

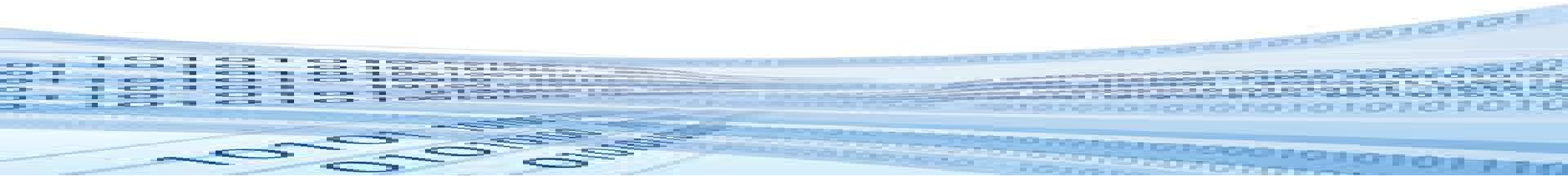
# UD8.- Programació Orientada a Objectes I

Mòdul: Programació  
1r DAM  
Curs 2018-2019



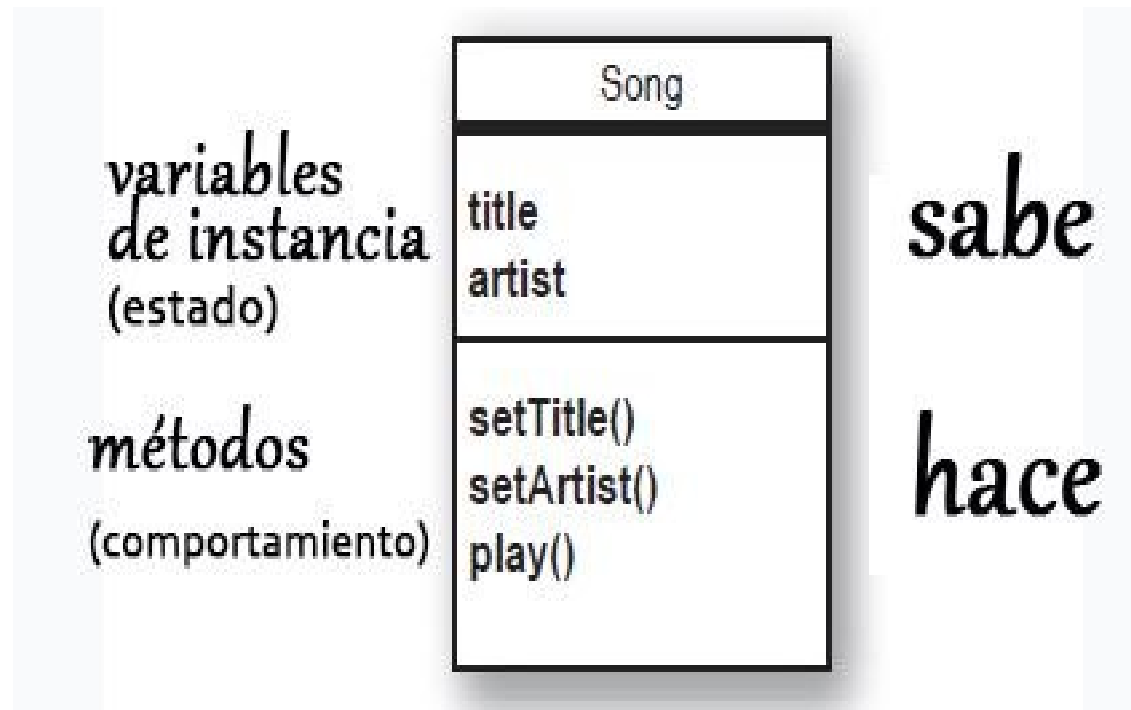
# CONTINGUTS

- Comportament dels objectes
- Crear una classe
- Agregar atributs
- Agregar mètodes
- Crear instàncies d'una classe
- Els constructors
- Els destructors
- La referència *this*
- Membres (atributs i mètodes) genèrics (*static*)



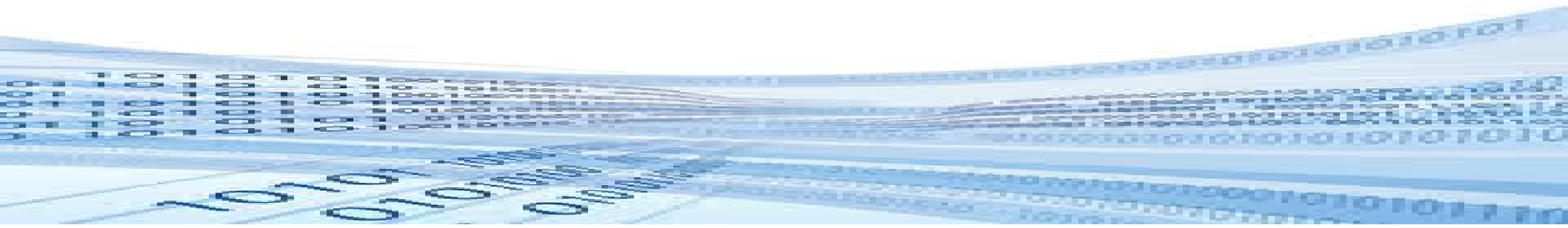
# Comportament dels objectes(I)

- Recorda: una classe és una “plantilla” per als objectes



# Comportament dels objectes(II)

- **Estat = variables d'instància**
- **Comportament = mètodes**
- L'estat d'un objecte afecta al comportament.
- El comportament afecta a l'estat.
- Cada instància d'una classe, cada objecte, té un estat diferent.
- Els mètodes utilitzen les variables d'instància (també poden emprar altres variables locals) i segons elles actuen i també les modifiquen.



