



# Java

## Tema 6: Uso avanzado de clases

---

### Ampliación de clases

1



### Contenidos

---

1. Atributos y métodos estáticos
2. Envolveres de tipo (wrapper class)
3. Excepciones

2



# Java

## Atributos y métodos estáticos

3



### El modificador static: características

- ❑ Los atributos y métodos de una clase precedidos de la palabra **static** se denominan **elementos de clase**.

```
public class Prueba{ private static int variable; ...}
```

- ❑ Son compartidos entre todos los objetos/instancias de la clase.
- ❑ Se pueden usar sin crear un objeto de la clase.

```
NombreClase.metodoStatic() --> Math.random()
```

```
NombreClase.atributostatic --> Math.PI
```

- ❑ Los métodos **static** tiene varias restricciones:
  - Sólo pueden llamar a otros métodos **static**
  - No pueden tener la referencia **this** ni **super**
  - No pueden modificar variables de instancia
- ❑ Las variables **static** se inicializan solo una vez, al inicio de la ejecución, y antes que las de instancia.

4

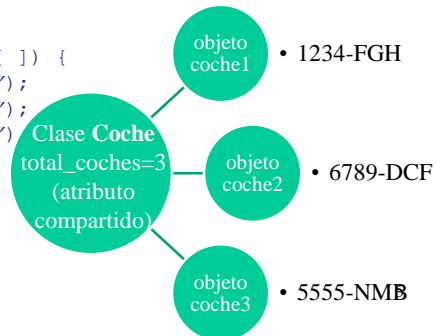


## static: Uso como contador de objetos

```
public class Coche{
    public String matricula, modelo;
    public static int total_coches;

    public Coche(){
        total_coches++; //sirve como contador de n° de objetos coche
    }
}
```

```
public class GestionCoches {
    public static void main (String args[ ]) {
        Coche coche1 = new Coche ("1234-FGH");
        Coche coche2 = new Coche ("6789-DCF");
        Coche coche3 = new Coche ("5555-NMB")
    }
}
```



## static: uso como identificador de objetos

```
public class Coche{
    public String matricula, modelo;
    public int num_serie;
    public static int total_coches;

    public Coche(){
        total_coches++; //sirve como contador de n° de objetos coche
        num_serie = total_coches; // aprovechamos para almacenar
                                   // ese valor en otra variable
    }
}
```

```
public class GestionCoches {
    public static void main (String args[ ]) {
        Coche c1 = new Coche();
        Coche c2 = new Coche();
        System.out.println("Total de coches: " + Coche.total_coches);
        System.out.println("N° serie c1: " + c1.num_serie);
        System.out.println("N° serie c2: " + c2.num_serie);
    }
}
```

Output - TEMA6 ejemplos fase A (run) X

```
run:
Total de coches: 2
N° serie c1: 1
N° serie c2: 2
```

6



## El modificador static

---

Ejemplo:

```
public class Matematicas {
    public static double sumar(double a, double b) {
        return a+b;
    }
    public static double restar(double a, double b) {
        return a-b;
    }
}

public class UsoDeStatic {
    public static void main (String args[ ]) {
        //se pueden usar sin crear un objeto
        System.out.println("suma= " + Matematicas.sumar(3.0,2.0) );
        System.out.println("restar= "+ Matematicas.restar(6.0,3.0)
    );
    }
}
```

7



## El modificador static

---

¿Cuándo tiene sentido usar métodos `static`?

**Cuando hacemos clases que no tienen estado (atributos)  
y tiene métodos sueltos que no comparten información  
(ejemplo Math)**

**O**

**Cuando solo necesitamos una instancia de la clase  
(ejemplo System)**

Ejemplo:

```
java.lang.Math → Math.pow(b,e)
java.lang.System → System.getProperty(String)
```

8



# Java

## Envolventes de tipo (wrapper class)

9



## Envolventes de tipo

- ❑ Las envolventes de tipo son clases del api de java, que **encapsulan un tipo primitivo dentro de un objeto**.
- ❑ Se encuentran en el package java.lang, no es necesario hacer import.
- ❑ Las envolventes de tipo son:

**Character, Boolean, Byte, Short, Integer, Long, Float y Double.**

Tipo primitivo

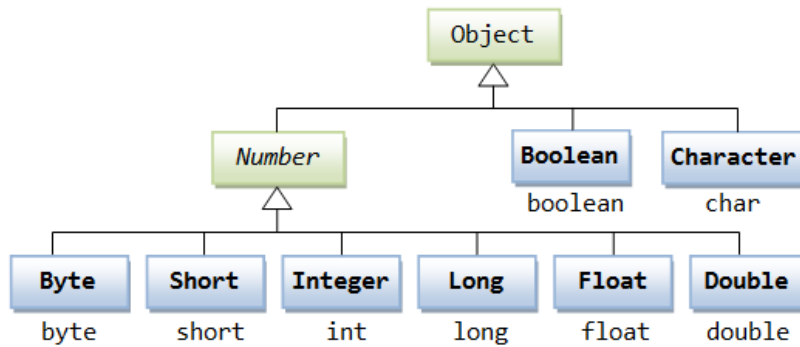
Objetos

<code>int i = 6;</code>	→	<code>Integer i1 = 6;</code>
		<code>Integer i2 = new Integer("6");</code>
<code>long l = 34;</code>	→	<code>Long l1 = 34L;</code>
<code>float f = 3.7f;</code>	→	<code>Float f1 = 3.7f;</code>
<code>double d = 65.45;</code>	→	<code>Double d1 = 65.45;</code>
<code>char c = 'a';</code>	→	<code>Character c1 = 'a';</code>

10



## Envoltentes de tipo



11



## Envoltentes de tipo

❑ ¿Para que se usan las envoltentes de tipos?

1) Para **convertir** cadenas de texto a tipos primitivos y viceversa

2) Para poder **pasar cualquier tipo de datos numérico** a un método

*Ejemplo: Métodos de suma con todas las combinaciones de tipos de datos*

```
int sumar(int a,int b);
long sumar(long a, long b);
long sumar(long a, int b);
long sumar(int a, long b);
float sumar(float a, float b);
double sumar(double a, double b);
double sumar(double a, float b);
double sumar(float a, double b);
.....
float sumar(float a, int b);
float sumar(float a, long b);
```

12



## Envolventes de tipo

---

- ❑ ¿No sería más cómodo hacer esto?

```
Number sumar(Number a, Number b);
```

**Number** es una clase de la que heredan todas las clases envolventes numéricas (Integer, Long, Float, Double...)

- ❑ Es posible, pero la función anterior viene a decir que se puede pasar cualquier objeto con tal de que sea un:

```
Integer  
Long  
Float  
Double
```

13



## Envolventes de tipo

---

- ❑ El **Autoboxing** es una funcionalidad de Java que convierte automáticamente una variable cuyo tipo de datos es primitivo a su envolvente correspondiente, y viceversa.

```
int num1 = 4;  
Integer num2 = new Integer(5);
```

```
int      resultadoA = num2; //es Integer y se convierte a int  
Integer resultadoB = num1; //es int y se convierte a Integer
```

```
        (int)  (Integer)  
resultadoA = num1 + num2;  
resultadoB = num1 + num2;
```

14



## Envolventes de tipo

### ❑ Métodos mas utilizados

Método	Descripcion
String <b>toString()</b>	Convierte el valor almacenado a String
int <b>intValue()</b> double <b>doubleValue()</b> float <b>floatValue()</b> .....	Convierte el valor almacenado al tipo de datos primitivo indicado( int, double, float)
boolean <b>equals</b> (Object o)	Indica si el valor contenido es igual al parámetro o
static Integer <b>valueOf</b> (String i) statit Double <b>valueOf</b> (String i) .....	Convierte el parámetro String a objeto Integer Convierte el parámetro String a objeto Double
static int <b>parseInt</b> (String i) static double <b>parseDouble</b> (String i) .....	Convierte el parámetro String a un int Convierte el parámetro String a un double

15



## Envolventes de tipo: ejemplos conversión

```
String cadena = "54321";  
Integer num1 = Integer.valueOf(cadena);  
int      num2 = Integer.parseInt(cadena);  
double   num3 = Double.parseDouble(cadena);
```

```
String cadena = "123.55";  
Double  num4 = Double.valueOf(cadena);  
double  num5 = Double.parseDouble(cadena);
```

```
Integer n = 12  
String  texto  = n.toString();  
int     entero  = n.shortValue();  
double  decimal = n.doubleValue();
```

16





## Envolventes de tipo: clase Character

---

- ❑ Métodos static mas utilizados de la clase Character:

```
boolean digito = Character.isDigit('2');  
boolean digito2 = Character.isDigit('a');  
  
boolean letra = Character.isLetter('b');  
boolean letra2 = Character.isLetter('2');  
  
System.out.println(Character.toUpperCase('a'));
```

