INTRODUCCIÓ A LA PROGRAMACIÓ ORIENTADA A OBJECTES I A ESDEVENIMENTS

Laura Igual

Departament de Matemàtica Aplicada i Anàlisi
Facultat de Matemàtiques
Universitat de Barcelona

Temari

INTRODUCCIÓ A LA PROGRAMACIÓ ORIENTADA A OBJECTES I A ESDEVENIMENTS

Bloc 1:

Mòdul i abstracció de dades

Índex Bloc 1:
Mòdul i abstracció de dades

1. Introducció
2. Complexitat intrínseca de les aplicacions
3. Descomposició de problemes complexos
4. Factors de qualitat
5. Modularitat
    1. Descomposició funcional
    2. Descomposició basada en objectes
6. TADs
7. Exemples de metodologies del software.

INTRODUCCIÓ

Introducció

- Anys 50,
  - Apareixen els primers llenguatges de programació.
  - Programes petits utilitzant assaig i error
- Finals dels anys 60:
  - Hardware més potent ➡ augmenta la dificultat dels problemes abordats
  - Sistemes grans sense estructuració ➡
    - Impossibles de mantenir
    - Molt costosos i escassament fiables
  - Necessitat d'ordenar i detallar el camí de resolució del problema abans de començar a programar.
  - Apareix l'*Enginyeria del Software*

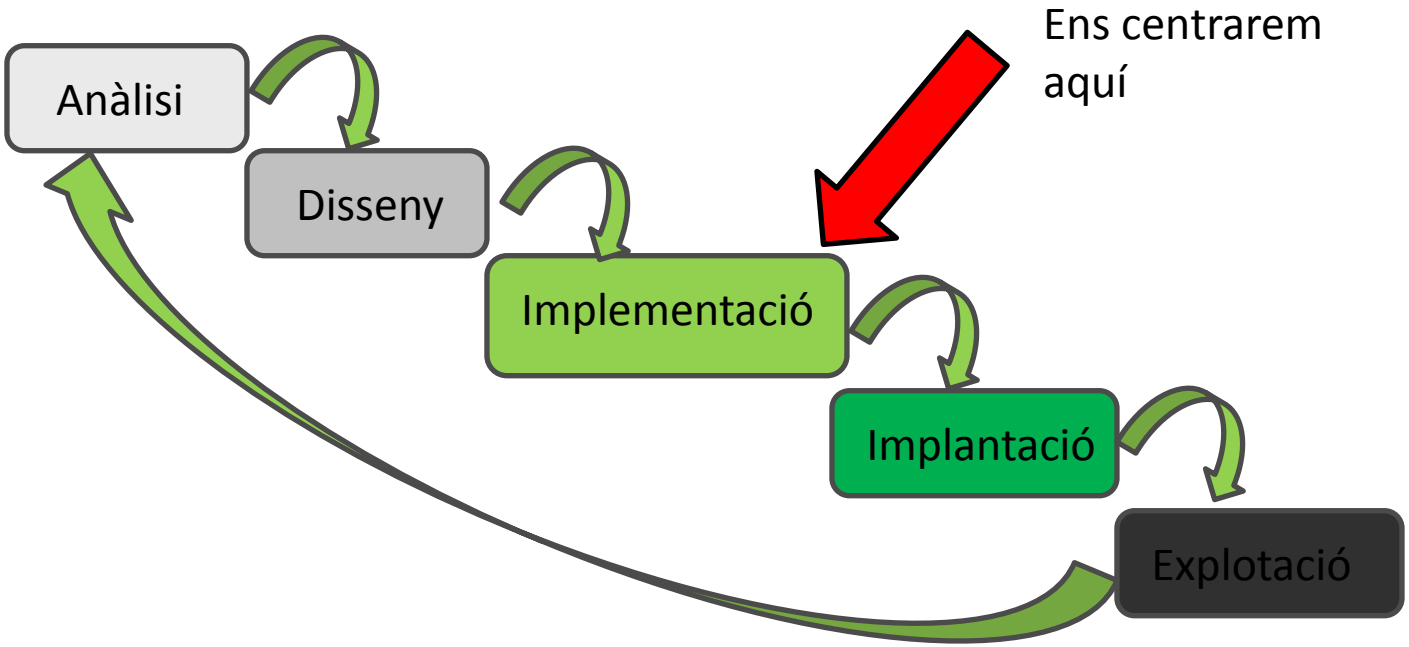
    ... evoluciona fins avui en dia

Introducció

- L'objectiu general de la **Enginyeria del Software** és produir software de qualitat
- Per qualitat s'entén l'adequació del software als requisits exigits
- El camí per a obtenir software de qualitat és mitjançant un plantejament rigorós del problema

Introducció

- Cicle de vida del software



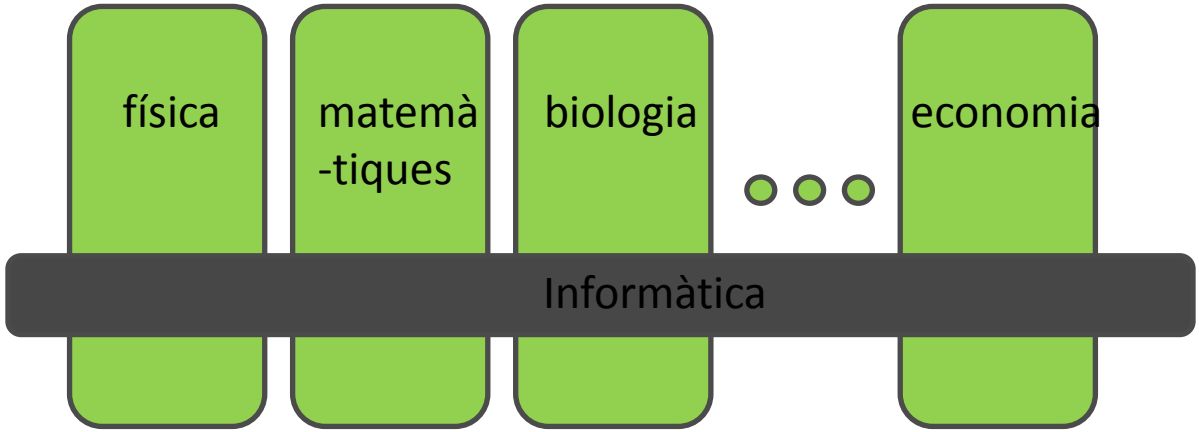
El **Cicle de vida del software o procés de desenvolupament de software** és aquell en el que les necessitats de l'usuari són traduïdes en requisits de software, aquestos transformats en disseny i el disseny implementat en codi.

COMPLEXITAT INTRÍNSECA DE LES APLICACIONS

Complexitat intrínseca de les aplicacions

1. El domini del problema.

- La informàtica és una disciplina transversal
- Tracta problemes de caràcter interdisciplinar



- Dificultat en transmetre necessitats i expectatives dels usuaris
- Canvis en els requisits durant el desenvolupament
- Per exemple: requisits d'un sistema de navegació aèria

Complexitat intrínseca de les aplicacions

2. Sistemes grans.
   - Pot haver desenes de desenvolupadors implicats que, a més, poden estar separats geogràficament
   - Problemes de comunicació i coordinació
3. Parts comunes entre aplicacions són molt sovint dificilment reutilitzables.

Complexitat intrínseca de les aplicacions

4. Dificultat per caracteritzar el comportament de sistemes discrets.
   - Nombre molt gran d'estats, per això les proves no són completes.

Descomposició de problemes complexos

- No podem fer que la dificultat desaparegui. però, podem desenvolupar tècniques i eines que ens permetin **tractar-la i gestionar-la**

FACTORS DE QUALITAT

Factors de qualitat externs

- Correcció
- Robustesa
- Extensibilitat
- Reutilització
- Compatibilitat
- Eficiència
- Portabilitat
- Facilitat d’us
- Funcionalitat
- Oportunitat

Factors de qualitat

- Correcció:

  Capacitat dels productes de software per a realitzar amb exactitud les seves tasques tal i com es defineixen en les especificacions.

Factors de qualitat

- Robustesa:

  Capacitat dels sistemes de reaccionar apropiadament davant condicions excepcionals.

Factors de qualitat

- Extensibilitat:

  Facilitat d'adaptar els productes de software als canvis d'especificació.

Factors de qualitat

- Reutilització:

  Capacitat dels elements de software de servir per a la construcció de diferents aplicacions.

Factors de qualitat

- Compatibilitat:

  Facilitat de combinar uns elements de software amb altres.

Factors de qualitat

- Eficiència

  Capacitat d'un sistema software per exigir la menor quantitat possible de recursos hardware, tals com temps del processador, espai ocupat de memòria interna i externa o ample de banda utilitzat en els dispositius de comunicació

Factors de qualitat

- Portabilitat

  Facilitat de transferir els productes software a diferents entorns hardware i software

Factors de qualitat

- Facilitat d’us

  Facilitat amb la qual persones amb diferents formacions i aptituds poden aprendre a utilitzar els productes software i aplicar-los a la resolució de problemes.

  També cobreix la facilitat d'instal·lació, d’operació i de supervisió.

Factors de qualitat

- Funcionalitat

  Conjunt de possibilitats que proporciona un sistema.

Factors de qualitat

- Oportunitat

  Capacitat d'un sistema de software de ser llançat quan els usuaris ho desitgen o abans.

Factors de qualitat

Qualitats claus:

- Correcció
- Robustesa

→ Fiabilitat

- Reutilització
- Extensibilitat

→ Modularitat

Factors de qualitat

- **Reutilització**
  - Beneficis esperats:
    - Oportunitat
    - Fiabilitat
    - Eficiència
    - Consistència
    - Inversió

"S'ha de ser consumidor de reutilització abans de ser productor."

Quins seran els factors interns de qualitat?

MODULARITAT

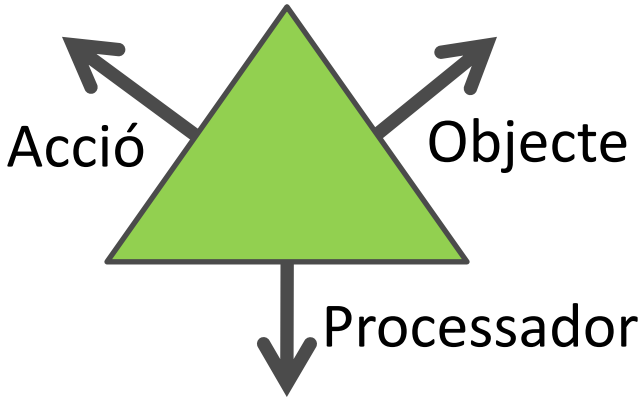
Introducció a la Modularitat

- Objectius principals a l'hora de fer disseny del software:
  1. Fiabilitat
  2. Extensibilitat
  3. Reutilització

- Requereixen mètodes sistemàtics de descomposició dels sistemes en mòduls

Modularitat

- Quins criteris s'han d'utilitzar per trobar els mòduls del nostre software?
  - Les 3 forces de la computació:



- Mòduls com:
  - Unitats de descomposició funcional
  - Unitats basades en els principals tipus d'objectes

Modularitat

- Diferència entre
  - **Els enfocaments tradicionals** construeixen cada mòdul sobre alguna **unitat de descomposició funcional**- un cert aspecte de la acció
  - **L'enfocament orientat a objectes** construeix cada mòdul al voltant **d'algun tipus d'objecte.**

Modularitat

- Element clau:

  Continuïtat modular

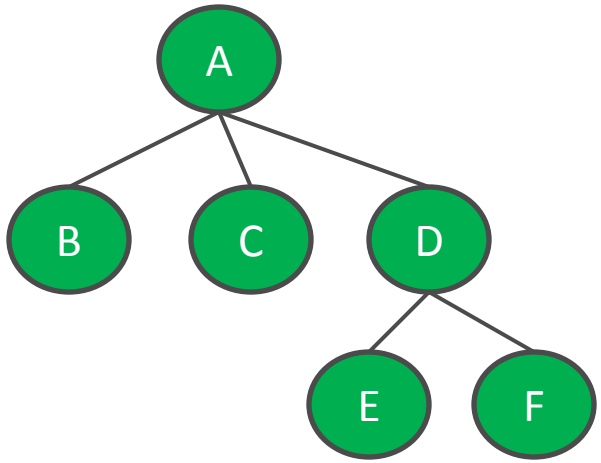
  – Un mètode de disseny satisfà aquest criteri si ens proporciona **arquitectures estables**.

    - mantenen la quantitat de canvi en el diseny proporcional al tamany dels canvis en la especificació.

  – Important, si es considera l'evolució del sistema a llarg termini.

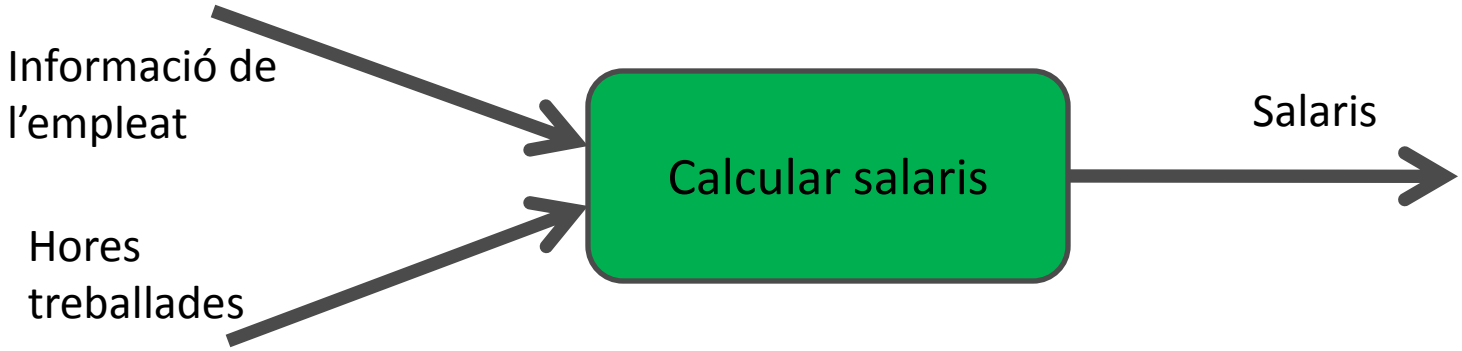
Descomposició funcional

- Disseny descendent:
  - Construeix un sistema per refinament successiu.
  - Pot veure's com el desenvolupament en forma d'un arbre
- Procés:
  - Comença expressant un enunciat de la funció al nivell més alt d'abstracció.
  - Continua amb una seqüència de passos de refinament, reduint el nivell d'abstracció
  - Descompon cada operació en una combinació d'una o més operacions més senzilles



Exemple

- Sistema de nòmines



Pros i Contras de la descomposició funcional

- Avantatges:
  - Disciplina de pensament lògic i ben organitzada
  - Es pot ensenyar amb eficàcia
  - Promou el desenvolupament ordenat de sistemes
  - Ajuda al dissenyador a trobar un camí a través de la complexitat que sovint presenten els sistemes en les seves etapes inicials de disseny.
- Limitacions quan tractem un problema gran:
  - Assumir que el sistema té només una funció principal (el "cim")
  - Base de la descomposició modular: propietats subjectes a canvi.

Funcions i evolució

Perills del disseny descendent

- Es centra en la interfície externa per resoldre la següent pregunta:
  - Què fa el sistema per a l'usuari final?
- Exemple:
  - Programa amb dos versions:
    - Per "lots"
    - Interactiva
- Èmfasis prematur en restriccions d'ordre.
- Exemple (no de software)
  - Ordre de: "trobar una casa", "obtenir un préstec", "signar un contracte"

Descomposició basada en objectes

- Podem trobar una caracterització més estable d'un sistema?
  - Exemple: sistema de nòmines

    → Tipus d'objectes manipulats pel sistema.

    L'esquema orientat a objectes definiria:
    - Tasques de l'empresa
    - Persona
    - Contracte
    - Aplicació

Exemple

- Sistema de nòmines

→ Tipus d'objectes manipulats pel sistema.

L'esquema orientat a objectes definiria:

- Tasques de l'empresa
- Persona
- Contracte
- Aplicació

Construcció del software orientat a objectes

- La construcció del software orientat en objectes és el mètode de desenvolupament de software que basa l'arquitectura de qualsevol sistema software en mòduls deduïts dels tipus abstractes d'objectes que manipula.

No pregunti primer què fa el sistema:
pregunti a què li ho fa!

Procés ascendent

Qüestions

- Cerca dels tipus d'objectes
- Descripció de tipus d'objectes
- Descripció de les relacions i estructura del software

Unitat estables de reutilització

Rutina (unitat de descomposició funcional)

vs.

Tipus abstracte d'objectes

TIPUS ABSTRACTE DE DADES (TADS)

Tipus Abstracte de Dades (TADs)

- Estableixen les bases teòriques del mètode OO
- Criteris per obtenir descripcions apropiades dels objectes
  - Precises i no ambigües
  - Completes
  - No redundants
- Perill de l'especificació excessiva

Tipus Abstracte de Dades (TADs)

- Definició:

  “Un **TAD** és una ens tancat i autosuficient, que no requereix d’un context específic per a poder ser utilitzat en un programa, la qual cosa garanteix portabilitat i reutilització del software i minimitza els efectes que puguin produir un canvi al interior del TAD”

  →**Encapsulació de dades**: ocultació de la representació del tipus de dades respecte de las aplicacions.

Referències del bloc 1

- Bertrand Meyer, "**Construcción de software orientado a objetos", Prentice Hall, 1998.**
  - Modularitat i TADs, capítols 5 i 6 del llibre.
- Grady Booch "**Software Architecture and UML**" (Rational Software). Presentació P. Letelier.
- Estructura de dades: http://sanchezcrespo.org/Docencia/ED/Estructura_de_Datos_01.ppt

Lectures recomanades

- Capítol 1 del llibre de Bertrand Meyer, "**Construcción de software orientado a objetos", Prentice Hall, 1998.**

- **"No Silver Bullet Essence and Accidents of Software Engineering"** Computer Magazine by Frederick P. Brooks. (April 1987)

EXEMPLE:

Descomposició funcional descendent i Descomposició orientada a objectes

Exemple extret del capítol 20 del llibre: "**Construcción de software orientado a objetos**", Bertrand Meyer. Prentice Hall, 1998.

Exemple

- Sistema de reserves per a una companyia Aérea
  - Estats: passos de processament
    - Identificació de l'usuari,
    - Consulta sobre vols,
    - Consulta sobre places,
    - Reserva.

Exemple

- Sistema interactiu multi-panel
  - Patró general:
    - *Estat* (panel amb certes consultes)
    - *Transició* (selecció del següent pas a realitzar)

Exemple: Panel

– Consulta de vols –

Vol des de: Barcelona Destí: París

Sortida prevista: 22 Maig Arribada: 22 Maig

Companyia aérea:

Requisits especials:

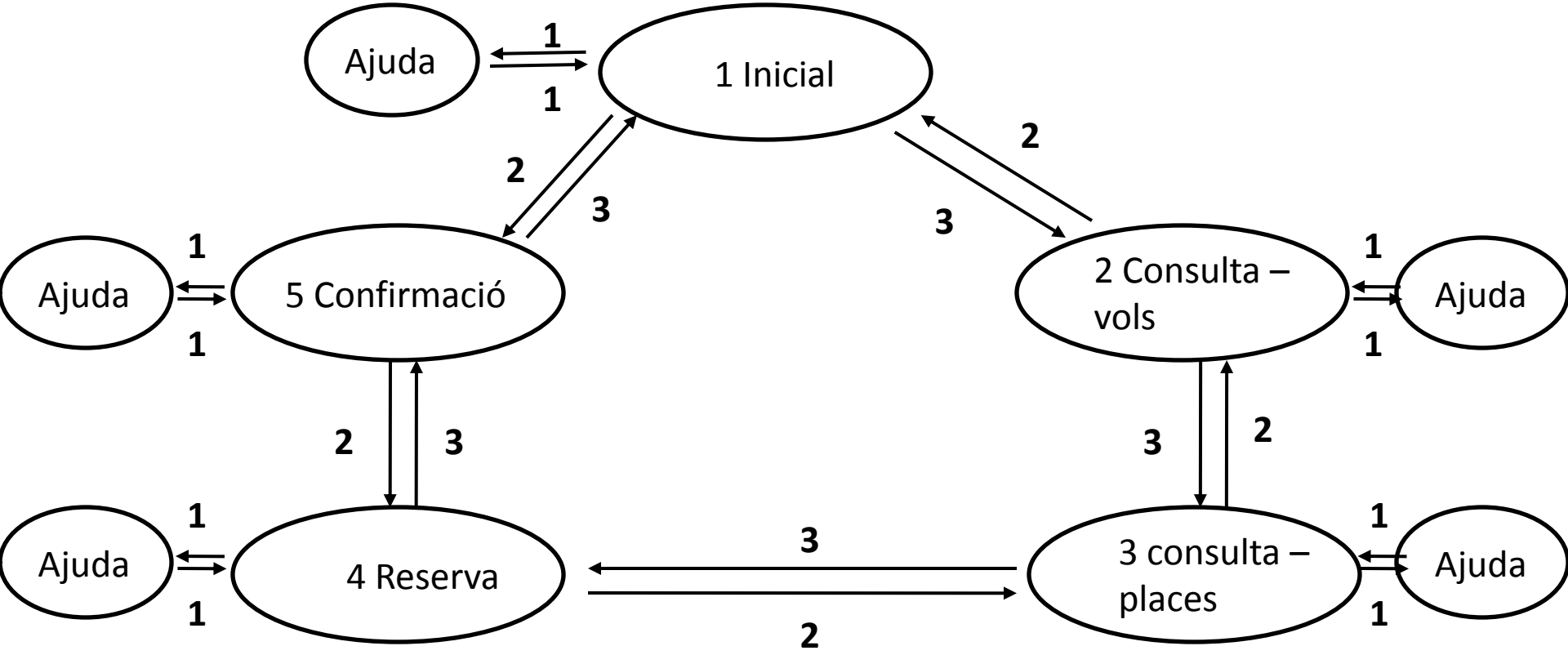
VOLS DISPONIBLES: 1

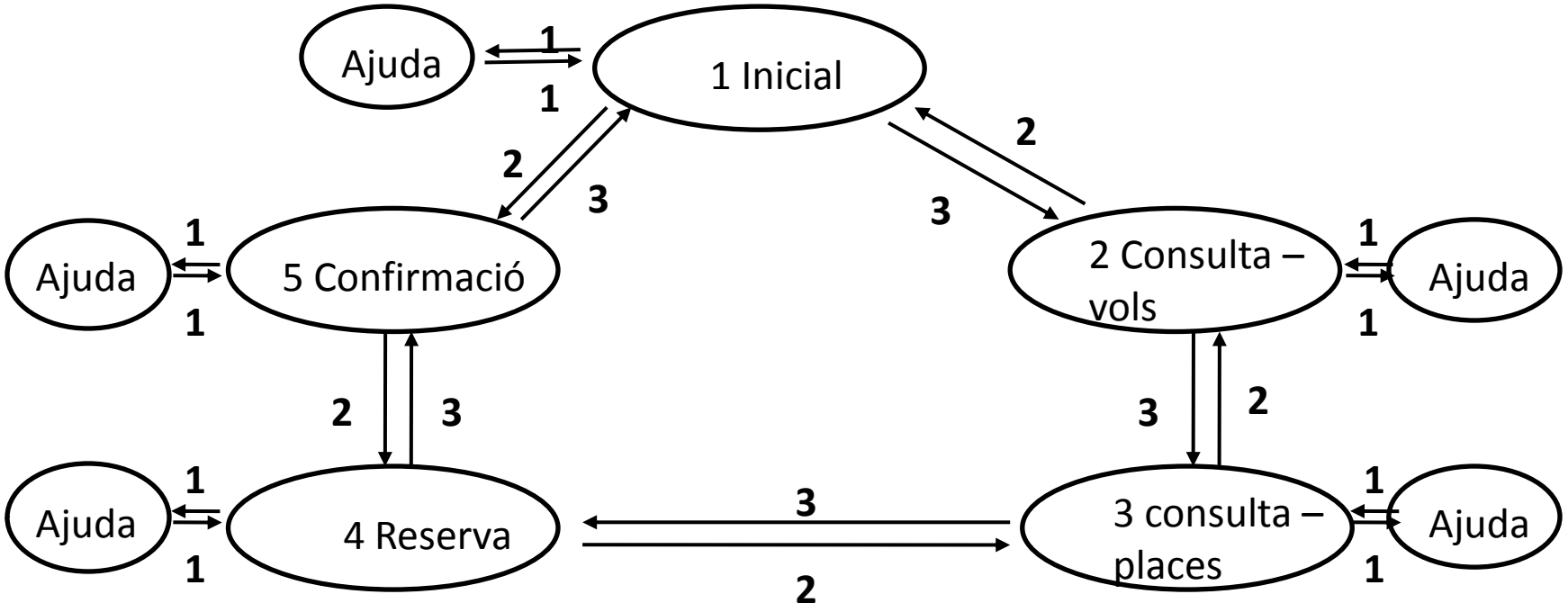
Vuelo: AA 42 Sortida 8:25 Arribada 10:05 Escala: -

Escollir una opció:

0 – **Sortida**
1 – **Ajuda**
2 – **Següent petició**
3 – **Reserva plaça**

Exemple: Un diagrama de transició





Taula de transició:

Estat / Opció	0	1	2	3
1 (Inicial)	-1	0	5	2
2 (Vols)		0	1	3
3 (Places)		0	2	4
4 (Reserves)		0	3	5
5 (Confirm.)		0	4	1
0 (Ajuda)		Tornar		
-1 (Final)				

Exemple: Esquemes del programa

- Primer intent simple
- Solució funcional descendent
- Solució orientada a objectes

Exemple: Primer intent simple

- $B_{consulta}$

  "Mostra el panel de Consultes de Vols"

  repetir

    "Llegir la resposta de l'usuari i escollir C com a següent pas"

    si "Hi ha un error a la resposta" llavors "Mostrar el misstage d'error apropiat" **final**

  fins no "error a la resposta" **final**

  "Processar la resposta"

  cas C **en**

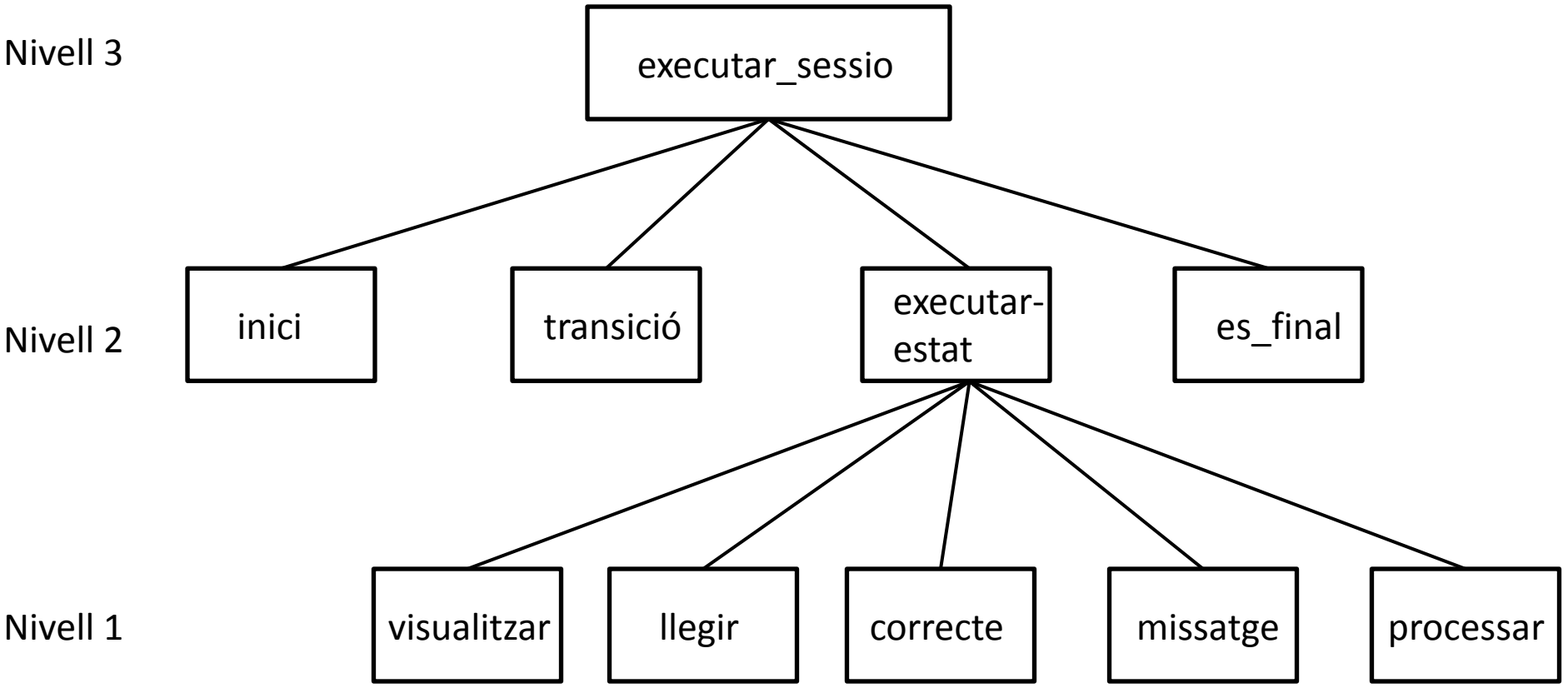
    C_0 : **goto** BConsulta

    C_1 : **goto** B_{Ajuda}

    C_2 : **goto** $B_{Reserva}$

Es repeteix aquest bloc per a cada estat.

Exemple:
Descomposició funcional descendent



Exemple: Rutines

```
executar_sessio es
   -- Executa una sessió completa del sistema
   interactiu

   local
       estat, seguent: INTEGER
   fer
       estat := inici
       repetir
             executar_estat(estat, → seguent)
             -- La rutina executar_estat actualitza
             el valor de següent, a més d'executar
             les accions associades al estat.
             estat := transicio(estat, seguent)
       fins es_final(estat) final
   final
```

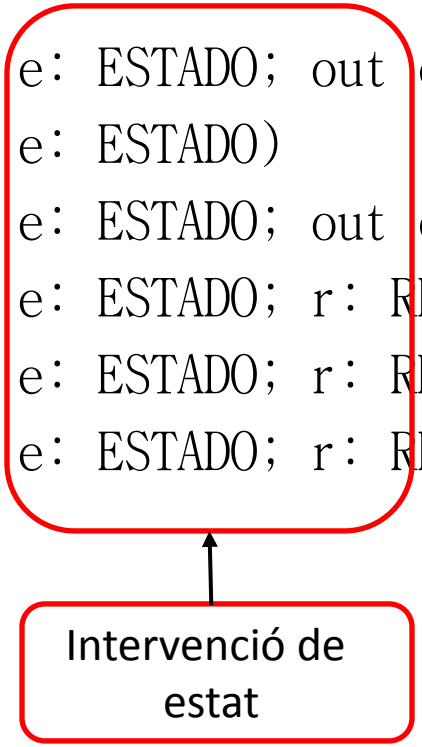
Exemple: Rutines

```
executar_estat (in e: INTEGER; out op: INTEGER) es
   -- Executa les accions associades al estat e,
   tornant en
   -- op l'opció escollida per l'usuari per al següent
   estat.
   local
       r: RESPOSTA; ok: BOOLEAN
   fer
       repetir
             visualitzar(e)
             llegir(e, → r)
             ok := correcte(e, r)
             si no ok llavors missatge(e,r) final
       fins que ok final
       processar(e,r)
       op := seguent_opcio(r)
   final
```

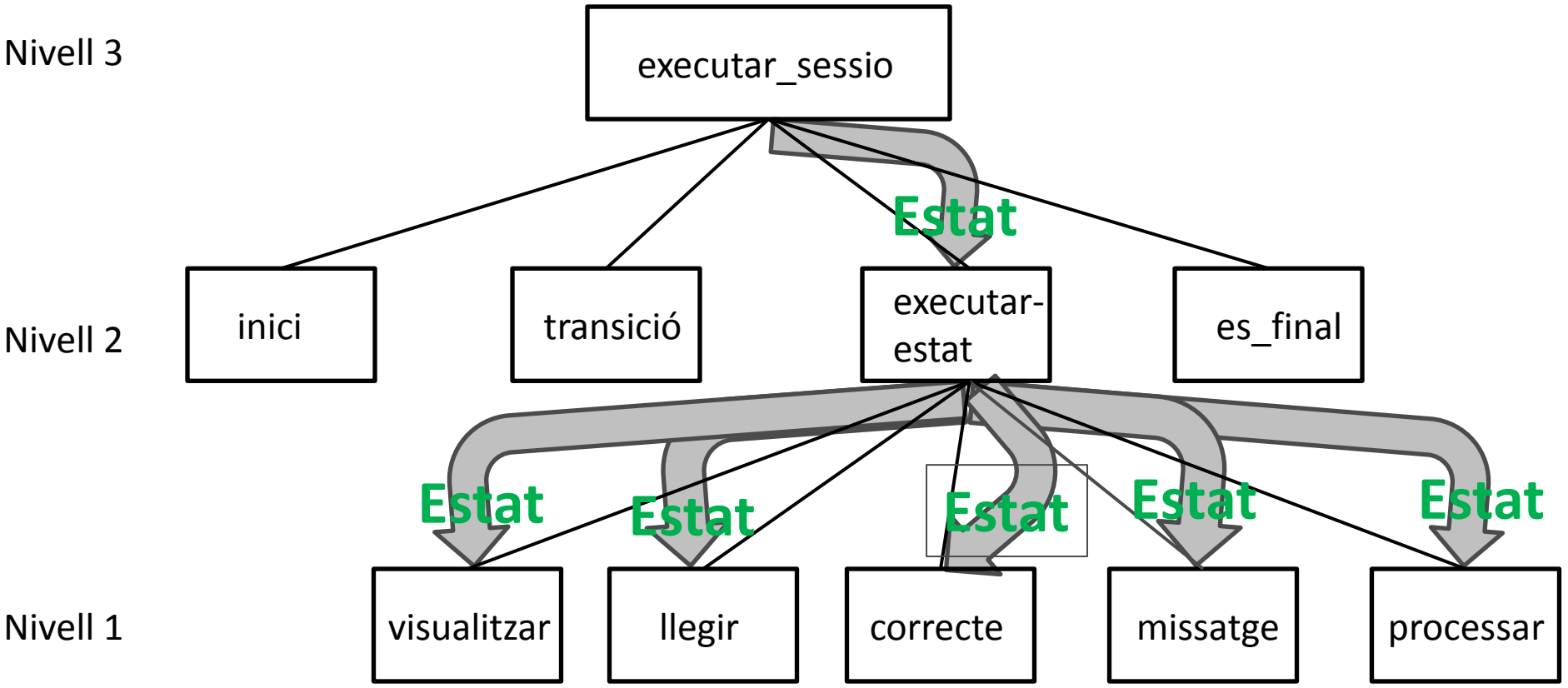
Exemple: Inconvenients de la solució

- Genèric vs. específic
- Considerem les signatures de les rutines:

```
executar_estat(in e: ESTADO; out e: ELECCIÓ)
visualitzar   (in e: ESTADO)
llegir        (in e: ESTADO; out e: RESPOSTA)
correcte      (in e: ESTADO; r: RESPOSTA): BOOLEAN
missatge      (in e: ESTADO; r: RESPOSTA)
proces        (in e: ESTADO; r: RESPOSTA)
```



Exemple: El flux de dades



Exemple: Rutines

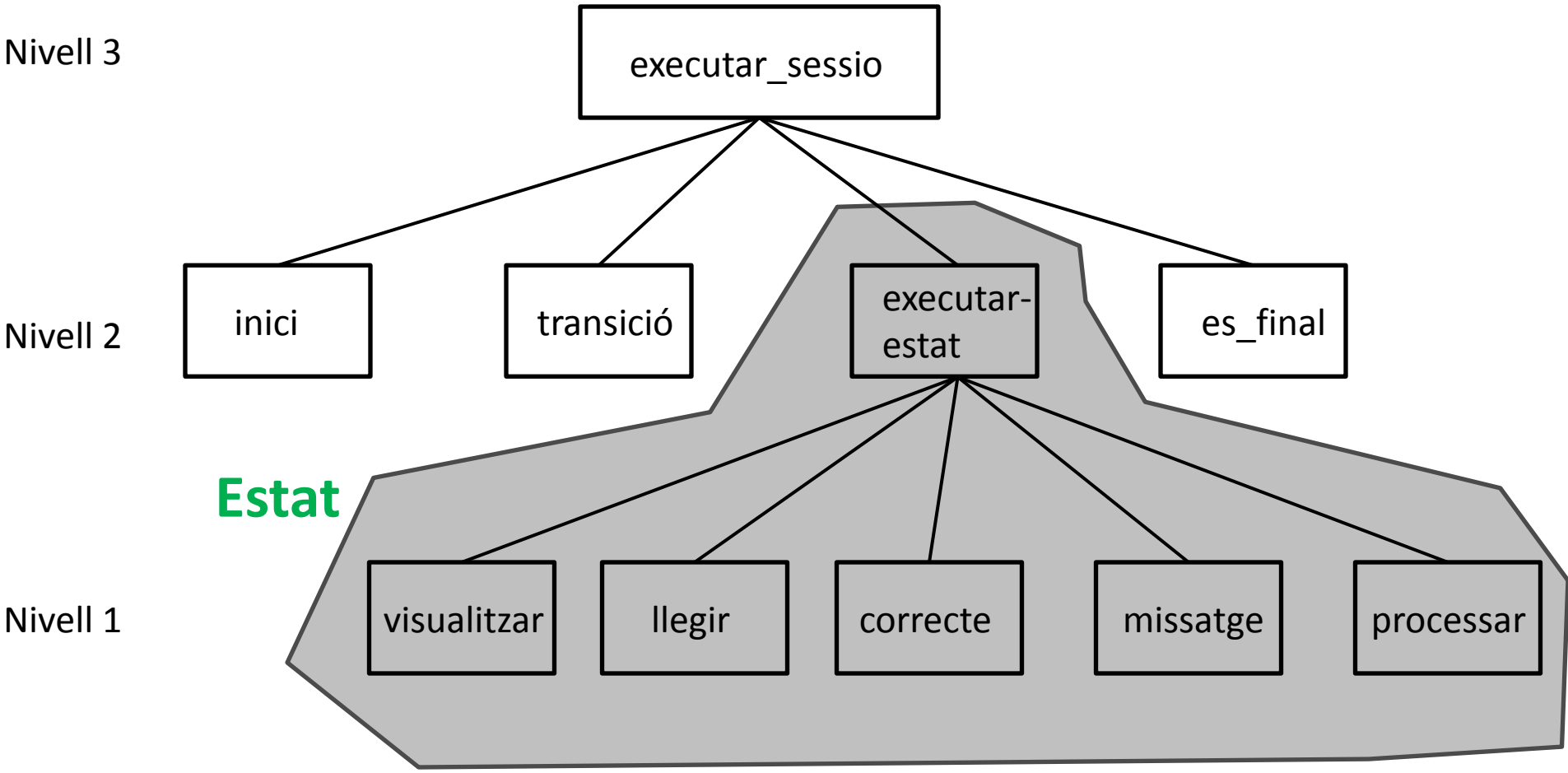
```
inspeccionar
        e
quan Inicial llavors
        ...
quan Consulta_sobre_vols llavors
        ...
...
final
```

Exemple:
Arquitectura orientada a objectes

Llei d'inversió:

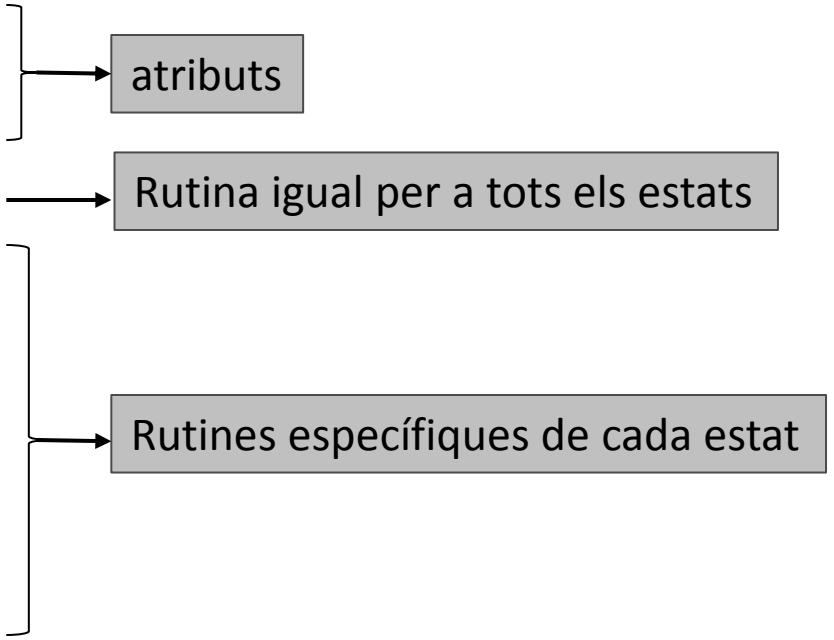
"Si les rutines intercanvien massa dades, posar les rutines en les dades."

Exemple:
Característiques d'estat

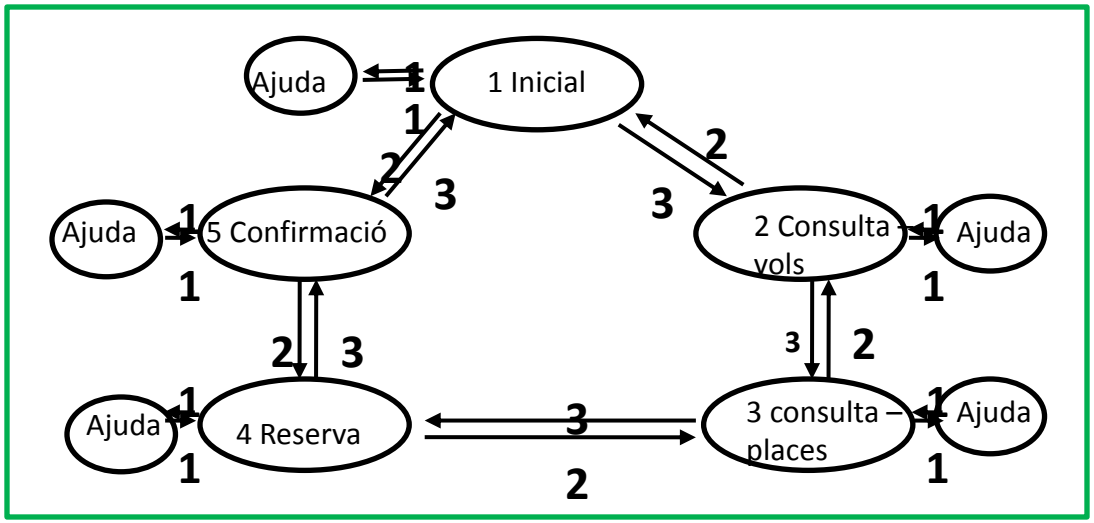


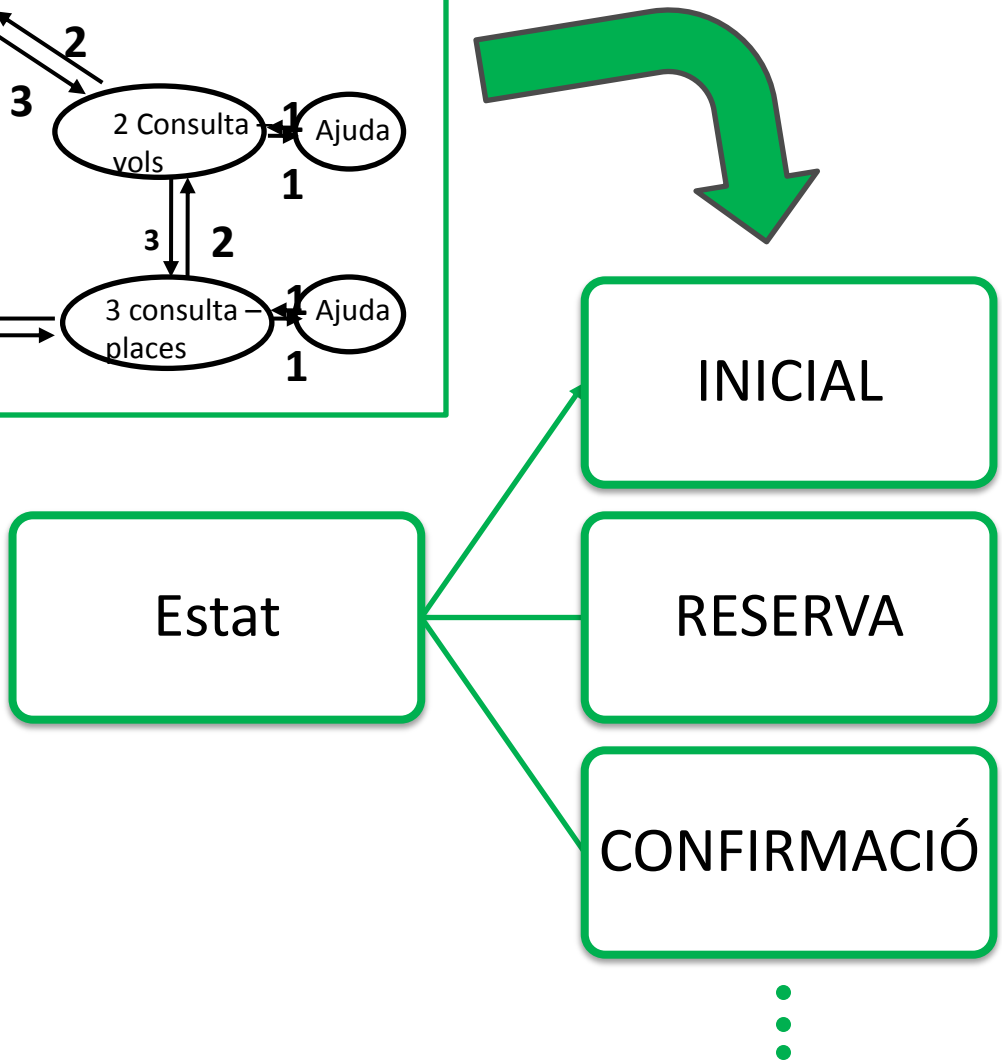
Exemple: Classe

```
classe ESTADO caracteristica
   entrada: RESPOSTA
   opció: INTEGER
   executar es fer ... final
   visualitzar es ...
   llegir es ...
   correcte: BOOLEAN es ...
   missatge es ...
   processar es ...
final
```

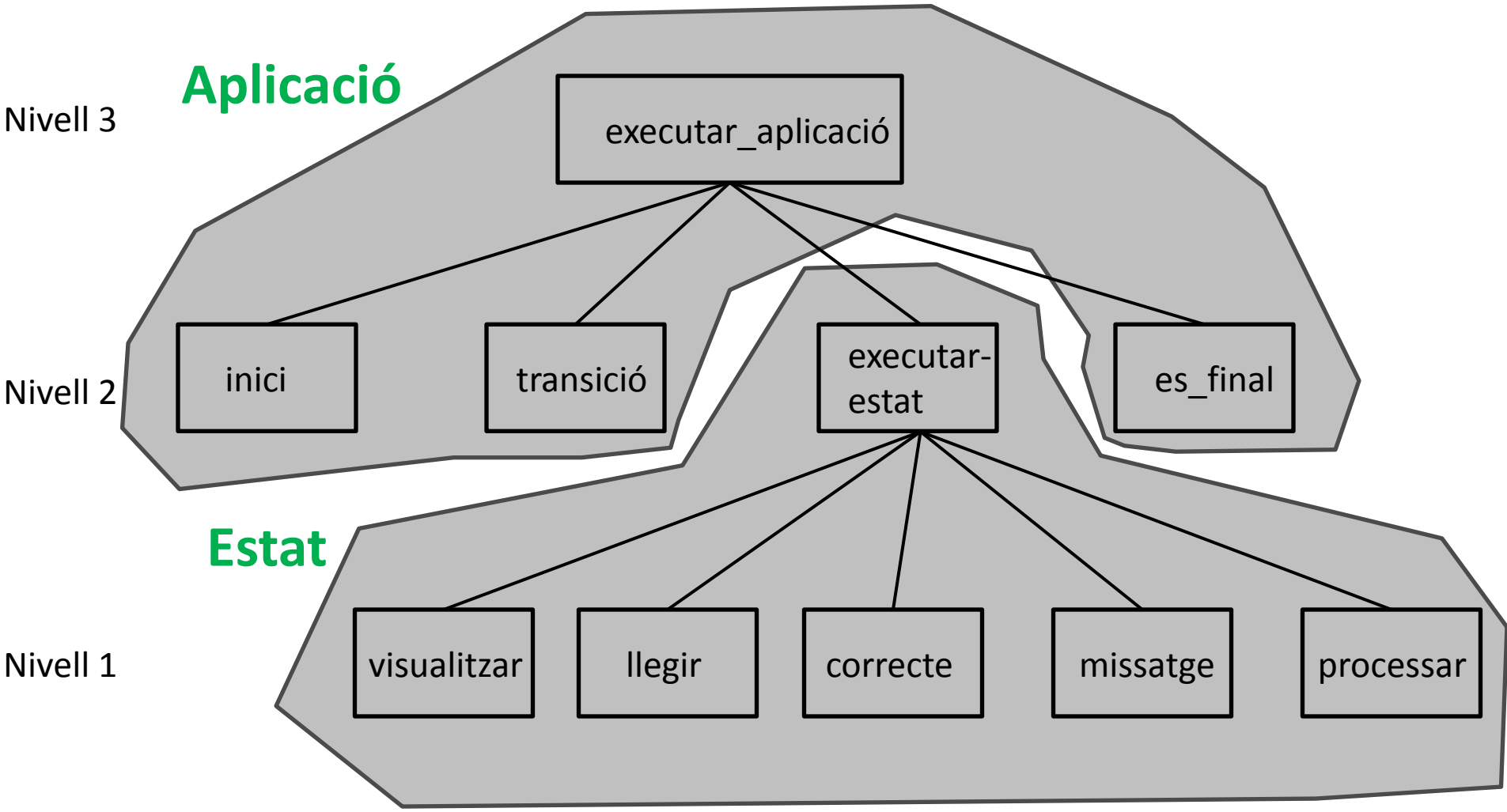


Exemple: Herència





Exemple:
Característiques d'ESTAT i APLICACIÓ



Referències

- Font utilitzada per l'Exemple:

  Capítol 20 del llibre de Bertrand Meyer, "**Construcción de software orientado a objetos", Prentice Hall, 1998.**

Resum

- El disseny és el procés sobre el qual s'assenta la qualitat del software
- Per qualitat s'entén l'adequació del software als requisits exigits
- El disseny modular efectiu redueix la complexitat, facilita els canvis i produeix com a resultat una implementació més senzilla

INTRODUCCIÓ A LA PROGRAMACIÓ ORIENTADA A OBJECTES I A ESDEVENIMENTS

Bloc 2:

Programació Orientada a Objectes

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 mon 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.

CONCEPTES FONAMENTALS: CLASSES I OBJECTES

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).

Objectes

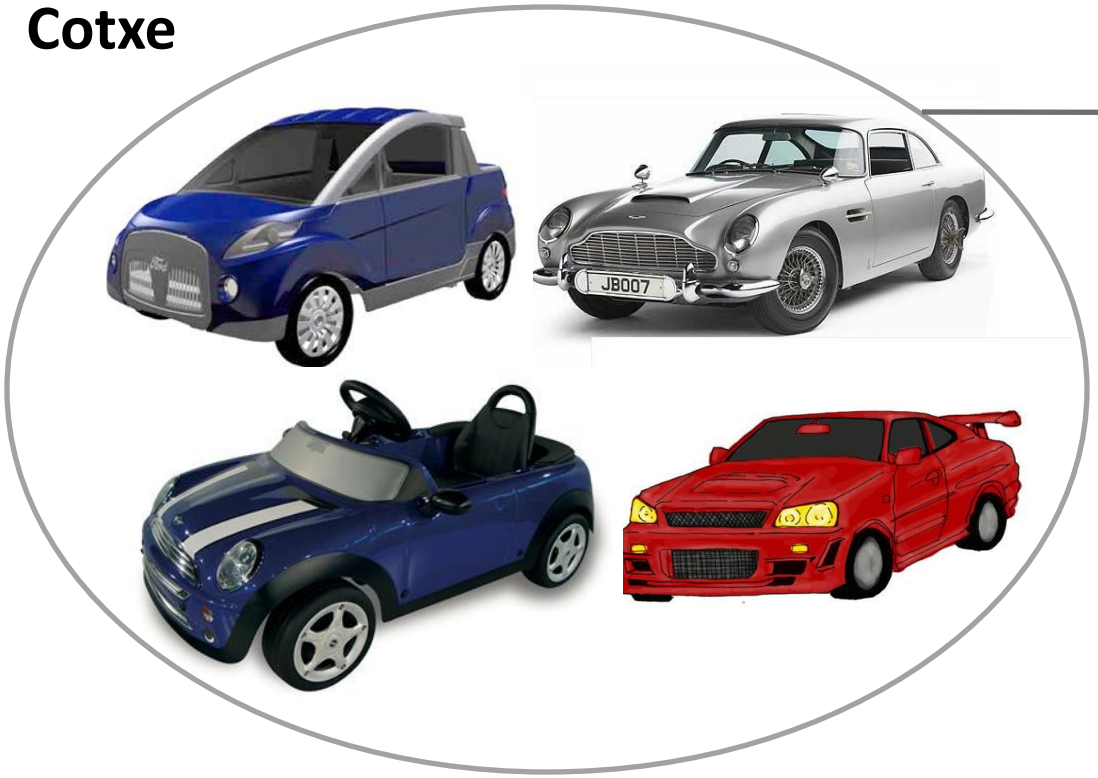
- Els objectes es poden usar per representar entitats del mon real
- Un objecte està format per:
  - Un conjunt de dades (**atributs o estat**)
  - Un conjunt d'operacions (**mètodes o comportament**)
- **El comportament d'un objecte pot modificar el seu estat**
- Cada objecte té un identificador únic pel qual pot ser referenciat.
- L'usuari no gestions 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

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

Cotxe



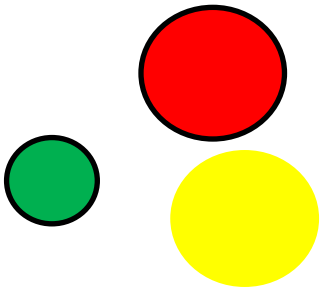
Classe cotxe:
Patró que defineix atributs i mètodes comuns a tots els exemples de **cotxes.**

Classe cotxe
marca model color número portes

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,...