# Variables d'instància i variables locals(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 d'instància i variables locals(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.



# Comportament dels objectes(III)

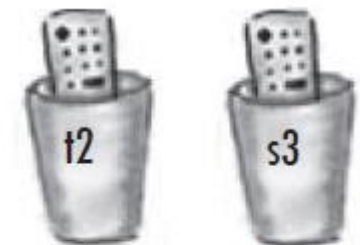


```
void play() {  
    soundPlayer.playSound(title);  
}
```



5 instàncies de la classe Song

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



Song Song

2 variables de referència a objectes de tipus Song

# Comportament dels objectes(IV)

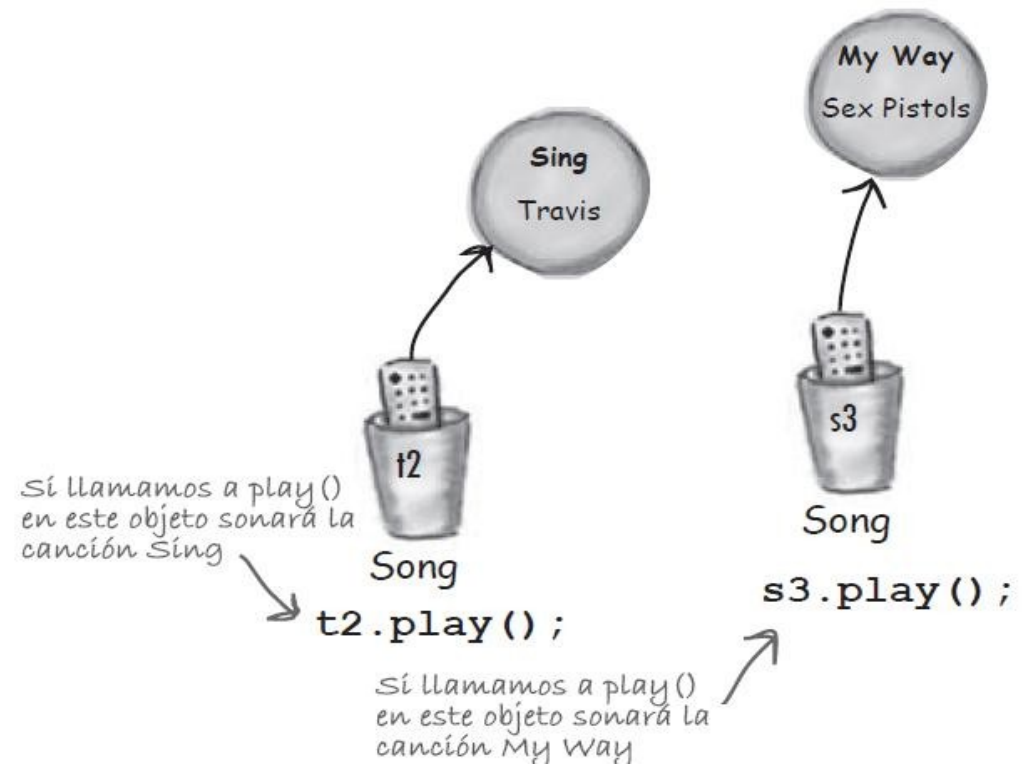


```
t2.setArtist("Travis");
```

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

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

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





# Comportament dels objectes(V)

Dog
size 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 ladrillo será diferente.  
El método varía según el estado.

# Crear una nova classe(I)

- Sintaxis

```
[modificador_accés] class NomClasse  
{  
    //atributs de la classe  
    //mètodes de la classe  
}
```

- Recordeu que una classe ha de ser creada en un fitxer amb el nom NomClasse.java
- En un fitxer pot haver més d'una classe, però sols una amb el modificador *public*.

# Modificadors d'accés de classes

- Java té 4 modificadors d'accés a les classes:

Modificador	Definició
<i>public</i>	La classe és accessible des d'altres <i>packages</i> .
<i>(per defecte)</i>	La classe serà visible en totes les classes declarades en el mateix <i>package</i> .
<i>abstract</i>	Les classes no poden ser instanciades. Servixen per a definir subclasses. Ja ho vorem...
<i>final</i>	Cap classe pot heretar d'una classe final. Ja ho vorem...

