícitamente no forme parte de ningún paquete, pertenece, por defecto, al paquete virtual.

El paquete virtual está asociado al directorio donde se guarda el código fuente.

Los métodos y variables de instancia de clases del paquete virtual son accesibles desde otras clases siempre y cuando éstas se encuentren en el mismo paquete virtual, es decir, siempre y cuando se guarden en el mismo directorio.

Partiremos del siguiente código:

```
public class Clase2{
    String nombre="Jesus";
    void mostrarNombre(){
        System.out.println("Nombre= " + nombre);
    }
}
```

Pasos a seguir si se trabaja con Eclipse:

- En el menú Build Path de las propiedades del proyecto, pestaña Source, se debe crear un nuevo Link Source que apunte a la carpeta 2 creada en la ruta C:/.
- Agregar al proyecto tema7 un fichero java de nombre Clase2, pero antes de aceptar, pulsar el botón de Examinar e indicarle la nueva ruta añadida.
- En esta misma pantalla, activar el check que permite añadir output folders a la nueva ruta creada. Aparecerá un nuevo campo bajo la ruta C:/2.
- En este nuevo campo hay que añadir la propia ruta C:/2
- Crear la nueva clase y añadirla en la nueva ruta creada.
- A continuación, compilar. Se observará que Clase2.class aparece, junto con Clase2.java, en c:\2

Si no se emplea IDE se guardará y compilará en c:\2

La clase **Clase2** pertenece al paquete virtual asociado al directorio c:\2 .

Después escribimos el siguiente código:

```
public class Clase1 {
    public static void main(String args[]){
        Clase2 c2=new Clase2();
        System.out.println(c2.nombre);
        c2.mostrarNombre();
        System.out.println("FIN DE PROGRAMA");
    }
}
```

Pasos a seguir si se trabaja con Eclipse:

- Agregar al proyecto tema7 un fichero java de nombre Clase1, pero antes de aceptar, pulsar el botón de Examinar e indicarle que se alojará en la ruta **c:\1**.
- Teclar el código de Clase1.java y, antes de compilar, botón derecho sobre el proyecto y seleccionar **Configure Build Path** y modificar el **Output Path** con el valor **c:\1**.
- A continuación, compilar. Se observará que **Clase1.class aparece, junto con Clase1.java, en c:\1**

Si no se emplea IDE se guardará y compilará en **c:\1**.

La clase **Clase1** pertenece al paquete virtual asociado al directorio **c:\1**

A continuación, se intenta compilar Clase1.java.

Se produce **error de compilación** debido a que el compilador no es capaz de encontrar la clase Clase2 pues se encuentra en un paquete distinto del que forma parte Clase1.

¿Cómo se soluciona?

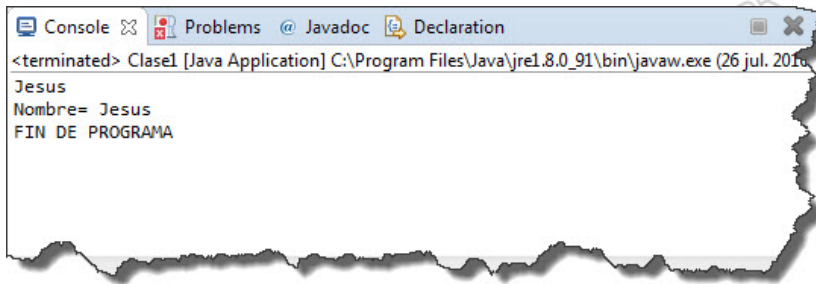
Ambos códigos deben guardarse o en **c:\1** ó en **c:\2**. De este modo, pertenecerán al mismo paquete virtual.

Hecho esto, se compilarán sin ningún problema.

Para ejecutar sin ide se debe teclear los siguiente en la línea de comandos:

```
java Clase1
```

El resultado que debe aparecer por consola es:



Creación de paquetes propios

Para crear nuestros propios paquetes debemos dar los siguientes pasos:

Escoger nombres

El primer paso es escoger los nombres que tendrá la estructura del paquete.

Oracle recomienda usar el nombre del dominio de internet como nombre de la estructura de paquetes invirtiendo sus elementos. Con eso se evita el conflicto entre nombres ya que no puede haber dos nombres de dominio iguales.

Así, si mi dominio es **www.javarioja.com**, la estructura de mi paquete será **com.javarioja**.

Si no se dispone de dominio propio suele usarse nombres que representen nuestra estructura..

En el curso se podría emplear como nombre del paquete principal el nombre del curso:

si el curso se llama Java y lo imparte adr, la estructura de paquete será:

adr.cursojava

donde *adr* es el **paquete principal** y *cursojava* un **subpaquete**

Estructura de directorios

Crear una estructura de directorios idéntica a la estructura del paquete, que cuelgue de cualquier directorio del disco duro. Los IDEs se encargan de crear esta estructura de directorios de forma transparente para el programador.

Al directorio del que cuelga el paquete se le denomina **directorio raíz** de la estructura del paquete (directorio del que cuelga la estructura de directorios del paquete)

En los ejemplos del tema, se utilizará `c:\ejemplo`.

`c:\adr` = directorio raíz de la estructura de directorios del paquete **adr**.

`adr\cursojava` = estructura de directorios del paquete

El directorio `cursojava` es el que debe contener los ficheros **.class** de las clases e interfaces que va a almacenar el paquete.

Configurar el classpath

El siguiente paso se trata de configurar adecuadamente la **variable de entorno classpath** para que el compilador pueda localizar las clases del paquete y así poder ser utilizadas por códigos de otros paquetes previa importación del mismo.

Hay dos posibilidades:

1. Trabajar sin un Editor de textos (IDE)

En Windows se va a **Panel de control/Rendimiento y mantenimiento/Sistema/Opciones avanzadas/Botón Variables de entorno**.

Si no se encuentra definida la variable de entorno `classpath`, se crea mediante el botón **Nueva** de la parte superior.

Su valor será la ruta punto (.) que indica directorio actual (la razón se explica en la nota sobre la variable de entorno `classpath`) y la ruta del directorio raíz de la estructura del paquete.

Las rutas se separarán mediante punto y coma.

No importan las mayúsculas a la hora de escribir las ruta.

Si la `classpath` ya está definida, se agregará la nueva ruta a continuación de la última existente.

En otros sistemas operativos preguntar al tutor.

2. Trabajar con un IDE

Si se trabaja con Eclipse se deben seguir estos pasos:

Pinchar sobre el proyecto con el botón derecho y seleccionar la opción **Properties**.

En la ventana que aparece seleccionamos **Java Build Path**.

Seleccionamos la pestaña **Libraries** y añadimos el directorio raíz de la estructura del paquete.

