Java Básico

Otros conceptos 1

Copyright

Paquetes Java

- Los paquetes Java son una característica mas del lenguaje que nos permite organizar el código en grupos.
- Adicionalmente, ayudan a evitar colisiones en los nombres de las clases. De manera que en un programa que va a usar un framework de un tercero, tenga un 99% de seguridad de que no tiene ninguna clase con el mismo nombre.
- Toda clase Java pertenece a un paquete. Si no se especifica nada, pertenece al 'paquete por defecto' (que es un paquete raíz sin nombre).

Paquetes Java

- Para especificar el paquete al que pertenece una clase se utiliza la *keyword*: package
- La sentencia package tiene que ser la primera línea del fichero con el código fuente de la clase.
- Declaración de un paquete:
  - *package nombre_del_paquete;*
- Ejemplo:
  - **package** edu.upco.einf.javahispano;

Paquetes Java

- El nombre de una clase no se limita al identificador utilizado en la definición, sino:

  Nombre de paquete + Identificador de la Clase

- Ejemplo:
  - La clase Circulo del paquete edu.upco.einf.figuras es la clase edu.upco.einf.figuras.Circulo
- Por tanto, al ir a utilizar una clase debemos conocer siempre el paquete al que pertenece para poder referenciarla porque si no el compilador no va a saber encontrarla.

Paquetes Java

- Existe una convención aceptada por todos los desarrolladores en cuanto a la nomenclatura de los paquetes Java:
  - Todas las palabras que componen el nombre del paquete van en minúsculas.
  - Se suele utilizar el nombre de dominio invertido para intentar asegurar nombres unívocos y evitar colisiones.
- Ejemplo:

  com.ibm.test;
  edu.upco.einf.practica10.figuras;
  es.chemi.juegos;

Paquetes Java

- Para utilizar una clase en nuestro código tenemos que escribir su nombre completo: paquete + clase.
- Pero existe otro mecanismo para facilitar la codificación que es el uso de la *keyword*: import.
- Declaración de un import:
  - *import nombre_del_paquete.nombre_de_la_clase;*
  - *import nombre_del_paquete.*;*
- Ejemplo:
  - **import** edu.upco.einf.figuras.Circulo;
  - **import** edu.upco.einf.figuras.*;

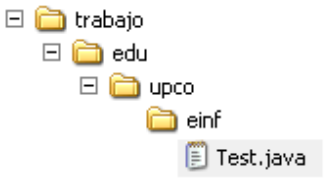
Paquetes Java

- Los import se ubican entre la sentencia package y la definición de la clase.
- Las clases importadas de esta manera pueden ser referenciadas en el código directamente por su nombre de clase sin necesidad de escribir el paquete.
- Un import genérico (es decir, con el *) importa solo las clases de ese paquete, pero no de los subpaquetes.

Paquetes Java

- Al igual que las clases Java tienen un reflejo en el Sistema de Archivos (una clase Java equivale a un fichero texto de extensión *.java), lo mismo ocurre con los paquetes Java.
- Los paquetes Java equivalen a directorios. Es decir, cada miembro del paquete (separado por puntos) se traduce a un directorio en el Sistema de Archivos.
- Ejemplo:

```
package edu.upco.einf;
public class Test { … }
```



Nota: \trabajo es el directorio de trabajo que está en el CLASSPATH

Paquetes Java

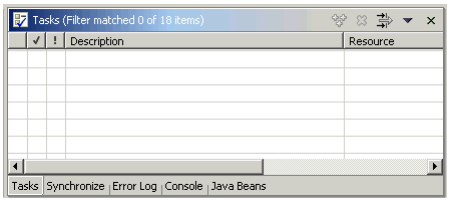
- Para compilar tenemos distintas opciones:
  - Desde c:\trabajo hacemos: javac edu\upco\einf\Test.java
  - Desde c:\trabajo\edu\upco\einf hacemos: javac Test.java
- Para ejecutar solo tenemos una opción:
  - Desde cualquier punto del sistema hacemos: java edu.upco.einf.Test
- Nota: el directorio c:\trabajo debe estar en el CLASSPATH.

