GRAU D'ENGINYERIA INFORMÀTICA

PROGRAMACIÓ II

Bloc 2:

Programació Orientada a Objectes (1)

Laura Igual
Departament de Matemàtiques i Informàtica
Facultat de Matemàtiques i Informàtica
Universitat de Barcelona

UNIVERSITAT DE BARCELONA

Índex Bloc 2:
Programació orientada a objectes

- Abstracció en el desenvolupament del *software*
- Conceptes fonamentals: classes i objectes
- Característiques de l'orientació a objectes
- Ús de classes i objectes
- Constructors i destructors
- Encapsulació
- Herència i jerarquia de classes
- Polimorfisme
- Lligadures
- Interfícies
- col·leccions

ABSTRACCIÓ EN EL DESENVOLUPAMENT DEL *SOFTWARE*

Abstracció en el desenvolupament de software

- Abstracció:

  Extracció de les característiques essencials d'un objecte i dels seus comportaments.
- Una vegada s'han identificat els objectes, identificar les seves relacions en el món real.

Abstracció en el desenvolupament de software

- **Orientació a Objectes** (OO) consisteix en organitzar el software com una col·lecció discreta d'entitats que incorporen:
  - les dades
  - el comportament d'aquestes dades.
- **Classes:** entitats en les que la OO estructura el software en dades i el conjunt d'operacions associades a aquestes dades.
- Un **programa** és un conjunt **d'objectes** (que pertanyen a diferents classes) que **interactuen entre si** per tal de resoldre el problema.

Programació Orientada a Objectes

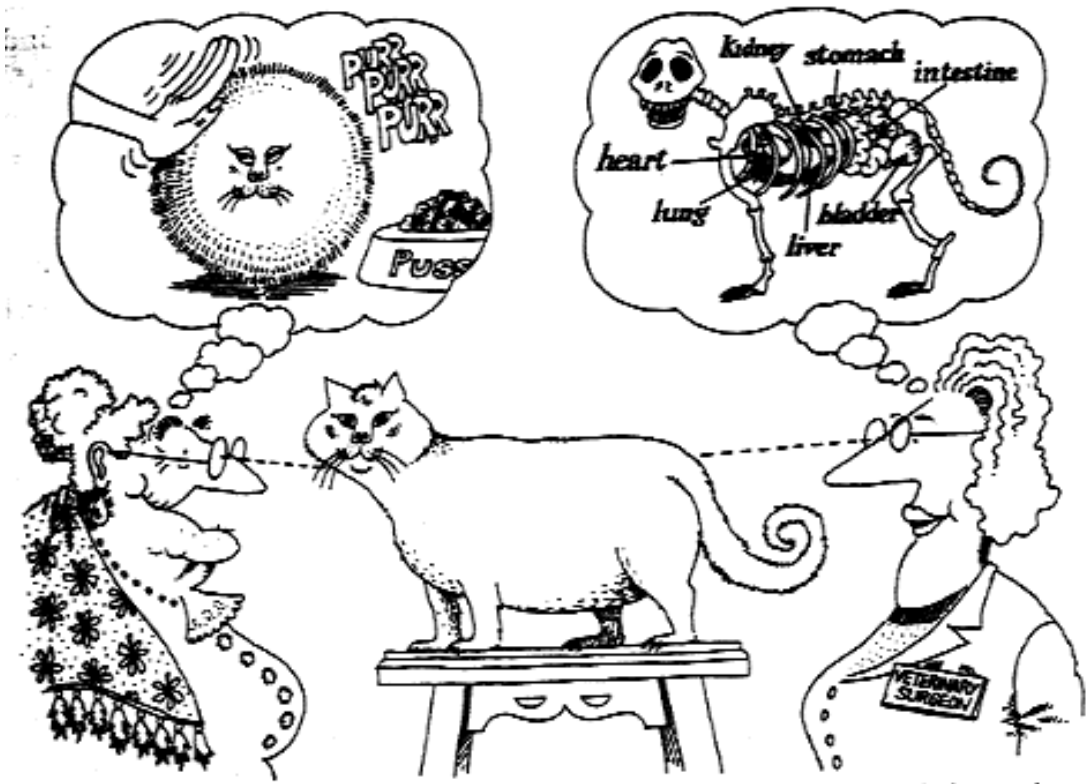
- La OO és una filosofia, no està lligada a cap llenguatge de programació.
- Java és un llenguatge de programació orientat a objectes (POO).

CARACTERÍSTIQUES DE L'ORIENTACIÓ A OBJECTES

Característiques de la OO

Abstracció

- Consisteix en agafar una informació i extreure'n les característiques més representatives



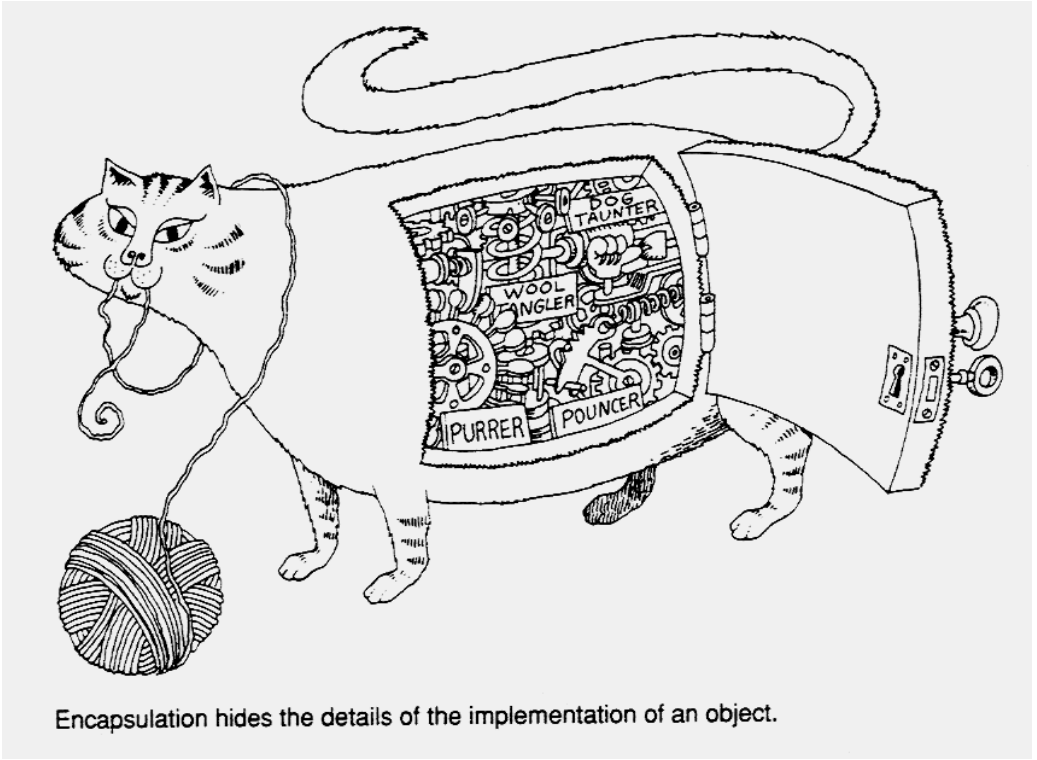
Abstraction focuses upon the essential characteristics of some object, relative to the perspective of the viewer.