Al aceptar se observará una nueva ruta en la `classpath`. Esta ruta permitirá a otros códigos ubicados en otros paquetes, utilizar las clases del paquete asociado al directorio raíz agregado a la `classpath`, siempre y cuando se importe dicho paquete.

Si se tiene el paquete `adr.cursojava` con su estructura de directorios asociada y se desea que el directorio raíz sea `c:\ejemplo`, la variable `classpath` la podemos establecer de esta manera:

`set classpath=.;c:\ejemplo`

Si se emplea Eclipse, se agregaría la ruta **`c:\ejemplo`** a su `classpath`.

Se crea un paquete de nombre **`utilmatematicas`** sin ningún subpaquete. Como directorio **raíz** se va a utilizar **`c:\matematicas\calculos`**. ¿Cómo debe configurarse la variable `classpath` para poder utilizar las clases del paquete?

`set classpath=.;c:\matematicas\calculos`

Si se emplea Eclipse, sería igual que antes, pero agregando la ruta **`c:\matematicas\calculos`**

Normalment se configuran varias rutas para tener la posibilidad de crear varios paquetes con diferentes directorios raíz:

`set classpath=.;<ruta 1>;<ruta 2>; ... <ruta n>;`

Declarar el paquete

Para declarar el paquete siempre se hace en la primera línea del código de la clase a crear, antes de cualquier `import` y declaración de clase.

```
package <nombre paquete>;
import java.util.*;
....
public class Prueba {
.....
}
}
```



Crear paquetes

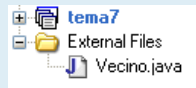
Ejemplos de uso de paquetes

Primero se creará la estructura de directorios asociada al paquete `adr.cursojava`. Como el directorio raíz del que va a colgar esta estructura de paquete va a ser `c:\ejemplo` (porque así lo decide el programador, aunque podría haber sido otro distinto), se creará dicha estructura colgando del mismo.

```
package adr.cursojava;

public class Vecino {
    String nVecino="German";
    public void saludo(){
        System.out.println("Clase Vecino1: paquete adr.cursojava");
        System.out.println("Me llamo " + nVecino);
    }
}
```

Se observará que se considera un External File. Es lo que debe ocurrir.



Pasos a seguir si se trabaja con Eclipse (con otro IDE o sin IDE consultar al tutor):

- Agregar al proyecto `tema7` un fichero java de nombre `Vecino`, pero antes de aceptar, pulsar el botón "Browse..." e indicarle `c:\ejemplo\adr\formación`.
- Teclar el código de `Vecino.java` y, antes de compilar, menú `Configure Build Path` y seleccionar como `Output Path` `c:\ejemplo`.
- A continuación, compilar.

Se observará que `Vecino.class` aparece, junto con `Vecino.java`, en `c:\ejemplo\adr\cursojava`

El siguiente código importará el paquete y mediante un objeto de la clase `Vecino` accederá a la variable de instancia `nVecino` y al método `saludo()` de la clase `Vecino` siempre que éstos sean públicos.

También debe ser pública la clase en la que se encuentran.

```

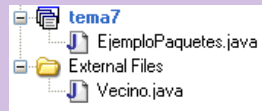
package adr.cursojava;

public class EjemploPaquetes {
    public static void main(String args[]){
        System.out.println("Clase EjemploPaquetes: paquete virtual");
        System.out.println("Ahora vas a acceder al contenido de la clase Vecino3 \ndel paquete virtual.");
        Vecino v=new Vecino();
        System.out.println("-----");
        v.saludo();
        System.out.println("Me llamo " + v.nVecino);
        System.out.println("FIN DE PROGRAMA");
    }
}

```

Los pasos, si se emplea IDE, ahora son:

- Agregar al proyecto tema7, el fichero java de nombre EjemploPaquetes.java. El valor de Location será eclipse_home\MyProjects\tema7. Gráficamente:

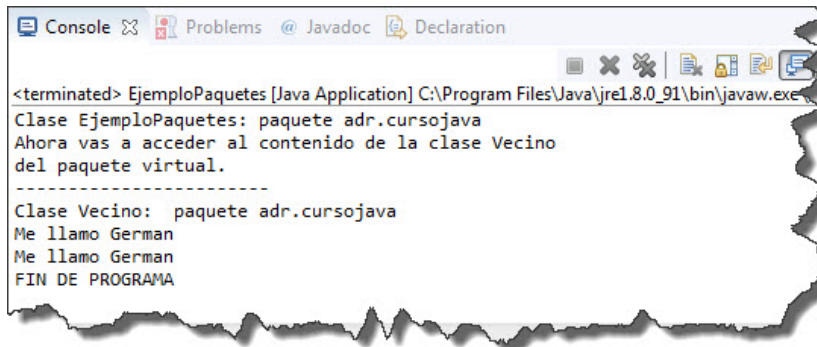


- Teclar el código de EjemploPaquetes.java y, antes de compilar, botón derecho sobre el proyecto y seleccionar Configure Build Path y modificar el Output Path con el valor eclipse_home\MyProjects\tema7.
- A continuación, compilar.

Todo irá bien si se ha agregado la ruta c:\ejemplo a la classpath del IDE o ".;c:\ejemplo" a la classpath del sistema, en caso de no emplear IDE.

Si no se ha hecho esto se produce un error al compilar ya que el compilador es incapaz de encontrar Vecino.class

Por consola:



```

<terminated> EjemploPaquetes [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Clase EjemploPaquetes: paquete adr.cursojava
Ahora vas a acceder al contenido de la clase Vecino
del paquete virtual.
-----
Clase Vecino: paquete adr.cursojava
Me llamo German
Me llamo German
FIN DE PROGRAMA

```

¿Como busca el compilador la clase Vecino para que la línea Vecino v = new Vecino(); tenga sentido?

1. Lo primero que hace es buscar la clase Vecino en el paquete java.lang y en el paquete virtual.
 1. Si no la encuentra, se analiza el primer import del código para comprobar si la clase se encuentra en el paquete que el import especifica;
 2. Si no se encuentra en el primer import continúa con el resto de import y si no se encuentra en ninguno se produce un error de compilación.
2. La variable de entorno classpath está relacionada con la búsqueda de los paquetes externos especificados en las sentencias import. Así, si la variable classpath esta configurada del siguiente modo:


```
set classpath=.;c:\ejemplo
```

 se le está diciendo al compilador que busque en el directorio actual del DOS (asociado al punto) un directorio llamado adr y dentro de él otro directorio llamado cursojava.
3. Si no se encuentra **adr** colgando del directorio actual (es lo que ocurre en este caso), se intenta con la segunda ruta del classpath, es decir, **se busca adr dentro del directorio c:\ejemplo.**
4. Si el compilador lo encuentra trata de encontrar en su interior cursojava;
5. En este caso lo encuentra, localiza la clase Vecino y, por fin, acaba el proceso de compilación correctamente.

