

0101010
0100101
1101010

UD6.- PROGRAMACIÓ ORIENTADA A OBJECTES (POO)

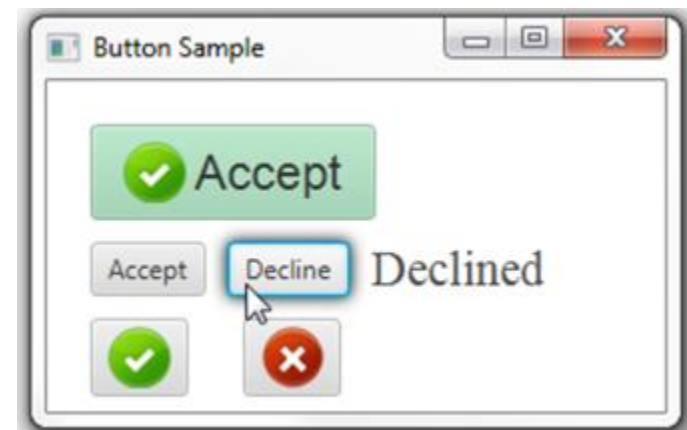
Programació – 1er DAW/DAM

0. CONTINGUTS

- CLASSES I OBJECTES
 - . QUÈ ÉS UNA CLASSE?
 - . DISSENY
 - . REPRESENTACIÓ
 - . QUÈ ÉS UN OBJECTE?
- PRINCIPIS BÀSICS DE LA POO
- DEFINICIÓ DE CLASSES
- UTILITZACIÓ DE CLASSE. OBJECTES
- MISSATGES I MÈTODES
- MÈTODES ESPECIALS
 - . CONSTRUCTORS
 - . CONSULTORS I MODIFICADORS
- EL MODIFICADOR `static`

1. INTRODUCCIÓ

- Fins ara som capaços de **solucionar problemes** fent ús de **seleccions, bucles, mètodes i arrays**.
- A causa de la **complexitat** dels desenvolupaments actuals, aquests **recursos** són **insuficients**.
 - Interfícies gràfiques.
 - Jocs.
 - **Aplicacions de gestió** que abasten tota la cadena de producció d'una empresa.



1.1 UN MÓN D'OBJECTES

- El món està **ple d'objectes**; un *estudiant*, una *taula*, un *cercle*, un *botó*, un *préstec*, una *maleta*...
 - Un objecte pot ser **real (cotxe)** o **abstracte (reserva en un hotel)**
 - Tot objecte té una **identitat** (que ho fa únic) i un **comportament** (definiu les accions que pot dur a terme).



1.2 QUÈ ÉS LA POO?

- **Metodologia de programació** en què la **resolució** d'un **problema** es basa en la **simulació d'un escenari real** mitjançant l'ús d'**objectes** i la **interacció** entre ells.



1.4 PERQUÈ POO?

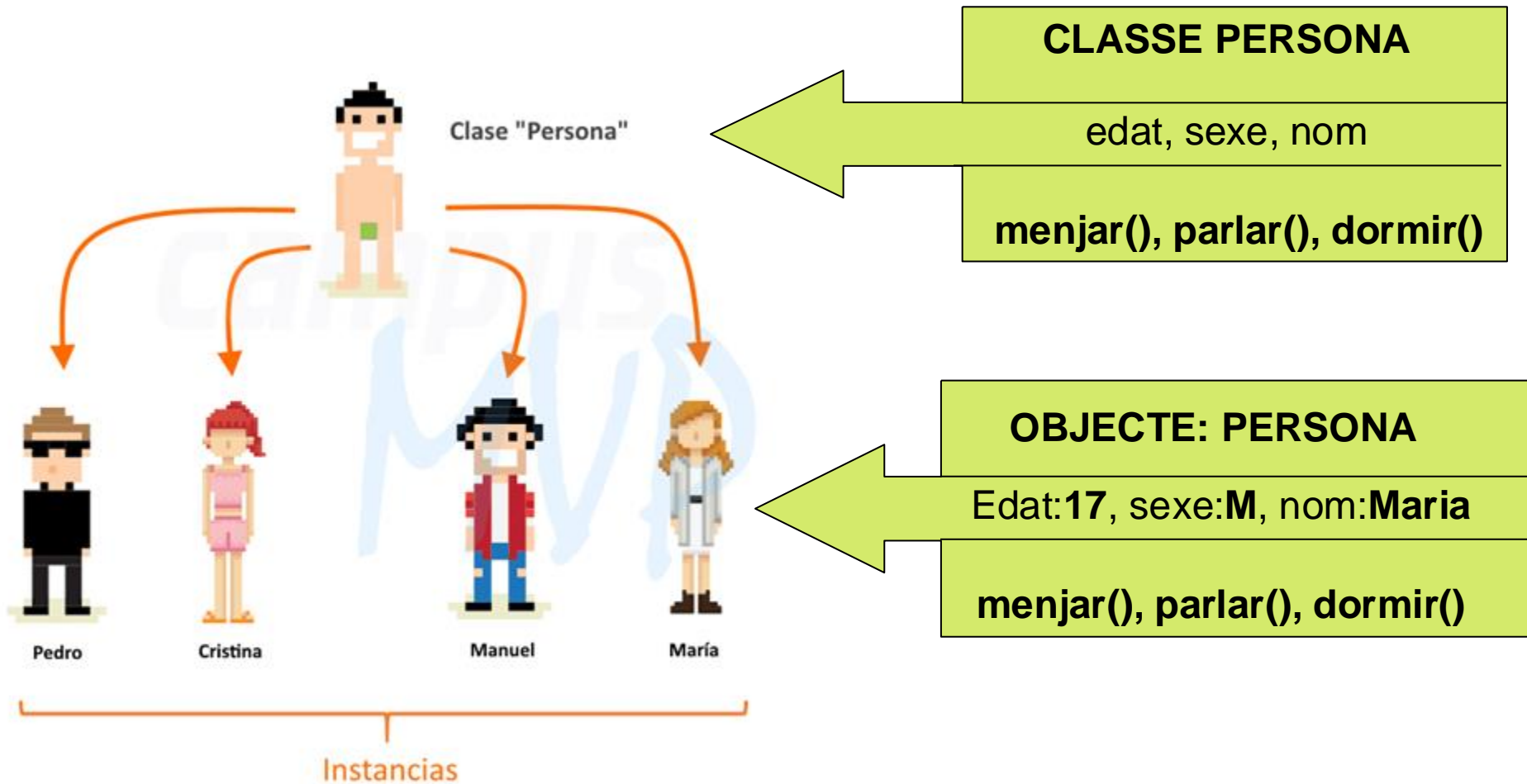
- La tècnica de **disseny orientat a objectes** segueix sovint el mateix mètode que apliquem en la resolució de problemes de la **vida diària**.
- **Pensar en objectes és més natural**; El dissenyador pensa en termes d'objectes i no en detalls de baix nivell.
- **Senzillesa**; els programes es creen a partir de peces petites (objectes).
- **Reutilització**; afavoreix la **reusabilitat** del codi i el **treball en equip**.