Figura extreta de la pàgina 39 del llibre: "**Object-Oriented Analysis and Design with Applications**". Grady Booch. The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.

Característiques de la OO

Encapsulament

- Amaga a l'usuari la implementació interna de l'objecte



Encapsulation hides the details of the implementation of an object.

Figura extreta de la pàgina 46 del llibre: "**Object-Oriented Analysis and Design with Applications**". Grady Booch. The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.

Característiques de la OO

Herència

- Defineix una relació entre classes.
- Una classe (**superclasse**) defineix un conjunt de propietats comuns a altres classes (**subclasses**).
- Les classes es poden organitzar en jerarquies

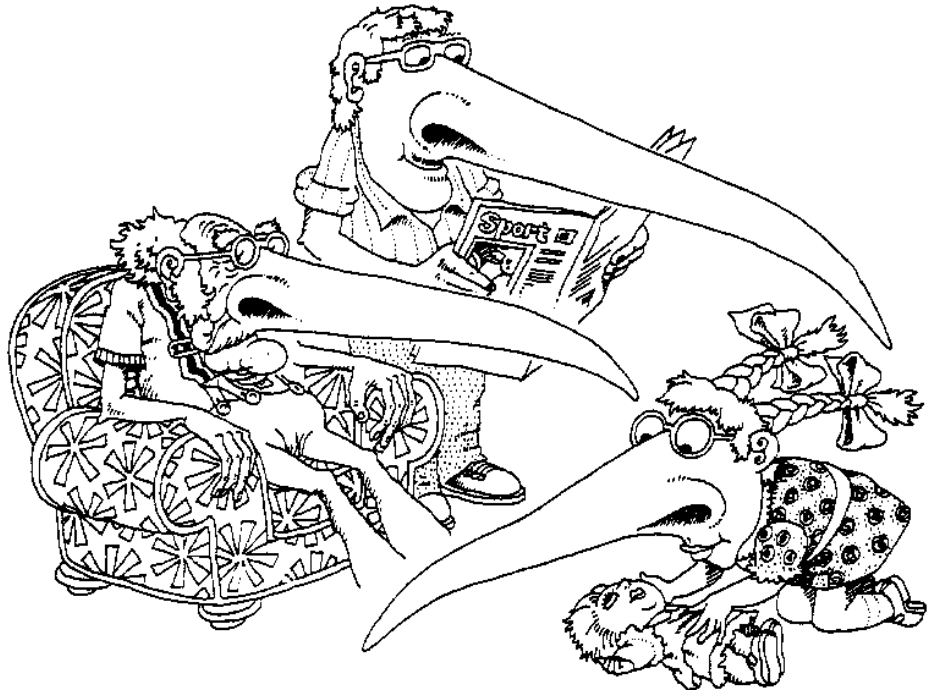


Figura extreta de la pàgina 109 del llibre: "**Object-Oriented Analysis and Design with Applications**". Grady Booch. The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.

CONCEPTES FONAMENTALS: CLASSES I OBJECTES

Objectes

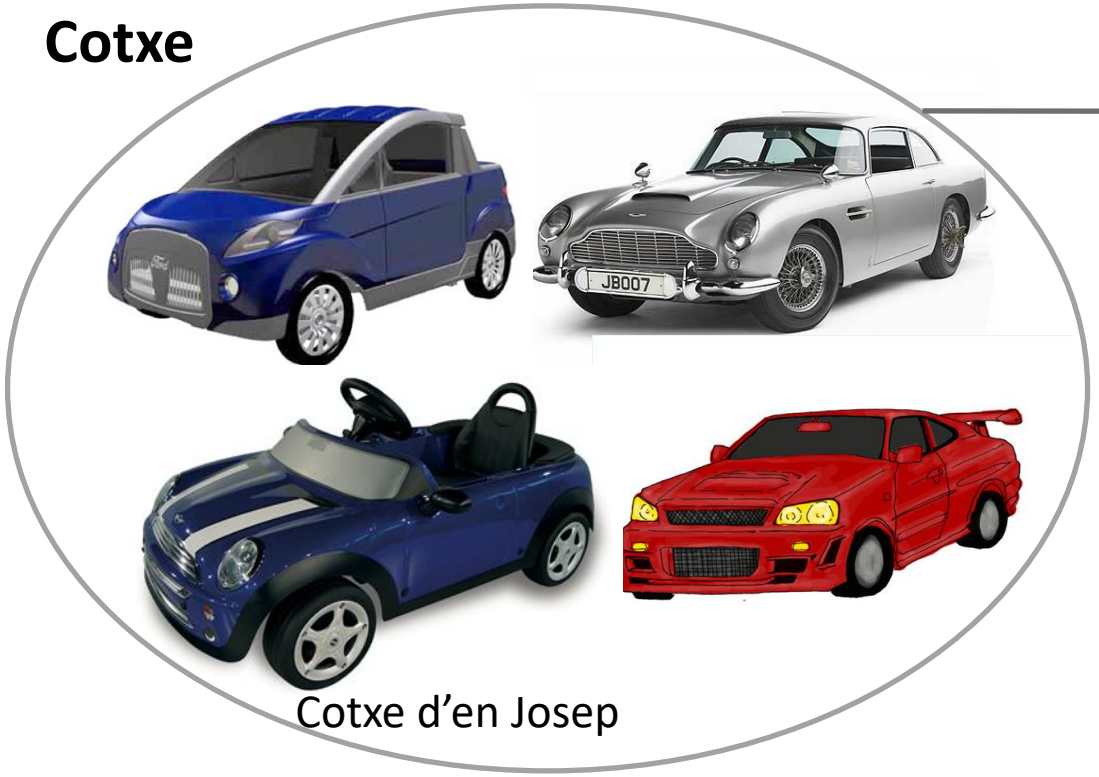
- Els objectes es poden usar per representar entitats del mon real
- Un objecte està format per:
  - Un conjunt de dades (**atributs o estats**)
  - Un conjunt d'operacions (**mètodes o comportament**)
- **El comportament d'un objecte pot modificar el seu estat**
- Cada objecte té un identificador únic pel qual pot ser referenciat.
- L'usuari no gestiona l'objecte directament → fa una petició a l'objecte d'un servei que ofereix ell mateix.