Los paquetes de la API asociados a sentencias import se encuentran gracias a la configuración de la variable del sistema path durante el proceso de instalación del J2SE.

¿Qué ocurriría si se tuvieran dos clases de nombre coincidente en dos paquetes distintos y se importaran los dos paquetes?

Código de la clase cucu.OtroVecino:

```

package cucu;
public class OtroVecino{
    public String nVecino1 = "Pepe";
}

```

Si trabajamos con Eclipse:

- Agregar al proyecto tema7 un fichero java de nombre OtroVecino, pero antes de aceptar, pulsar el botón de los puntitos de "Location" e indicarle c:/ejemplo/cucu (deberá crearse previamente el directorio cucu colgando de c:\ejemplo)
- Teclear el código de OtroVecino.java y, antes de compilar, menú Project/Project Properties y seleccionar como Output Path "c:/ejemplo". A continuación, compilar. Se observará que OtroVecino.class aparece, junto con OtroVecino.java, en c:/ejemplo/cucu

Sin IDE, se guarda y compila en c:\ejemplo\cucu

Código de la clase adr.java.OtroVecino:

```
package adr.cursojava;  
  
public class OtroVecino {  
    public String nVecino2 = "German";  
}
```

Si trabajamos con Eclipse:

- Agregar al proyecto tema7 un fichero java de nombre OtroVecino, pero antes de aceptar, pulsar el botón de los puntitos de "Location" e indicarle c:\ejemplo\adr\cursojava, en mi caso.
- Teclear el código de Vecino.java y, antes de compilar, menú Project/Project Properties y seleccionar como Output Path c:\ejemplo. A continuación, compilar. Se observará que OtroVecino.class aparece, junto con OtroVecino.java, en c:\ejemplo\adr\cursojava.

Sin IDE, se guarda y compila en c:\ejemplo\adr\cursojava

Código de la clase que importa ambos paquetes: esta clase pretende mostrar por consola los nombres de los dos vecinos:

```

import adr.cursojava.*;
import cucu.*;

public class EjemploPaquetesVarios {
    public static void main(String args[]) {
        System.out.println("Clase EjemploPaquetesVarios: paquete virtual");
        OtroVecino ov=new OtroVecino();
        System.out.println("-----");
        System.out.println("Vecino paq cucu:"+ov.nVecino1);
        System.out.println("Vecino paq adr.cursojava:"+ov.nVecino2);
        System.out.println("-----");
    }
}

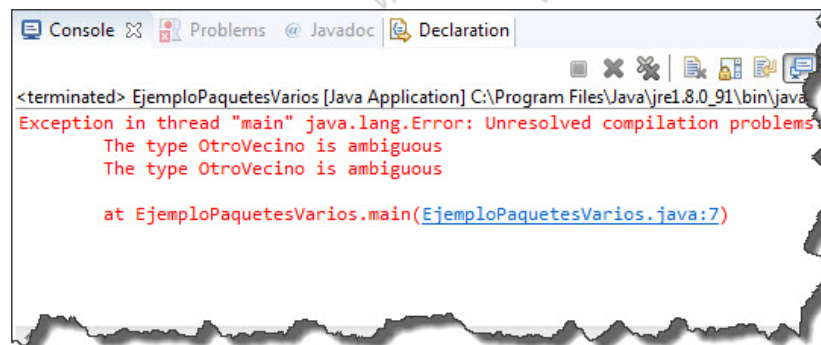
```

Se guarda y compila en c:\cursojava\tema7 o, si se emplea Eclipse, se asigna a Location eclipse_home\MyProjects\tema7.

Antes de compilar, botón derecho sobre el proyecto y seleccionar **Properties** y modificar el **Output Path** con el valor eclipse_home\MyProjects\tema7. Luego, intentar compilar ...

¿qué ocurre?

Se producen errores de compilación

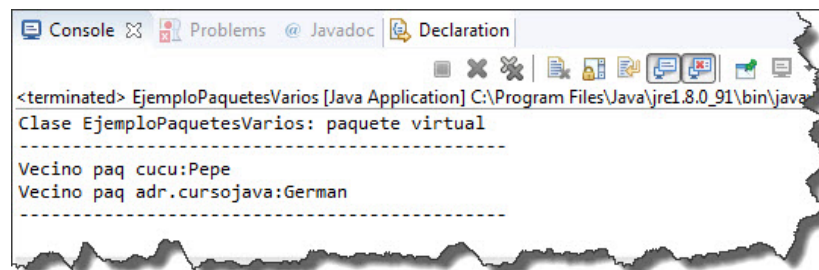


Una posible solución para resolver esta ambigüedad:

```
import adr.cursojava.*;

public class EjemploPaquetesVarios {
    public static void main(String args[]){
        System.out.println("Clase EjemploPaquetesVarios: paquete virtual");
        cucu.OtroVecino ov1 = new cucu.OtroVecino();
        OtroVecino ov2=new OtroVecino();
        System.out.println("-----");
        System.out.println("Vecino paq cucu:"+ov1.nVecino1);
        System.out.println("Vecino paq adr.cursojava:"+ov2.nVecino2);
        System.out.println("-----");
    }
}
```

Por consola:



Ejecución de clases de paquetes externos

Para ejecutar un programa Java cuya estructura de clases forma parte de un paquete externo, debe especificarse el nombre de la clase principal precedida del nombre del paquete.

```
package adr.cursojava;

public class PaqueteEjecutable{
    public String saludo="cucu";
    public void saludo(){
        System.out.println("Ejecutar codigo paq adr.cursojava");
    }
    public static void main(String args[]){
        System.out.println("Cucu");
        PaqueteEjecutable pe=new PaqueteEjecutable();
        pe.saludo();
        System.out.println("Me llamo "+pe.saludo);
    }
}
```

Si trabajamos con Eclipse:

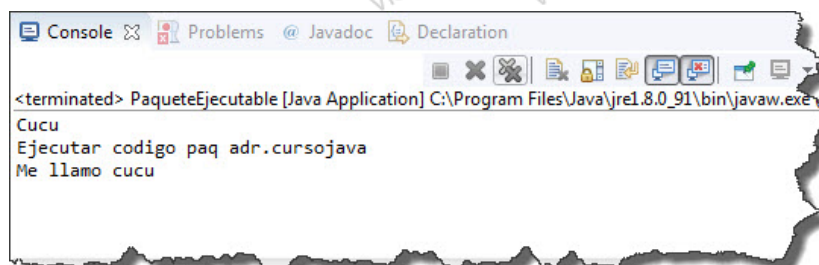
- Agregar al proyecto tema7 un fichero java de nombre PaqueteEjecutable, pero antes de aceptar, pulsar el botón de Examinar e indicarle `c:\ejemplo\jadr\cursojava`, en mi caso.
- Teclear el código de `PaqueteEjecutable.java` y, antes de compilar, botón derecho sobre el proyecto y seleccionar Properties y modificar el Output Path con el valor `"c:\ejemplo"`. A continuación, compilar. Se observará que `PaqueteEjecutable.class` aparece, junto con `PaqueteEjecutable.java`, en `c:\ejemplo\jadr\cursojava`
- Luego, se ejecuta.

