

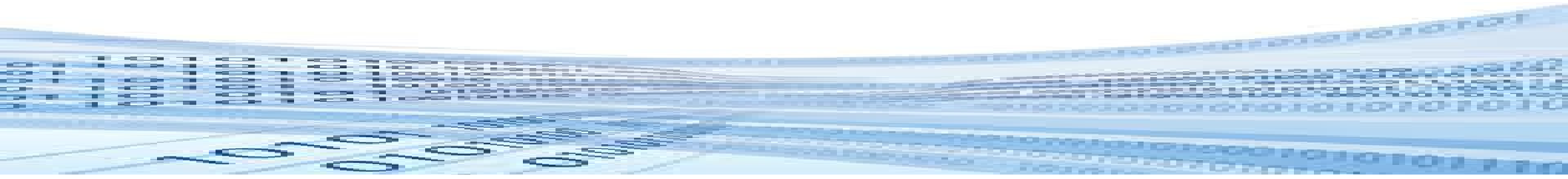
# UD8.- Programación Orientada a Objetos I

Módulo: Programación  
1.º DAM



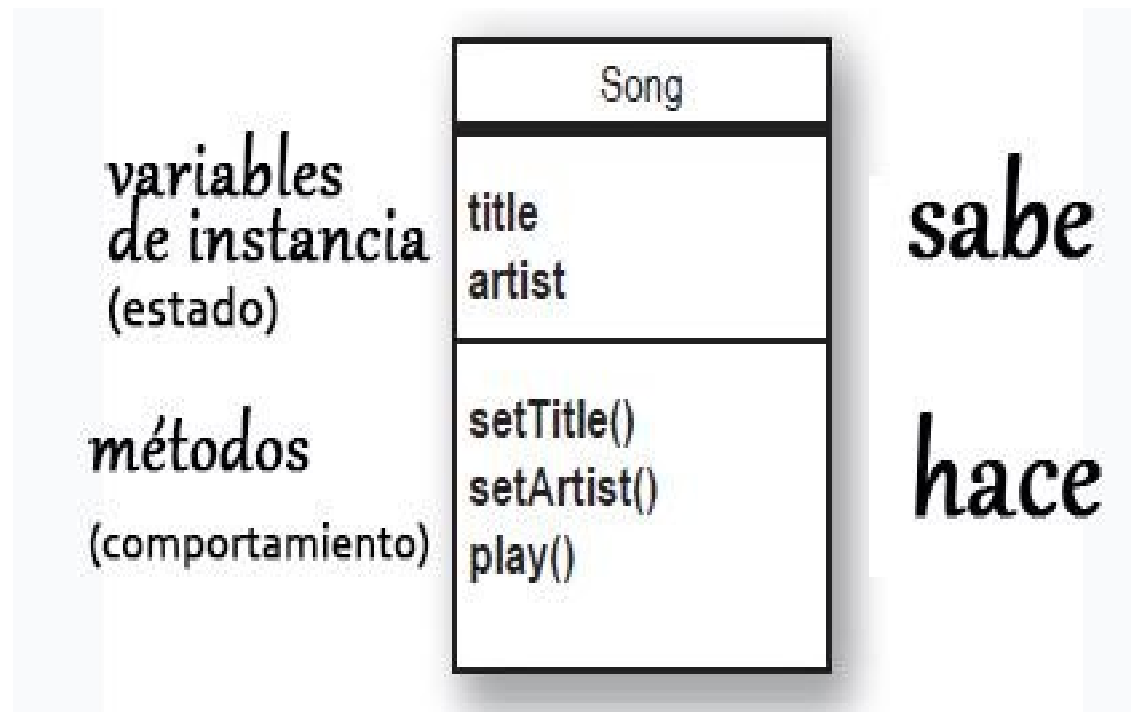
# CONTENIDOS

- Comportamiento de los objetos
- Crear una clase
- Agregar atributos
- Agregar métodos
- Crear instancias de una clase
- Los constructores
- Los destructores
- La referencia *this*
- Miembros (atributos y métodos) genéricos (*static*)



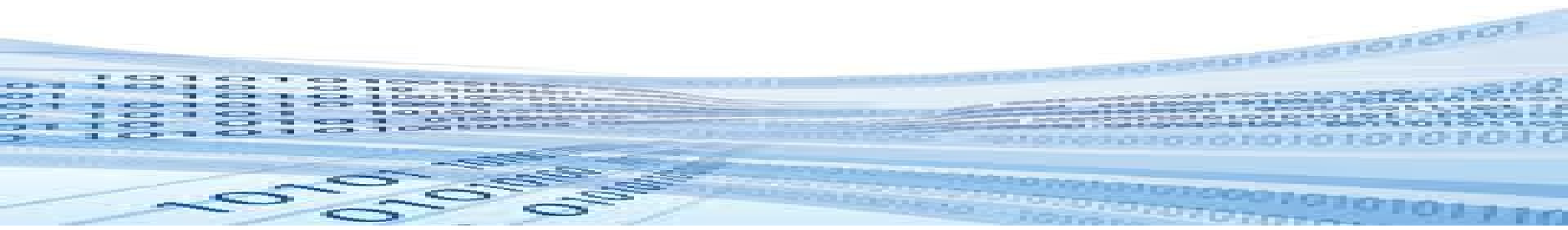
# Comportamiento de los objetos (I)

- Recuerda: una clase es una “plantilla” para los objetos



# Comportamiento de los objetos (II)

- **Estado = variables de instancia**
- **Comportamiento = métodos**
- El estado de un objeto afecta al comportamiento.
- El comportamiento afecta en el estado.
- Cada instancia de una clase, cada objeto, tiene un estado diferente.
- Los métodos utilizan las variables de instancia (también pueden emplear otras variables locales) y según ellas, actúan y también las modifican.