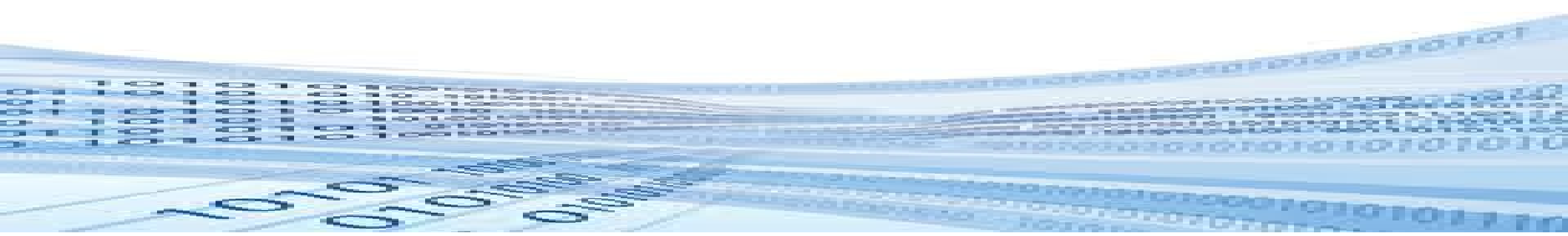
# Com agregar atributs?

- Sintaxis general

```
[modificadorÀmbit] [static][final][transient][volatile] tipus nom_atribut
```

- De moment, versió reduïda

```
[modificadorÀmbit] [static][final] tipus nom_atribut
```



# Com agregar atributs. Exemples

- Els atributs es recomanen que sempre siguin *private*

```
public class Persona{  
    private String nom;  
    private int edat;  
    ...  
}
```

```
public class Triangle{  
    private int costat1, costat2, costat3;  
    ...  
}
```

# Modificadors d'accés de classes internes

- Java té 7 modificadors d'accés a les classes internes:

Paraula clau	Definició
<i>public</i>	La classe és accessible des d'altres <i>packages</i> .
(per defecte)	La classe serà visible en totes les classes declarades en el mateix <i>package</i> .
<i>final</i>	Cap classe pot heretar d'una classe final. Ja ho vorem...
<i>abstract</i>	Les classes no poden ser instanciades. Servixen per a definir subclasses. Ja ho vorem...
<i>private</i>	La classe només es visible en l'arxiu on està definida
<i>protected</i>	La classe serà visible en totes les classes declarades en el mateix <i>package</i> .
<i>static</i>	Les classes no poden ser instanciades. Servixen per a definir subclasses. Ja ho vorem...

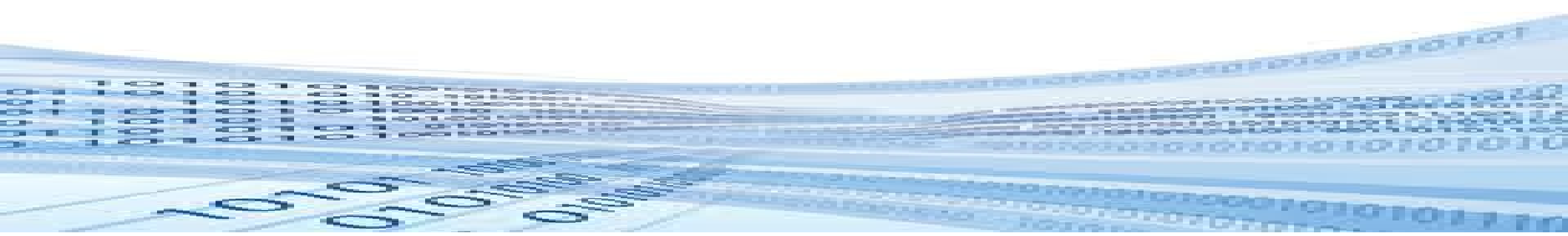
# Com agregar mètodes?(I)

- Sintaxis general

```
[modificadorÀmbit] [static][abstract][final][native][synchronized]  
    tipus_retornat nomMètode ([llistaParàmetres])  
[throws llistaExcepcions]
```

- De moment, versió reduïda

```
[modificadorÀmbit] [static]  
    tipus_retornat nomMètode ([llistaParàmetres])
```



# Modificadors d'accés d'atributs i mètodes(I)

- Java té 4 modificadors d'accés que qualifiquen a atributs i mètodes:

Paraula clau	Definició
<i>public</i>	L'element és accessible des de qualsevol lloc.
<i>protected</i>	L'element és accessible dins del <i>package</i> on està definit i, a més a més, en les subclasses.
<i>package</i> (per defecte)	L'element sols és accessible dins del <i>package</i> on està definit.
<i>private</i>	L'element sols és accessible dins del fitxer en el qual està definit.