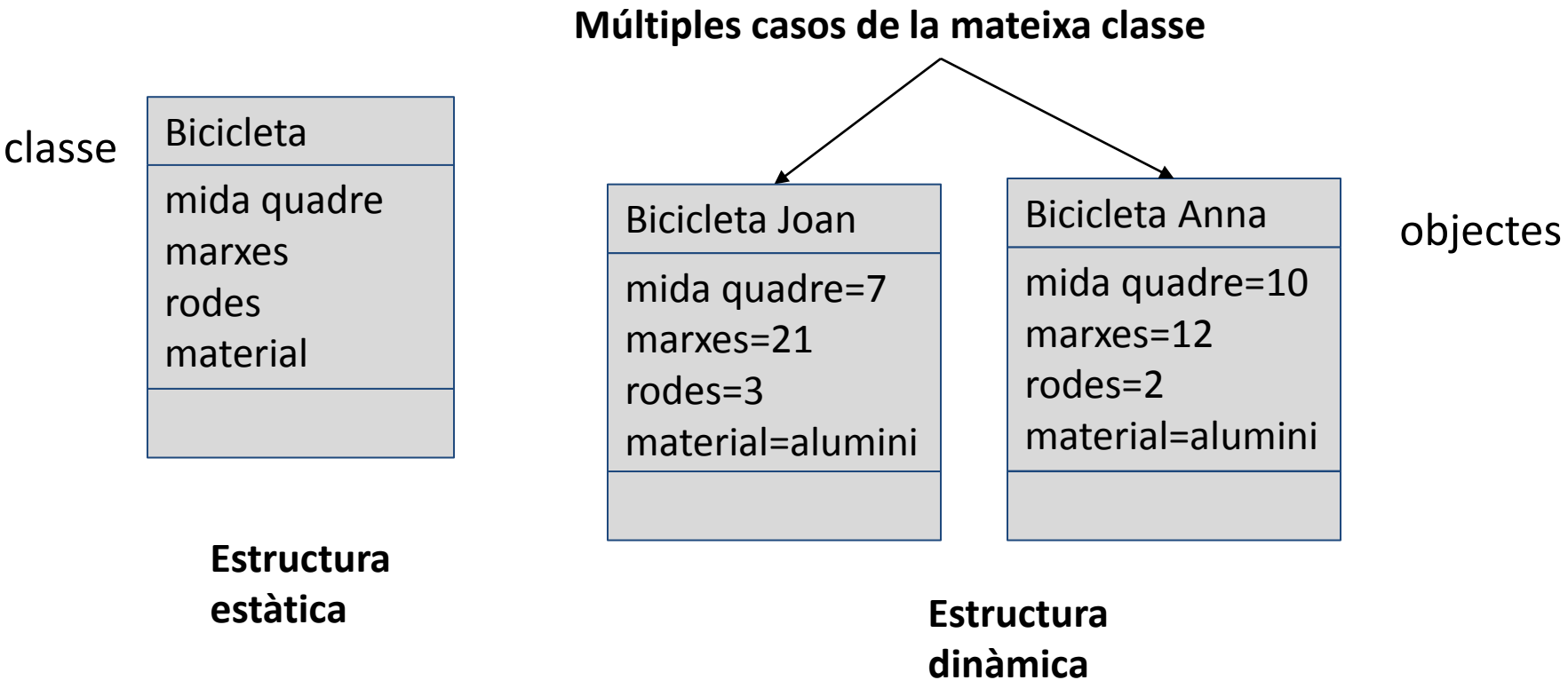
Un objecte representa la materialització d'aquest concepte.

Objectes de bicicletes

Info abstracta →

Bicicleta
tamany rodes material color
moure reparar

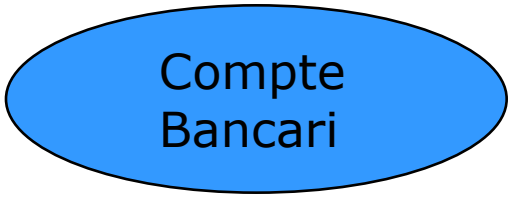
Objectes i classes



Objectes i classes

Una classe
(el concepte)

Un objecte
(la materialització,
la realització)



Compte bancari de Joan
Saldo: 5,257€

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

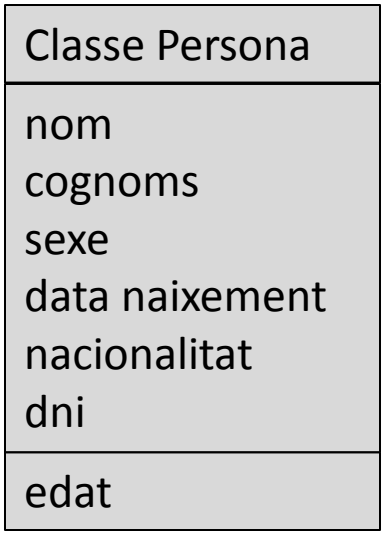
Múltiples objectes
de la mateixa classe

Compte bancari de Maria
Saldo: 16,833€

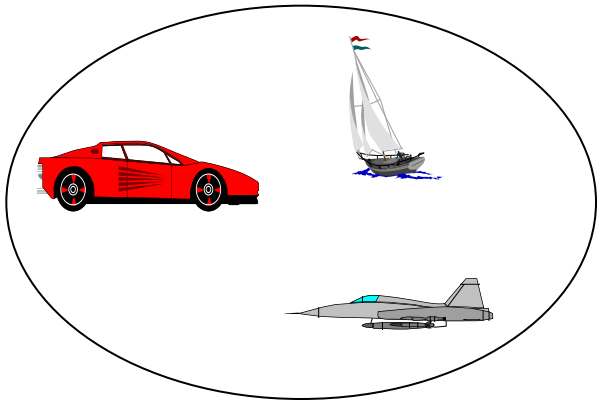
- L'estat d'un compte de banc inclou el seu **Saldo**
- Els comportaments associats amb un compte de banc inclouen la capacitat de fer ingressos i extraccions
- El comportament d'un objecte pot, per tant modificar el seu estat

Exemple

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

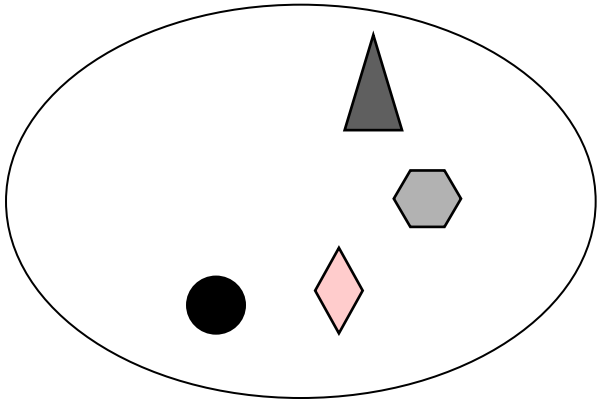


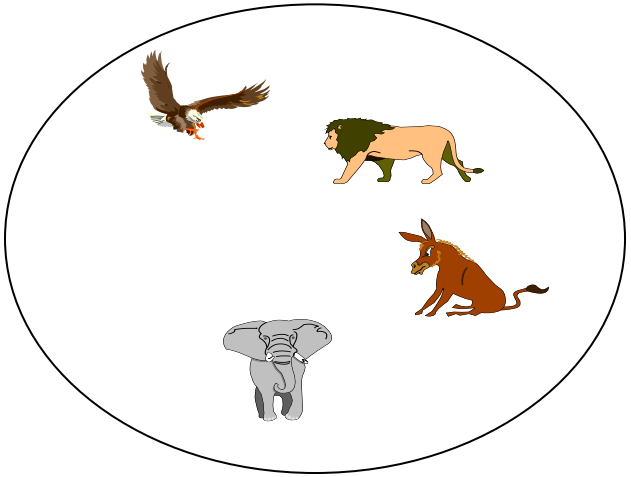
Exemples de classes



Vehicle

Figura





Animal

Classes

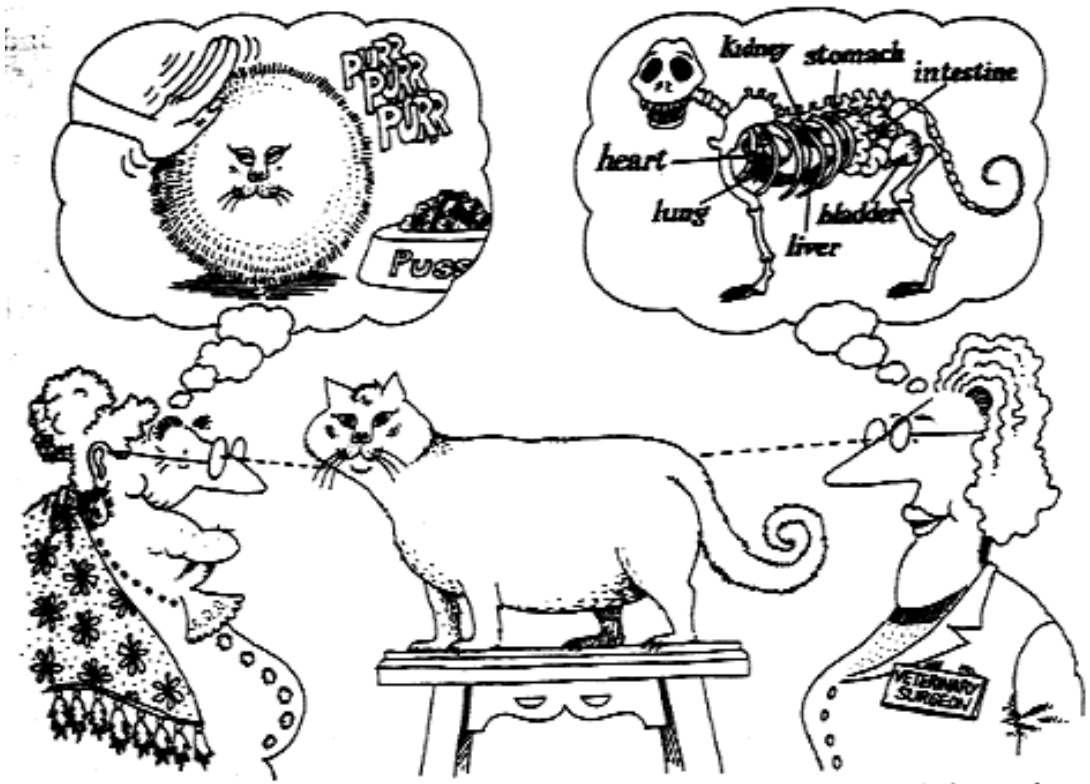
- Una classe és un plànol (o pla d'obra) d'un objecte
  - La classe utilitza els mètodes per definir els comportaments de l'objecte.
  - Es poden crear múltiples objectes d'una mateixa classe.
  - Els objectes comparteixen el nom dels atributs i les operacions, però cada objecte té un valor concret per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

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

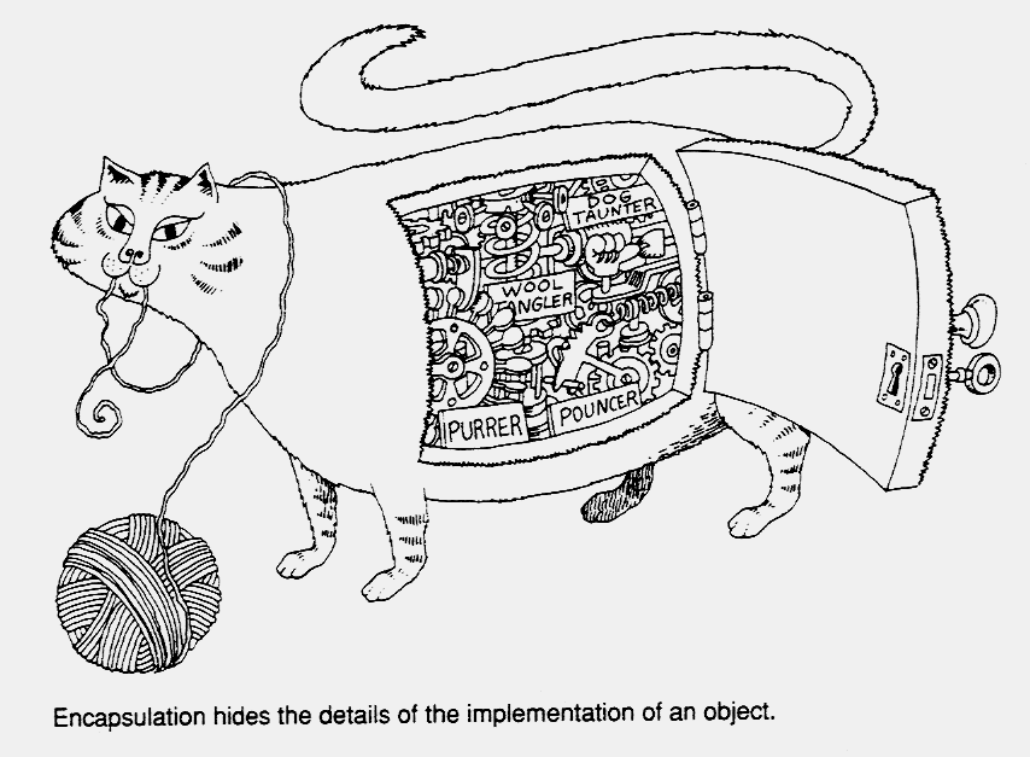


Figura extreta de la pàgina 46 del llibre: "**Object-Oriented Analysis and Design with Applications**". Grady Booch. The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.

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

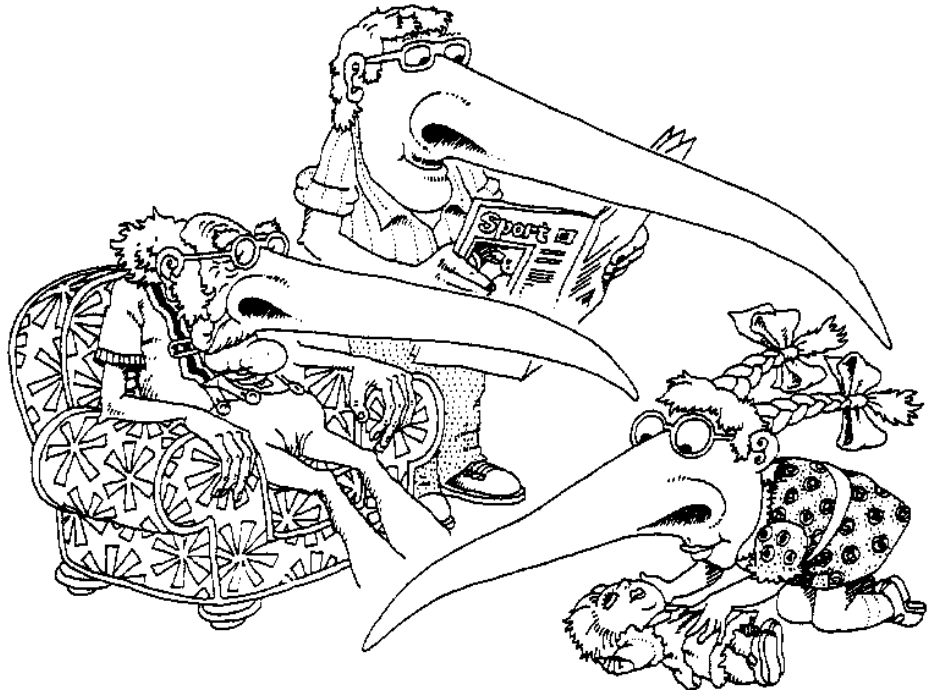
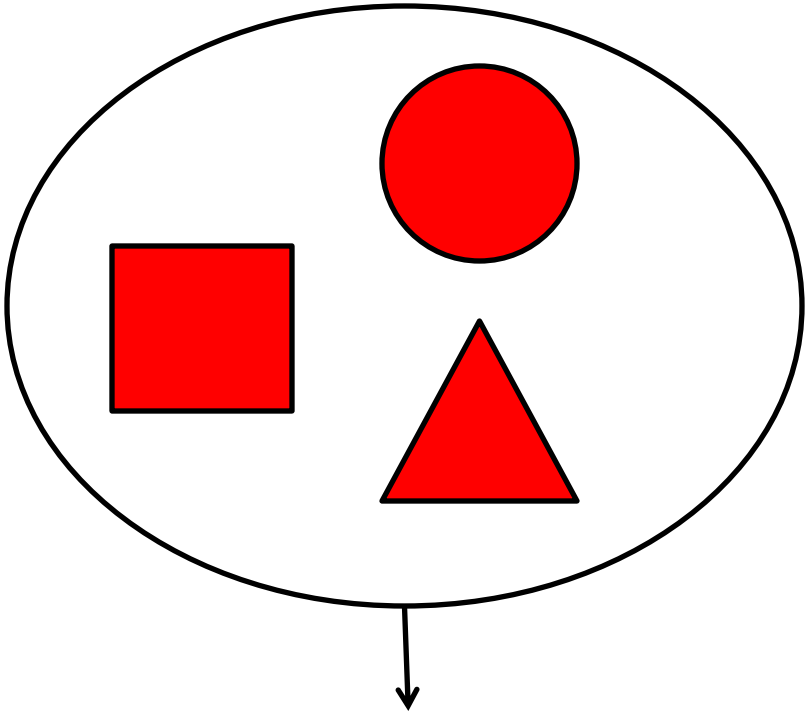


Figura extreta de la pàgina 109 del llibre: "**Object-Oriented Analysis and Design with Applications**". Grady Booch. The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.

Exemple

- Herència → Jerarquia de classes
- Classe 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

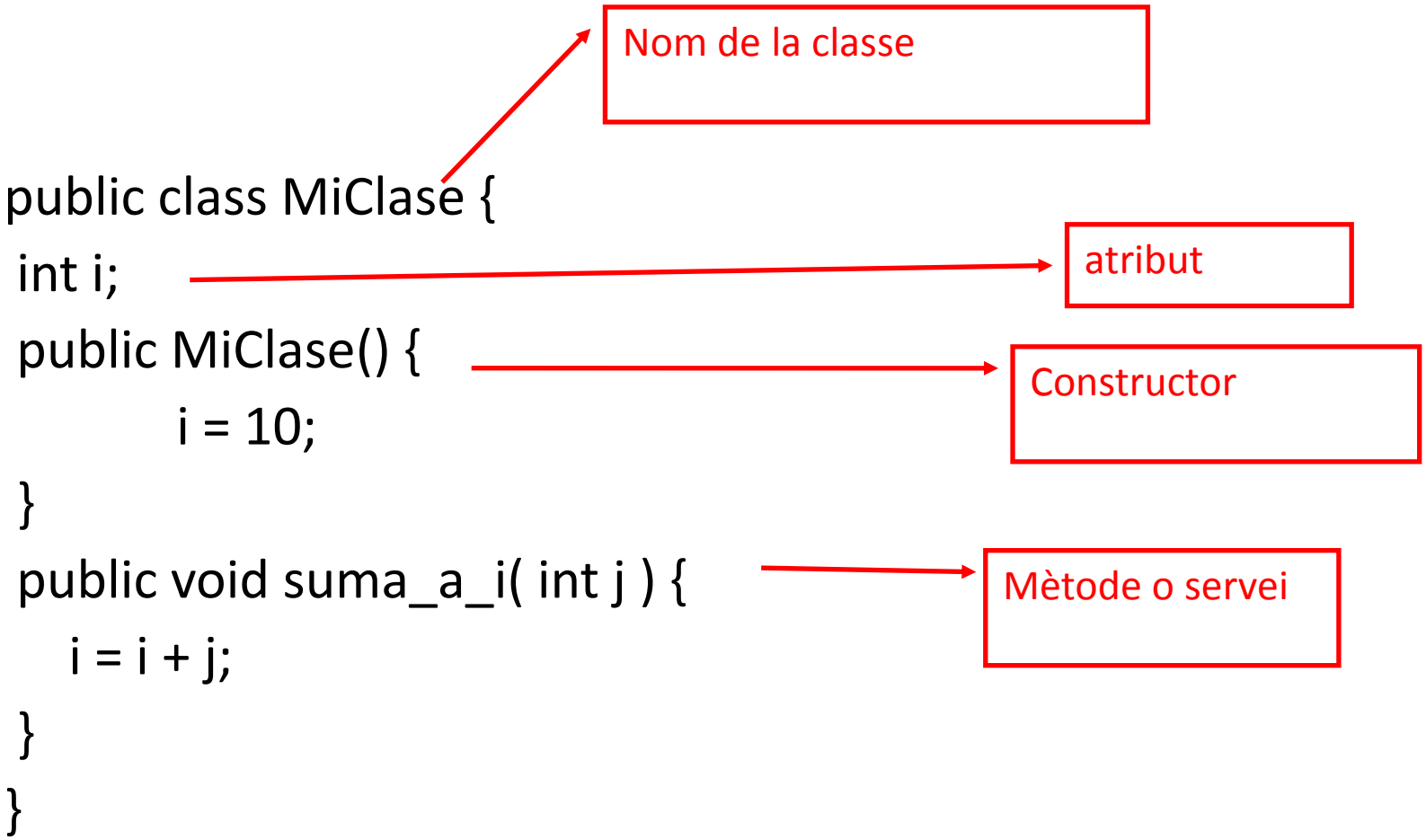
Quadrat
color posició a pantalla àrea perímetre dimensió costats

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

Exemple

Implementar el joc d'endevinar el número que la màquina a pensat

Tres classes:

- **GuessGame** – joc d'endevinar
- **Player** – jugador
- **GameLauncher** – llançador del joc

```
public class GuessGame {
Player p1;
Player p2;
Player p3;
public void startGame() {
        p1 = new Player();
        p2 = new Player();
        p3 = new Player();
        int guessp1 = 0;
        int guessp2 = 0;
        int guessp3 = 0;
        boolean p1isRight = false;
        boolean p2isRight = false;
        boolean p3isRight = false;
        int targetNumber = (int) (Math.random() * 10);
        System.out.println("I'm thinking of a number between 0 and
            9...");
        while(true) {
                System.out.println("Number to guess is " +
                    targetNumber);
                p1.guess();
                p2.guess();
                p3.guess();
                guessp1 = p1.number;
                System.out.println("Player one guessed " + guessp1);
                guessp2 = p2.number;
                System.out.println("Player two guessed " + guessp2);
                guessp3 = p3.number;
                System.out.println("Player three guessed " + guessp3);
```

3 variables d'instancia pels 3 jugadors

```
                if (guessp1 == targetNumber) {
                        p1isRight = true;
                }
                if (guessp2 == targetNumber) {
                        p2isRight = true;
                }
                if (guessp3 == targetNumber) {
                        p3isRight = true;
                }
                if (p1isRight || p2isRight || p3isRight) {
                        System.out.println("Player one got it right? " +
                            p1isRight);
                        System.out.println("Player two got it right? " +
                            p2isRight);
                        System.out.println("Player three got it right? " +
                            p3isRight);
                        System.out.println("Game is over.");
                        break; // game over, so break out of the loop
                } else {
                        // we must keep going because nobody got it right!
                        System.out.println("Players will have to try
                        again.");
                        } // end if/else
                    } // end loop while
                } // end method
} // end class
```

Exemple

```
public class Player {
        int number = 0; // where the guess goes
        public void guess() {
                number = (int) (Math.random() * 10);
                System.out.println("I'm guessing "+ number);
        }
}
public class GameLauncher {
        public static void main (String[] args) {
                GuessGame game = new GuessGame();
                game.startGame();
        }
}
```

Exemple

Sortida per pantalla:

```
%java GameLauncher
I'm thinking of a number between 0 and 9...
Number to guess is 7
I'm guessing 1
I'm guessing 9
I'm guessing 9
Player one guessed 1
Player two guessed 9
Player three guessed 9
Players will have to try again.
Number to guess is 7
I'm guessing 3
I'm guessing 0
I'm guessing 9
Player one guessed 3
Player two guessed 0
Player three guessed 9
Players will have to try again.
Number to guess is 7
I'm guessing 7
I'm guessing 5
I'm guessing 0
Player one guessed 7
Player two guessed 5
Player three guessed 0
We have a winner!
Player one got it right? true
Player two got it right? false
Player three got it right? false
Game is over.
```

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

```
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, mentres 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 declarar de forma separada

Ús de classes

Creant Objectes II

- Usarem l'operador new per crear un objecte

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

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

Ús de classes

Creant Objectes III

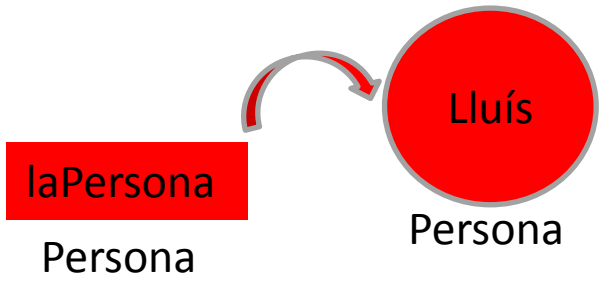
Tres passos: declaració, creació i assignació:

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

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

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

```
laPersona.setNom("Lluís")
```

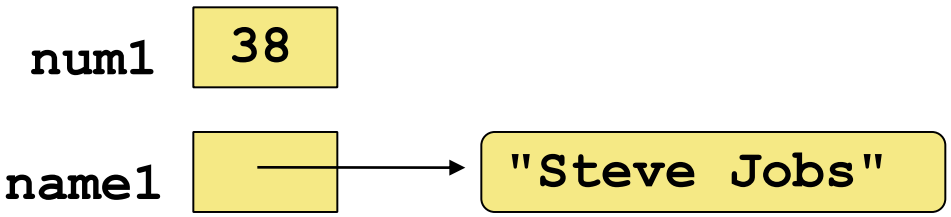


Ús de classes

Referències

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

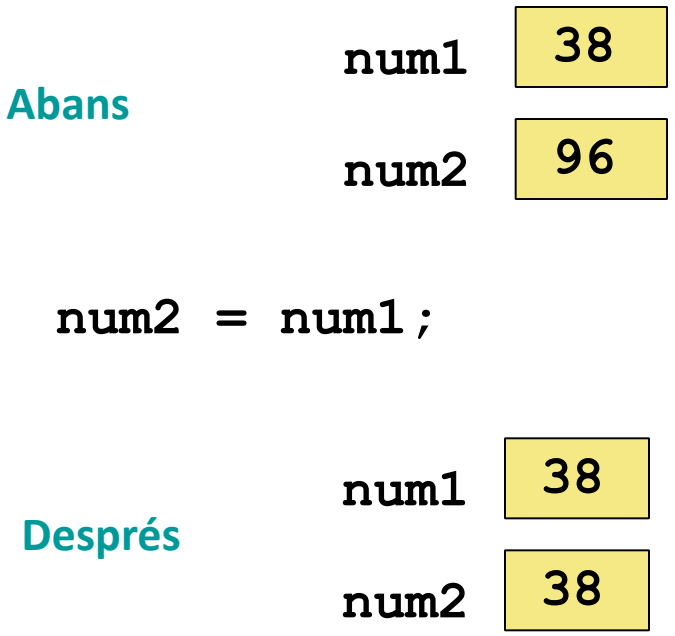
- Exemples :



Ús de classes

Assignació

- De tipus primitius: Exemple:

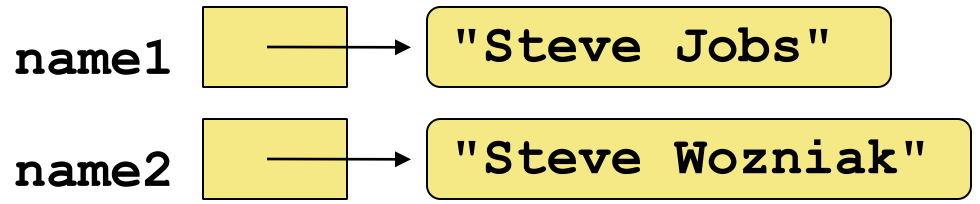


Ús de classes

Assignació de referències

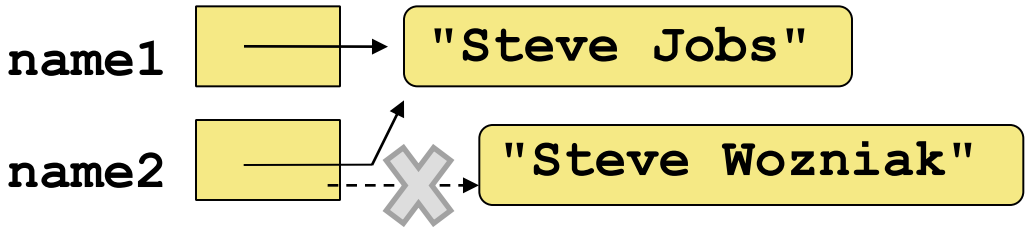
- Per referències a objectes, l'assignació còpia l'adreça

Abans




```
name2 = name1;
```

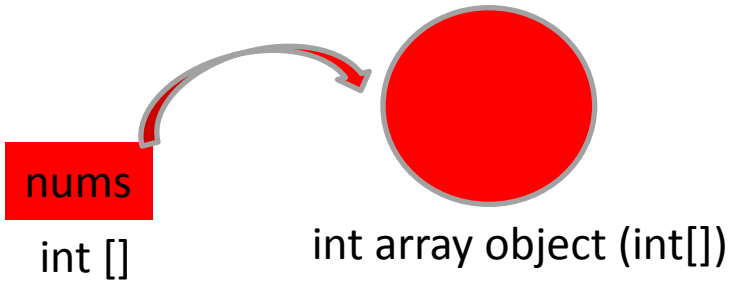
Després



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;
```

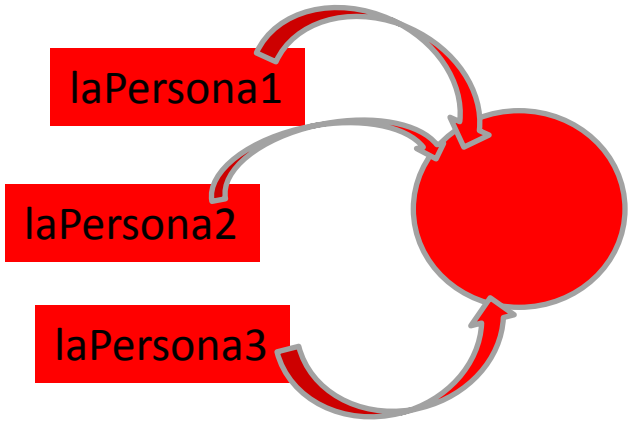


- L'array pot contenir primitives o objectes.

Ús de classes

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: 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

Exercici 1: Assignació de referències

```
public class Aliases {
  public static void main(String[]args) {
        Persona x = new Persona();
        x.edat = 23;
        Persona y = new Persona();
        y.edat = 25;
        Persona z;
              z = y;
              y = x;
              x = z;

        x.edat = 26;
        System.out.println("L'edat en l'objecte referenciat per x és:" + x.edat);
        System.out.println("L'edat en l'objecte referenciat per y és:" + y.edat);
        System.out.println("L'edat en l'objecte referenciat per z és:" + z.edat);
  }
}
```

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

→ Sortida per pantalla?

Exercici 2: Hi ha errors de compilació?

```
class BooksTestDrive {

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

Exemple

```
public class Dog {
   String name;
   public static void main(String[] args) {
      Dog dog1 = new Dog();
      dog1.bark();
      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].bark();
         x = x+1;
      }
   }
   public void bark() {
      System.out.println(name + " says Ruff!");
   }
   public void eat() { }
   public void chaseCat() { }
}
```

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();
   }
}
```

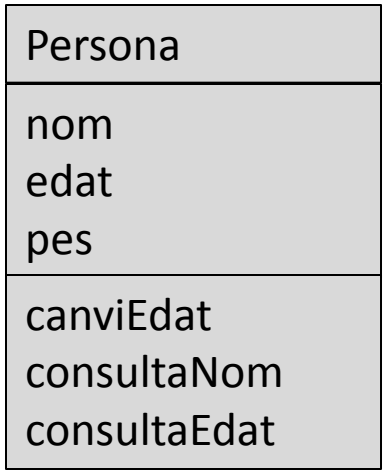
Exercici: col·lecció

```
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();
    }
}
```

CONSTRUCTORS I DESTRUCTORS

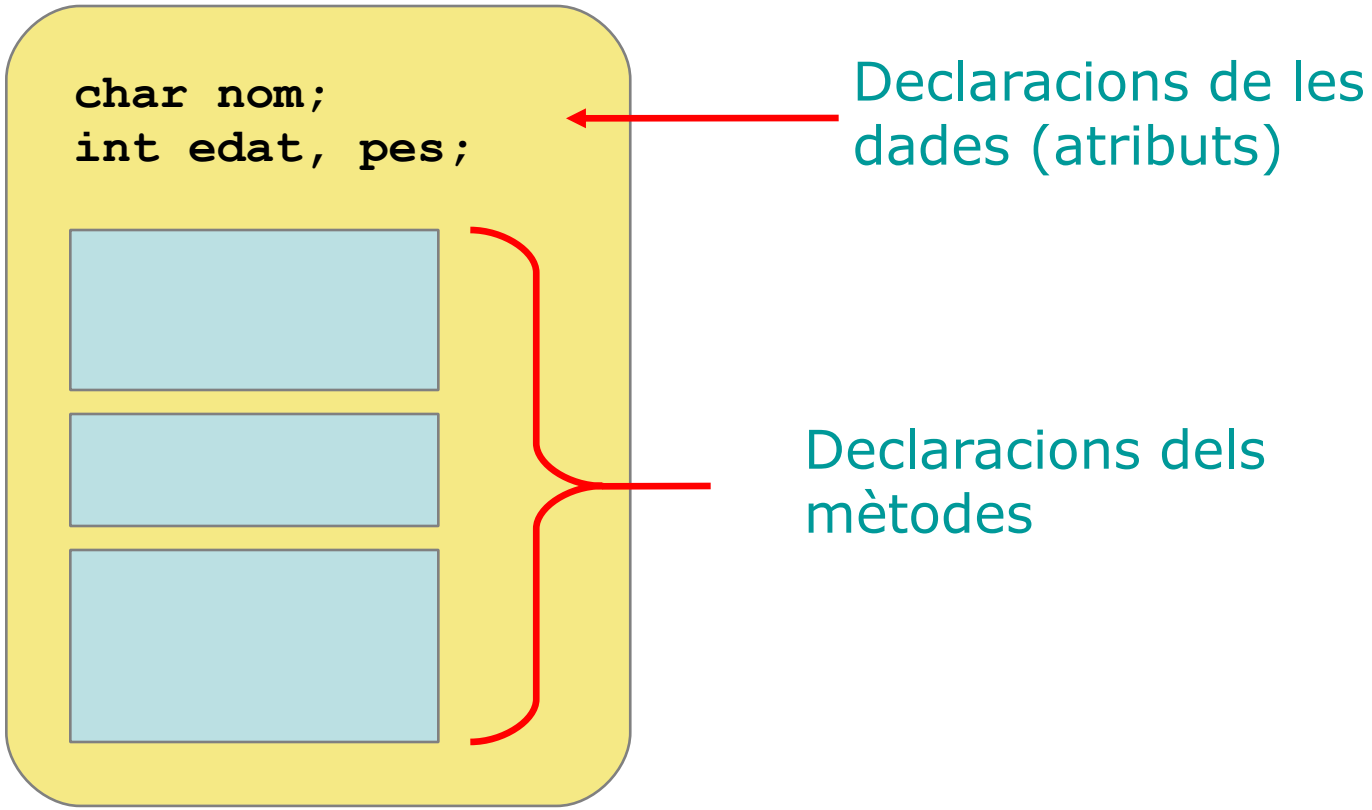
Creació de classes

- Un objecte té **estat** i **comportament**
- Per exemple, la classe **`Persona`**
  - El seu estat està definit per Nom i Edat
  - El seu comportament principal és **`CanviaEdat`, `ConsultaNom i 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");

   }

}
```

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;
  public MiClase() {
      i = 10;
  }
  public void suma_a_i(int j) {
      for(int k=0; k<j; k++){
            i += k;
      }
  }
}
```

Variable d'instànica

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

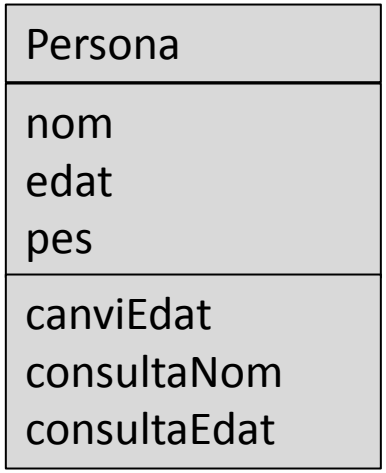
```
public class Dau{

   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 roll() {
        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

- 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

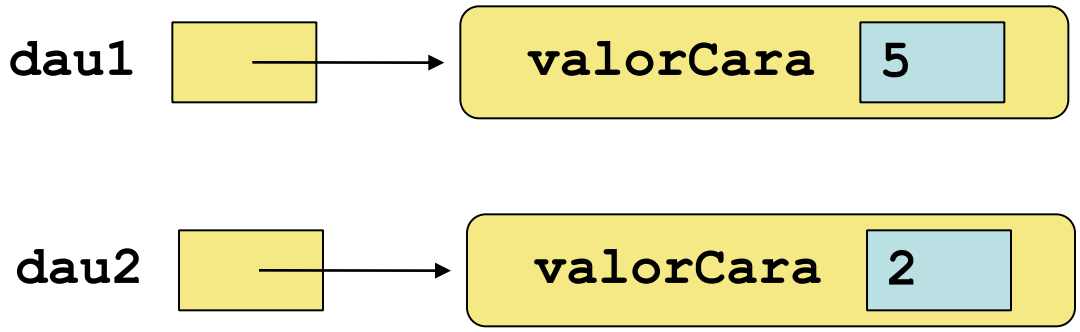
→Cada vegada que es crea un objecte Persona, es crea també una variable Edat i Nom.



- Els objectes comparteixen les definicions de mètodes, però cada objecte té el seu propi espai de dades

Creació de classes.

Dades d'Instància



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

ENCAPSULACIÓ

Encapsulació

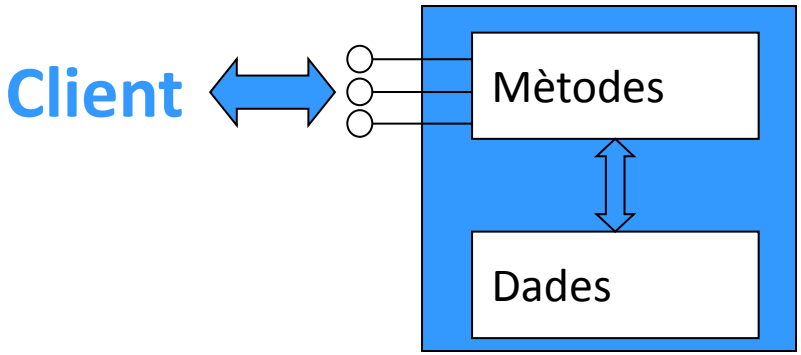
- Hi ha dues visions d'un objecte:
  - **Interna** → els detalls de les variables i dels mètodes que la classe defineix
  - **Externa** → els serveis que un objecte proveeix i com l'objecte interactua amb la resta del sistema

- 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 modificadors de visibilitat:
  - *public*,
  - *protected* i
  - *private*

Encapsulació: Modificadors de visibilitat

- Els membres declarats ***public*** es poden referenciar des de qualsevol lloc
- Els membres declarats ***private*** es poden referenciar només des de la classe (però un objecte pot veure el valor d'un membre *private* d'un altre objecte de la mateixa classe)

ⓘ Els membres declarats sense modificador de visibilitat tenen visibilitat per defecte i poden ser referenciats des de qualsevol classe del mateix package → **friendly**

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 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.

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

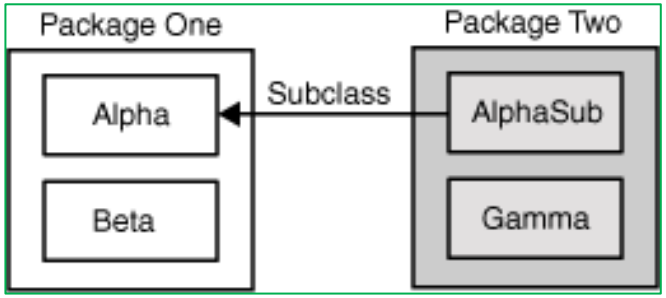
```
package unPaquet;

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

- Visibility:



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.

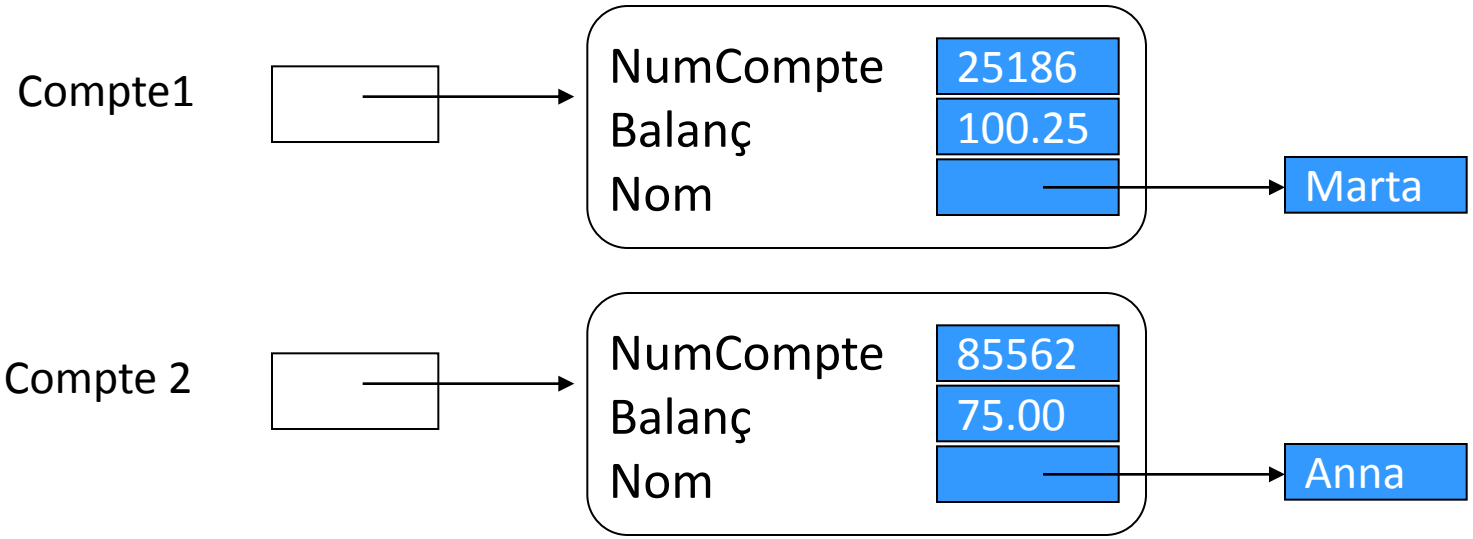
Exemple encapsulació

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



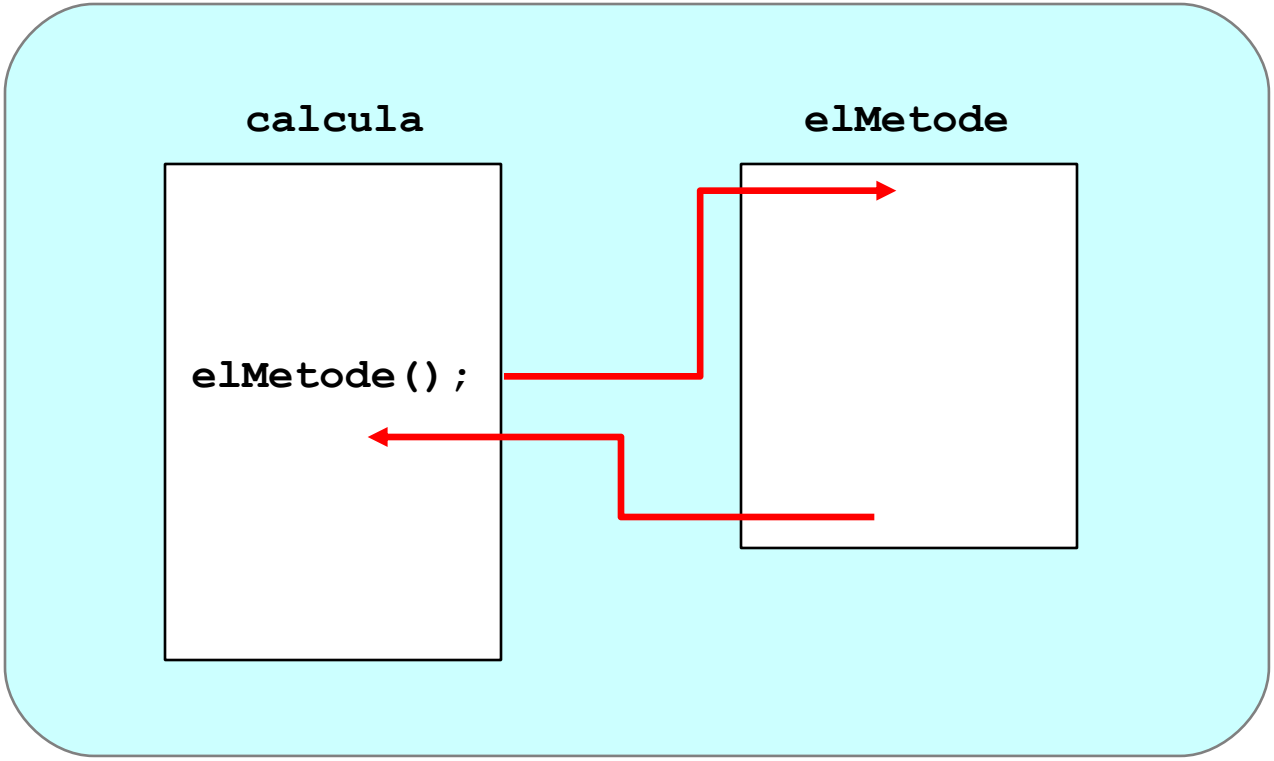
Exercici: Implementeu la classe **`Account`** en JAVA

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

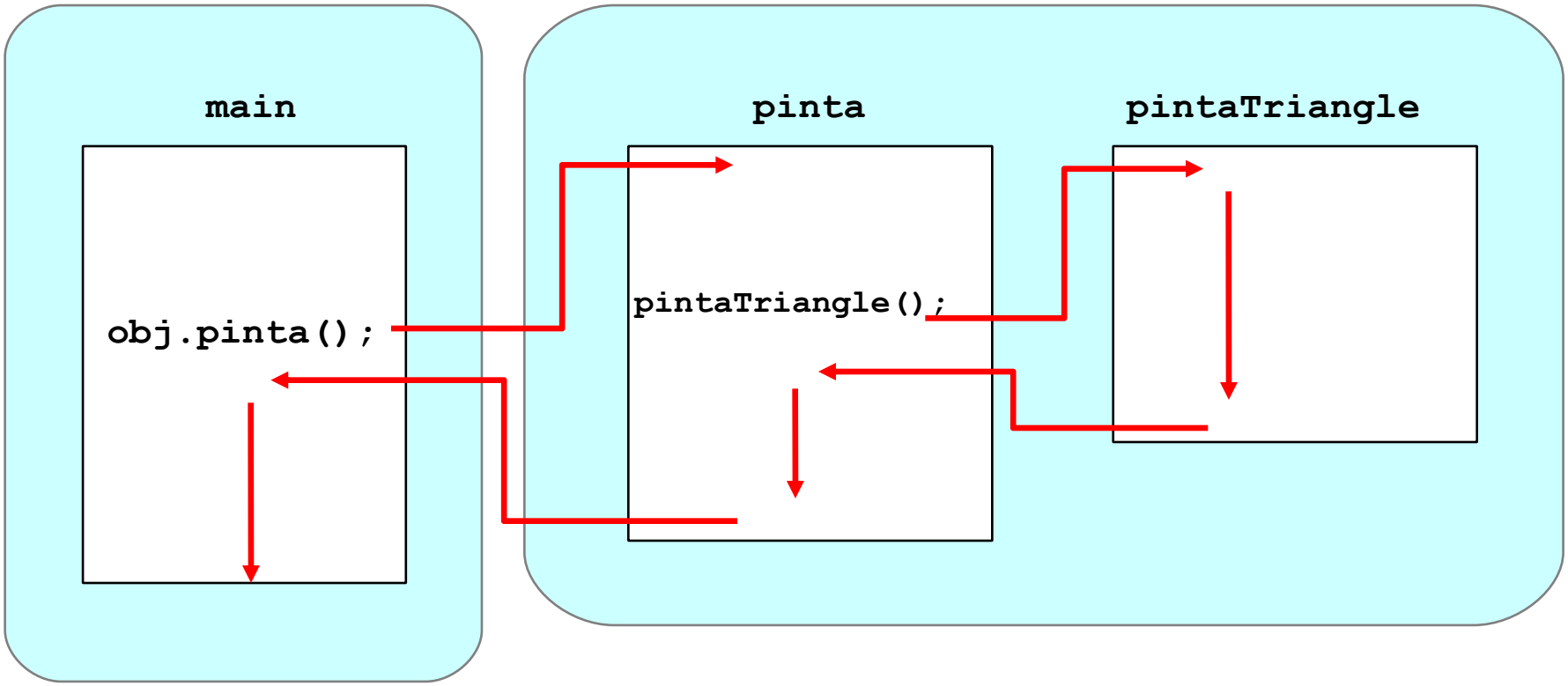
Encapsulació

Flux de control d'una invocació



Encapsulació

Flux de control d'una invocació

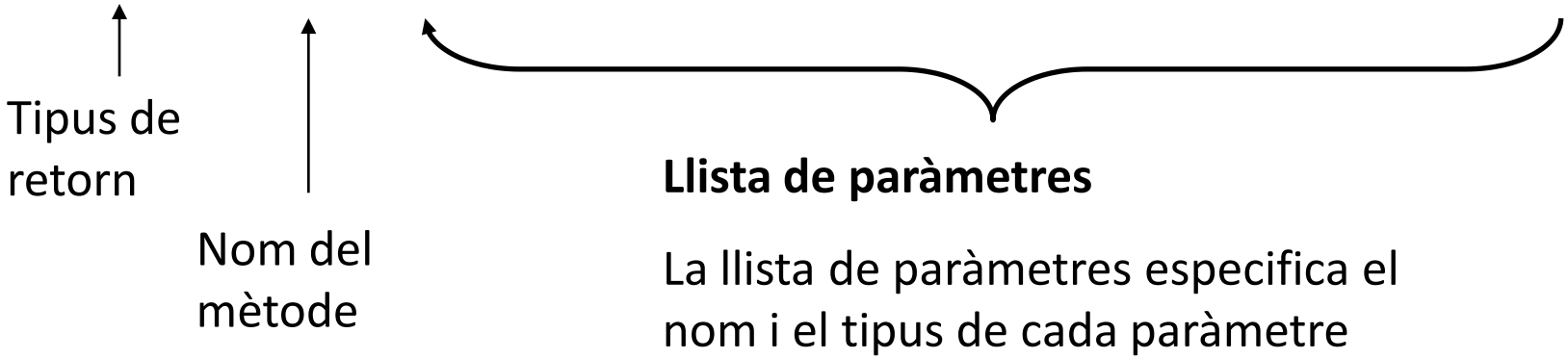


Encapsulació

Capçalera d'un mètode

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

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



Encapsulació

Cos d'un mètode

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


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

`sum` i `result` són dades locals

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

Encapsulació

La instrucció return

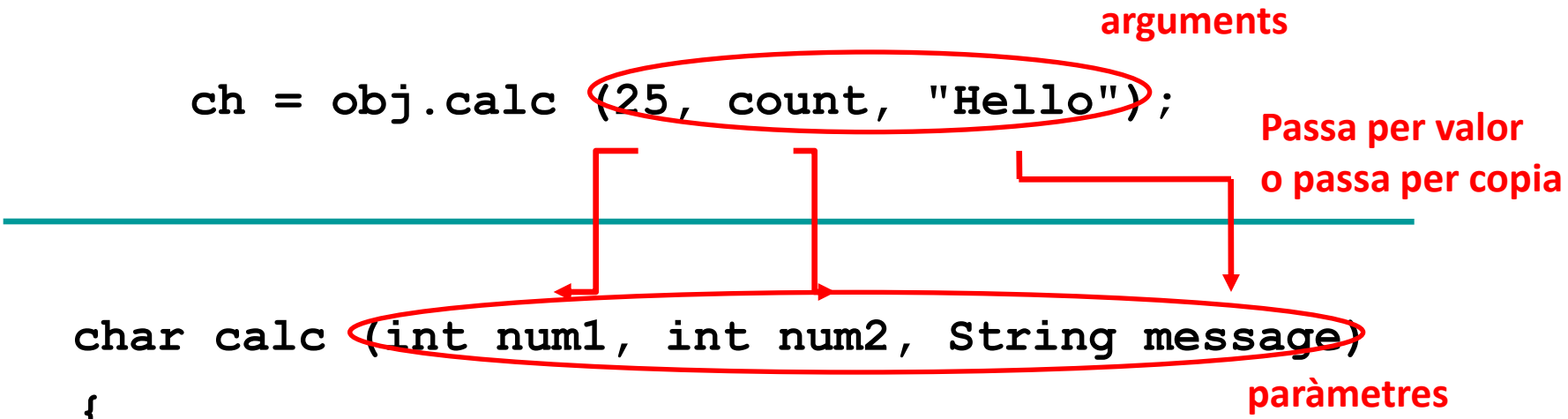
- El tipus de retorn d'un mètode indica el tipus de valor que el mètode retorna al que crida
- Un mètode que no retorna un valor té un tipus de retorn ***void***
- Una instrucció return especifica el valor que es retornarà

```
return expression;
```

Creació de classes. Encapsulació

Paràmetres

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



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 asignes explicitament cap valor o no cridis a un métode setter.
- Exemples:
  - integers 0
  - floating points 0.0
  - booleans false
  - referencies null

Diferència entre variable d'instància i variable local

- 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 reusar 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?