# Variables de instancia y variables locales (I)

- 1 Las variables de instancia son declaradas dentro de la clase pero no dentro del método:

```
class Horse {  
    private double height = 15.2;  
    private String breed;  
    // more code...  
}
```

- 2 Las variables locales son declaradas dentro de un método:


```
class AddThing {  
    int a;  
    int b = 12;  
  
    public int add() {  
        int total = a + b;  
        return total;  
    }  
}
```

# Variables de instancia y variables locales (II)

**3** ¡Las variables locales DEBEN inicializarse antes de usarse!

```
class Foo {  
    public void go() {  
        int x;  
        int z = x + 3;  
    }  
}
```

¡Esto no compila! Podemos declarar una variable sin valor, pero tan pronto como intentes usarla, el compilador dará un error.



# Comportamiento de los objetos (III)

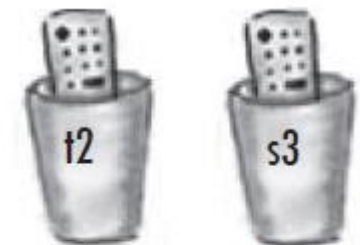


```
void play() {  
    soundPlayer.playSound(title);  
}
```



5 instancias de la clase Song

```
Song t2 = new Song();  
Song s3 = new Song();
```



Song Song

2 variables de referencia a objetos de tipo Song

# Comportamiento de los objetos (IV)

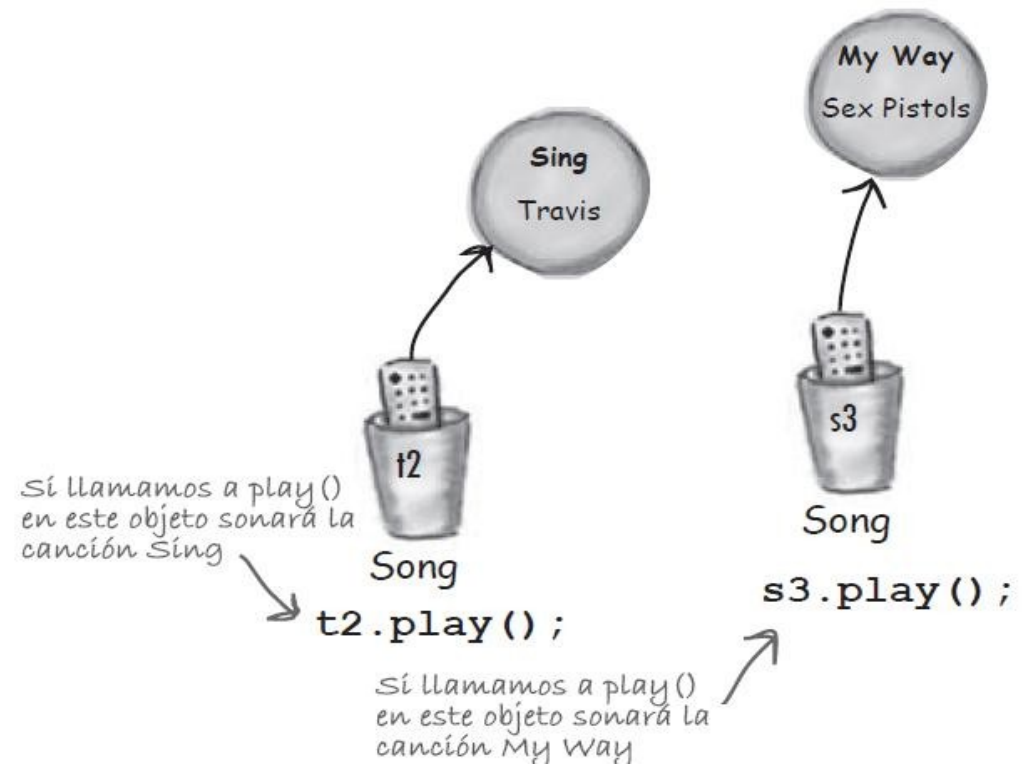


```
t2.setArtist("Travis");
```

```
t2.setTitle("Sing");
```

```
s3.setArtist("Sex Pistols");
```

```
s3.setTitle("My Way");
```





# Comportamiento de los objetos (V)

| Dog          |
|--------------|
| size<br>name |
| bark()       |

```
class Dog {  
    int size;  
    String name;  
  
    void bark() {  
        if (size > 60) {  
            System.out.println("Woof! Woof!");  
        } else if (size > 14) {  
            System.out.println("Ruff! Ruff!");  
        } else {  
            System.out.println("Yip! Yip!");  
        }  
    }  
}
```

En función del tamaño (size),  
el ladrido será diferente.  
El método varía según el estado.

# Crear una nueva clase (I)

- Sintaxis

```
[modificador_acceso] class NomClasse {  
    //atributos de la clase  
    //métodos de la clase  
}
```

- Recordad que una clase debe ser creada en un fichero con el nombre NomClasse.java
- En un fichero puede haber más de una clase, pero solo una con el modificador *public*.

# Modificadores de acceso de clases

- Java tiene 4 modificadores de acceso a las clases:

| Modificador          | Definición   |
|----------------------|--|
| <i>public</i>        | La clase es accesible desde otros <i>packages</i> .                                    |
| <i>(por defecto)</i> | La clase será visible en todas las clases declaradas en el mismo <i>package</i> .      |
| <i>abstract</i>      | Las clases no pueden ser instanciadas. Sirven para definir subclases. Ya lo veremos... |
| <i>final</i>         | Ninguna clase puede heredar de una clase final. Ya lo veremos...                       |

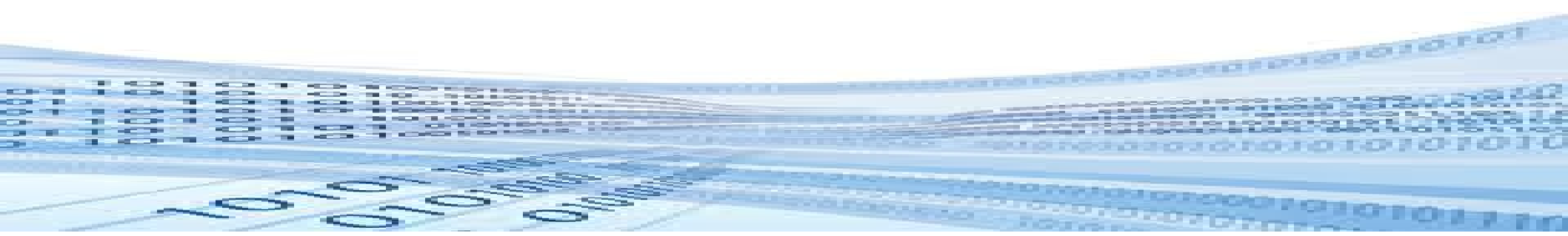
# ¿Cómo agregar atributos?