# Modificadors d'accés d'atributs i mètodes(II)

Paraula clau	Fitxer	Package (directori)	Subclasse (mateix package)	Subclasse (diferent package)	Tots
<i>public</i>	SI	SI	SI	SI	SI
<i>protected</i>	SI	SI	SI	SI	NO
<i>per defecte</i> ( <i>package-private</i> )	SI	SI	SI	NO	NO
<i>private</i>	SI	NO	NO	NO	NO

# Crear instàncies d'una classe (II)

- Per a crear un objecte, utilitzem la paraula reservada **new** seguida d'un mètode que es diu igual que la classe (el **constructor**)

```
variable=new Classe();
```

- Així aconseguim una variable que apunta a l'objecte creat.
- Podem declarar l'objecte i després crear-lo:

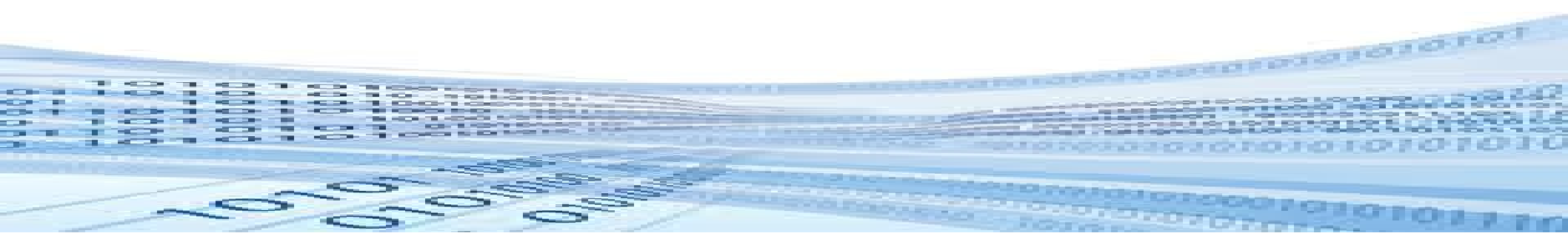
```
Persona client1, client2;
```

```
client1=new Persona();
```

```
client2=new Persona();
```

- Podem declarar i crear l'objecte al mateix temps:

```
Persona client1=new Persona();
```



# Com agregar mètodes?(II)

- Recordeu que els mètodes poden estar sobrecarregats
  - Dos o més mètodes amb el mateix nom però amb una llista de paràmetres diferent:
    - Distint número de paràmetres o
    - Almenys un paràmetre de tipus diferent
- La sobrecàrrega de mètodes fan que el mètode siga més flexible per als usuaris del mètode.

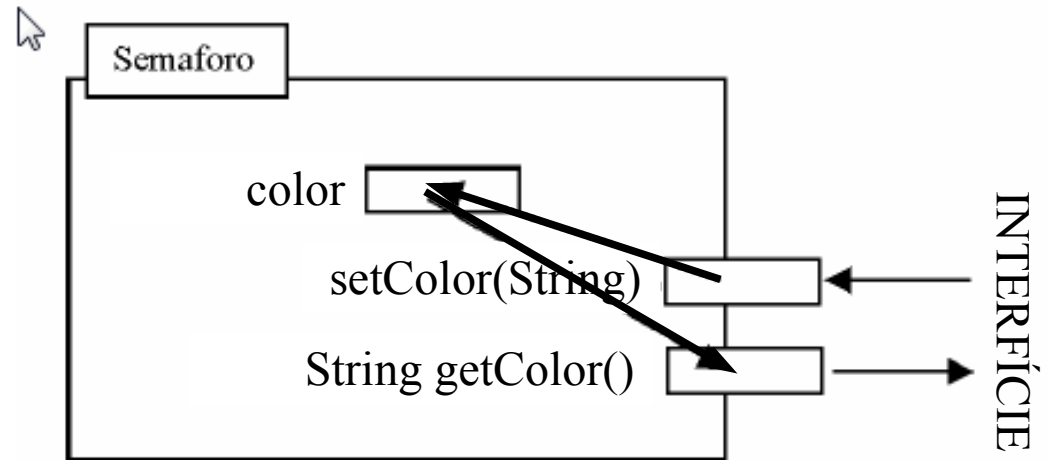


# Com agregar mètodes. Exemple

```
public class Triangle{  
  
    private int costat1, costat2, costat3;  
    ...  
  
    public void esEquilater(){  
        if (costat1==costat2) && (costat2==costat3)  
            System.out.println("Es equilàter");  
        else  
            System.out.println("No és equilàter");  
    }  
}
```

