

GRAU D'ENGINYERIA INFORMÀTICA

PROGRAMACIÓ II

CURS 12-13

Bloc 2:

Programació Orientada a Objectes (2)

Laura Igual

Departament de Matemàtica Aplicada i Anàlisi

Facultat de Matemàtiques

Universitat de Barcelona

Índex Bloc 2:

Programació orientada a objectes

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

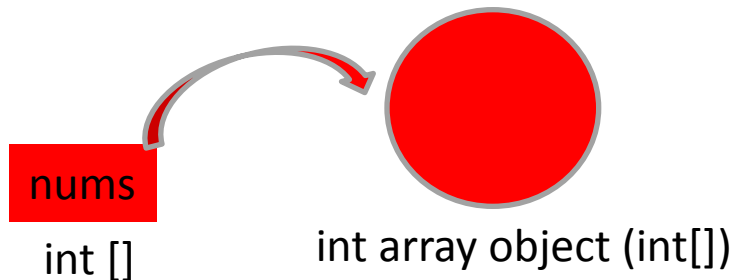
ÚS DE CLASSES I OBJECTES

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 2: Hi ha errors de compilació?

```
class BooksTestDrive {  
  
    public static void main(String[] args) {  
        Books[] myBooks;  
        myBooks = new Books[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;  
        }  
    }  
}
```

Una col.lecció és sempre un objecte

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

Exercici 2: solució

```
class BooksTestDrive {  
    public static void main(String[] args) {  
        Books[] myBooks ;  
        myBooks = new Books[3];  
        int x = 0;  
        myBooks[0]= new Books();  
        myBooks[1]= new Books();  
        myBooks[2]= new Books();  
        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 Books {  
    String title;  
    String author;  
}
```

Exercici: col·lecció

```
public class ExempleConstructor1 {  
    Figura[] figures;  
    public ExempleConstructor1(){  
        figures = new Figura[1];  
    }  
    public void metodeMostrarPrimera(){  
        Figura figura = figures[0];  
        System.out.println(" La primera figura te color: " + figura.getColor());  
    }  
}
```

```
public static void main(String[] args){  
    ExempleConstructor1 exemple = new ExempleConstructor1();  
    exemple.metodeMostrarPrimera();  
}
```

Mètode main

```
}
```

Exercici: solució

```
public class ExempleConstructor1 {  
    Figura[] figures;  
    public ExempleConstructor1(){  
        figures = new Figura[1];  
        Figura figura = new Figura();  
        figures[0]=figura;  
    }  
    public void metodeMostrarPrimera(){  
        Figura figura = figures[0];  
        System.out.println(" La primera figura te color: " + figura.getColor());  
    }  
}
```

```
public static void main(String[] args){  
    ExempleConstructor1 exemple = new ExempleConstructor1();  
    exemple.metodeMostrarPrimera();  
}
```

Mètode main

```
}
```



```

public class Dog {
    String name;
    public static void main(String[] args) {
        Dog dog1 = new Dog();
        dog1.bordar();
        dog1.name = "Bart";

        Dog[] myDogs = new Dog[3];
        myDogs[0] = new Dog();
        myDogs[1] = new Dog();
        myDogs[2] = dog1;

        myDogs[0].name = "Fred";
        myDogs[1].name = "Marge";

        System.out.print("last don't name is ");
        System.out.println(myDogs[2].name);

        int x = 0;
        while (x < myDogs.length) {
            myDogs[x].borda();
            x = x+1;
        }
    }
    public void borda() {
        System.out.println(name + "diu Guau!");
    }
    public void menja() { }
    public void casaGat() { }
}