Sin IDE, el código fuente se guardará y compilará en `c:\ejemplo\jadr\cursojava`.

Para ejecutar hay que poner la línea:

```
java adr.cursojava.PaqueteEjecutable
```

Por consola:



Modificadores de acceso

Son palabras clave de Java que fijan el grado de accesibilidad de clases, interfaces, métodos y variables.

La siguiente tabla muestra los principales modificadores y su ámbito de actuación:

Modificador	Clases o interfaces	Métodos	Variables
public	X	X	X
private		X	X
protected	X	X	X
final	X	X	X
sin modificador	X	X	X
private protected		X	X
abstract	X	X	
static		X	X
synchronized		X	
volatile			X
native		X	
transient			X

Como las variables de instancia y los métodos están dentro de clases, cuando no se tiene acceso a una clase tampoco se tendrá acceso a sus variables de instancia y métodos, independientemente del modificador de acceso que se les aplique.

Tipos de modificadores

Modificador public

Si se define una clase, interface, método o variable con el modificador "public" se **permite el acceso a dichos elementos desde todas las clases**.

Es el nivel de protección menos restrictivo. MUY EMPLEADO.

Sin modificador

Si se define una clase, interface, método o variable sin modificador ninguno se **permite el acceso a los elementos anteriores desde las clases del mismo paquete**.

Modificador protected

Si se define una clase, interface, método o variable con el modificador "protected" se **permite el acceso desde sus subclases y desde todas las clases del mismo paquete**.

Modificador private

Si se define una clase, interface, método o variable con el modificador "private" se permite el acceso a los elementos anteriores desde los miembros internos de la clase.

Es el nivel de protección más estricto.

Los métodos y variables privados de una superclase no son accesibles desde su subclase.

Este modificador es la base de un concepto importante en la programación orientada a objetos como es la encapsulación.

Básicamente se trata de declarar las variables de instancia de una clase como privadas y permitir el acceso a las mismas únicamente desde métodos setters y getters de la clase. Se verán ejemplos más adelante. MUY EMPLEADO.

Con el modificador "private protected" se permite el acceso a los elementos anteriores desde las subclases de la clase donde se han definido dichos métodos y variables. POCO USADO.

Modificador final

Si se define una clase con el modificador "final" indica que dicha **clase no puede heredarse, no admite subclases**.

Si se define un método con el modificador "final" indicará que **el método no puede redefinirse en una subclase**.

Si se define una variable con el modificador "final" indicará que **la variable no puede modificarse en tiempo de ejecución**. Es una constante.

Modificador transient

El modificador "transient" es aplicable sólo a variables.

Cuando se define una variable con el modificador "transient" indica que **la variable no se tendrá en cuenta a la hora de serializar la clase** donde se encuentra definida. POCO USADO.

Modificador volatile

El modificador "volatile" es aplicable cuando el valor de la variable será modificado por distintas Hilos. Cuando se define una variable con el modificador "volatile" indica que **al cambiar el valor de la variable, desde los hilos, se leerá siempre el valor más reciente**. MUY POCO USADO.

Modificador native

El modificador "native" es aplicable sólo a métodos.

Cuando se define un método con el modificador "native" indica que **el método se ha escrito en un lenguaje de programación distinto de Java** como C, C++, ensamblador, etc. POCO USADO.

Modificador synchronized

El modificador "synchronized" es aplicable sólo a métodos.

Cuando se define un método con el modificador "synchronized" **impide que varios hilos ejecuten el mismo método simultáneamente**.

Se usa sobre todo cuando se trabaja con la clase Thread..

Modificador static

Un atributo de una clase se puede modificar con el modificador static para indicar que **este atributo no pertenece a las instancias de la clase si no a la propia clase.**

Esto quiere decir que si tenemos varias instancias de una misma clase, cada una de ellas no tendrán una copia propia de ese atributo, si no que todas estas instancias compartirán una misma copia del atributo. A los atributos definidos como static, se les suele llamar **atributos de la clase.**

De la misma manera podemos utilizar la palabra static en la definición de los métodos haciendo que estos sean **métodos de la clase**, y de la misma manera que con los atributos, podemos llamar a estos métodos mediante el nombre de la clase.

Debemos crear estos métodos con unas condiciones, **estos métodos estáticos solo pueden acceder a atributos estáticos o llamar a otros métodos estáticos.**

Modificador abstract

Si se define una clase con el modificador "abstract" indica que dicha clase no puede instanciarse. Las clases abstractas suelen tener métodos abstractos, aunque no es obligatorio. Admiten también métodos no abstractos.

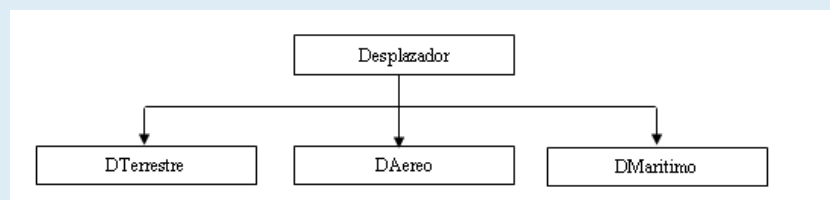
Los métodos abstractos se caracterizan por:

- No tener código, al igual que los que declaran las interfaces
- Admitirse sólo en clases abstractas.
- La utilización que hacen de ellos las subclases de la clase abstracta llenándolos del código adecuado en función del comportamiento que se necesite para la subclase.

En la programación orientada a objetos, las clases abstractas sirven para definir un comportamiento muy general, mediante métodos abstractos sin código, de algo que pretende modelarse.

Será a través de subclases donde se definan comportamientos más específicos, implementando los métodos abstractos sin código de la superclase como interese.

Sea el siguiente diagrama de clases que sirve para modelar desplazamientos a través de diversos medios.



Clase Desplazador: clase abstracta que modela el desplazamiento de los seres humanos de un lugar a otro.

Contiene dos métodos abstractos sin código y un método con modificador protected:


```

abstract void aumentarVelocidad();
abstract void ponerEnMarcha();
protected void ponerNombre(String nombre){...};

```

Los dos métodos abstractos se llenarán de código o implementarán en las tres subclases adecuadamente.