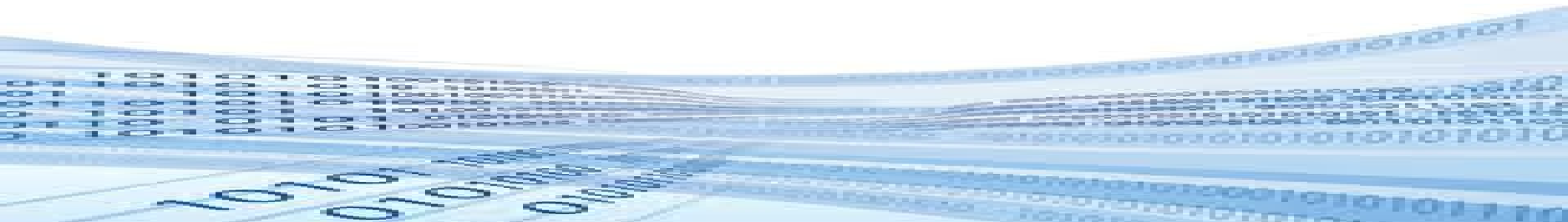
# Com agregar mètodes. Exemple

```
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 instàncies d'una classe (I)

- Quan creem una classe, estem definint una plantilla amb la qual es definirà allò que els objectes saben (atributs o variables d'instància) i el que fan (mètodes). Una vegada fet això, es pot instanciar la classe (crear objectes).
- Per a declarar un objecte d'una classe:  
tipus variable;
- Amb açò tenim un apuntador capaç d'adreçar l'objecte, però **no tenim l'objecte**:
  - De moment la variable no apunta a cap objecte
  - Es diu que conté la referència **null**



# Accedir als elements d'un objecte

- Per accedir als **atributs** d'un objecte

`objecte.atribut`

Exemple: `client1.edat;`

**ATENCIÓ:** amb els atributs `private`, això dona error. Després vorem com fer-ho.

- Per accedir als **mètodes** d'un objecte

`objecte.metode()`

Exemple: `client1.esMajorDeEdat();`

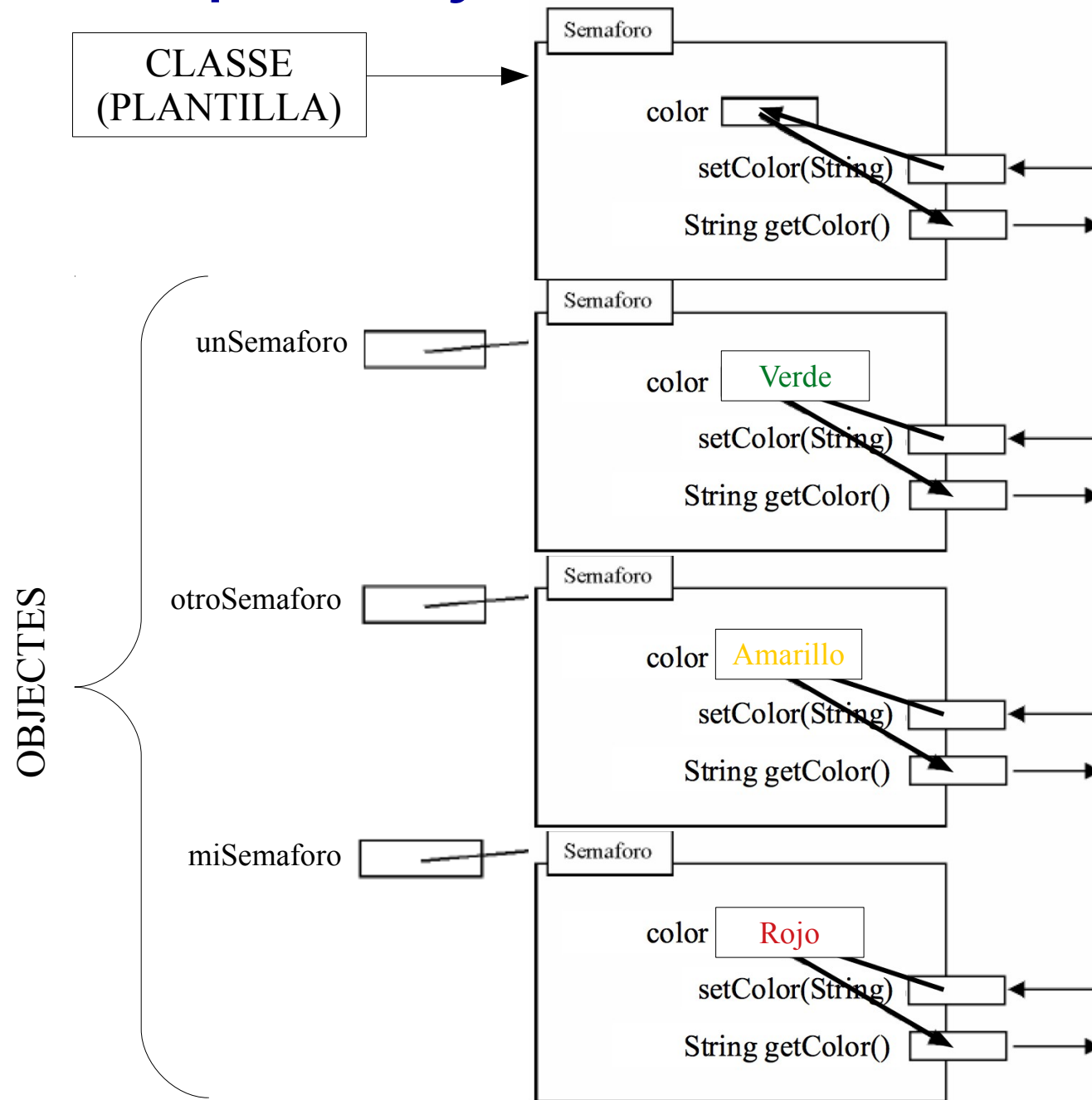
# Exemple: objets de la classe Semaforo