2. CLASSES I OBJECTES

Què és una classe?

- **Plantilla** o estructura preliminar que descriu les **característiques d'un objecte** (variables d'instància) i el seu **comportament** (mètodes).
- **Variables d'instància:** dades que contindran tots els objectes de la classe
 - Es declaren amb un nom i un tipus de dades.
- **Mètodes:** Són les operacions que podran realitzar els objectes de la classe.

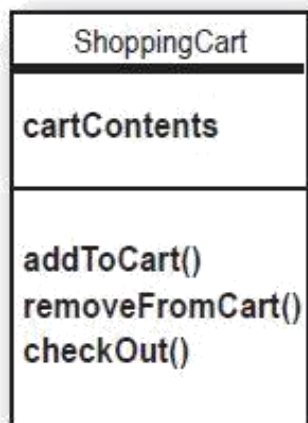
2. CLASSES I OBJECTES



Els termes **objecte** e **instància** són **intercanviables**

2.1 DISSENY DE CLASSES

- Quan es dissenya una **classe**, es pensa en els **objectes** de la classe que es crearan:
 - El que **sap e identifica** l'objecte; (components, característiques)
 - El que l'objecte **fa** (comportament)



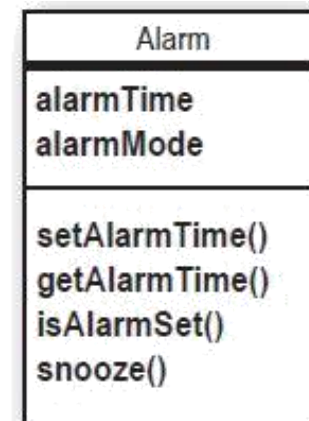
sabe

hace



sabe

hace



sabe

hace

2.1 DISSENY DE CLASSES

- El que l'objecte **sap** → **variables d'instància** (atributs o propietats)
- El que l'objecte **fa** → **mètodes**



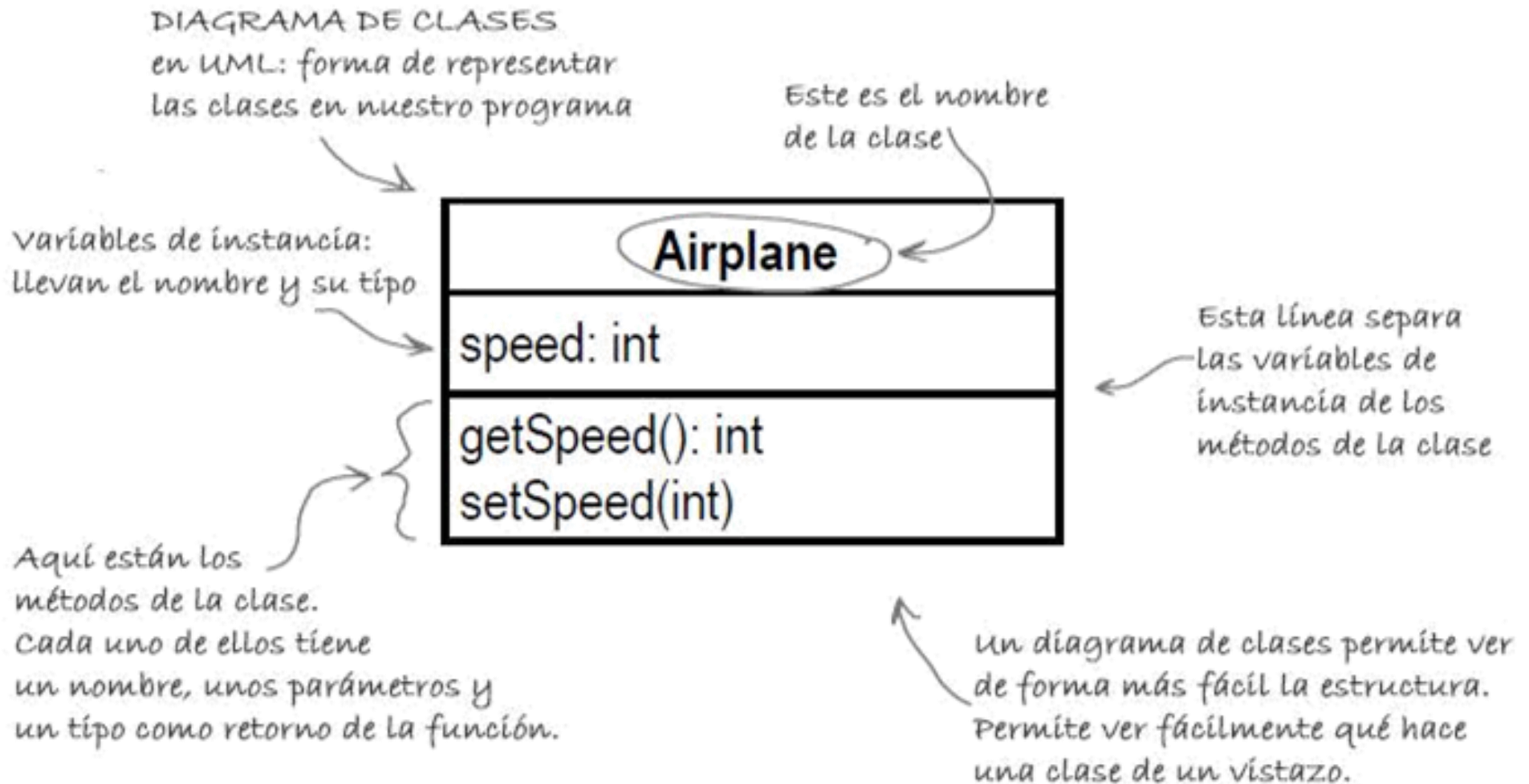
2.1.1 REPRESENTACIÓ

- **UML** (*Unified Modelling Language*): Llenguatge de modelatge de dades.
- **Diagrama de classes**: És un **diagrama UML** que utilitzarem per definir les classes (atributs i mètodes) i les relacions entre elles.