Exemple

```
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;}
}
```

Exemple

```
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:

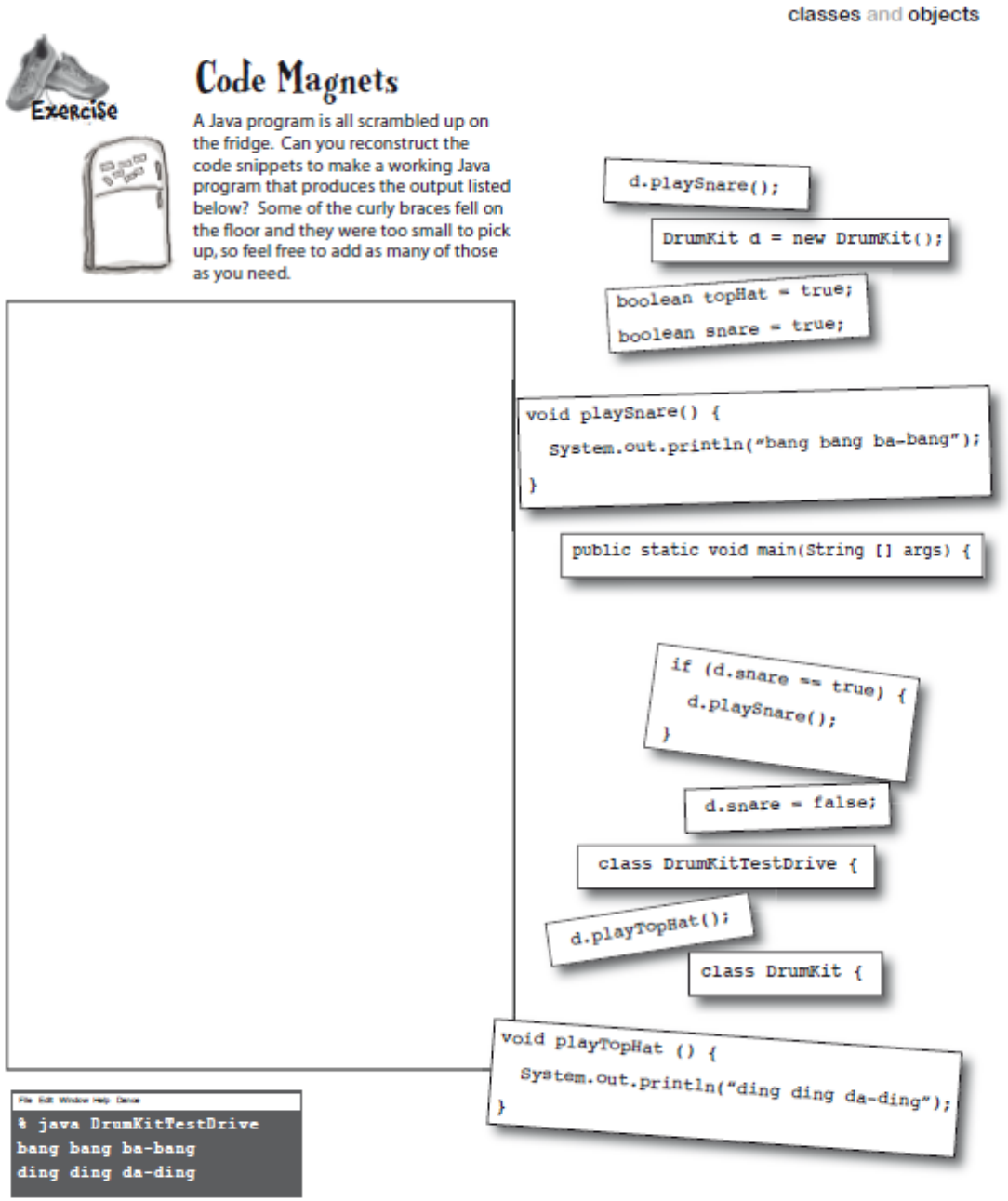


Figura extreta dela pàgina 43 del llibre: **Head First Java**. Bert Bates, Kathy Sierra. O'Reilly Media, 2005.

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

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:

Classe Wrapper	Tipus primitiu
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

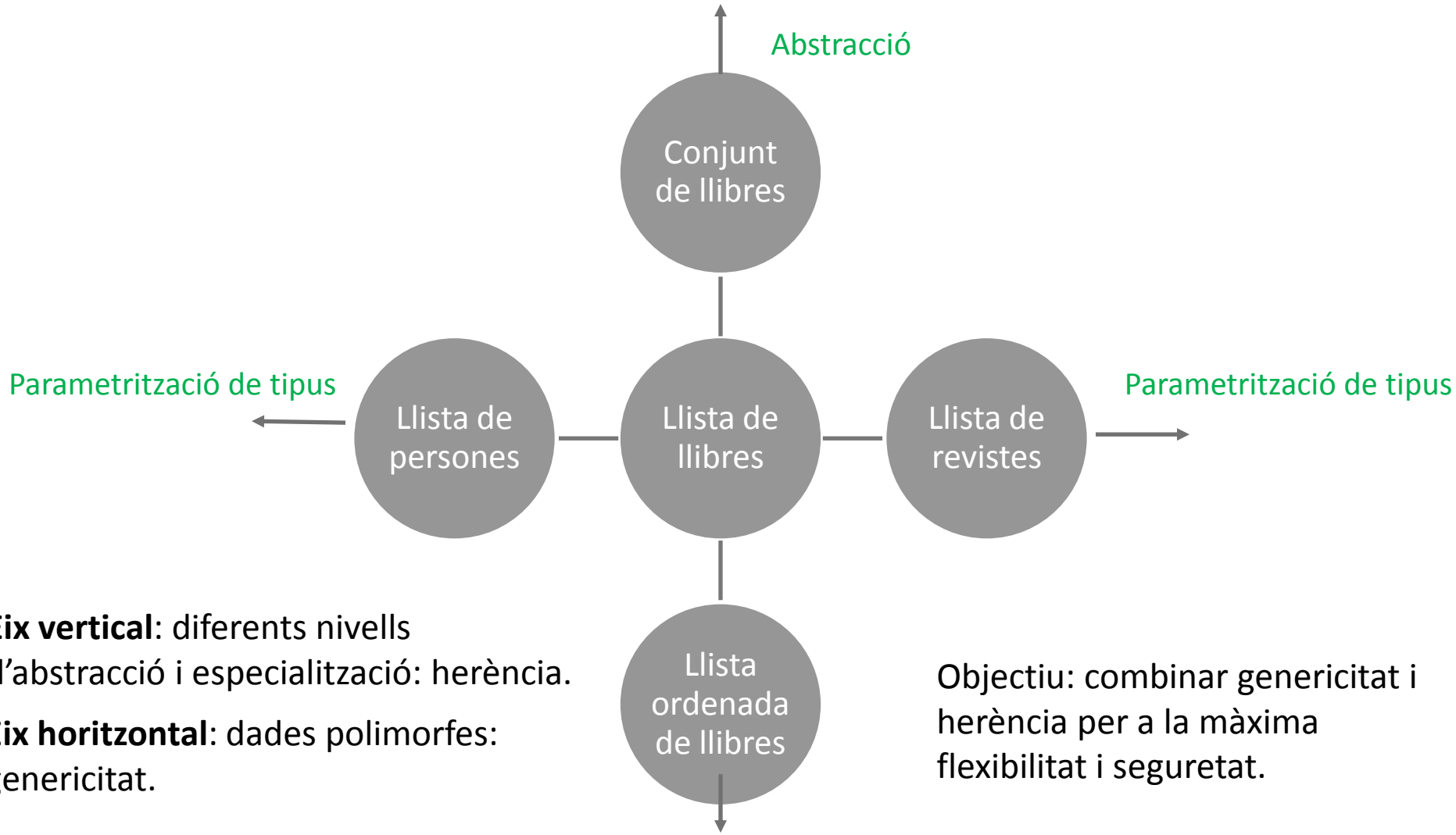
Casting

- *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;
```

HERÈNCIA I JERARQUIA DE CLASSES

Generalització



Eix vertical: diferents nivells d'abstracció i especialització: herència.

Eix horitzontal: dades polimorfes: genericitat.

Objectiu: combinar genericitat i herència per a la màxima flexibilitat i seguretat.

Herència

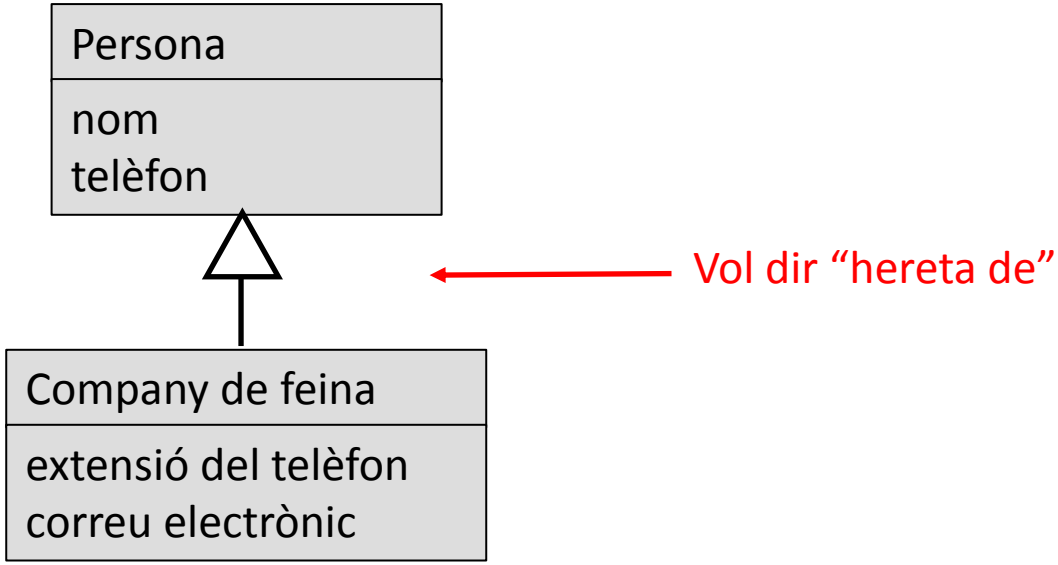
- L'herència és un mecanisme que permet definir una classe nova a partir d'una d'anterior descrivint les diferències entre elles.
- Característica pròpia de la programació orientada a objectes.
- Facilita la reutilització
- Concepte de relacions de generalització i especialització

Tipologies d'herència

- Depenent de la manera d'arribar-hi a l'herència, s'anomenen:
  - Herència per especialització
  - Herència per generalització

Herència per especialització

- Sorgeix de la necessitat de crear una classe nova que afegeixi unes propietats i un comportament a una altra classe del domini ja existent.
- Quan afegim funcionalitat a un disseny ja donat.
- Exemple:



Herència per generalització

- Apareix amb la finalitat d'homogeneïtzar el comportament de les parts comunes a certes classes.
- Quan es crea el disseny de classes.

Cotxe
rodes potencia seients

Moto
rodes potencia requereix casco

Camió
rodes potencia carrega

Millor utilitzar herènica

Vehicle
rodes potencia

Cotxe
seients

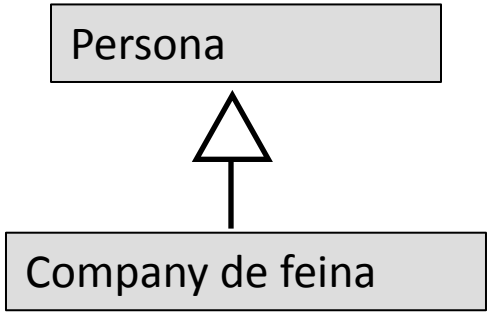
Moto
requereix casco

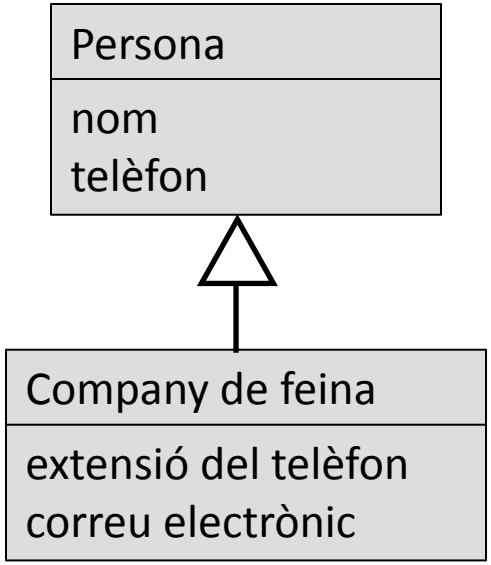
Camió
carrega

177

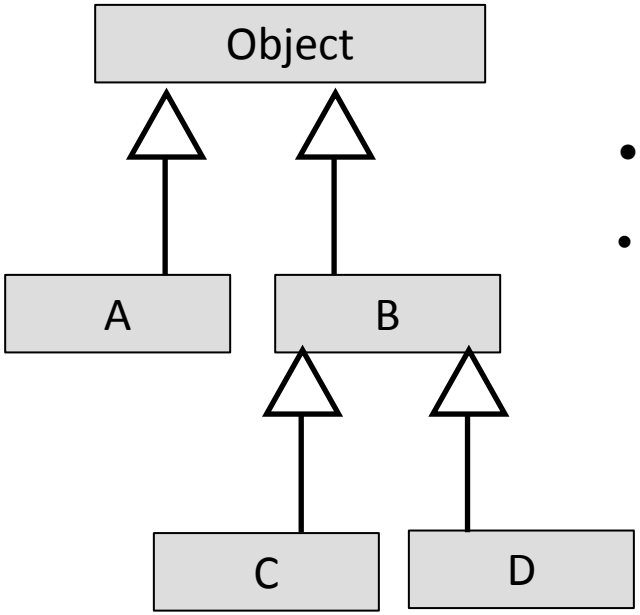
Herència

- Terminologia:





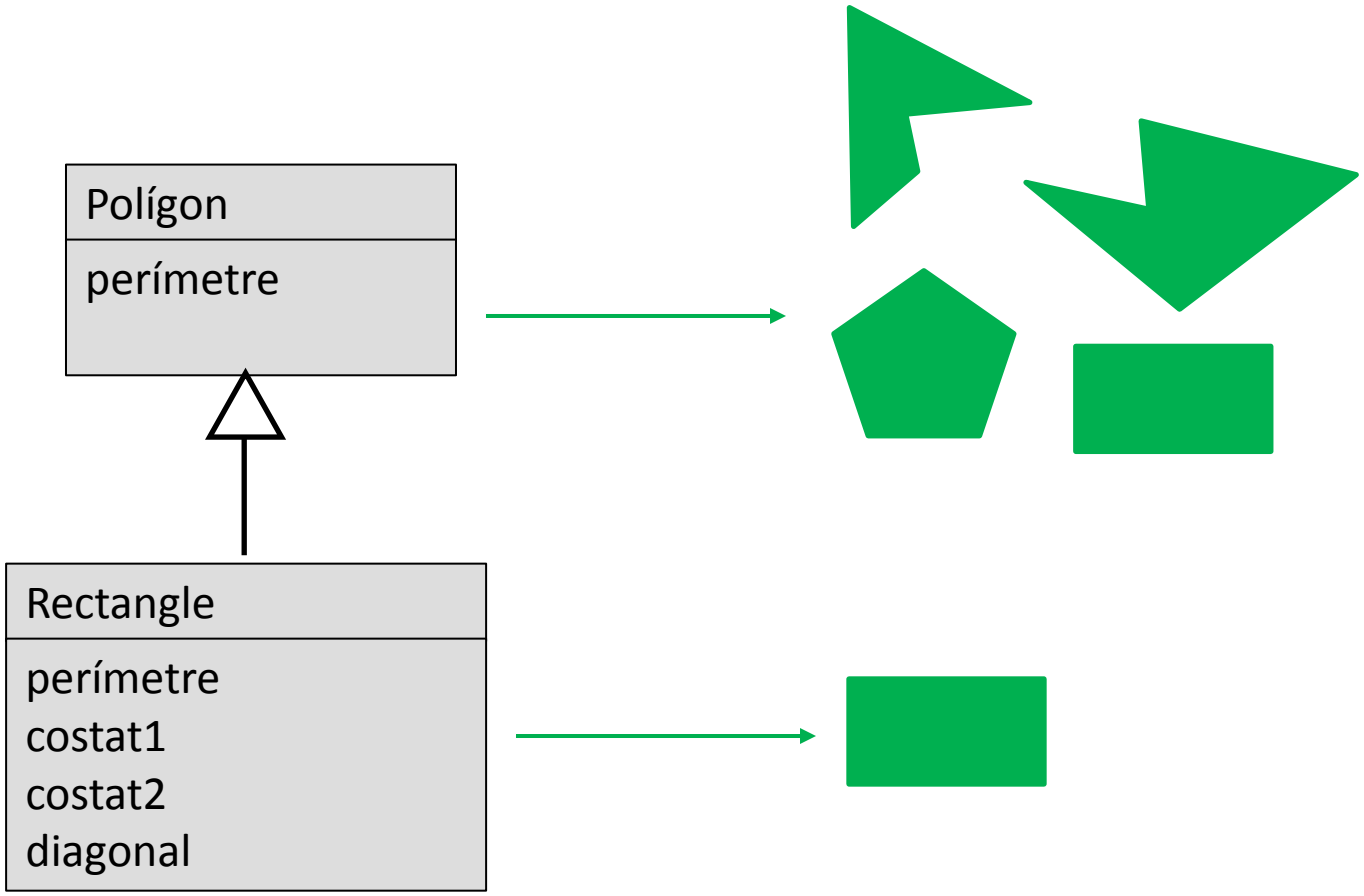
- Java:



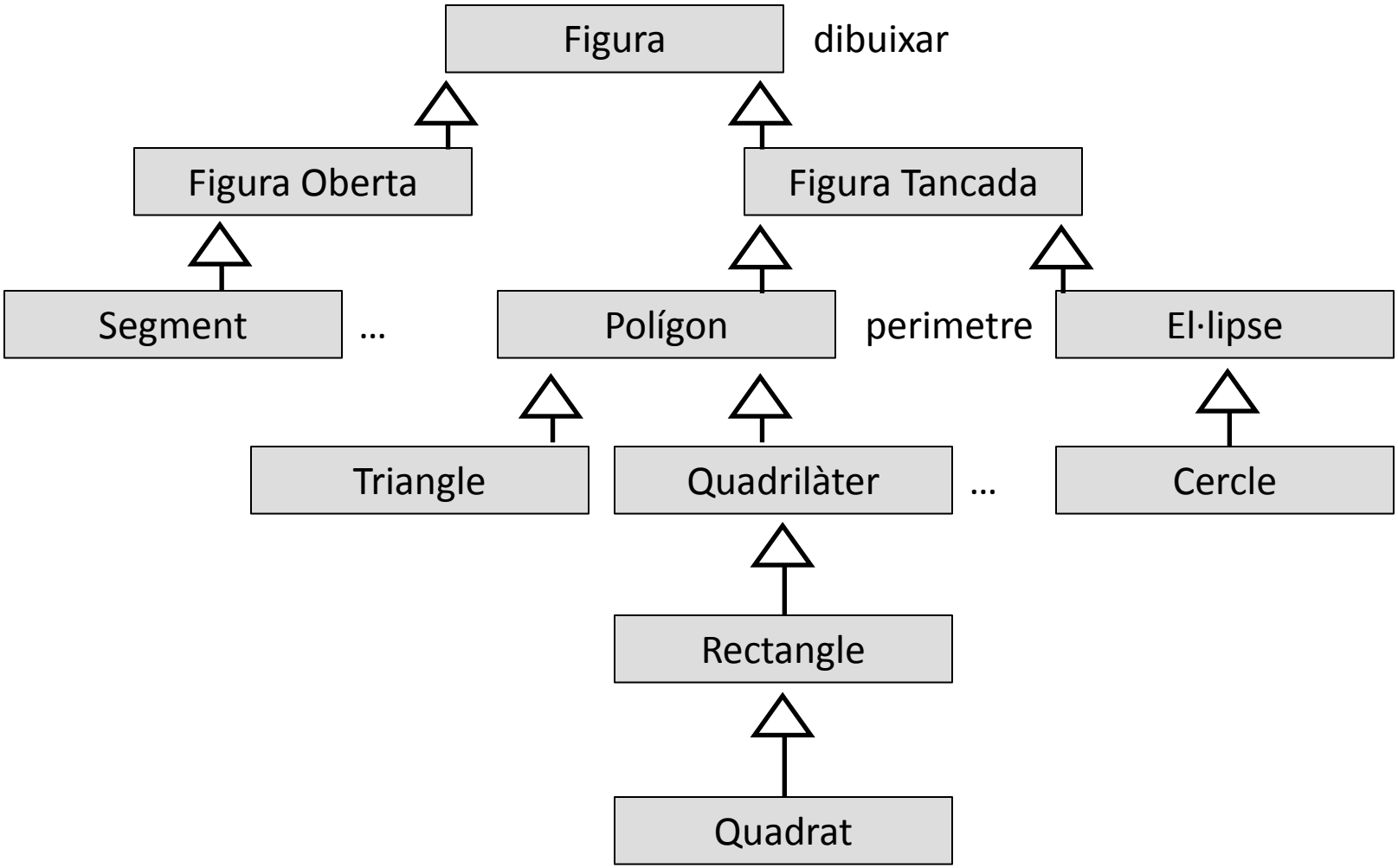
- `Object` és la classe arrel (paquet **`java.lang`**)
- **`Object`** descriu les propietats comunes a tots els objectes
  - `C` i `D` són **subclasses** de `B`
  - `B` és la **superclasse** de `C` i `D`

Herència

- Exemple



Jerarquia de classes



Consideracions sobre l'herència

- Els atributs i mètodes de la superclasse estaran **sempre definits en la subclasse.**
  - Aquesta restricció només s'aplica als atributs i mètodes definits amb la **visibilitat public o protected.**
  - Les classes filles no tenen accés als atributs i mètodes definits com a **private**.

Consideracions sobre l'herència

Atributs:

- En una classe filla, podem afegir **nous atributs.**
- S'ha de vigilar a l'hora de **triar els noms** dels nous atributs.

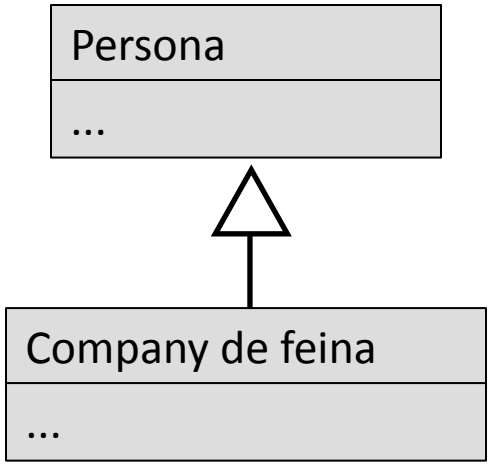
Consideracions sobre l'herència

Mètodes:

- Podem realitzar 3 tasques diferents:
  - Afegir mètodes nous
  - Implementar mètodes declarats prèviament com abstractes
  - Tornar a implementar mètodes.

Afegir mètodes nous

- Atributs nous → necessitem mètodes nous
  - Mètodes que realitzin tasques específiques de la classe filla.
- Els mètodes definits en la subclasse es consideraran mètodes d'aquesta i només s'hi podrà accedir des d'instàncies d'aquesta o de les seves classes filla.



Tipus de classes

- **Abstract**
- **Final**
- **Public**
- **Synchronizable**

Tipus de classes

- **Classe abstracta:**

  No s'instancia, sinó que s'utilitza com classe base per a l'herència.

- Exemple:

Classificació animal:

  - Mamífer,
  - Bípede,
  - Quadrúpede,

  → D'aquests conjunts no hi ha instàncies concretes

  La balena és un mamífer, però de la subespècie dels cetacis

  El cavall és un quadrúpede, de la subespècie dels equins

Tipus de classes

- **Classe final**

  Se declara com la classe que termina una cadena d'herència. No es pot heretar d'ella.

- Exemple:

La classe **Math** és una classe final.

Tipus de classes

- **Classes public**

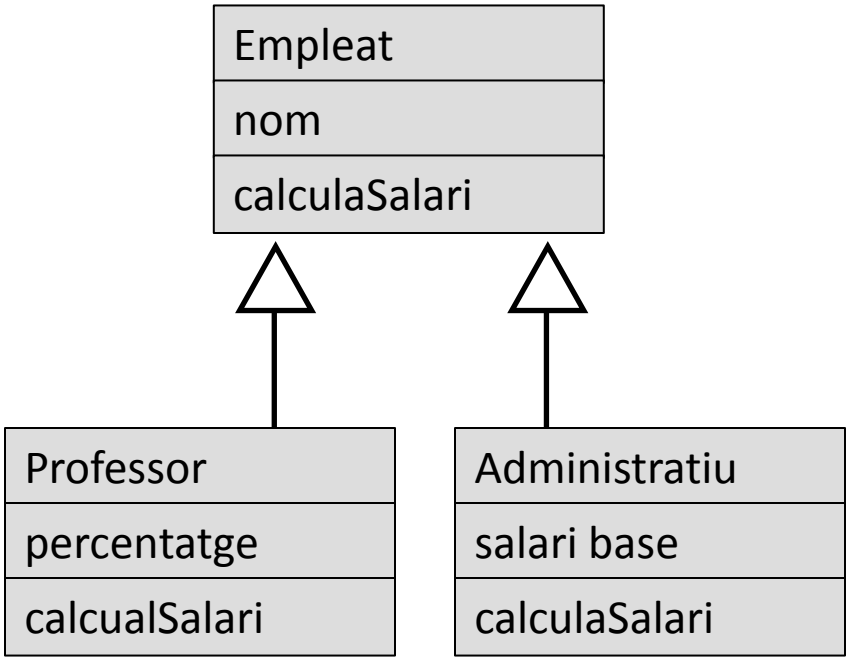
  Són accessibles des d'altres classes, o directament o per herència.
- Són accessibles dins del mateix paquet en el que s'han declarat.
- Per a accedir des d'altres paquets, primer tenen que ser importades.

Tipus de classes

- **Classe synchronizable**
- Aquest modificador especifica que tots els mètodes definits en la classe són sincronitzats, es a dir, que no es poden accedir al mateix temps a ells des de diferents threads;
- El sistema s'encarrega de col·locar els flags necessaris per a evitar-ho.
- Aquest mecanisme fa que des de threads diferents es puguin modificar les mateixes variables sense que hagi problemes de sobreescriptura.

Implementar mètodes abstractes

- **Mètode abstracte** és aquell que té definida la seva interfície (nom, tipus, nombre de paràmetres i valor de retorn), però no té implementat el codi que atendrà les peticions.
- Exemple:



Implementar mètodes abstractes

- Si una classe té declarat com a mínim un mètode abstracte, es diu que la **classe** és **abstracta**.
- Les classes abstractes obliguen les classes que hereten d'aquesta a implementar els mètodes no implementats.

En cas que una classe filla continuï sense implementar un mètode abstracte, aquesta ha de ser també abstracta.

Sobreescriptura de mètodes

- Ens permet modificar el comportament d'un mètode definit prèviament en la classe mare per que realitzi altres tasques.

- Cal tornar a definir-lo i implementar-lo amb una **signatura igual o diferent**.

Amb Java

- Per definir una herència:

  paraula reservada ***extends***

  \+ nom de la classe de la qual s'hereta

- Per accedir als mètodes definits a la classe mare:

  paraula reservada ***super***

Ús d'herència

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

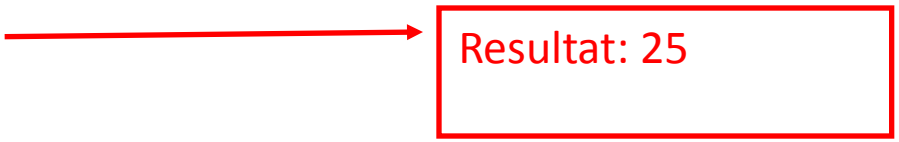
```
import MiClase;
public class MiNuevaClase extends MiClase {
  public void suma_a_i( int j ) {
    i = i + ( j/2 );
    super.suma_a_i( j );
  }
}
```

sobreescriptura

Fa referència al mètode de la classe mare

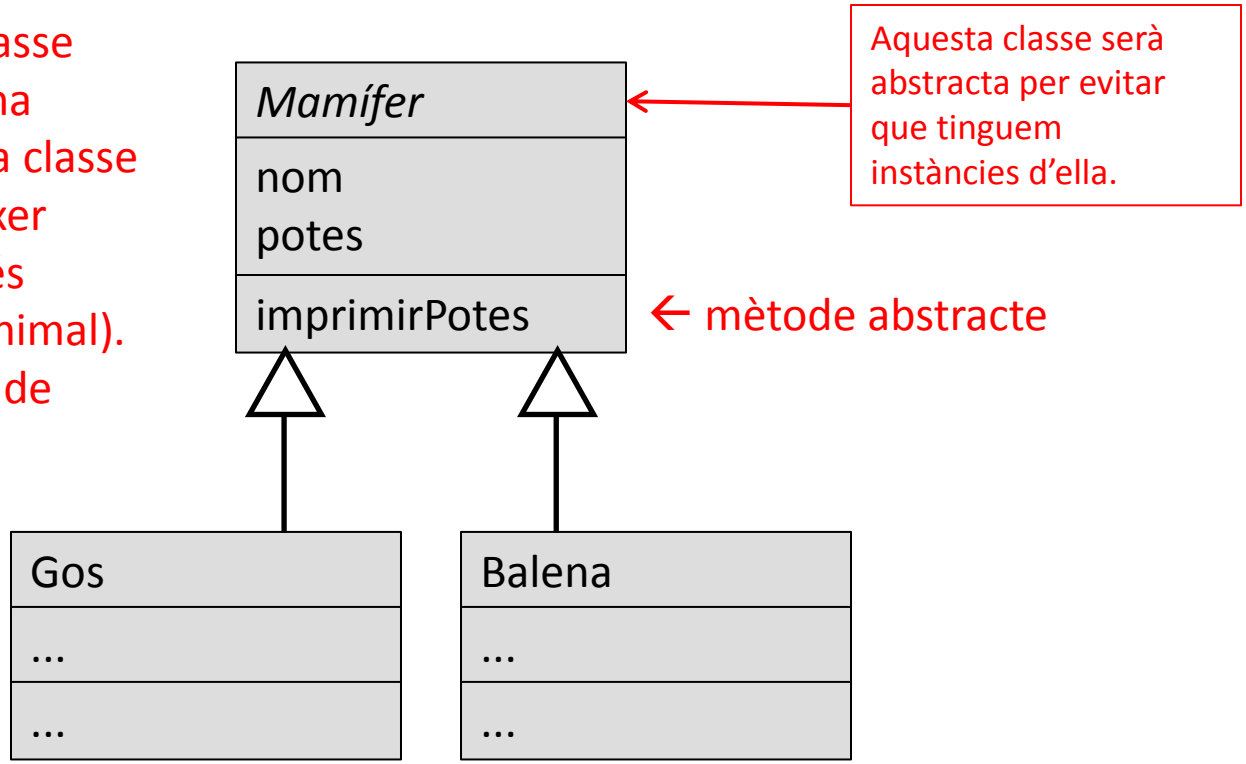
Ús d'herència

```
public static void main(String[] args) {
        MiNuevaClase mnc;
        mnc = new MiNuevaClase();
        mnc.suma_a_i( 10 );

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



Exemple: Classe abstracta

Mamífer serà una classe abstracta, ja que hi ha informació d'aquesta classe que no es pot conèixer sense especificar més (saber més sobre l'animal). Exemple: el número de potes de l'animal.



```java
public abstract class Mamifer {
    private String nom;
    private int potes;
    public void imprimirPotes() {
        System.out.println(nom + " té " + potes + " potes\n");
    }
    public Mamifer(String nom, int potes) {
        this.nom = nom;
        this.potes = potes;
    }
}
```

Mamifer.java

```java
public class Gos extends Mamifer {
    public Gos(String nom){
        super(nom, 4);
    }
}
public class Balena extends Mamifer {
    public Balena(String nom){
        super(nom, 0);
    }
}
```

Gos.java

Balena.java

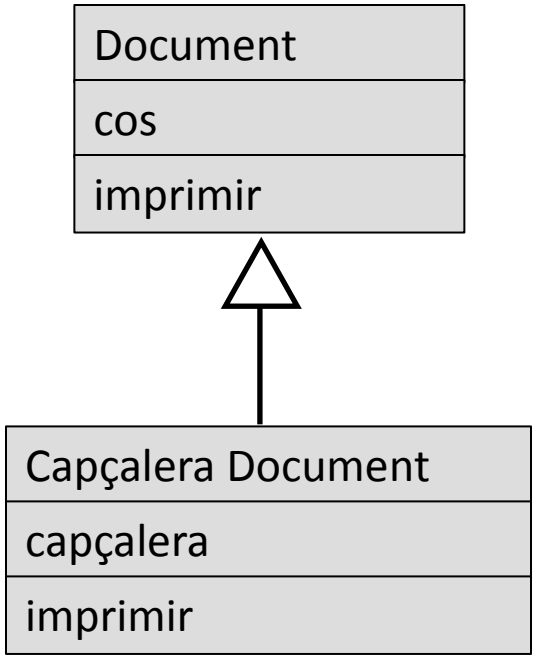
S'invoca a la versió del mètode de la primera superclasse que el continga

```java
public class CreaGos {
    public static void main(String [] args) {
        Gos bobi = new Gos("Bobi");
        bobi.imprimirPotes(); /*Està a la classe mamífer*/
    }
}
```

CreaGos.java

Exemple

- Herència amb sobreescriptura de mètodes:



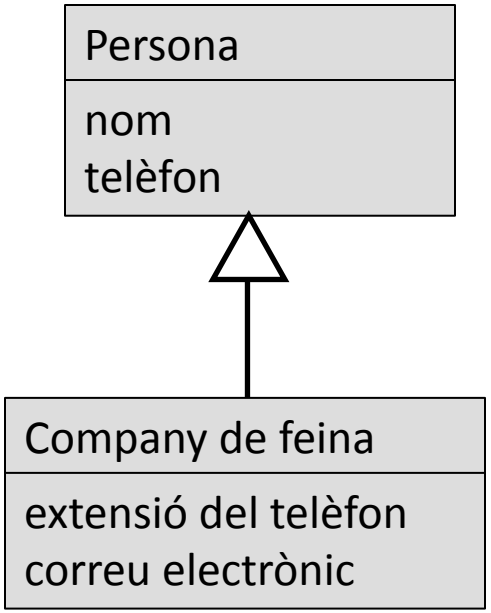
Entenem DocumentCapçalera com un tipus específic de document que té a més d'un cos de document una capçalera.

Les funcions a realitzar pel mètode imprimir ara seran diferents, ja que tenim una informació diferent emmagatzemada.

Per fer...

Exemple

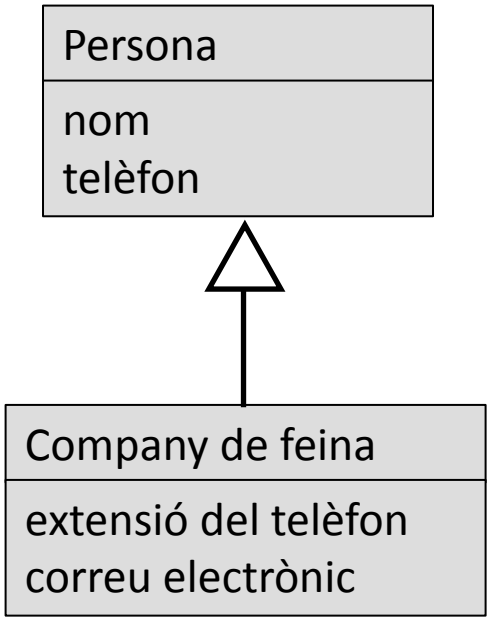
- Implementar l'exemple:



Solució:

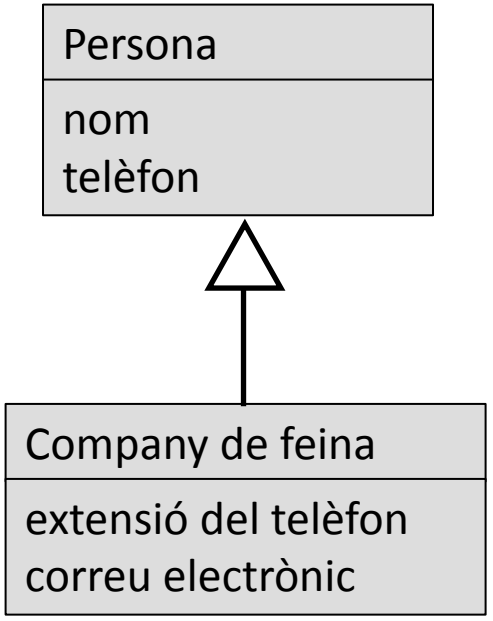
Persona.java

```
public class Persona {
    private String nom;
    private String telefon;
    // constructor
    public Persona (String pNom, String pTelefon) {
       nom = pNom;
       telefon = pTelefon;
    }
    // Getters i setters
    public String getNom() {
       return nom;
    }
    public String getTelefon() {
       return telefon;
    }
    public void setNom(String pNom)  {
       nom = pNom;
    }
    public void setTelefon(String pTelefon) {
       telefon = pTelefon; }
}
```



Company.java

```
public class Company extends Persona {
    private String extTel;
    private String email;
    // constructors:
    public Company(String pNom, String pTelefon) {
       super(pNom, pTelefon);
       extTel = "";
       email = "";}
    public Company(String pNom, String pTelefon, String pExtTel, String pEmail) {
       super(pNom, pTelefon);
       extTel = pExtTel;
       email = pEmail;}

    // Getters i setters
    public String getExtTel() {
       return extTel;}

    public String getEmail() {
       return email;}

    public void setExtTel(String pExtTel) {
       extTel = pExtTel;}

    public void setEmail(String pEmail) {
       email = pEmail;}
}
```



Exercici:

Donat el codi anterior de les classe Persona i Company indicar si hi ha errors de compilació en les següents classes del mateix paquet:

1. Classe TestCompanys1

```
public class TestCompanys1 {
   public static void main(String[] args){
        Company nouCompany = new Company();
    }
}
```

2. Classe TestCompanys2

```
public class TestCompanys2 {
    public static void main(String[] args){
        String nom="Joan";
        String telefon="93111111";
        String telefonActual;
        Company nouCompany = new Company(nom, telefon);
        System.out.println(nouCompany.getNom());
        telefonActual = "93222222";
        nouCompany.setTelefon(telefonActual);
    }
}
```

Solució Exercici:

Donat el codi de les classe Persona i Company indicar si hi ha errors de compilació en les següents classes del mateix paquet:

1. Classe TestCompanys1

```
public class TestCompanys1 {
   public static void main(String[] args){
        Company nouCompany = new Company();
    }
}
```

Error de compilació:
La classe Company no té constructor sense paràmetres.

2. Classe TestCompanys2

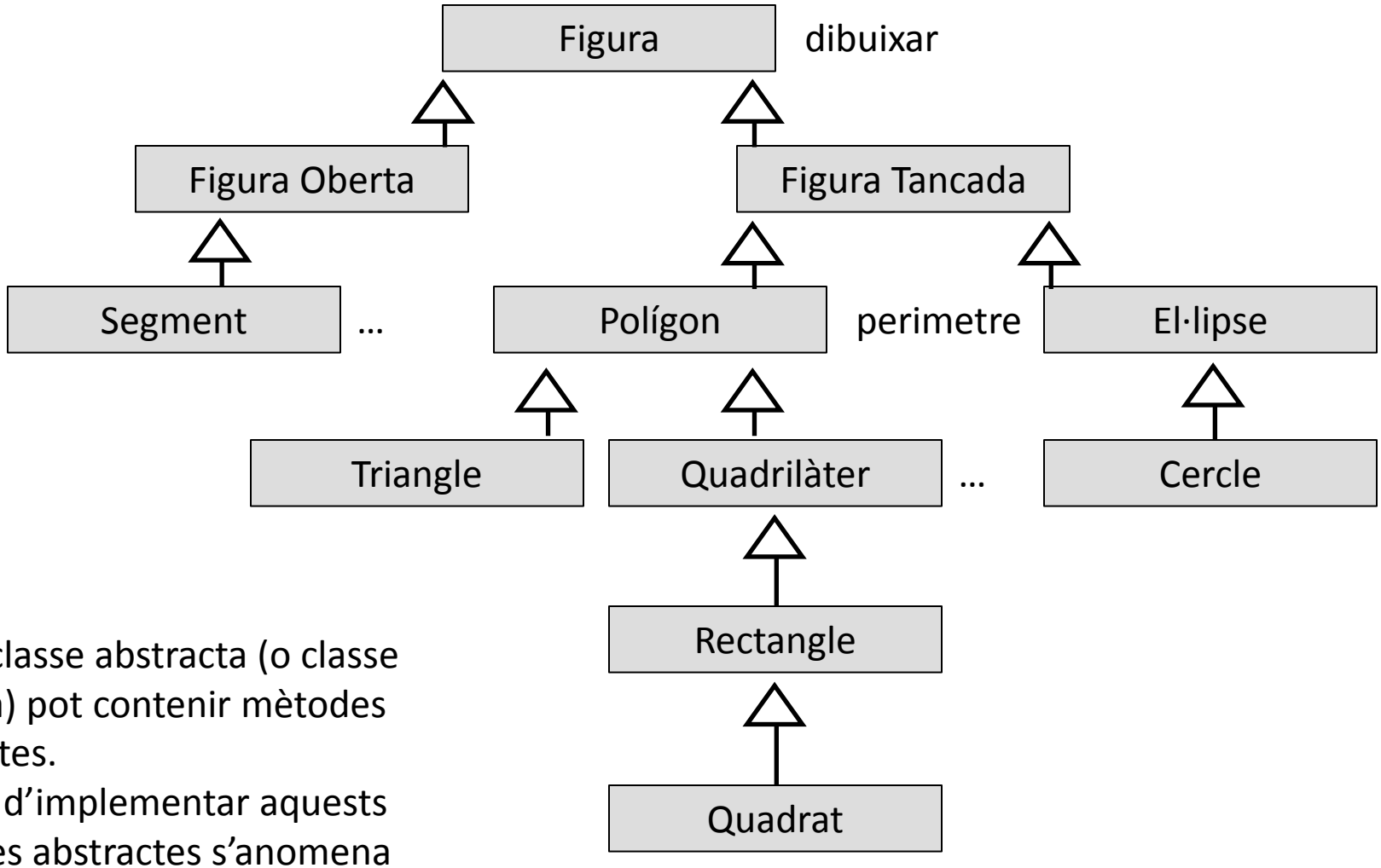
```
public class TestCompanys2 {
    public static void main(String[] args){
        String nom="Joan";
        String telefon="93111111";
        String telefonActual;
        Company nouCompany = new Company(nom, telefon);
        System.out.println(nouCompany.getNom());
        telefonActual = "93222222";
        nouCompany.setTelefon(telefonActual);
    }
}
```

Donarà error?

Donarà error?

No. Encara que la classe Company no té els mètodes getNom I getTelefon implementats, la superclasse Persona si que els té.

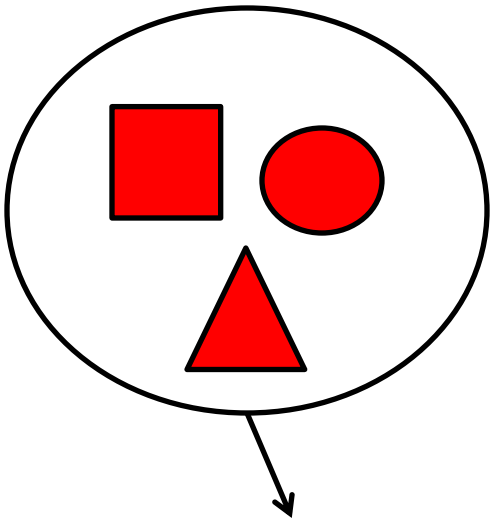
Exemple de jerarquia de classes



- Una classe abstracta (o classe diferida) pot contenir mètodes abstractes.
- El fet d'implementar aquests mètodes abstractes s'anomena ***fer efectiu*** un mètode.

Exemple

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



Figures geomètriques

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

No pot haver-hi una instància d'una classe abstract

Atributs i mètodes heretats

Quadrat
color posició a pantalla àrea perímetre costats
calcula àrea calcula perímetre retorna color assigna color

Circumferència
color posició a pantalla àrea perímetre radi
calcula àrea calcula perímetre retorna color assigna color

Triangle
color posició a pantalla àrea perímetre dimensió 3costats
calcula àrea calcula perímetre retorna color assigna color

Figura.java

```
public abstract class Figura {
   protected String color;
    protected double x, y;
    protected double area;
    protected double perimetre;

    // Mètodes abstractes:
    public abstract double calculaArea();
    public abstract double calculaPerimetre();

    //Retorna el Color
    public String getColor(){
       return color;
    }
    //Assigna el Color
     public void setaColor(String color){
       this.color=color;
    }

    //Retorna la posició de la Figura
        public double [] getPosicion(){
           double [] posicioxy = {x, y};
           return posicioxy;
        }
        //Assigna la posició de la Figura
        public void setPosicio(double[] posicioxy){
           x=posicioxy[1];
           y=posicioxy[2];
        }
} // Final de la classe Figura
```

Quadrat.java

```
public class Quadrat extends Figura {
   private double costat; // longitud dels costats
   // constructors
   public Quadrat() {
            costat=0.0;
   }
   public Quadrat(double costat) {
            this.costat = costat;
   }
   // Calcula l'àrea del quadrat:
   public double calculaArea() {
      area = costat * costat ;
      return area;
   }
   // Calcula el valor del perímetre:
   public double calculaPerimetre(){
      perimetre = 4 * costat;
      return perimetre;
   }
}
```

Cercle.java

```
public class Cercle extends Figura {
        public static final double PI=3.14159265358979323846;
        public double  radi;
         // constructors
           public Cercle(double x, double y, double radi) {  crearCercle(x,y,radi); }
           public Cercle (double radi) {  crearCercle(0.0,0.0,radi);  }
           public Cercle (Cercle c){ crearCercle(c.x,c.y,c.radi); }
           public Cercle() { crearCercle(0.0, 0.0, 1.0); }
           // Mètode de suport
           private void crearCercle(double x, double y, double radi) {
                 this.x=x; this.y=y; this.radi =radi;
           }
         // calcula l'area del cercle
           public double calculaArea() {
              area = PI * radi * radi;
              return area;
           }
           // calcula el valor del perímetre
           public double calculaPerimetre() {
              perimetre = 2 * PI * radi;
              return perimetre;
           }
} // fi de la classe Cercle
```

Exercici

- Amplia la implementació de la classe **Cercle** que hereta de la classe absracta Figura amb
  - Un contador de cercles,
  - Dos mètodes propis,
    - Un mètode d'objecte per comparar cercles i
    - Un mètode de classe per comparar cercles.

Cercle.java

```
public class Cercle extends Figura {
    static int numCercles = 0;
    public static final double PI=3.14159265358979323846;
    public double radi;

    // constructors
    public Cercle(double x, double y, double radi) {
        this.x=x; this.y=y; this.radi =radi;
        numCercles++;}
    public Cercle(double radi) { this(0.0, 0.0, radi); }
    public Cercle(Cercle c) { this(c.x, c.y, c.radi); }
    public Cercle() { this(0.0, 0.0, 1.0); }

     // calcula l'area del cercle
       public double calculaArea() {
          area = PI * radi * radi;
          return area;
       }
       // calcula el valor del perímetre
       public double calculaPerimetre(){
          perimetre = 2 * PI * radi;
          return perimetre;
       }

    // mètode d'objecte per a comparar cercles
    public Cercle elMajor(Cercle c) {
            if (this.radi>=c.radi)
                        return this;
            else return c;
    }

    // mètode de classe per a comparar cercles
    public static Cercle elMajor(Cercle c, Cercle d) {
            if (c.radi>=d.radi)
                        return c;
            else return d;
    }

} // fi de la classe Cercle
```

TestCercles.java

```
public class TestCercles {
        public static void main(String[] args){
                Cercle cercleGran;
                System.out.println("número de cercles = " + Cercle.numCercles);
                Cercle cercle1 = new Cercle(1.5);
                System.out.println("número de cercles = " + Cercle.numCercles);
                Cercle cercle2 = new Cercle(2.5);
                System.out.println("número de cercles = " + Cercle.numCercles);
                Cercle cercle3 = new Cercle(3.5);
                System.out.println("número de cercles = " + Cercle.numCercles);

                cercleGran = Cercle.elMajor(cercle1, cercle2);
                System.out.println("El radi del cercle gran és = " + cercleGran.getRadi());

                cercleGran = cercle3.elMajor(cercle2);
                System.out.println("El radi del cercle gran és = " + cercleGran.getRadi());

        }
} // fi de la classe
```

Sortida per pantalla →

```
número de cercles = 0
número de cercles = 1
número de cercles = 2
número de cercles = 3
El radi del cercle gran és = 2.5
El radi del cercle gran és = 3.5
```

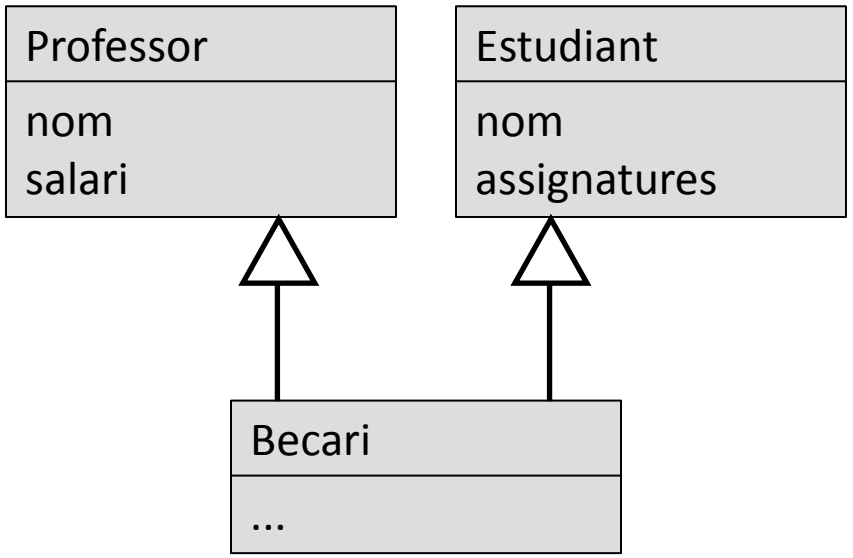
```
public class Cercle extends Figura {
   // quantitat d'objectes d'aquesta classe que existeixen.
   static int numCercles = 0;
   // constant PI
   private static final double PI=3.14159265358979323846;
   // radi del cercle
   private double radi;

   // constructors
   public Cercle(double x, double y, double radi) {
         this.x=x; this.y=y; this.radi =radi;
         // actualitza la quantitat d'objectes d'aquesta classe
         sumarCercle();
   }
   public Cercle(double radi) { this(0.0, 0.0, radi); }
   public Cercle(Cercle c) { this(c.x, c.y, c.radi); }
   public Cercle() { this(0.0, 0.0, 1.0); }
    // calcula l'area del cercle
    public double calculaArea() {
         area = (double) (PI * radi * radi);
         return area;  }
      // calcula el valor del perímetre
      public double calculaPerimetre(){
         perimetre = (double) (2 * PI * radi);
         return perimetre;
      }
   // mètode d'objecte per a comparar cercles
   public Cercle elMajor(Cercle c) {
         if (this.radi>=c.radi) return this;
         else return c;    }
```

```
   // mètode de classe per a comparar cercles
   public static Cercle elMajor(Cercle c, Cercle d) {
   if (c.radi>=d.radi) return c; else return d;
   }
   public double getRadi(){
      return this.radi;
   }
   // destructor
   protected void finalize() {
      // actualitza la quantitat d'objectes d'aquesta classe que existeixen:
      restarCercle();
   }

   // mètode de classe que incrementa en un la quantitat d'objectes d'aquesta
classe que existeixen.
   private static void sumarCercle(){
         numCercles++;
   }

   // mètode de classe que decrementa la quantitat d'objectes creats dins
d'aquesta classe.
   private static void restarCercle(){
      numCercles--;
   }

} // fi de la classe Cercle
```

Una altra possible implementació de la classe Cercle.

Herència múltiple

- L'herència en què la classe nova és generada a partir de dues o més classes alhora.
- Exemple:

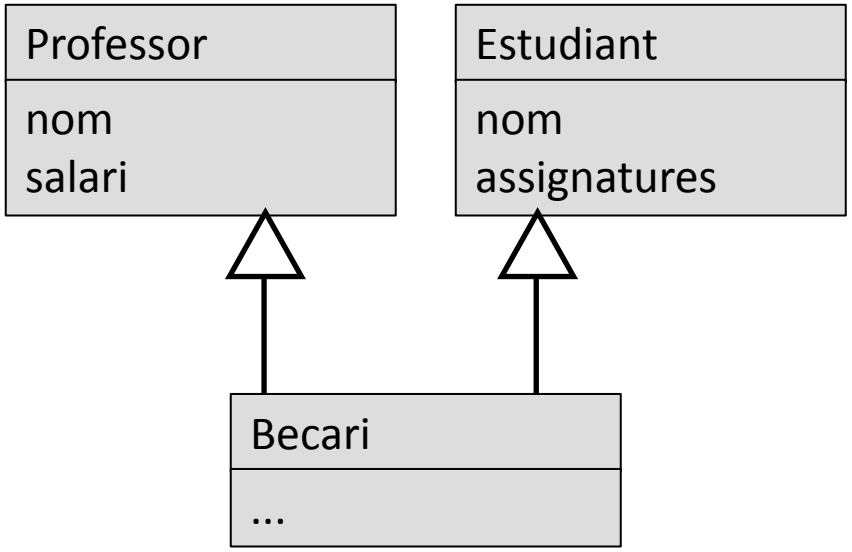


Quin pot ser el problema?

Problema: el becari té dos atributs nom

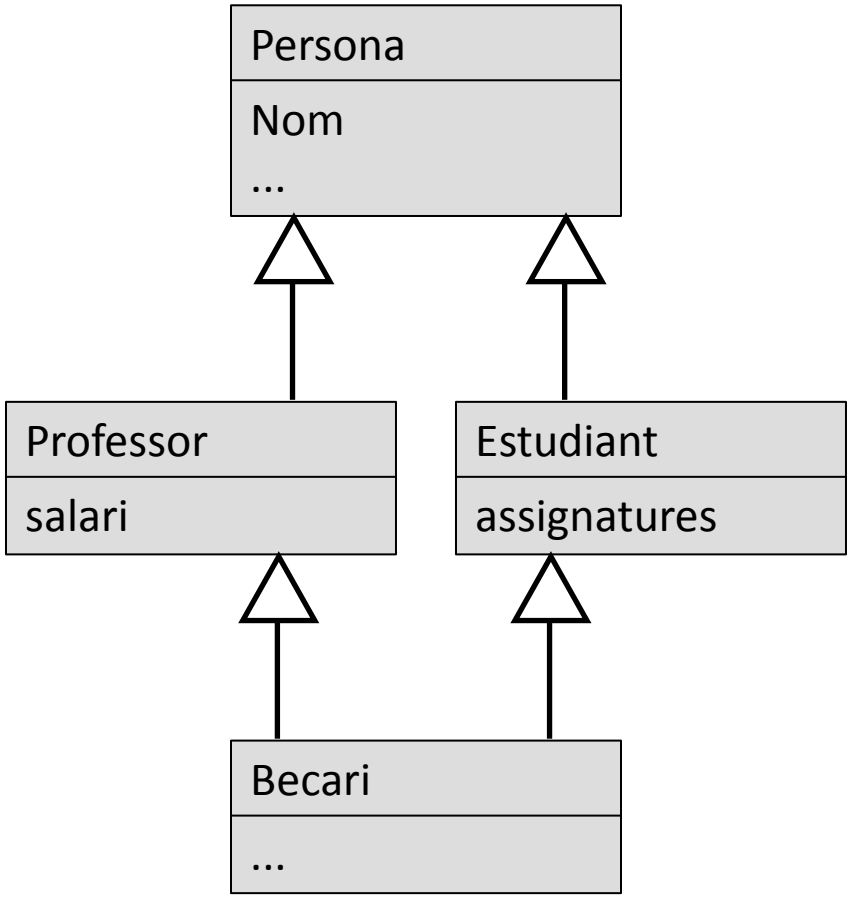
Herència múltiple: Duplicitat d'atributs i mètodes

- Podem trobar que una classe té un atribut o mètode repetit perquè hereta de classes que contenen el mateix atribut o mètode.
- Calen mecanismes per a pal·liar aquesta problemàtica.



Duplicitat d'atributs i mètodes

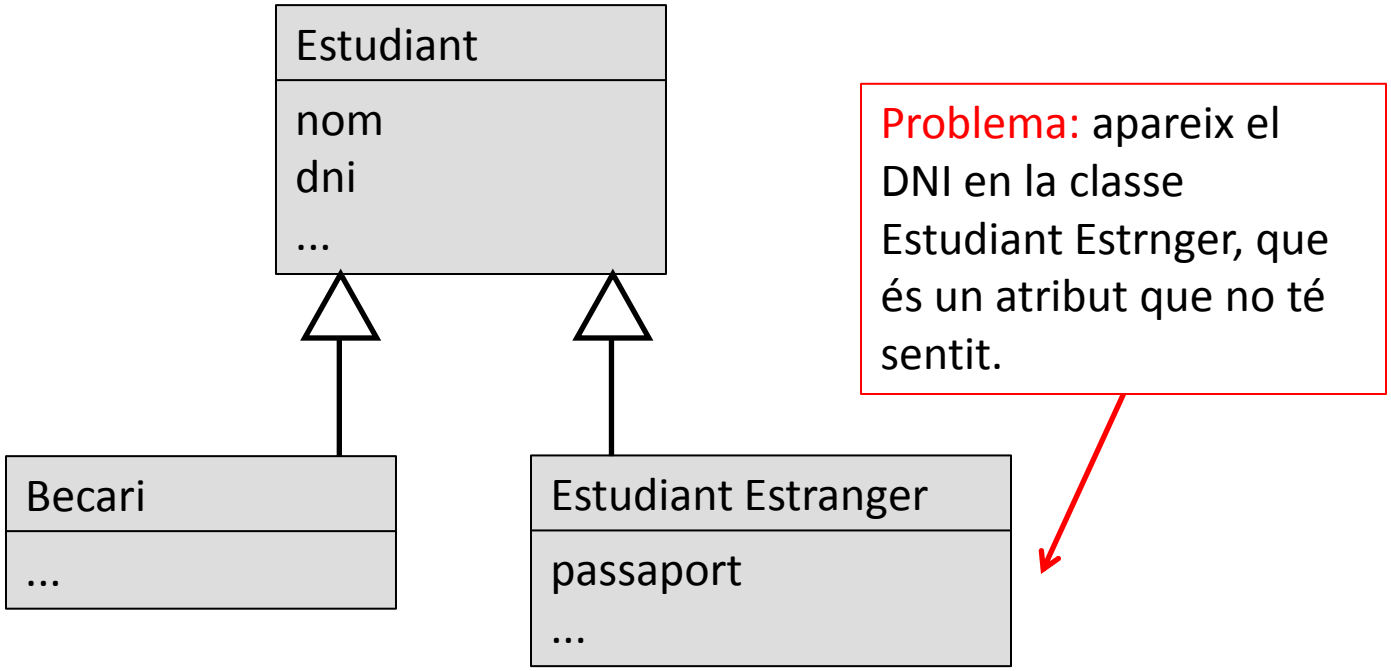
- Cas en que sempre es produiran duplicitats:



Errors típics de l'herència

1. Creació de superclasses poc generals
2. Ús de subclasses en comptes d'una superclasse

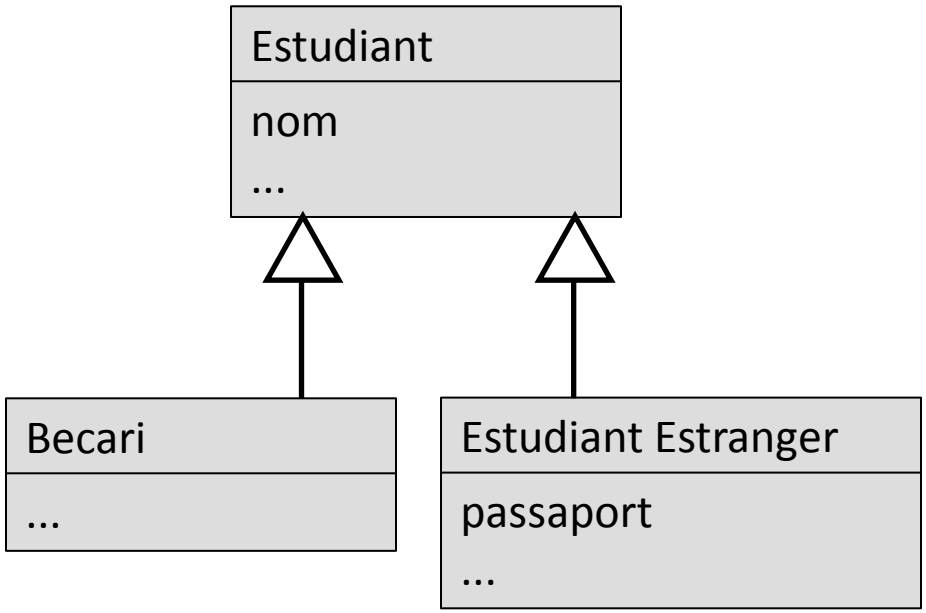
Creació de superclasses poc generals



Evitar-ho al disseny.

Ús de subclasses en comptes d'una superclasse

- Si volem emmagatzemar una relació d'estudiants, podem definir un vector per a emmagatzemar les instàncies de la classe Becari i Estudiant Estranger;
- En recuperar-los, com que poden estar barrejats, hem d'utilitzar la superclasse Estudiant, però si volem accedir a mètodes definits en la subclasse hem d'utilitzar el càsting.



Conversió de tipus

- El càsting (o conversió de tipus) ens permet utilitzar una instància d'una classe com si es tractés d'una instància d'un altre tipus.

Tot i que la definició anterior es completament certa, cal matitzar-la, ja que podrem realitzar el procés de càsting sempre que la conversió sigui possible.

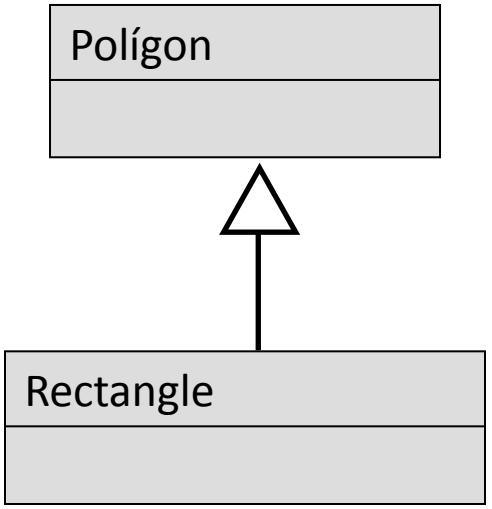
Conversió de tipus

- **<u>Conversió implícita:</u>** (automàtica)
  - *Tipus primitius* a un que suporti un rang major de valors