```

Example

CONSTRUCTORS I DESTRUCTORS

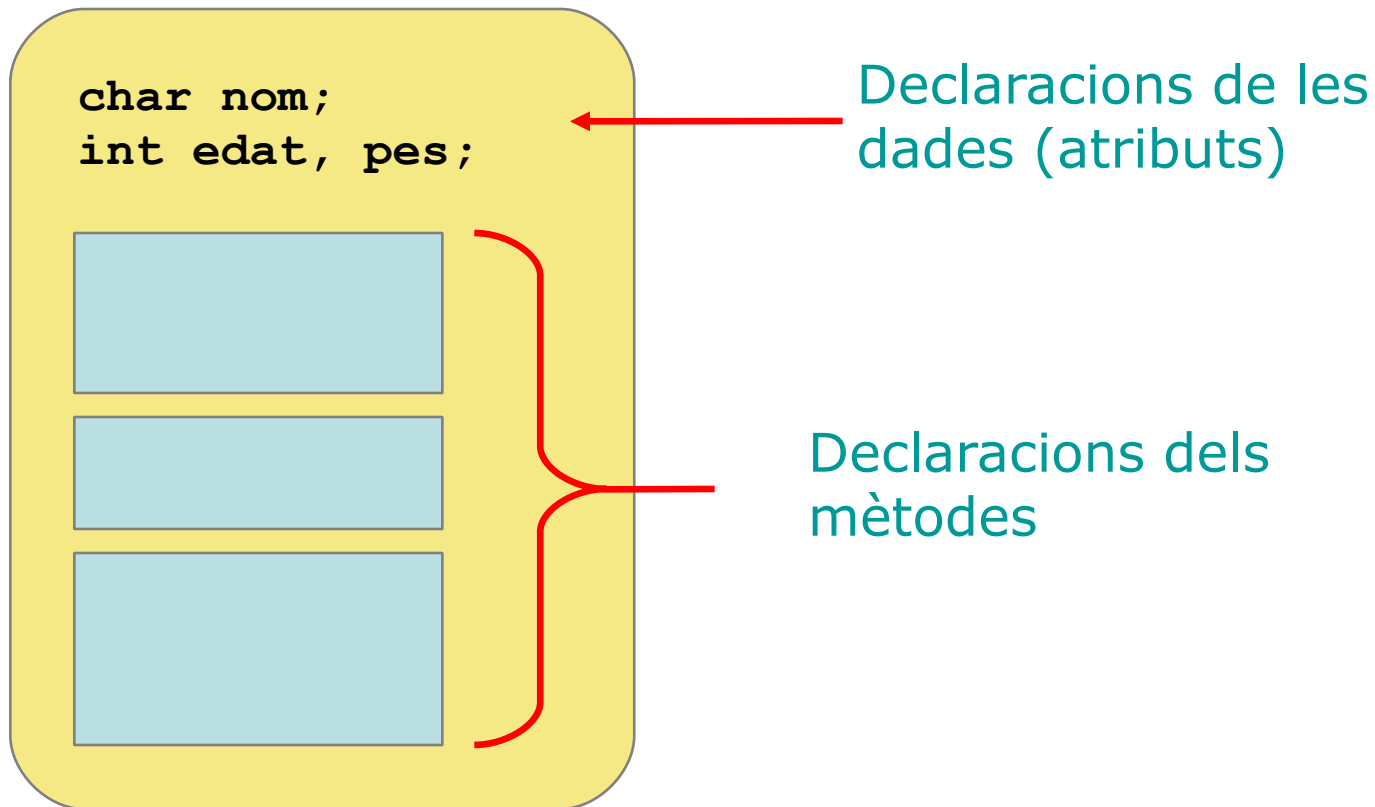
Creació de classes

- Un objecte té estat i comportament
- Per exemple, la classe Persona
 - El seu estat està definit per Nom, Edat i Pes.
 - El seu comportament principal és CanviaEdat, ConsultaNom i ConsultaEdat

Persona
nom edat pes
canviEdat consultaNom consultaEdat

Creació de classes. Encapsulació

Classe



Creació de classes. Encapsulació

- Un **constructor** és un mètode especial que inicialitza l'objecte en el moment de la seva creació.
- Té el mateix nom que la classe.
- Pot haver més d'un constructor en la mateixa classe.
- Si hi ha més d'un constructor, s'han de diferenciar:
 - O bé pel tipus de paràmetres,
 - O pel nombre de paràmetres
- El constructor s'usa per fixar un valor inicial.

Creació de classes. Encapsulació

- Un **destructor** és un mètode que realitza les tasques prèvies a l'eliminació de l'objecte.
- Una classe pot definir un mètode destructor quan, a més d'alliberar la memòria ocupada per l'objecte que s'elimina, sigui necessari especificar l'execució d'alguna operació.

Exemple (Java)

```
public class MiClase {  
    int i;  
    public MiClase() {  
        i = 10;  
    }  
    public void suma_a_i( int j ) {  
        i = i + j;  
    }  
    // Tanca el canal quan l'objecte és reciclat  
    protected void finalize() {  
        close();  
    }  
}
```

Mètode que es crida
automàticament quan es
destrueix l'objecte

Exemple (Java)

```
public class MiClase {  
    int i;  
    public MiClase() {  
        i = 10;  
    }  
    ...  
  
    public void finalize() {  
        // Fer alguna tasca per eliminar  
        System.out.println("Destructor de la classe A");  
    }  
}
```

En aquest cas no hem efectuat cap tasca dins el mètode destructor, atès que en el mètode constructor no hem demanat recursos, però és una cosa que hem de tenir en compte per tal de crear aplicacions correctes.

Creació de classes. Encapsulació

Àmbit de dades

- Les dades **declarades a nivell de classe** poden ser referenciades per tots els mètodes de la classe
- Les dades **declarades dins d'un mètode** només es poden usar en aquest mètode (són **locals**)

Exemple

```
public class MiClase {  
    int i; ← Variable d'instància  
    public MiClase() {  
        i = 10;  
    }  
    public void suma_a_i(int j) {  
        for(int k=0; k<j; k++){  
            i += k;  
        }  
    }  
}
```

← Variable local

Creació de classes

Exemple

Considerem un dau de sis cares.

- El seu estat està definit per la cara superior.
- El seu comportament principal és que pot ser llançat.