- Sintaxis general

```
[modificadorÁmbito] [static][final][transient][volatile] tipo nombre_atributo
```

- De momento, versión reducida

```
[modificadorÁmbito] [static][final] tipo nombre_atributo
```



# Como agregar atributos. Ejemplos

- Los atributos se recomienda que siempre sean *private*

```
public class Persona {  
    private String nombre;  
    private int edad;  
    ...  
}
```

```
public class Triangulo {  
    private int lado1, lado2, lado3;  
    ...  
}
```

# Modificadores de acceso de clases internas

- Java tiene 7 modificadores de acceso a las clases internas:

| Palabra clave    | Definición   |
|------------------|--|
| <i>public</i>    | La clase es accesible desde otros <i>packages</i> .                                    |
| (por defecto)    | La clase será visible en todas las clases declaradas en el mismo <i>package</i> .      |
| <i>final</i>     | Ninguna clase puede heredar de una clase final. Ya lo vemos...                         |
| <i>abstract</i>  | Las clases no pueden ser instanciadas. Sirven para definir subclases. Ya lo veremos... |
| <i>private</i>   | La clase solo es visible en el archivo donde está definida                             |
| <i>protected</i> | La clase será visible en todas las clases declaradas en el mismo <i>package</i> .      |
| <i>static</i>    | Las clases no pueden ser instanciadas. Sirven para definir subclases. Ya lo veremos... |

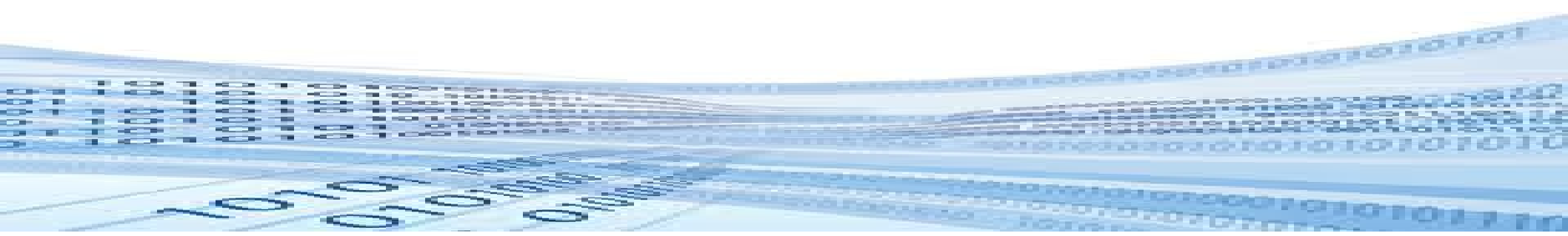
# ¿Cómo agregar métodos? (I)

- Sintaxis general

```
[modificadorÁmbito] [static][abstract][final][native][synchronized]  
    tipo_devuelto nombreMétodo ([listaParámetros])  
[throws llistaExcepciones]
```

- De momento, versión reducida

```
[modificadorÁmbito] [static]  
    tipo_devuelto nombreMétodo ([listaParámetros])
```



# Modificadores de acceso de atributos y métodos (I)

- Java tiene 4 modificadores de acceso que califican a atributos y métodos:

| Palabra clave                   | Definición  |
|---------------------------------|---|
| <i>public</i>                   | El elemento es accesible desde cualquier lugar.   |
| <i>protected</i>                | El elemento es accesible dentro del <i>package</i> donde está definido y, además, en las subclases. |
| <i>package</i><br>(por defecto) | El elemento solo es accesible dentro del <i>package</i> donde está definido.                        |
| <i>private</i>                  | El elemento solo es accesible dentro del fichero en el cual está definido.                          |



# Modificadores de acceso de atributos y métodos (I)

| Palabra clave                           | Fichero | Package<br>(directorio) | Subclase<br>(mismo package) | Subclase<br>(diferente package) | Todos |
|---|---------|-------------------------|-----------------------------|---------------------------------|-------|
| <i>public</i>                           | SÍ      | SÍ                      | SÍ                          | SÍ                              | SÍ    |
| <i>protected</i>                        | SÍ      | SÍ                      | SÍ                          | SÍ                              | NO    |
| <i>por defecto</i><br>(package-private) | SÍ      | SÍ                      | SÍ                          | NO                              | NO    |
| <i>private</i>                          | SÍ      | NO                      | NO                          | NO                              | NO    |

# Crear instancias de una clase (II)

- Para crear un objeto, utilizamos la palabra reservada **new** seguida de un método que se llama igual que la clase (el **constructor**)

```
variable=new Clase();
```

- Así conseguimos una variable que apunta al objeto creado.
- Podemos declarar el objeto y después crearlo:

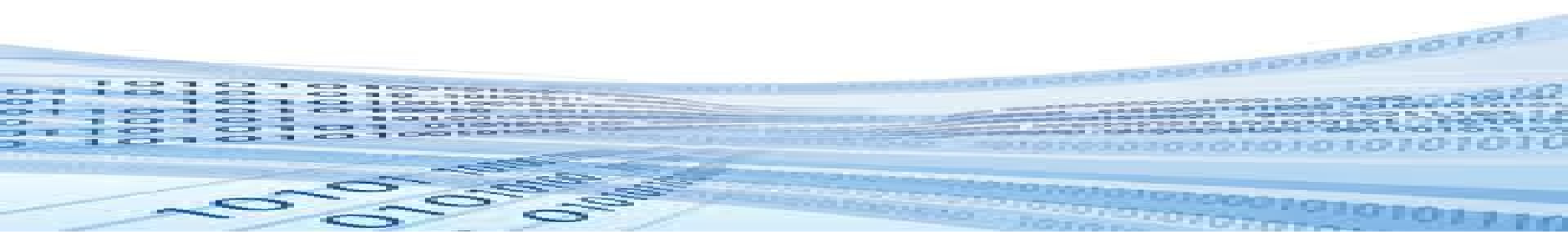
```
Persona cliente1, cliente2;
```

```
cliente1=new Persona();
```

```
cliente2=new Persona();
```

- Podemos declarar y crear el objeto al mismo tiempo:

```
Persona cliente1=new Persona();
```



# ¿Cómo agregar métodos? (II)

- Recordad que los métodos pueden estar sobrecargados
  - Dos o más métodos con el mismo nombre pero con una lista de parámetros diferente:
    - Distinto número de parámetros o
    - Al menos un parámetro de tipo diferente
- La sobrecarga de métodos hacen que el método sea más flexible para los usuarios del método.