```
float saldo = 300;    //podem assignar-li un enter
int codi = 3.7;       //Donarà ERROR
```

  - *Referències*: tot objecte conté una instància de les seves superclasses
    - ***cast-up***
    - sempre vàlid

```
Poligon poligon;
Rectangle rectangle = new Rectangle();
poligon = rectangle;
```



Conversió de tipus

- **<u>Conversió explícita:</u>**
  - *Tipus primitius:* perdent informació

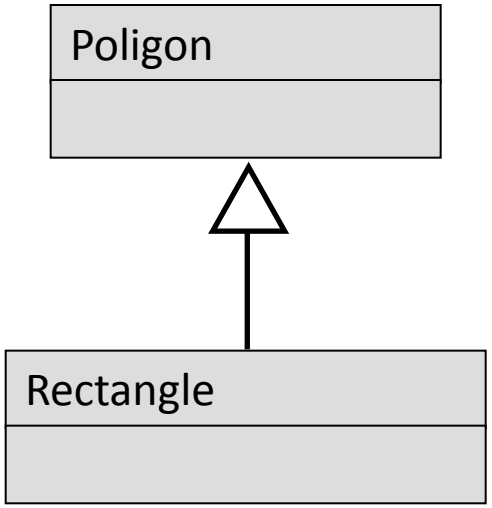
```
long l = 200;
int i = (int)l;
```

  - *Referències:* assignar a un objecte d'una subclasse un de la superclasse
    - ***cast-down*** o *narrowing*
    - No sempre vàlid
    - L'error es pot produir:
      - en temps d'execució (**`ClassCastException`**)
      - en temps de compilació si no és ni tan sols una subclasse.

Conversió explícita de referències

- Pot donar un error en execució:

```
Poligon [] poligons = new Poligon [30];
…
Rectangle r = (Rectangle)poligons[i];
```



- Donaria error en compilació:

```
Compte c = (Compte)poligons[i];
```

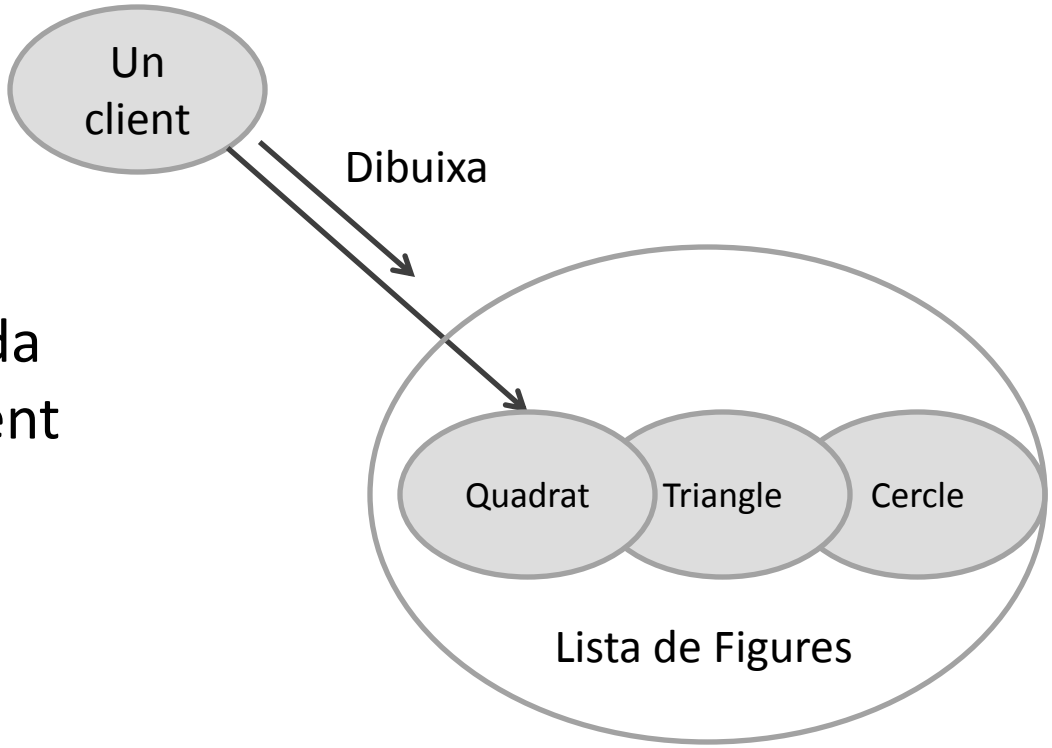
POLIMORFISME

Polimorfisme

- *Origen: poli (‘diversos’) i morfos (‘forma’)*
- El polimorfisme està lligat estretament amb l’herència
- És la propietat per la qual es poden realitzar tasques diferents invocant la mateixa operació, segons el tipus d’objecte sobre el qual s'invoca.

Polimorfisme

- El Polimorfisme provocarà un canvi de comportament d'una operació depenent de l'objecte al qual s'aplica.



- L'operació és única, però cada classe defineix el comportament d'aquella operació.

Polimorfisme

- És la propietat d'ocultar l'estructura interna d'una jerarquia de classes implementant un conjunt de mètodes de manera independent i diferenciada en cada classe de la jerarquia.
- El polimorfisme **apareix** quan definim un mètode en una classe de la jerarquia (generalment la superclasse) i el reescrivim en, com a mínim, alguna de les classes que formen la jerarquia.
- La reescriptura de mètodes només pot existir en subclasses de la classe en què es defineix o implementa el mètode per primera vegada.

Polimorfisme

- El concepte de polimorfisme es pot aplicar tant a **mètodes** com a **tipus de dades**.
- Així neixen els conceptes de:
  - *Mètodes polimórfics,* són aquells mètodes que poden avaluar-se o ser aplicats a diferents tipus de dades de forma indistinta;
  - *Tipus polimòrfics*, són aquells tipus de dades que contenen al menys un element amb tipus no especificat.

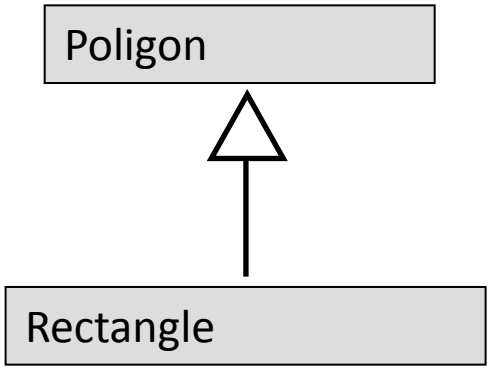
Polimorfisme

- Assignació polimorfa:

  Dues maneres:

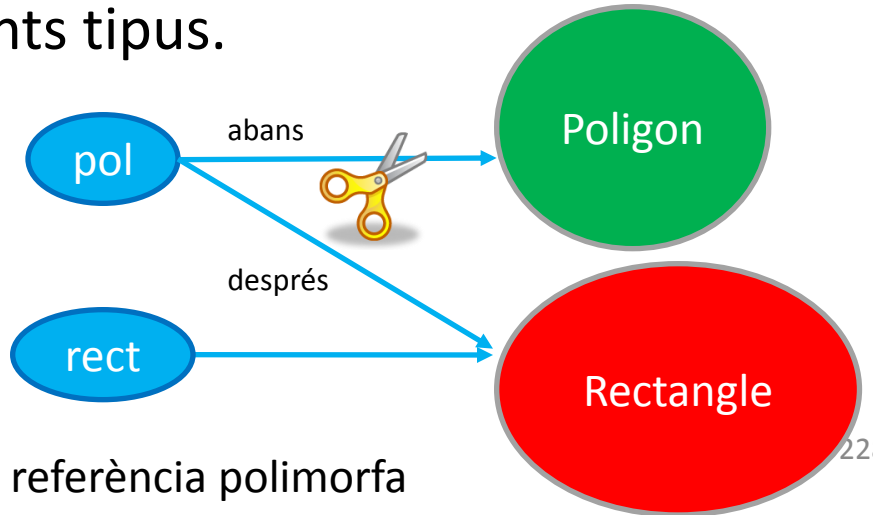
```
Poligon pol = new Poligon();
Rectangle rect = new Rectangle();
pol = rect;
```

```
Poligon pol = new Rectangle();
```



- Connexió polimorfa (assignació i passo de paràmetres): quan l'origen i el destí tenen diferents tipus.

- Que passa durant una connexió polimorfa?



Reconnexió de la referència polimorfa

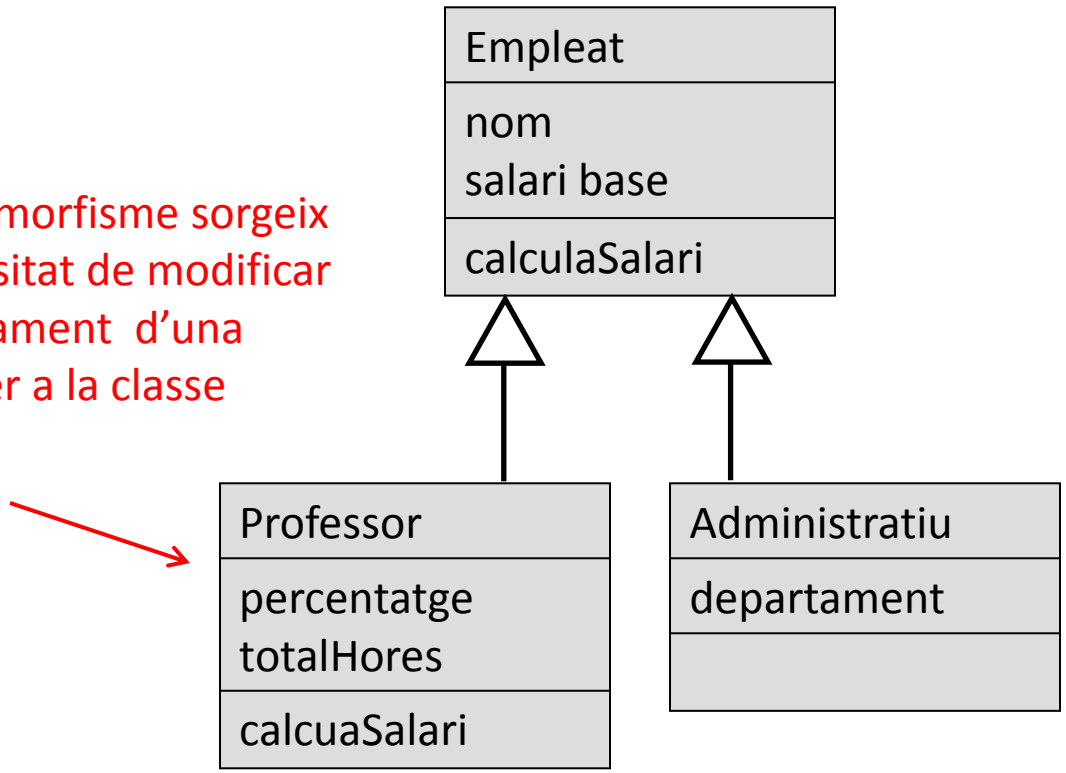
Polimorfisme

- El polimorfisme permet que es decideixi en temps d'execució i de manera automàtica quin dels mètodes cal executar: el mètode heretat o, en cas que existeixi, el mètode sobreescrit.

Exemple 1

- Implementació de l'exemple:

Aquí el polimorfisme sorgeix de la necessitat de modificar el comportament d'una operació per a la classe Professor



```
public abstract class Empleat{
    private String nom;
    private float salariBase;
    public Empleat( String nom, float salariBase) {
            this.nom = nom;
            this.salariBase = salariBase;
    }
    public String getNom() {
            return nom; }
    public float getSalariBase() {
            return salariBase; }
    public void setNom(String nom) {
             this.nom = nom; }
    public void setsalariBase(float salariBase ) {
            this.salariBase = salariBase; }
    public float calculaSalari() {
            float salari = (float) (salariBase * 1.5);
            return salari;
    }
}
```

Empleat.java

Administratiu.java

```
public class Administratiu extends Empleat {
    private String department;
    public Administratiu (String nom, float salariBase, String department) {
        super(nom, salariBase);
        this.department = department;
    }
    public String getDepartment() {
        return department;
    }
    public void setDepartment(String department) {
        this.department = department;
    }
}
```

Professor.java

```
public class Professor extends Empleat {
    private float percentatge;
    private float totalHores;
    // constructor
    public Professor(String nom, float salariBase, float percentatge, float totalHores) {
        super(nom, salariBase);
        this.percentatge = percentatge;
        this.totalHores = totalHores;}
    // Getters i setters
    public float getPercentatge() {
        return percentatge;}
    public float getTotalHores() {
        return totalHores;}
    public void setPercentatge(float percentatge) {
        this.percentatge = percentatge;}
    public void setTotalHores(float totalHores) {
        this.totalHores = totalHores;}
    // Reescriptura del mètode calculaSalari
    public float calculaSalari() {
        return super.calculaSalari() + (percentatge * totalHores);}
}
```

Test.java

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

      Empleat admin;
       admin = new Administratiu("Joana",1000, "dep");
       System.out.println("salari administratiu = " + admin. calculaSalari());

       Empleat empleat;
      // Preguntar a l'usuari que vol introduir a l'aplicació:
       Scanner sc=new Scanner(System.in);
       System.out.println("Indica 1 per introduir un professor i 2 per introduir un administratiu: ");
       int resposta = sc.nextInt();
      if(resposta==1){
          empleat = new Professor("Joana",1000, 10, 200);
       }else{
        empleat = new Administratiu("Joana",1000, "dep");
      }
       System.out.println("salari professor = " + empleat.calculaSalari());
   }
}
```

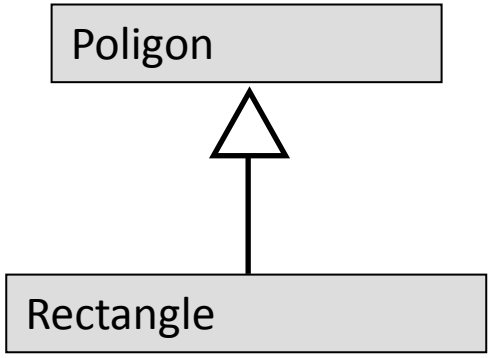
→ 1.500.0

Depenent de la resposta de l'usuari, sortirà: 1.500.0 o 3.500.0

Exemple 2

```
public class Poligon {
    public void imprimirIdentitat(){
        System.out.println("Sóc Poligon");
    }
}
```

```
public class Rectangle extends Poligon{
    @Override
    public void imprimirIdentitat(){
        System.out.println("Sóc Rectangle");
    }
}
```




```
public class Test {
    public static void main(String[] args){
        Poligon[] pol = new Poligon[2];

        Poligon elemA = new Poligon();
        Rectangle elemB = new Rectangle();

        pol[0] = elemA;
        pol[1]= elemB;

        pol[0].imprimirIdentitat();
        pol[1].imprimirIdentitat();
    }
}
```

Sortida per pantalla

```
Sóc Poligon
Sóc Rectangle
```

Polimorfisme vs. Sobrecàrrega

- És important diferenciar entre sobrecàrrega i el polimorfisme (sobreescriptura).

La sobrecàrrega consisteix a definir un mètode nou amb una signatura diferent (nombre i tipus de paràmetres). La sobrecàrrega es pot detectar en temps de compilació.	El polimorfisme és la substitució d'un mètode per un altre en una subclasse mantenint la signatura original. El polimorfisme es resol en temps d'execució.

Exemple 3

- És important diferenciar entre sobrecàrrega i el polimorfisme (sobreescriptura).

```
public class ExempleSobrecarrega{
   public void metodeExemple(){
      System.out.println("mètode sense
      parametres");
   }
 public void metodeExemple(int x){
      System.out.println("mètode amb els
      parametres" + x);
   }
}
```

```
public class ExemplePolimorfismeMare{
      public void metode(){
            System.out.println("mètode original");
      }
}
```

```
public class ExemplePolimorfisme extends
      ExemplePolimorfismeMare{
      public void metode(){
            System.out.println("mètode sobreescrit");
      }
}
```

Exemple 3

- Com has d'implementar el mètode per que aparegui per pantalla el missatge?:
  mètode original
  mètode sobreescrit

Solució:

```
public class ExemplePolimorfismeMare{
        public void metode(){
                System.out.println("mètode original");
        }
}
```

```
public class ExemplePolimorfisme extends
        ExemplePolimorfismeMare{
        public void metode(){
                super.metode();
                System.out.println("mètode sobreescrit");
        }
}
```

Exercicis

1. Genereu dues classes, A i B, amb els constructors per defecte (és a dir, sense cap argument), que tornin per pantalla algun missatge per saber quin és el constructor de cada una de les classes. Després, genereu una nova classe C que hereti de A, definiu un objecte de B dins de C i un constuctor que no faci res. En acabar, definiu un objecte de la classe C i mostreu els resultats.

Exercicis

2. Modifiqueu l'exercici anterior per tal que les classes A i B tinguin un constructor amb arguments. Escriviu un constructor per a la classe C i executeu totes les inicialitzacions necessàries dins el constructor de C.

Exercicis

3. Genereu una classe anomenada Root que contingui una instància de cada una de les tres classes `Comp1`, `Comp2` i `Comp3`. Deriveu una classe nova, `Node` a partir de la classe `Root` que contingui una instància de les tres classes "component". En el constructor de totes les classes, col·loqueu un missatge de manera que retorni per pantalla un identificador de la classe on és i mireu el resultat.

Exercicis

4. Genereu una classe que tingui un mètode sobrecarregat. Genereu una classe A que tingui un mètode per a tornar per pantalla un paràmetre que, si és enter, n'hi sumi 1; si és real, n'hi sumi 3,5, i si és una cadena de caràcters, hi afegeixi un guió davant i un altre al darrere.

Exercicis

5. Creeu tres classes, A, B i C, de manera que formin una jerarquia de classes. La classe A és la superclasse i les classes B i C hereten de la classe A. Implementeu un mètode en la classe A anomenat toString que no rebi paràmetres i que retorni una cadena que indiqui que és un mètode de la classe A i, posteriorment, sobreescriviu aquest mètode en les classes B i C. Creeu una instància de cada classe i executeu aquest mètode.

LLIGADURES

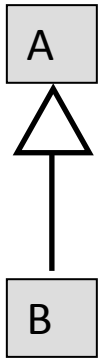
Tipus de lligadures: estàtic i dinàmic

- Donada una assignació polimorfa
- Exemple:

  Una variable de la classe A és una referència a un objecte de la classe B:

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

- Llavors, es diu que:
  - A és el ***tipus estàtic*** de la variable **`a`** i
  - B es el ***tipus dinàmic*** de **`a`**.
- El tipus estàtic sempre es determina en temps de compilació i és fix, mentre que el tipus dinàmic només es pot conèixer en temps d'execució i pot variar.



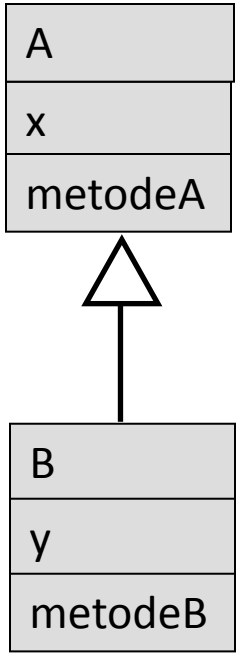
Tipus de lligadures: estàtic i dinàmic

- Java només permet invocar els mètodes i accedir a les variables conegudes per al **tipus estàtic** de a.

```
A a = new B();
a.metodeA();   // Ok
a.metodeB();   // error de compilació
               // metodeB no està definit per a A
```

accés:

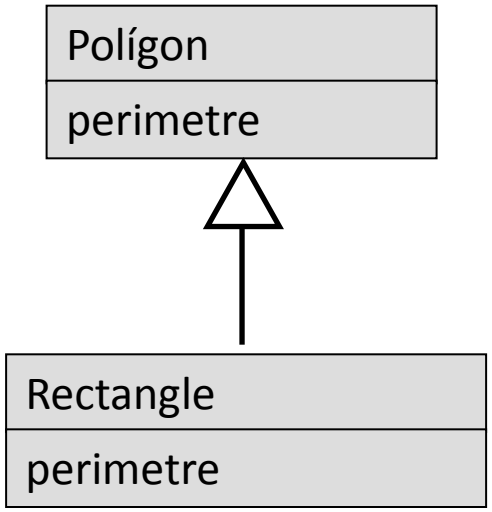
```
a.x;           // Ok
a.y;           // error de compilació
```



Lligadura dinàmica

En POO quan realitzem una connexió polimorfa i cridem a una operació redefinida

```
// Pot referenciar a un objecte Polígon o Rectangle

Poligon poligon;
float peri;
Rectangle rectangle = new Rectangle();
poligon = rectangle;
peri = poligon.perimetre();
```



El compilador no té informació per a resoldre la crida.
Per defecte utilitzaria el tipus de la referència, i per tant generaria una crida a Poligon.perimetre()
Però la referència poligon pot apuntar a un objecte de la classe Rectangle amb una versió diferent del mètode

Lligadura dinàmica

- La solució consisteix en esperar a resoldre la crida en temps d'execució, quan es coneix realment els objectes connectats a **poligon**, i quina és la versió del mètode **perimetre** apropiada.
- Aquest enfocament de resolució de crides s'anomena **lligadura dinàmica**

- Entenem per **resolució d'una crida** el procés pel qual es substituirà una crida a una funció per un salt a la direcció que conté el codi d'aquesta funció.

Exemple

```
public class Poligon {
    public void imprimirIdentitat(){
        System.out.println("Sóc Poligon");
    }
}
```

```
public class Rectangle extends Poligon{
    @Override
    public void imprimirIdentitat(){
        System.out.println("Sóc Rectangle");
    }
}
```

```
public class Test {
    public static void main(String[] args){
        Poligon[] pol = new Poligon[2];

        Poligon elemA = new Poligon();
        Rectangle elemB = new Rectangle();

        pol[0] = elemA;
        pol[1]= elemB;

        pol[0].imprimirIdentitat();
        pol[1].imprimirIdentitat();
    }
}
```

Sortida per pantalla →

```
Sóc Poligon
Sóc Rectangle
```

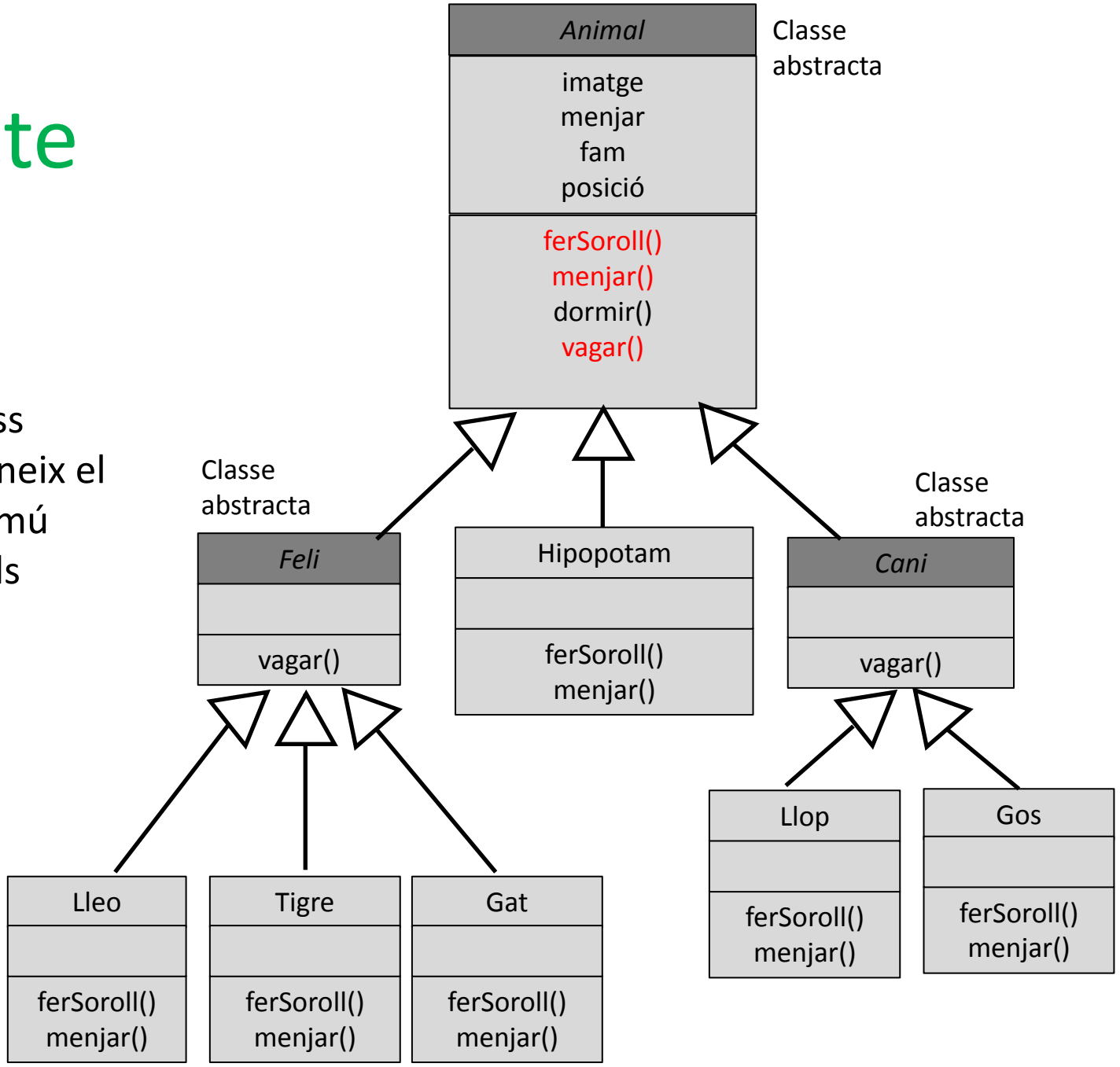
INTERFÍCIES

Introducció

- Introducció d'interfícies amb un exemple
- Construïm la jerarquia d'herències de la classe Animal.

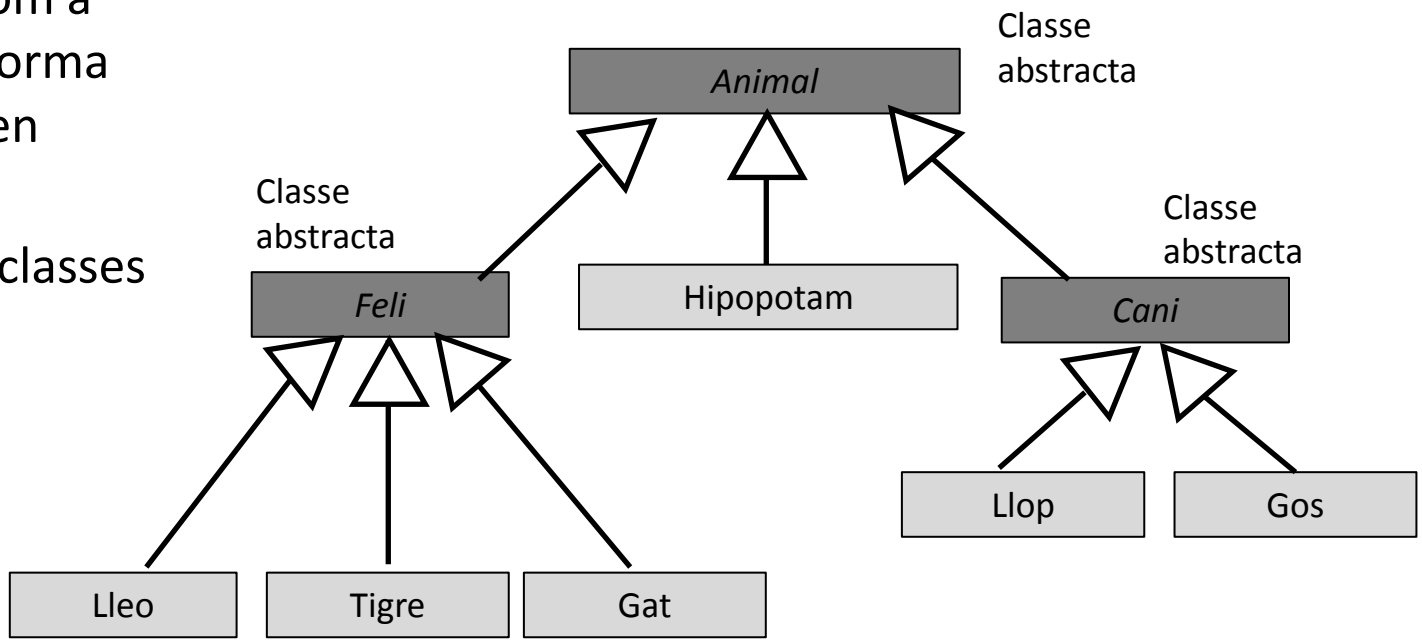
Contracte

- Comencem definint un contracte:
  - La superclass Animal defineix el protocol comú per a tots els animals.



Contracte

- A més, definim algunes de les superclasses com a abstractes de forma que no es poden instanciar.
- La resta de les classes s'anomenen concretes.



Array polimòrfic

```
public class LlistaAnimals {
   private Animal[] animals = new Animal[5];
   private int nextIndex=0;

   public void add(Animal a){
      if (nextIndex < animals.length){
         animals[nextIndex] = a;
         System.out.println("Animal afegit a la posició " + nextIndex);
         nextIndex++;
      }
   }
}
```

LlistaAnimals.java

Array polimòrfic

```
public class TestLlistaAnimal {
    public static void main(String[] args){
        LlistaAnimals llista = new LlistaAnimals();
        Gos gos = new Gos();
        Gat gat = new Gat();
        llista.add(gos);
        llista.add(gat);
    }
}
```

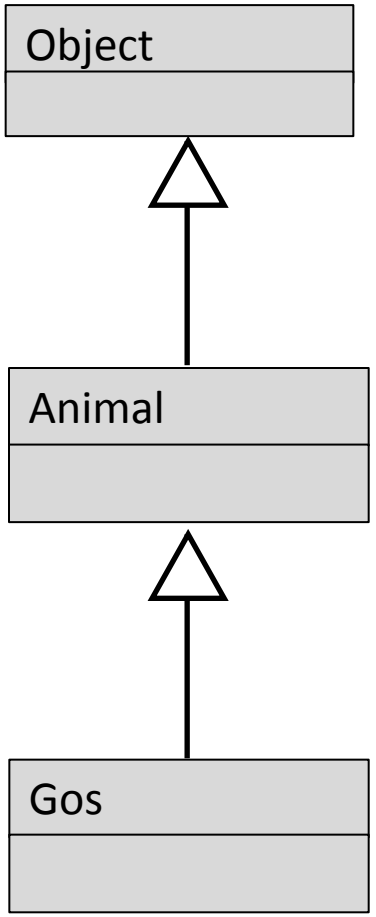
TestLlistaAnimals.java

Estem afegint tot tipus d'animals a l'array

Animal afegit a la posició 0
Animal afegit a la posició 1

Llista polimòrfica

- També es podria optar per fer servir la classe Object que és encara més genèrica i referenciar a qualsevol tipus d'objectes.
- Però això porta alguns inconvenients!!!



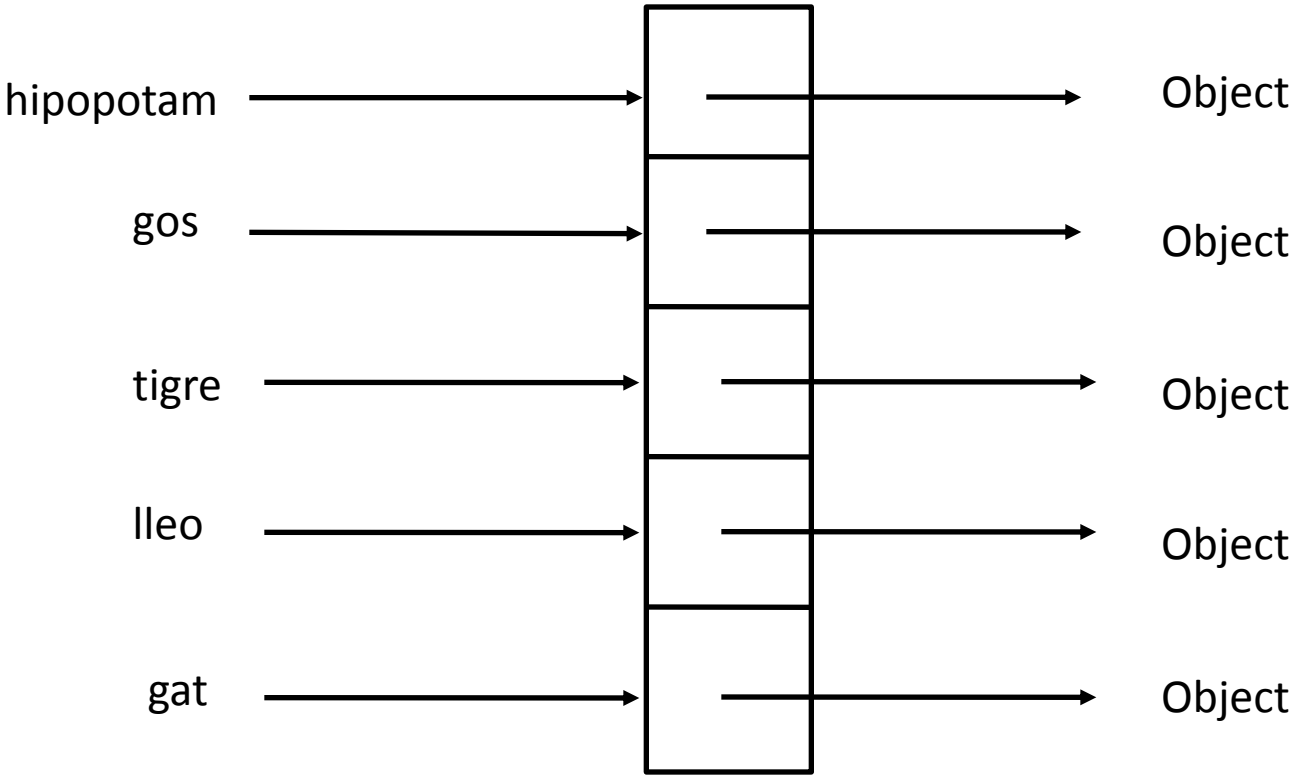
Llista polimòrfica

```
ArrayList laLlistaAnimals = new ArrayList();
```

Llista per contenir tot tipus d'Objectes.

```
Gos gos = laLlistaAnimals.get(0);
```

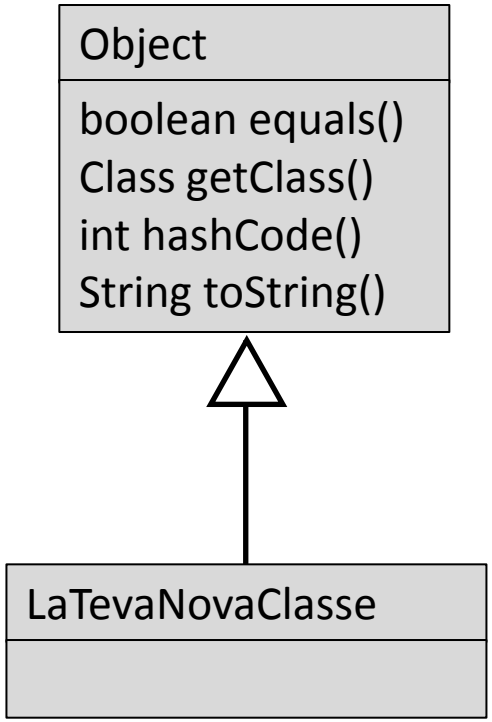
No compilarà!



Posis el que posis en cada posició quan recuperis els objectes aquests seran de tipus Object.

Classe Object

- Qualsevol classe implementada per tu hereta de la classe Object.



1. equals(Object o)

```
Dog a = new Dog();
Cat c = new Cat();
if (a.equals(c)) {
     System.out.println("true");
} else {
    System.out.println("false");
}
```

```
% java TestObject
false
```

2. getClass()

```
Cat c = new Cat();
System.out.println(c.getClass());
```

```
% java TestObject
class Cat
```

3. hashCode()

```
Cat c = new Cat();
System.out.println(c.hashCode());
```

```
% java TestObject
8202111
```

4. toString()

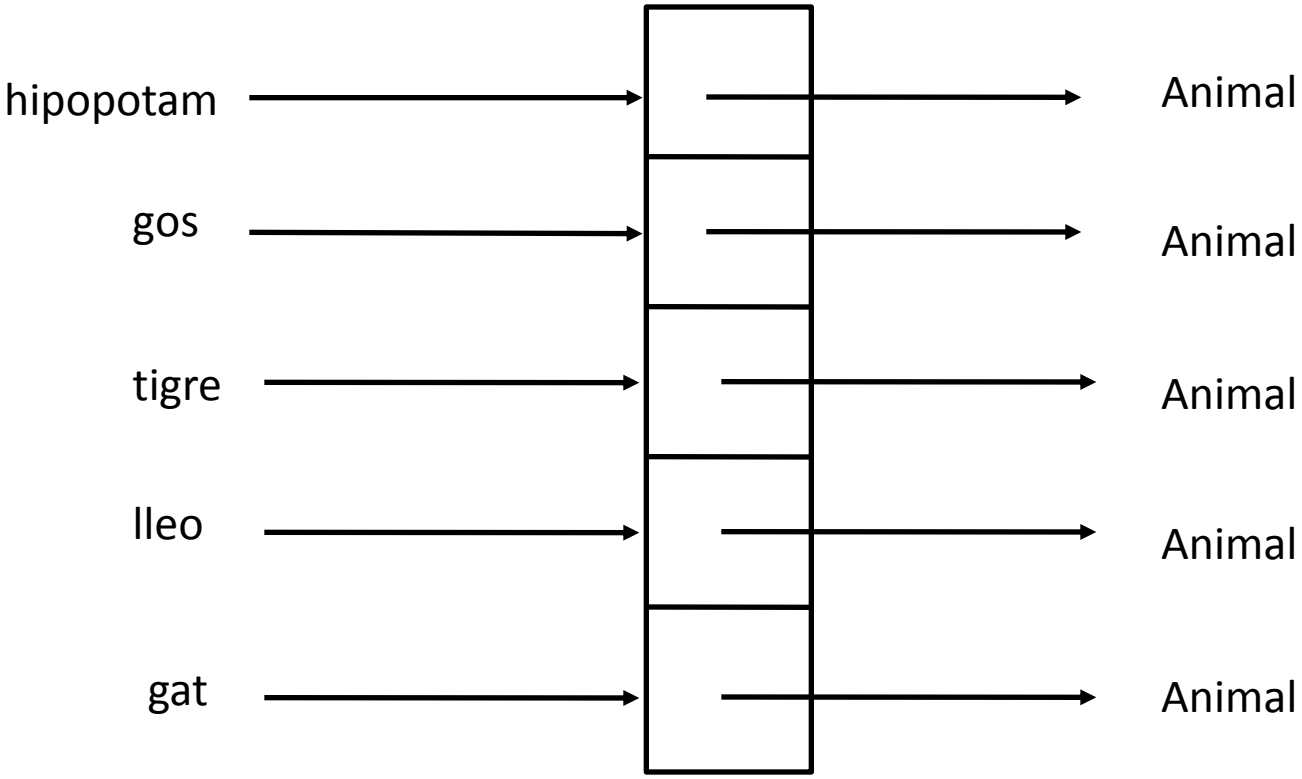
```
Cat c = new Cat();
System.out.println(c.toString());
```

```
% java TestObject
Cat@7d277f
```

Array polimòrfic

```
ArrayList<Animal> laLlistaAnimals = new ArrayList<Animal>();
```

Quan pot ser útil?



Exemple

```
package paquetInterficies;
import java.util.ArrayList;

public abstract class Animal {
    public abstract void ferSoroll();
}
```

Animal.java

Exemple

```
package paquetInterficies;
import java.util.ArrayList;

public class Gat extends Animal{
 public void ferSoroll(){
       System.out.println("miau");
   }
}
```

Gat.java

```
package paquetInterficies;
import java.util.ArrayList;

public class Gos extends Animal{
   public void ferSoroll(){
       System.out.println("guau");
   }
}
```

Gos.java

Exemple

```
package paquetInterficies;
import java.util.ArrayList;

public class TestAnimals {
   public static void main(String[] args){
      ArrayList<Animal> arrayAnimals = new ArrayList<Animal>();

      Gos gos = new Gos();
      Gat gat = new Gat();
      arrayAnimals.add(gos);
      arrayAnimals.add(gat);
      arrayAnimals.get(0).ferSoroll();
      arrayAnimals.get(1).ferSoroll();
   }
}
```

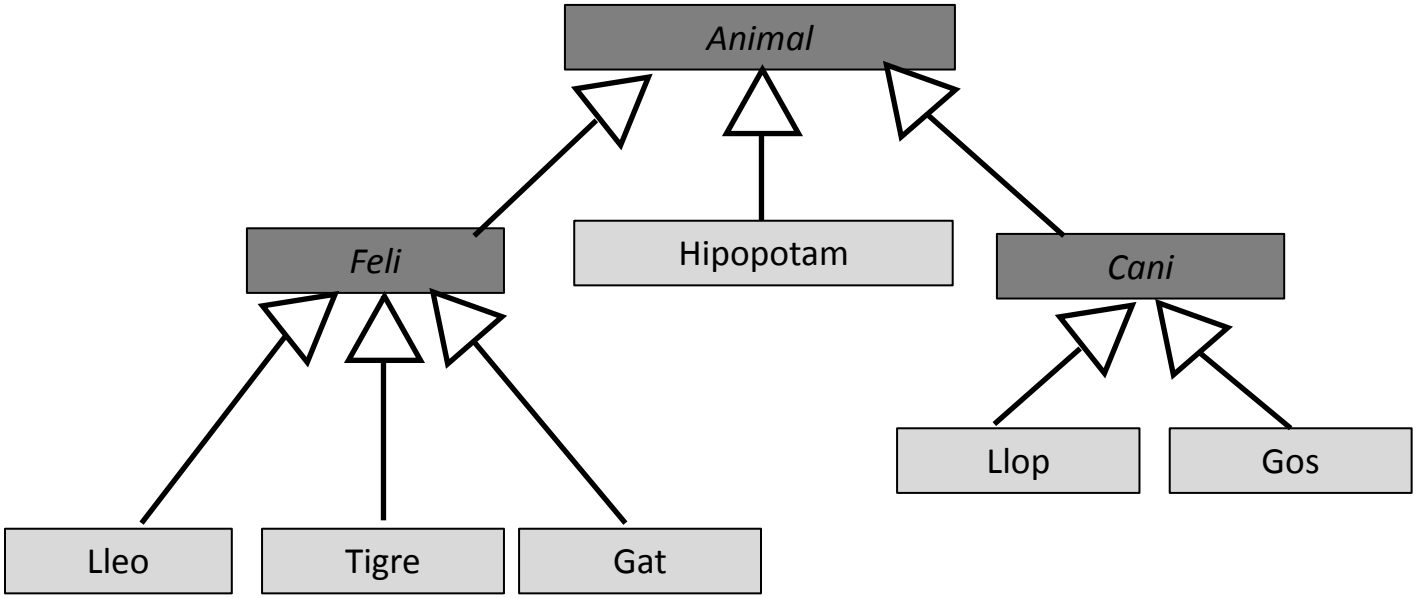
ferSoroll és un mètode polimòrfic

Sortida per pantalla:
guau
miau

TestAnimal.java

Possibles dissenys:
Volem afegir els comportaments de les mascotes

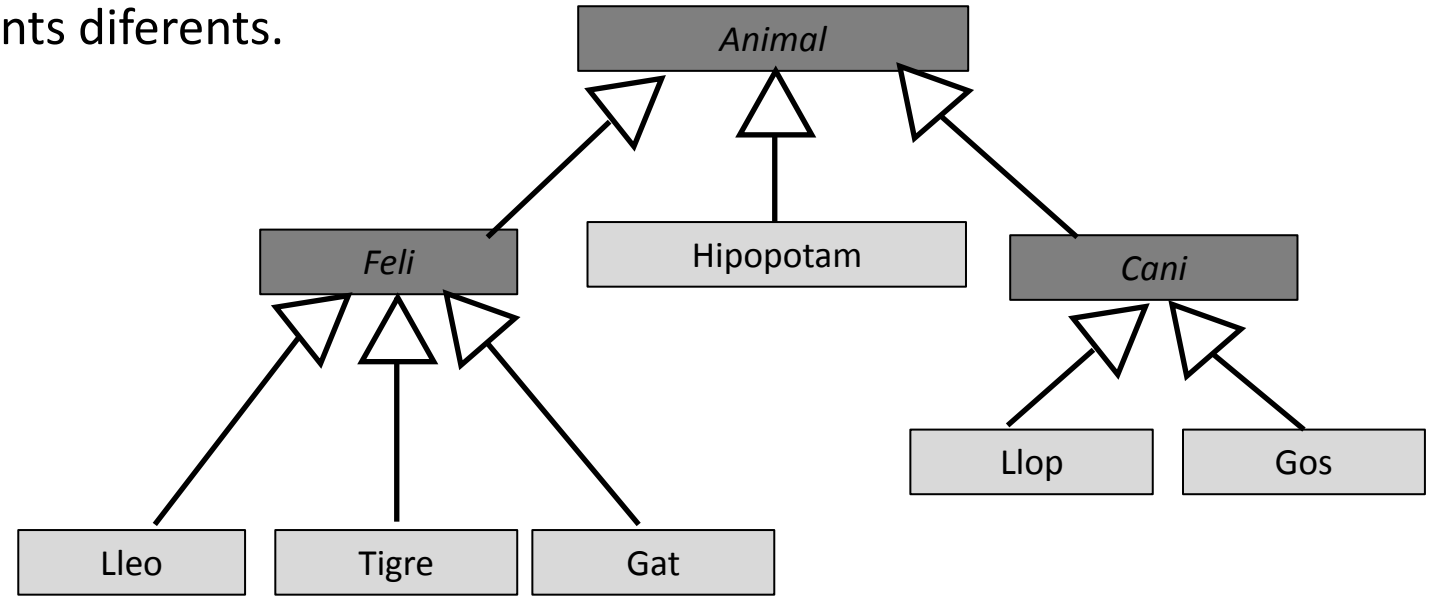
- Veiem diferents opcions de disseny per reutilitzar algunes de les classes existents en un programa d'una tenda de **mascotes**.



Opció 1

- Posem els mètodes de mascota en la classe Animal.

  Pros: No modifiquem les classes existents i les noves classes que afegim heretaran aquests mètodes.
  Contres: Un Hipopòtam no és una mascota!
  A més, un gat i un gos tenen comportaments diferents.

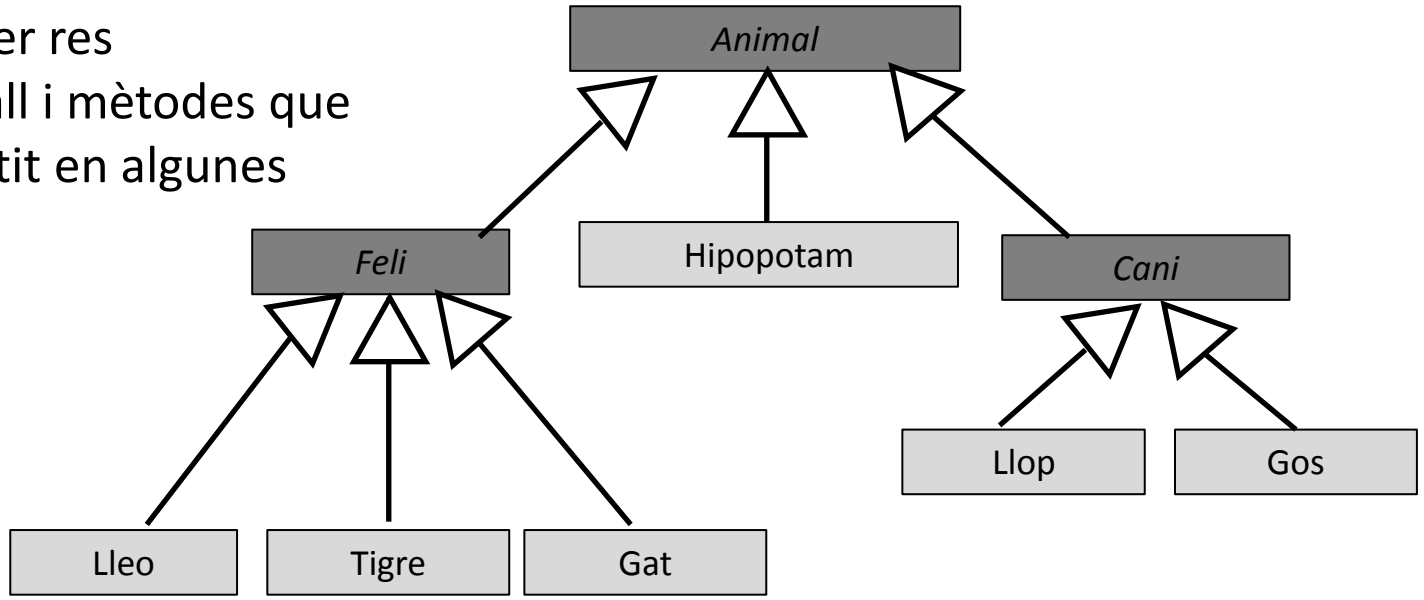


Opció 2

- Posem els mètodes de mascota en la classe Animal, però fem els **mètodes abstractes** forçant les subclasses de Animal a sobreescrire'ls.

Pros: Els mateixos que l'opció 1, però a més podem definir no-mascotes. Com? Fent que les implementacions no facin res.

Contres: S'han d'implementar tots els els mètodes abstractes de la classe Animal encara que sigui per no fer res
→ molt treball i mètodes que no tenen sentit en algunes classes.