Serveis:

- Engegar
- Apagar
- Canviar d'emisora
- Pujar el volum
- Disminuir el volum

Primer és l'objecte i després la classe

Cotxe



Cotxe d'en Josep

Classe cotxe:
Patró que defineix atributs i mètodes comuns a tots els exemples de **cotxes.**

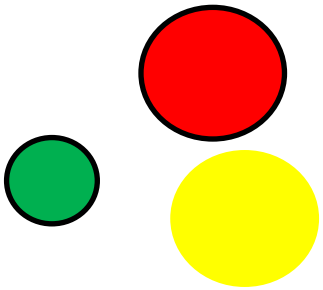
Classe cotxe
marca model color número portes

- **Classe**: descriu un grup d'entitats amb característiques comunes
- **Objecte**: descriu un membre concret del grup.

Els papers de la classe i l'objecte

- Una **classe** és una abstracció d'un cert concepte.
- Un **objecte** es defineix mitjançant una classe.
- Es diu que un objecte és una **instància** d'una classe.

Objectes de cerlces



Info abstracta →

Cercle
radi color
dibuixa borra mou

La **classe** representa un concepte:
Figura, Bicicleta,...

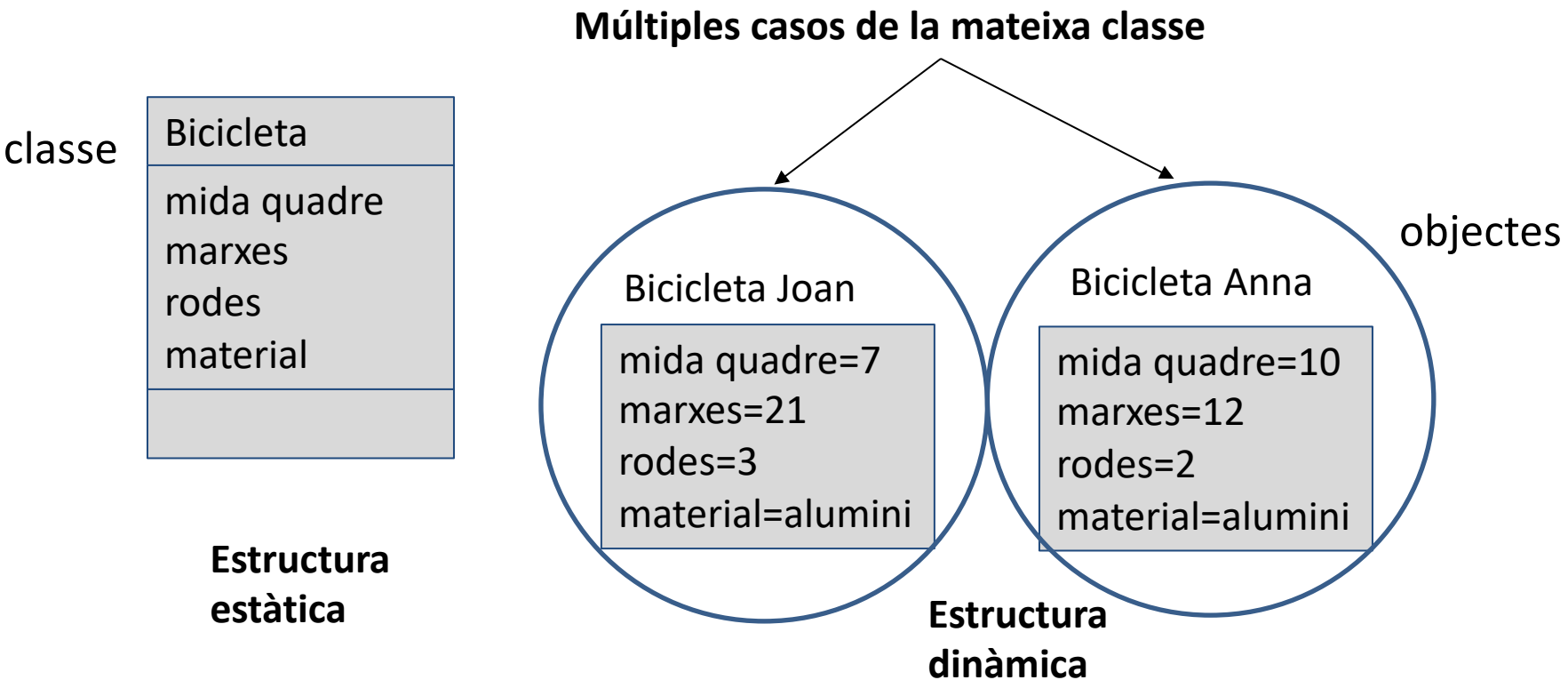
Un objecte representa la materialització d'aquest concepte: cercle de radi 2, bicicleta de color vermell,...