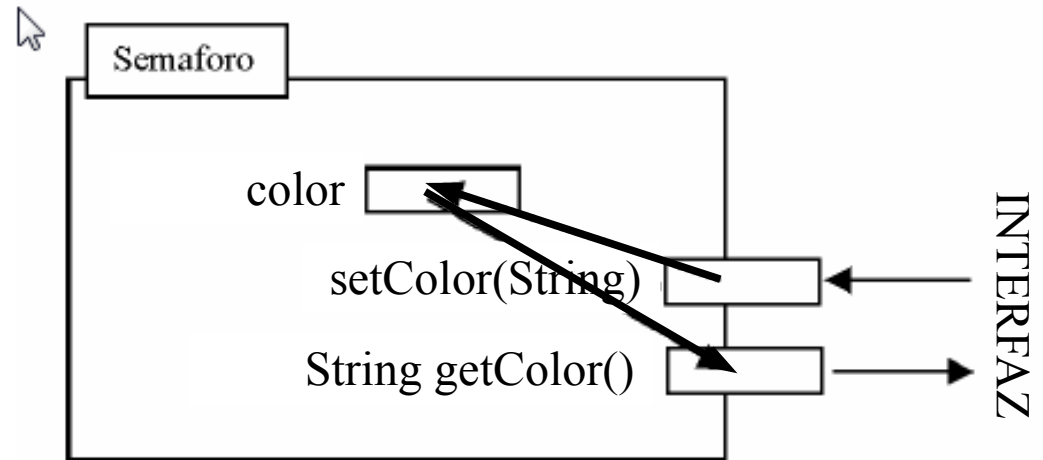


# Cómo agregar métodos. Ejemplo

```
public class Triangulo{  
  
    private int lado1, lado2, lado3;  
    ...  
  
    public void esEquilatero() {  
        if (lado1==lado2) && (lado2==lado3)  
            System.out.println("Es equilátero");  
        else  
            System.out.println("No es equilátero");  
    }  
}
```

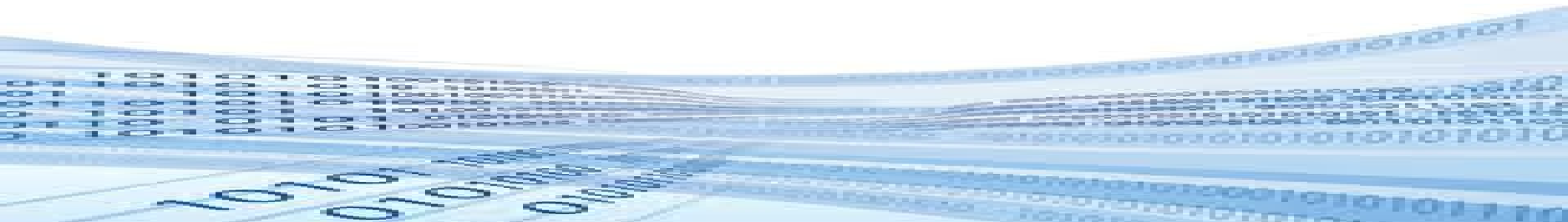
# Cómo agregar métodos. Ejemplo

```
public class Semaforo {  
    private String color;  
  
    public Semaforo(String unColor) {  
        color = unColor;  
    }  
  
    public String getColor() {  
        return color;  
    }  
  
    public void setColor(String otroColor) {  
        color = otroColor;  
    }  
}
```



# Crear instancias de una clase (I)

- Cuando creamos una clase, estamos definiendo una plantilla con la cual se definirá aquello que los objetos saben (atributos o variables de instancia) y lo que hacen (métodos). Una vez hecho esto, se puede instanciar la clase (crear objetos).
- Para declarar un objeto de una clase:  
    tipo variable;
- Con esto tenemos un apuntador capaz de dirigir el objeto, pero **no tenemos el objeto**:
  - De momento la variable no apunta a ningún objeto
  - Se dice que contiene la referencia **null**