El método `protected` se invocará desde los constructores de las subclases.

Clase DTerrestre: subclase de Desplazador que implementa el método **aumentarVelocidad()** presionando un pedal y el método **ponerEnMarcha()** girando llave de contacto.

Clase DAereo: subclase de Desplazador que implementa el método **aumentarVelocidad()** modificando ángulo de ataque y el método **ponerEnMarcha()** pulsando botón.

Clase DMarítimo: subclase de Desplazador que implementa el método **aumentarVelocidad()** desplegando velas y el método **ponerEnMarcha()** izando velas.

Esta tabla muestra la implementación que hace cada subclase de los métodos abstractos heredados de la clase abstracta:

Métodos heredados	Subclases de Desplazador		
	DTerrestre	DAereo	DMarítimo
void aumentarVelocidad()	presionando pedal	modificando ángulo de ataque	desplegando velas
void ponerEnMarcha()	girando llave de contacto	pulsando botón	izando velas

Cada clase va a estar en un código fuente distinto.

```

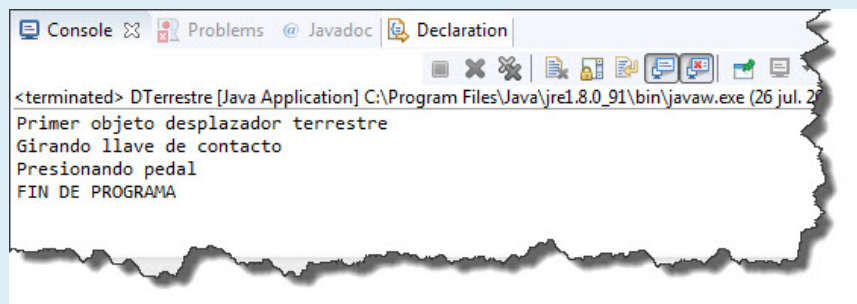
public abstract class Desplazador{
    abstract void aumentarVelocidad();
    abstract void ponerEnMarcha();
    protected void ponerNombre(String nombre){
        System.out.println(nombre);
    }
}

```

```

public class DTerrestre extends Desplazador{
    DTerrestre(String nom){
        ponerNombre(nom);
        ponerEnMarcha();
        aumentarVelocidad();
    }
    void aumentarVelocidad(){
        System.out.println("Presionando pedal");
    }
    void ponerEnMarcha(){
        System.out.println("Girando llave de contacto");
    }
    public static void main(String args[]){
        DTerrestre dt=new DTerrestre("Primer objeto desplazador terrestre");
        System.out.println("FIN DE PROGRAMA");
    }
}

```

Por consola:

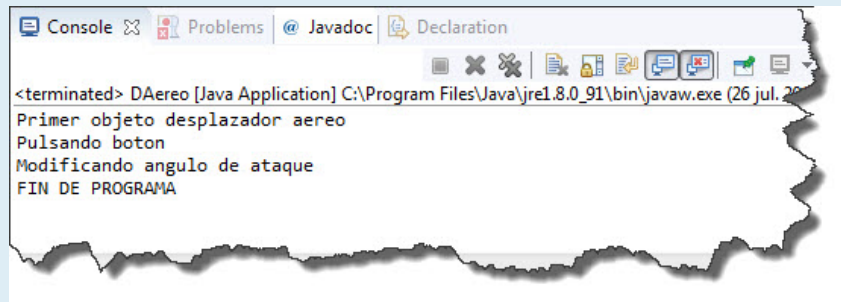
Si usamos el siguiente código:

```

public class DAereo extends Desplazador{
    DAereo(String nom){
        ponerNombre(nom);
        ponerEnMarcha();
        aumentarVelocidad();
    }
    void aumentarVelocidad(){
        System.out.println("Modificando angulo de ataque");
    }
    void ponerEnMarcha(){
        System.out.println("Pulsando boton");
    }
    public static void main(String args[]){
        DAereo da=new DAereo("Primer objeto desplazador aereo");
        System.out.println("FIN DE PROGRAMA");
    }
}

```

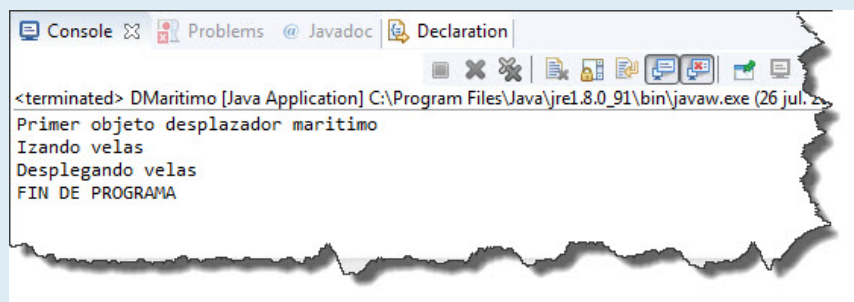
Por consola:



Y por último:

```
public class DMaritimo extends Desplazador{
    DMaritimo(String nom){
        ponerNombre(nom);
        ponerEnMarcha();
        aumentarVelocidad();
    }
    void aumentarVelocidad(){
        System.out.println("Desplegando velas");
    }
    void ponerEnMarcha(){
        System.out.println("Izando velas");
    }
    public static void main(String args[]){
        DMaritimo dm=new DMaritimo("Primer objeto desplazador maritimo");
        System.out.println("FIN DE PROGRAMA");
    }
}
```

Por consola:



En la tabla adjunta se muestran los modificadores de acceso más empleados a nivel de métodos y variables contenidos en clases:

ACCESO	public	protected	sin modificador	private
Desde la misma clase	SI	SI	SI	SI
Desde cualquier clase en el mismo paquete	SI	SI	SI	NO
Desde cualquier clase en distinto paquete	SI	NO	NO	NO
Desde una subclase en el mismo paquete	SI	SI	SI	NO
Desde una subclase en distinto paquete	SI	SI	NO	NO

Ejemplos

Se muestran ahora una serie de ejemplos que permiten comprobar la tabla anterior:

Se intenta acceder a una variable de instancia y a un método en el interior de una clase pública almacenada en un paquete (jesusfernandez.trilcejf), desde otra que está en otro paquete (el virtual en este caso).

Las clases del ejemplo ya se han visto en la sección de paquetes.

Pasos a seguir si se trabaja con un IDE:

Agregar al proyecto tema7 un fichero java de nombre Vecino1, pero antes de aceptar, pulsar el botón de los puntitos de "Location" e indicarle c:\ejemplo\adr\cursojava, en mi caso.

Teclear el código de Vecino1.java y, antes de compilar, menú Project/Project Properties y seleccionar como Output Path c:\ejemplo. A continuación, compilar. Se observará que Vecino1.class aparece, junto con Vecino1.java, en c:\ejemplo\adr\cursojava.