```
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");  
        }  
    }  
}
```



# Exemple: objets de la classe Semaforo

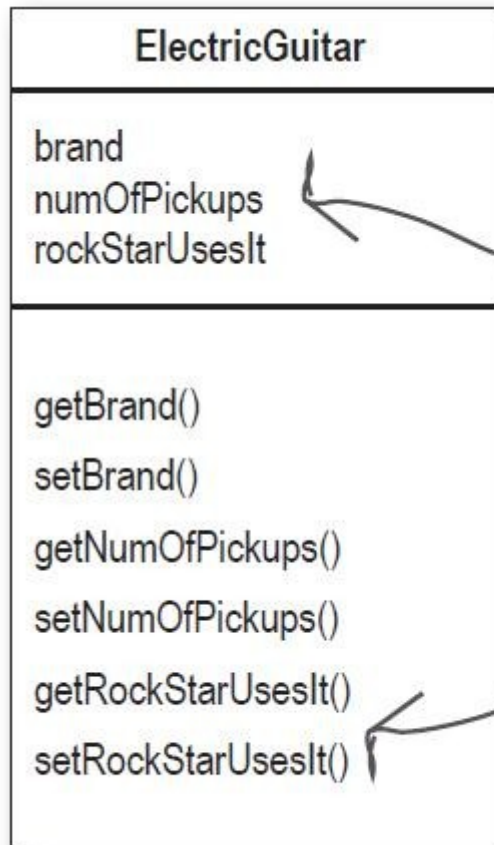


# Setters (modificadors) i Getters (consultors)

- És una bona pràctica (**I S'HA DE FER**):
  - Crear els atributs amb el modificador **private**.
  - Crear mètodes públics per accedir als atributs:
    - Per a consultar-los (**getters**)
    - Per a modificar-los (**setters**)
  - Des d'altres classes externes no es podran ni accedir ni modificar els atributs si no es fa mitjançant els *getters* i *setters*.
- Beneficis de l'**encapsulament**:
  - Que ningú accedisca per equivocació o sobreescriba funcionalitats quan no deu.
  - Un programador que utilitzi un mètode, sols necessita saber què fa, no com ho fa (caixa negra).



# Setters (modificadores) i Getters (consultors)



Nota: iusar 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;  
    }  
}
```

# Encapsulament

```
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();
```

```
    }
```

```
}
```

# Encapsulament

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 i Getters. Exemple

```
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();  
    }  
}
```



# Els constructors(I)

- Mètode especial d'una classe que és cridat automàticament sempre que es crea un objecte, és a dir, a l'emprar la instrucció new
- La seua funció és iniciar l'objecte.
  - Es recomana que els constructors inicialitzen totes les variables d'instància de l'objecte

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

# Els constructors(II)

- Per a declarar un constructor, és suficient amb declarar un mètode amb el mateix nom que la classe.
  - No es declara tipus de dades retornat pel constructor, ni tan sols *void*.
- Si no n'hi ha cap constructor en la classe, Java s'inventa un que no té arguments e inicialitza tots els atributs als valors per defecte.

enteros	0
decimales	0.0
booleanos	falso
referencias	null

- Java sols s'inventa els constructors si no n'hi ha cap.
- Si n'hi ha algun constructor, Java es limita a fer allò que el constructor diu.



# Els constructors(III)

- És possible declarar diferents constructors (sobrecàrrega de mètodes) a l'igual que la resta de mètodes de la classe.

```
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;  
    }  
    ...  
}
```

# Els constructors. Exemple

```
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);  
  
        ...  
    }  
}
```

- Per a crear “Rectangulos” podem utilitzar qualsevol dels 3 constructors que hem definit en la classe (plantilla).

# Destructors?

- En Java n'hi ha un recolector de basura (*garbage collector*) que s'encarrega de gestionar els objectes que es deixen d'utilitzar i alliberar l'espai que ocupen en memòria.
- Este procés és automàtic i impredecible i treballa en un fil (*thread*) de baix prioritat.
- En termes generals, este procés de recolecció de basura treballa quan detecta que algun objecte fa molt de temps que ja no s'utilitza en el programa.
- L'eliminació depén de la màquina virtual. Normalment, es realitza de forma periòdica.
- Podem invocar el mètode estàtic `System.gc()` per a “aconsellar” a la màquina virtual de Java que execute el recolector, però en ningú cas està assegurada la seua execució.

# La referència *this*

- La paraula reservada *this* és una referència al propi objecte amb el qual estem treballant.
- Exemple:

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

- En este exemple, cal la referència *this* per clarificar quan s'utilitzen les propietats posX i posY i quan els arguments amb el mateix nom.