El treballarem al **mòdul d'ENTORNS DE
DESENVOLUPAMENT.**

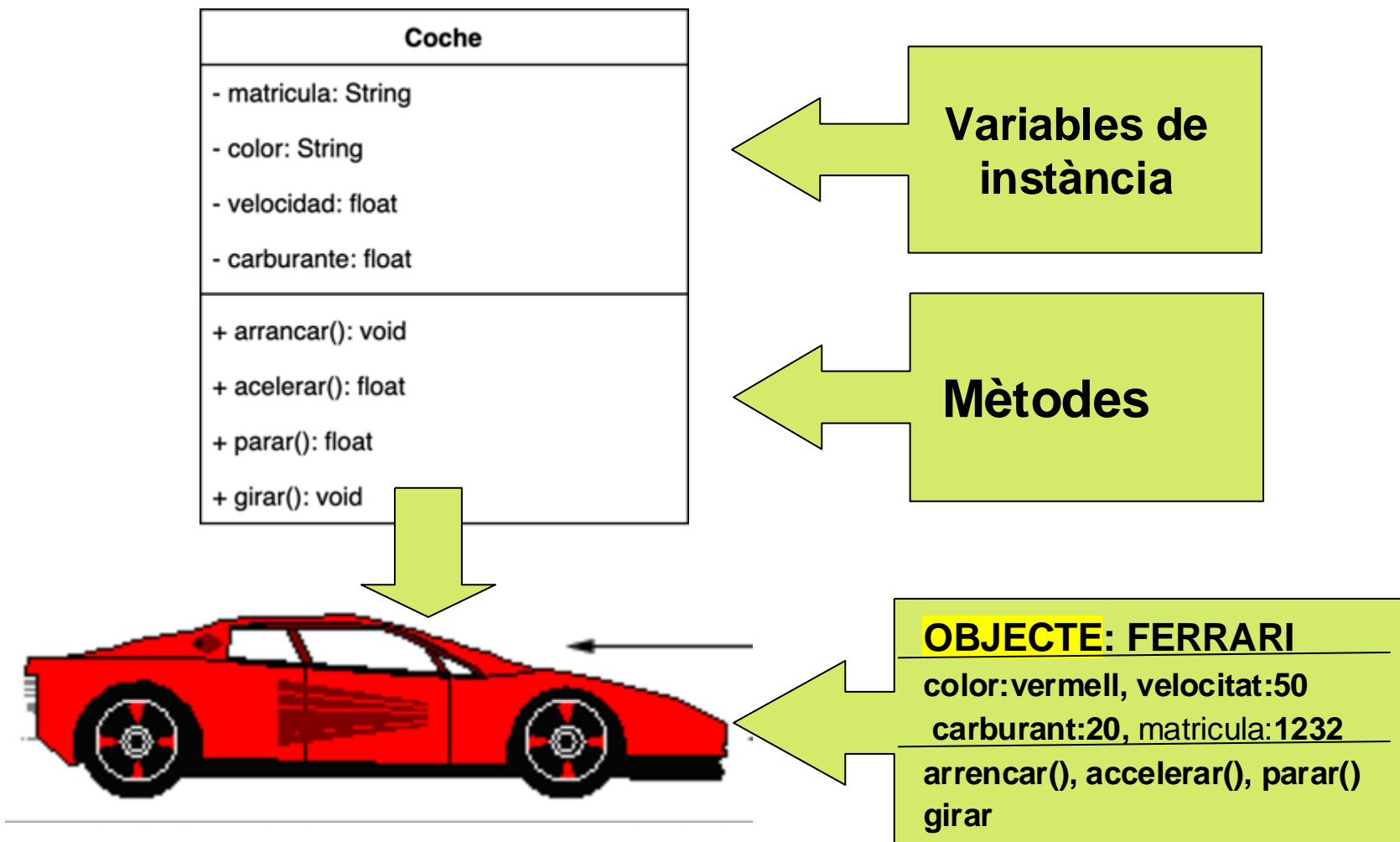
De moment, només comentarem les **característiques
bàsiques**

2.1.1 REPRESENTACIÓ



2.1.1 REPRESENTACIÓ

- Classe: **Cotxe**



2.2 ACTIVITAT PRÈVIA

- **Activitat 6.1-** Pensa en allò que podria tenir la classe **Televisio**, el que **hauria de saber** i allò que **hauria de fer**.



2.3 QUÈ ÉS UN OBJECTE?

- Una **instància** (exemplar) d'una **classe**.
- Característiques:
 - **Identitat**: TOT objecte és únic i diferent dels altres.
 - **Estat**: informació **que conté** en un moment donat. (variable en el temps)
 - **Comportament**: Els **objectes realitzen tasques** que es veuen reflectides en **canvis en el vostre estat** o interaccionant amb altres mètodes

CLASSE

Televisio
marca: String encendido: boolean; canal: int
encendre(); canviarCanal(canal: int);