Objectes de bicicletes

Info abstracta →

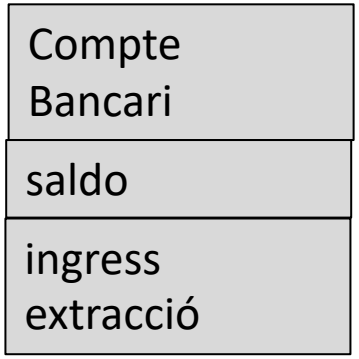
Bicicleta
tamany rodes material color
moure reparar

Objectes i classes



Objectes i classes

Una classe
(el concepte)



Múltiples objectes
de la mateixa classe

Un objecte
(la materialització,
la realització)

Compte bancari de Joan
Saldo: 5,257€

Compte bancari de Blas
Saldo: 1,245,069€

Compte bancari de Maria
Saldo: 16,833€

- L'estat d'un compte de banc inclou el seu **Saldo**
- Els comportaments associats amb un compte de banc inclouen la capacitat de fer ingressos i extraccions
- El comportament d'un objecte pot, per tant modificar el seu estat

Exemple

- Classe: Persona
- Objectes: Maria, Joan,...

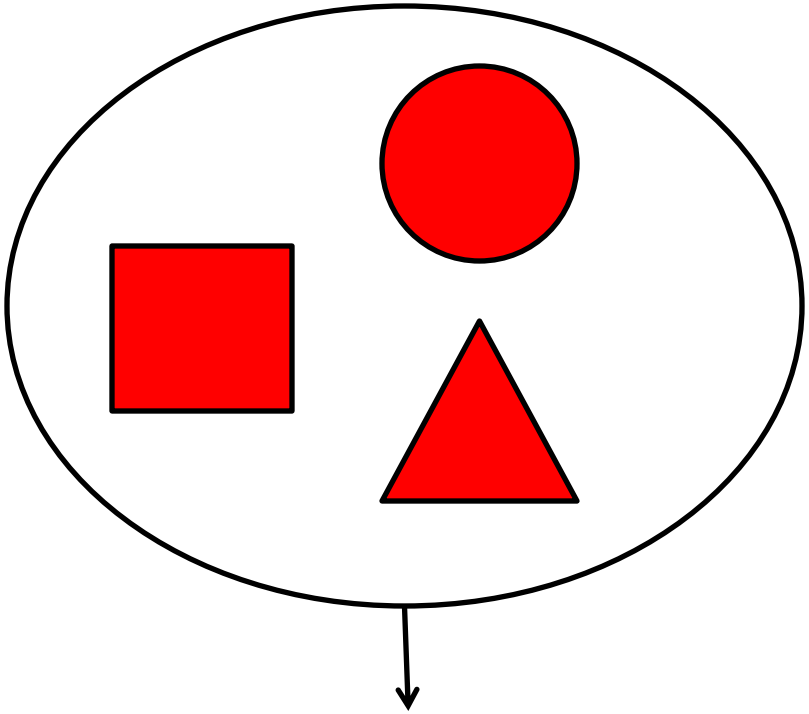
Classe Persona
nom cognoms sexe data naixement nacionalitat dni
edat

Classes

- Una classe és un plànol (o pla d'obra) d'un objecte
  - La classe utilitza els mètodes per definir els comportaments de l'objecte.
  - Es poden crear múltiples objectes d'una mateixa classe.
  - Els objectes comparteixen el nom dels atributs i les operacions, però cada objecte té un valor concret per cada atribut.
  - La classe que conté el mètode main d'un programa JAVA representa el programa complet.

Exemple

- Classe més abstracta: Figura geomètrica
- Classe: Triangle, quadrat, cercle, ...



Figures geomètriques → Dibuixar

Figura geomètrica
color posició a pantalla àrea perímetre
calcula àrea calcula perímetre retorna color assigna color

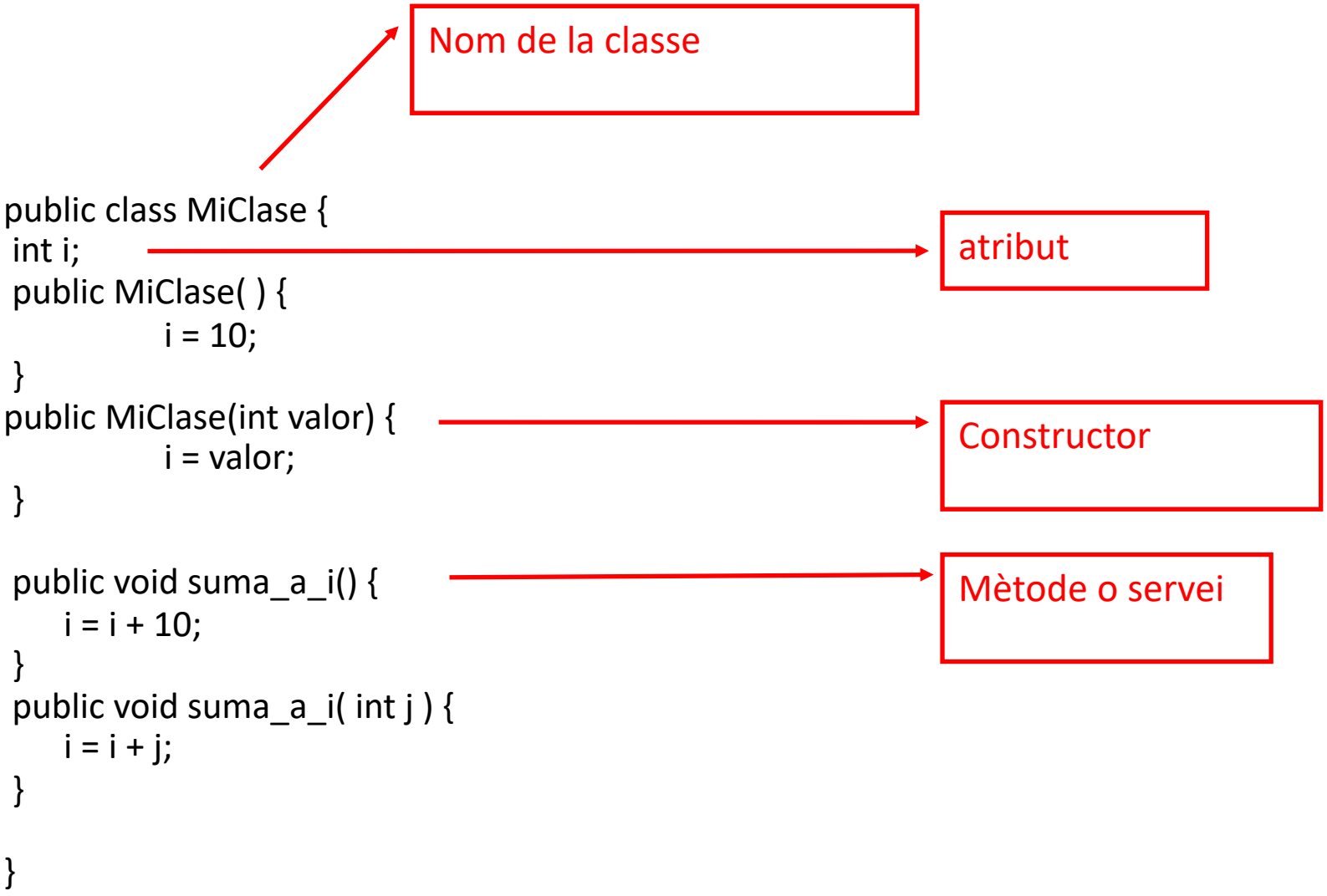
Quadrat
color posició a pantalla àrea perímetre dimensió costats