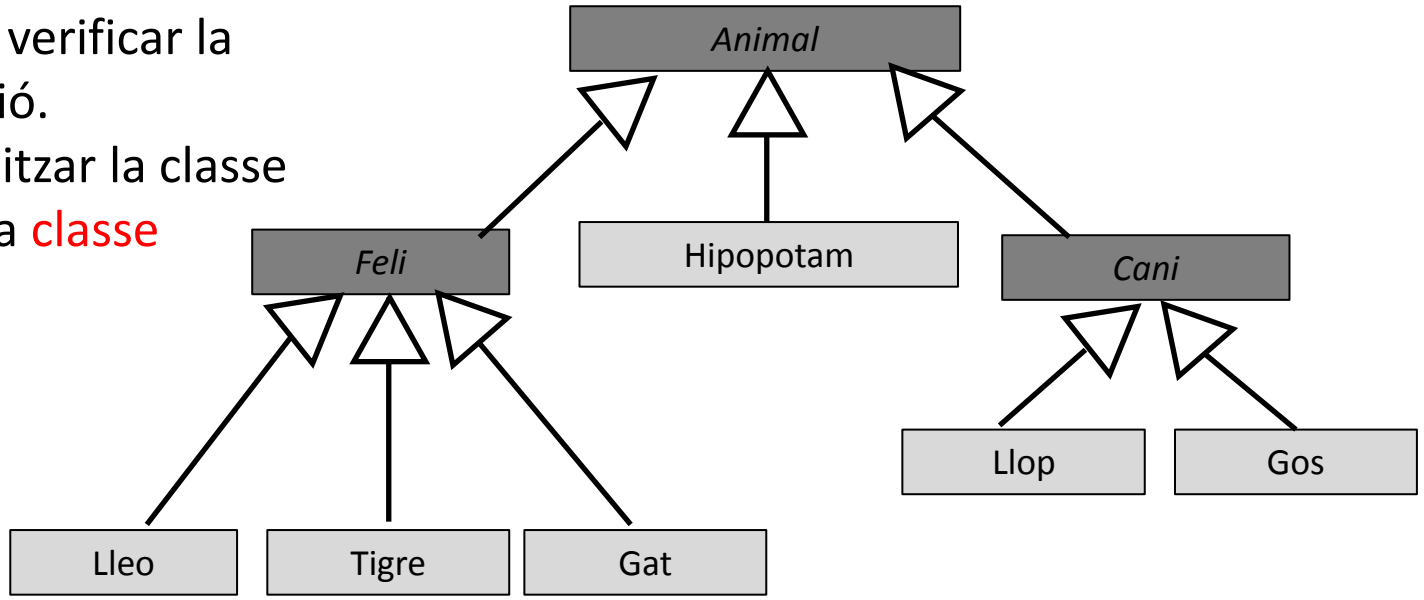


En aquest cas, només hauríem de posar dins de la classe Animal, els mètodes que s'apliquen a totes les seves subclasses.

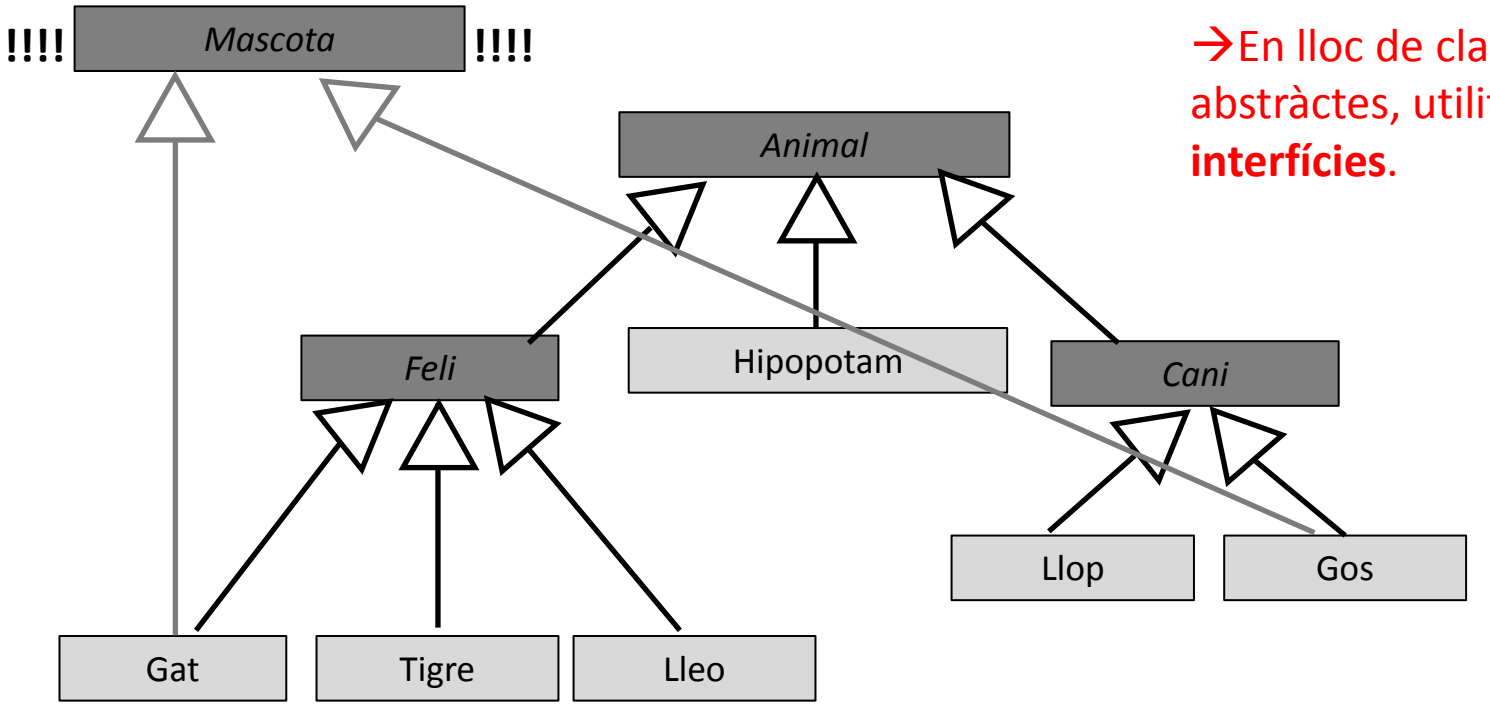
Opció 3

- Posem els mètodes de mascota només en les classes que ho són.

Pros: Desapareixen els hipopòtams com a mascotes i els mètodes estan on toca.
Contres: Tots els programadors hauran de conèixer el protocol.
No hi ha contracte que obliga el compilador a verificar la implementació.
No es pot utilitzar la classe Animal com la classe polimòrfica.



Necessitem dues superclasses



→En lloc de classes abstràctes, utilitzarem **interfícies**.

→ Herència múltiple

Interfícies

- Una interfície és un conjunt de **declaracions de mètodes** (sense definició)
- Una interfície també pot definir **constants** que són implícitament *public*, *static* i *final*, i sempre s'han d'inicialitzar en la declaració
- Totes les classes que implementen una interfície estan obligades a proporcionar una definició als mètodes de la interfície
- Una interfície defineix el protocol d'implementació d'una classe

Interfícies

- Una classe pot implementar més d'una interfície

→representa una alternativa a l'herència múltiple en Java.

- La paraula clau és:

implements + el nom de la interfície

interface *nom_interficie* {
  tipus_retorn nom_metode (*llista_arg*) ;
  . . .
}

class *nom_classe* **implements** *nom_interficie* {
  tipus_retorn nom_metode (*llista_arg*) {
    <codi>
  }
}

Implementació

```
public interface Mascota {
    public abstract void serAmigable();
    public abstract void jugar();
}
```

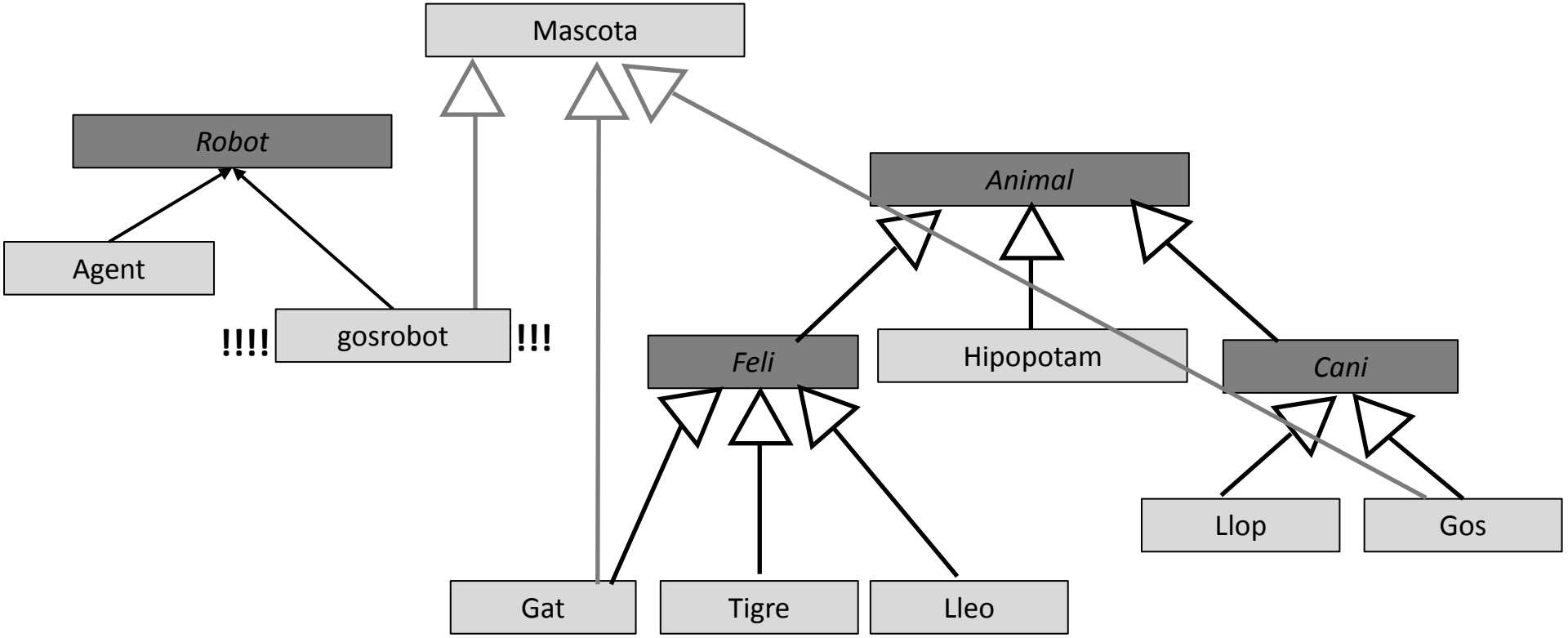
Mascota.java

Exemple

```
public class Gos extends Animal implements Mascota{
   public void ferSoroll(){
      System.out.println("guau");
   }
   public void serAmigable() {
      System.out.println("fa gràcies");
   }
   public void jugar() {
     System.out.println("juga");
   }
}
```

Gos.java

Classes de diferents arbres d'herència poden implementar la mateixa interfície



Interfície

Quan utilitzar una interfície en lloc d'una classe abstracta?

- Per la seva senzillesa es recomana utilitzar interfícies sempre que sigui possible.
- Si la classe ha d'incorporar atributs, o resulta interessant la implementació d'alguna de les seves operacions, llavors declarar-la com a classe abstracta.
- Dins la biblioteca de classes de **Java** es fa un ús intensiu de les interfícies per a caracteritzar les classes.

- Alguns exemples:
  - Per a que un objecte pugui ser guardat en un fitxer, la seva classe ha d'implementar la interfície *Serializable,*
  - Per a que un objecte sigui duplicable, la seva classe ha d'implementar la interfície *Cloneable,*
  - Per a que un objecte sigui ordenable, la seva classe ha d'implementar la interfície *Comparable.*

Extensió d'interfícies

- Les interfícies poden extendre altres interfícies
- La sintaxis es:

interface *nom_NovaInterficie* **extends** *nom_interficie* , . . . {
    tipus_retorn nom_metode (*llista_arguments*) ;
    . . .
}

Exemple: Interfícies

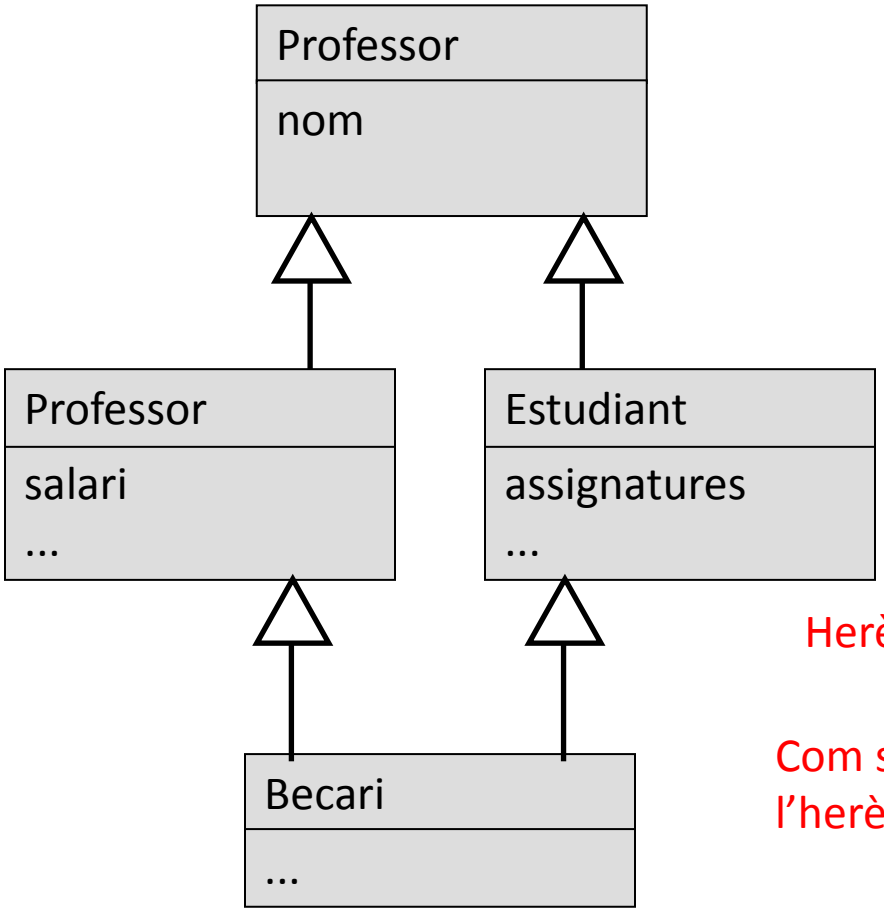
```
public interface VideoClip {
  // comença la reproducció del video
  void play();
  // reprodueix el clip en un bucle
  void bucle();
  // para la reproducció
  void stop();
}

//I una classe que implementa la interfície:
class LaClasse implements VideoClip {
 void play() { <codi> }
 void bucle(){ <codi> }
 void stop() { <codi> }
}
```

Exemple: Interfícies

```
public interface VideoClip {
  // comença la reproducció del video
  void play();
  // reprodueix el clip en un bucle
  void bucle();
  // para la reproducció
  void stop();
}

//I una altra classe que també implementa la interfície:
Class LaAltraClasse implements VideoClip {
      void play()  { <codi nou> }
      void bucle() { <codi nou > }
      void stop()  { <codi nou > }
}
```

Interfície per herència múltiple

- Un exemple un poc més complex:
- Si volem implementar el següent disseny:



Herència múltiple

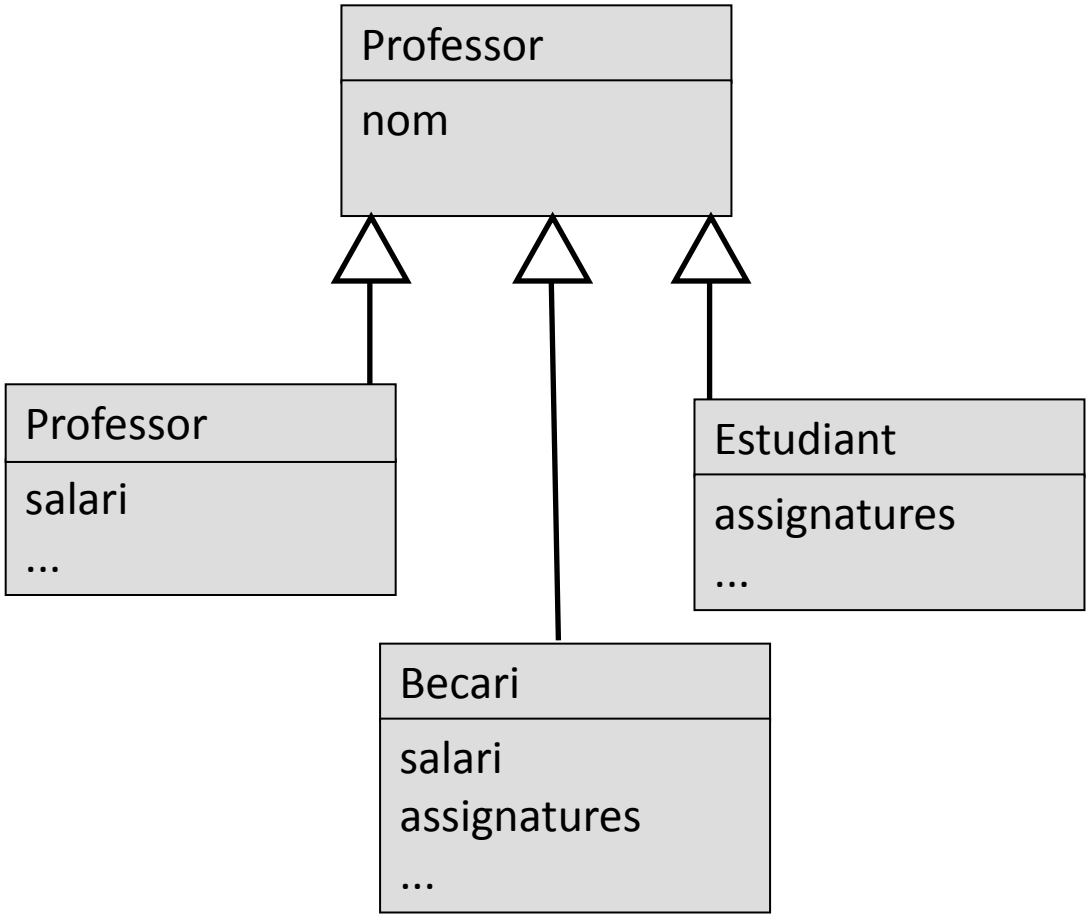
Com solucionem el problema de l'herència múltiple?

Observacions

- O simplifiquem el disseny o utilitzem interfícies per solucionar aquest problema
- Solució Standard:
  - Una classe per heretar
  - Una interfície per implementar
- Fent servir interfícies, hi ha diverses opcions d'implementació

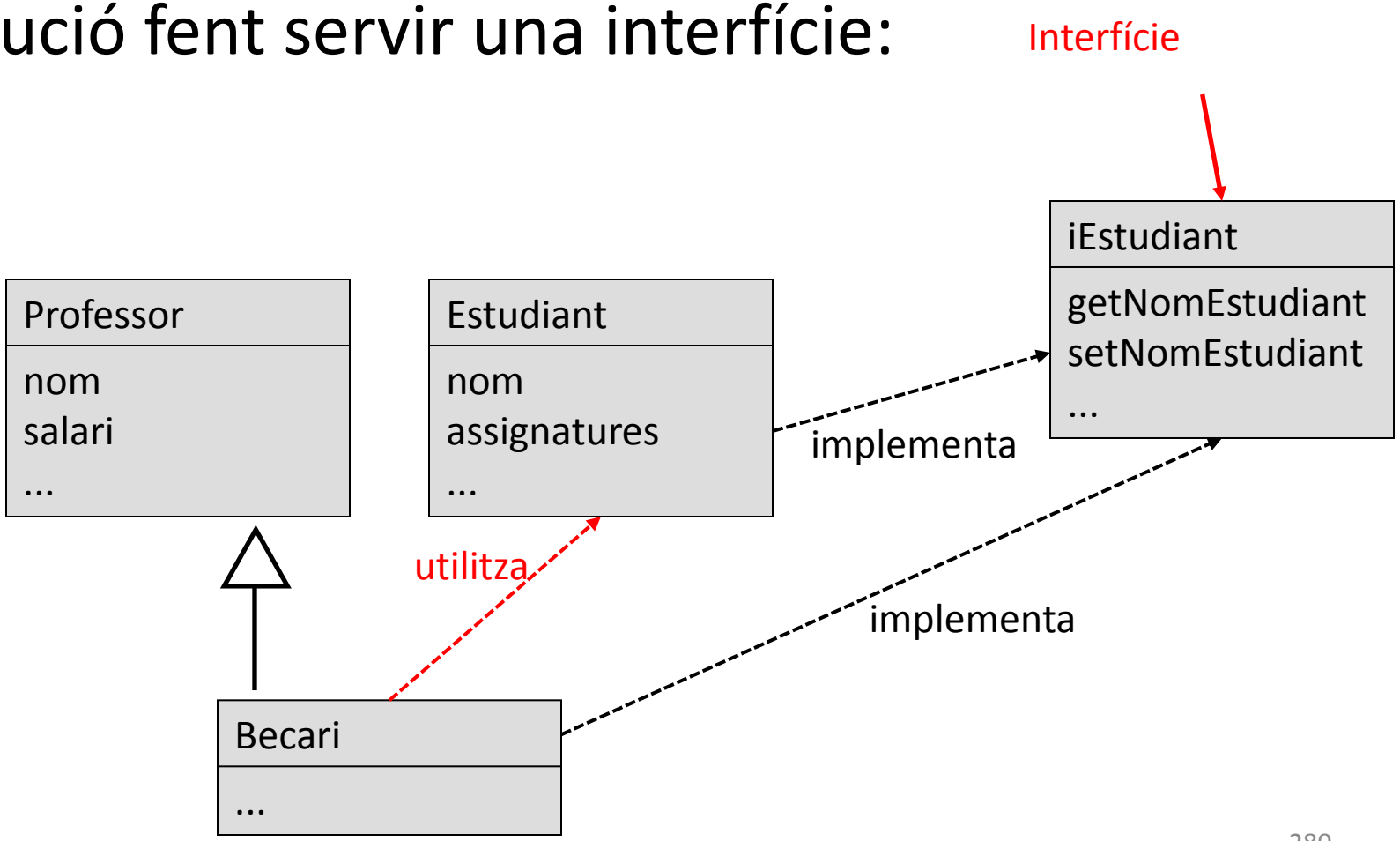
Interfície per herència múltiple

- Solució fent servir un nou disseny:



Interfície per herència múltiple

- Solució fent servir una interfície:



Solució 1: Exemple Interfícies

```
public class Professor{
	private String nom;
	private int salari;
	public Professor(String pNom, int pSalari) {
		nom = pNom;
		salari = pSalari;
	}
	public String getNom() {
		return nom;
	}
	public int getSalari() {
		return salari;
	}
}
```

Professor.java

Solució 1: Exemple Interfícies

```
public interface IEstudiant {

    public String getNomEstudiant ();
    public void setNomEstudiant (String pNom);

}
```

IEstudiant.java

Solució 1: Exemple Interfícies

```
public class Estudiant implements IEstudiant {

    private String nom;
    public Estudiant(String pNom) {
        nom = pNom;
    }
    public String getNomEstudiant () {
        return nom;
    }
    public void setNomEstudiant (String nom) {
        this.nom = nom;
    }
}
```

Estudiant.java

Solució 1: Exemple Interfícies

```
public class Becari extends Professor implements IEstudiant {


    private Estudiant estudiant;

    public Becari(String nom, int salari) {
        super(nom, salari);
        estudiant = new Estudiant(nom);
    }
    public String getNomEstudiant() {
        return estudiant.getNomEstudiant();
    }
    public void setNomEstudiant(String nom) {
        estudiant.setNomEstudiant(nom);
    }
}
```

Becari.java

Defineix un objecte de la classe Estudiant

Observacions

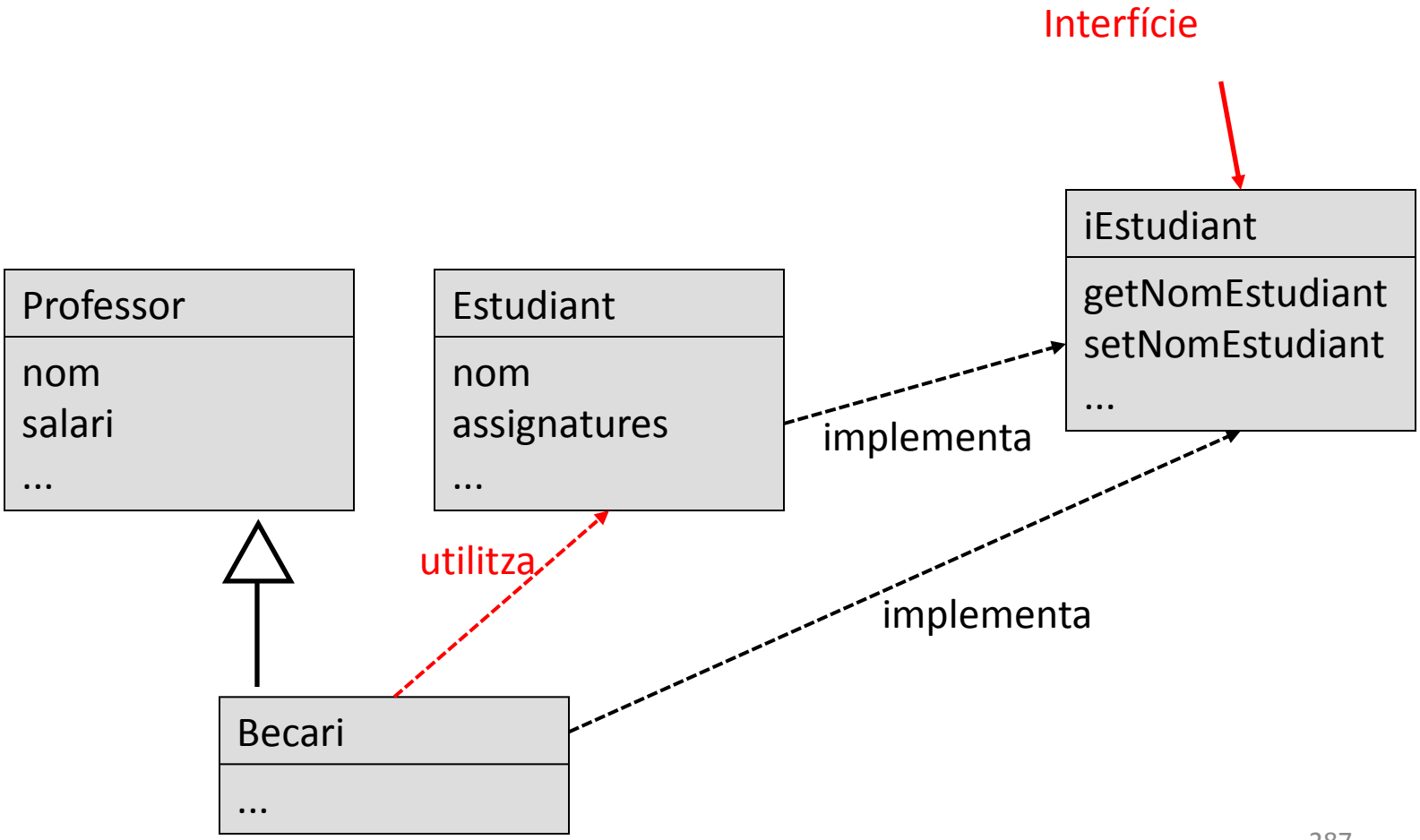
- Problema d'aquesta implementació:
  - Si canviem el nom de l'estudiant, el nom del professor no canvia

Solució 1: Exemple Interfícies

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

       Becari becari;
       becari = new Becari("Joan",1000);
       System.out.println("salari becari = " + becari.getSalari());

       System.out.println("nom del professor = " + becari.getNom());
       System.out.println("nom de l'estudiant = " + becari.getNomEstudiant());

       becari.setNomEstudiant("Joan Francesc");

       System.out.println("nom del professor = " + becari.getNom());
       System.out.println("nom de l'estudiant = " + becari.getNomEstudiant());

   }
}
```

Solució 2: Exemple Interfícies

- Solució fent servir una interfície:



Solució 2: Exemple Interfícies

```
public class Professor{
        private String nom;
        private int salari;
        public Professor(String pNom, int pSalari) {
                nom = pNom;
                salari = pSalari;
        }
        public String getNom() {
                return nom;
        }
        public int getSalari() {
                return salari;
        }
        public void setNom(String nom) {
          this.nom=nom;
        }
        public void setSalari(int salari) {
          this.salari=salari;
        }
}
```

Professor.java

Solució 2: Exemple Interfícies

```
public interface IEstudiant {

    public String getNomEstudiant ();
    public void setNomEstudiant (String pNom);
    public String getAssignatures();
    public void setAssignatures(String assignatures);

}
```

IEstudiant.java

Solució 2: Exemple Interfícies

```
public class Estudiant implements IEstudiant {

        private String nom;
        private String assignatures;
        public Estudiant(String pNom) {
                nom = pNom;
        }
        public String getNomEstudiant () {
                return nom;
        }
        public void setNomEstudiant (String nom) {
                this.nom = nom;
        }
        public String getAssignatures () {
                return assignatures;
        }
        public void setAssignatures (String assignatures) {
                this.assignatures = assignatures;
        }
}
```

Estudiant.java

Solució 2: Exemple Interfícies

```
public class Becari extends Professor implements IEstudiant {
      private Estudiant estudiant;
       public Becari(String nom, int salari) {
          super(nom, salari);
          estudiant = new Estudiant(nom);
       }
       public String getNomEstudiant() {
              return super.getNom();
       }
       public void setNomEstudiant(String nom) {
              super.setNom(nom);
       }
      public String getAssignatures() {
        return estudiant.getAssignatures();
      }
      public void setAssignatures(String assignatures) {
         estudiant.setAssignatures(assignatures);
      }
}
```

Becari.java

Solució 2: Exemple Interfícies

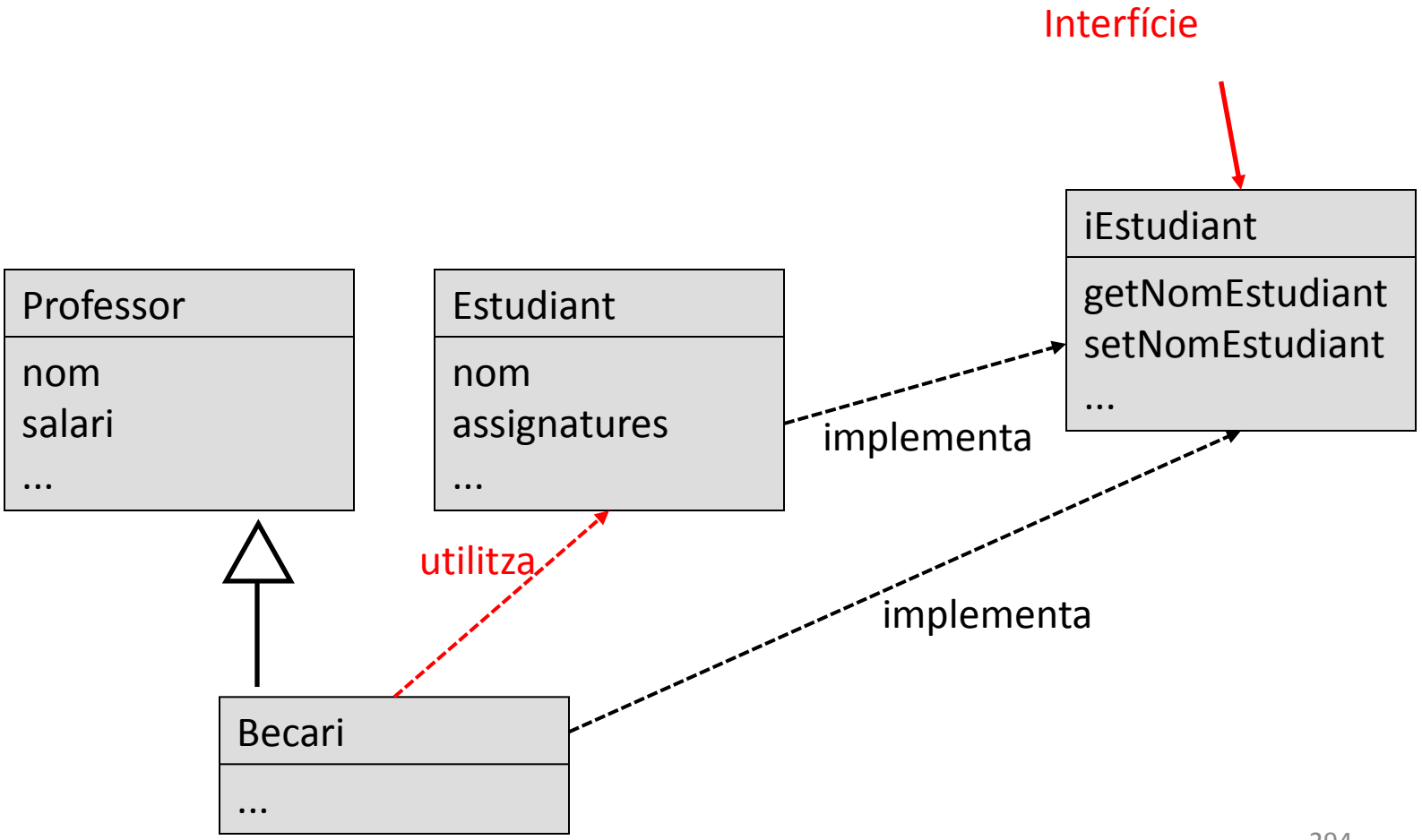
```
public class TestBecari {
 public static void main(String[] args) {

       Becari becari;
       becari = new Becari("Joan",1000);
       System.out.println("salari becari = " + becari.getSalari());

       System.out.println("nom del professor = " + becari.getNom());
       System.out.println("nom de l'estudiant = " + becari.getNomEstudiant());

       becari.setNomEstudiant("Joan Francesc");

       System.out.println("nom del professor = " + becari.getNom());
       System.out.println("nom de l'estudiant = " + becari.getNomEstudiant());

   }
}
```

Observacions

- Problema d'aquesta implementació:
  - L'objecte becari té dos mètodes per accedir al nom un és getNom() i l'altre getNomEstudiant()
- Hi ha una altra opció de disseny per evitar el problema de l'herència múltiple?

Solució 3: Exemple Interfícies

- Solució fent servir una interfície:



Solució 3: Exemple Interfícies

```
public class Professor{
        private String nom;
       private int salari;
       public Professor(String pNom, int pSalari) {
                nom = pNom;
                salari = pSalari;
       }
       public String getNom() {
                return nom;
       }
       public int getSalari() {
                return salari;
       }
      public void setNom(String nom) {
         this.nom=nom;
       }
       public void setSalari(int salari) {
         this.salari=salari;
       }
}
```

Professor.java

Solució 3: Exemple Interfícies

```
public interface IEstudiant {

      public String getNom();
      public void setNom(String pNom);
       public String getAssignatures();
      public void setAssignatures(String assignatures);
}
}
```

IEstudiant.java

Solució 3: Exemple Interfícies

```
public class Estudiant implements IEstudiant {

        private String nom;
        private String assignatures;

        public Estudiant(String pNom) {
                nom = pNom;
        }
        public String getNom() {
                return nom;
        }
        public void setNom(String nom) {
                this.nom = nom;
        }
        public String getAssignatures () {
                return assignatures;
        }
        public void setAssignatures (String assignatures) {
                this.assignatures = assignatures;
        }
}
```

Estudiant.java

Solució 3: Exemple Interfícies

```
public class Becari extends Professor implements IEstudiant {

    private Estudiant estudiant;

     public Becari(String nom, int salari) {
        super(nom, salari);
        estudiant = new Estudiant(nom);
     }

     public String getAssignatures() {
       return estudiant.getAssignatures();
    }

    public void setAssignatures(String assignatures) {
        estudiant.setAssignatures(assignatures);
    }
```

Becari.java

Solució 3: Exemple Interfícies

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

       Becari becari;
       becari = new Becari("Joan",1000);
       System.out.println("salari becari = " + becari.getSalari());

       System.out.println("nom del professor = " + becari.getNom());
       System.out.println("nom de l'estudiant = " + becari.getNomEstudiant());

       becari.setNomEstudiant("Joan Francesc");

       System.out.println("nom del professor = " + becari.getNom());
       System.out.println("nom de l'estudiant = " + becari.getNomEstudiant());

   }
}
```

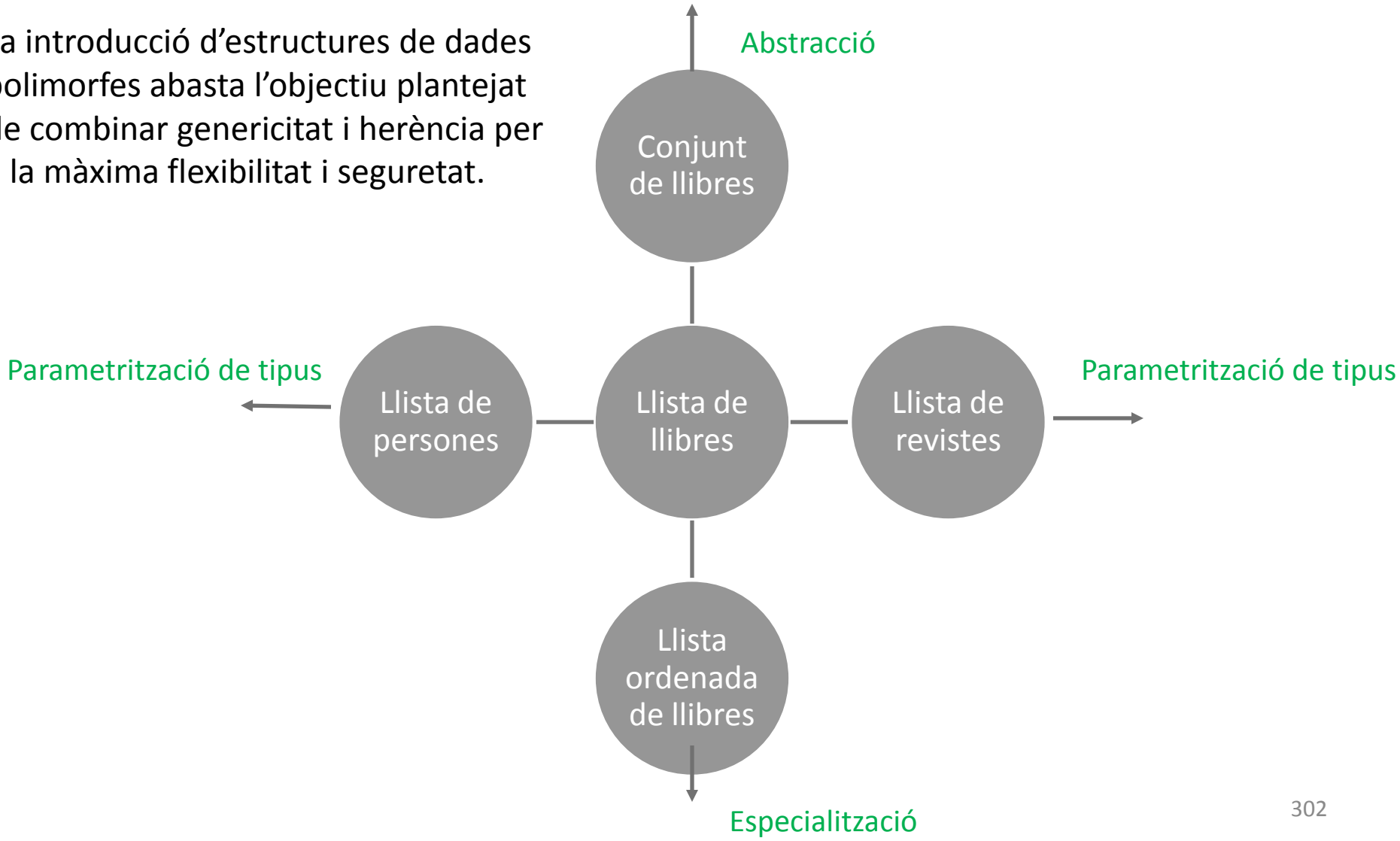
Observacions

- En aquest cas, no cal la sobreescritura dels mètodes getNom i setNom a la classe Becari.

COL·LECCIONS: INTRODUCCIÓ A UN EXEMPLE PRÀCTIC

Generalització (Recordatori)

La introducció d'estructures de dades polimorfes abasta l'objectiu plantejat de combinar genericitat i herència per a la màxima flexibilitat i seguretat.



Característiques bàsiques de la POO (Recordatori)

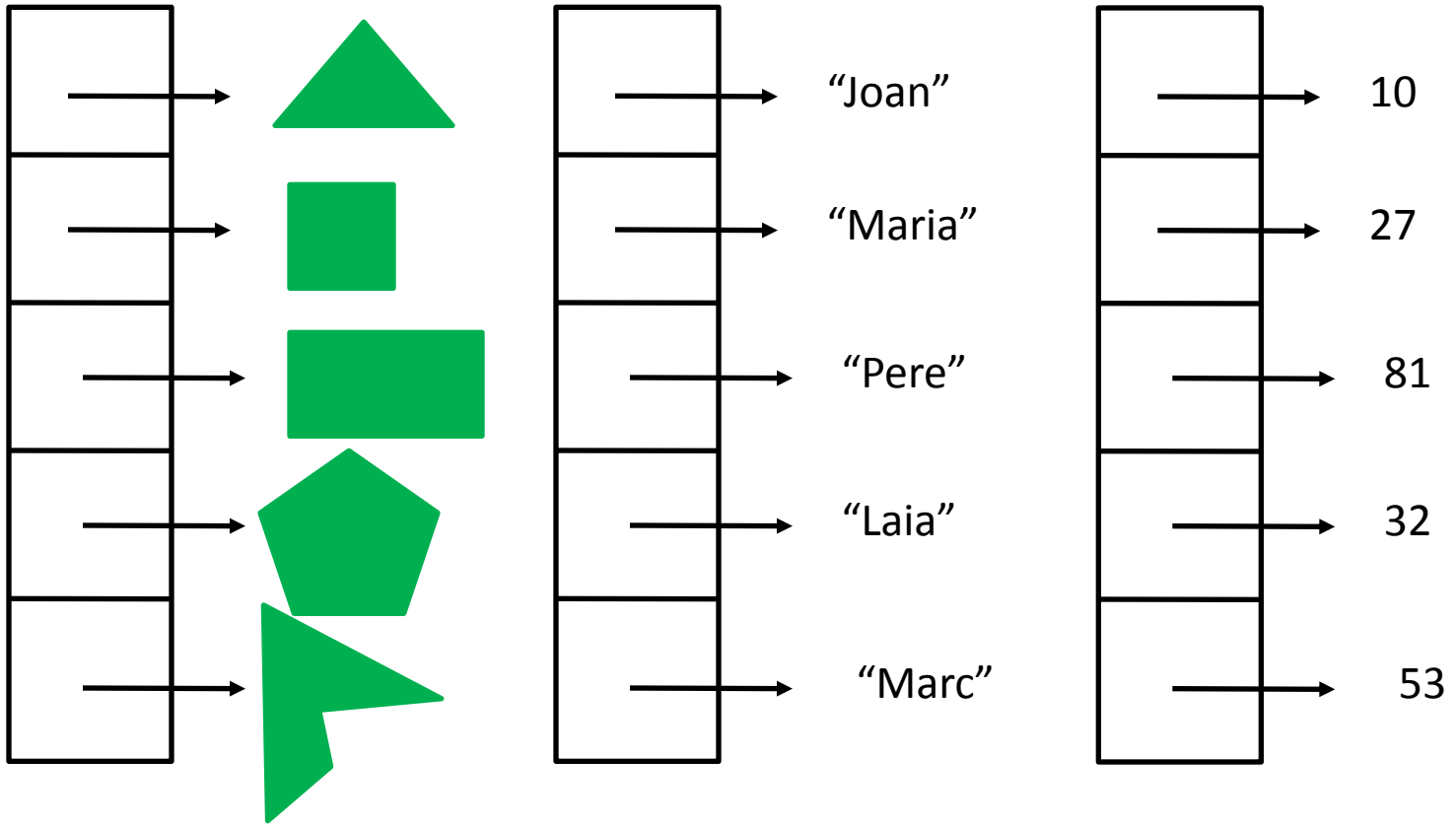
- **Genericitat** :

  És la propietat que permet definir mètodes que tenen com a paràmetres elements de qualsevol tipus.

- Exemple:
  - Si volem definir un objecte que sigui un vector o una llista que pugui rebre elements de qualsevol tipus.
  - Això té sentit perquè el comportament d'aquest objecte sempre és el mateix (afegir, esborrar, inserir, etc.) independentment del tipus contingut.
  - D'aquesta manera, el podríem usar una vegada com a vector d'enters, una altra com a vector de caràcters, etc.
- La genericitat és bàsica per a reutilitzar codi.

Estructures de dades

- Estructures de dades polimorfes: que contenen objectes de tipus diferents (**tots descendents d'un tipus comú**).

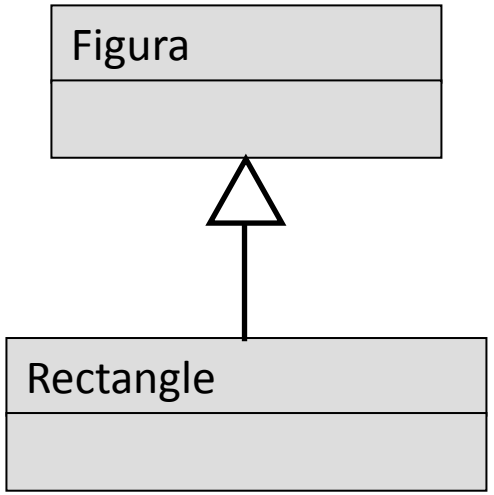


Informació de classes en temps d'execució (Recordatori)

- Després de realitzar una connexió polimorfa és freqüent la **necessitat de tornar a recuperar l'objecte original**, per a accedir a les seves operacions pròpies
- **Exemple:**

```
Figura [] figures = new Figura[10];
...
Figures[0]= new Rectangle();
Figures[1]= new Cercle();
....
Figura fig;
for (int i=0; i<10; i++){
  fig = figures[i];
}
```

Connexions polimorfes



Interessaria recuperar un Rectangle o Cercle en lloc d'una Figura.

305

Informació de classes en temps d'execució (Recordatori)

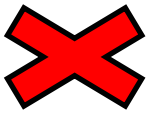
- Es tracta de l'operació inversa al polimorfisme (upcasting), denominada **downcasting**
  - Si el polimorfisme implica una generalització, el downcasting implica una especialització.
- Al contrari que el upcasting, el downcasting no pot realitzar-se directament mitjançant una connexió amb una referència de la classe de l'objecte.
- Recordatori:

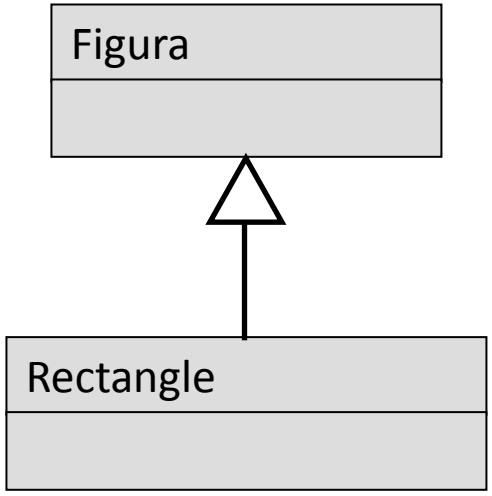
Upcasting

```
Figura figura = new Rectangle();
```

Downcasting

```
Rectangle rectangle = new Figura();
```





Informació de classes en temps d'execució

- Un casting permet forçar la connexió a la referència
- Un intent de casting impossible generarà una excepció ***ClassCastException*** en temps d'execució
- Possibles accions:
  - Podem capturar aquesta excepció per a determinar si l'objecte apuntat per la referència és del tipus esperat o no, realitzant accions diferents en cada cas *try …. catch*
  - O, podem utilitzar **instanceof** per determinar si l'objecte és de la classe esperada abans de realitzar el casting.

Genericitat en Java

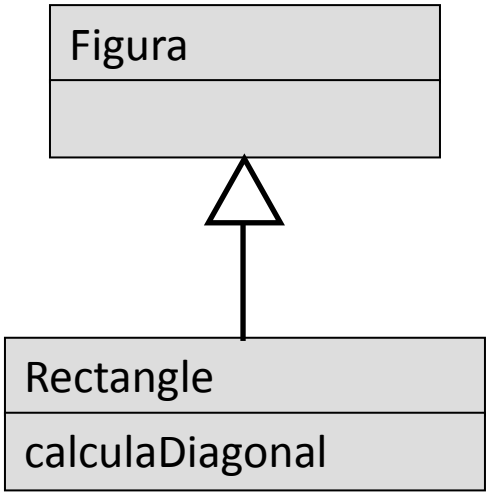
- Si defineixo una estructura de dades de tipus `Object`
- Ja que tot tipus és compatible amb l'arrel, obtenim les propietats:

- **Inserció**:
  - Puc inserir qualsevol tipus d'objectes
  - El control l'ha d'implementar el programador

- **Extracció**:
  - Recupero elements de tipus `Object`
  - Fa falta fer una conversió explícita

Exemple: diagonal màxima

- El mètode diagonal, és un mètode pròpi de la subclasse.



Exemple: diagonal màxima

```
Figura [] figures = new Figura[10];
     …
float actual, maxDiagonal=0;
for (int i=0; i<10; i++){
    actual = figures[i].calculaDiagonal();
    if (actual>maxDiagonal)
        maxDiagonal=actual;
}
```

Mètode propi de la classe Rectangle

Donarà error de compilació!

¿Què passa si no és un rectangle?
Tindríem que preguntar pel tipus

Identificació del tipus en temps d'execució

- `if (figures[i] instanceof Rectangle) …`
- `java.lang` conté la classe **`Class:`**
  - Conèixer el nom de la classe d'un objecte:

    `String getName()`
  - Saber si un objecte és instància de la classe o d'una subclasse:

    `boolean isInstance(Object o)`

- `if figures[i].getClass().getName().equals("Rectangle")…`

instanceof vs. equivalencia de Class

- `instanceof` o `isInstance`
  "Ets d'aquesta classe o d'una classe derivada d'aquesta?"

- Comparant els objectes `Class`
  "Ets exactament d'aquesta classe?"

- Exemple: Rectangle és una subclasse de la classe Figura

```
Rectangle r = new Rectangle();
(r instanceof Figura) → true
(r.getClass().equals(Figura.class)) → false
```

En l'exemple d'Interfícies

```
public abstract class Animal {
   public abstract void ferSoroll();
}
```

```
public class Gat extends Animal{
 public void ferSoroll(){
       System.out.println("miau");
   }
 public void esgarrapar(){
       System.out.println("esgarrapa");
 }
}
```

```
public class Gos extends Animal{
   public void ferSoroll(){
      System.out.println("guau");
   }
   public void serAmigable() {
      System.out.println("fa gràcies");
   }
   public void jugar() {
     System.out.println("juga");
   }
   public void persegueix() {
     System.out.println("persegueix");
   }
}
```

En l'exemple d'Interfícies

```
public class TestLlistaAnimals {
 public static void main(String[] args){
      ArrayList<Animal> llistaAnimals =
       new ArrayList<Animal>();
      Gos gosEx = new Gos();
      Gat gatEx = new Gat();
      llistaAnimals.add(gosEx);
      llistaAnimals.add(gatEx);
       //...
      // continuació del mètode main:
      Gos gos;
      Gat gat;
      Iterator<Animal> itrLlista = llistaAnimals.iterator();
      Animal animal;
      while(itrLlista.hasNext()) {
          animal = itrLlista.next();
          System.out.println("He extret el animal del tipus:" + animal.getClass());
          // animal.persegueix(); // Error de compilació
          // animal.esgarrapar(); // Error de compilació
          // No puc fer aquestes crides abans he de fer un cast de la següent
manera:
          if (animal instanceof Gos){
            gos = (Gos) animal;
            gos.persegueix();
          }else if (animal instanceof Gat){
            gat = (Gat) animal;
            gat.esgarrapar();
          }
      }
 }
}
```

Sortida per pantalla:

```
He extret el animal del tipus:class unAltreInterficies.Gos
persegueix
He extret el animal del tipus:class unAltreInterficies.Gat
esgarrapa
```

Framework Col·leccions

- Una **col·lecció** és un objecte que agrupa múltiples elements en una única unitat.
- Normalment representen elements d'informació dins d'un grup natural, com
  - una bústia de correu (una col·lecció de correus),
  - un directori (una col·lecció de fitxers),
  - una guia telefònica (una associació entre noms i números de telèfon).
- La llibreria standard de Java ens ofereix classes i interfícies que ens permeten manegar col·leccions d'objectes
- **Piles**, **Cues**, **Llistes**, **Conjunts** són casos particulars de col·leccions d'objectes

Col·leccions

- Encara que ArrayList serà la que més utilitzeu, hi ha altres col·leccions útils:
  - LinkedList,
  - HashSet,
  - HashMap,

ArrayList vs. LinkedList

- LinkedList: una altra implementació d'una llista.
- Una qüestió d'implementació:

- Quan necessiteu accedir de forma seqüencial i teniu un nombre poc variable d'elements → ArrayList.
- Quan necessiteu esborrar o inserir al davant o al mig moltes vegades el contingut de la llista → LinkedList.

Creació d'una ***LinkedList***

```
LinkedList map = new LinkedList ();
```

Mapes: Exemples d'Ús

- Conté clau I valor

- Creació d'una ***Map***

```
Map map = new HashMap();
map.put("joan", "777777777");
map.put("ana", "888888888");
map.put("jordi", "999999999");

// això imprimeix '888888888'
System.out.println("El telèfon de ana és: "+ map.get("ana"));
```

El telèfon de ana és: 888888888

Mapes: Exemples d'Ús

- Exemple d'ús (Copieu el codi i proveu-lo)

?: operador condicional ternari
test ? expression1 : expression2

```
import java.util.*;
public class Freq {
      private static final Integer ONE = new Integer(1);
      public static void main(String args[]) {

        Map m = new HashMap();
        // Initialize frequency table from command line
        for (int i=0; i < args.length; i++) {
            Integer freq = (Integer) m.get(args[i]);
            m.put(args[i], (freq==null ? ONE : new Integer(freq.intValue() + 1)));
        }
        System.out.println(m.size()+" distinct words detected:");
        System.out.println(m);
      }
}
```

Resultat execució: "java Freq 1 2 3 2 2 3 2 "
3 distinct words detected:
{3=2, 2=4, 1=1}

Col·leccions i iteradors

La interfície `Iterator` I

- Un iterador és un objecte que proveeix una forma de processar una col·lecció d'objectes, un a un, seguint una seqüència.
- Un iterador ens permet recorre els elements d'una col·lecció d'objectes

- Un iterador es crea formalment implementant la interfície `Iterator<E>`, que conté 3 mètodes:
  - **`hasNext`** → retorna un resultat booleà que és cert si a la col·lecció queden objectes per processar
  - **`next`** → retorna el següent objecte a processar
  - **`remove`** → elimina l'objecte més recentment retornat pel mètode `next`

Col·leccions i iteradors

La interfície `Iterator` I

```
public interface Iterator<E>
{
    E next();
    Boolean hasNext();
    void remove();  //opcional
}
```

- Alguna cosa és iterable si es pot iterar sobre ell. Per poder iterar usem un iterador. Una classe és iterable si és capaç de retornar-nos un iterador

```
public interface Iterable<E> {
  public Iterator<E> iterator();
}
```

Col·leccions i iteradors

La interfície `Iterator` II

- Implementant la interfície Iterator una classe formalment estableix que els objectes d'aquesta classe són iteradors
- El programador ha de decidir com implementar les funcions d'iteració
- Un iterador, per tant, caracteritza una seqüència

Col.leccions: Exemples d'Ús

- **Creació d'una col·lecció d'objectes**

```
Collection c = new ArrayList();
c.add("Hello");
c.add("World");
```

- **Recorregut** d'una col·lecció amb un **iterador**

```
for (Iterator i = c.iterator(); i.hasNext(); ) {
    String s = (String)i.next();
    System.out.println(s);
}
```

- **Recorregut** d'una col·lecció amb un ***for .. each***

```
for (Object item : c) {
    System.out.println(item.toString());
}
```

Tipus parametritzats

- També podem construir els nostres pròpis tipus parametr¡zats.