OBJECTE

Samsung
1
true

marca

canal

engegat

ESTAT

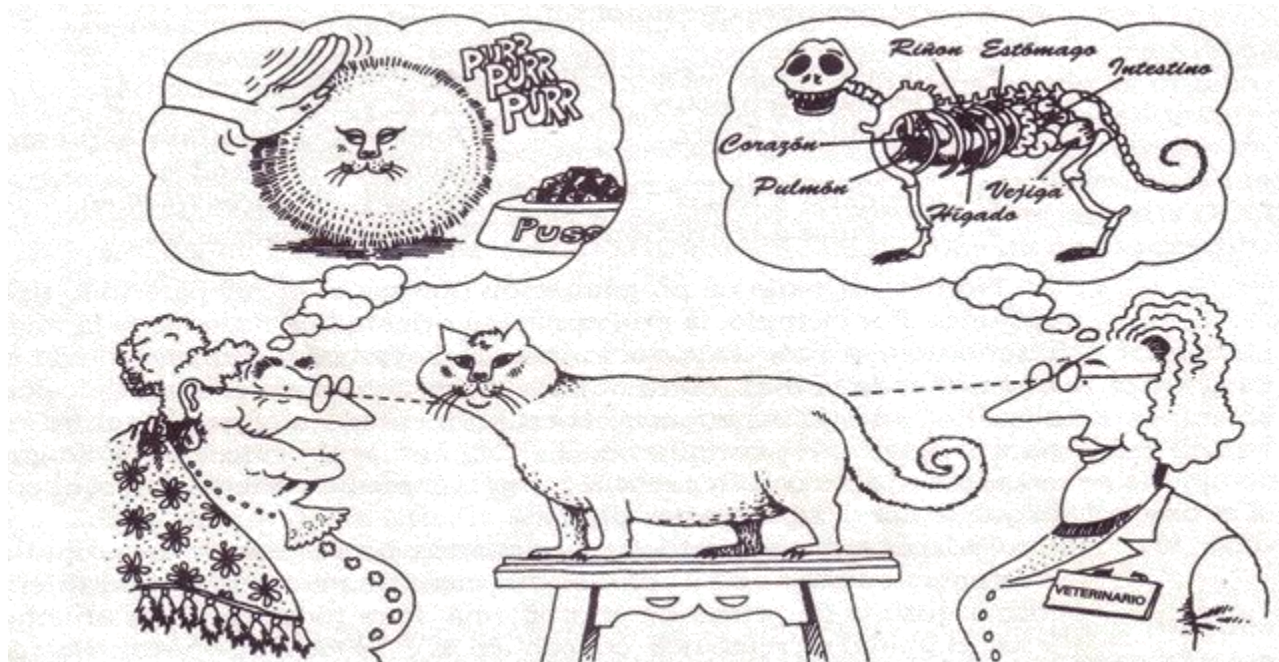
Televisor
Samsung
Encès a
canal 1

3. PRINCIPIS DE LA POO

- Tot llenguatge orientat a objectes es fonamenta en 4 pilars:
 - Abstracció
 - Encapsulació / Ocultació
 - Herència (Ho veurem més endavant)
 - Polimorfisme (Ho veurem més endavant)

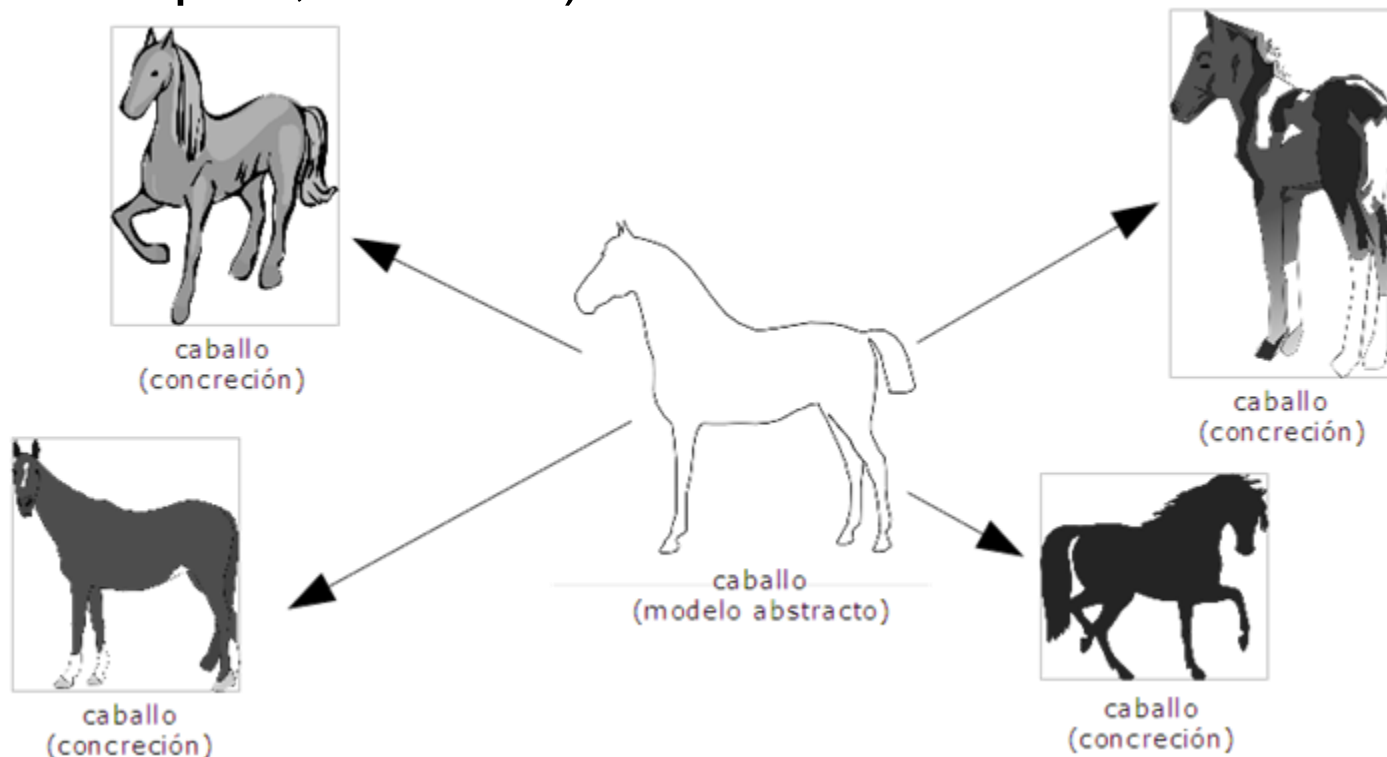
3.1 ABSTRACCIÓ

- **Mecanisme** que ens permet **determinar**, a partir de l'observació de la realitat, les **classes** i les seus **característiques**; *dades i comportament*
- Seran **diferents segons l'observador**. *Domini de l'aplicació*



3.1 ABSTRACCIÓ. EXEMPLE

- **Exemple:** tenim cavalls de diferent raça. El seu aspecte exterior és molt diferent, però sabem que tots pertanyen a la classe Cavall perquè en realitzem una **abstracció** o identificació dels **elements comuns** que tenen (cua, quatre potes, són ràpids, corren...).