Circumferència
color posició a pantalla àrea perímetre Radi

Triangle
color posició a pantalla àrea perímetre dimensió 3costats

ÚS DE CLASSES I OBJECTES

Ús de classes



Ús de classes

```
public static void main(String[] args) {
        MiClase mc;
        mc = new MiClase();
        mc.i++;
        mc.suma_a_i(10);

        System.out.println(mc.i);
    }
```

crea una instància de la classe

Crida a un mètode de l'objecte

Ús de classes

- La **sobrecàrrega** és la capacitat de poder associar més d'un significat a un mateix identificador que apareix dins d'un programa.

- Es pot produir sobrecàrrega en:
  - Els noms dels mètodes
  - Els operadors

Ús de classes

- Sobrecàrrega del constructor de la classe

```
public class MiClase {
 int i;
 public MiClase() {
  i = 10;
  }
 public MiClase(int i) {
  this.i = i;
 // i = valor
 }
 public void suma_a_i( int j ) {
      i = i + j;
 }
}
```

La paraula **this** és una referència al objecte (l'argument implícit) sobre el que s'està aplicant el mètode.

this.i es refereix a la variable membre, mentre que **i** és l'argument del mètode.

Ús de classes

Creant Objectes

- Una variable conté un tipus primitiu o una **referència a un objecte** (reference)
- Un nom de classe pot utilitzar-se com a tipus per declarar una variable que referència a un objecte

```
String title;
MiClasse unExemple;
```

- En aquesta declaració **no** es crea cap objecte
- Una variable que referència un objecte conté l'adreça de l'objecte
- L'objecte en si mateix s'ha de crear de forma separada

Ús de classes

Creant Objectes II

- Usarem l'operador new per crear un objecte

```
String title;
title = new String("Java Software");
```

Crida al constructor de la classe String, que és un mètode especial que prepara l'objecte

Ús de classes

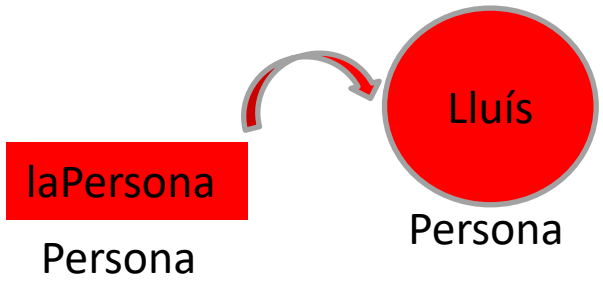
```
class Persona {
    public String nom;
}
```

Creant Objectes III

Tres passos: declaració, creació i assignació:

```
Persona laPersona = new Persona("Lluís");
```

1. Declaració d'una variable de referència
2. Creació d'un objecte
3. Assignació del nou objecte Persona a la variable laPersona.

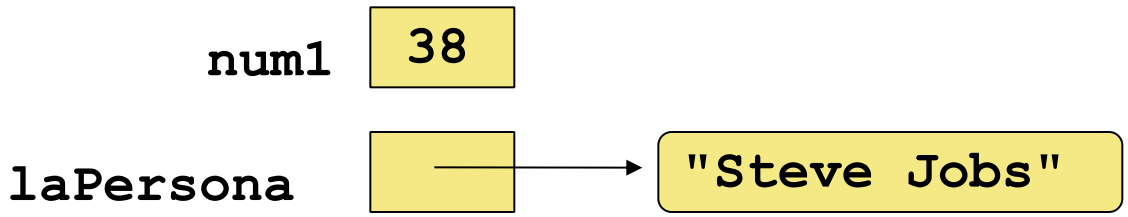


Ús de classes

Referències

- Una variable de tipus primitiu conté el valor però una variable objecte conté l'adreça de l'objecte

- Exemples : int num1=38;

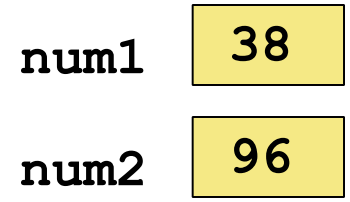


Ús de classes

Assignació

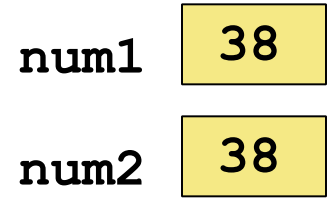
- De tipus primitius: Exemple:

Abans




```
num2 = num1;
```

Després

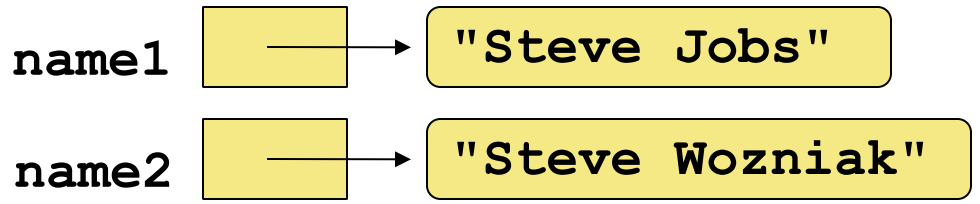


Ús de classes

Assignació de referències

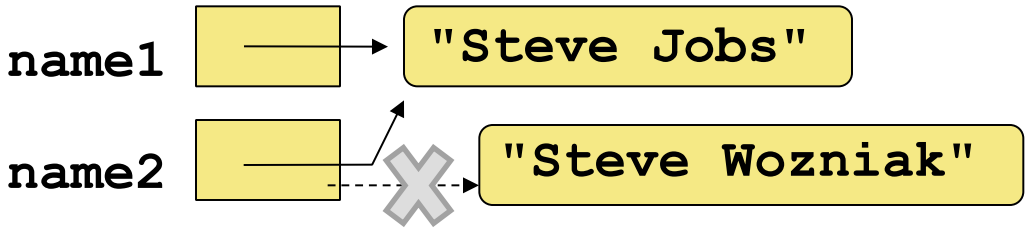
- Per referències a objectes, l'assignació còpia l'adreça