Si no se emplea IDE se guardará y compilará en c:\ejemplo\adr\cursojava

```
package adr.cursojava;

public class Vecino1 {
    String nVecino="German";
    public void saludo(){
        System.out.println("Clase Vecino1: paquete jesusfernandez.trilcejf");
        System.out.println("Me llamo "+nVecino);
    }
}
```

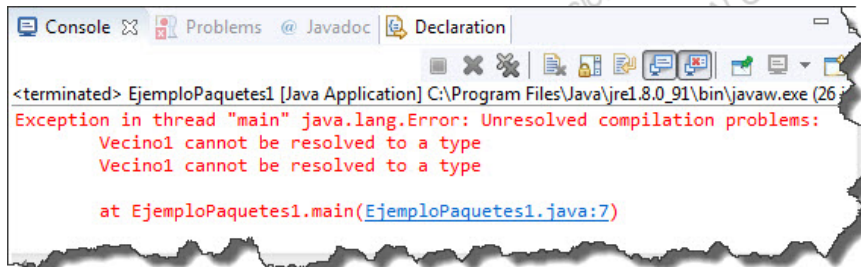
Esta clase compila correctamente sin ningún problema.

Ahora vamos a crear la clase que intentará acceder a Vecino1. Pertenece al paquete virtual (no tiene paquete definido).

```
import jesusfernandez.trilcejf.*;

public class EjemploPaquetes1 {
    public static void main(String args[]) {
        System.out.println("Clase EjemploPaquetes1: paquete virtual");
        System.out.println("Ahora vas a acceder al contenido de la clase Vecino1 \ndel paquete jesusferna
ndeaz.trilcejf.");
        Vecino1 v=new Vecino1();
        System.out.println("-----");
        v.saludo();
        System.out.println("Me llamo "+v.nVecino);
        System.out.println("FIN DE PROGRAMA");
    }
}
```

Al tratar de compilar esta clase se produce un error de compilación debido a que la variable nVecino no es public.



Se intenta acceder a una variable de instancia y un método en el interior de una clase sin modificador de acceso almacenada en un paquete, desde otra que está en otro paquete. Con IDE, se seguirán los mismos pasos que en Vecino1

Si no se emplea IDE se guardará y compilará en c:\ejemplo\adr\cursojava

```
package adr.cursojava;

class Vecino2 {
    public String nVecino="German";
    public void saludo() {
        System.out.println("Clase Vecino2: paquete jesusfernandez.trilcejf");
        System.out.println("Me llamo "+nVecino);
    }
}
```

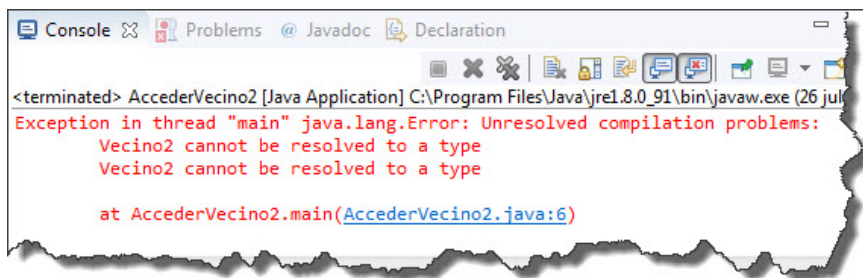
Código de la clase desde la que se intenta acceder a Vecino2.

```

class AccederVecino2 {
    public static void main(String args[]){
        System.out.println("Clase AccederVecino2: paquete virtual");
        Vecino2 ov2 = new Vecino2();
        System.out.println("-----");
        System.out.println("Vecino2 paq adr.cursojava."+ov2.nVecino2);
        System.out.println("-----");
    }
}

```

Al tratar de compilar producirá errores. Si se intenta ejecutar:



A una clase sin modificador de acceso no puede accederse desde otra que no pertenezca al mismo paquete.

Se intenta acceder a una variable de instancia y un método en el interior de una clase pública o sin modificador de acceso almacenada en un paquete desde otra que está en el mismo paquete (en este caso en el paquete virtual).

Se puede acceder sin problemas de cualquiera de estas cuatro formas:

Forma 1:

```

public class Vecino3 {
    public String nVecino = "German";
    public void saludo(){
        System.out.println("Hola desde la clase Vecino3");
    }
}

```

Forma 2:

```

public class Vecino3 {
    String nVecino = "German";
    void saludo(){
        System.out.println("Hola desde la clase Vecino3");
    }
}

```

Forma 3:

```

class Vecino3 {
    public String nVecino = "German";
    public void saludo(){
        System.out.println("Hola desde la clase Vecino3");
    }
}

```

Forma 4:

```

class Vecino3 {
    String nVecino = "German";
    void saludo(){
        System.out.println("Hola desde la clase Vecino3");
    }
}

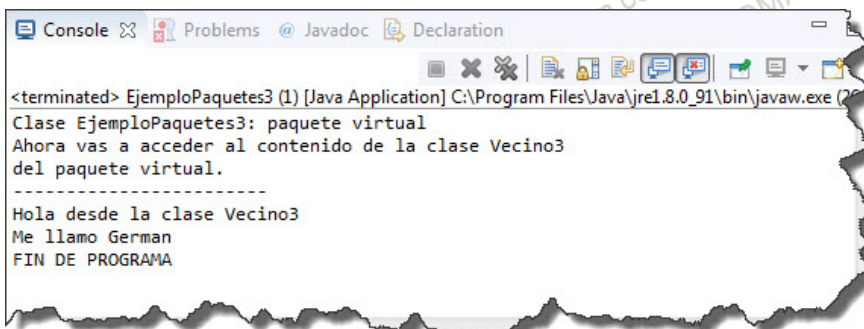
```

Código de la clase que acceder a Vecino3:

```

public class EjemploPaquetes3{
    public static void main(String args[]){
        System.out.println("Clase EjemploPaquetes3: paquete virtual");
        System.out.println("Ahora vas a acceder al contenido de la clase Vecino3 \ndel paquete virtual.");
        Vecino3 v=new Vecino3();
        System.out.println("-----");
        v.saludo();
        System.out.println("Me llamo "+v.nVecino);
        System.out.println("FIN DE PROGRAMA");
    }
}

```

Por consola:

Se intenta acceder a una variable de instancia privada declarada en una clase desde otra que pertenece al mismo paquete.

Código de la clase en la que se declara la variable de instancia privada.