Exemple

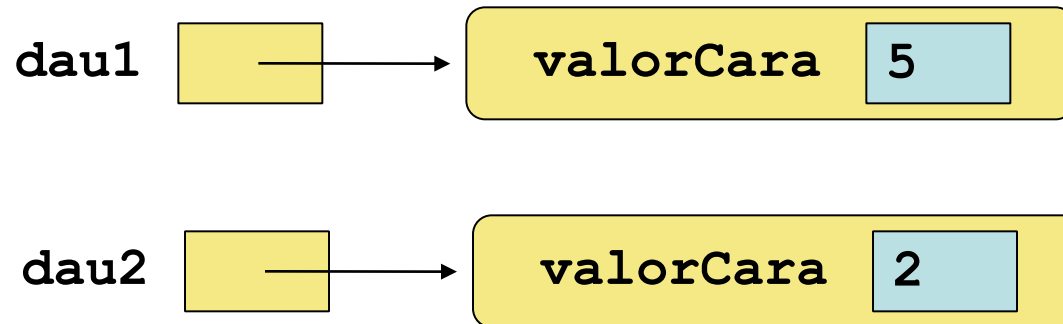
```
public class Dau{

    private final int MAX = 6; // valor de cara màxim
    private int valorCara; // valor actual mostrat al dau.

    // constructor
    public Dau() {
        valorCara = 1;
    }
    // Llançar el dau i tornar el resultat
    public int llansa() {
        valorCara = (int) (Math.random() * MAX) +1;
        return valorCara;
    }
    // Modificador de valorCara
    public void setValorCara(int valor) {
        valorCara = valor;
    }
    // Consultor de valorCara
    public int getValorCara() {
        return valorCara;
    }
}
```

Creació de classes.

Dades d'Instància



Cada objecte té la seva pròpia variable `valorCara`, i per tant el seu propi estat

Creació de classes.

Dades d'Instància

- Les dades d'instància són les variables que cada instància (objecte) té una versió pròpia d'ella
- Una classe declara el tipus de les dades, però no reserva espai per ells
- Els objectes comparteixen les definicions de mètodes, però cada objecte té el seu propi espai de dades

→ Cada vegada que es crea un objecte Persona, es crea també una variable Edat i Nom.

Persona
nom edat pes
canviEdat consultaNom consultaEdat

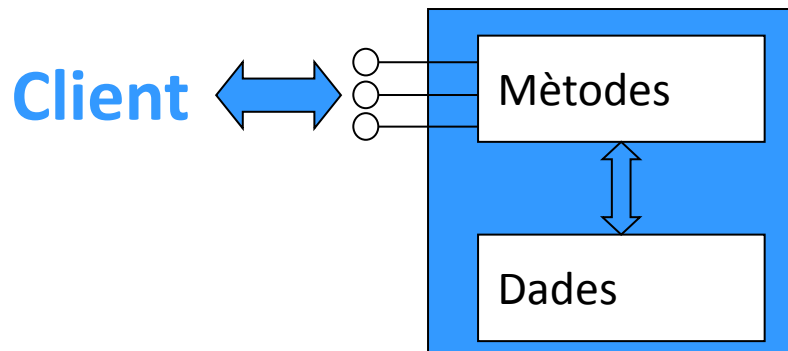
ENCAPSULACIÓ

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

- 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*

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

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 NumeroDelCarnetDeldentidad;  

```

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.

Exemple 1: Modificadors de visibilitat

```
package unPaquet;
```

```
public class A {
```

```
    private int x;
```

```
    public A() {
```

```
        x=1;
```

```
    }
```

```
}
```

```
package unPaquet;
```

```
public class C {
```

```
    A a;
```

```
    public C(){
```


```
        a=new A();
```

```
    }
```

```
    public void meteodeC(){
```

```
        System.out.println("el valor de a és:" + a.x);
```

```
    }
```



Error de compilació.
La variable privada x
no és visible des d'un
altra classe.

Exemple 1: Modificadors de visibilitat

```
package unPaquet;
```


```
public class A {  
    private int x;  
    public A() {  
        x=1;  
    }  
}
```

```
public int getx(){  
    return this.x;  
}  
public void setx(int x){  
    this.x=x;  
}
```

```
}
```

```
package unPaquet;
```

```
public class C {  
    A a;  
    public C(){  
        a=new A();  
    }  
    public void meteodeC(){  
        System.out.println("el valor de a és:" + a.getx());  
    }  
}
```



L'accés es fa
mitjançant el
mètode get.

Exemple 2: Modificadors de visibilitat

```
package unPaquet;
```

```
public class A {
```

```
    public int x;
```

```
    public A() {
```

```
        x=1;
```

```
    }
```

```
}
```

```
package unAltrePaquet;
```

```
Import unPaquet.A;
```

```
public class C {
```

```
    A a;
```

```
    public C(){
```

```
        a=new A();
```

```
    }
```

```
    public void meteodeC(){
```

```
        System.out.println("el valor de a és:" + a.x);
```

```
    }
```

No hi ha error de compilació.
La variable pública x és visible des de qualsevol classe

Exemple 3: Modificadors de visibilitat

```
package unPaquet;
```

```
public class A {  
    int x;  
    public A() {  
        x=1;  
    }  
}
```

```
package unAltrePaquet;  
Import unPaquet.A;
```

```
public class B extends A {  
    public B() {  
        System.out.println("Constructor de B");  
    }  
    public void metodeB() {  
        System.out.println("el valor de x és:" + this.x);  
    }  
}
```

Error de compilació.
La variable friendly x
no és visible des d'un
altre paquet.

Exemple 3: Modificadors de visibilitat