Paquetes Java

- Por tanto para utilizar una clase tenemos tres alternativas:
  - Utilizar su nombre completo: paquete + clase.
  - Importar la clase: import paquete + clase.
  - Importar el paquete completo: import paquete + *.
- Un import no implica la inclusión de código como ocurre en un #include de C++. Simplemente son vías de acceso para buscar el código. El código se va cargando según se necesita.

Ejemplo

```
public class UnaClase
{
  private int param;

  public UnaClase(int param)
  {
    this.param = param;
  }
}
```

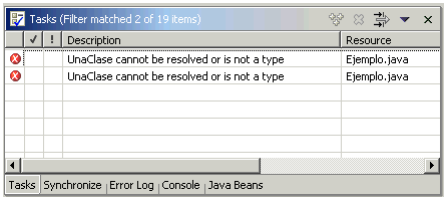



```
public class Ejemplo
{
  public static void main(String[] args)
  {
    UnaClase c = new UnaClase(12);
  }
}
```

No hay errores. Ambas clases están en el paquete por defecto y por tanto se encuentran.

Ejemplo

```
package edu.upco.einf;

public class UnaClase
{
  private int param;

  public UnaClase(int param)
  {
    this.param = param;
  }
}
```

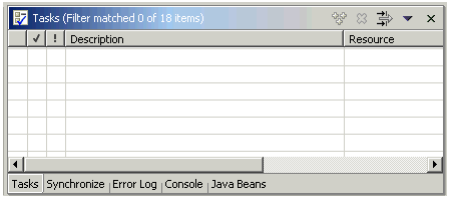



```
public class Ejemplo
{
  public static void main(String[] args)
  {
    UnaClase c = new UnaClase(12);
  }
}
```

Hay errores. UnaClase está en un paquete distinto a Ejemplo y por tanto no la encuentra.

Ejemplo

```
package edu.upco.einf;

public class UnaClase
{
  private int param;

  public UnaClase(int param)
  {
    this.param = param;
  }
}
```

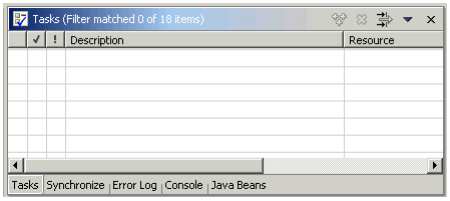



```
public class Ejemplo
{
  public static void main(String[] args)
  {
    edu.upco.einf.UnaClase c = new edu.upco.einf.UnaClase(12);
  }
}
```

No hay errores. Estamos referenciando a UnaClase a través de su nombre completo.

Ejemplo

```
package edu.upco.einf;

public class UnaClase
{
  private int param;

  public UnaClase(int param)
  {
    this.param = param;
  }
}
```

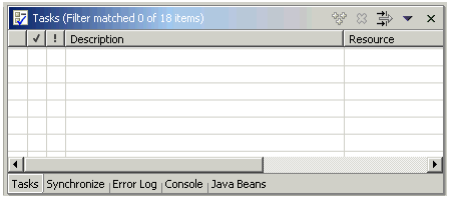



```
import edu.upco.einf.UnaClase;

public class Ejemplo
{
  public static void main(String[] args)
  {
    UnaClase c = new UnaClase(12);
  }
}
```

No hay errores. Hemos añadido un import de la clase UnaClase.