# Atributs *static*

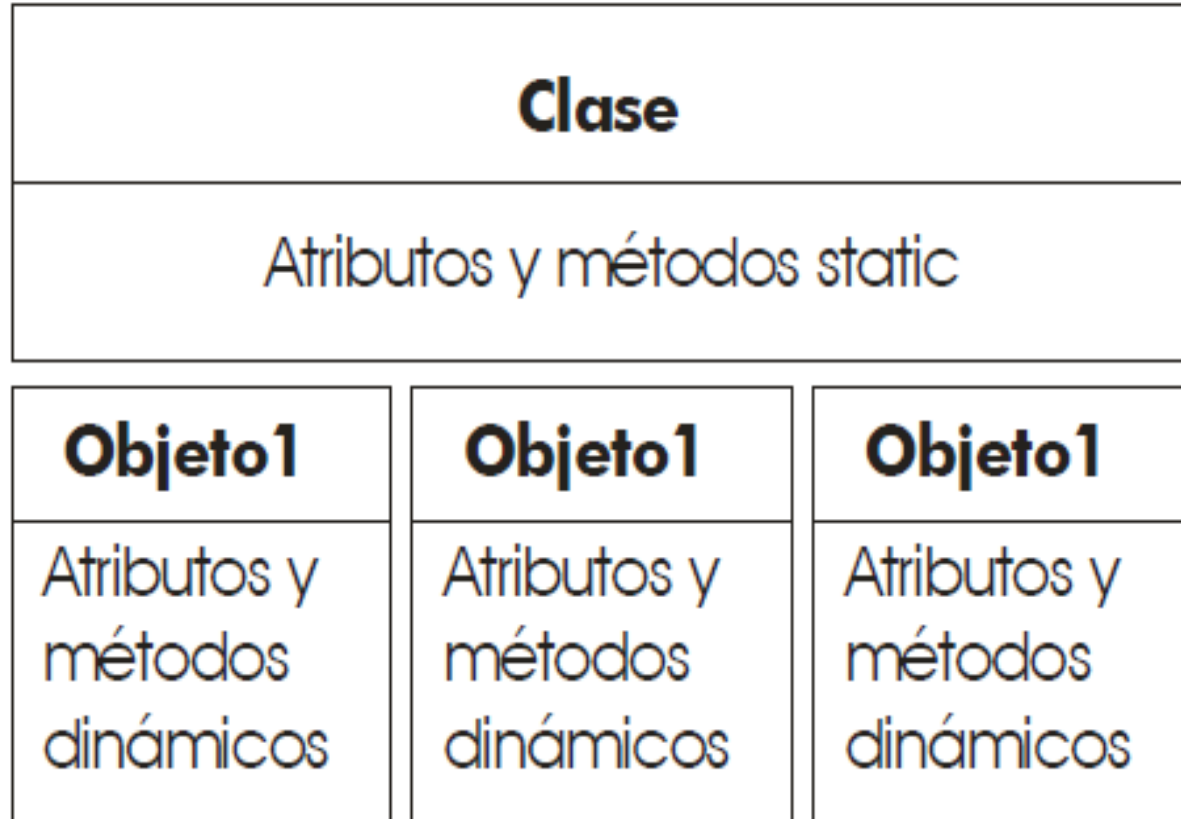
- Existeixen dos tipus d'atributs (variables d'instància):
  - **Atributs d'objecte**
    - Són variables o objectes que guarden valors diferents per a instàncies diferents de la classe (per a objectes diferents).
    - Si no s'especifica de forma explícita, els atributs són d'objecte.
  - **Atributs de classe**
    - Són variables o objectes que guarden el mateix valor per a tots els objectes instanciats a partir de la classe.
    - Es declaren amb la paraula reservada ***static***.

# Mètodes *static*

- Els mètodes static són mètodes de classe.
  - No és necessari crear un objecte de la classe (instanciar la classe) per poder cridar a un mètode static.
  - S'utilitzarà el nom de la classe “com si fora un objecte”.
- Els mètodes de classe (static) únicament poden accedir als seus atributs de classe (static) i mai als atributs d'objecte (dinàmics).
- Fins ara, els hem utilitzat:
  - Sempre que es declarava una classe executable
    - Per poder executar el mètode main() no es declara cap objecte d'eixa classe.
    - Quan hem creat els mètodes sense crear objectes en la UD6.



# Atributs i mètodes *static*



*Diagrama de funcionamiento de los métodos y atributos static*

# Mètodes *static*

- Es crearan mètodes i atributs genèrics quan este mètode o atribut val o dona el mateix resultat en tots els objectes.
- S'utilitzaran mètodes normals (dinàmics) quan el mètode dona resultats diferents segons l'objecte.
- Per exemple, una classe que represente avions:
  - l'alçada seria un atribut dinàmic (distint per a cada objecte)
  - el número total d'avions seria un atribut static (el mateix per a tots els avions)



# Agrupació de classes

- Els *packages* són una forma d'agrupar varies classes:
  - per a estructurar les classes en grups relacionats
  - per a aprofitar la possibilitat de donar als membres d'una classe la visibilitat a nivell de *package*
- Quan no s'especifica el nom del *package* al qual pertany una classe, passa a estar en el *package* per defecte.
- La declaració del *package* es fa a l'inici del fitxer .java:
  - `package x.y.x;`  
Significa que la classe definida en eixe fitxer serà del *package* x.y.z