```
package unPaquet;
```

```
public class A {
```

```
    protected int x;
```

```
    public A() {
```

```
        x=1;
```

```
    }
```

```
}
```

```
package unAltrePaquet;
```

```
Import unPaquet.A;
```

```
public class B extends A {
```

```
    public B() {
```

```
        System.out.println("Constructor de B");
```


```
    }
```

```
    public void metodeB() {
```

```
        System.out.println("el valor de x és:" + this.x);
```

```
    }
```

```
}
```

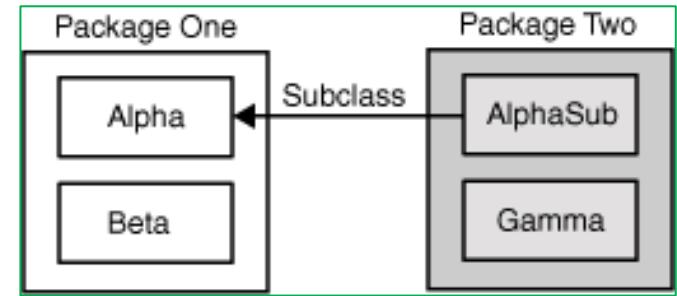


Ara aquest accés és correcte.

La variable protected x és visible per una subclasse de A.

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: Consultors i modificadors

- Si les dades són privades, com accedim a elles? Com les modifiquem?
 - Un mètode **consultor** retorna el valor actual d'una variable
 - Un mètode **modificador** canvia el valor d'una variable
- Sovint, el nom dels mètodes consultor i modificador són *getX* i *setX*, on X és el nom del valor
- Sovint s'anomenen “getters” i “setters”