# Acceder a los elementos de un objeto

- Para acceder a los **atributos** de un objeto

`objeto.atributo`

Ejemplo: `cliente1.edad;`

**ATENCIÓN:** con los atributos `private`, esto genera error. Después veremos como hacerlo.

- Para acceder a los **métodos** de un objeto

`objeto.metodo()`

Ejemplo: `cliente1.esMayorDeEdad();`

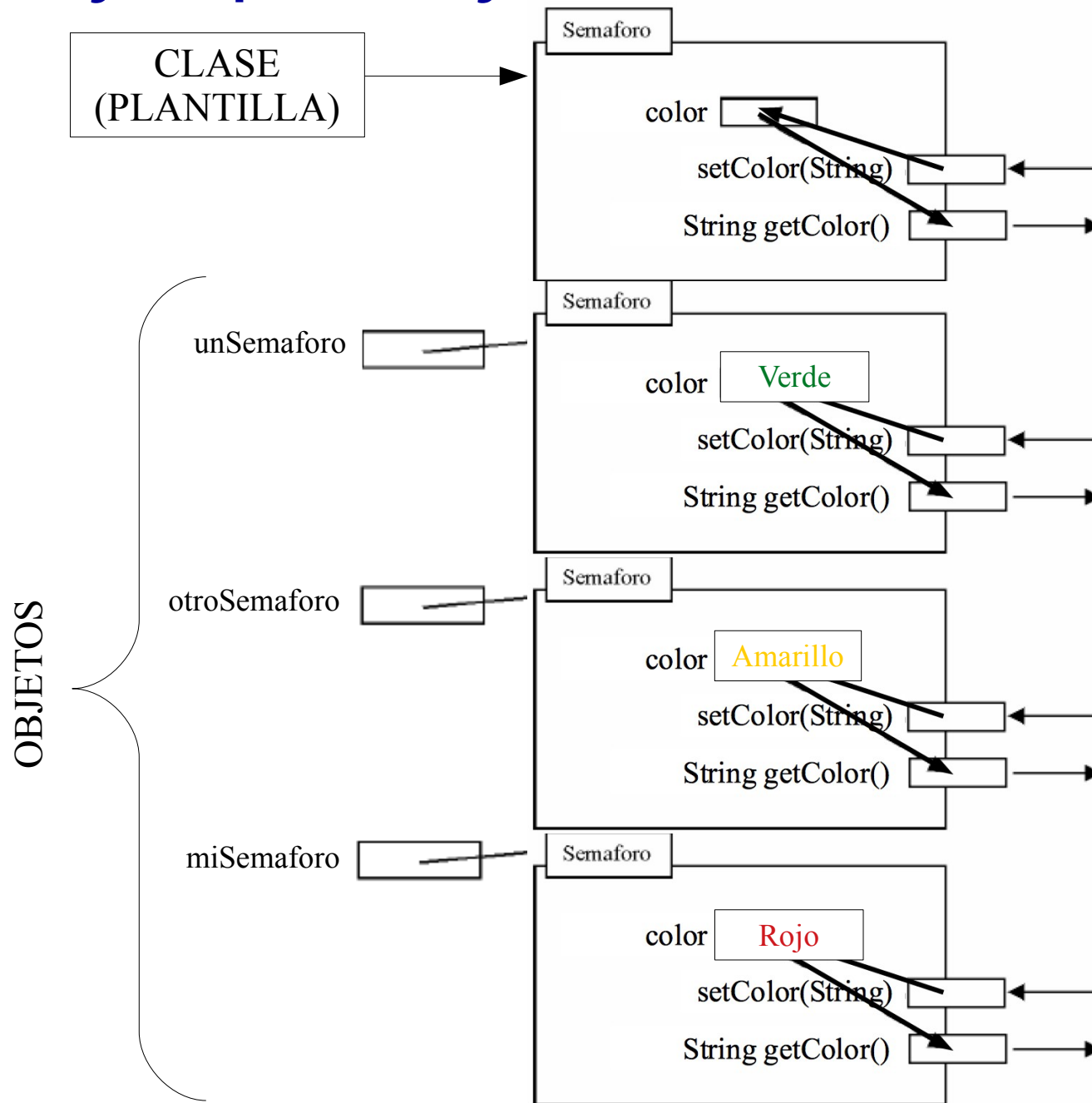
# Ejemplo: objetos de la clase Semaforo

```
public class Semaforo {  
    private String color;  
  
    public Semaforo(String unColor) {  
        color = unColor;  
    }  
  
    public String getColor() {  
        return color;  
    }  
  
    public void setColor(String otroColor) {  
        color = otroColor;  
    }  
}
```

```
public class PruebaSemaforo {  
    public static void main(String[] args) {  
        Semaforo unSemaforo = new Semaforo("Verde");  
        Semaforo otroSemaforo = new Semaforo("Amarillo");  
        Semaforo miSemaforo = new Semaforo("Rojo");  
  
        System.out.println(unSemaforo.getColor());  
        System.out.println(otroSemaforo.getColor());  
  
        if(miSemaforo.getColor().equals("Rojo")) {  
            System.out.println("No pasar");  
        }  
    }  
}
```



# Ejemplo: objetos de la clase Semaforo

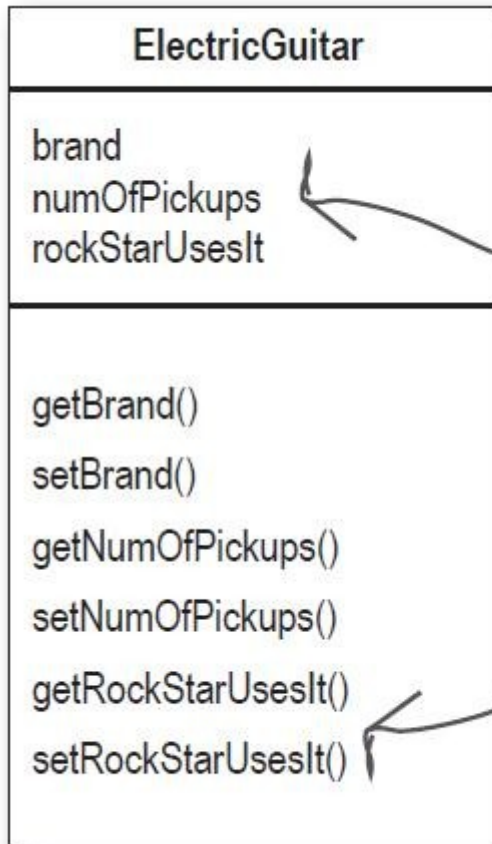


# Setters (modificadores) y Getters (consultores)

- Es una buena práctica (**Y DEBÉIS HACERLO**):
  - Crear los atributos con el modificador ***private***.
  - Crear métodos públicos para acceder a los atributos:
    - Para consultarlos (***getters***)
    - Para modificarlos (***setters***)
  - Desde otras clases externas no se podrán ni acceder ni modificar los atributos si no se hace mediante los *getters* y *setters* .
- Beneficios del **encapsulamiento**:
  - Que nadie acceda por equivocación o sobrescriba funcionalidades cuando no debe.
  - Un programador que utilice un método, solo necesita saber qué hace, no como lo hace (caja negra).