3.2 ENCAPSULACIÓ / OCULTACIÓ

- **Encapsulació**

- Una classe està formada per **propietats** (variables d'instància) i mètodes.
- **No** es poden definir **variables ni mètodes fora d'una classe**, és a dir, no hi ha variables ni mètodes globals.

- **Ocultació**

- Hi ha una **part privada** en definir les classes que únicament és utilitzada per aquesta classe de manera directa.
- Hi ha una **part pública** que es pot utilitzar a qualsevol part del codi.

3.2 ENCAPSULACIÓ / OCULTACIÓ

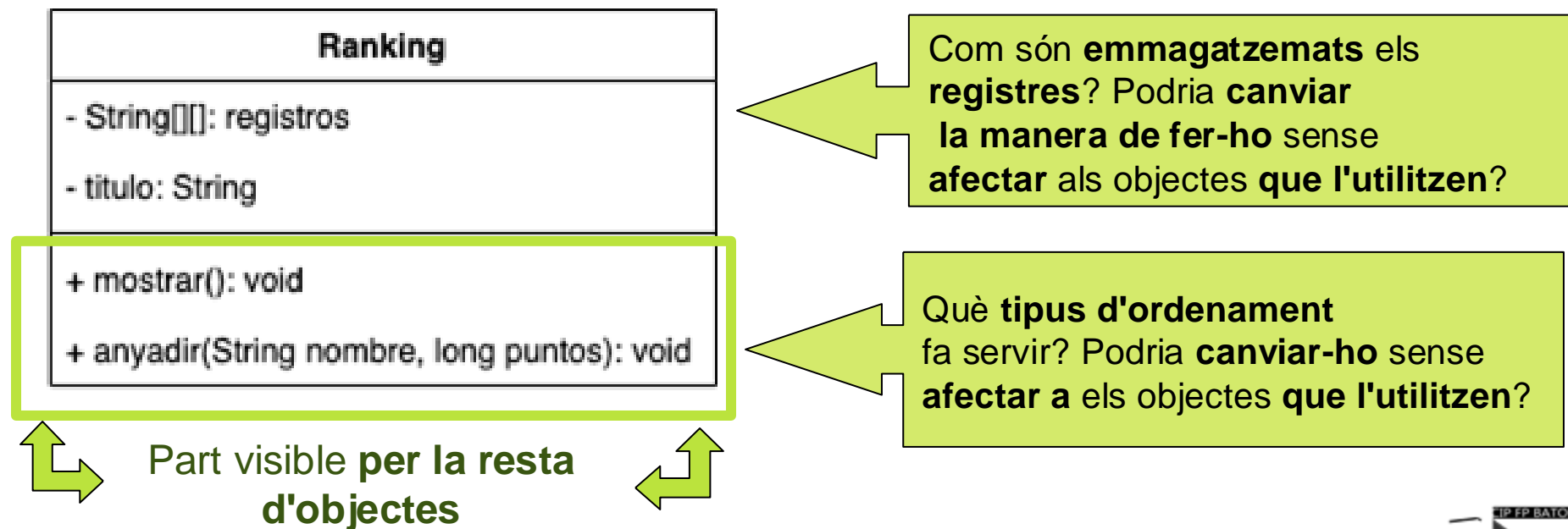
- Els objectes seran “caixes negres”: sabem **que fan** a través de la interfície **però no** sabem **com** ho fa.
- La **interfície** quedarà definida pel el seu **comportament** accessible a través dels **mètodes públics**

**No és necessari
conèixer la
implementació interna
de la classe per a poder
utilitzar-la**



3.2 ENCAPSULACIÓ / OCULTACIÓ

- La **encapsulació** permet que un **canvi** en una **classe** siga **transparent** per a la resta de l'aplicació. Sempre que es **mantinga** la seva **interfície**.
- **No propagació d'errors** davant canvis.



3.2 ENCAPSULACIÓ / OCULTACIÓ

- **Exemple:** un terminal d'autoservei és senzill d'utilitzar per a l'usuari. s'amaguen els **detalls d'implementació** (que és complexa) i **s'exposen** només les **funcions d'alt nivell** (*Treure diners, consultar saldo,...*)



3.3 ACTIVITAT PRÈVIA

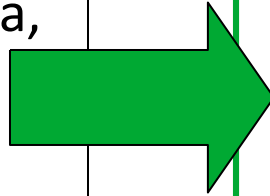
Activitat 6.2- En parelles, pensa els atributs i mètodes que podria tenir una classe **Vivenda** des del punt de vista de:

- Una **aplicació de venda i lloguer** immobles.
- Una **aplicació de venda de productes online** on els habitatges són els punts on enviar els productes.