Abans




```
name2 = name1;
```

Després

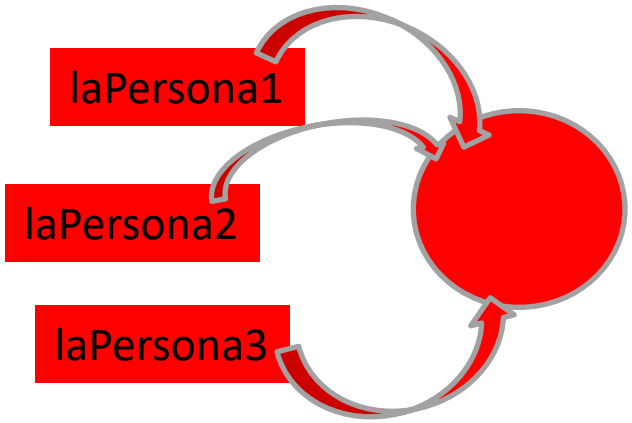


Ús de classes

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

Aliases

- Dos o més referències que es refereixen al mateix objecte s'anomenen ***aliases***
- Canviant un objecte a través d'una referència, canvia tots els seus aliases perquè realment només hi ha un objecte.




```
Persona laPersona1 = new Persona(20);
Persona laPersona2;
Persona laPersona3;
laPersona2 = laPersona1;
laPersona3 = laPersona1;
laPersona1.edat = 21;
System.out.println(laPersona3.nom
"té" + laPersona3.edat + "anys");
```

Ús de classes: Exemple Aliases

```
public class Aliases {
  public static void main(String[ ]args) {
        Persona x = new Persona();
        Persona y = new Persona();
        x.nom = "Joan";
        y.nom = "Lluís";
        Persona z;
        z = x;
        x = y;
        x.nom = "Marc";
        System.out.println("El nom en l'objecte referenciat per x és:“ + x.nom);
        System.out.println("El nom en l'objecte referenciat per y és:“ + y.nom);
        System.out.println("El nom en l'objecte referenciat per z és:“ + z.nom);
  }
}
```

```
class Persona {
   public String nom;
}
```

→ Sortida per pantalla:

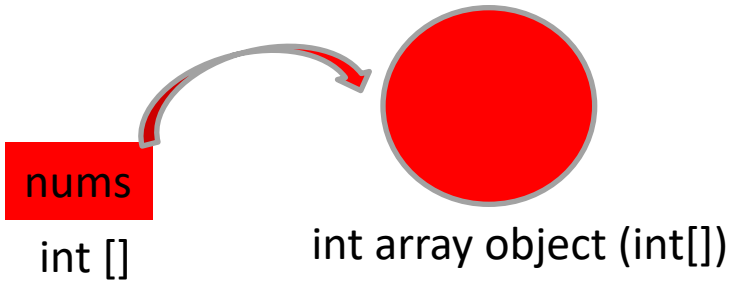
Marc
Marc
Joan

Ús de classes

- Un array també és un objecte

```
int [] nums;
nums = new int[7];

nums[0]=6;
nums[1]=19;
nums[2]=2;
nums[3]=32;
nums[4]=5;
nums[5]=15;
nums[6]=11;
```



- L'array pot contenir primitives o objectes.

Exercici :
Hi ha errors de compilació?

```
class BooksTestDrive {
    public static void main(String[] args) {
        Book []myBooks;
        myBooks = new Book [3];
        int x = 0;
        myBooks[0].title = "The Grapes of Java";
        myBooks[1].title = "The Java Gatsby";
        myBooks[2].title = "The Java CookBook";
        myBooks[0].author = "Bob";
        myBooks[1].author = "Sue";
        myBooks[2].author = "Ian";
        while (x<3){
            System.out.print(myBooks[x].title) ;
            System.out.print("by");
            System.out.print(myBooks[x].author);
            x= x+ 1;
        }
    }
}
```

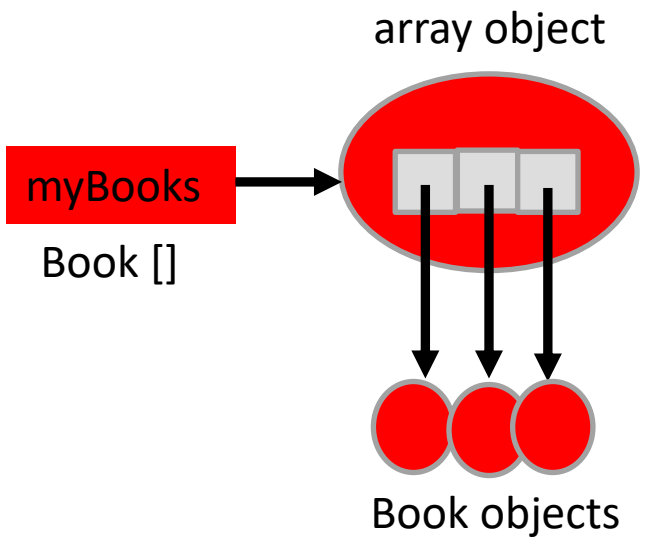
```
class Book {
        String title;
        String author;
}
```

Exercici 2: solució

```
class BooksTestDrive {
    public static void main(String[] args) {
        Book [] myBooks ;
        myBooks = new Book[3];
        int x = 0;
        myBooks[0]= new Book();
        myBooks[1]= new Book();
        myBooks[2]= new Book();
        myBooks[0].title = "The Grapes of Java";
        myBooks[1].title = "The Java Gatsby";
        myBooks[2].title = "The Java CookBook";
        myBooks[0].author = "Bob";
        myBooks[1].author = "Sue";
        myBooks[2].author = "Ian";
        while (x<3){
            System.out.print(myBooks[x].title) ;
            System.out.print("by");
            System.out.print(myBooks[x].author);
            x= x+ 1;
        }
    }
}
```