# Setters (modificadores) y Getters (consultores)



Nota: usar estos nombres y convenciones significa que estás siguiendo un importante estándar en Java!

```
class ElectricGuitar {  
  
    String brand;  
    int numOfPickups;  
    boolean rockStarUsesIt;  
  
    String getBrand() {  
        return brand; // devuelve la variable 'brand'  
    }  
  
    void setBrand(String aBrand) {  
        brand = aBrand; // modifica la variable 'brand'  
                        // al valor del parámetro pasado  
    }  
  
    int getNumOfPickups() {  
        return numOfPickups;  
    }  
  
    void setNumOfPickups(int num) {  
        numOfPickups = num;  
    }  
  
    boolean getRockStarUsesIt() {  
        return rockStarUsesIt;  
    }  
  
    void setRockStarUsesIt(boolean yesOrNo) {  
        rockStarUsesIt = yesOrNo;  
    }  
}
```

# Encapsulamiento

```
class GoodDog {
```

```
    private int size;
```

la variable size  
será privada

```
    public int getSize() {
```

```
        return size;
```

```
    }
```

los métodos  
getter y setter  
serán públicos

```
    public void setSize(int s) {
```

```
        size = s;
```

```
    }
```

Aunque pensemos  
que los métodos no  
tienen nuevas  
funcionalidades,  
más tarde podremos  
añadirlas cambiando  
el código si queremos.

```
    void bark() {
```

```
        if (size > 60) {
```

```
            System.out.println("Woof! Woof!");
```

```
        } else if (size > 14) {
```

```
            System.out.println("Ruff! Ruff!");
```

```
        } else {
```

```
            System.out.println("Yip! Yip!");
```

```
        }
```

```
    }
```

```
}
```

```
class GoodDogTestDrive {
```

```
    public static void main (String[] args) {
```

```
        GoodDog one = new GoodDog();
```

```
        one.setSize(70);
```

```
        GoodDog two = new GoodDog();
```

```
        two.setSize(8);
```

```
        System.out.println("Dog one: " + one.getSize());
```

```
        System.out.println("Dog two: " + two.getSize());
```

```
        one.bark();
```

```
        two.bark();
```

```
    }
```

```
}
```

# Encapsulamiento

Cualquier lugar en el que puede ser usado un valor, también puede usarse una llamada a un método que devuelve ese tipo

En vez de:

```
int x = 3 + 24;
```

puedes usar:

```
int x = 3 + one.getSize();
```



# Setters y Getters. Ejemplo

```
public class Punto {  
  
    private int x, y;  
  
    public void setCoordenadas(int a, int b) {  
        x = a; // o bien this.x=a;  
        y = b; // o bien this.y=b;  
    }  
  
    public void setCoordenadaX(int a) {  
        x = a; // o bien this.x=a;  
    }  
  
    public void setCoordenadaY(int a) {  
        y = a; // o bien this.y=a;  
    }  
  
    public int getCoordenadaX() {  
        return x;  
    }  
  
    public int getCoordenadaY() {  
        return y;  
    }  
}
```

```
package punto2;  
import java.util.Scanner;  
public class PuntoApp {  
    public static void main(String[] args) {  
        int coorX, coorY;  
        Punto punto1;  
        punto1=new Punto();  
        Scanner teclado=new Scanner(System.in);  
        System.out.print("Ingrese coordenada x :");  
        coorX=teclado.nextInt();  
        // punto1.x=coorX; dona error per ser private!!!!!!  
        System.out.print("Ingrese coordenada y :");  
        coorY=teclado.nextInt();  
        punto1.setCoordenadas(coorX, coorY);  
  
        System.out.println("Hablamos del punto ( "  
            +punto1.getCoordenadaX()+" , "+punto1.getCoordenadaY()+" )");  
        punto1.imprimirCuadrante();  
    }  
}
```

# Los constructores (I)

- Método especial de una clase que es invocado automáticamente siempre que se crea un objeto, es decir, al utilizar la instrucción new.
- Su función es iniciar el objeto.
- Se recomienda que los constructores inicialicen todas las variables de instancia del objeto

```
public class Rectangulo {  
    ...  
    public Rectangulo(int x1, int y1, int w, int h) {  
        x=x1;  
        y=y1;  
        ancho=w;  
        alto=h;  
    }  
    ...  
}
```

# Los constructores (II)

- Para declarar un constructor, es suficiente con declarar un método con el mismo nombre que la clase.
- No se declara tipo de datos devuelto por el constructor, ni siquiera *void*.
- Si no hay ningún constructor en la clase, Java se inventa uno que no tiene argumentos e inicializa todos los atributos a los valores por defecto.

|             |       |
|-------------|-------|
| enteros     | 0     |
| decimales   | 0.0   |
| booleanos   | falso |
| referencias | null  |

- Java solo se inventa los constructores si no hay ninguno.
- Si hay algún constructor, Java se limita a hacer aquello que el constructor dice.



# Los constructores (III)

- Es posible declarar diferentes constructores (sobrecarga de métodos) al igual que el resto de métodos de la clase.

```
public class Rectangulo {  
    private int x;  
    private int y;  
    private int ancho;  
    private int alto;  
  
    public Rectangulo() {  
        x=0;  
        y=0;  
        ancho=0;  
        alto=0;  
    }  
    public Rectangulo(int x1, int y1, int w, int h) {  
        x=x1;  
        y=y1;  
        ancho=w;  
        alto=h;  
    }  
    public Rectangulo(int w, int h) {  
        x=0;  
        y=0;  
        ancho=w;  
        alto=h;  
    }  
    ...  
}
```

# Los constructores. Ejemplo

```
public class app {  
    public static void main(String[] args) {  
        Rectangulo r1 = new Rectangulo();  
        Rectangulo r2 = new Rectangulo(2,4,8,4);  
        Rectangulo r3 = new Rectangulo(16, 8);  
  
        ...  
    }  
}
```

- Para crear “Rectangulos” podemos utilizar cualquiera de los 3 constructores que hemos definido en la clase (plantilla).

# ¿Destructores?

- En Java existe un recolector de basura (*garbage collector*) que se encarga de gestionar los objetos que se dejan de utilizar y liberar el espacio que ocupan en memoria.
- Este proceso es automático e impredecible y trabaja en un hilo (*thread*) de baja prioridad.
- En términos generales, este proceso de recolección de basura trabaja cuando detecta que algún objeto hace mucho de tiempo que ya no se utiliza en el programa.
- La eliminación depende de la máquina virtual. Normalmente, se realiza de forma periódica.
- Podemos invocar el método estático `System.gc()` para “aconsejar” a la máquina virtual de Java que ejecute el recolector, pero en ningún caso está asegurada su ejecución.