Encapsulació: Restriccions en els modificadors

- Els modificadors permeten introduir restriccions en els valors que s'assignen als atributs (p.e. Que es trobi dins dels límits)
- Exemple, en el cas d'un Dau:
 - que els numero que toca estigui entre 1 i MAX.

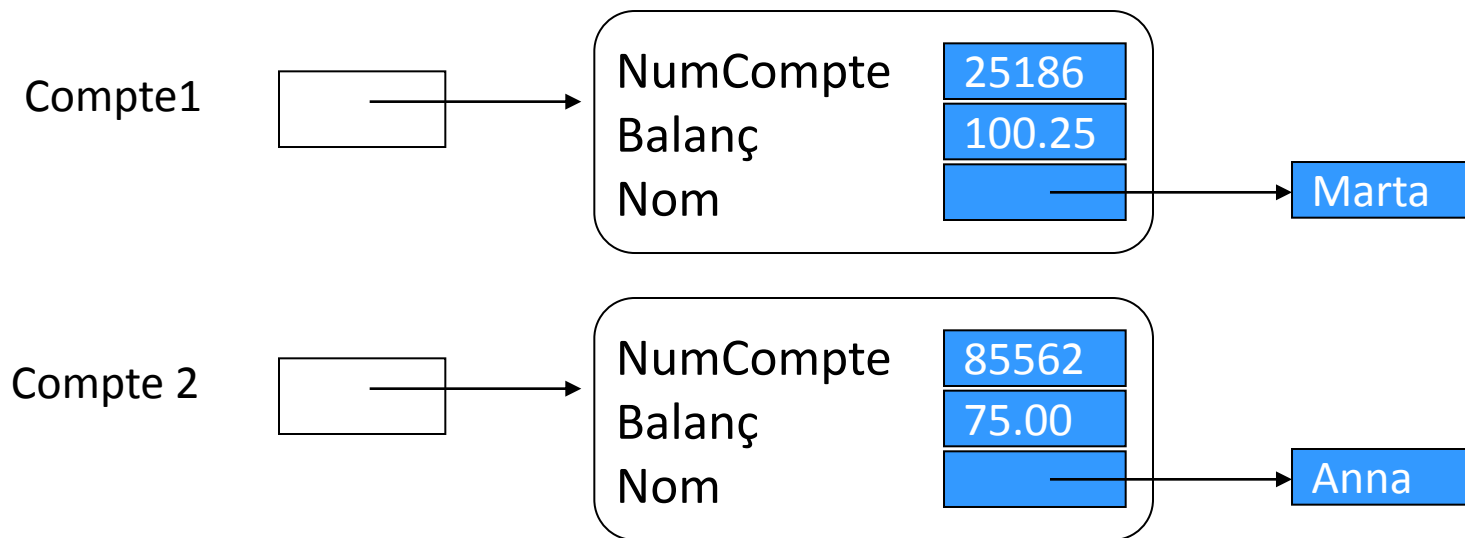
Exemple encapsulació

Compte bancari

- Representem un compte bancari mitjançant una classe **Account**
- El seu estat inclou el numero de compte, el saldo actual i el nom del propietari
- Els serveis són: afegir o extraure diners i afegir interessos

Exemple encapsulació

Compte bancari



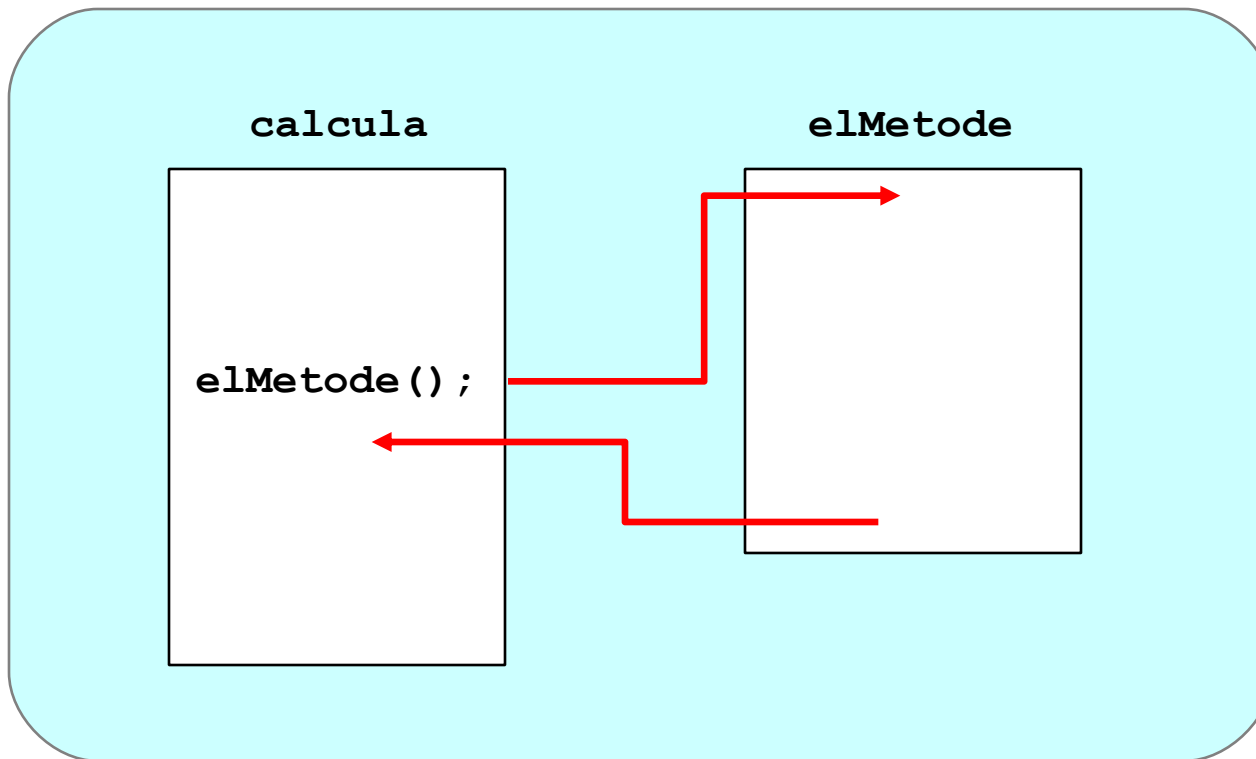
Exercici: Implementeu la classe **Account** en JAVA

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

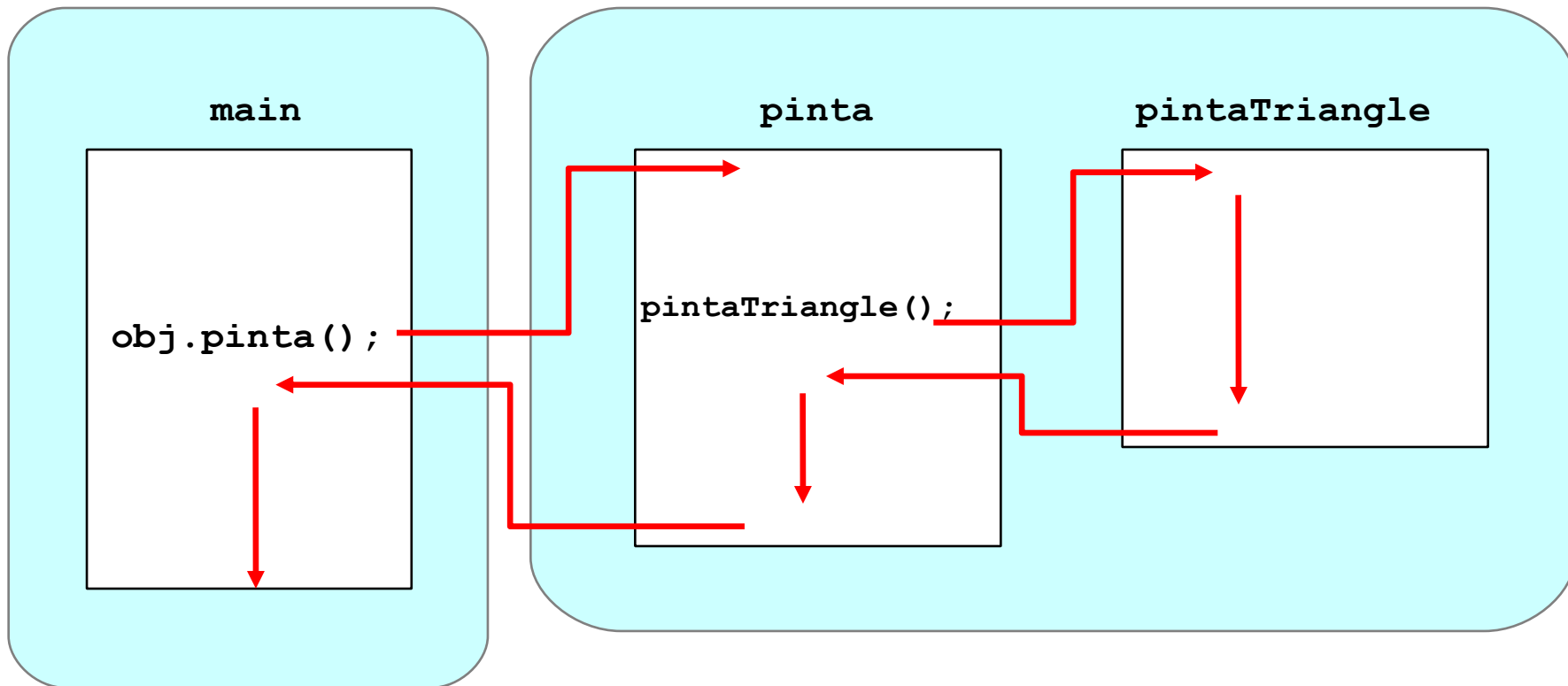
Encapsulació

Flux de control d'una invocació



Encapsulació

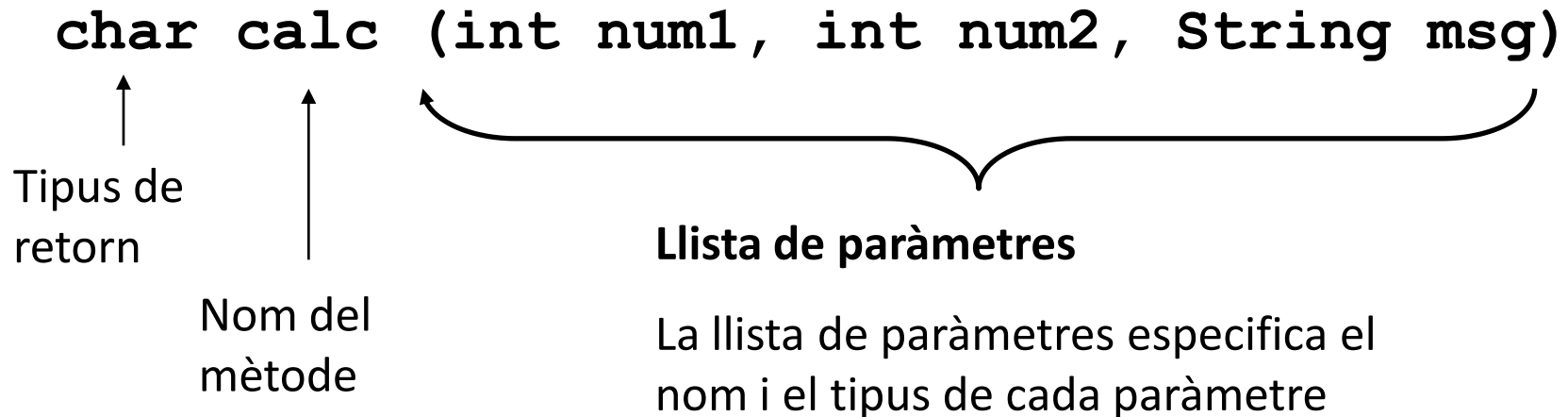
Flux de control d'una invocació



Encapsulació

Capçalera d'un mètode

- La declaració d'un mètode s'inicia amb una capçalera de mètode



Encapsulació


Cos d'un mètode