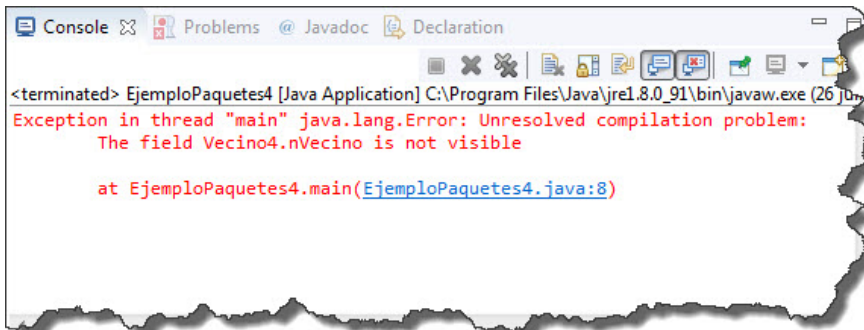
```
public class Vecino4{
    private String nVecino="German";
    void saludo(){
        System.out.println("Hola desde la clase Vecino4");
        System.out.println("Me llamo "+nVecino);
    }
}
```

Código de la clase desde la que se accede a Vecino4.

```
public class EjemploPaquetes4{
    public static void main(String args[]){
        System.out.println("Clase EjemploPaquetes4: paquete virtual");
        System.out.println("Ahora vas a acceder al contenido de la clase Vecino4 \ndel paquete virtual.");
        Vecino4 v=new Vecino4();
        System.out.println("-----");
        v.saludo();
        System.out.println("Me llamo "+v.nVecino);
        System.out.println("FIN DE PROGRAMA");
    }
}
```

Se producirá un error al compilar esta última clase ya que no se puede acceder a un atributo privado (variable o método) desde una clase distinta de aquella en la que están declarados.

Por consola:



Se accede a una variable privada y a métodos privados desde la clase en la que se declaran.

Código de la clase Vecino5, contiene una variable privada y métodos de acceso a esta variable:


```

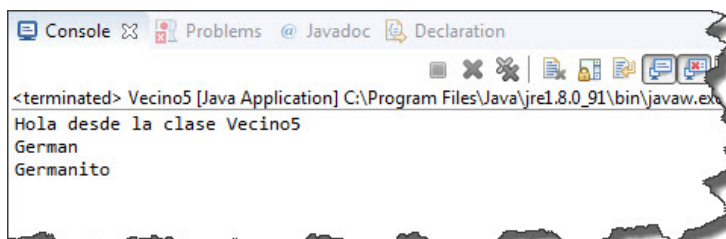
public class Vecino5{
    private String nombreVecino="German";
    public void saludo(){
        System.out.println("Hola desde la clase Vecino5");
    }

    //Método setter: su finalidad es asignar un valor a la variable de instancia nombreVecino
    private void setNombreVecino(String nombre){
        nombreVecino=nombre;
    }

    //Método getter: su finalidad es obtener el valor de la variable de instancia nombreVecino
    private String getNombreVecino(){
        return nombreVecino;
    }
    public static void main(String args[]){
        Vecino5 v=new Vecino5();
        v.saludo();
        System.out.println(v.getNombreVecino());
        v.setNombreVecino("Germanito");
        System.out.println(v.getNombreVecino());
    }
}

```

Esta clase ya contiene el método main por lo que se puede ejecutar directamente. El resultado por consola sería:



Modificador protected.

Se va a emplear el modificador protected.

Se diferencia del sin modificador en que se puede acceder a una variable de instancia o a un método protected de una clase que forma parte de un paquete, desde otra que no sea del mismo paquete siempre y cuando esta última herede de aquella.

Código de la clase Vecino6.

```

package adr.cursojava;

public class Vecino6{
    protected String nVecino="German";
}

```

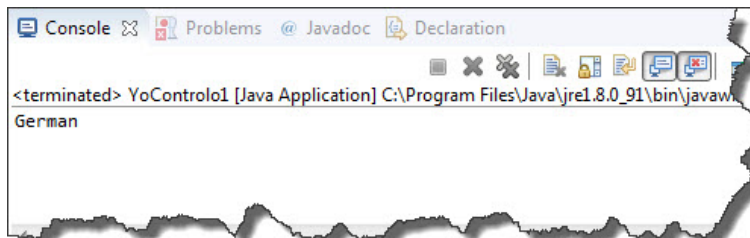
Código de la clase desde la que se accede a Vecino6.

```
import adr.cursojava.*;

class MiVecino6 extends Vecino6{
    void mostrar(){
        System.out.println(nVecino);
    }
}

public class YoControlo1{
    public static void main(String args[]){
        MiVecino6 mv=new MiVecino6();
        mv.mostrar();
    }
}
```

Por consola:



Ejercicios

Ejercicio 1. Paquetes

30

Crear un código que contenga la clase pública **ArrayIndices** y que siga el esquema de variables de instancia y métodos del esqueleto adjunto, de modo que **forme parte del paquete adr.cursojava**.

Lo necesario para empezar

```
package adr.cursojava;

public class ArrayIndices {
    public int enteros[];
    public int[] crearArray(int inferior,int superior){
        /*Este método va a recibir dos enteros que serán el primer y el
        * último elemento del array que devuelve.
        * El resto de elementos se obtendrán sumando
        * uno al primero hasta que se llegue al último.
        * Si el número asociado al argumento inferior es mayor que el
        * asociado al argumento superior se mostrará por consola el
        * mensaje "El último elemento debe ser mayor o igual que el primero.".
        * Después el programa finalizará su ejecución.
        */
    }

    public double productoEleArray(){
        /* Devuelve el producto de los elementos del array */
    }
    public double sumaEleArray(){
        /*Devuelve la suma de los elementos del array */
    }
}
```

Recomendaciones

En otro código fuente distinto, que se guardará en c:\cursojava\tema7, se definirá una clase que siga el esquema de variables de instancia y métodos adjuntos.

Se encargará de pasarle al método **crearArray(..)** de la clase **ArrayIndices** los valores adecuados para que **se muestre por consola lo que se especifica debajo**.

```

import adr.cursojava.*;
import java.io.*;

import adr.cursojava.ArrayIndices;

public class YoControlo{
    int primerEle,ultimoEle;
    void introducirEleInicialFinal(){
        /*
         * Debe pedir por consola al usuario el primer y último elemento
         * del array que va a generarse en la clase ArrayIndices. Si se
         * introduce incorrectamente alguno de los números que van a ser
         * primer y último elemento del array el programa mostrará un mensaje
         * de formato incorrecto y finalizará su ejecución
         */

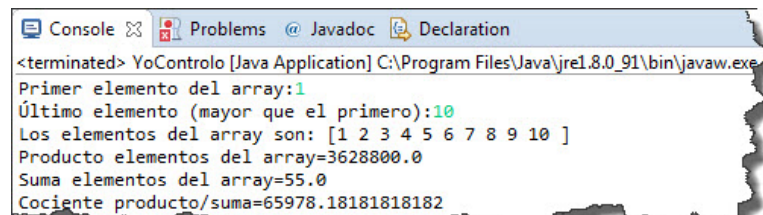
    }

    void mostrarEleArray(int misEnteros[]){
        /* Mostrará por consola los elementos del array */
    }

    public static void main(String args[]){
        /*
         * Debe crear los objetos adecuados e invocar a los métodos
         * oportunos para que todo funcione correctamente.
         * Mostrará por consola el producto y la suma de los elementos
         * del array y el cociente entre ambos.
         */
    }
}