4. DEFINICIÓ DE CLASSES

Sintaxi

```
[modificador_accés] class NomClasse  
{  
    // atributs (variables de instància,  
    propietats...) de la classe  
    // mètodes (comportament,  
    accions...) de la classe  
}
```



```
public class Televisio {  
    // Atributs  
    // Mètodes  
    ...  
}
```

- Recorda que **una classe** ha de ser creada a **un fitxer** amb el nom **NomClasse.java** (**CamelCase**)

4. DEFINICIÓ DE CLASSES

Modificadors d'accés

Paraula clau	definició
<i>public</i>	La classe és accessible des d'altres <i>packages</i> .
<i>package</i> (Per defecte)	La classe serà visible a totes les classes declarades al mateix <i>package</i> .
<i>abstract</i>	<i>Les classes no poden ser instanciades. Serveixen per definir subclasses. Ja ho veurem...</i>
<i>final</i>	<i>Cap classe no pot heretar d'una classe final. Ja ho veurem...</i>

4.1 DECLARACIÓ D'ATRIBUTS

- Sintaxi

```
[Modificador D'Accés] [static] [final] tipus nom_atribut
```

```
public class Televisio {  
    String marca;  
    int canal;  
    float volum;  
    ...  
    // mètodes  
    ...  
}
```

4.1 DECLARACIÓ D'ATRIBUTS

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

paraula clau	definició
<i>private</i>	L'element només és accessible dins del fitxer on està definit.
<i>package</i> (Per defecte)	L'element només és accessible dins del <i>package</i> on està definit.
<i>protected</i>	L'element és accessible dins del <i>package</i> on està definit i, a més, a les subclasses (<i>concepte que veurem en la següent unitat</i>).
<i>public</i>	L'element és accessible des de qualsevol lloc.

4.1 DECLARACIÓ D'ATRIBUTS

- Sintaxi

[ModificadorD'Accés] [static] [final] tipus nom_atribut

```
public class Televisio {  
    private String marca;  
    private int canal;  
    private float volum;  
    ...  
    // mètodes  
    ...  
}
```

Tots els **atributs** haurien
ser **privats**.

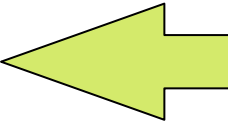
Només **accessibles**
des dels mètodes
de **pròpia classe**
(**Ocultació**)

4.2 DECLARACIÓ DE MÈTODES

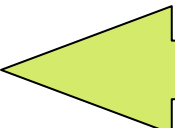
Sintaxi

[ModificadorAccés] [static] [tipus_retorn|void] **nomMètode** ([Paràmetres])

```
public class Televisio {  
  
    private String marca;  
  
    private int canal;  
  
    private float volum;  
  
    ...  
  
    public void establirCanal(int nombre){}  
  
    public void establirVolum(int velocitat){}  
  
    public void encendre(){}  
  
    private int iluminarPixel(int pixel, float rgb){}  
  
}
```



Els mètodes públics podran ser **cridats** per **altres objectes**



Els mètodes privats només podran ser **utilitzats** des dels mètodes del **propi objecte (Ocultació)**



5. UTILITZACIÓ DE CLASSES

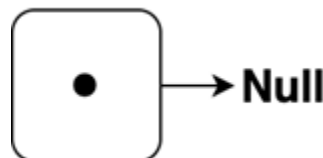
Declaració d'un objecte

- Quan **declarem** una classe, estem definint un **nou tipus de dades** en el sistema, que **utilitzarem** per **crear** (instanciar) **objectes**.
 - El primer pas serà **declarar la variable** amb què farem referència al nou objecte.

NomClasse variable;

Televisio televisio;

- Amb això tenim un apuntador capaç d'apuntar l'objecte, però **no tenim l'objecte**. (la variable conté la referència **null**)



5. UTILITZACIÓ DE CLASSES

Instanciació d'un objecte

- Per **crear** un **objecte**, utilitzem la **paraula reservada** **new** seguida d'un mètode que es diu com la classe. **(El constructor)**.

```
Televisio lg = new Televisio();
```

```
Televisio samsung = new Televisio();
```

- També podem **crear la variable** **i**, en un pas posterior, l'objecte.

```
Televisio lg;
```

```
lg = new Televisio();
```

6. MISSATGES I MÈTODES

- A **POO** el problema es resol mitjançant la **interacció** entre els diferents **objectes del sistema**.
 - Aquestes accions es duen a terme a través del **pas de missatges**.

missatge = invocació a un mètode

```
Televisio lg = new Televisio();  
lg.establirCanal(4)
```

Utilitzem el **operador “.”** precedit de l'**objecte** al que volem **enviar el missatge**

6.1 TREBALL AMB CLASSES I OBJECTES

- Què necessitem (**de moment**)?
 - Una **classe** que modelarà el que volem representar.
 - Una altra classe per provar-la, que contindrà el mètode **main()**.

```
public class TestDog {
```

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

```
        Dog toby = new Dog();
```

```
        Dog boby = new Dog();
```

```
        toby.bark();
```

```
        boby.bark();
```

```
    }
```

```
}
```

```
public class Dog {
```

```
    private int size;
```

```
    private String breed;
```

```
    private String name;
```

```
    void bark() {
```

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

```
    }
```

```
}
```



7. MÈTODES ESPECIALS

7.1 Constructors

- Mètode que és anomenat automàticament sempre que es crea un objecte, És a dir, en emprar la instrucció **new**.
- La seva funció és inicialitzar el estat de l'objecte, és a dir assignar un valor inicial a cadascun dels seus atributs

```
public class Rectangle {  
  
    private int width;  
  
    private int height;  
  
    ...  
    public Rectangle(int width, int height) {  
        this.width = width;  
        this.height = height;  
    }  
    ...  
}
```

```
Rectangle r1 = new Rectangle(20,40)  
Rectangle r2 = new Rectangle(40,500)  
Rectangle r3 = new Rectangle(70,20)
```

7. MÈTODES ESPECIALS

7.1 Constructors

- Per **declarar** un **constructor**, n'hi ha prou amb declarar un **mètode** amb el **mateix nom** que la **classe**.
 - **No** es declara el **tipus** de dades **tornat** pel constructor.
- Si **no es defineix cap constructor**, Java inventa un que no té arguments e inicialitza tots els **atributs a valors per defecte**.
 - Això passa només si no hi ha **cap constructor definit**.
 - Si hi ha algun constructor, **Java** es limita a fer allò que el constructor diu.