Tipus parametritzats

```
public class TaulaBicicleta{
    private Bicicleta [] taula;
    private int numElements;
    ....
    public void afegir(Bicicleta element){
            ...
    }
}
```

```
public class TaulaPellicula{
        private Pellicula [] taula;
        private int numElements;
        ....
        public void afegir(Pellicula element){
                ...
        }
}
```

```
public class Taula<T> {
        private T [] taula;
        private int numElements;
        ....
        public void afegir(T element){
                ...
        }
}
```

← Genèric

→ Exemple: ArrayList

Exemples d'Ús

- **Exemple** 1: Definició de mètodes que treballen contra la interface Collection.

  Col·leccions de tipus heterogeni

  CreaColeccio.java

- **Exemple** 2: Col·leccions de tipus homogeni

  CreaColeccioHomogenea.java

Col·leccions i iteradors: Exemple 1

```
import java.util.*;

public class CreaColeccio {
  public static void main(String[] args) {
    Collection myCollection1 = new ArrayList();
    Collection myCollection2 = new HashSet();

    fillCollection(myCollection1);       fillCollection(myCollection2);
    showCollection(myCollection1);   showCollection(myCollection2);
    treuMaria(myCollection1);            treuMaria(myCollection1);
    diguesSiEstaMaria(myCollection1); diguesSiEstaMaria(myCollection2);
  }
```

Col·leccions i iteradors: Exemple 1

```
public static void fillCollection(Collection c) {
   c.add(34);
   c.add("Pepe");
   c.add(new Gato("Sasha"));
 }

public static void showCollection(Collection c) {
   if (c.isEmpty()) { System.out.println("La col·lecció esta buida");
   } else {
       System.out.println("La col·lecció conté " + c.size() + " elements:");
       System.out.println(c);
   }
 }
```

Col·leccions i iteradors: Exemple 1

```
 public static void treuMaria(Collection c) {
    c.remove(“Maria");
  }

public static void diguesSiEstaMaria (Collection c) {
    if (c.contains(“Maria")) {
      System.out.println(“Maria està dins de la col·lecció");
    } else {
      System.out.println(“Maria no està a la col·lecció");
    }
  }
}
```

Col·leccions i iteradors: Exemple 2

```
public class CreaColeccioHomogenea {
 public static void main(String[] args) {
      Collection<Gat> myCollection1 = new ArrayList<Gat>();
      showCollection(myCollection1);
      fillCollection(myCollection1);
      showCollection(myCollection1);
 }
```

```
class Gat {
          String nom;
          Gat(String n) {
                nom = n;
         }
          public String toString() {
                return nom;
         }
         public void miolar(){
                System.out.println("miau");
         }
}
```

Col·leccions i iteradors: Exemple 2

```
public static void fillCollection(Collection<Gat> c) {
   c.add(new Gat(“Misu"));
   c.add(new Gat("Marramiau"));
   c.add(new Gat("Sasha"));
 }
public static void showCollection(Collection<Gat> c) {
   if (c.isEmpty()) {
     System.out.println("La col·lecció està buida");
   } else {
     System.out.println("La col·lecció conté " + c.size() + " elements:");
     System.out.println(c);
   }
 }
```

Exemple: iteradors

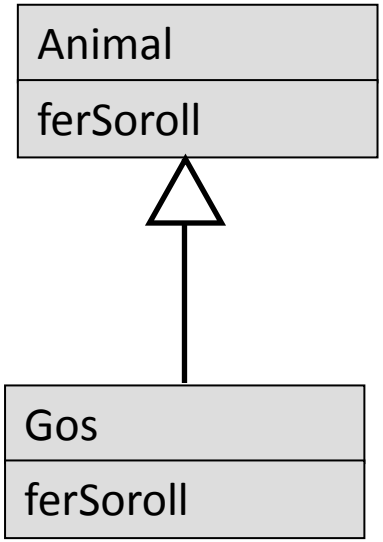
```
public static void fesMiolar(Collection<Gat> c){
   Iterator<Gat> it = c.iterator();
   while(it.hasNext()) {
     Gat g = it.next();
     g.miolar();
   }
```

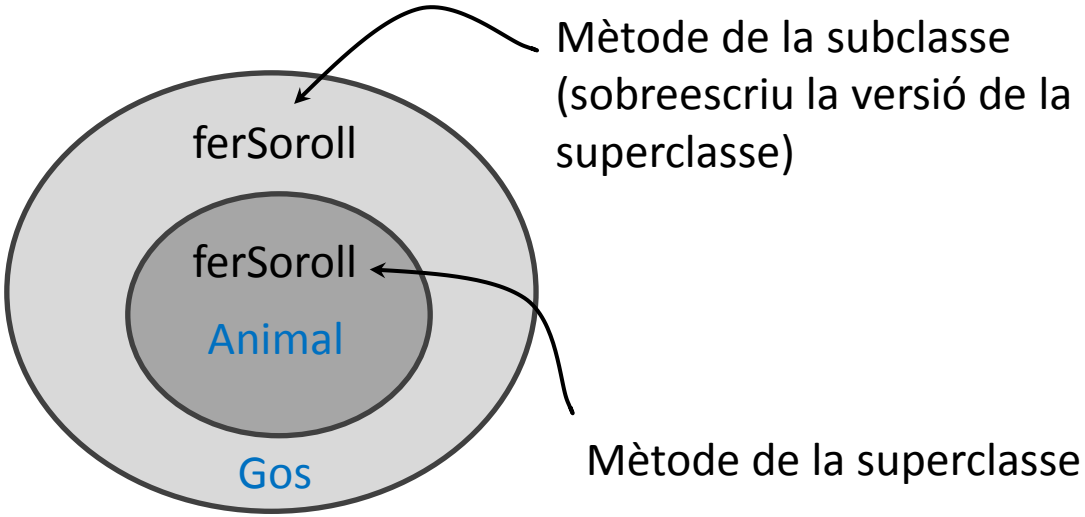
REPÀS

Consideracions sobre l'herència

- Herència és un mecanisme que permet definir una classe nova a partir d'una d'anterior descrivint les diferències entre elles.
- Quan es defineix una subclasse no es pot anul·lar res definit anteriorment en la jerarquia de classes; ni mètodes ni atributs. Els atributs i mètodes de la superclasse estaran sempre definits, per herència, en la subclasse.
  - Aquesta restricció únicament s'aplica als atributs i mètodes definits amb la visibilitat public o protected. Les classes filla no tenen accés als atributs i mètodes definits com a private.

Herència





Nota:
Una referència a un objecte de la subclasse (Gos) sempre cridarà a la versió de la subclasse del mètode sobreescrit. Això és el polimorfisme.
Però el codi de la subclasse pot cridar super.ferSoroll() per invocar la versió de la superclasse.

La paraula reservada **super** és una referència a la porció de la superclasse d'un objecte.
Quan el codi de la subclasse utilitza super, com en super.ferSoroll(), la versió del mètode de la superclasse s'executarà.

Exemple (1)

```
...
ArrayList  integerList = new ArrayList();
integerList.add( new Integer(1));
integerList.add( new Integer(2));

Iterator listIterator = integerList.iterator();
while(listIterator.hasNext()) {
     Integer item = (Integer) listIterator.next();
}
. . .
```

Els elements de la lista són de tipus Object.

El programador ha de fer el casting

Exemple (2)

El mateix exemple afegint un error:

```
...
ArrayList  integerList = new ArrayList();
integerList.add( new Integer(1));
integerList.add( new Integer(2));
integerList.add("Joan");

Iterator listIterator = integerList.iterator();
while(listIterator.hasNext()) {
    Integer item = (Integer) listIterator.next();
}
. . .
```

No hi ha cap restricció dels elements

El compilador no se n'adona del casting il·legal.
Serà detectat en temps d'execució.

Exemple (3)

// Eliminar paraules de 4 lletres de la col·lecció c. Els elements haurien de ser String

```
static void expurgate(Collection c) {
    for (Iterator i = c.iterator(); i.hasNext(); )
        if (((String) i.next()).length() == 4)
            i.remove();
}
```

No utilitzar un bucle de recorregut d'indexos per fer això.
No funcionarà! **Exercici:** provar-ho.

El mateix exemple modificat per a utilitzar tipus generics:

// Eliminar paraules de 4 lletres de la col·lecció c

```
static void expurgate(Collection<String> c) {
    for (Iterator<String> i = c.iterator();  i.hasNext(); )
        if (i.next().length() == 4)
            i.remove();
}
```

Exemple (4)

Programa que simula un zoològic:

```
public class Zoo{
        private ArrayList<Animal> hostes = new  ArrayList<Animal>();
        public void nuevoAnimal(Animal a){
                hostes.add(a);
        }
}
```

Codi genèric per tots els animals

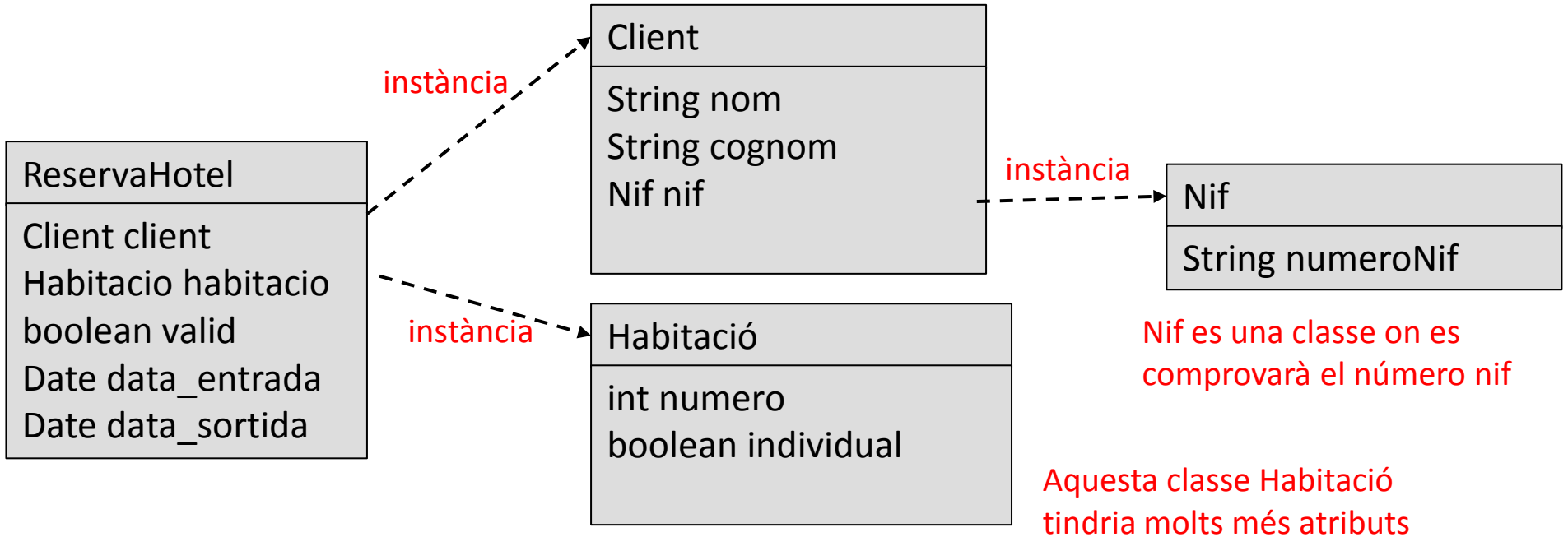
- Si no utilitzem polimorfisme tindríem que saber en temps de disseny quins animals tindràs exactament.
- Per tant dissenyes una classes que sigui lo suficientment genèrica com per a que accepti qualsevol tipus d'animal sense necessitat de saber quin tipus d'animal és: ArrayList<Animal>

Exercici

- Es vol implementar una aplicació de gestió de reserves d'hotel seguint el diagrama de classes definit a continuació. On apareixen el nom de les classes i la llista dels seus atributs.
- Implementa les classes:
  - **ReservaHotel**
  - **Habitacio**
  - **Client**
  - **GestorReserves**

- Al mètode main de la classe **GestorReserves** s'ha de crear una reserva i validar-la.

Exercici: Diagrama de classes



Exemple: Implementació

```
public class GestorReserves {
   public static void main(String[] args){
        Habitacio habitacio = new Habitacio(1);
        Client client = new Client();
         // demanar la informació sobre el client
         // Crear una nova reserva del hotel:
        ReservaHotel novaReserva = new ReservaHotel(habitacio, client);
        // Validar quan ja s'ha pagat la reserva:
        novaReserva.validar();
   }
}
```

No cal definir el constructor per defecte si no vols crear una reseva sense assignar habitació al client.

El constructor per defecte existeix mentre no es sobrecarregui amb qualsevol conjunt de parametres (inclos sense parametres).

```
public class ReservaHotel{
  Habitacio habitacio;
  Client client;
  Date data_entrada;
  Date data_sortida;
  boolean valid;
  // constructor de la classe:
  public ReservaHotel(Habitacio habitacio, Client client){
         this.habitacio = habitacio;
         this.client = client;
         valid = false;
  }
  // mètode per validar la reserva:
  public void validar(){
     valid = true;
  }
// més mètodes ...
}
```

```
public class Habitacio{
    int numero;
    boolean individual;
    // constructor de la classe:
    public Habitacio(int numero){
        this.numero= numero;
        this. individual = false;
     }
    public Habitacio(int numero, boolean individual){
          this.numero= numero;
          this. individual = individual;
     }
    //… setters & getters …
}
```

Exemple: Implementació

Una altra opció d'Implementació de la classe ReservaHotel:

```
public class ReservaHotel{
  Habitacio habitacio;
  Client client;
  Date data_entrada;
  Date data_sortida;
  boolean valid = false;
  // constructors de la classe:
  public ReservaHotel(){
        valid = false;
  }
  public ReservaHotel(Habitacio habitacio, Client client){
        this();
        this.habitacio = habitacio;
        this.client = client;
  }
  // mètode per validar la reserva:
  public void validar(){
     valid = true;
  }
// més mètodes ...
}
```

Si volem crear una reseva sense assignar habitació i client ho podem fer així.

Exemple

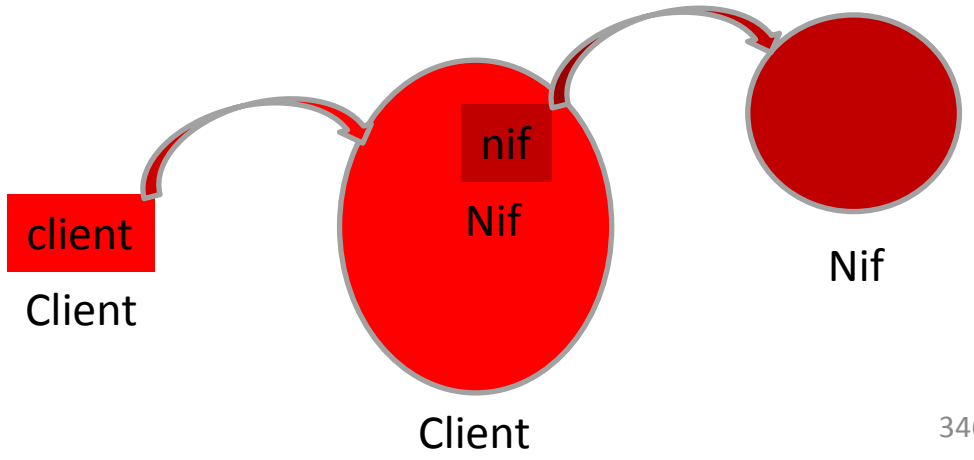
```
public class Client{
   String nom;
   String cognom;
   Nif nif;

// constructor de la classe:
   public Client(String nif){
      nif = new Nif(nif);
   }
    // mètodes d'accés I d'escriptura de la classe:
   public void setNom(String nom){
      this.nom=nom;
   }
   public void setCognom(String cognom){
      this.cognom = cognom;
   }
    public String getNom(){
      return this.nom;
   }
   public String getCognom(){
      return this.cognom;
   }
  // més mètodes
}
```

Sempre que creem un nou client ha de tenir un Nif associat.

Exemple: Observació

```
public class GestorReserves {
   public static void main(String[] args){
       Habitacio habitacio = new Habitacio(1);
       Client client = new Client("44444444P");
        // demanar la informació sobre el client
        // Crear una nova reserva del hotel:
       ReservaHotel novaReserva = new ReservaHotel(habitacio, client);
       // Validar quan ja s'ha pagat la reserva:
       novaReserva.validar();
   }
}
```

Quan instanciem un objecte de la classe Client, estem instanciant un objecte de la classe Nif.



Referències del bloc 2

- Bertrand Meyer, "**Construcción de software orientado a objetos",** Prentice Hall, 1998.
- Grady Booch (Rational Software), "**Software Architecture and UML**". Presentació P. Letelier.
- Grady Booch. "**Object-Oriented Analysis and Design with Applications**". The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994. ISBN: 0-8053-5340-2. 589 páginas. Traducción española, Addison-Wesley, 1996.
- Bert Bates, Kathy Sierra. **Head First Java**. O'Reilly Media, 2005.
- Michael T. Goodrich, Roberto Tamassia "**Data Structures and Algorithms in Java**" John Wiley & Sons, Inc.
- **Programació orientada a objectes**. Joan Arnedo Moreno, Daniel Riera i Terrén. Manuals de la UOC. Col·lecció 105. ISBN:978-84-9788-582-9.05/2007.
- http://java.sun.com/docs/books/tutorial/collections/

INTRODUCCIÓ A LA PROGRAMACIÓ ORIENTADA A OBJECTES I A ESDEVENIMENTS

Bloc 3:

Programació Orientada a Events

Índex Bloc 3:
Programació Orientada a Events

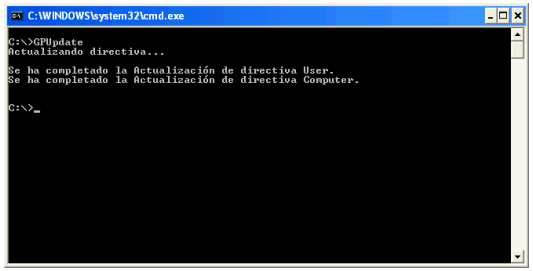
- Mecanismes d'interacció
  - Interacció mitjançant flux seqüencial
  - Interacció mitjançant programació orientada a events
- Programació d'Interfícies Gràfiques d'Usuari
- Model de gestió d'events: Exemple d'implementació d'una finestra.
- Events i Listeners
- Components i Contenidors
- Classes adapter i calsses internes: Exemple d'implementació d'una finestra que es tanca.
- Layout manager
- Mes sobre swing components: Exemples
- Look and feel
- Panells i gràfics
- Animacions

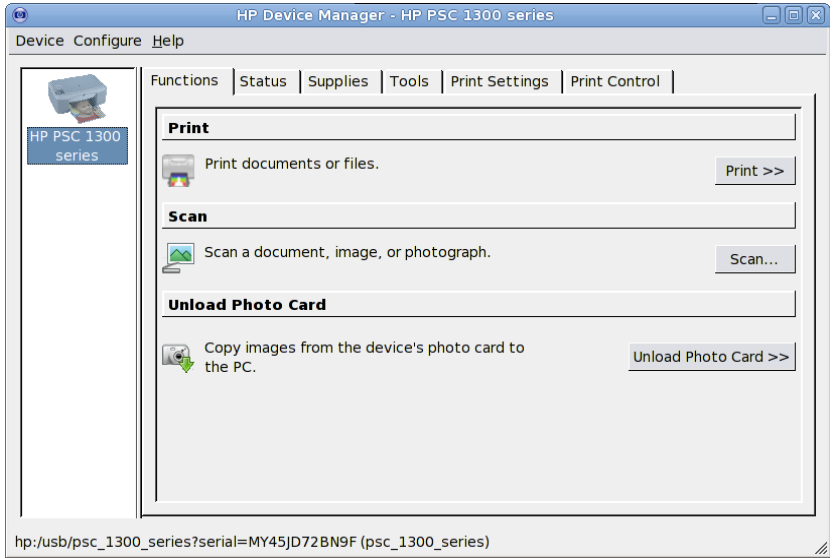
MECANISMES D'INTERACCIÓ

Introducció

- Programació orientada a events (POE)

- Línia de comandos vs. Interfícies gràfiques d'usuari





Mecanismes d'interacció

1) Estil tradicional d'interacció amb els usuaris
2) Programació Orientada a Events

Mecanismes d'interacció

1) Estil tradicional d'interacció amb els usuaris:
   - Un programa que necessita una entrada provinent de l'usuari l'obtindrà mitjançant l'execució repetida d'escenaris de la forma:

     1 ...Efectuar algun càlcul...
     2 Imprimir ("Per favor, escriu el valor del paràmetre x")
     3 Llegir entrada
     4 x = valor llegit
     5 ... Seguir endavant amb el càlcul fins que necessiti un altre valor de l'usuari ...

   - Segueixen un **flux seqüencial** en el que es tenen cicles: entrada->processament->sortida

Mecanismes d'interacció

2) Programació Orientada a Events

- Els papers s'inverteixen, **les operacions ...**
  - **No es produeixen** perquè el software ha arribat a una determinada fase de la seva execució,
  - **Es produeixen** perquè un determinat event ha donat lloc a l'execució d'un determinat component de software.

- L'entrada determina l'execució del software i no al revés.

Concepte d'Event

- Missatge de software que indica que alguna cosa ha succeït:
  - Accions de l'usuari sobre una GUI,
  - Temporitzacions,
  - Canvi d'estat,
  - ...
- Exemples:
  - Polsar una tecla
  - Fer un click d'un ratolí,
  - Mantenir polsat el botó del ratolí
  - Soltar el botó del ratolí
  - Acaba de sonar una cançó
  - Passa un minut

Concepte d'Event

- Hi ha un catàleg d'events
- Un objecte event representarà una acció de l'usuari
- També es poden definir events personalitzats que un component software pot enviar explícitament mitjançant una crida a procediment.
- POO ajuda a desenvolupar l'esquema de programació orientada a events.

POE per programació
d'Interfícies Gràfiques d'Usuari

- POE s'utilitza en el context de programació **d'Interfícies Gràfiques d'Usuari** (GUI: Graphics User Interface)
- Quan programem una GUI hem de tenir en compte la **varietat de possibles interaccions** amb l'usuari.
  - En lloc d'un únic flux d'entrada de dades per consola, les GUIs **permeten moltes més accions de l'usuari**.
- Per exemple:
  - Pressionar botons gràfics,
  - Escriure text en un camp de text,
  - Moure elements gràfics.
- Elements:
  - Finestres,
  - Menus,
  - Botons,
  - Panells,

} Gestionen entrada de l'usuari
Proporcionen un cas especial de context

INTERFÍCIE GRÀFICA D'USUARI

Creació d'una Interfície Gràfica d'Usuari

Per construir una GUI fa falta:

1. **Un contenidor**, que és la finestra o part de la finestra on es situaran els components (botons, barres de desplaçament, menús, etc.) i a on es visualitzarà el que desitgem.

2. **Els components**: menús, botons de comandament, barres de desplaçament, caixes i àrees de text, botons de opció i selecció, etc.

3. **El model d'events**. L'usuari controla l'aplicació actuant sobre els components, d'ordinari amb el ratolí o amb el teclat.

   Cada vegada que l'usuari realitza una determinada acció, es produeix l'event corresponent, que el sistema operatiu transmet al paquet de gestió.

GUI en Java

- Java inclou, com a part de la seva biblioteca de classes estàndard, un conjunt de **components** per a crear interfícies gràfiques d'usuari
- Aquests elements s'agrupen en dos paquets:
  - **AWT** (Abstract Window Toolkit)
  - **SWING** (AWT millorat)

GUI en Java

- java.awt
  - Els components AWT depenen de les facilitats gràfiques oferides per cada sistema operatiu: els programes escrits amb AWT tindran un “look and feel” diferent en Windows i en UNIX
- java.swing
  - SWING és 100% Java i, per tant, completament independent de la plataforma.
  - Les components gràfiques es pinten en temps d’execució (per aquest fet les aplicacions SWING solen ser un poc més lentes que les AWT).

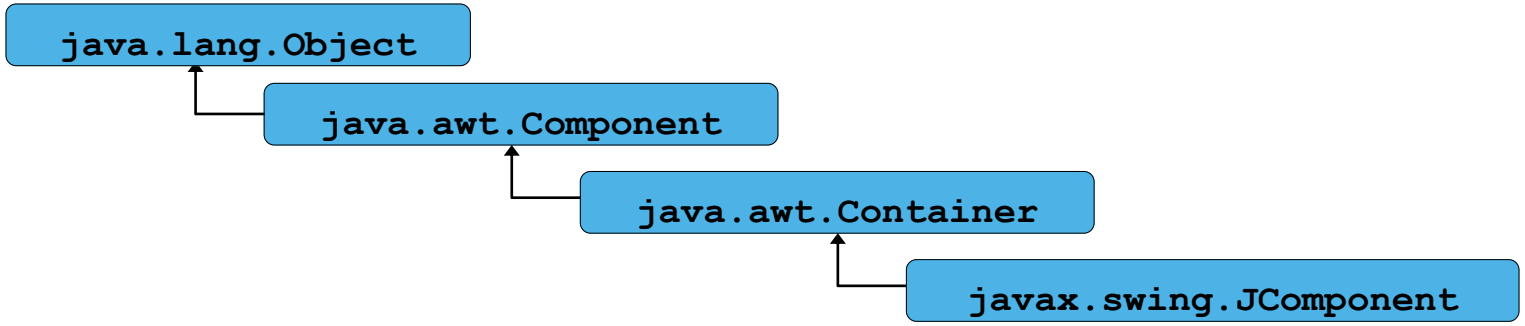
- En la pràctica les aplicacions Java amb GUIs solen barrejar AWT i SWING.

GUI en Java

- El AWT crea un objecte d'una determinada classe d'event, derivada de AWTEvent.
- Aquest event es tramés a un determinat mètode per a que el gestioni.
- En Java el component o objecte que rep l'event ha de "registrar" o indicar prèviament quin objecte es va a fer càrrec de gestionar aquell event → **Model de Delegació d'Events.**

GUI en Java

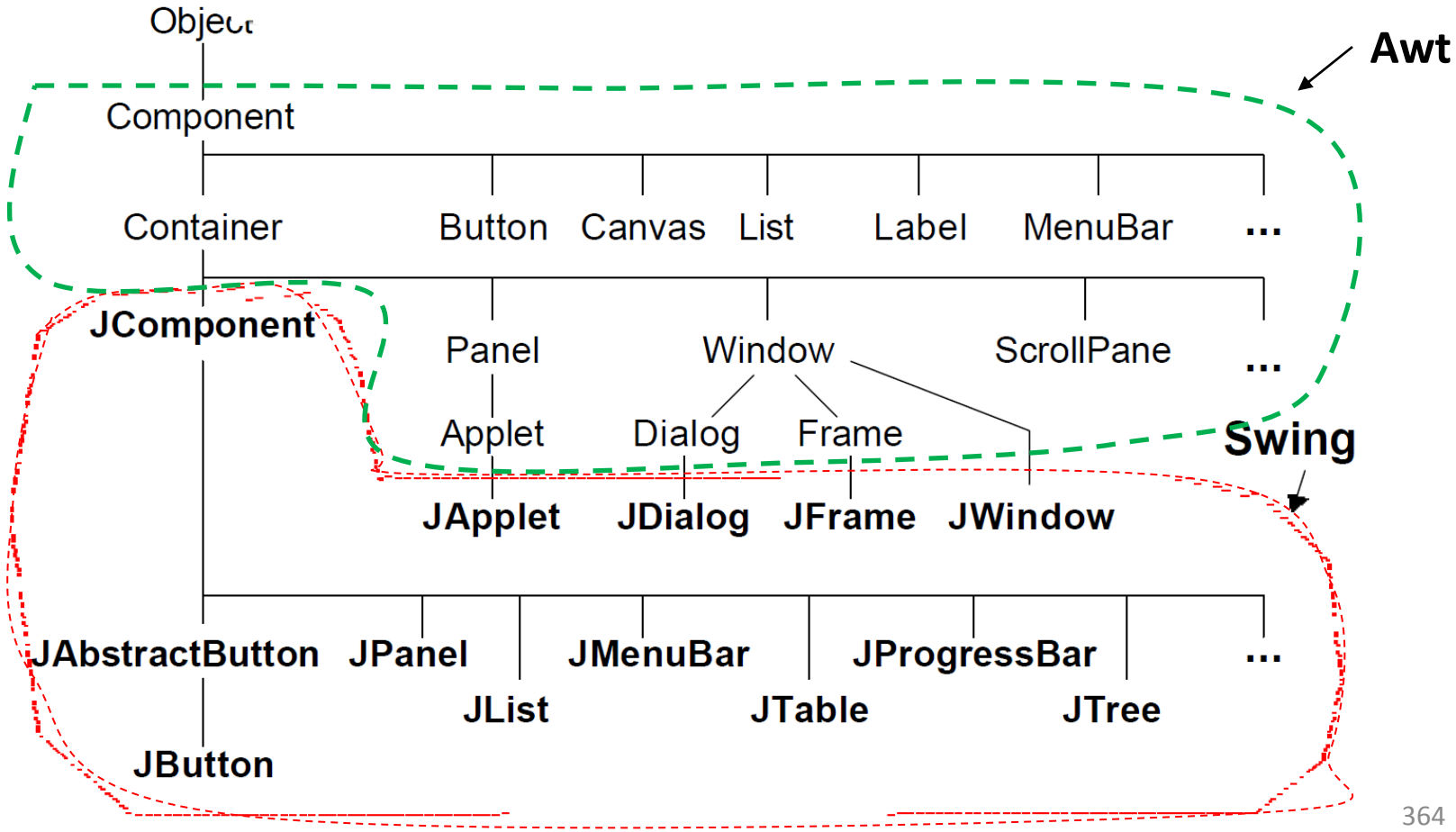
- Jerarquia d'herències dels components de Swing:



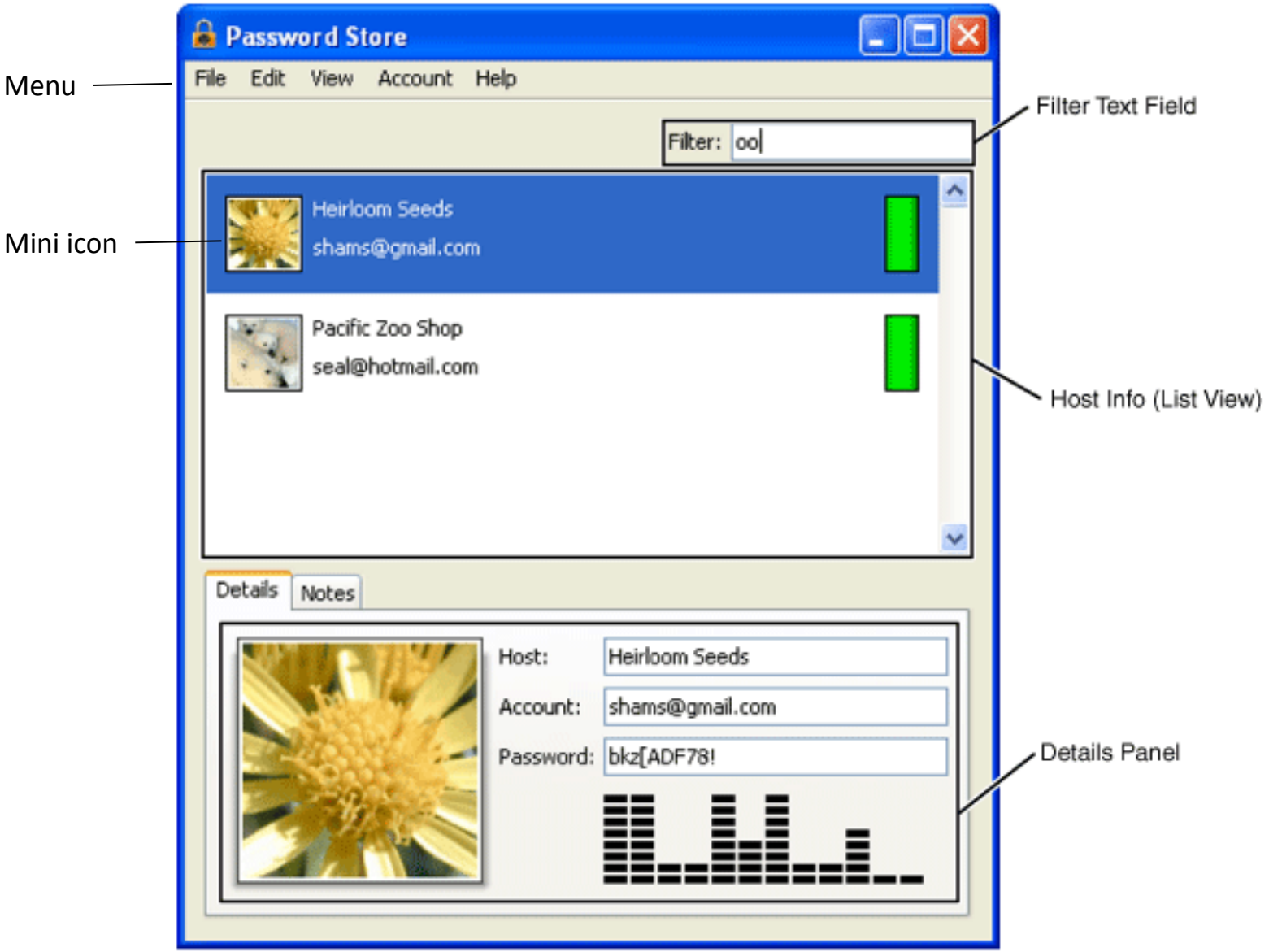
  - **Component** defineix mètodes que poden ser usats en les seves subclasses
    - Per exemple: **paint** and **repaint**
  - **Container** – col·lecció de components relacionades
    - Mètode **add** per afegir components a la finestra
  - **JComponent** - superclasse de la major part dels components de Swing
    - Moltes de les funcionalitats dels components hereten d'aquestes classes

Catàleg de components

Relació jeràrquica entre components:



Exemple


Password Store
File
Edit
View
Account
Help
Filter:
oo
Heirloom Seeds
shams@gmail.com
Pacific Zoo Shop
seal@hotmail.com
Details
Notes
Host:
Heirloom Seeds
Account:
shams@gmail.com
Password:
bkz[ADF78!
Menu
Mini icon
Filter Text Field
Host Info (List View)
Details Panel

Més sobre components

- Cada element gràfic de GUI és un component
- Cada component és **una instància d'una classe**
- Una component es crea com qualsevol altre objecte en Java

http://java.sun.com/docs/books/tutorial/uiswing/components/index.html

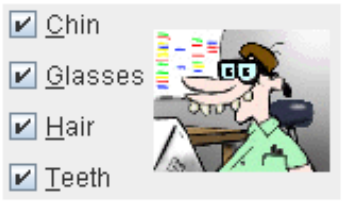
Classe Component

Métodos de Component	Función que realizan
boolean isVisible(), void setVisible(boolean)	Permiten chequear o establecer la visibilidad de un componente
boolean isShowing()	Permiten saber si un componente se está viendo. Para ello tanto el componente debe ser visible, y su container debe estar mostrándose
boolean isEnabled(), void setEnabled(boolean)	Permiten saber si un componente está activado y activarlo o desactivarlo
Point getLocation(), Point getLocationOnScreen()	Permiten obtener la posición de la esquina superior izquierda de un componente respecto al componente-padre o a la pantalla
void setLocation(Point), void setLocation(int x, int y)	Desplazan un componente a la posición especificada respecto al container o componente-padre
Dimension getSize(), void setSize(int w, int h), void setSize(Dimension d)	Permiten obtener o establecer el tamaño de un componente
Rectangle getBounds(), void setBounds(Rectangle), void setBounds(int x, int y, int width, int height)	Obtienen o establecen la posición y el tamaño de un componente
invalidate(), validate(), doLayout()	invalidate() marca un componente y sus contenedores para indicar que se necesita volver a aplicar el Layout Manager. validate() se asegura que el Layout Manager está bien aplicado. doLayout() hace que se aplique el Layout Manager
paint(Graphics), repaint() y update(Graphics)	Métodos gráficos para dibujar en la pantalla
setBackground(Color), setForeground(Color)	Métodos para establecer los colores por defecto

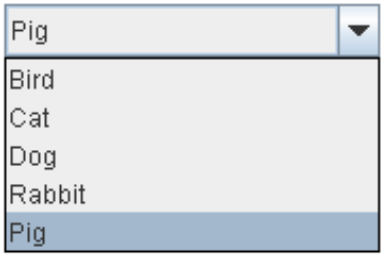
Tabla 5.4. Métodos de la clase Component.

Controls bàsics

JButton Botons

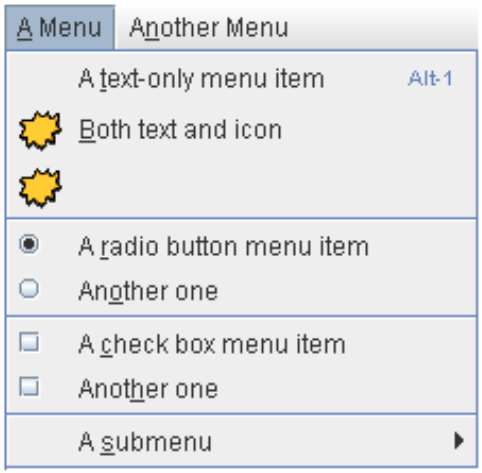


JCheckBox Botons



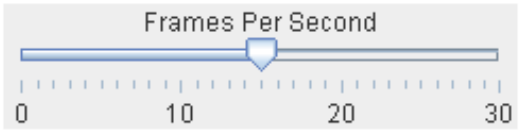
JComboBox Caixes combo

JList Llistes



Menús JMenu

JRadioButton Botons



JSlider Controls lliscants

JSpinner Controls numèrics

JTextField Camps de text (amb/sense format)

JPasswordField

Informació sobre Components

- http://java.sun.com/docs/books/tutorial/ui/features/components.html

MODEL DE GESTIÓ D'EVENTS: EXEMPLE D'IMPLEMENTACIÓ D'UNA FINESTRA.

Exemple 1: FINESTRA

- Seguim els passos bàsics:

1. Importar paquets javax.swing.XXX
2. Disposar un contenidor:
   - –JFrame
3. Agregar components al contenidor
4. Mostrar el contenidor

http://download.oracle.com/javase/tutorial/uiswing/TOC.html

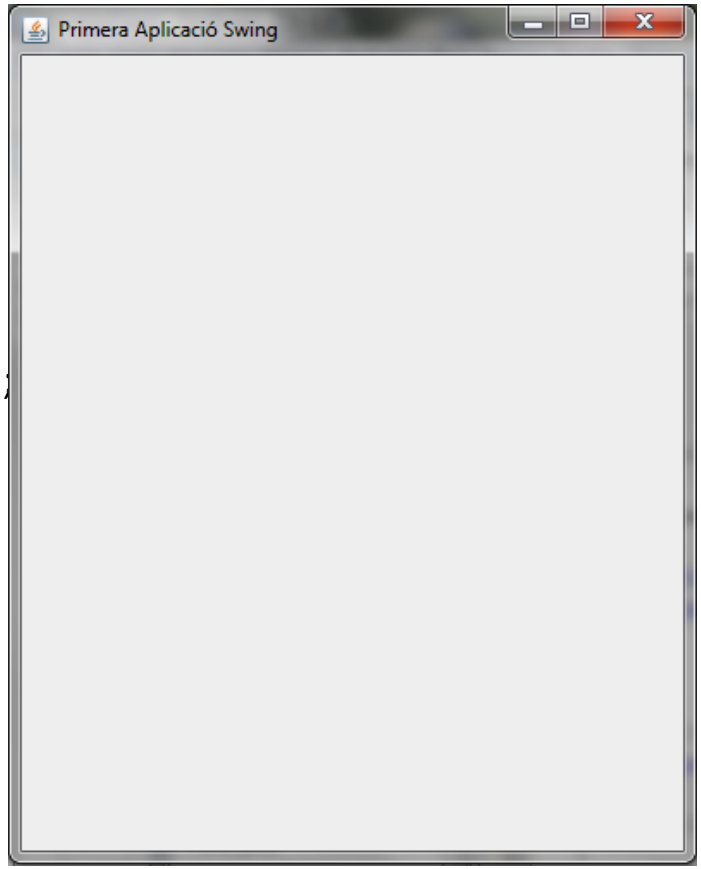
Exemple 1: Dos mètodes de creació de finestres

- La classe JFrame implementa un objecte finestra
- Per a crear una finestra, hi ha dues maneres principals de fer-ho:
  1. Crear un objecte de la classe JFrame
  2. Estendre la classe JFrame

Exemple 1: Dos mètodes de creació de finestres

- Primera manera:
  - creant un objecte de tipus JFrame: jFrameWindow

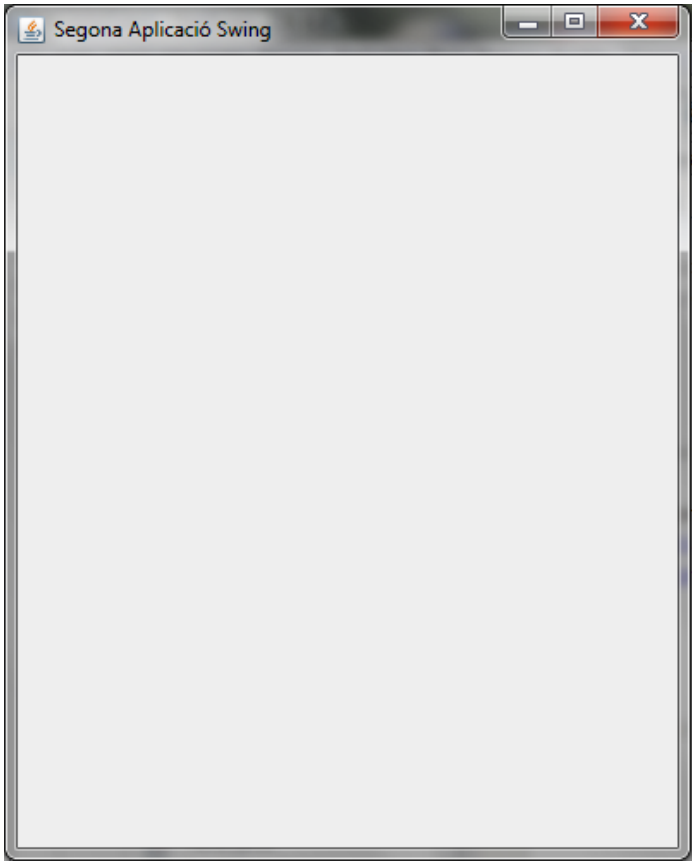
```
import javax.swing.*;
public class Finestra{
	public static void main(String []args) {
		JFrame jFrameWindow = new JFrame();
		jFrameWindow.setSize(400,500);
		jFrameWindow.setTitle(“Primera Aplicació Swing");
		jFrameWindow.setVisible(true);
	}
}
```



Exemple 1: Dos mètodes de creació de finestres

- Segona manera:
  - estenent la classe JFrame

```
import javax.swing.*;
public class Finestra extends JFrame {
	public Finestra() {
		this.setSize(400,500);
		this.setTitle(" Segona Aplicació Swing");
		this.setVisible(true);
	}
	public static void main(String []args) {
		Finestra finestra = new Finestra();
	}
}
```

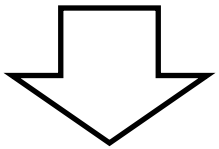


Exemple 1: Dos mètodes de creació de finestres

Fins aquí hem creat una aplicació senzilla de dues maneres.

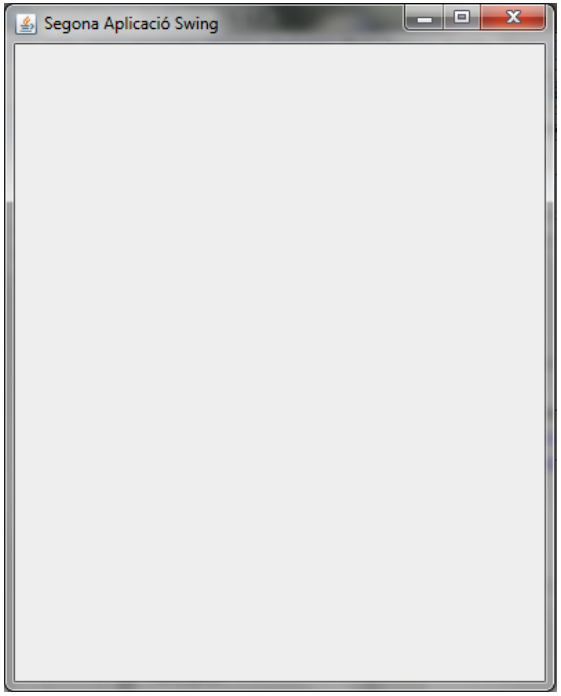
Però, en cap dels dos casos

l'aplicació no fa res



Afegim un botó que tingui una funcionalitat senzilla. Es a dir, que faci alguna cosa quan el premem.

Caldrà capturar els events que es llancen



Exemple 2: FINESTRA amb botó

```
public static void main(String []args) {
    JFrame frame = new JFrame();
    JButton boto = new JButton ("Apreta'm");
    frame.getContentPane().add(boto);
    frame.setSize(300,300);
    frame.setVisible(true);
}
```

No afegim un botó al **frame** directament→ Pensar en el frame com el marc de la finestra.
Afegim coses al **pane** (cristall) de la finestra

Definim el tamany

El fem visible

En realitat, el que hem de fer és el següent:
frame.getContentPane().add(BorderLayout.CENTER, boto);
Ho veurem més endavant amb detall.

Una vegada tenim el botó. Veurem:

- Cóm controlarem el seu tamany?
- Cóm controlarem el "look and feels"?
- Què passa quan el premem?
- Cóm podem fer que passin coses quan el premem?

Què passa quan premem el botó?

- Alguna cosa passa: l'aspecte canvia

Cóm podem fer que passin més coses quan el premem?

Cóm podem fer que passin més coses quan el premem?

- Necessitem:
  1. Un mètode que es cridi quan el botó es prem.
  2. Una forma de saber quan s'ha d'invocar aquest mètode, es a dir, una forma de saber quan es prem el botó.

→Estem interessats en:

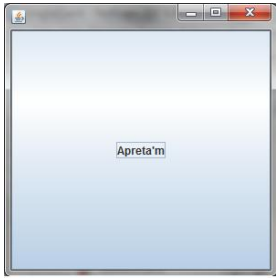
L'event: l'usuari prem el botó

Java UI – Manejar events

- En Java els events són representats per objectes
- Exemples:
  - Fer clic en un botó
  - Arrastrar el ratolí
  - Polsar Enter
- Els components AWT y Swing generen, llancen (*fire*) events
- java.awt.AWTEvent

Exemple 2: afegim funcionalitat

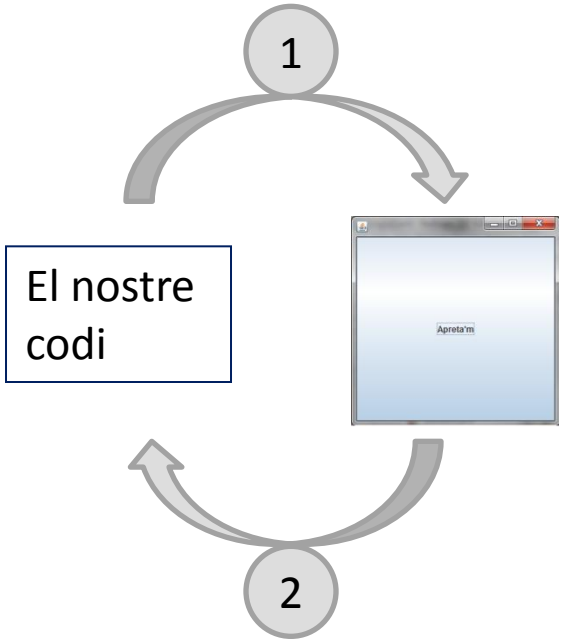
Volem que el text del botó canviï de "Apreta'm" a "He estat apretat".



- Necessitem:
  - Un mètode per canviar el text del botó:

```
public void changeIt() {
        boto.setText("He estat apretat");
}
```

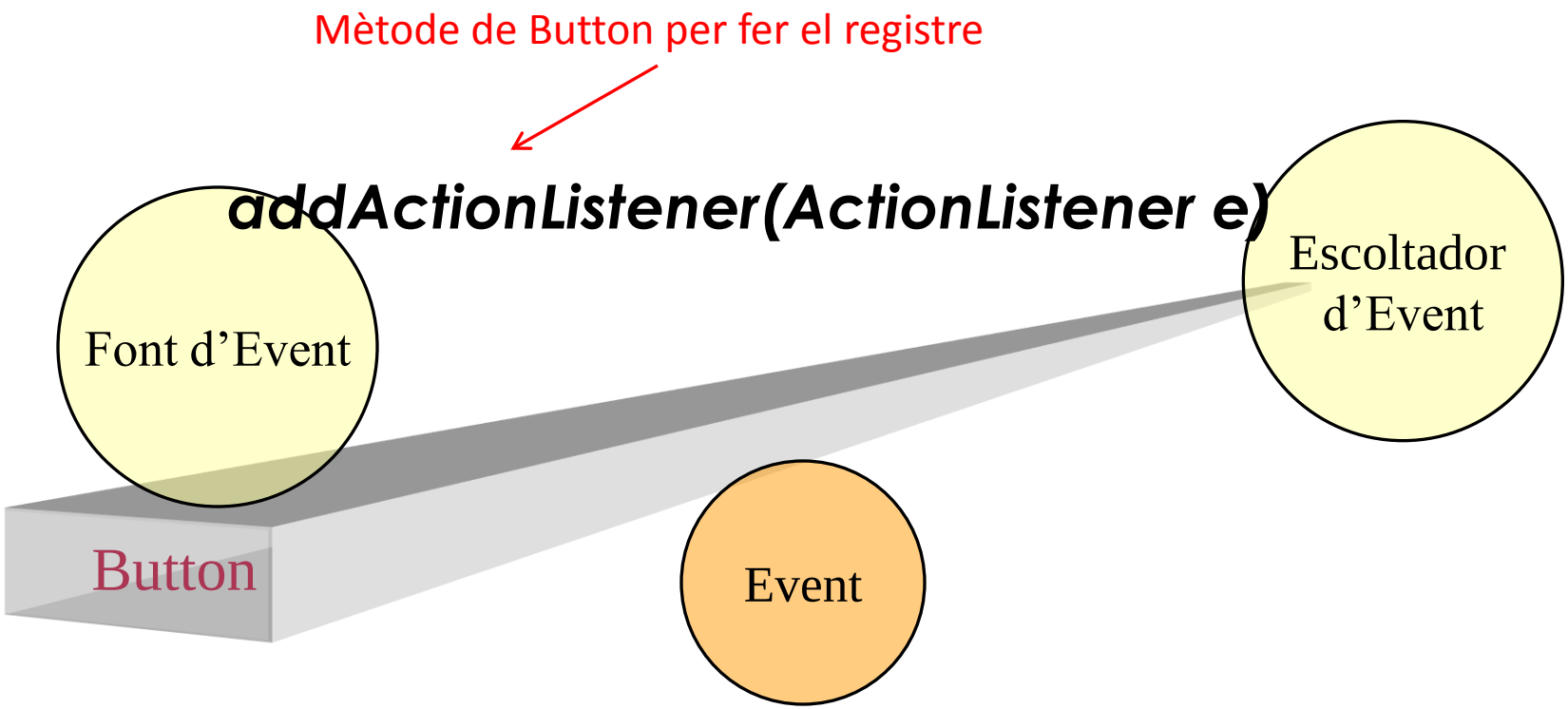
1. El botó ha de saber que ens interessa quan el premen
2. El botó ha de poder tornar a cridar-nos quan el premen



Model de Delegació d'Events

- Cada component pot generar events.
- En cada component es poden registrar **escoltadors (listeners) d'events** (dels tipus d'events que ells poden generar).
- Quan el component generi un event, invoca a tots els seus manejadors d'events.

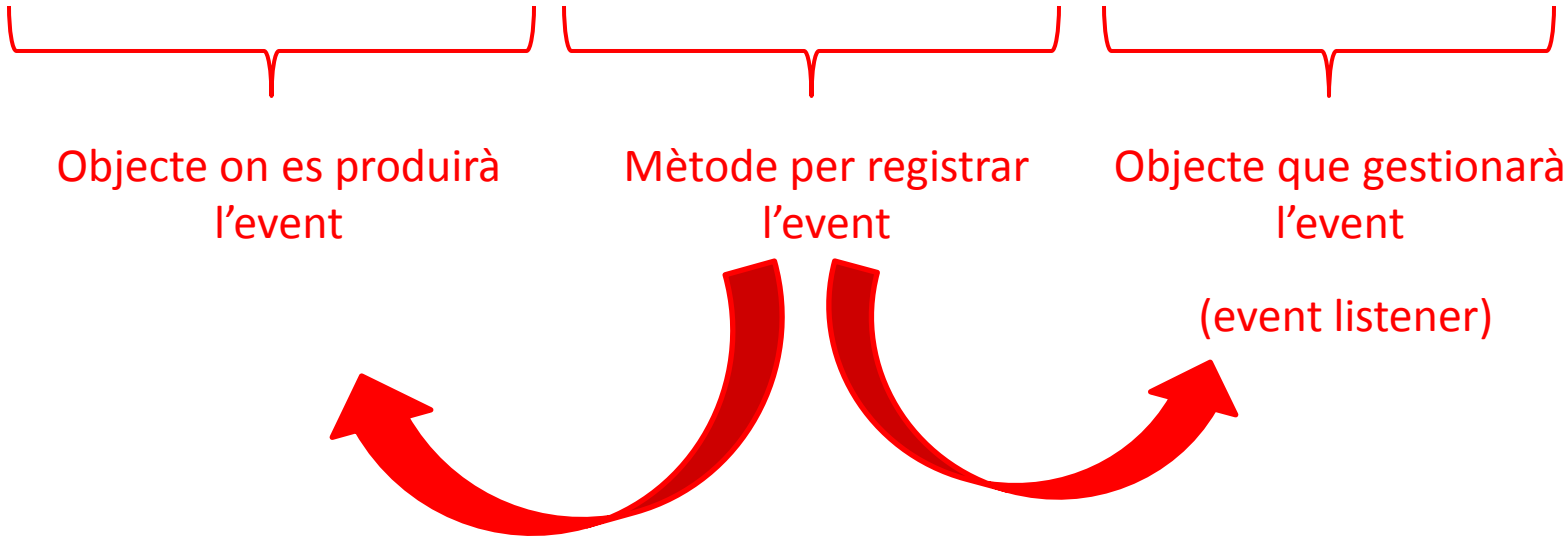
Model de Delegació d'Events



Model de Delegació d'Events

1. Registrar el Listener

```
eventSourceObject.addActionListener(ActionListener e)
```



2. Definir els mètodes de la interfície: implementar el mètode per que faci el que volem.

Connectar un Listener amb una font d'events

- Definir una classe que implementi la interfície Listener (o que estengui una classe que la implementi)

```
public class InterficieSimple extends JFrame implements ActionListener {…
```

- Afegir la implementació de la interfície dins de la classe InterficieSimple .
- En el cas de ActionListener, només té un mètode **actionPerformed** a implementar.

```
…
public void actionPerformed(ActionEvent e) {
        // aquí és on implemento el que s'ha de fer quan l'acció (event) succeeix
…
```

- Registrar el Listener amb la font

```
…
JButton okButton = new JButton("OK");
okButton.addActionListener(this);
…
```

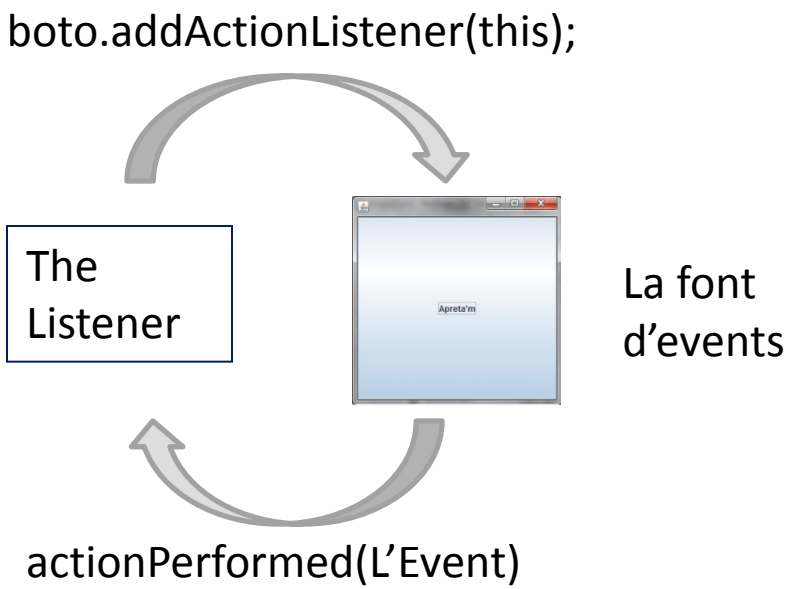
Exemple 2: Continuació

Voliem que el text del botó canviï de “Apreta’m” a “He estat apretat”.