Una col.lecció és sempre un objecte

```
class Book {
        String title;
        String author;
}
```



ENCAPSULACIÓ

Encapsulació

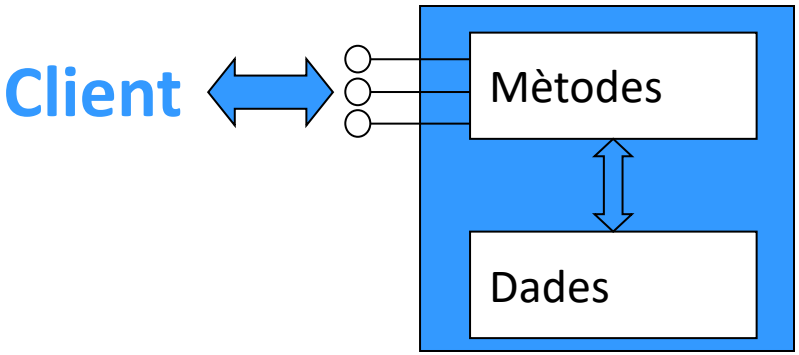
- Hi ha dues visions d'un objecte:
  - **Interna** → els detalls de les variables i dels mètodes que la classe defineix
  - **Externa** → els serveis que un objecte proveeix i com l'objecte interactua amb la resta del sistema

- Des del punt de vista extern, un objecte és una entitat encapsulada que proveeix un conjunt de serveis

- Aquest serveis defineixen la **interfície de l'objecte**

Encapsulació

- Un objecte (anomenat **client**) pot usar un altre objecte a través dels serveis que aquest proveeix (cridant als seus mètodes)
- L'objecte ha de ser auto-governat.
- Un objecte encapsulat es pot veure com una "caixa negra". La part interna s'amaga al client.
- El client invoca els mètodes de la interfície de l'objecte, que gestionen les dades de la instància.



Encapsulació: Modificadors de visibilitat

- Les **variables públiques** violen l'encapsulament i per tant s'han d'evitar
- Els **mètodes públics** es denominen **mètodes de servei**, perquè ofereixen serveis que poden ser invocats pels clients de l'objecte
- Un mètode creat només per assistir un mètode de servei es denomina **mètode de suport** i no s'ha de declarar amb visibilitat pública

Encapsulació: Modificadors de visibilitat

	public	private
Variables	Viola encapsulament	Força encapsulament
Mètodes	Serveis a clients	Soporta a altres mètodes de la classe

Els modificadors de visibilitat de Java es treballarà en la sessió de problemes.

Modificadors de visibilitat

- Nivell d'accés que es vol per a les variàbles d'instància i els mètodes:
  - **public**
  - **private**
  - **protected**
  - **friendly** (or 'default' sense declaració específica)

Modificadors de visibilitat

- **public**

  public void QualsevolPotAccedir(){}

  Qualsevol classe des de qualsevol lloc pot accedir a les variables i mètodes d'instància públics.

- **private**

  private String NumeroDelCarnetDeIdentidad;

  Les variables i mètodes d'instància privats només poden ser accedits des de dins de la classe. No són accessibles des de les subclasses.

Modificadors de visibilitat

- **friendly** (també anomenades ‘default’)

  void MetodeDelMeuPaquet(){}

  Per defecte, si no s’especifica el control d’accés, les variables i mètodes d’instància se declaren friendly (amigues).

  Són accessibles per tots els objectes dins del mateix paquet, però no per els externs al paquet.

- **protected**

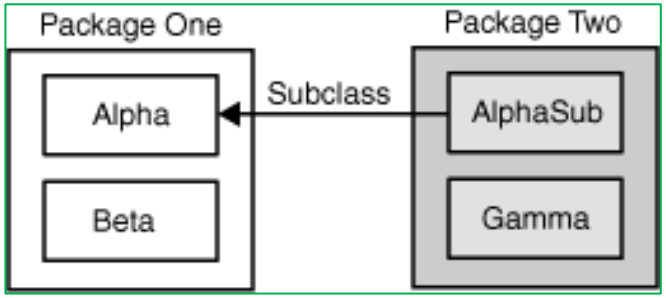
  protected void NomesSubClasses(){}

  Molt semblant a l’accés friendly, amb la següent excepció: la classe on es declara i les subclasses de la mateixa poden accedir a les variables i mètodes d’instància protegits.

Encapsulació: Modificadors de visibilitat

- L'encapsulament en Java s'aconsegueix amb els **modificadors de visibilitat**
- Un modificador és una paraula reservada Java que especifica característiques particulars d'un mètode o de les dades
  - Per exemple, el modificador **final** per definir constants
- Java té 3 paraules pels modificadors de visibilitat:
  - ***public***,
  - ***protected*** i
  - ***private***

Modificadors de visibilitat



- Visibilitat respecte a Alpha:

Modifier	Alpha	Beta	Alphasub	Gamma
public	Y	Y	Y	Y
protected	Y	Y	Y	N
friendly	Y	Y	N	N
private	Y	N	N	N