# Packages i import

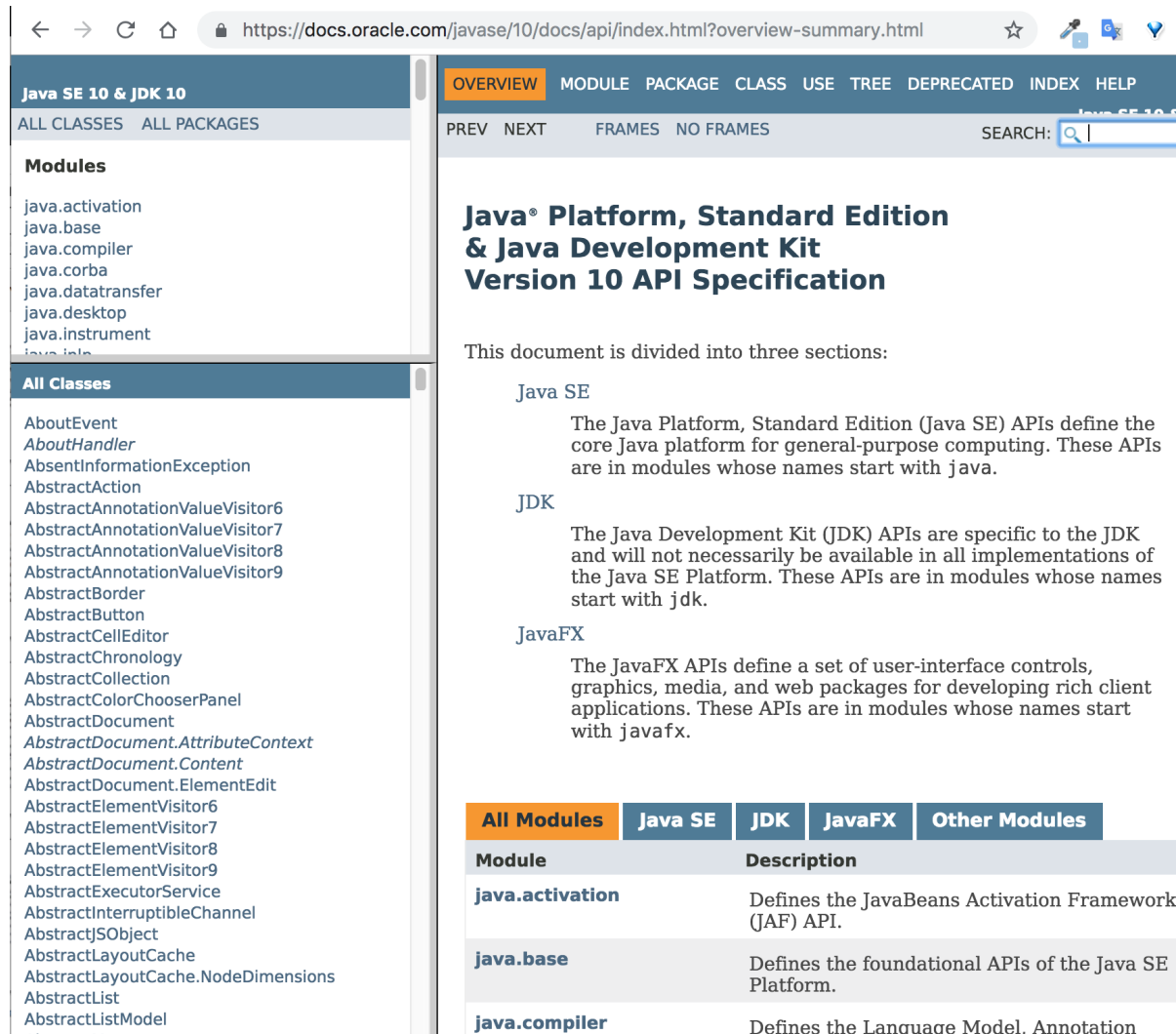
- El nom complet d'una classe és el nom del *package* en el qual es troba la classe, punt i després el nom de la classe.
    - Exemple: si la classe Cotxe està dins del *package* locomocio, el nom complet de Cotxe es locomocio.Cotxe
  - Mitjançant el comando import s'evita haver de col·locar el nom complet.
  - El comando import es col·loca abans de definir la classe:
    - Exemple: import locomocio.Cotxe;
  - Gràcies a esta instrucció es pot utilitzar la classe Cotxe sense necessitat d'indicar el *package* on es troba.
  - Es pot emprar el símbol \* com a comodí.
    - Exemple: import locomocio.\*;
- //Importa totes les classes del *package* locomocio

# Llibreries de classes(I)

- Hem vist que en Java n'hi ha una gran quantitat de classes ja definides i utilitzables. Venen agrupades en packages o llibreries estàndard:
  - `java.lang` - classes essencials, números, Strings, objectes, compilador... (es l'únic package que s'inclou automàticament en tots els programes Java)
  - `java.io` - classes que manegen entrades i eixides
  - `java.util` - classes útils, com estructures genèriques, maneig de data i hora, números aleatoris, entrada estàndard, ...
  - `java.net` – suport per a xarxes: URL, TCP, UDP, IP ...
  - `java.awt` – interfície gràfica, finestres, imatges ...
  - `java.applet` – creació d'applets

# Llibreries de classes(II)

- Recorda que a <https://docs.oracle.com/javase/10/docs/api/> tens documentació molt útil sobre l'API de Java. CONSULTA-LA!!!



The screenshot shows the Java SE 10 & JDK 10 API documentation page. The left sidebar contains a list of modules under the heading 'Modules'. The main content area is titled 'Java® Platform, Standard Edition & Java Development Kit Version 10 API Specification'. Below the title, it states 'This document is divided into three sections: Java SE, JDK, and JavaFX'. Each section has a brief description of the APIs. At the bottom, there is a table with columns for 'All Modules', 'Java SE', 'JDK', 'JavaFX', and 'Other Modules'. The table lists modules like 'java.activation', 'java.base', and 'java.compiler' with their descriptions.

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