Ejemplo

```
package edu.upco.einf;

public class UnaClase
{
  private int param;

  public UnaClase(int param)
  {
    this.param = param;
  }
}
```

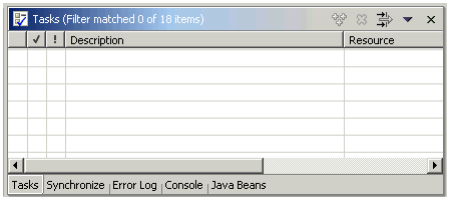



```
import edu.upco.einf.*;

public class Ejemplo
{
  public static void main(String[] args)
  {
    UnaClase c = new UnaClase(12);
  }
}
```

No hay errores. Hemos añadido un import de todas las clases del paquete de UnaClase.

Ejemplo

```
package edu.upco.einf;

public class UnaClase
{
  private int param;

  public UnaClase(int param)
  {
    this.param = param;
  }
}
```




```
package edu.upco.einf;

public class Ejemplo
{
  public static void main(String[] args)
  {
    UnaClase c = new UnaClase(12);
  }
}
```

No hay errores. Las dos clases pertenecen al mismo paquete.

Paquetes Java

- Las clases System, String, Math, etc… pertenecen al paquete java.lang.*.
- ¿Cómo compilaban todas nuestras prácticas si no conocíamos los paquetes Java (y por tanto la *keyword* import)?
- Porque el compilador por defecto siempre añade la siguiente línea a nuestro código:

```
import java.lang.*;
```

Paquetes Java

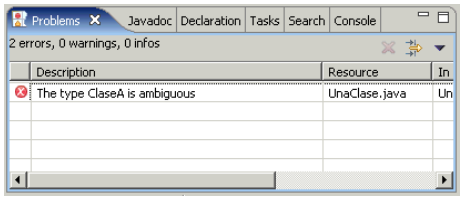
- Aunque no es frecuente, es posible que provoquemos ambigüedades en el uso de los imports, y por tanto errores de compilación.
- ¿Qué ocurre al usar una clase cuyo nombre existe a la vez en dos paquetes que hemos importado? ¿Cuál de las dos clases es la que se debe utilizar?
- En esos casos, hay que importar o referirse a la clase conflictiva mediante su identificador completo: paquete + clase.

Ejemplo

```
package edu.upco.iinf;

public class ClaseA
{
}
```

```
package edu.upco.itig;

public class ClaseA
{
}
```

```
import edu.upco.iinf.*;
import edu.upco.itig.*;

public class UnaClase
{
  public static void main(String[] args)
  {
    ClaseA a = new ClaseA();
  }
}
```

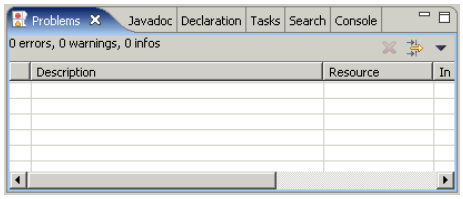


Hay errores. Existe una ambigüedad en el uso de ClaseA.

Ejemplo

```
package edu.upco.iinf;

public class ClaseA
{
}
```

```
package edu.upco.itig;

public class ClaseA
{
}
```



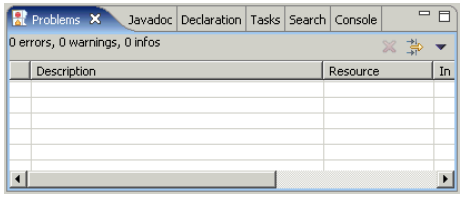

```
import edu.upco.iinf.*;
import edu.upco.itig.*;

public class UnaClase
{
  public static void main(String[] args)
  {
    edu.upco.iinf.ClaseA a = new edu.upco.iinf.ClaseA();
  }
}
```

No hay errores. No hay ambigüedad en el uso de ClaseA.