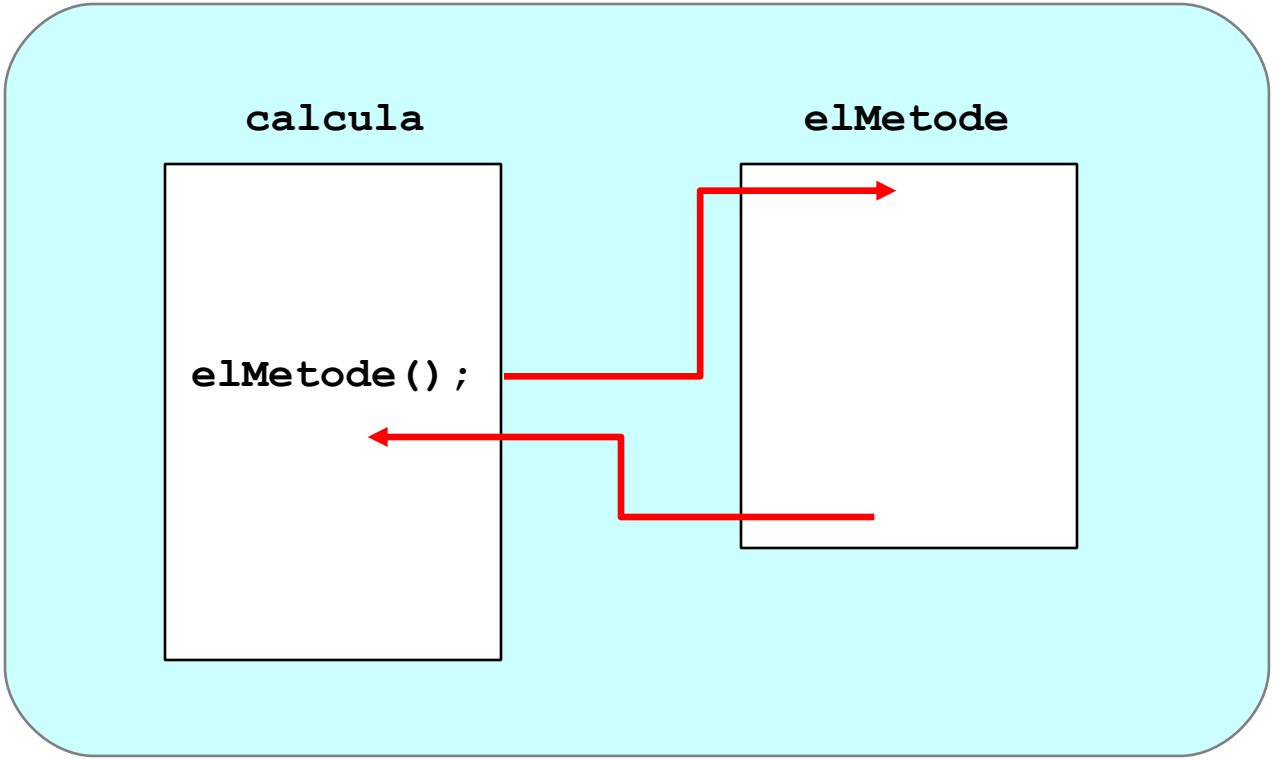
- http://download.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

Encapsulació: Declaracions de mètodes

- Una **declaració de mètode** especifica el codi que s'executarà quan el mètode sigui invocat
- Quan s'invoca un mètode, el **flux de control** salta al mètode i executa el seu codi
- Quan acaba, el flux retorna a la posició on el mètode va ser cridat i continua
- La invocació pot o no retornar un valor, depenent de com s'ha definit el mètode

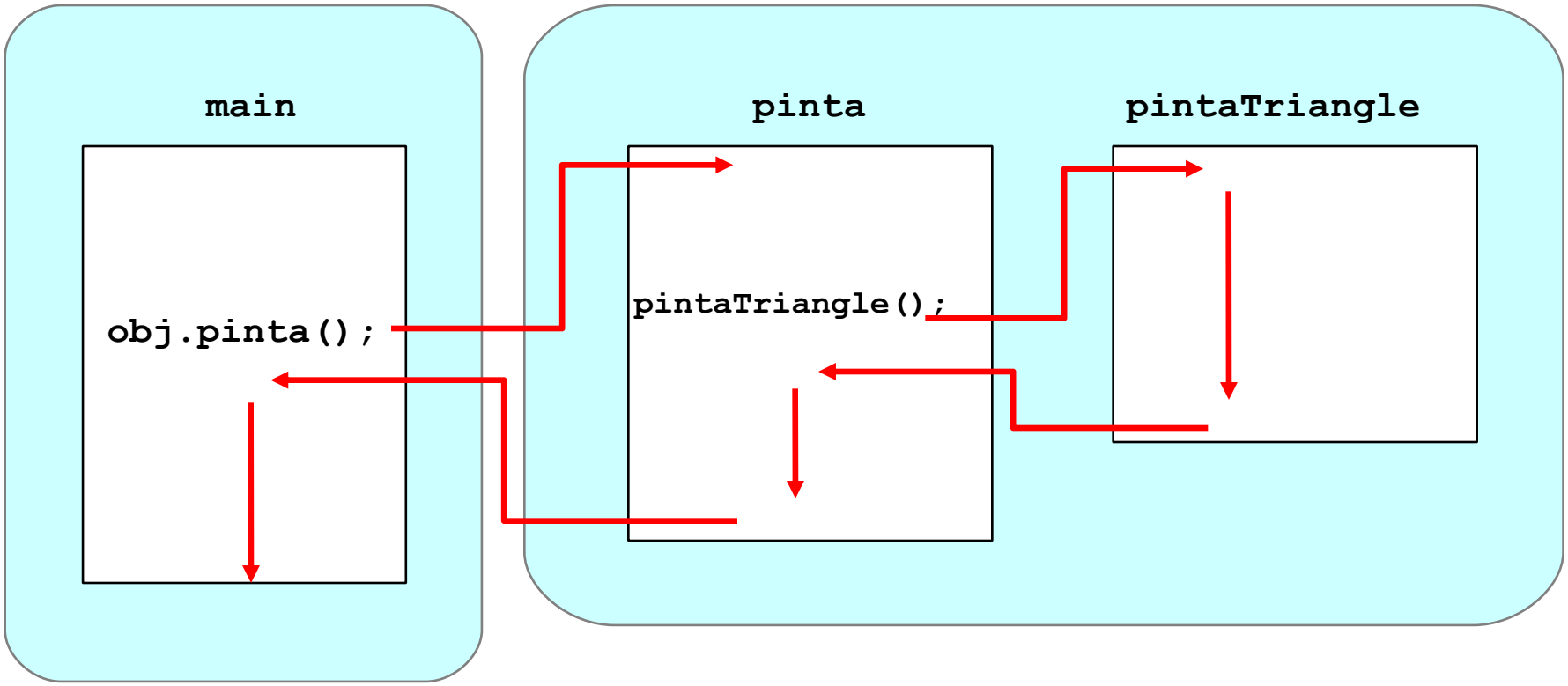
Encapsulació

Flux de control d'una invocació



Encapsulació

Flux de control d'una invocació



Referències

- Bertrand Meyer, “**Construcción de software orientado a objetos”, Prentice Hall, 1998.**

- “Software Architecture and UML” de Grady Booch (Rational Software). Presentació P. Letelier.

- Bert Bates, Kathy Sierra. **Head First Java**. O’Reilly Media, 2005.