```
char calc (int num1, int num2, String msg)
{
    int sum = num1 + num2;
    char result = msg.charAt(sum) ;
return result;
}
```

sum i **result** són dades locals

Creades cada cop que es crida al mètode i destruïdes quan s'acaba la seva execució

L'expressió de retorn
ha de ser consistent
amb el tipus de retorn



Encapsulació

La instrucció return

- El tipus de retorn d'un mètode indica el tipus de valor que el mètode retorna al que crida
- Un mètode que no retorna un valor té un tipus de retorn ***void***
- Una instrucció return especifica el valor que es retornarà

return expression;

Creació de classes. Encapsulació

Paràmetres

Quan es crida a un mètode. Els paràmetres actuals es **copien** en els paràmetres formals

`ch = obj.calc (25, count, "Hello");`

arguments

Passa per valor
o passa per copia

`char calc (int num1, int num2, String message)`

paràmetres

{

`int sum = num1 + num2;`

`char result = message.charAt (sum);`

`return result;`

}

Encapsulació: Declaració i inicialització de les variables d'instància.

- Les variables d'instància (o atributs) sempre tenen un valor per defecte encara que no li assignes explícitament cap valor o no cridis a un mètode setter.
- Exemples:
 - integers 0
 - floating points 0.0
 - booleans false
 - referencies null

Diferència entre variable d'instància i variable local (recordatori)

- Les variables locals estan declarades dins d'un mètode.
- Les variables locals no tenen un valor per defecte, s'han d'inicialitzar abans d'utilitzar-les, sinó donarà un error de compilació.