7. MÉTODOS ESPECIALES

7.1 Constructores

És possible **declarar diferents constructors** (sobrecàrrega mètodes)

```
public class Rectangle {  
    private int width;  
    private int height;  
    public Rectangle() {  
        this.width = 80;  
        this.height = 40;  
    }  
    public Rectangle(int width, int height) {  
        this.width = width;  
        this.height = height;  
    }  
    public Rectangle (int height) {  
        this.width = 200;  
        this.height = height;  
    }  
}
```



7.2 LA PARAULA RESERVADA **this**

La paraula reservada **this** referència al propi objecte amb què estem treballant.

```
public class Rectangle {  
  
    private int width;  
  
    private int height;  
  
    ...  
    public Rectangle (int width, int height) {  
        this.width = width;  
        this.height = height;  
    }  
    ...  
}
```

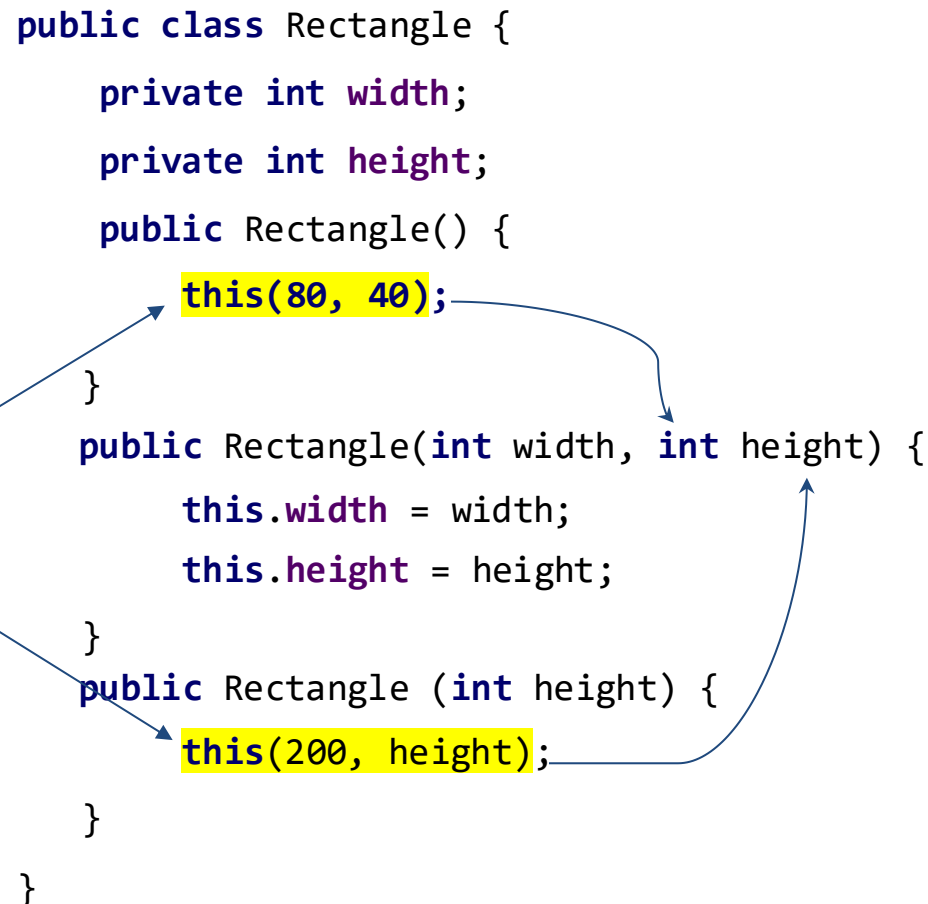
A l'exemple, la referència **this** ens permet clarificar quan s'utilitzen les propietats **width/height** i quan els arguments amb el mateix nom.

7.2 LA PARAULA RESERVADA *this*

A més, també la **paraula reservada** *this* ens permetrà, dins d'un mètode constructor, invocar a un altre, ja definit, a la mateixa classe.

En ambdòs casos, s'està invocant al constructor que rep dos paràmetres, també definit a la pròpia classe.

```
public class Rectangle {  
    private int width;  
    private int height;  
    public Rectangle() {  
        this(80, 40);  
    }  
    public Rectangle(int width, int height) {  
        this.width = width;  
        this.height = height;  
    }  
    public Rectangle (int height) {  
        this(200, height);  
    }  
}
```



7.3 ACTIVITATS PRÈVIES

Activitat 6.3.- Implementa la classe **Rectangle** definida anteriorment. Seguidament du a terme les accions següents.

- Afegeix un **mètode** que calcule i torneu l'àrea del **Rectangle**.
- Crea una classe **TestRectangle** que contindrà el mètode **main()**. Seguidament crea **6 rectangles** amb les dimensions següents i mostra l'àrea de cada una de elles.

Nom objecte	Amplada	Alçada
r1	100	200
r2	40	100
r3	80	40
r4	56	90
r5	100	150
r6	200	120

7.3 ACTIVITATS PRÈVIES

Activitat 6.4- Indica la eixida per pantalla del següent programa:

```
public class TestFitxa {

    public static void main (String [] args) {

        Fitxa fitxaGroga = new Fitxa("Groc");
        Fitxa fitxaBlava = new Fitxa("Blau");
        System.out.printf ("El jugador %s és a la casella %d\n",
fitxaGroga.getColor(), fitxaGroga.getCasillaActual());

        fitxaBlava.avançar(10);
        System.out.printf ("El jugador %s és a la casella %d\n",
fitxaBlava.getColor(), fitxaBlava.getCasillaActual());
    }
}
```

```
public class Fitxa {

    private int casella;
    private String color;

    Fitxa(String color) {
        this.casella = 1;
        this.color = color;
    }

    public void avançar(int num) {
        this.casella + = num;
    }

    public int getCasillaActual() {
        return casella;
    }

    public String getColor() {
        return color;
    }
}
```


7.5 ACTIVITATS PRÈVIES

Activitat 6.5.- Crea una classe **Persona**. De cada persona volem emmagatzemar el nom, cognoms, edat i si està casat o no.