17



# Java



---

## Excepciones

18



## Excepción

---

- ❑ Una **excepción** es un objeto que se genera automáticamente cuando se produce un error durante la ejecución de un programa, y puede ser previsto y controlado.
- ❑ Las excepciones permiten:
  - Encapsular los errores dentro de las clases
  - Separar el flujo de ejecución normal del tratamiento de errores.
- ❑ Las excepciones pueden ser:
  - ❑ **tratadas-capturadas (try-catch)**
  - ❑ **lanzadas (throw)** en los programas.

19



## Tipos de excepción

---

- ❑ **Checked:**
  - Su **tratamiento es obligatorio** y el compilador así lo comprueba.
  - Todas aquellas clases descendientes de **Exception**, menos **RuntimeException**
  - *Ejemplo:* **FileNotFoundException** generada por la clase **FileInputStream**
- ❑ **Unchecked:**
  - Su **tratamiento NO es obligatorio** y el compilador no lo comprueba.
  - Todas aquellas clases descendientes de **RuntimeException**

20



## Captura y manejo de excepciones

- ❑ Se capturan y manejan mediante los bloques **try...catch**:

```
try {  
    // código que puede generar una excepción  
} catch (Exception ex) {  
    // Tratamiento de la excepción  
}
```

21



## Captura y manejo de excepciones

- ❑ Tratamiento de una excepción.

Ejemplo: Se puede dar una excepción si  $b=0$  y  $a=0$

```
public double dividir(int a,int b) {  
    try {  
        return a/b;  
    } catch (ArithmeticException ex) {  
        if (a>0) {  
            return Double.POSITIVE_INFINITY;  
        } else if (a<0) {  
            return Double.NEGATIVE_INFINITY;  
        } else {  
            // lanzamos otra excepción  
            throw new RuntimeException("Indeterminación 0/0",ex);  
        }  
    }  
}
```

Ya no se puede tratar 0/0 porque es una indeterminación, así que lanzamos otra excepción

22



## Captura y manejo de excepciones

- ❑ Si no sabemos como tratar una excepción lo mejor es relanzarla con el tipo `RuntimeException`

```
try {  
    // código que puede generar excepción  
} catch (Exception ex ) {  
    // relanzamos la excepción  
    throw new RuntimeException(ex);  
}
```

23



## Creacion de excepciones

1. Podemos crear nuestras propias excepciones

```
public class MiExcepcion extends RuntimeException{  
    public MiExcepcion(){ super("Error generado"); }  
    public MiExcepcion(String mensaje){  
        super("codigo " + mensaje + " no permitido");  
    }  
}
```

2. Lanzar nuestra excepción cuando lo consideremos necesario:

```
public class Validar {  
    public static void esValido(String codigo) {  
        if (codigo.equals("B100") || codigo.equals("C200") )  
            throw new MiExcepcion(codigo);  
    }  
}
```

24



## Creacion de excepciones

3. Controlar y capturar la excepción que nosotros hemos lanzado en el lugar adecuado:

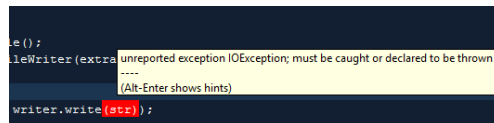
```
public class Test {  
    public static void main ( String args[ ] ) {  
        String codigo="B100";  
        try {  
            Validar.esValido(codigo);  
        } catch (MiExcepcion e) {  
            System.out.println(e.getMessage());  
        }  
    }  
}
```

25



## Creación de excepciones: sentencia throws

- ❑ Si nuestra excepción **hereda de Exception** (se convertirá en checked y será obligado su control), Java nos dará dos opciones:



- ❑ **Opción A:** Poner el try catch (*must be caught*)

```
public static void esValido(String codigo) {  
    try{  
        if (codigo.equals("B100")) throw new MiExcepcion(codigo);  
    } catch (MiExcepcion e){ ... }  
}
```

- ❑ **Opción B:** ó añadir **throws** en el encabezado del método (*declared to be thrown*), para que este avise cuando lo utilicen.

```
public static void esValido(String codigo) throws MiExcepcion {  
    if (codigo.equals("B100")) throw new MiExcepcion(codigo);  
}
```

26