```
class Exemple {  
    public void fer(){  
        int x;  
        int z = x+3; ← No compilarà  
    }  
}
```

Dades locals

- Els paràmetres formals d'un mètode creen automàticament variables locals quan s'invoca el mètode
- Quan el mètode finalitza, totes les variables locals es destrueixen
- Les variables d'instància, declarades a nivell de la classe, existeixen mentre existeixi l'objecte

Encapsulació

Més sobre constructors

- Un constructor no pot retornar cap tipus, ni tan sols void
- A cada classe X se li assigna un constructor per defecte X().
- Si es defineix qualsevol altre constructor, el constructor per defecte ja no existeix.
- Un constructor pot cridar a un altre constructor usant `this(x,y,z)`, tot i que no és recomanable. Si és necessari reutilitzar codi, ambdós constructors poden cridar a un mètode de suport dins de la classe.

Exercicis de repàs

1. Enumereu el tipus de mètodes que coneixes i feu-ne una petita descripció.
2. Creieu que és viable tenir un mètode accessor de lectura (o consultors) amb visibilitat privada?
3. Creieu que té sentit que tots els mètodes accessors d'escriptura (o modificadors) tinguin visibilitat pública?
4. Quina diferència hi ha entre les responsabilitats de classe i les d'instància?
5. Poseu un exemple de responsabilitats de classe, diferent dels constructors i destructors, per a una classe Vehicle.
6. Per quin motiu el constructor ha de ser un mètode amb responsabilitat de classe i no ho pot ser amb responsabilitat d'instància?

Example

```
public class PoorDogTestDrive {  
    public static void main(String[] args)  
    {  
        PoorDog one = new PoorDog();  
        System.out.println("Dog size is " + one.getSize());  
        System.out.println("Dog name is " + one.getName());  
    }  
}
```

```
class PoorDog  
{  
    private int size;  
    private String name;  
    public int getSize() { return size;}  
    public String getName() { return name;}  
}
```

Example

```
public 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();  
    }  
}
```

```
class GoodDog  
{  
    private int size;  
    public int getSize() { return size;}  
    public void setSize(int s) { size = s; }  
    void bark() {  
        if (size < 60) {  
            System.out.println("Wooof! Wooof!");  
        } else if (size > 60) {  
            System.out.println("Ruff! Ruff!");  
        } else {  
            System.out.println("Yip! Yip!");  
        }  
    }  
}
```

Exercici 1

Busca els errors dins d'aquests codis:

A

```
class Platina{
boolean potGravar = false;
    void playCinta() {
        System.out.println("tape playing");
    }
    void gravaCinta() {
        System.out.println("tape recording");
    }
}

class PlatinaTest {
    public static void main(String [] args) {
        t.potGravar = true;
        t. playCinta();
        if (t.potGravar == true) {
            t.gravaCinta();
        }
    }
}
```

B

```
class DVDPlayer {
    boolean potGravar = false;

    void recordDVD() {
        System.out.println("DVD gravant");
    }
}

class DVDPlayerTest {
    public static void main(String [] args) {
        DVDPlayer d = new DVDPlayer();
        d.potGravar = true;
        d.playDVD();
        if (d.potGravar == true) {
            d.recordDVD();
        }
    }
}
```


Solució 1

A

```
class Platina{
    boolean potGravar = false;
    void playCinta() {
        System.out.println("tape playing");
    }
    void gravaCinta() {
        System.out.println("tape recording");
    }
}
```

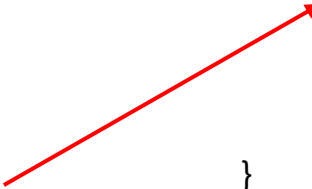
```
class PlatinaTest {
    public static void main(String [] args) {
        Platina t = new Platina( );
        t.potGravar = true;
        t.playCinta();
        if (t.potGravar == true) {
            t.gravaCinta();
        }
    }
}
```

B

```
class DVDPlayer {
    boolean potGravar = false;
    void recordDVD() {
        System.out.println("DVD gravant");
    }
    void playDVD ( ) {
        System.out.println("DVD playing");
    }
}
```

```
class DVDPlayerTest {
    public static void main(String [] args) {
        DVDPlayer d = new DVDPlayer();
        d.potGravar = true;
        d.playDVD();
        if (d.potGravar == true) {
            d.recordDVD();
        }
    }
}
```

La línia
d.playDVD();
no compilaria sense el mètode



Exercici 2

Composa el codi a partir dels trossos:

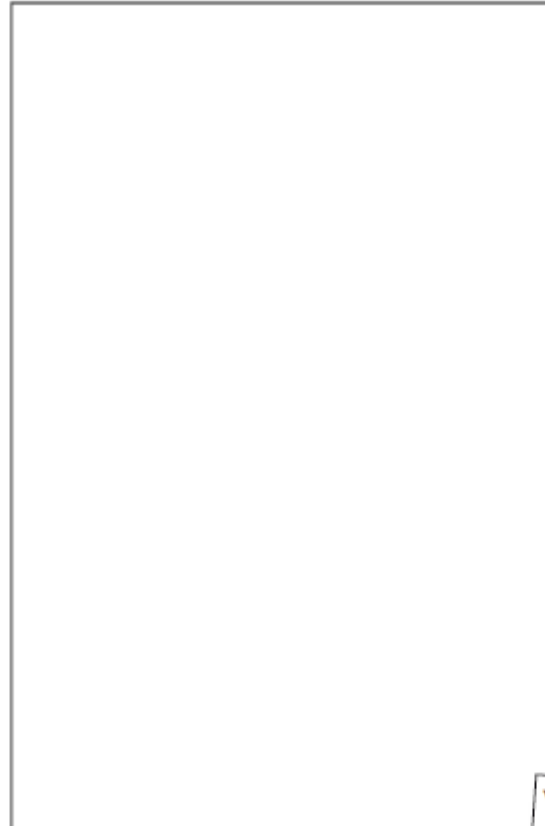


Exercise



Code Magnets

A Java program is all scrambled up on the fridge. Can you reconstruct the code snippets to make a working Java program that produces the output listed below? Some of the curly braces fell on the floor and they were too small to pick up, so feel free to add as many of those as you need.