- Implemetem un mètode per canviar el text del botó:

```
public void changeIt() {
        boto.setText(“He estat apretat”);}
```

- Registrem el botó a l’objecte que tractarà els seus events.



Comunicació entre el Listener i la font

- Quan implementem una interfície Listener donem al botó una forma de tornar a cridar-nos
- La interfície és on el mètode de crida està **declarat**, però no implementat.

<<interface>> ActionListener
actionPerformed(ActionEvent ev)

<<interface>> ItemListener
itemstateChange(ItemEvent ev)

Codi de l'exemple 2

```
import javax.swing.*;
import java.awt.event.*;

public class InterficieSimple implements ActionListener {
   JButton boto;
   public static void main (String[] args){
      InterficieSimple gui = new InterficieSimple ();
      gui.go();
   }

   public void go(){
      JFrame finestra = new JFrame();
      boto = new JButton("Apreta'm");

      boto.addActionListener(this);

      finestra.getContentPane().add(boto);

      finestra.setSize(300,300);
      finestra.setVisible(true);
   }

   public void actionPerformed(ActionEvent event) {
      boto.setText("He estat apretat");
   }
}
```

Dins estan les classes:
ActionListener i ActionEvent

La classe InterficieSimple implementa la interfície ActionListener

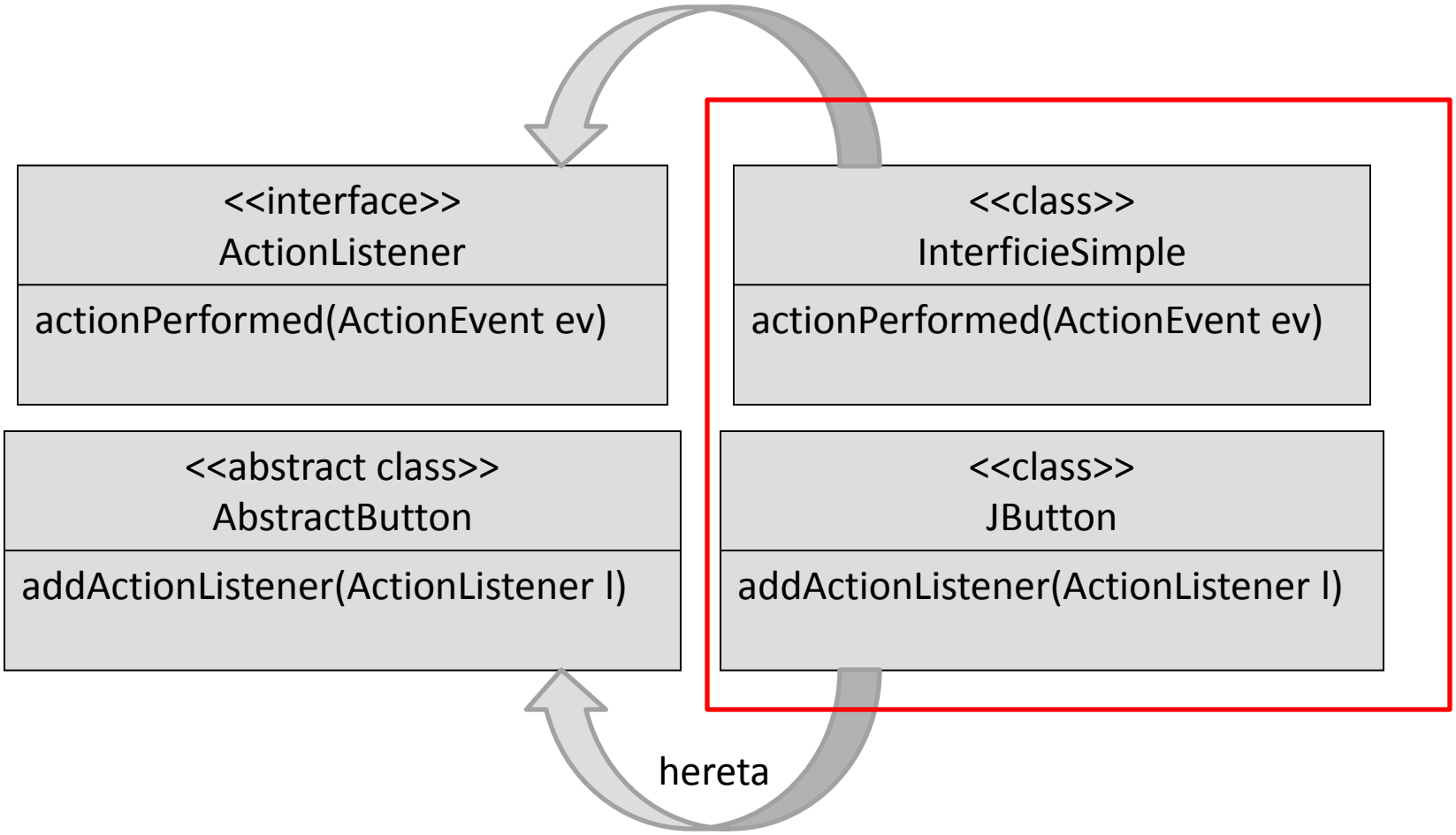
Aquí, afegim aquest objecte a la llista de listeners.

Aquest mètode s'invoca quan una acció succeeix.

Aquí és on faig el que s'ha de fer quan l'acció succeeix

Comentaris

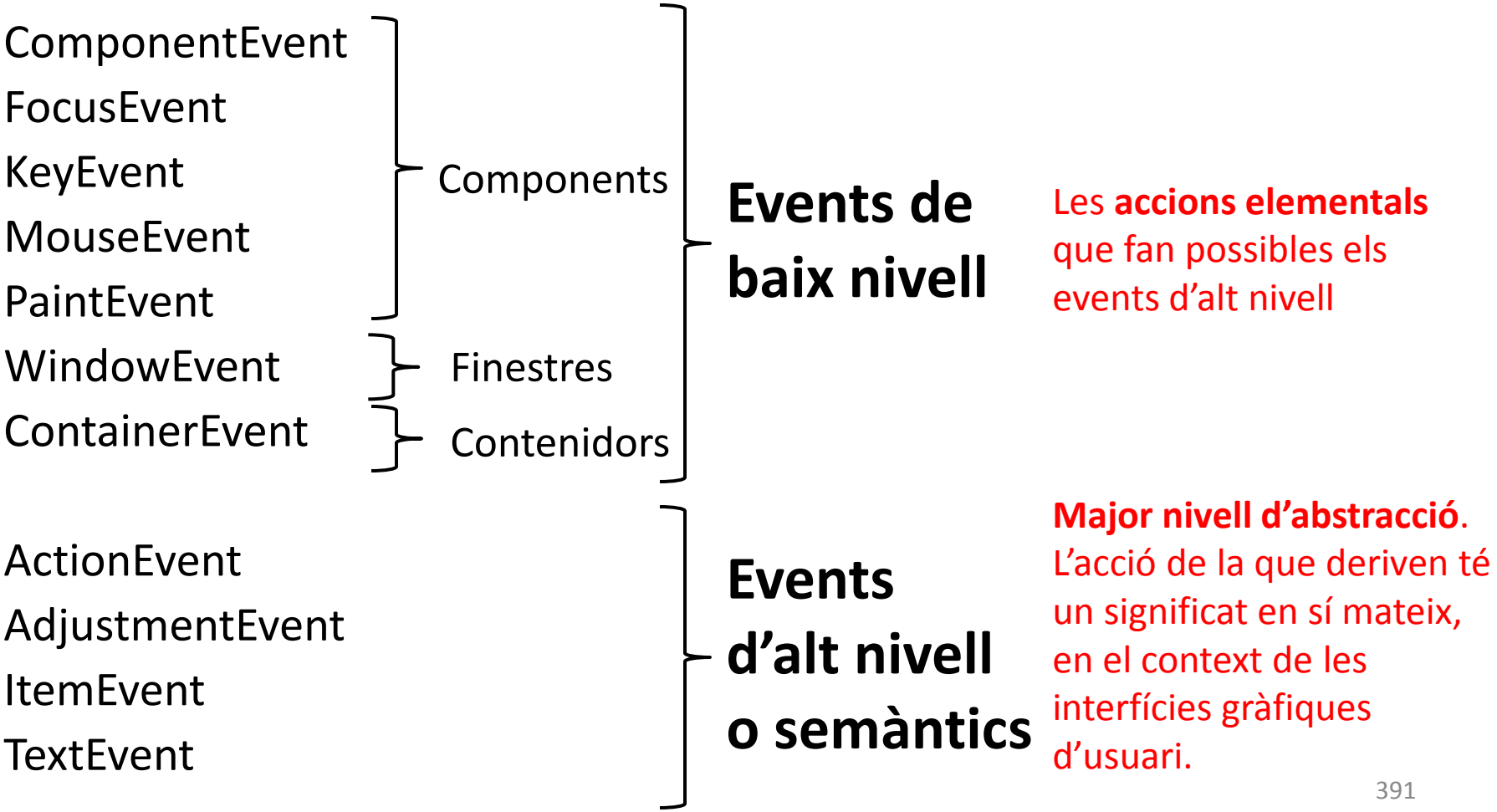
implementa



EVENTS I LISTENERS

Events AWT

En Java els event poden ser d'alt o de baix nivell.



Events AWT

En Java els event poden ser d'alt o de baix nivell.

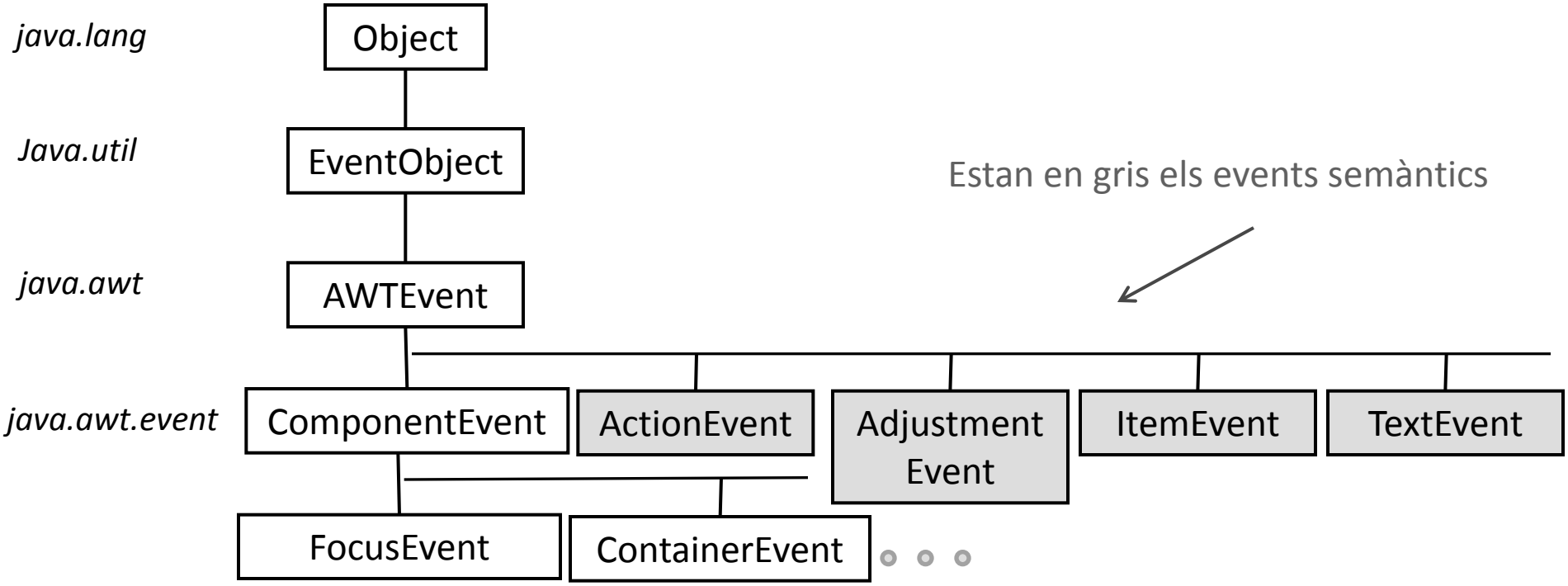
ComponentEvent
FocusEvent
KeyEvent
MouseEvent
PaintEvent
WindowEvent
ContainerEvent

Es produeixen amb les operacions elementals del ratoli, teclat, containers i windows.

ActionEvent (*clicar sobre botons o escollir comandos en menús)*
AdjustmentEvent (*canviar valors en barres de desplaçament*).
ItemEvent (seleccionar/desseleccionar valors)
TextEvent (canviar el text)

Jerarquia d'Events

Tots els events de Java són objectes de classes que pertanyen a una determinada jerarquia de classes:



Componentes i events soportats per el AWT de Java

Tots aquests events també es poden produir en les subclasses de Component

Tenen relació amb el context que defineix la component

Component	Eventos generados	Significado
Button	ActionEvent	Clicar en el botón
Checkbox	ItemEvent	Seleccionar o deseleccionar un item
CheckboxMenuItem	ItemEvent	Seleccionar o deseleccionar un item
Choice	ItemEvent	Seleccionar o deseleccionar un item
Component	ComponentEvent	Mover, cambiar tamaño, mostrar u ocultar un componente
	FocusEvent	Obtener o perder el focus
	KeyEvent	Pulsar o soltar una tecla
	MouseEvent	Pulsar o soltar un botón del ratón; entrar o salir de un componente; mover o arrastrar el ratón (tener en cuenta que este evento tiene dos Listener)
Container	ContainerEvent	Añadir o eliminar un componente de un container
List	ActionEvent	Hacer doble click sobre un item de la lista
	ItemEvent	Seleccionar o deseleccionar un item de la lista
MunuItem	ActionEvent	Seleccionar un item de un menú
Scrollbar	AdjustementEvent	Cambiar el valor de la scrollbar
TextComponent	TextEvent	Cambiar el texto
TextField	ActionEvent	Terminar de editar un texto pulsando Intro
Window	WindowEvent	Acciones sobre una ventana: abrir, cerrar, iconizar, restablecer e iniciar el cierre

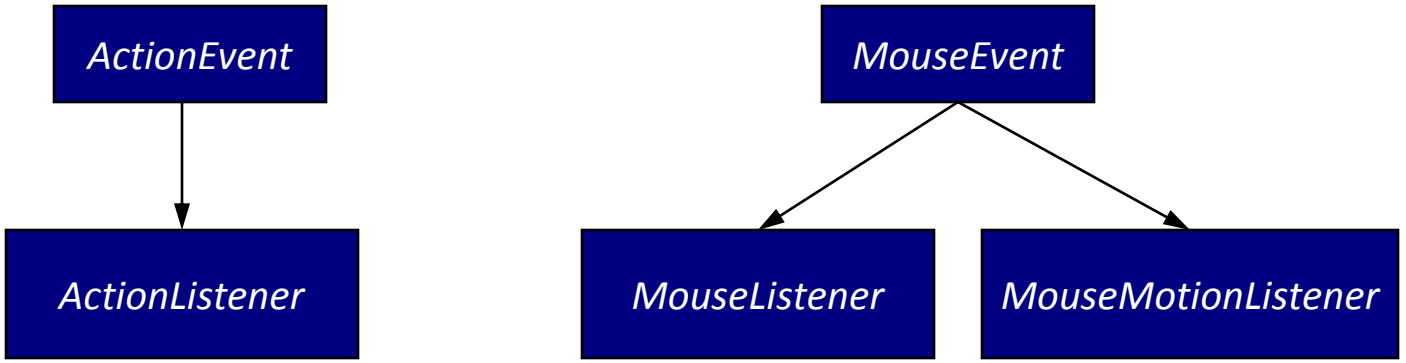
Tabla 5.1. Componentes del AWT y eventos específicos que generan.

Events semàntics

- No són disparats per tots els components
- Exemple 1: **ItemEvent**
  - Disparat per JComboBox
  - No disparat per JButton
- Exemple 2: **ActionEvent**
  - Disparat per JComboBox
  - Disparat per JButton

Escoltadors (Listeners) d'events

- Interfícies que manipulen els events (java.util.EventListener).
- Cada classe Event té la seva corresponent interfície Listener
- Pot haver-hi varios Listeners per al mateix tipus d'events



Escoltadors (Listeners) d'events

- Cada listener és un objecte que implementa la interfície corresponent al tipus d'event a escoltar:
  - –ActionListener
  - –WindowListener
  - –MouseListener
  - –KeyListener
  - –FocusListener
  - –Altres...

Events i Interfícies Listener

- Cada tipus d'event té una interfícies Listener associada:

ComponentEvent	ComponentListener
FocusEvent	FocusListener
KeyEvent	KeyListener
MouseEvent	MouseListener,MouseMotionListener
WindowEvent	WindowListener
ContainerEvent	ContainerListener
ActionEvent	ActionListener
AdjustmentEvent	AdjustmentListener
ItemEvent	ItemListener
TextEvent	TextListener

Listeners

Exemples de Listeners:

```
public interface ActionListener extends EventListener {
        public void actionPerformed(ActionEvent e);
}
```

```
public interface ItemListener extends EventListener {
        public void itemStateChanged(ItemEvent e);
}
```

```
public interface MouseListener extends EventListener {

    public void mouseClicked(MouseEvent e);

    public void mousePressed(MouseEvent e);

    public void mouseReleased(MouseEvent e);

    public void mouseEntered(MouseEvent e);

    public void mouseExited(MouseEvent e);

}
```

Listeners

```
public interface WindowListener extends EventListerner {
        void windowActivated(WindowEvent e);
        void windowClosed(WindowEvent e);
        void windowClosing(WindowEvent e);
        void windowDeactivated(WindowEvent e);
        void windowDeiconified(WindowEvent e);
        void windowIconified(WindowEvent e);
        void windowOpened(WindowEvent e);
}
```

```
public interface ComponentListener extends EventListener {
            public void componentHidden(ComponentEvent e);
            public void componentMoved(ComponentEvent e);
            public void componentResized(ComponentEvent e);
            public void componentShown(ComponentEvent e);
}
```

Registre de Listeners

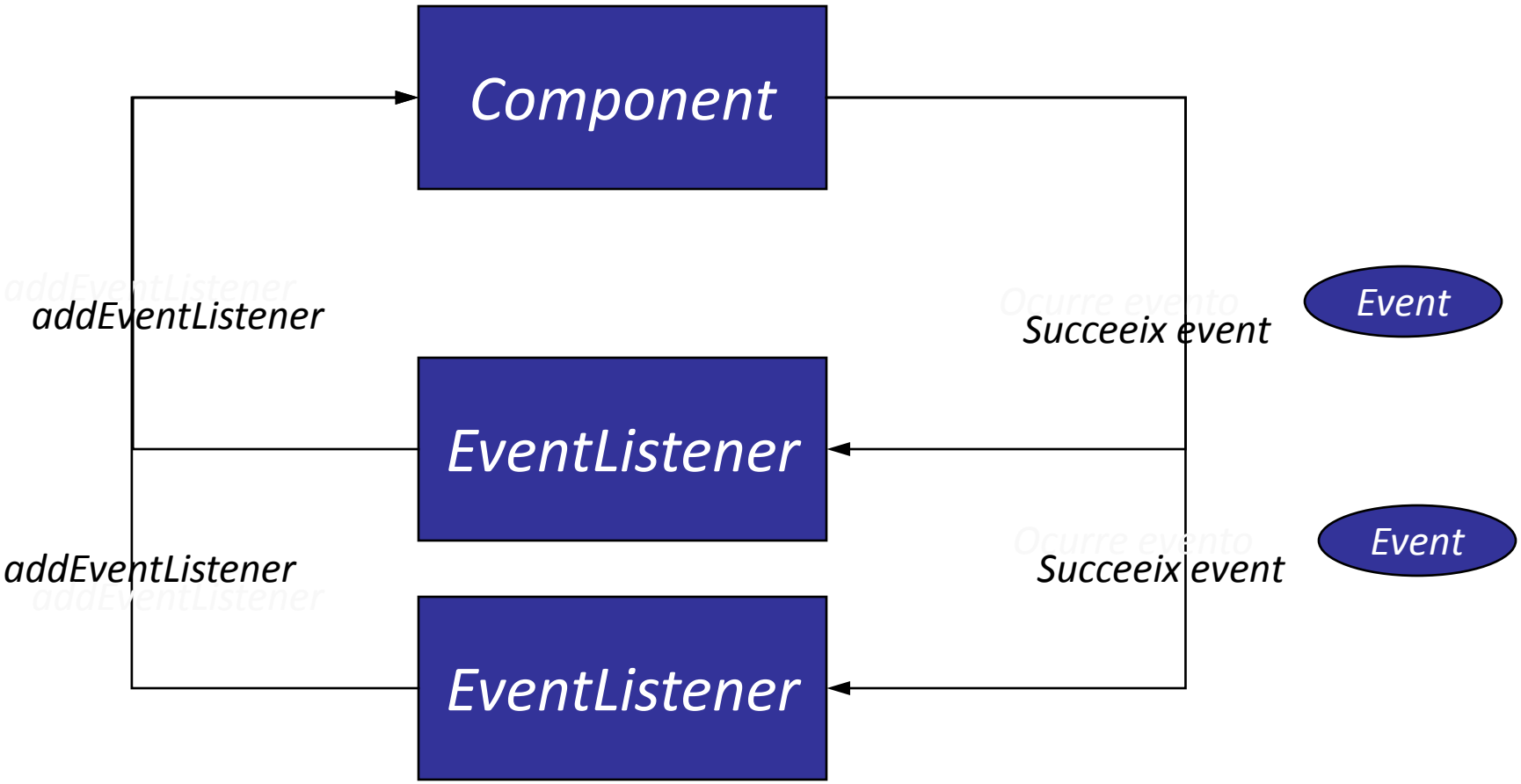
- Un Listener, en primer lloc, ha de registrar-se amb la/les font/s que pugui/n generar els events del seu interés. Ho faran mitjançant un mètode de la forma:

  public void addXXXListener(XXXListener e)

  Per exemple: addActionListener, addItemListener, …

- Per al mateix event en un component, pot haver diversos Listeners registrats
  - Un event pot provocar nombroses respostes
  - Els events són passats a tots els seus Listeners

Múltiples Listeners



COMPONENTS

Components

Les components estan classificades com

1. Contenidor d'alt nivell
2. Contenidor intermedi
3. Contenidor específic
4. Control bàsic
5. Displays no editables
6. Displays interactius

http://java.sun.com/docs/books/tutorial/uiswing/components/index.html

Contenidors d'alt nivell

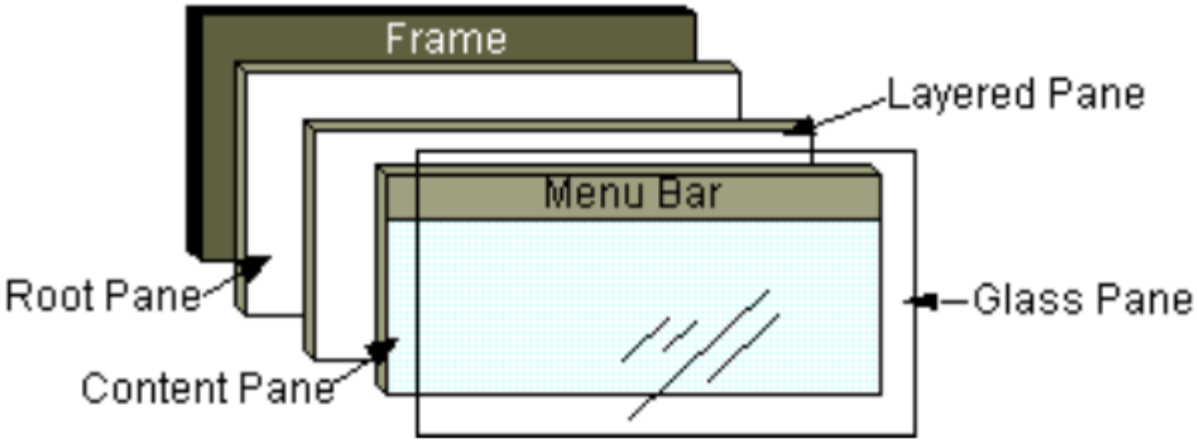
- JFrame - Frames o Marcos (finestres)
  - Finestra principal
- JDialog – Diàlegs
  - Una finestra independent que ens servirà per a proporcionar informació temporal a la finestra principal de l'aplicació. La majoria serveixen per mostrar un missatge d'error o warnings als usuaris, però poden mostrar imatges, arbres de directoris, etc.

- JApplet – Applets
  - Una component d'una aplicació que s'executa en el context d'un altre programa, per exemple un navegador web.

(http://download.oracle.com/javase/tutorial/deployment/TOC.html)

JRootPane

- El JRootPane es crea quan s'instancia un dels contenidors d'alt nivell de Swing.

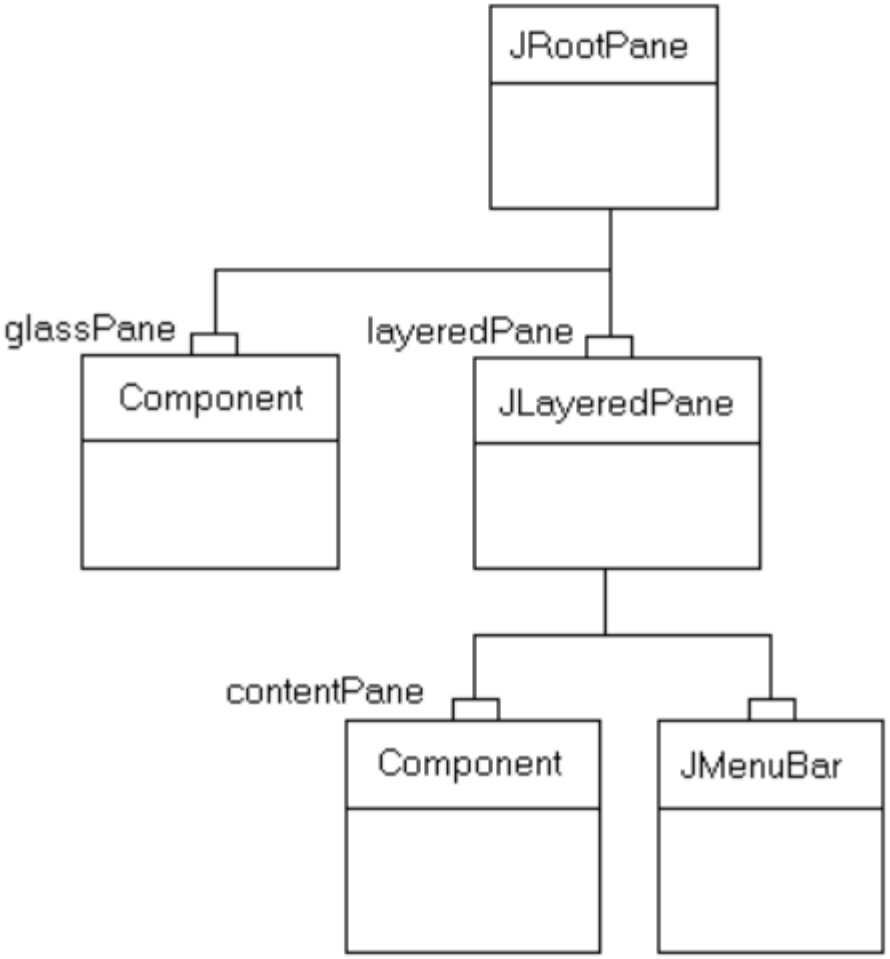


http://download.oracle.com/javase/tutorial/uiswing/components/rootpane.html

Jerarquia de JRootPane

- Totes els contenidors d'alt nivell deleguen les seves operacions a un JRootPane
- Per afegir components al JRootPane, s'afegeix l'objecte al contentPane del JRootPane de la següent manera:

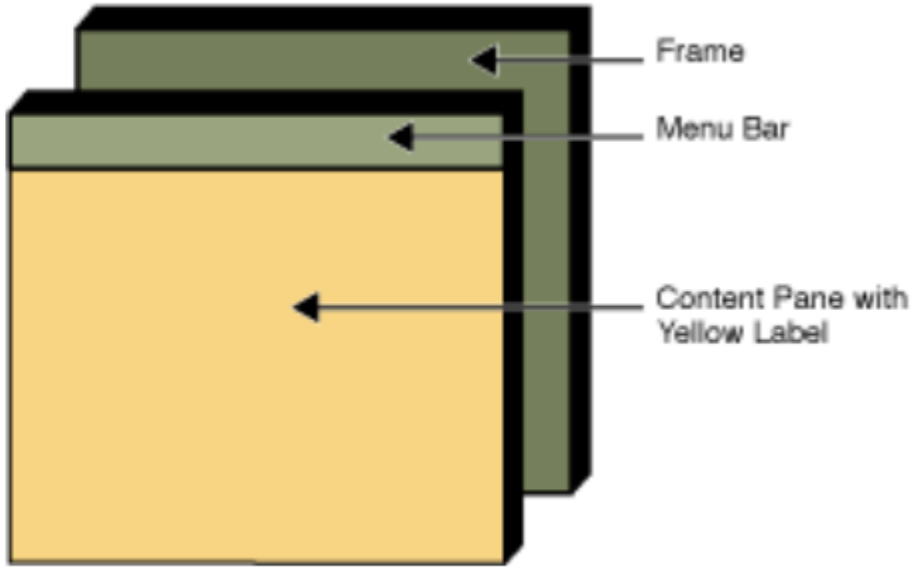
rootPane.getContentPane().add(child)



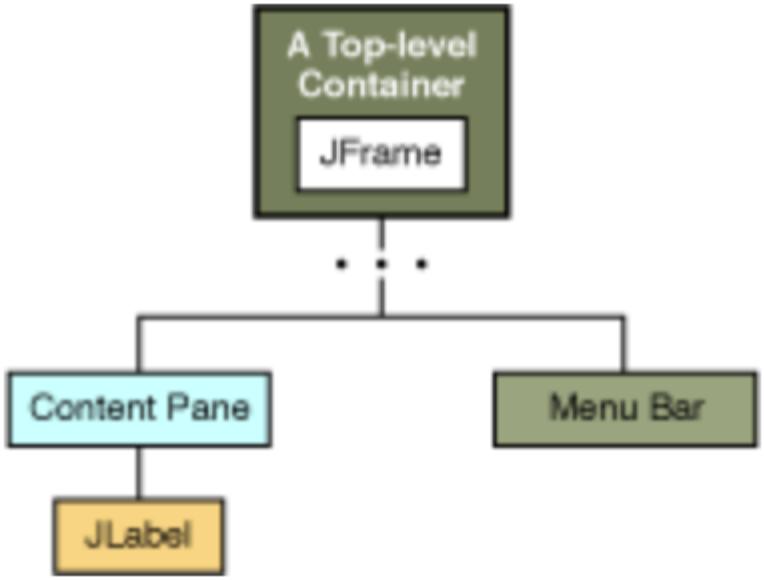
http://download.oracle.com/javase/1.4.2/docs/api/javax/swing/JRootPane.html

JRootPane

- El JRootPane es crea quan s'instancia un dels contenidors d'alt nivell de Swing.



http://download.oracle.com/javase/tutorial/uiswing/components/toplevel.html



http://download.oracle.com/javase/tutorial/uiswing/components/toplevel.html

Contenidors intermedis

- Panell (panel)
- Panell lliscants (scroll pane)
- Panell dividit (split pane)
- Panell amb solapes (tabbed pane)
- Barra d'eines (tool bar)
- Normalment, s'utilitzen per a agrupar components, perque les components estan relacionades o només perque agrupar-les fa que la distribució sigui més senzilla.
- Un panell pot fer servir qualsevol controlador de distribució (layout manager), ho veurem després.

Control bàsic

- Botons (Buttons),
- Caixes combo (Combo boxes),
- Barra lliscant (Sliders),...

Elements d'interfície que els usuaris poden manipular per prémer un botó, seleccionar una opció o fixar un valor.

Displays no editables

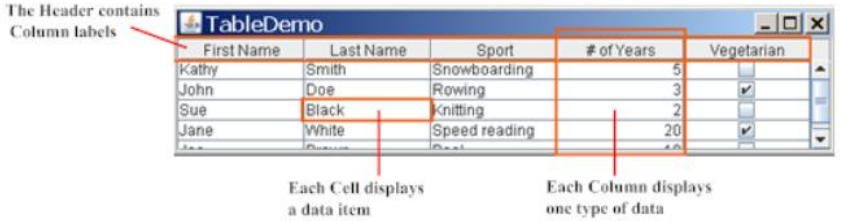
- Etiquetes (labels)
- Barres de progrés (progress bars)
- Pistes d'eines (tool tips)

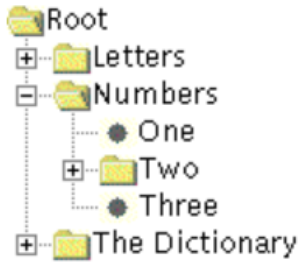
Per afegir una pista d'eina a un botó:

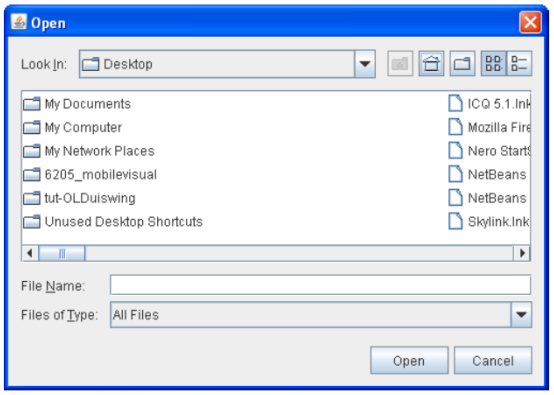
```
b1.setToolTipText("Clica aquest botó per desactivar el botó del mig.");
```

Displays interactius

- Selector de colors (JColorChooser)
- Taula (JTable)
- Text (JTextComponent)
- Selector de fitxers (JFileChooser)
- Arbres (JTree)



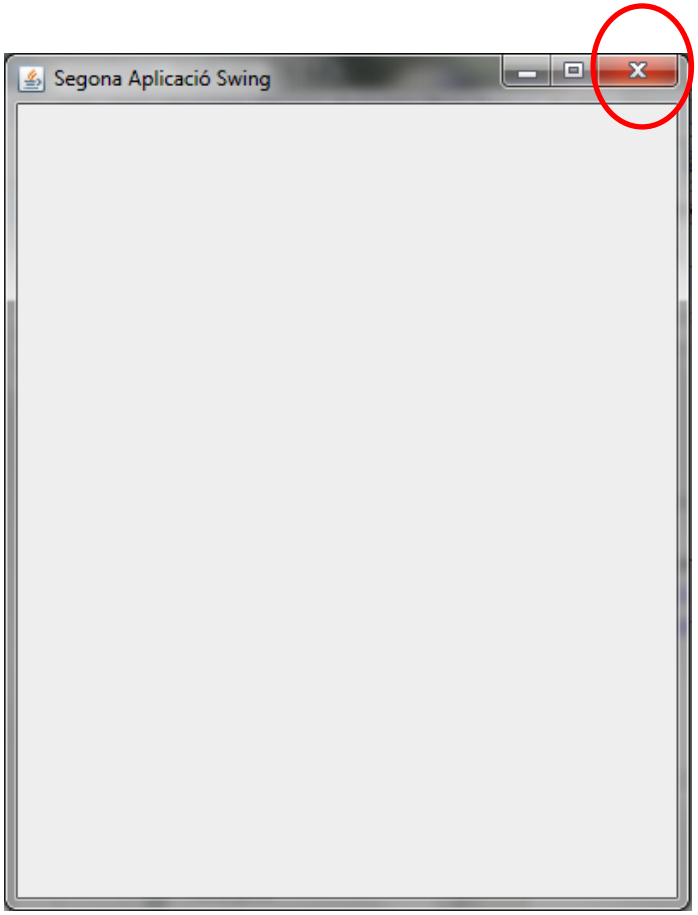




http://download.oracle.com/javase/tutorial/uiswing/components/

CLASSES ADAPTER I CALSSES INTERNES: EXEMPLE D'IMPLEMENTACIÓ D'UNA FINESTRA QUE ES TANCA.

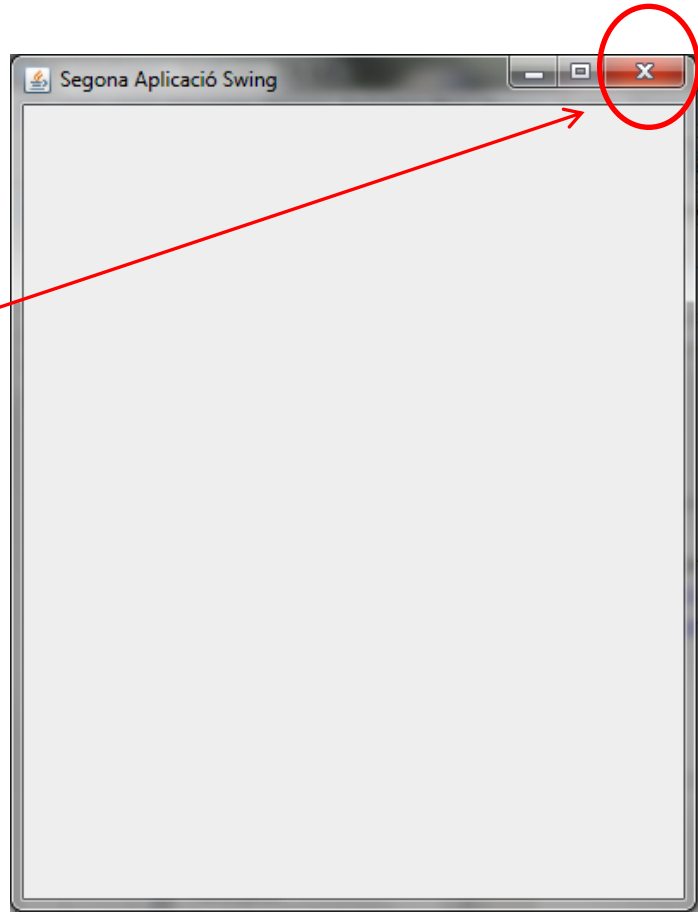
Exemple 3: FINESTRA que es tanca



Exemple 3: dos mètodes de creació de finestres

A l'exemple 1 vam crear una aplicació senzilla de dues maneres.

Però, en cap dels dos casos l'aplicació termina quan fem click en el botó de tancar de la finestra

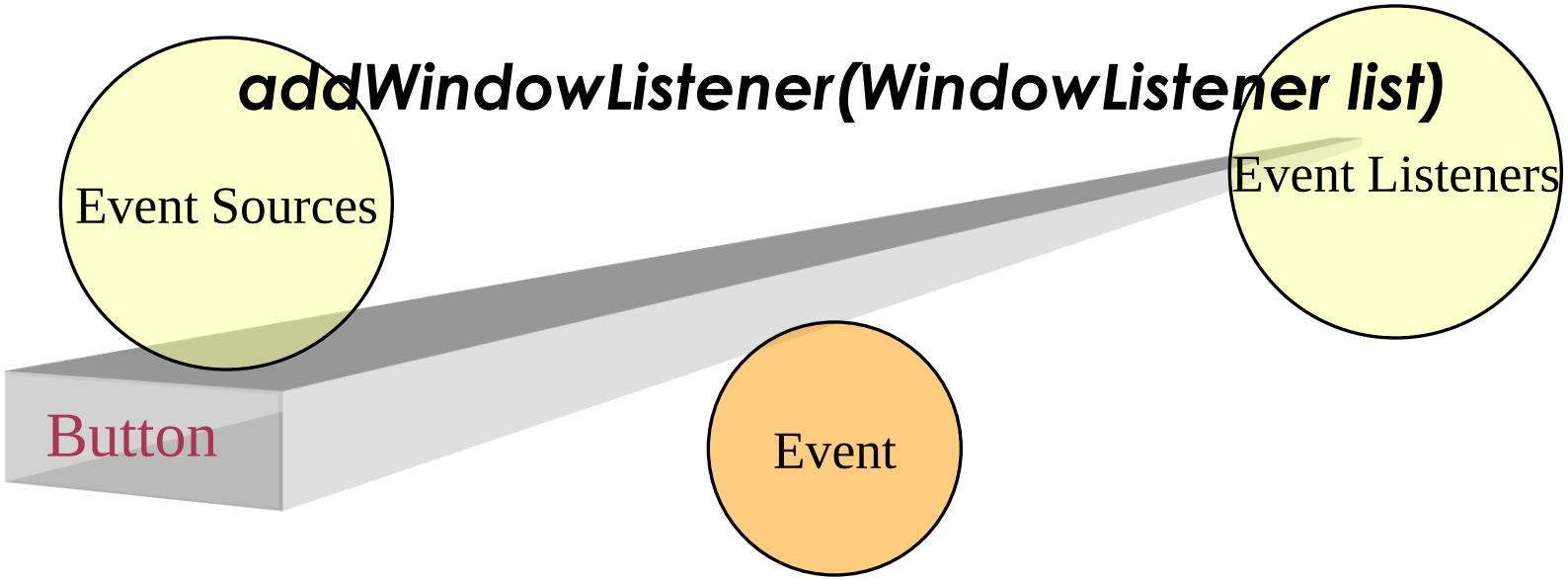


Caldrà relacionar el botó per tancar la finestra amb l'acció de terminar l'aplicació

Caldrà capturar els events que es llancen

Exemple 3: Afegim Listener

- Volem fer que l'aplicació termine quan es tanca la finestra
- Per això, definim un event i el gestionem.



- A continuació mostrem quatre implementacions possibles.

Recordem: WindowListener

```
public interface WindowListener extends EventListerner {
        void windowActivated(WindowEvent e);
        void windowClosed(WindowEvent e);
        void windowClosing(WindowEvent e);
        void windowDeactivated(WindowEvent e);
        void windowDeiconified(WindowEvent e);
        void windowIconified(WindowEvent e);
        void windowOpened(WindowEvent e);
}
```

Exemple 3 (a)

```
// Fitxer FinestraTancable.java
import java.awt.event.*;
import javax.swing.*;
class FinestraTancable extends JFrame implements WindowListener {
        // Constructor
        public FinestraTancable () {
                setTitle("La meva finestra tancable");
                setSize( 300, 200);
                // causa  events de window  per a ser enviat al objecte window listener
                addWindowListener(this);
          }
        // mètodes de la interfície WindowsListener
         public void windowActivated( WindowEvent e) {}
         public void windowDeactivated( WindowEvent e) {}
         public void windowIconified( WindowEvent e) {}
         public void windowDeiconified( WindowEvent e) {}
         public void windowOpened( WindowEvent e) {}
         public void windowClosed( WindowEvent e) {}
         // reesecriure  el mètode windowClosing  per sortir del programa
         public void windowClosing( WindowEvent e) {
           System.exit( 0); // sortida  normal
         }
} // Final de la classe FinestraTancable
class Main {
  public static void main( String[] args) {
    FinestraTancable finestra= new FinestraTancable();
    finestra.setVisible(true); // fa el JFrame visible
  }
} // Final de la classe Main
```

La classe implementa la interfície WindowsListener

La classe regisra l'objecte com a listener

mètodes de la interfície WindowsListener

Exemple 3 (b): Separant les classes

Hi ha 7 mètodes en la Interfície WindowListener. Aquí només ens interessem per l'event tancament de finestra

La interfície WindowListener és implementada per la classe MyWindowListener, així la seva instància pot respondre als events de la finestra en la qual ell va ser registrat.

// Fitxer FinestraTancable2.java

```
import java.awt.event.*;
import javax.swing.*;
class FinestraTancable2 extends JFrame {

        // Constructor
       public FinestraTancable2 () {
            setTitle("La meva finestra tancable");
            setSize( 300, 200);
            // causa  events de window  per a ser enviat al
            //objecte window listener
            addWindowListener( new MyWindowListener());
        }
} // Final de la classe FinestraTancable2
```

```
class MyWindowListener implements WindowListener {
  // Do nothing methods required by interface
  public void windowActivated( WindowEvent e) {}
  public void windowDeactivated( WindowEvent e) {}
  public void windowIconified( WindowEvent e) {}
  public void windowDeiconified( WindowEvent e) {}
  public void windowOpened( WindowEvent e) {}
  public void windowClosed( WindowEvent e) {}
   // reesecriure  el mètode windowClosing  per sortir del
programa
  public void windowClosing( WindowEvent e) {
    System.exit( 0); // normal exit
  }
} // Final de la classe MyWindowListener
```

Un objecte MyWIndowListener és registrat amb addWindowLister

```
class Main {
  public static void main( String[] args) {
        FinestraTancable2 finestra = new FinestraTancable2();
        finestra.setVisible(true); // fa el JFrame visible
  }
} //Final de la classe Main
```

Classes Adapter

- Són classes que implementen una interfície Listener amb mètodes buits ("dummy"), utilització herència.
- Útils només per a interfícies Listeners amb més d'un mètode
- Principalment, per raons de conveniència
- Exemples:
  - MouseAdapter
  - WindowAdapter

Exemple 3 (c): Classes Adapter

```
// Fitxer FinestraTancable3.java
import java.awt.event.*;
import javax.swing.*;
class FinestraTancable3 extends JFrame {
  // Constructor
    public FinestraTancable3() {
        setTitle("La meva finestra tancable");
        setSize( 300, 200);
        TancarFinestra tf = new TancarFinestra();
        this.addWindowListener(tf);
  }
} // fi de la classe FinestraTancable3
```

Un objecte TancarFinestra és registrat amb addWindowLister

Definim una classe auxiliar que hereta de WindowAdapter

```
class TancarFinestra extends WindowAdapter {
        public void windowClosing(WindowEvent we){
                System.exit(0); // normal exit
        }
} //Final de la classe TancarFinestra
```

```
class Main {
  public static void main( String[] args) {
        FinestraTancable3 finestra = new FinestraTancable3();
        finestra.setVisible(true); // fa el JFrame visible
  }
} //Final de la classe Main
```

Classes Adapter

- Desavantatge: Java no permet herència múltiple
- Solució: utilitzar classes internes anònimes

Exemple 3 (d): Classes internes anònimes de Java

```
// Fitxer FinestraTancable4.java

import java.awt.event.*;
import javax.swing.*;
class FinestraTancable4 extends JFrame {
 // Constructor
   public FinestraTancable4() {
        setTitle("La meva finestra tancable");
        setSize( 300, 200);
        this.addWindowListener(new WindowAdapter() {
                                    public void windowClosing() {
                                              finestraTancableaddWindowListener();}
                                   });
 }
private void  finestraTancableaddWindowListener(){
        System.exit(0);
}
} // Final de la classe FinestraTancable
```

No s'està creant un nou objecte de **WindowAdapter** (entre altres coses perquè la classe és **abstract**), sinó estenent la classe WindowAdapter, encara que la paraula **extends** no aparegui

```
class Main {
  public static void main( String[] args) {
        FinestraTancable4 finestra = new FinestraTancable4();
        finestra.setVisible(true); // fa el JFrame visible
  }
} //Final de la classe Main
```

Classes internes

- En Java una classe pot ser definida en qualsevol lloc
  - Niuada dins d'altres classes
  - En la invocació d'un mètode
- Tenen accés als membres i mètodes de totes les classes externes a elles
- Poden tenir noms o ser anònimes
- Poden estendre d'una altra classe o implementar interfícies
- Molt útils per a la manipulació d'events

Classes internes amb nom

- Es defineixen com les classes normals
- No poden ser ***public***

Ha d'estar dins de la classe si volem que tingui accés als mètodes de la classe externa.

```
public class ApplicationFrame {
        ….
        class ButtonHandler implements ActionListener {
                public void actionPerformed(ActionEvent e) {
                        doTheOKThing();
                }
         }
private void doTheOKThing() { // here's where I handle OK
}
….
        JButton okButton = new JButton("OK");
        okButton.addActionListener(new ButtonHandler());  // create inner class listener
        ….
```

Classes internes amb nom

```
public class MyClass extends JPanel {
…
anObject.addMouseListener(new MyAdapter());
        …
        class myAdapter extends MouseAdapter {
                public void mouseClicked(MouseEvent e) {
                        // blah
                } // end mouseClicked
        } // end inner class
} // end MyClass
```

Classes internes anònimes

- Definida dins de addXXXListener:
  (new *className*( ) { *classBody* });
  (new *interfaceName*() { *classBody* });
- Dins del cos no pot definir constructors.

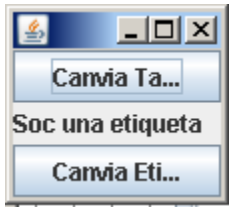
```
public class ApplicationFrame {
….
        JButton okButton = new JButton(“OK”);
        okButton.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent event) {
                        doTheOKThing();
                }
        } );
        ….
        private void doTheOKThing() {
                // here’s where I handle the OK
        }
….
```

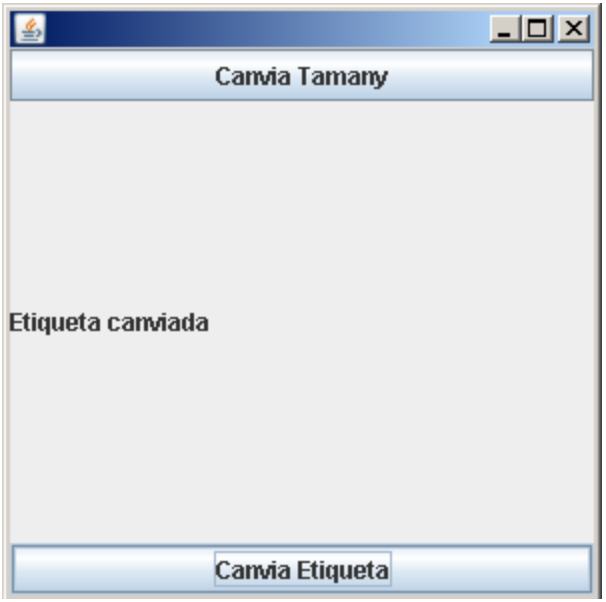
Exemple

- Com obtenim events ActionEvent per a dos botons quan cada botó necessita fer una cosa diferent?

Classes internes

Exemple 4: Com obtenim events ActionEvent per a dos botons quan cada botó necessita fer una cosa diferent?

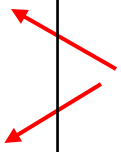




Exemple 4: Opició A

- Implementar dos mètodes actionPerformed()

```
class LaMevaGUI implements ActionListener {
  // codi aqui
       public void actionPerformed(ActionEvent ev) {
          frame.setSize(300, 300);
       }
      public void actionPerformed(ActionEvent ev) {
          etiqueta.setText("Etiqueta canviada");
       }
} // fi de la classe LaMevaGUI
```



Però això és impossible

Exemple 4: Opició B

- Registrar el mateix listener amb dos botons

```
class LaMevaGUI implements ActionListener {
  // declarem variables aqui
      public void go() {
            //construir gui
            boto1 = new JButton();
            boto2 = new JButton();
            boto1.addActionListener(this);
            boto2.addActionListener(this);
            // més codi aquí
      }
      public void actionPerformed(ActionEvent ev) {
            if(ev.getSource() == boto1 ){
                  frame.setSize(300, 300);
            }else{
                  etiqueta.setText("Etiqueta canviada");
            }
      }
} // fi de la classe LaMevaGUI
```

Registre'm el mateix listener

Mirem quin event és

Opció correcta, però pot arribar a ser complicada de gestionar

Exemple 4: Opició C

- Crear dues classes ActionListener separades

```
class LaMevaGUI  {
      JFrame frame;
      JLabel etiqueta;
      void gui() {
            // codi per instanciar els dos listeners i registrar
            // un amb el boto1 i un altre amb el boto2
      }
} // fi de la classe
```

```
class Boto1Listener implements ActionListener {
      public void actionPerformed(ActionEvent ev) {
            frame.setSize(300, 300);
      }
} // fi de la classe
```

```
class Boto2Listener implements ActionListener {
      public void actionPerformed(ActionEvent ev) {
            etiqueta.setText("Etiqueta canviada");
      }
}// fi de la classe
```

Problema! Aquesta classe no té referència a la variable etiqueta

```
import java.awt.BorderLayout;
import javax.swing.*;
import java.awt.event.*;

public class LaMevaGUI {
  JFrame frame;
  JLabel etiqueta;
  public static void main(String[] args){
            LaMevaGUI gui = new LaMevaGUI();
            gui.go();
   }
  public void go() {
   frame = new JFrame();
   frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   JButton botoTamany = new JButton("Canvia Tamany");
   botoTamany.addActionListener(new BotoTamanyListener());
   JButton botoEtiqueta = new JButton("Canvia Etiqueta");
   botoEtiqueta.addActionListener(new BotoEtiquetaListener());

   etiqueta = new JLabel("Soc una etiqueta");

   frame.getContentPane().add(BorderLayout.NORTH, botoTamany);
   frame.getContentPane().add(BorderLayout.SOUTH, botoEtiqueta);
   frame.getContentPane().add(BorderLayout.CENTER, etiqueta);

   frame.setSize(100, 100);
   frame.setVisible(true);
}// Fi mètode go
```

Classes internes

```
class BotoTamanyListener implements ActionListener
    {
      public void actionPerformed(ActionEvent ev) {
          frame.setSize(300, 300);
       }
    }
    class BotoEtiquetaListener implements
     ActionListener {
      public void actionPerformed(ActionEvent ev) {
         etiqueta.setText("Etiqueta canviada");
      }
    }
} // Fi de la classe LaMevaGUI
```

Consells

- S'ha d'implementar tots els mètodes de la interfície
- No hi ha garantia de quin Listener és notificat primer. No escriure codi contant amb un ordre específic.

Repàs

Resum dels passos que es poden seguir per a crear una aplicació interactiva

senzilla orientada a events, amb interfície gràfica d'usuari:

1. Determinar els **components** que van a constituir la interfície d'usuari (botons, caixes de text, menús, etc.).
2. Crear una classe per a l'aplicació que contingui la funció **main**().
3. Crear una classe **Finestra**, que hereti de **JFrame**, i que respongui als events
4. La funció **main**() haurà de crear un objecte de la classe **Finestra** i mostrar-la per pantalla amb el tamany i posició adequades.
5. Afegir a l'objecte **Finestra** tots els components que ha de contenir.
6. Definir els objectes **Listener** (objectes que s'ocuparan de respondre als events, les classes dels quals implementen les diferents interfícies Listener) per a cada un dels events que han d'estar suportats.
8. Finalment, s'han d'implementar els mètodes de les interfícies **Listener** que es faran càrrec de la gestió dels events.