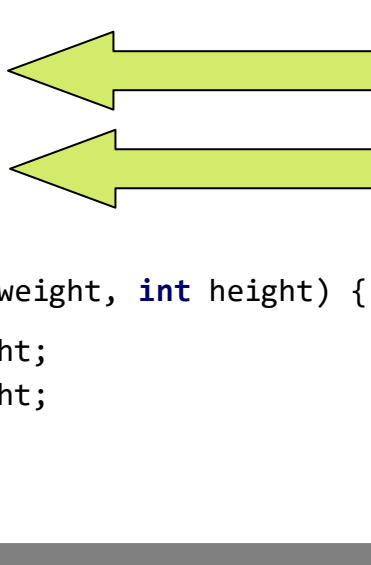
- Crea un **mètode constructor** que ens permeti **inicialitzar totes les propietats** de classe.
- Crea un **segon mètode constructor** que **només rebi el nom, cognoms, i l'edat**. Heu d'establir per defecte que aquesta persona no està casada.
- Escriu un **mètode saluda()**, que en cridar-lo faci que la persona es presenti, mostrant per pantalla el següent missatge *"Hola, sóc XXX, la meua edat és XX anys i xx estic casat"*

Crea una classe **TestPersona** que cree 2 objectes amb les dades que vulgues (1 casat i 1 solter) i mostra la informació de cadascun d'ells

7.4 MODIFICADORS I CONSULTORS

- Com hem comentat anteriorment, **els atributs d'una classe** han de declarar-se amb el modificador **private** (Principi d'ocultació)
 - Aleshores, com **accedim** a la **amplada** (width) i **altura** (height) del rectangle des d'una altra classe o objecte?
 - I si necessitem **modificar l'amplada** del rectangle?

```
public class Rectangle {  
  
    private int weight;  
  
    private int height;  
  
    ...  
    public Rectangle (int weight, int height) {  
        this.weight = weight;  
        this.height = height;  
    }  
    ...  
}
```



7.4 MODIFICADORS I CONSULTORS

- La declaració de mètodes **getters** (consultors) i **setters** (modificadors) són un **estàndard en java**
 - **Getters:** Permeten **consultar** el **valor** d'un **atribut** d'instància i **tenen la forma:**

```
public tipus getNomAtribut() {  
    return nomAtribut;  
}
```

- **Setters:** Permeten **modificar** el **valor** d'un atribut de la instància i **tenen la forma:**

```
public void setNomAtribut(tipus nomAtribut){  
    this.nomAtribut = nomAtribut;  
}
```

7.4 MODIFICADORS I CONSULTORS

Exemple:

```
public class Rectangle {  
  
    private int width;  
  
    private int height;  
  
    public Rectangle (int width, int height) {  
        this.width = width;  
        this.height = height;  
    }  
    public int getWidth() {  
        return width;  
    }  
    public void setWidth(int width) {  
        this.width = width;  
    }  
    public int getHeight() {  
        return height;  
    }  
    public void setHeight(int height) {  
        this.height = height;  
    }  
}
```



Només hem de crear els
getters/setter
estrictament necessaris

7.4 MODIFICADORS I CONSULTORS

Beneficis de la **ocultació/encapsulació**:

- Que ningú **accedisca per equivocació** o sobre-escriga valors quan no ha de fer-ho.
- **Un programador** que **utilitze** un **mètode**, **només** necessita **saber què fa**, no com ho fa (**caixa negra**).
- Si hi ha un **error** a la classe, aquest es **limitarà a l'àmbit de la classe**.
- Es poden fer canvis i millores sense afectar la resta de classes sempre que es mantinga la **interfície pública (mètodes públics)**

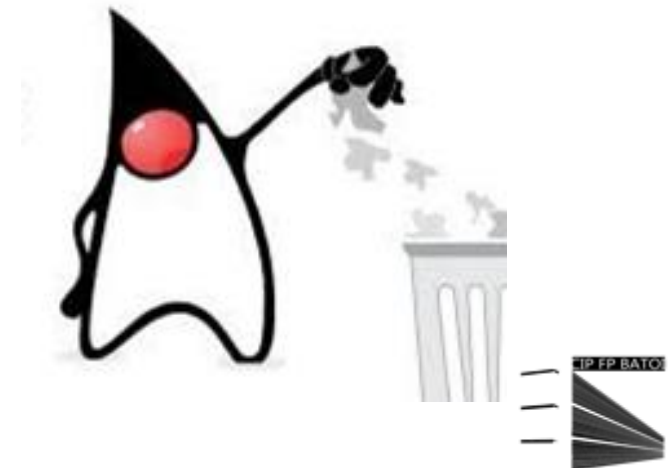
7.5 ACTIVITATS PRÈVIES

Activitat 6.6.- Afegeix mètodes **consultors** per a la propietat **width** i **height** de la classe **Rectangle**. Seguidament crea un **modificador** només per a la propietat **width** de tots els rectangles i realitza les **següents accions**:

- Fent ús del mètode modificador, modifica la propietat **width** de tots els rectangles, mitjançant un valor aleatori entre 10 i 200.
- Crea un **mètode** **mostrarInfo** a la classe **TestRectangle** que **accepte com a paràmetre** un objecte de tipus **Rectangle** i, fent ús dels seus **mètodes** consultors mostre l'amplada, alçada i àrea del rectangle rebut.
- Modifica la **classe** **Rectangle** i crea un mètode **mostrarInfo** que en anomenar-lo mostre la seua amplada, la seua alçada i la seua àrea. Per finalitzar, invoca'l des de la classe **TestRectangle** per a cadascun dels objectes de tipus **Rectangle** de què disposem.

8. DESTRUCTORS

- A Java hi ha un recol·lector de brossa (***garbage collector***) que s'encarrega de gestionar els objectes que es deixen d'utilitzar i alliberar l'espai que ocupen en memòria.
- Aquest procés és **automàtic i imprevisible** i treballa en un fil (*thread*) de **baixa prioritat**.
- En termes generals, aquest procés de recol·lecció de brossa treballa quan detecta que algun **objecte fa molt de temps que ja no s'utilitza** al programa i l'elimina.



9. EL MODIFICADOR **static**

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

9.1 ACTIVITATS PRÈVIES

Activitat 6.7.- Afegeix una **variable de classe** (*static*) a la classe **Rectangle** que actue com **comptador del nombre d'objectes** que hi ha d'una classe en un moment donat.

*Per això, **hauràs de fer** que el constructor **actualitze aquesta variable** en 1 cada vegada que es cree un objecte d'aquest tipus. Per finalitzar prova el seu funcionament creant un array de **10 rectangles**.*

- Això és tot... de moment :-)