```
File Edit Window Help Demo
% java DrumKitTestDrive
bang bang ba-bang
ding ding da-ding
```

```
d.playSnare();
```

```
DrumKit d = new DrumKit();
```

```
boolean topHat = true;
```

```
boolean snare = true;
```

```
void playSnare() {
    System.out.println("bang bang ba-bang");
}
```

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

```
if (d.snare == true) {
    d.playSnare();
}
```

```
d.snare = false;
```

```
class DrumKitTestDrive {
```

```
d.playTopHat();
```

```
class DrumKit {
```

```
void playTopHat () {
    System.out.println("ding ding da-ding");
}
```

Solució

```
class DrumKit {
    boolean topHat = true;
    boolean snare = true;

    void playTopHat() {
        System.out.println("ding ding da-ding");
    }
    void playSnare() {
        System.out.println("bang bang ba-bang");
    }
}

class DrumKitTestDrive {
    public static void main(String [] args) {
        DrumKit d = new DrumKit();
        d.playSnare();
        d.snare = false;
        d.playTopHat();
        if (d.snare == true) {
            d.playSnare();
        }
    }
}
```

```
% java DrumKitTestDrive
bang bang ba-bang
ding ding da-ding
```

MES SOBRE US DE LES CLASSES

Ús de classes

Llibreries de classes

- Una llibreria de classes és una col·lecció de classes que podem usar per desenvolupar programes.
- La llibreria de classes standard de Java forma part de qualsevol entorn de desenvolupament Java.
- Es poden obtenir llibreries de tercers o construir-les.

Ús de classes

La declaració import

- Podem usar una classe usant el seu nom completament qualificat:

```
java.util.Scanner
```

- O podem importar la classe i després utilitzar només el nom de la classe:

```
import java.util.Scanner;
```

- Per importar totes les classes d'un paquet es pot utilitzar el *

```
import java.util.*;
```

Ús de classes

La declaració import II

- Totes les classes del paquet `java.lang` s'importen automàticament en tots els programes.
- Equivalent a:

```
import java.lang.*;
```
- Exemples: `System` o `String`;

Ús de classes

Classes Wrapper

- El paquet java.lang conté classes wrapper (envoltori) que es corresponen amb cada tipus primitiu:

<u>Classe Wrapper</u>	<u>Tipus primitiu</u>
Byte	byte
Short	short
Integer	int
Long	long
Float	float
Double	double
Character	char
Boolean	boolean
Void	void

Ús de classes

Classes Wrapper II

- La següent declaració crea un objecte Integer que representa un enter 40 com un objecte

Integer age = new Integer(40) ;

- Un objecte d'una classe *wrapper* pot ser usat en qualsevol situació on un valor primitiu no pot. Per exemple es pot emmagatzemar en un contenidor d'objectes.

Ús de classes

Classes Wrapper III

- Les classe *wrapper* també contenen mètodes estàtics que ajuden a gestionar el tipus associat
- Per exemple, la classe `Integer` conté un mètode per convertir un enter en un `String` al seu valor int:

```
num = Integer.parseInt(str) ;
```

- Les classes wrapper contenen constants molt útils
- La classe `Integer` té `MIN_VALUE` i `MAX_VALUE` que mantenen el major i menor nombre que es pot guardar en un `int`

Conversió de dades

- Algunes vegades es convenient convertir dades d'un tipus a un altre.
- Aquestes conversions no canvien el tipus de la variable o el valor que s'emmagatzema. Només converteixen el valor en aquella part del càlcul.

Conversió de dades

- Les conversions s'han de gestionar amb cura per evitar perdre informació.
- Widening conversions (conversions d'ampliació) són les més segures, ja que van de tipus de dades petit a gran.
- Narrowing conversions (conversions per reducció) poden perdre informació, ja que van de tipus gran a petit.

Conversió de dades

Casting

- *Casting* és la tècnica de conversió més poderosa i perillosa, ja que permet conversions d'ampliació i reducció.
- Per realitzar un ***cast***, el tipus es col·loca entre parèntesis davant del valor que es desitja convertir
- Exemple:

```
int total;  
int compt;  
float resultat;  
resultat = (float) total / compt;
```

Conceptes bàsics explicats

- Ús de classe i objecte:
 - Atributs,
 - Mètodes d'objecte i de classe
 - Instànciació
 - Sobrecàrrega
 - Variables de tipus primitiu i variables referències
 - Declaració de variables, creació d'objectes i assignació de referències
 - Aliases
 - Signatura del mètode
 - Constructors i destructors
 - Àmbit de dades
 - Dades d'instància
 - Encapsulació
 - Modificadors de visibilitat
 - Conversió de dades
 - ...

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.