

# Programación

Unidad 6: Paquetes, static y final

# Unidad 6

- Paquetes
- Métodos estáticos
- Atributos estáticos
- Clases finales
- Métodos finales
- Atributos finales
- Definición de constantes

# Paquetes

# Unidad 6: Paquetes, static y final

## Paquetes

Los paquetes Java son una característica más 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).

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 com.cesurformacion.app;
```

# Unidad 6: Paquetes, static y final

## Paquetes

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

**Nombre de paquete + Identificador de la Clase**

Ejemplo:

La clase ***Circulo*** del paquete ***com.cesurformacion.figuras*** es la clase ***com.cesurformacion.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.

# Unidad 6: Paquetes, static y final

## Paquetes

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.dptoinf.figuras;`
- `es.easports.juegos;`

# Unidad 6: Paquetes, static y final

## Paquetes

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 com.cesurformacion.figuras.Circulo;  
import com.cesurformacion.figuras.*;
```

# Unidad 6: Paquetes, static y final

## Paquetes

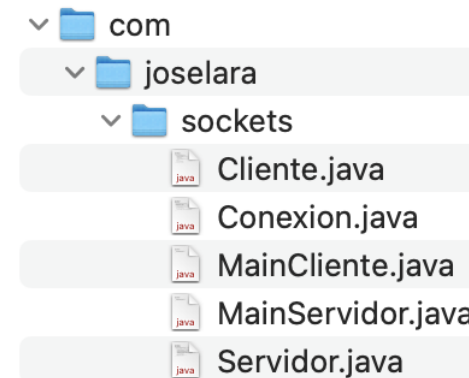
Los import se ubican entre la sentencia package y la definición de la clase.

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.

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.

```
package com.joselara.sockets;  
import java.io.IOException;  
  
public class MainCliente  
{
```





# Unidad 6: Paquetes, static y final

## Paquetes

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 + \*

```
package com.empresa.modelo;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
package com.empresa.modelo;

public class Ejemplo {

    public static void main(String[] args) {
        ClaseUno c = new ClaseUno(12);
        c.getParametro();
    }
}
```

# Unidad 6: Paquetes, static y final

## Paquetes

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

```
package com.empresa.modelo;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
package com.empresa.modelo;

public class Ejemplo {

    public static void main(String[] args) {
        ClaseUno c = new ClaseUno(12);
        c.getParametro();
    }
}
```

# Unidad 6: Paquetes, static y final

## Paquetes

```
package com.empresa.modelo;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
public class Ejemplo {

    public static void main(String[] args) {
        ClaseUno c = new ClaseUno(12);
        c.getParametro();
    }
}
```

# Unidad 6: Paquetes, static y final

## Paquetes

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

```
package com.empresa.modelo;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
public class Ejemplo {

    public static void main(String[] args) {
        ClaseUno c = new ClaseUno(12);
        c.getParametro();
    }
}
```

# Unidad 6: Paquetes, static y final

## Paquetes

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

```
package com.empresa.modelo;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
package com.empresa.clases;

public class Ejemplo {

    public static void main(String[] args) {
        com.empresa.modelo.ClaseUno c = new com.empresa.modelo.ClaseUno(12);
        c.getParametro();
    }
}
```

# Unidad 6: Paquetes, static y final

## Paquetes

*No hay errores. Hemos añadido un import de la clase ClaseUno.*

```
package com.empresa.modelo;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
package com.empresa.clases;

import com.empresa.modelo.ClaseUno;

public class Ejemplo {

    public static void main(String[] args) {
        ClaseUno c = new ClaseUno(12);
        c.getParametro();
    }
}
```

# Unidad 6: Paquetes, static y final

## Paquetes

*No hay errores. Hemos añadido un import de la clase ClaseUno.*

```
package com.empresa.modelo;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
package com.empresa.clases;

import com.empresa.modelo.*;

public class Ejemplo {

    public static void main(String[] args) {
        ClaseUno c = new ClaseUno(12);
        c.getParametro();
    }
}
```

# Unidad 6: Paquetes, static y final

## Paquetes

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.*;
```

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**.



# Unidad 6: Paquetes, static y final

## Paquetes

```
package com.empresa.modelo;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
package com.empresa.util;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
package com.empresa.clases;

import com.empresa.modelo.*;
import com.empresa.util.*;

public class Ejemplo {

    public static void main(String[] args) {
        ClaseUno c = new ClaseUno(12);
        c.getParametro();
    }
}
```

# Unidad 6: Paquetes, static y final

## Paquetes

*Hay errores. Existe una ambigüedad en el uso de ClaseUno.*

```
package com.empresa.modelo;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
package com.empresa.util;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
package com.empresa.clases;

import com.empresa.modelo.*;
import com.empresa.util.*;

public class Ejemplo {

    public static void main(String[] args) {
        ClaseUno c = new ClaseUno(12);
        c.getParametro();
    }
}
```

# Unidad 6: Paquetes, static y final

## Paquetes

*No hay errores. No hay ambigüedad en el uso de ClaseUno*

```
package com.empresa.modelo;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
package com.empresa.util;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
package com.empresa.clases;

import com.empresa.modelo.*;
import com.empresa.util.*;

public class Ejemplo {

    public static void main(String[] args) {
        com.empresa.util.ClaseUno c = new com.empresa.util.ClaseUno(12);
        c.getParametro();
    }
}
```

# Unidad 6: Paquetes, static y final

## Paquetes

*Hay errores. Existe una ambigüedad en el uso de ClaseUno.*