Ejemplo

```
package edu.upco.iinf;

public class ClaseA
{
}
```

```
package edu.upco.itig;

public class ClaseA
{
}
```




```
import edu.upco.iinf.ClaseA;
import edu.upco.itig.*;

public class UnaClase
{
  public static void main(String[] args)
  {
    ClaseA a = new ClaseA();
  }
}
```

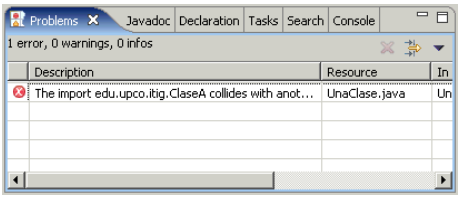
No hay errores. No hay ambigüedad en el uso de ClaseA.

Ejemplo

```
package edu.upco.iinf;

public class ClaseA
{
}
```

```
package edu.upco.itig;

public class ClaseA
{
}
```

```
import edu.upco.iinf.ClaseA;
import edu.upco.itig.ClaseA;

public class UnaClase
{
  public static void main(String[] args)
  {
    ClaseA a = new ClaseA();
  }
}
```



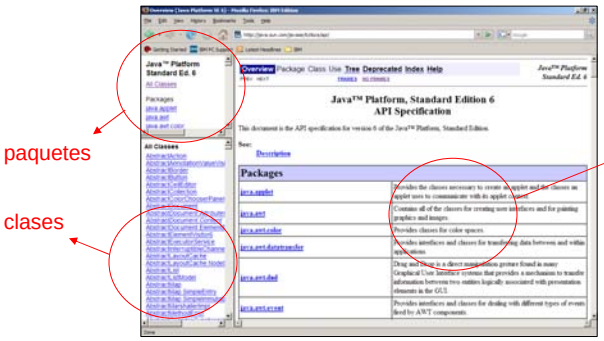
Hay errores. Se están importando dos clases con el mismo nombre.

Paquetes Java

- Hemos visto que existe el llamado ‘paquete por defecto’ al que pertenecen todas aquellas clases que no indican de forma explícita un paquete determinado en su código.
- Desde la versión 1.4.x, el compilador no permite importar desde una clase que pertenece a un paquete explícito, una clase que pertenece al ‘paquete por defecto’.
- Esto no tendría que ser un problema en la mayoría de los casos porque siempre deberíamos ubicar las clase en paquetes Java de forma explícita.

El API

- On-line: http://java.sun.com/javase/6/docs/api/index.html



- Impresa: Java in a Nutshell, 5th Edition

Modificadores de acceso

- Existen cuatro tipos de moficiadores de acceso y por tanto 'cuatro' keywords:
  - public -> (público).
  - protected -> (protegido).
  - -> (paquete, identificado por la ausencia de keyword).
  - private -> (privado).
- Están ordenados de menor a mayor restricción.
- El modificador de acceso indica quién puede acceder a dicha clase, atributo o método.

Modificadores de acceso

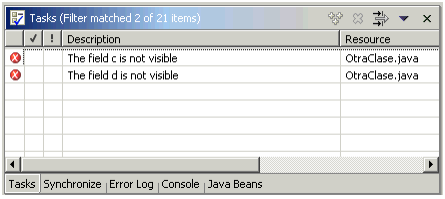
Acceso a…	public	protected	package	private
Clases del mismo paquete	Si	Si	Si	No
Subclases de mismo paquete	Si	Si	Si	No
Clases de otros paquetes	Si	No	No	No
Subclases de otros paquetes	Si	Si	No	No

Modificadores de acceso

- Los modificadores de acceso se utilizan en las definiciones de:
  - Clases e interfaces: solo se permiten public y package.
  - Atributos: se permiten cualquiera de los cuatro.
  - Métodos: se permiten cualquiera de los cuatro.

Ejemplo

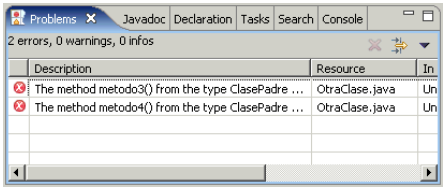
```
package edu.upco.einf;
public class ClasePadre
{
  public int a;
  protected int b;
  int c;
  private int d;
}

import edu.upco.einf.ClasePadre;
public class OtraClase extends ClasePadre
{
  public void miMetodo()
  {
    int tmp = 0;
    tmp = a;
    tmp = b;
    tmp = c;
    tmp = d;
  }
}
```



Hay errores. Ni c ni d son accesibles desde OtraClase.

