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

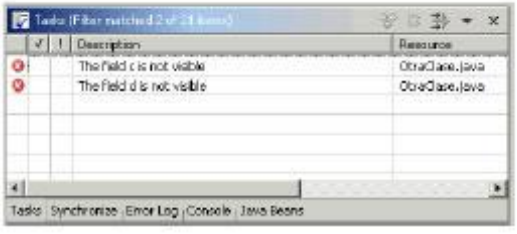
Acceso a…	public	protected	package	private
Clases del mismo paquete	Si	Si	Si	No
Subclases de mismo paquete	Si	Si	Si	No
Clases de otros paquetes	Si	No	No	No
Subclases de otros paquetes	Si	Si	No	No

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.

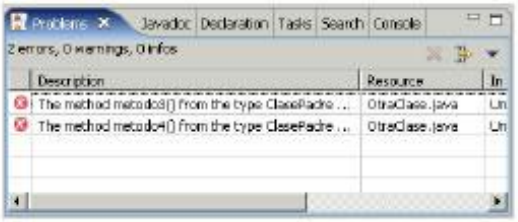
Ejemplo

```
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;
  }
}
```



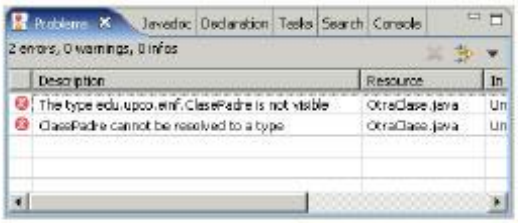
Ejemplo

```
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();
  }
}
```



Ejemplo

```
package edu.upco.einf;

class ClasePadre
{

}

import edu.upco.einf.ClasePadre;

public class OtraClase extends ClasePadre
{

}
```



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');
```

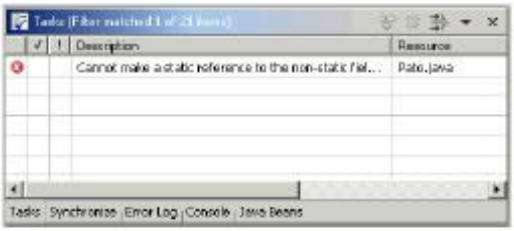
```
int a = String.indexOf('a');
```

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.

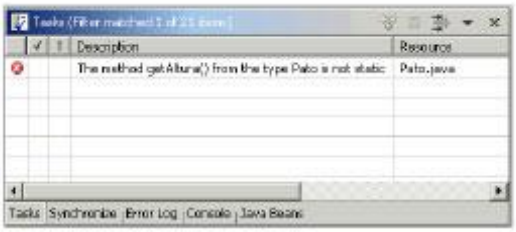
Ejemplo

```
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;
  }
}
```



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;
  }
}
```



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;
}
```

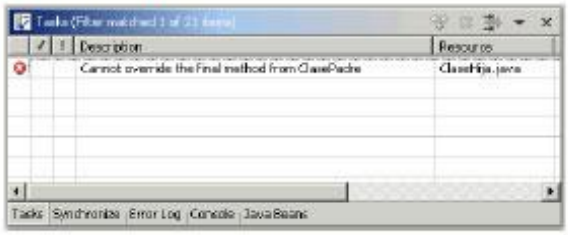
Métodos finales

- 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

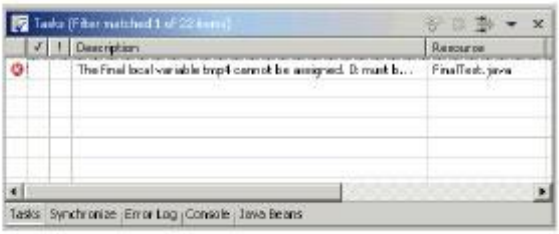
- Para definir un atributo como final utilizamos la *keyword*: final
- Declaración de un atributo final:
  - *modificador_acceso final tipo nombre [= valor_inicial];*
- Ejemplo:

```
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?

1

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

2

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

Ejercicio

¿Cuál de estos programas compila sin errores?

3

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

4

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

Ejercicio

¿Cuál de estos programas compila sin errores?

5

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

6

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

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.