```

Datos a mostrar por consola

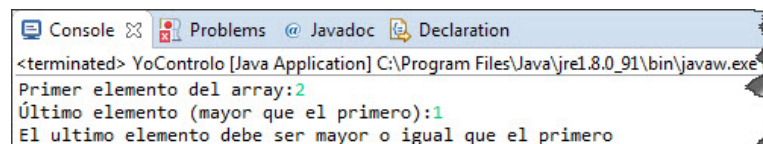


```

<terminated> YoControlo [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Primer elemento del array:1
Último elemento (mayor que el primero):10
Los elementos del array son: [1 2 3 4 5 6 7 8 9 10 ]
Producto elementos del array=362880.0
Suma elementos del array=55.0
Cociente producto/suma=65978.18181818182

```

Por consola si se introduce como último elemento un valor menor que el primero:

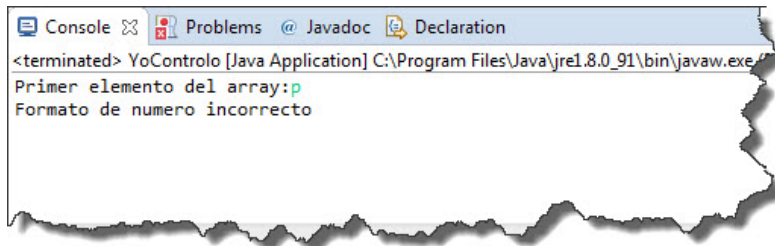


```

<terminated> YoControlo [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Primer elemento del array:2
Último elemento (mayor que el primero):1
El ultimo elemento debe ser mayor o igual que el primero

```

Por consola si se introduce un formato de número incorrecto:

A screenshot of a Java IDE's console window. The window has tabs for 'Console', 'Problems', 'Javadoc', and 'Declaration'. The console output shows the following text: '<terminated> YoControlo [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe', 'Primer elemento del array:p', and 'Formato de numero incorrecto'. The text is displayed in a monospaced font on a light background.

```
<terminated> YoControlo [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Primer elemento del array:p
Formato de numero incorrecto
```

Ejercicio 2. Usar paquetes ya creados

30

Consiste en **completar el código** que se muestra debajo, utilizando como clase de apoyo, la clase **Matrix** del paquete externo **Acme**.

Lo que se necesita para comenzar

```

import Acme.*;
import java.io.*;

public class MiMatriz{
    public static void main(String args[])throws Exception{
        InputStreamReader isr=new InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(isr);

        System.out.print("Numero filas=");
        int filas=Integer.parseInt(br.readLine());

        System.out.print("Numero columnas=");
        int columnas=Integer.parseInt(br.readLine());

        /*
         * Crear un objeto Matrix, que modela a una matriz 2D, mediante el número de filas y columnas introducido por el usuario
         * A continuación, pedir al usuario, mediante dos bucles anidados, los elementos de la matriz
         */

        /*
         * Mostrar los elementos de la matriz mediante dos bucles anidados.
         * A continuación, y mediante métodos adecuados de Matrix, la suma de sus elementos, el elemento mayor,
         * el elemento menor y la media aritmética de sus elementos.
         */

        /*
         * Calcular la matriz traspuesta de la anterior mediante un método adecuado de Matrix
         * Mostrar elementos matriz traspuesta y su numero de filas y columnas
         */

        /*
         * Calcular matriz suma de la inicial y la traspuesta mediante un método adecuado de Matrix
         * Mostrar elementos matriz suma
         */

        /*
         * Calcular matriz producto de la inicial y la traspuesta mediante un método adecuado de Matrix
         * Mostrar elementos matriz producto
         */

    }
}

```

Recomendaciones

Lo primero que debemos hacer es descargar y añadir a nuestro proyecto el paquete Acme que contiene la clase Matrix. Para ello daremos los siguientes pasos:

1. Descargar el paquete que se encuentra en <http://teleformacion.fer.es/general/prt/c/java/7/Acme.tar.gz>
2. Se descomprime el fichero en c:\cursojava y se obtiene Acme.tar. El fichero se trata igual que cualquier fichero zip, cualquier descompresor estilo Winzip servirá para obtener el contenido del fichero.
3. Se descomprime Acme.tar, también en c:\cursojava y se obtiene la carpeta c:\cursojava\Acme. Dentro de esta carpeta se encuentran todas las clases del paquete.
4. Para poder utilizarlas en nuestros códigos, la variable classpath debe apuntar al directorio a partir del que se ha colgado el directorio raíz del paquete, es decir, Acme.
¿Cuál es ese directorio?
c:\cursojava
Por tanto, si se emplea Eclipse, se agregará a su classpath la ruta c:\cursojava

Datos a mostrar por consola:

```

Console Problems Javadoc Declaration
<terminated> MiMatriz [Java Application] C:\Program Files\Java\jre1.8.0_91\bin\javaw.exe
Numero filas=3
Numero columnas=3
Elemento [0][0]=1
Elemento [0][1]=2
Elemento [0][2]=3
Elemento [1][0]=4
Elemento [1][1]=5
Elemento [1][2]=6
Elemento [2][0]=7
Elemento [2][1]=8
Elemento [2][2]=9
Elementos matriz:
1.0 2.0 3.0
4.0 5.0 6.0
7.0 8.0 9.0
Suma elementos matriz=45.0
Elemento mayor=9.0
Elemento menor=1.0
Media aritmetica de los elementos= 5.0
Elementos matriz traspuesta:
1.0 4.0 7.0
2.0 5.0 8.0
3.0 6.0 9.0
Numero filas matriz traspuesta=3
Numero columnas matriz traspuesta=3
Elementos matriz suma:
2.0 6.0 10.0
6.0 10.0 14.0
10.0 14.0 18.0
Elementos matriz producto:
14.0 32.0 50.0
32.0 77.0 122.0
50.0 122.0 194.0

```

Recursos

Enlaces de Interés



http://www.javahispano.org/antiguo_javahispano_org/2001/9/1/uso-de-paquetes-en-java.html

http://www.javahispano.org/antiguo_javahispano_org/2001/9/1/uso-de-paquetes-en-java.html

Uso de paquetes en Java