Ejemplo

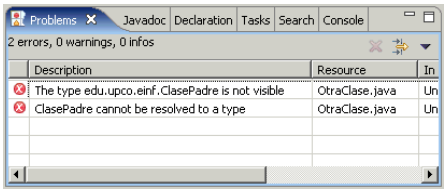
```
package edu.upco.einf;
public class ClasePadre
{
  public void metodo1() { }
  protected void metodo2() { }
  void metodo3() { }
  private void metodo4() { }
}

import edu.upco.einf.ClasePadre;
public class OtraClase extends ClasePadre
{
  public void miMetodo()
  {
    metodo1();
    metodo2();
    metodo3();
    metodo4();
  }
}
```



Hay errores. Ni metodo3 ni metodo4 son accesibles desde OtraClase.

Ejemplo

```
package edu.upco.einf;

class ClasePadre
{

}

import edu.upco.einf.ClasePadre;

public class OtraClase extends ClasePadre
{

}
```



Hay errores. ClasePadre no es accesibles desde OtraClase.

Métodos estáticos

- Existen casos en los que nos encontramos con clases cuyos métodos no dependen en absoluto de los atributo de la clase.
- Por ejemplo, la clase java.lang.Math:
  - Su método round recibe un número decimal y lo devuelve redondeado.
  - Su método sqrt recibe un número y devuelve su raiz cuadrada.
  - Su método min recibe dos números y devuelve el menor.
  - Etc…

Métodos estáticos

- Son métodos que parece no pertenecer a una entidad concreta. Son genéricos, globales, independientes de cualquier estado.
- ¿Tiene sentido instanciar un objeto para ejecutar algo que no depende de nada de dicho objeto?
- La respuesta es no. Y para ello contamos en Java con los métodos estáticos.

Métodos estáticos

- Para definir un método estático utilizamos la *keyword*: static
- Declaración:

```
modifi_acceso static tipo_retorno nombre([tipo parametro,..])
{
}
```

- Ejemplo:

```
public static void miMetodo()
{
}
```

Métodos estáticos

- Para ejecutar por tanto un método estático no hace falta instanciar un objeto de la clase. Se puede ejecutar el método directamente sobre la clase.
- Ejemplo:
  - `int a = Math.min(10,17);`
- Mientras que los métodos convencionales requieren de un objeto:
  - `String s = new String("Hola");`
    `int a = s.indexOf('a');`

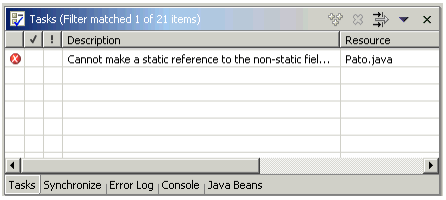
`int a = String.indexOf('a');` ~~(tachado)~~

Métodos estáticos

- Una clase puede perfectamente mezclar métodos estáticos con métodos convencionales.
- Un ejemplo clásico es el método main:
  - `public static void main(String[] args) { … }`
- Un método estático jamás puede acceder a un atributo de instancia (no estático).
- Un método estático jamás puede acceder a un método de instancia (no estático).
- Pero desde un método convencional si que se puede acceder a atributos y métodos estáticos.

Ejemplo

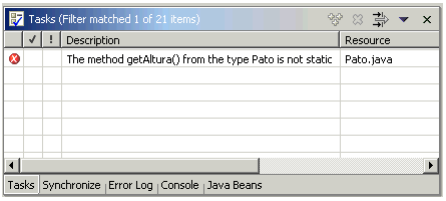
```
public class Pato
{
  private int altura;

  public static void main(String[] args)
  {
    System.out.println("La edad es" + altura);
  }

  public int getAltura()
  {
    return altura;
  }

  public void setAltura(int param)
  {
    altura = param;
  }
}
```



Hay errores. ¿Qué altura muestra? Si no existe ningún pato.