```
package com.empresa.modelo;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
package com.empresa.util;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
package com.empresa.clases;

import com.empresa.modelo.*;
import com.empresa.util.*;

public class Ejemplo {

    public static void main(String[] args) {
        ClaseUno c = new ClaseUno(12);
        c.getParametro();
    }
}
```

# Unidad 6: Paquetes, static y final

## Paquetes

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

```
package com.empresa.modelo;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
package com.empresa.util;

public class ClaseUno {
    private int parametro;

    public ClaseUno(int parametro) {
        super();
        this.parametro = parametro;
    }

    public int getParametro() {
        return parametro;
    }

    public void setParametro(int parametro) {
        this.parametro = parametro;
    }
}
```

```
package com.empresa.clases;

import com.empresa.modelo.ClaseUno;
import com.empresa.util.ClaseUno;

public class Ejemplo {

    public static void main(String[] args) {
        ClaseUno c = new ClaseUno(12);
        c.getParametro();
    }
}
```

# Métodos estáticos

# Unidad 6: Paquetes, static y final

## Métodos estáticos

Son métodos que parecen **no pertenecer** a **una entidad concreta**. Son genéricos, globales, independientes de cualquier estado.

¿Tiene sentido instanciar un objeto para ejecutar algo que no depende en nada de dicho objeto?

La respuesta es **no**. Y para ello contamos en Java con los métodos estáticos.

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

Declaración:

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

Ejemplo:

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

# Unidad 6: Paquetes, static y final

## 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');
```





# Unidad 6: Paquetes, static y final

## 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 **NUNCA** 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 **sí** que se puede acceder a atributos y métodos estáticos.

# Atributos estáticos

# Unidad 6: Paquetes, static y final

## 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;
```

# Unidad 6: Paquetes, static y final

## Atributos estáticos

Hay que tratarlos con cuidado puesto que son fuente de problemas difíciles de detectar. Como **todos los objetos** de una misma clase **comparten** 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 **proprios** de cada objeto.

# Clases finales

# Unidad 6: Paquetes, static y final

## Clases finales

Para definir una clase como final utilizamos la keyword: **final**

Declaración de una clase final:

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

Ejemplo:

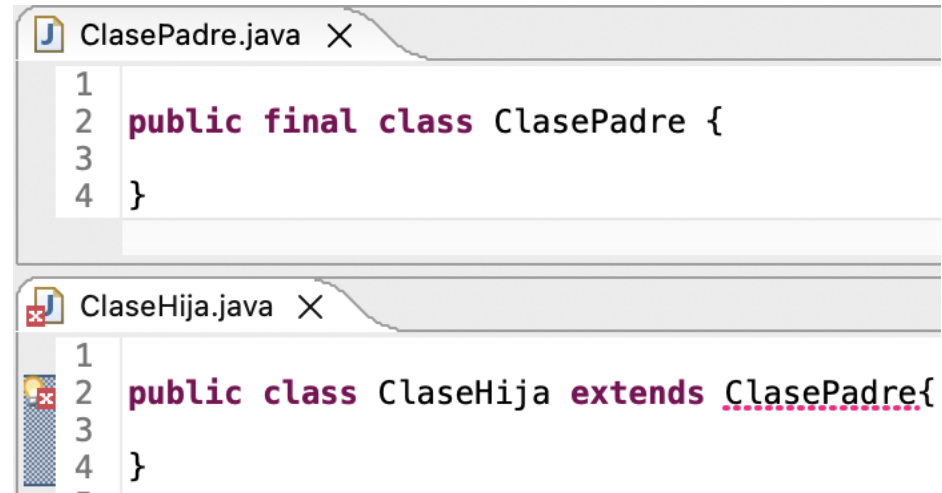
```
public final class MiClase {}
```

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

# Unidad 6: Paquetes, static y final

## Clases finales

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



```
ClasePadre.java X
1
2 public final class ClasePadre {
3
4 }

ClaseHija.java X
1
2 public class ClaseHija extends ClasePadre{
3
4 }
```

# Métodos finales



# Unidad 6: Paquetes, static y 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;  
}
```

Definiendo un método como final conseguimos que **ninguna otra clase pueda sobrecribirlo**.

# Atributos finales

# Unidad 6: Paquetes, static y final

## 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;
```

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

# Unidad 6: Paquetes, static y final

## Atributos finales

Hay errores. No podemos modificar valor3.

```
public class FinalTest {  
  
    private final int valor1=3;  
    private final int valor2;  
  
    public FinalTest(int valor2) {  
        super();  
        this.valor2 = valor2;  
    }  
  
    public void metodoUno(final int valor3) {  
        System.out.println(valor1);  
        System.out.println(valor2+valor3);  
    }  
  
    public void metodoDos() {  
        final int valor3=5;  
        valor3=9;  
    }  
}
```

# Unidad 6: Paquetes, static y final

## Atributos finales

¿Cuál de estos programas compila sin errores?

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

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

# Unidad 6: Paquetes, static y final

## Atributos finales

¿Cuál de estos programas compila sin errores?

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

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

# Unidad 6: Paquetes, static y final

## Atributos finales

¿Cuál de estos programas compila sin errores?

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

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

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

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

# Definición de constantes



# Unidad 6: Paquetes, static y 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.

# Unidad 6: Paquetes, static y final

## 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 una interface.