# La referencia *this*

- La palabra reservada ***this*** es una referencia al propio objeto con el cual estamos trabajando.
- Ejemplo:

```
class punto {  
    int posX, posY;//posición del punto  
    punto(posX, posY){  
        this.posX=posX;  
        this.posY=posY;  
    }  
}
```

- En este ejemplo, hace falta la referencia *this* para clarificar cuando se utilizan las propiedades posX y posY y cuando los argumentos con el mismo nombre.

# Atributos *static*

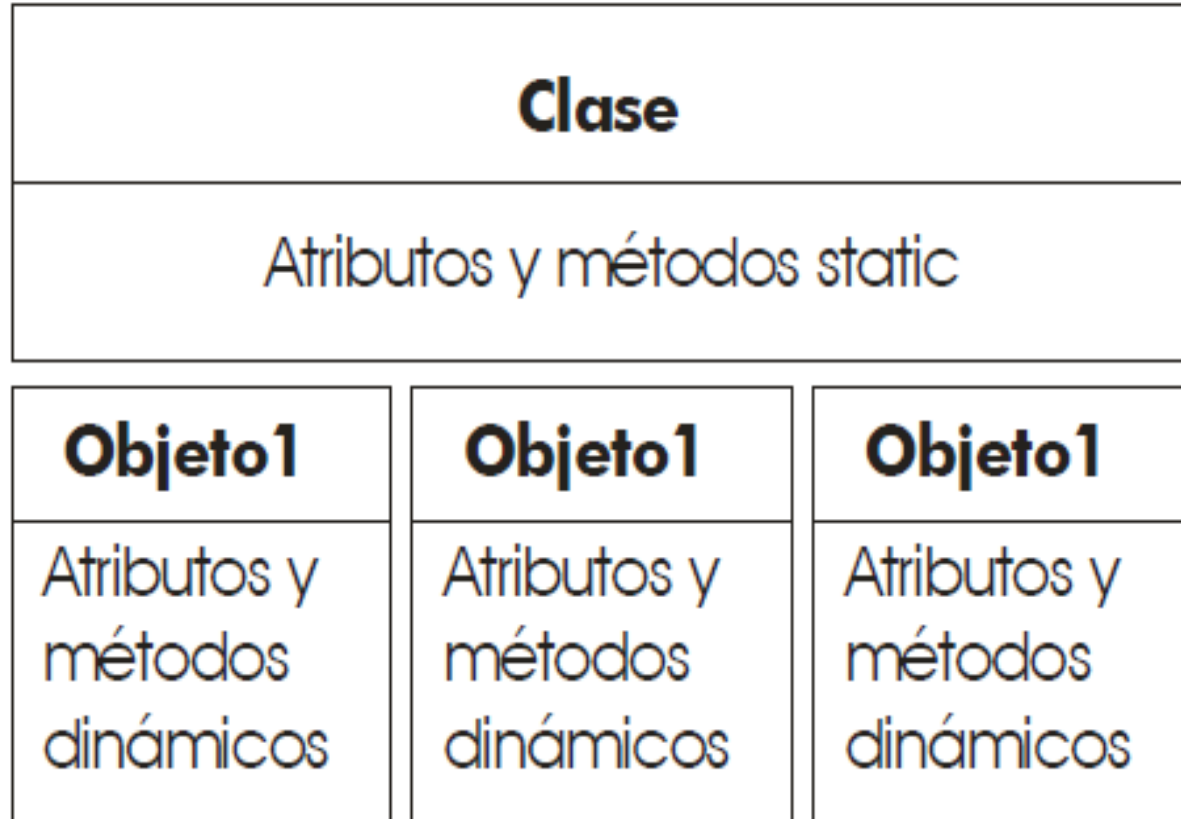
- Existen dos tipos de atributos (variables de instancia):
  - **Atributos de objeto**
    - Son variables u objetos que guardan valores diferentes para instancias diferentes de la clase (para objetos diferentes).
    - Si no se especifica de forma explícita, los atributos son de objeto.
  - **Atributos de clase**
    - Son variables u objetos que guardan el mismo valor para todos los objetos instanciados a partir de la clase.
    - Se declaran con la palabra reservada ***static***.

# Métodos *static*

- Los métodos `static` son métodos de clase.
  - No es necesario crear un objeto de la clase (instanciar la clase) para poder invocar a un método `static`.
  - Se utilizará el nombre de la clase “como si fuera un objeto”.
- Los métodos de clase (`static`) únicamente pueden acceder a sus atributos de clase (`static`) y nunca a los atributos de objeto (dinámicos).
- Hasta ahora, los hemos utilizado:
  - Siempre que se declaraba una clase ejecutable:
    - Para poder ejecutar el método `main()` no se declara ningún objeto de esa clase.
    - Cuándo hemos creado los métodos sin crear objetos en la UD6.



# Atributos y métodos *static*



*Diagrama de funcionamiento de los métodos y atributos static*

# Métodos *static*

- Se crearán métodos y atributos genéricos cuando este método o atributo vale o da el mismo resultado en todos los objetos.
- Se utilizarán métodos normales (dinámicos) cuando el método da resultados diferentes según el objeto.
- Por ejemplo, una clase que represente aviones:
  - la altura sería un atributo dinámico (distinto para cada objeto)
  - el número total de aviones sería un atributo static (el mismo para todos los aviones)



# Agrupación de clases

- Los *packages* son una forma de agrupar varias clases:
  - para estructurar las clases en grupos relacionados
  - para aprovechar la posibilidad de dar a los miembros de una clase la visibilidad a nivel de package.
- Cuando no se especifica el nombre del *package* al cual pertenece una clase, pasa a estar en *el package* por defecto.
- La declaración del *package* se hace al inicio del fichero .java:
  - `package x.y.x;`  
Significa que la clase definida en ese fichero será del *package* x.y.z