Ejemplo

```
public class Pato
{
  private int altura;

  public static void main(String[] args)
  {
    System.out.println("La edad es" + getAltura());
  }

  public int getAltura()
  {
    return altura;
  }

  public void setAltura(int param)
  {
    altura = param;
  }
}
```



Hay errores. ¿Qué altura utiliza el método getAltura()? Si no existe ningún pato.

Atributos estáticos

- Los atributos estáticos (o variables estáticas) son atributos cuyo valor es compartido por todos los objetos de una clase.
- Para definir un atributo estático utilizamos la *keyword*: static
- Definición de un atributo estático:
  - *modifi_acceso static tipo nombre [= valor_inicial];*
- Ejemplo:
  - **public static int** contador = 0;

Atributos estáticos

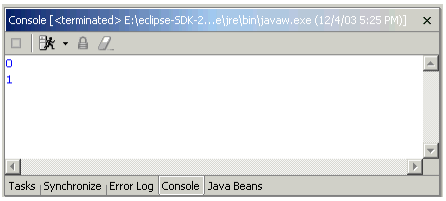
- Hay que tratarlos con cuidado puesto que son fuente de problemas difíciles de detectar.
- Como todos los objetos de una misma clase comparte el mismo atributo estático, hay que tener cuidado porque si un objeto ‘a’ modifica el valor del atributo, cuando el objeto ‘b’ vaya a usar dicho atributo, lo usa con un valor modificado.
- Recordemos que sin embargo los atributos convencionales (de instancia) son propios de cada objeto.

Atributos estáticos

- Los atributos estáticos son cargados en memoria cuando se carga la clase. Siempre antes de que:
  - Se pueda instanciar un objeto de dicha clase.
  - Se pueda ejecutar un método estático de dicha clase.
- Para usar un atributo estático no hace falta instanciar un objeto de la clase.
- Ejemplo:
  - System.out.println(“Hola”);
  - out es un atributo estático de la clase java.lang.System.

Ejemplo

```
public class Jugador
{
  public static int contador = 0;
  private String nombre = null;

  public Jugador(String param)
  {
    nombre = param;
    contador++;
  }
}

public class Test
{
  public static void main(String[] args)
  {
    System.out.println(Jugador.contador);
    Jugador uno = new Jugador("Ronnie");
    System.out.println(Jugador.contador);
  }
}
```

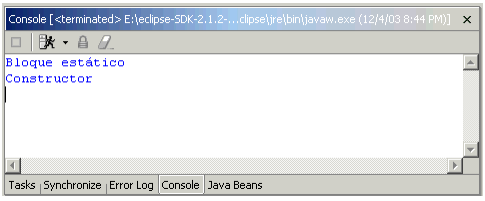


Bloques de código estáticos

- Los bloques de código estático son trozos de código que se ejecutan al cargar una clase en memoria (no al instanciar objetos de esa clase).
- Para definir un bloque estático utilizamos la *keyword*: static
- Definición de un bloque estático:
  - *static { …. }*
- Ejemplo:
  - **static** { System.out.println(“Hola”); }

Ejemplo

```
public class BloqueStatic
{
  static
  {
    System.out.println("Bloque estático");
  }

  public BloqueStatic()
  {
    System.out.println("Constructor");
  }
}

public class TestStatic
{
  public static void main(String[] args)
  {
    BloqueStatic bs = new BloqueStatic();
  }
}
```



Clases finales

- Para definir una clase como final utilizamos la *keyword*: final
- Declaración de una clase final:

```
modificador_acceso final class nombre_clase
{
}
```

- Ejemplo:

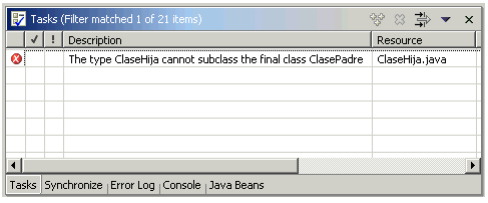
```
public final class MiClase
{
}
```

Clases finales

- Definiendo una clase como final conseguimos que ninguna otra clase pueda heredar de ella.

```
public final class ClasePadre
{
}

public class ClaseHija extends ClasePadre
{
}
```

Hay errores. ClaseHija no puede heredar de ClasePadre porque está es final.



Métodos finales

- Para definir un método como final utilizamos la *keyword*: final
- Declaración de un método final:
  - *modif_acceso final tipo_retorno nombre([tipo param,..])*
    {
    }
- Ejemplo:

```
public final int suma(int param1, int param2)
{
   return param1 + param2;
}
```

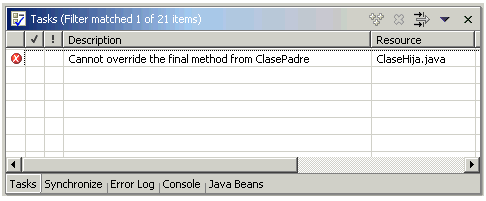
Métodos finales

- Definiendo un método como final conseguimos que ninguna otra clase pueda sobrescribirlo.

```
public class ClasePadre
{
  public final int suma(int a, int b)
  {
    return a + b;
  }
}
```

Hay errores. ClaseHija no puede sobrescribir los métodos final de ClasePadre.

```
public class ClaseHija extends ClasePadre
{
  public int suma(int a, int b)
  {
    return a * b;
  }
}
```



Atributos finales

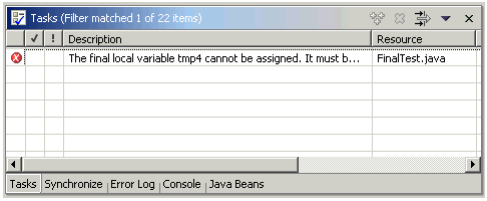
- Para definir un atributo como final utilizamos la *keyword*: final
- Declaración de un atributo final:
  - *modificador_acceso final tipo nombre [= valor_inicial];*
- Ejemplo:

```
protected final boolean sw = true;
public final int i;
```

Atributos finales

- Definiendo un atributo como final conseguimos constantes. Es decir, una vez inicializados no se puede cambiar su valor.

Hay errores. No podemos modificar tmp4.




```
public class FinalTest
{
  final int tmp1 = 3;//Ya no podemos cambiar tmp1
  final int tmp2;

  public FinalTest()
  {
    tmp2 = 42;  //Ya no podemos cambiar tmp2.
  }
  public void hacerAlgo(final int tmp3)
  {
   // No podemos cambiar tmp3.
  }
  public void hacerAlgoMas()
  {
    final int tmp4 = 7; //No podemos cambiar tmp4
    tmp4 = 5;
  }
}
```

Ejercicio

- ¿Cuál de estos programas compila sin errores?

```
public class MiClase
{
  static int x;
  public void hacerAlgo()
  {
    System.out.println(x);
  }
}
```

```
public class MiClase
{
  int x;
  public static void hacerAlgo()
  {
    System.out.println(x);
  }
}
```

Ejercicio

- ¿Cuál de estos programas compila sin errores?

```
public class MiClase
{
  static int x;
  public void hacerAlgo()
  {
    System.out.println(x);
  }
}
```

Compila. Desde un método de instancia se puede acceder a un atributo estático.

```
public class MiClase
{
  int x;
  public static void hacerAlgo()
  {
    System.out.println(x);
  }
}
```

No compila. Desde un método estático no se puede acceder a un atributo de instancia.

Ejercicio

- ¿Cuál de estos programas compila sin errores?

```
public class MiClase
{
  final int x = 5;
  public void hacerAlgo()
  {
    System.out.println(x);
  }
}
```

```
public class MiClase
{
  static final int x = 12;
  public void hacerAlgo()
  {
    System.out.println(x);
  }
}
```

Ejercicio

- ¿Cuál de estos programas compila sin errores?

```
public class MiClase
{
  final int x = 5;
  public void hacerAlgo()
  {
    System.out.println(x);
  }
}
```

Compila. Se está accediendo a un atributo final de instancia desde un método de instancia.

```
public class MiClase
{
  static final int x = 12;
  public void hacerAlgo()
  {
    System.out.println(x);
  }
}
```

Compila. Se está accediendo a un atributo final y estático desde un método de instancia.

Ejercicio

- ¿Cuál de estos programas compila sin errores?

```
public class MiClase
{
  static final int x = 12;
  public void hacerAlgo(final int x)
  {
    System.out.println(x);
  }
}
```

```
public class MiClase
{
  int x = 12;
  public static void hacerAlgo(final int x)
  {
    System.out.println(x);
  }
}
```

Ejercicio

- ¿Cuál de estos programas compila sin errores?

```
public class MiClase
{
  static final int x = 12;
  public void hacerAlgo(final int x)
  {
    System.out.println(x);
  }
}
```

Compila. Se está accediendo a una variable local final.

```
public class MiClase
{
  int x = 12;
  public static void hacerAlgo(final int x)
  {
    System.out.println(x);
  }
}
```

Compila. Se está accediendo a una variable local final.

Definición de constantes

- Las constantes en Java se suelen definir mediante la combinación de las keyword: static y final.
- Declaración de una constante:
  - *modificador_acceso static final tipo nombre = valor;*
- Ejemplo:
  - **public static final double** PI = 3.141592653589;
- Por convención las constantes se suelen llamar con todas las letras en mayúsculas.

Definición de constantes

- Algunos ejemplos existentes:
  - java.lang.Math.PI: el número PI.
  - java.lang.Math.E: el número E.
  - javax.swing.SwingConstants.CENTER: centrado.
  - java.awt.event.KeyEvent.VK_ENTER: tecla de intro.
- En ocasiones cuando se crea una clase solo con constantes, se suele hacer mediante un interface.

static imports

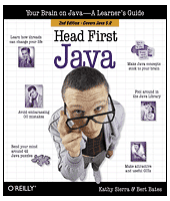
- Java SE 5.0 añade una novedad al respecto, permitiendo la importación de atributos y métodos estáticos, de manera que no haya que nombrar a la clase para su acceso.
- La declaración de este nuevo tipo de import es:
  - *import static nombredelpaquete.nombredelaclase.miembro;*
  - *import static nombredelpaquete.nombredelaclase.*;*
- Ejemplo:
  - **import static** java.lang.System.out;
  - **import static** java.lang.Math.*;
- Veremos mas detalles en el capítulo JavaSE 5.0

Ejemplo

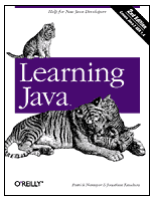
```
public class AntesTest
{
  public static void main(String[] args)
  {
    System.out.println(System.currentTimeMillis());
  }
}


import static java.lang.System.out;
import static java.lang.System.currentTimeMillis;

public class AhoraTest
{
  public static void main(String[] args)
  {
    out.println(currentTimeMillis());
  }
}
```

Bibliografía

- Head First Java (2nd edition)
  Kathy Sierra y Bert Bates.
  O'Reilly



- Learning Java (2nd edition)
  Patrick Niemeyer y Jonathan Knudsen.
  O'Reilly.



- Thinking in Java (4th edition)
  Bruce Eckel.
  Prentice Hall.

- The Java tutorial
  http://java.sun.com/docs/books/tutorial/