# Packages e import

- El nombre completo de una clase es el nombre del *package* en el cual se encuentra la clase, punto y después el nombre de la clase.
  - Ejemplo: si la clase Coche está dentro del *package* locomocion, el nombre completo de Coche es locomocion.Coche
- Mediante la palabra reservada **import** se evita tener que colocar el nombre completo.
- import se coloca antes de definir la clase:
  - Ejemplo: import locomocion.Coche;
- Gracias a esta instrucción se puede utilizar la clase Coche sin necesidad de indicar el *package* donde se encuentra cada vez que deseemos utilizarla.
- Se puede emplear el símbolo \* como comodín.
  - Ejemplo: import locomociom.\*;

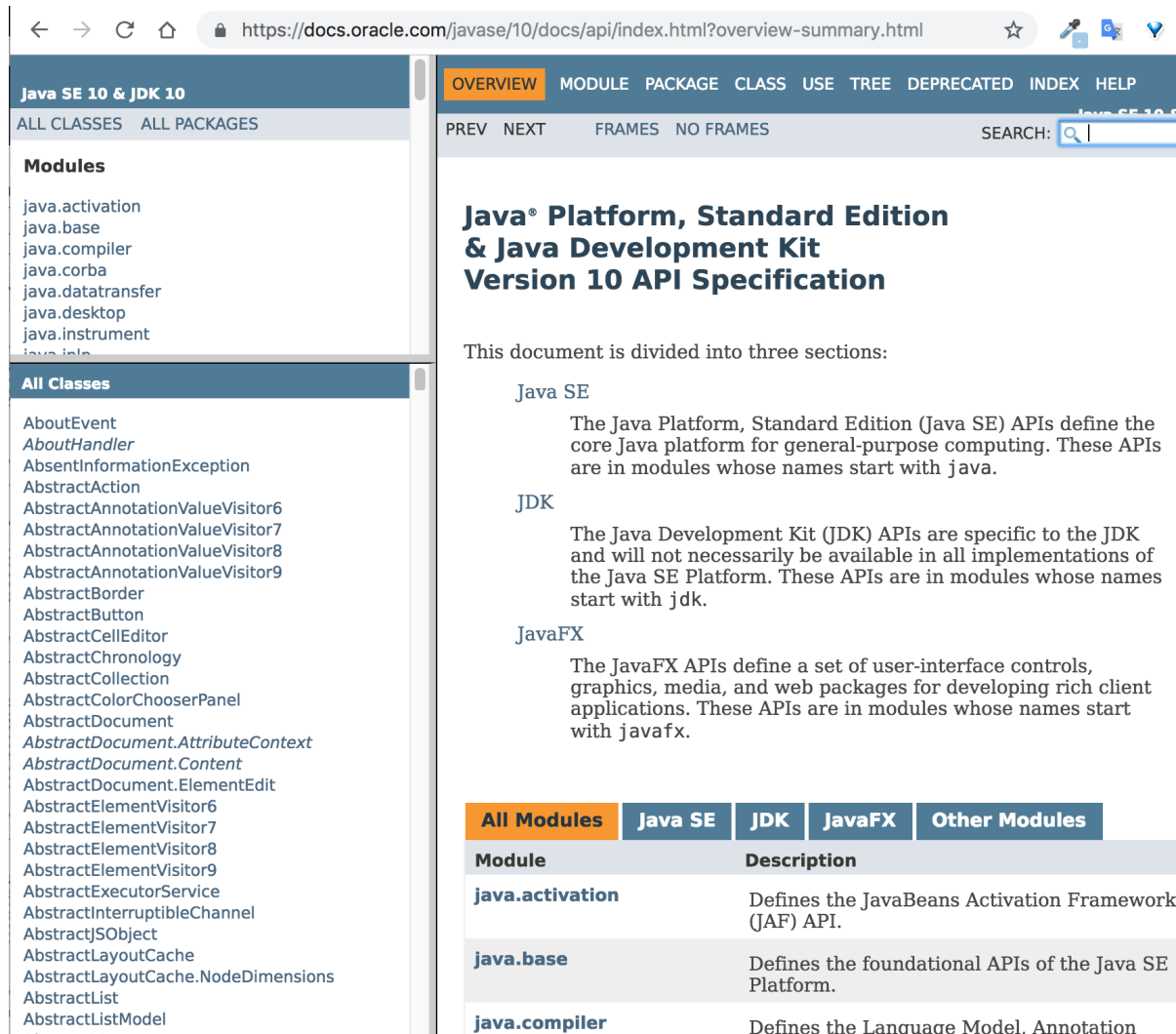
//Importa todas las clases del *package* locomocion

# Librerías de clases (I)

- Hemos visto que en Java hay una gran cantidad de clases ya definidas y utilizables. Vienen agrupadas en packages o librerías estándares:
  - `java.lang` - clases esenciales, números, Strings, objetos, compilador... (es el único package que se incluye automáticamente en todos los programas Java)
  - `java.io` - clases que manejan entradas y salidas
  - `java.util` - clases útiles, como estructuras genéricas, manejo de fecha y hora, números aleatorios, entrada estándar, ...
  - `java.net` - apoyo para redes: URL, TCP, UDP, IP ...
  - `java.awt` - interfaz gráfica, ventanas, imágenes ...
  - `java.applet` - creación de applets

# Librerías de clases (II)

- Recuerda que a <https://docs.oracle.com/javase/10/docs/api/> tienes documentación muy útil sobre el API de Java. CONSÚLTALA!!!



The screenshot shows the Java SE 10 & JDK 10 API documentation page. The left sidebar lists modules and classes. The main content area shows the 'Overview' section with a table of modules.

| All Modules            | Java SE  | JDK | JavaFX | Other Modules |
|------------------------|--|-----|--------|---------------|
| Module                 | Description  |     |        |               |
| <b>java.activation</b> | Defines the JavaBeans Activation Framework (JAF) API.  |     |        |               |
| <b>java.base</b>       | Defines the foundational APIs of the Java SE Platform. |     |        |               |
| <b>java.compiler</b>   | Defines the Language Model, Annotation                 |     |        |               |