LAYOUT MANAGER

Layout Manager

- Controla les components que estan contingudes en una altra component associada al Layout Manager

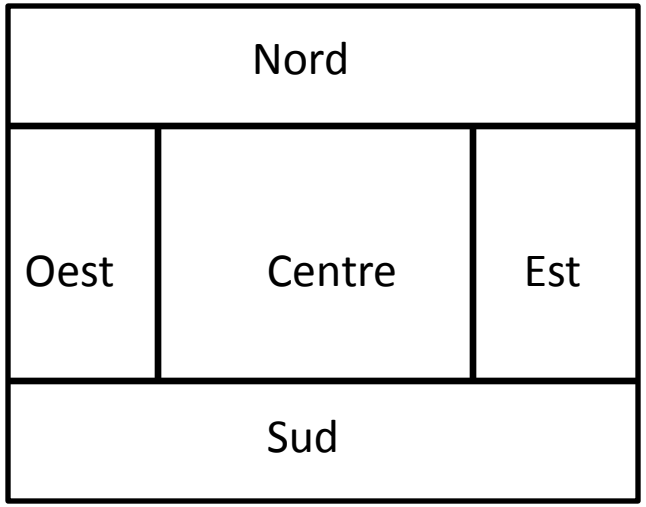
Exemple:

Si un frame conté un panell que conté un botó, el layout manager del panell controlarà el tamany i posició del botó, mentre que el layout manager del frame controlarà el tamany i posició del panell. El botó no necessitarà un layout manager.

BorderLayout

- Divideix el fons del component en 5 regions.
- Es pot afegir només una component per regió.
- Normalment, no s'aconsegueixen els tamanys preferits pels components.
- És el layout manager per defecte per a un **frame**

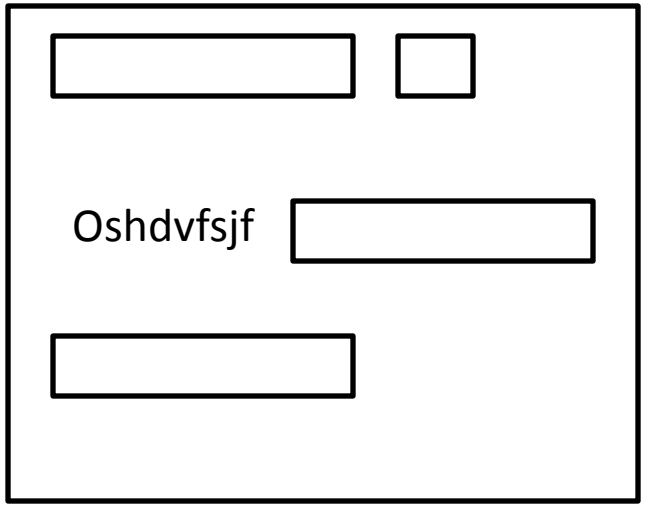
Exemple: BorderLayout



FlowLayout

- Actua com un processador de text, amb components en lloc de paraules
- Cada component és del tamany predefinit i es van afegint d'esquerra a dreta en l'ordre que són afegits
- És el layout manager per defecte per a un **panel**

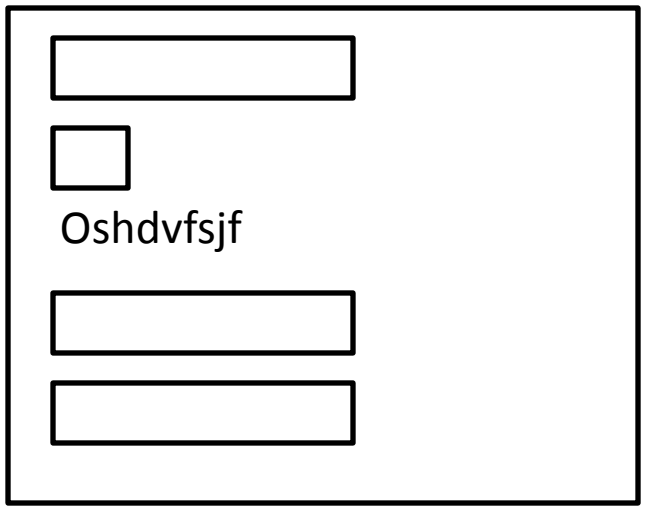
Exemple: FlowLayout



BoxLayout

- És com FlowLayout en que cada component és del tamany predefinit i en que es van col·locant en l'ordre que són afegits.
- Però, BoxLayout pot apilar els components verticalment (o horitzontalment).
- En lloc de tindre un wrapping automàtic , es pot forçar a començar una nova línia.

Exemple: BoxLayout



Canvi de Layout

- Per fer un canvi de Layout manager:

```
rootPane.getContentPane().setLayout(new BoxLayout());
```

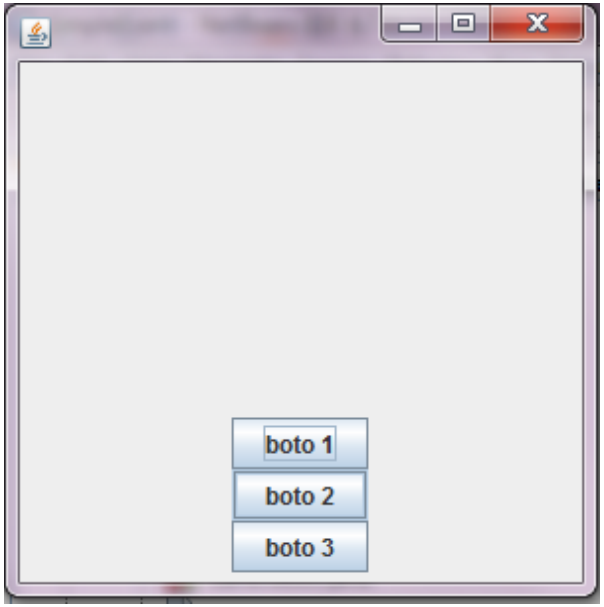
Exemple

```
public class ProvaLayout {
  public static void main (String[] args) {
      ProvaLayout gui = new ProvaLayout();
      gui.go();
   }
   public void go(){
      JFrame frame = new JFrame();

      JPanel panelA = new JPanel();
      JPanel panelB = new JPanel();

      panelB.setBackground(Color.red);

      panelB.add(new JButton("boto 1"));
      panelB.add(new JButton("boto 2"));
      panelB.add(new JButton("boto 3"));
      panelA.add(panelB);

      frame.getContentPane().add(BorderLayout.SOUTH,panelA);

      frame.setSize(300,300);
      frame.setVisible(true);
   }
}
```

Posició seleccionada a la distribució del frmae

Exemple

```
public class ProvaLayout {
  public static void main (String[] args) {
      ProvaLayout gui = new ProvaLayout();
      gui.go();
   }
   public void go(){
      JFrame frame = new JFrame();

      JPanel panelA = new JPanel();
      JPanel panelB = new JPanel();

      panelB.setBackground(Color.red);
      panelB.setLayout(new BoxLayout(panelB, BoxLayout.Y_AXIS));
      panelB.add(new JButton("boto 1"));
      panelB.add(new JButton("boto 2"));
      panelB.add(new JButton("boto 3"));
      panelA.add(panelB);

      frame.getContentPane().add(BorderLayout.SOUTH,panelA);

      frame.setSize(300,300);
      frame.setVisible(true);
   }
}
```

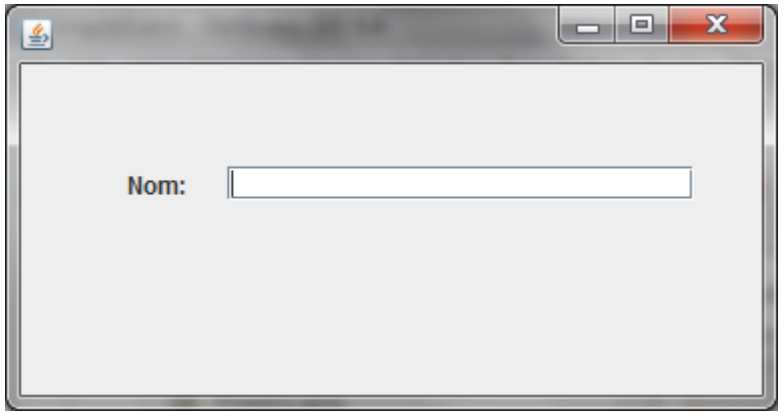


MÉS SOBRE SWING COMPONENTS: EXEMPLES

JTextField

JTextField camp = new JTextField(20); ← 20 columnes

JTextField camp = new JTextField("El meu missatge");



- Mètodes importants:
- String getText() i setText(String str)
  Permeten establir o obtenir el text del component
- addActionListener(ActionListener al),
- removeActionListener(ActionListener al)
  Permet registrar o borrar l'objecte que gestionarà l'event
- selectAll(), select(int start, int end)
  Selecciona tot o part del text
- requestFocus() (de la classe Component)
  Permet fer des del programa que un component obtinga el Focus.

JTextArea

- Pot contenir més d'una línia de text
- Per afegir barres lliscants s'ha d'afegir un ScrollPane

- Constructor:

10 linies 20 columnes

L'ample d'una columna és igual a l'ample d'un caràcter, en la font particular que s'està utiltzant.

```
JTextArea text = new JTextArea(10, 20);
JScrollPane scroller= new JScrollPane(text);
text.setLineWrap(true);
scroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALLWAYS);
scroller.setVerticalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
panel.add(scroller);
text.setText("Canviem el text");
text.append("botó apretat");
text.selectAll();
text.requestFocus();
```

JTextArea

- Afegir-li una scrollbar vertical:

```
JScrollPane scroller = new JScrollPane (list);
scroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
scroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
panel.add(scroller);
```

Exemple 1

- Implementeu una interfície gràfica d'usuari que contingui un botó i una àrea de text. Cada vegada que es prem el botó s'ha d'escriure el text "botó apretat" a l'àrea de text.

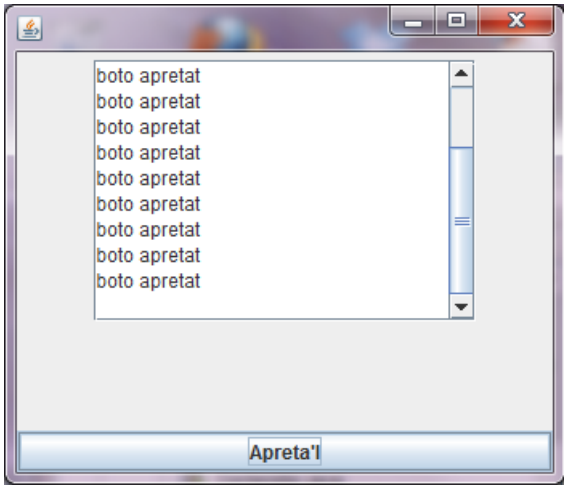
```
public class ExempleJTextArea implements ActionListener{
    JTextArea text;
    public static void main (String [] args){
       ExempleJTextArea gui  = new ExempleJTextArea();
       gui.go();
    }
    public void go(){
       JFrame frame = new JFrame();
       JPanel panel = new JPanel();
       JButton boto = new JButton("Apreta'l");
       boto.addActionListener(this);
       text = new JTextArea(10,20);
       text.setLineWrap(true);
       JScrollPane scroller = new JScrollPane(text);
       scroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
       scroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
       panel.add(scroller);
       frame.getContentPane().add(BorderLayout.CENTER, panel);
       frame.getContentPane().add(BorderLayout.SOUTH, boto);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.setSize(350,300);
       frame.setVisible(true);
    }
    public void actionPerformed(ActionEvent ev){
       text.append("boto apretat \n");
    } } // Fi clase ExempleJTextArea
```

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.ScrollPaneConstants;
```

ajusta el text al tamany de l'area de text

Barra lliscant

Concatena el text (no borra el que hi ha escrit)



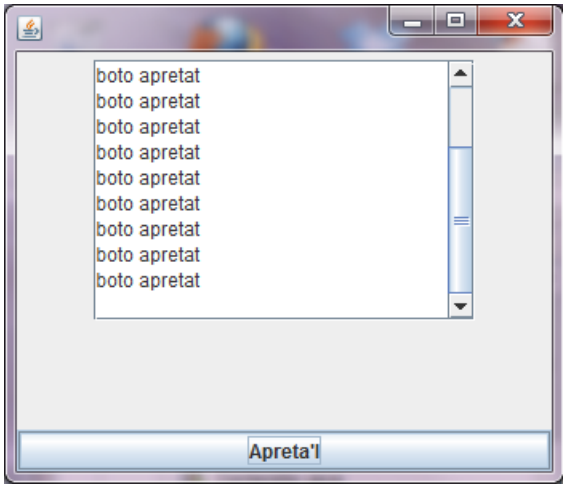
Exemple 1: una altra opció

- També es pot crear una classe **BotoListener** que implementi la interfície ActionListener.
- En aquest exemple, aquesta classe haurà de ser interna.

```
public class ExempleJTextArea 2{
    JTextArea text;
   public static void main(String [] args){
      ExampleJTextArea e = new ExampleJTextArea();
      e.go();
}
   public void go(){
      JFrame frame = new JFrame();
      JPanel panel = new JPanel();
      //JPanel panel = new JPanel(new BorderLayout()); //PREFERRED!
      JButton boto = new JButton("Apreta'l");
      boto.addActionListener(new BotoListener());
      text = new JTextArea(10,20);
      JScrollPane scroller = new JScrollPane(text);
      text.setLineWrap(true);
      scroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
      scroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
      panel.add(scroller);
      frame.getContentPane().add(BorderLayout.CENTER, panel);
      frame.getContentPane().add(BorderLayout.SOUTH, boto);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(350,300);
      frame.setVisible(true);}
   class BotoListener implements ActionListener{
      public void actionPerformed(ActionEvent ev){
          text.append("botó apretat \n");
        }
     }
} // Fi clase ExempleJTextArea
```

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.ScrollPaneConstants;
```

Classe interna



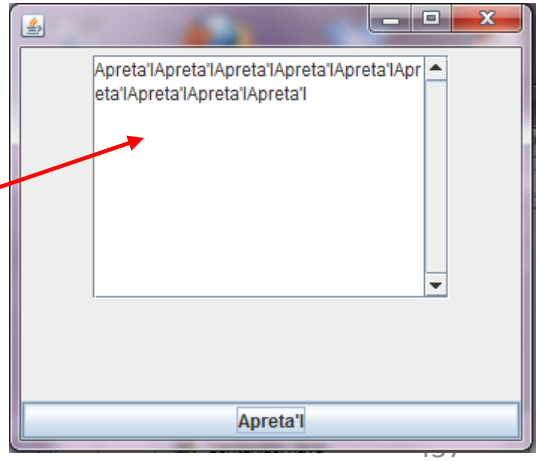
Exemple 2

- Implementeu una interfície gràfica d'usuari que contingui un botó i una àrea de text. Quan es prem el botó s'ha d'escriure el text contingut en ell a l'àrea de text.

```
public class Exemple2 implements ActionListener{
    JTextArea text;
    JButton boto;
    public static void main (String [] args){
       ExempleJTextArea gui  = new ExempleJTextArea();
       gui.go();
    }
    public void go(){
       JFrame frame = new JFrame();
       JPanel panel = new JPanel();
       boto = new JButton("Apreta'l");
       boto.addActionListener(this);
       text = new JTextArea(10,20);
       text.setLineWrap(true);
       JScrollPane scroller = new JScrollPane(text);
       scroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
       scroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
       panel.add(scroller);
       frame.getContentPane().add(BorderLayout.CENTER, panel);
       frame.getContentPane().add(BorderLayout.SOUTH, boto);
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
       frame.setSize(350,300);
       frame.setVisible(true);       }
    public void actionPerformed(ActionEvent ev){
            text.append(boto.getText());
    } } // Fi clase ExempleJTextArea
```

La declaració del botó s'ha de fer aquí

```
import java.awt.BorderLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JTextArea;
import javax.swing.ScrollPaneConstants;
```



JCheckBox

- Constructor:
  JCheckBox check = new JCheckBox(“Reproducció aleatòria”);

- Escoltar a un item event (quan es selecciona o desselecciona):
  check.addItemListener(this);

- Manejar l’event (i comprovar si està seleccionat o no)

```
public void itemStateChanged(ItemEvent ev){
    String onOrOff = ‘”off”;
    if (check.isSelected()) onOrOff =“on”;
    System.out.println(“Check box està” + onOrOff );
}
```

- Seleccionar-ho o desseleccionar-ho:

```
check.setSelected(true);
check.setSelected(false);
```

JList

- Constructor:

  String [] entradesLlista = {“alpha”, “beta”, “gamma”};

  JList list = new JList(entradesLlista);

- Afegir-li una scrollbar vertical:

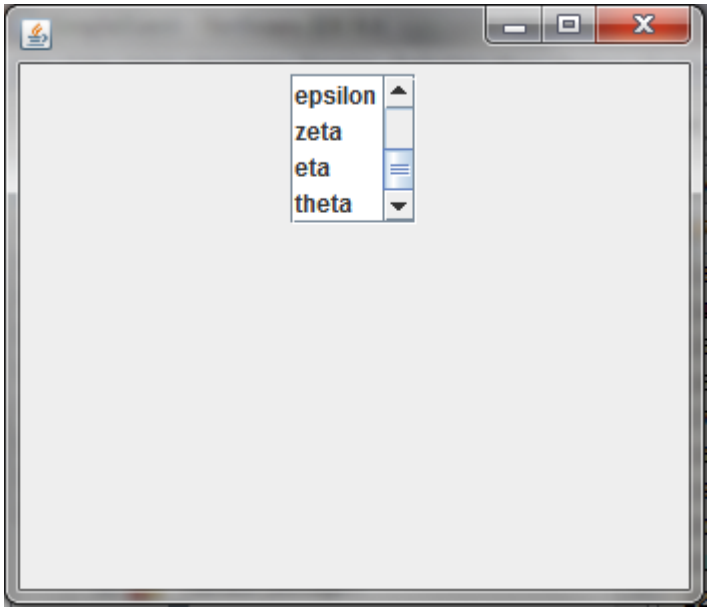
  JScrollPane scroller = new JScrollPane (list);

  scroller.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);

  scroller.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);

Exemple 3

- Implementeu una interfície gràfica d'usuari que conté una llista amb els Strings: "alpha", "beta", "gamma", "delta", "epsilon", "zeta","eta", "theta" i que quan seleccionem algun dels elements d'aquesta llista s'imprimeix per pantalla.



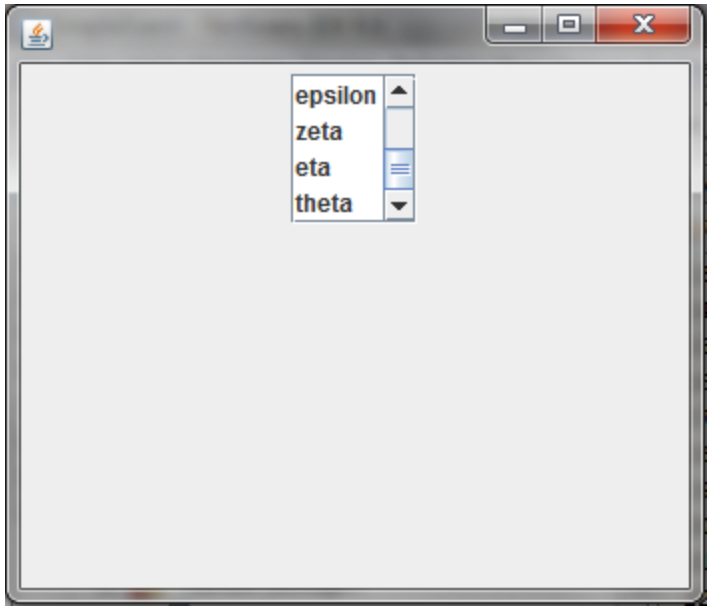
alpha
beta
alpha
gamma
delta
alpha

Exemple 3

```
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.ListSelectionModel;
import javax.swing.ScrollPaneConstants;
import javax.swing.event.ListSelectionEvent;
import javax.swing.event.ListSelectionListener;
```

Exemple 3

```
public class ExempleJList implements ListSelectionListener{
   String [] entradesLlista = {"alpha", "beta", "gamma", "delta", "epsilon", "zeta","eta", "theta"};
   JList list  = new JList(entradesLlista );
   public static void main (String [] args){
      ExempleJList gui  = new ExempleJList ();
      gui.go(); }
   public void go(){
      JScrollPane scr = new JScrollPane(list);
      JPanel panel = new JPanel();
      JFrame frame = new JFrame();
      scr.setVerticalScrollBarPolicy(ScrollPaneConstants.VERTICAL_SCROLLBAR_ALWAYS);
      scr.setHorizontalScrollBarPolicy(ScrollPaneConstants.HORIZONTAL_SCROLLBAR_NEVER);
      panel.add(scr);
      list.setVisibleRowCount(4);
      list.setSelectionMode(ListSelectionModel.SINGLE_SELECTION);
      list.addListSelectionListener(this);
      frame.getContentPane().add(BorderLayout.CENTER, panel);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(350,300);
      frame.setVisible(true);
   }
   public void valueChanged(ListSelectionEvent lse){
      if(lse.getValueIsAdjusting()){
         String selection = (String) list.getSelectedValue();
         System.out.println(selection);    }   }
} // Final classe Exemple
```



Exercici

- Ompli els forats (indicats amb punts suspensius) del següent codi utilitzant els coneixements previs sobre registre d'events i interfícies Listener, i la següent informació sobre la classes JOptionPane.

JOptionPane

- Els mètodes més importants de la classe JOptionPane són

Method Name	Description
showConfirmDialog	Asks a confirming question, like yes/no/cancel.
showInputDialog	Prompt for some input.
showMessageDialog	Tell the user about something that has happened.
showOptionDialog	The Grand Unification of the above three.

JOptionPane

- Capçalera d'alguns mètodes de la classe JOptionPane:

public static String **showInputDialog**(Object message) throws HeadlessException

public static String **showInputDialog**(Object message, Object initialSelectionValue)

public static String **showInputDialog**(Component parentComponent, Object message) throws HeadlessException

public static String **showInputDialog**(Component parentComponent, Object message, Object initialSelectionValue)

public static String **showInputDialog**(Component parentComponent, Object message, String title, int messageType) throws HeadlessException

public static Object **showInputDialog**(Component parentComponent, Object message, String title, int messageType, Icon icon, Object[] selectionValues, Object initialSelectionValue) throws HeadlessException

public static void **showMessageDialog**(Component parentComponent, Object message) throws HeadlessException

public static void **showMessageDialog**(Component parentComponent, Object message, String title, int messageType) throws HeadlessException

public static void **showMessageDialog**(Component parentComponent, Object message, String title, int messageType, Icon icon) throws HeadlessException

JOptionPane

- Capçalera d'alguns mètodes de la classe JOptionPane:

public static int **showConfirmDialog**(Component parentComponent, Object message) throws HeadlessException

public static int **showConfirmDialog**(Component parentComponent, Object message, String title, int optionType) throws HeadlessException

public static int **showConfirmDialog**(Component parentComponent, Object message, String title, int optionType, int messageType) throws HeadlessException

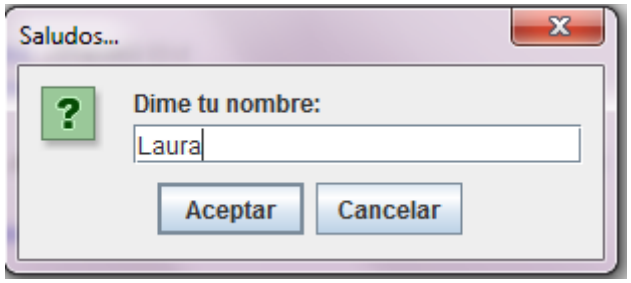
public static int **showConfirmDialog**(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon) throws HeadlessException

public static int **showOptionDialog**(Component parentComponent, Object message, String title, int optionType, int messageType, Icon icon, Object[] options, Object initialValue) throws HeadlessException

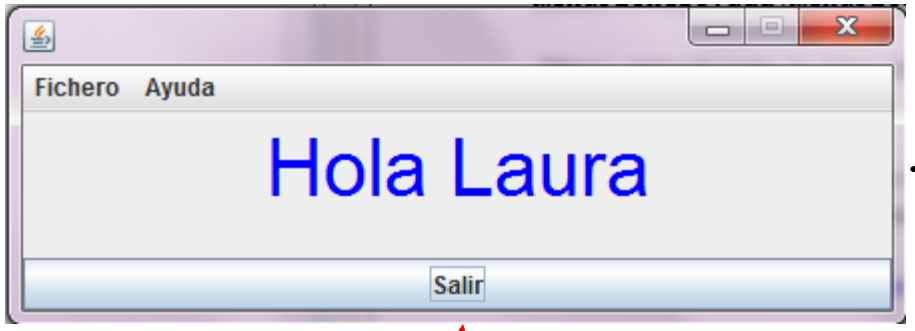
Exercici 2

- L'aplicació tindrà aquestes dues finestres:

Primera finestra:



Segona finestra:



Sortir de l'aplicació

Sortir de l'aplicació

S'obre una finestra per donar informació sobre qui ha fet l'aplicació

```
public class MenuDialogSwing2{
   private void go() {
      JFrame frame = new JFrame();
      frame.getContentPane().setLayout(new BorderLayout());
      frame.addWindowListener(.....);


      GestorMenus gestorMenus = new GestorMenus(frame);

      JMenu menuFichero = new JMenu("Fichero");
      JMenuItem menuFicheroSalir = new JMenuItem("Salir",'S');
      menuFicheroSalir.addActionListener(gestorMenus);
      menuFichero.add(menuFicheroSalir);

      JMenu menuAyuda = new JMenu("Ayuda");
      JMenuItem menuAyudaAcercaDe = new JMenuItem("Acerca de...",'A');
      menuAyudaAcercaDe.addActionListener(.....);
      menuAyuda.add(menuAyudaAcercaDe);

      JMenuBar barraMenu = new JMenuBar();
      barraMenu.add(menuFichero);
      barraMenu.add(menuAyuda);
      frame.getContentPane().add(barraMenu,BorderLayout.NORTH);
```

L'aplicació es tanca quan es tanca la finestra.

Objecte escoltador

?

```
public class MenuDialogSwing2{
   private void go() {
      JFrame frame = new JFrame();
      frame.getContentPane().setLayout(new BorderLayout());
      frame.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e){
                   System.exit(0);
                }
             });

      GestorMenus gestorMenus = new GestorMenus(frame);

      JMenu menuFichero = new JMenu("Fichero");
      JMenuItem menuFicheroSalir = new JMenuItem("Salir",'S');
      menuFicheroSalir.addActionListener(gestorMenus);
      menuFichero.add(menuFicheroSalir);

      JMenu menuAyuda = new JMenu("Ayuda");
      JMenuItem menuAyudaAcercaDe = new JMenuItem("Acerca de...",'A');
      menuAyudaAcercaDe.addActionListener(gestorMenus);
      menuAyuda.add(menuAyudaAcercaDe);

      JMenuBar barraMenu = new JMenuBar();
      barraMenu.add(menuFichero);
      barraMenu.add(menuAyuda);
      frame.getContentPane().add(barraMenu,BorderLayout.NORTH);
```

L'aplicació es tanca quan es tanca la finestra.

Objecte escoltador

Registem l'objecte menu a l'objecte escoltador

```
    JPanel panelNombre = new JPanel();
    panelNombre.setLayout(new FlowLayout());
    frame.getContentPane().add(panelNombre,BorderLayout.CENTER);
    JButton botonSalir = new JButton("Salir");
    frame.getContentPane().add(botonSalir,BorderLayout.SOUTH);
    botonSalir.addActionListener(.....);
```

?

```
    JLabel etiquetaNombre = new JLabel();
    panelNombre.add(etiquetaNombre);
    etiquetaNombre.setForeground(Color.blue);
    etiquetaNombre.setFont(new Font("Arial",Font.PLAIN,40));
    String nombre =.....;
    etiquetaNombre.setText("Hola " + nombre);
    frame.setBounds(250,200,440,150);
    frame.setVisible(true);
    frame.setResizable(false);
  }

  public static void main (String[] args) {
    MenuDialogSwing2 menu = new MenuDialogSwing2();
    menu.go();
  }
} // Final classe MenuDialogSwing2
```

?

Mètode main

```
        JPanel panelNombre = new JPanel();
        panelNombre.setLayout(new FlowLayout());
        frame.getContentPane().add(panelNombre,BorderLayout.CENTER);
        JButton botonSalir = new JButton("Salir");
        frame.getContentPane().add(botonSalir,BorderLayout.SOUTH);
        botonSalir.addActionListener(new ActionListener(){
                public void actionPerformed(ActionEvent evt) {
                   System.exit(0);
                }});
```

L'aplicació es tanca quan es prem el botó "Salir".

```
    JLabel etiquetaNombre = new JLabel();
        panelNombre.add(etiquetaNombre);
        etiquetaNombre.setForeground(Color.blue);
        etiquetaNombre.setFont(new Font("Arial",Font.PLAIN,40));
        String nombre = JOptionPane.showInputDialog(frame,"Dime tu nombre:","Saludos...",
JOptionPane.QUESTION_MESSAGE);
        etiquetaNombre.setText("Hola " + nombre);
        frame.setBounds(250,200,440,150);
        frame.setVisible(true);
        frame.setResizable(false);
    }
    public static void main (String[] args) {
      MenuDialogSwing2 menu = new MenuDialogSwing2();
      menu.go();
    }
} // Final classe MenuDialogSwing2
```

Obrim finestra per demanar el nom.

Mètode main

3

```
class GestorMenus implements ActionListener {
        ..............
}
```

Classe externa que implementa ActionListener

```
class GestorMenus implements ActionListener {

   JFrame finestra;
   GestorMenus(JFrame v) {
      finestra = v;
   }
   public void actionPerformed(ActionEvent evt) {
      if(evt.toString().indexOf("Salir") != -1)
         System.exit(0);
      else if(evt.toString().indexOf("Acerca de...") != -1)
         JOptionPane.showMessageDialog(finestra, "Saludos...\nAutor: Laura", "Acerca de...",
         JOptionPane.INFORMATION_MESSAGE);
   }
}
```

Classe externa que implementa ActionListener
per sortir si seleccionem l'opció "Salir" del menú "Fichero"
per obrir una finestra amb informació de l'aplicació en cas de seleccionar l'opició "Acerca de…" del menú "Ayuda".

Comentaris

- Indica quins són els imports necessaris.

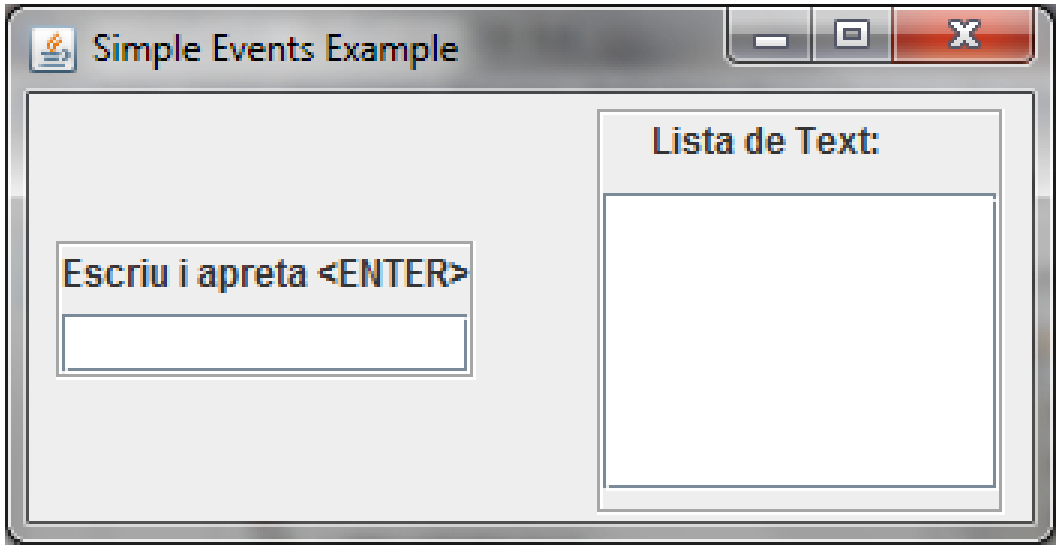
- Mètodes de la classe JOptionPane:

public static String **showInputDialog**(Component parentComponent, Object message, String title, int messageType) throws HeadlessException

- ERROR_MESSAGE
- INFORMATION_MESSAGE
- WARNING_MESSAGE
- QUESTION_MESSAGE
- PLAIN_MESSAGE

Exemple 5: Panel d'escriptura

Implementem aquesta interfície on podem escriure text al camp de l'esquerra i quan premem enter apareixerà a la lista de text.



Exemple 5: implementació A

```
// importa els símbols de AWT i Swing
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class SimpleEventsA{
   // ample i alt del frame
   static final int WIDTH=350;
   static final int HEIGHT=180;

   // Declara JTextField per a entrar text
   JTextField textField;
   // Declara JTextArea per rebre línies de textField
   JTextArea textList;
   // Declara JScrollPane per a JTextArea
   JScrollPane pane;

   public static void main(String args[]) {
      SimpleEventsA gui = new SimpleEventsA();
      gui.go();
   }// Fi mètode main
```

Per definir el tamany de la finestra.

Atributs

```
// Mètode go: aquí es fa quasi tot el treball

  public void go(){
    /******* Crea un contenidor per a textField ****/
    // Instancia un JPanel
    JPanel textPanel = new JPanel();
    // li posa un borde (per defecte no en te)
    textPanel.setBorder(BorderFactory.createEtchedBorder());
    // Fixa el layout del textPanel a BorderLayout
    textPanel.setLayout(new BorderLayout());

    // Crea una etiqueta i la afegeix al panell
    JLabel textTitle =new JLabel("Escriu i apreta <ENTER>");
    textPanel.add(textTitle, BorderLayout.NORTH);

    // Instancia un JTextField i l'afegeix al panell
    textField = new JTextField();
    textPanel.add(textField, BorderLayout.SOUTH);

    // Afegeix un strut al textPanel com a marge inferior
    textPanel.add(Box.createVerticalStrut(6));
```

← JPanel

← Afegim JLabel

← Afegim JTextField

← Afegim Marge

```
/******** Crea un contenidor pel textArea ******/
// Instancia un JPanel
JPanel listPanel = new JPanel();
// afegeix borde
listPanel.setBorder (BorderFactory.createEtchedBorder());
// Set el layout del textPanel
listPanel.setLayout(new BoxLayout(listPanel,BoxLayout.Y_AXIS));
```

```
// Crea una etiqueta i afegeix al panel
JLabel title = new JLabel("Llista de Text:");
listPanel.add(title);
// Afegeix un strut al BoxLayout
listPanel.add(Box.createVerticalStrut(10));
// Instancia una JTextArea sense text inicial
// 6 files, 10 columnes, i vertical scrollbars
textList = new JTextArea("", 6, 10);
// la fem read-only (només de lectura)
textList.setEditable(false);
```

```
// Afegeix textList a listPanel
pane = new JScrollPane(textList);
listPanel.add(pane);
// Afegeix un strut a listPanel com a margen inferior
listPanel.add(Box.createVerticalStrut(6));
```

```
// Afegeix un listener a textField quan se pulsa ENTER copia el text de
//textField a l'area de text. Les components estan interrelacionades
textField.addActionListener(new CampText());
```

← Registrem escoltador

```
// Afegeix els 2 panels al frame, separats per strut
JFrame frame = new JFrame("Simple Events Example");
frame.setLayout (new FlowLayout());
frame.add(textPanel);
frame.add(Box.createHorizontalStrut(30));
frame.add(listPanel);
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
frame.setSize(WIDTH, HEIGHT);
frame.setVisible(true);
```

```
}//Fi mètode go
```

```
class CampText implements ActionListener{
    public void actionPerformed(ActionEvent e) {
        // Afegeix el text de textField a textList
        textList.append(textField.getText());
        textList.append("\n");
        // Reset el textField
        textField.setText("");
    }
}
```

Classe interna que implementa la interfície ActionListener

```
}// Fi de la classe SimpleEvent
```

Exemple 5: implementació B

```
// importa els símbols de AWT and Swing
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

public class SimpleEventsB extends JFrame {

   // ample i alt del frame
   static final int WIDTH=350;
   static final int HEIGHT=180;
   // Declara JTextField per a entrar text
   JTextField textField;
   // Declara JTextArea per rebre línies de textField
   JTextArea textList;
   // Declara JScrollPane per a JTextArea
   JScrollPane pane;
```

```
public static void main(String args[]) {
    SimpleEventsB frame = new SimpleEventsB("Simple Events Example");
    // Standard adapter usat en quasi totes les
    // aplicacions per a tancar la finestra
    frame.addWindowListener(new WindowAdapter() {
       @Override
       public void windowClosing(WindowEvent e) {
       System.exit(0);
       }
    });
    // fixa el tamany de frame i el mostra
    frame.setSize(WIDTH, HEIGHT);
    frame.setVisible(true);
  }// Fi mètode main
```

Mètode main

```
// Constructor: aquí es fa quasi tot el treball
public SimpleEventsB(String lab) {
   // crida al constructor de JFrame: posa etiqueta
   super(lab);

   /******* Crea un contenidor per a textField ****/
   // Instancia un JPanel
   JPanel textPanel = new JPanel();
   // li posa un borde (per defecte no en te)
   textPanel.setBorder(BorderFactory.createEtchedBorder());
   // Fixa el layout del textPanel a BorderLayout
   textPanel.setLayout(new BorderLayout());
   // Crea una etiqueta i la afegeix al panel
   JLabel textTitle =new JLabel("Escriu i apreta <ENTER>");
   textPanel.add(textTitle, BorderLayout.NORTH);
   // Instancia un JTextField i afegeix a textPanel
   textField = new JTextField();
   textPanel.add(textField, BorderLayout.SOUTH);
   // Afegeix un strut al textPanel com a marge inferior
    textPanel.add(Box.createVerticalStrut(6));
```

```
/******** Crea un contenidor pel textArea ******/
// Instancia un JPanel
JPanel listPanel = new JPanel();
// afegeix borde
listPanel.setBorder (BorderFactory.createEtchedBorder());
// Set el layout del textPanel
listPanel.setLayout(new BoxLayout(listPanel,BoxLayout.Y_AXIS));
// Crea una etiqueta i afegeix al panel
JLabel title = new JLabel("Lista de Text:");
listPanel.add(title);
// Afegeix un strut al BoxLayout
listPanel.add(Box.createVerticalStrut(10));
// Instancia una JTextArea sense text inicial
// 6 files, 10 columnes, i vertical scrollbars
textList = new JTextArea("", 6, 10);
// la fem read-only (només de lectura)
textList.setEditable(false);
// Afegeix textList a listPanel
pane = new JScrollPane(textList);
listPanel.add(pane);
// Afegeix un strut a listPanel com a margen inferior
listPanel.add(Box.createVerticalStrut(6));
```

```
    // Afegeix un listener a textField quan se pulsa ENTER copia el text de
    //textField a l'area de text. Les components estan interrelacionades
    textField.addActionListener(new ActionListener() {
       public void actionPerformed(ActionEvent e) {
          // Afegeix el text de textField a textList
          textList.append(textField.getText());
          textList.append("\n");
          // Reset el textField
          textField.setText("");
       }});

    // Afegeix els 2 panels al frame, separats per strut
    Container c = getContentPane();
    c.setLayout (new FlowLayout());
    c.add(textPanel);
    c.add(Box.createHorizontalStrut(30));
    c.add(listPanel);
  }//Fi mètode constructor


}// Fi de la classe SimpleEvent
```

Classe interna anònima que implementa la interfície ActionListener

LOOK AND FEEL

Look and feel

- Swing està dissenyat per a que es pugui canviar el *look and feel* d'una aplicació GUI
  - ***look*** fa referència a l'**aparença** de les components de la GUI
  - ***feel*** fa referència al **funcionament** de les components de la GUI

- Els programes Java poden adoptar l'aparença de la plataforma sobre la que s'executen, o aparences especifiques.

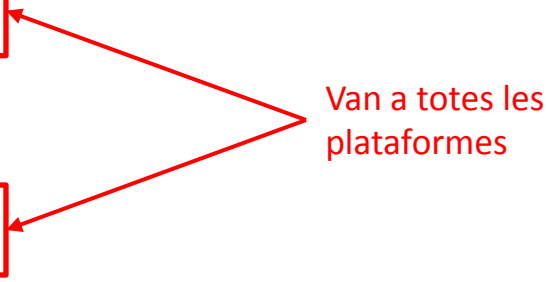
Look and feel

- *CrossPlatformLookAndFeel* or *Metal*

  Standard Java look and feel. Forma part de l'API (javax.swing.plaf.metal) i s'utilitza per defecte.
- *SystemLookAndFeel*: look and feel natiu per la plataforma. Dins del Java SDK.

Look and feel

- Java proporciona:
  - Metal
  - Nimbus
  - CDE/Motif
  - Windows
  - Windows classic



Look and feel

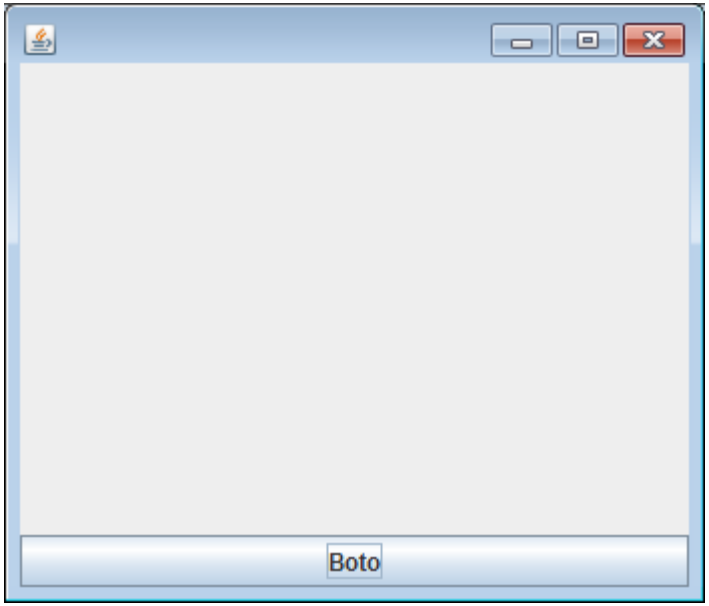
Platform	Look and Feel
Solaris, Linux with GTK+ 2.2 or later	GTK+
Other Solaris, Linux	CDE/Motif
IBM UNIX	IBM
HP UX	HP
Classic Windows	Windows
Windows XP	Windows XP
Windows Vista	Windows Vista
Macintosh	Macintosh

Look and feel

- Per fixar el look-and-feel podem utilitzar:

```
UIManager.LookAndFeelInfo plafinfo[] = UIManager.getInstalledLookAndFeels();
 boolean nimbusfound = false;
int nimbusindex = 0;
for (int look = 0; look < plafinfo.length; look++) {
    if (plafinfo[look].getClassName().toLowerCase().contains("nimbus")) {
          nimbusfound = true;
          nimbusindex = look;
    }
}
try {
    if (nimbusfound) {
       UIManager.setLookAndFeel(plafinfo[nimbusindex].getClassName());
    } else {
       UIManager.setLookAndFeel(UIManager.getCrossPlatformLookAndFeelClassName());
    }
} catch (Exception e) {       }
```

Look and feel

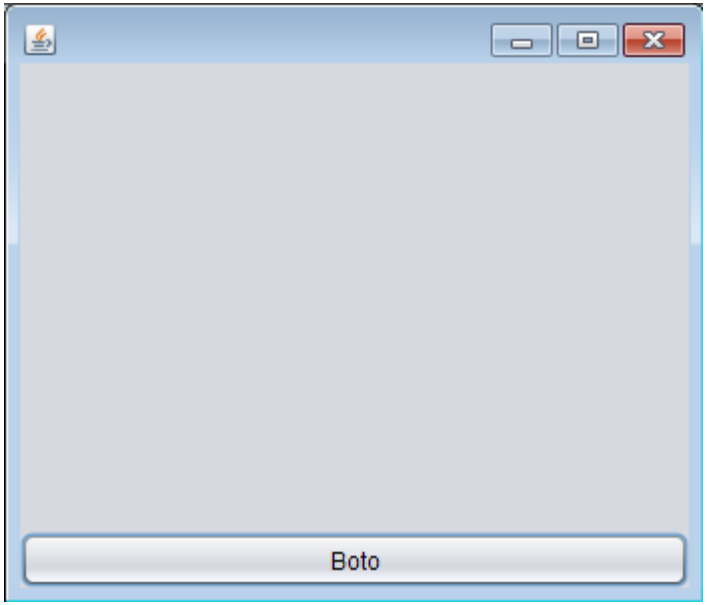


Default Look And Fell

NapkinLookAndFeel

Look and feel

Color Look And Fell



nimbus Look And Feel

Look and feel

```
import napkin.NapkinLookAndFeel;
```

```
public class ExempleLookFeel{
   public static void main(String [] args){
      ExempleLookFeel ex = new ExempleLookFeel();
     ex.goLookFeelNapkin();
}
public void goLookFeelNapkin(){
      try {
         UIManager.setLookAndFeel(new NapkinLookAndFeel());
      } catch (Exception unused) {
        // Ignore exception because we can't do anything.  Will use default.       }
      JFrame frame = new JFrame();
      JButton button = new JButton("Boto");
      frame.getContentPane().add(BorderLayout.SOUTH, button);
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(350,300);
      frame.setVisible(true);
   }
```

Creem una instància d'aquesta classe LookAndFeel

Exemple Colors

```
public class ExempleLookFeelProves {

   public static void main(String [] args){
      ExempleLookFeelProves e = new ExempleLookFeelProves();
      e.go();
   }

   public void go(){
            JFrame frame = new JFrame();
            JButton button = new JButton("Boto");
            button.setBackground(Color.red);
            frame.getContentPane().add(BorderLayout.SOUTH, button);

            frame.getContentPane().setBackground(Color.green);
            frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            frame.setSize(350,300);
            frame.setVisible(true);
    }
}
```

Canvia el color de fons del JButton

Canvia el color de fons del JFrame

PANELLS I GRÀFICS

Panells i Gràfics

- Hem treballat amb panells utilitzant-los com a contenidors d'altres components. D'aquesta forma podíem establir el diagrama de la interfície definint el diagrama de cada panel.
- Un panell es pot utilitzar també per a dibuixar imatges
- Veurem llavors cóm utilitzar un objecte de la classe JPanel com un àrea dedicada específicament al dibuix.
- L'usuari podrà dibuixar amb el ratolí o podrà dibuixar determinats objectes gràfics predeterminats de les llibreries gràfiques.

Graphics

1. Posar components en una finestra:

   frame.getContentPane().add(myBoto);

2. Dibuixar un gràfic sobre un component

   Graphics.fillOval(70,70,100,100);

3. Posar un JPEG en una component:

   Graphics.grawImage(myPic,10,10,this);

Exemple 6: GUI simple

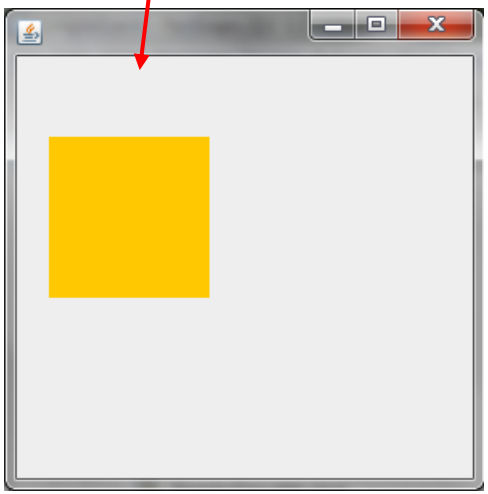
```
import java.awt.Color;
import java.awt.Graphics;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class GUIPanelDeDibuix {
  public static void main (String[] args) {
     GUIPanelDeDibuix gui = new GUIPanelDeDibuix();
     gui.go();
  }

  public void go() {
     JFrame frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
     PanelDeDibuix drawPanel = new PanelDeDibuix();
     frame.getContentPane().add(drawPanel);
     frame.setSize(300,300);
     frame.setVisible(true);
  }// tanca mètode go()
}// tanca classe GUIPanelDeDibuix
```

```
class PanelDeDibuix extends JPanel {

   public void paintComponent(Graphics g) {
      g.setColor(Color.orange);
      g.fillRect(20,50,100,100);
   }
}
```

Amplada i altura

Posició cantonada esquerra dalt

paintComponent



Mètode paintComponent()

- Mai cridar al mètode paintComponent() directament.

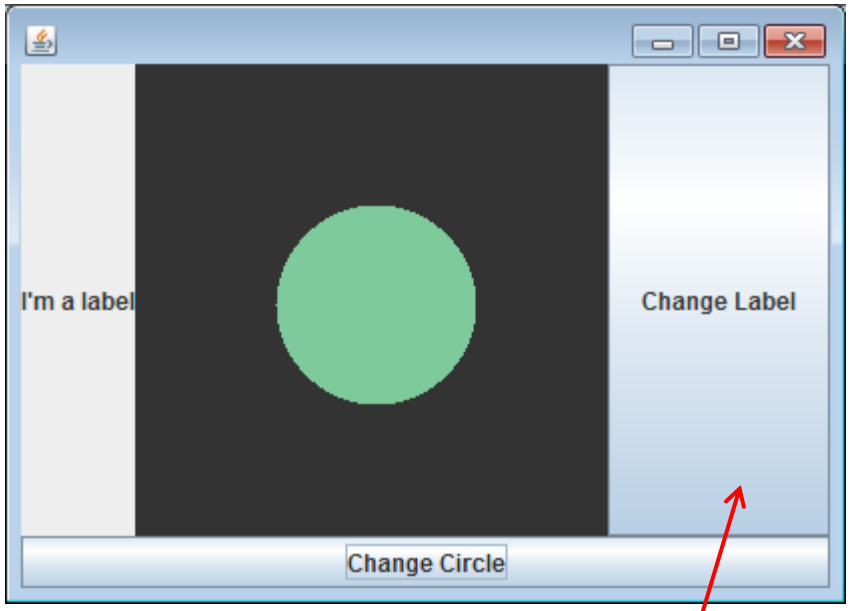
- Qui crida **paintComponent()?**
  - Es crida automàticament quan es fa visible la component.
  - Es crida indirectament des del listener definit per l'usuari via repaint()
    - Canvi de les variables de la instància.
    - Crida a repaint().

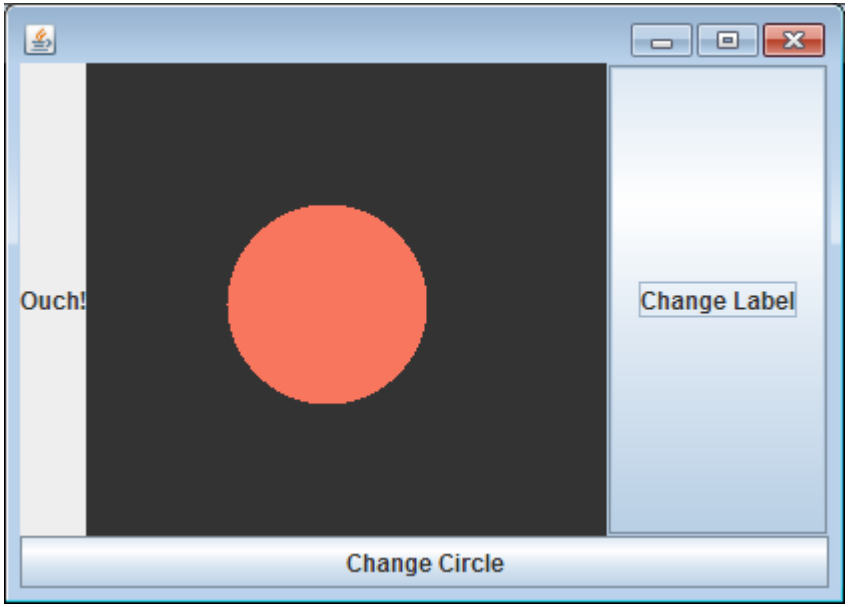
JComponent

- Mètode repaint()
- public void repaint() -> pinta tota la component
- public void repaint(long tm)
- public void repaint(int x, int y, int width, int height)
- public void repaint(long tm, int x, int y, int width, int height)

Exemple 7

- Implementeu una interfície gràfica d'usuari que contingui dos botons. Amb un canviem el color del cercle i amb l'altre canviem l'etiqueta





Aquest botó canvia el color del cercle

Aquest botó canvia el text de l'altra part de la finestra

Exemple 7

Classes internes

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class TwoButtons  {
   JFrame frame;
   JLabel label;
   public static void main (String[] args) {
      TwoButtons gui = new TwoButtons();
      gui.go();
   }
   public void go() {
      frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      JButton labelButton = new JButton("Change Label");
      labelButton.addActionListener(new LabelButtonListener());

      JButton colorButton = new JButton("Change Circle");
      colorButton.addActionListener(new ColorButtonListener());

      label = new JLabel("I'm a label");
      PanelDeDibuix drawPanel = new PanelDeDibuix ();
      frame.getContentPane().add(BorderLayout.SOUTH,
colorButton);
      frame.getContentPane().add(BorderLayout.CENTER, drawPanel);
      frame.getContentPane().add(BorderLayout.EAST, labelButton);
      frame.getContentPane().add(BorderLayout.WEST, label);

      frame.setSize(420,300);
      frame.setVisible(true);
   }
   class LabelButtonListener implements ActionListener{
      public void actionPerformed(ActionEvent event) {
         label.setText("Ouch!");
      }
   }
   class ColorButtonListener implements ActionListener{
      public void actionPerformed(ActionEvent event) {
         frame.repaint();
      }
   }
} // Fi class TwoButtons
```

Mètode main

Crida al mètode paintComponent

Exemple 7

…. Classes internes

```
class LabelButtonListener implements ActionListener{
  public void actionPerformed(ActionEvent event) {
     label.setText("Ouch!");
  }
}
class ColorButtonListener implements ActionListener{
  public void actionPerformed(ActionEvent event) {
     frame.repaint();
  }
}
} // Fi class TwoButtons
```

Crida al mètode paintComponent

paintComponent és un mètode de JComponent

Refresca el panell de dibuix

```
class PanelDeDibuix extends JPanel {

   public void paintComponent(Graphics g) {
      g.fillRect(0,0,this.getWidth(), this.getHeight());
      // make random colors to fill with
      int red = (int) (Math.random() * 255);
      int green = (int) (Math.random() * 255);
      int blue = (int) (Math.random() * 255);

      Color randomColor = new Color(red, green, blue);
      g.setColor(randomColor);
      g.fillOval(70,70,100,100);
   }
} // Fi class PanelDeDibuix
```

Els següents gràfics es pintaran en el color aleatori definit

EXEMPLES D’ANIMACIONS

Exemple 8: Animació simple

- Implementeu una interfície gràfica que dibuixa sobre una finestra de tamany 300x300 pixels un cercle verd a la posició inicial (70,70) i el va movent en diagonal fins arribar a la cantonada inferior dreta.

Exemple 8: Animació simple

```
import javax.swing.*;
import java.awt.*;
public class SimpleAnimation {
   int x = 70;
   int y = 70;
   public static void main (String[] args) {
      SimpleAnimation gui = new SimpleAnimation ();
      gui.go();
   }
   public void go() {
      JFrame frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      PanelDeDibuix drawPanel = new PanelDeDibuix();
      frame.getContentPane().add(drawPanel);
      frame.setSize(300,300);
      frame.setVisible(true);
      for (int i = 0; i < 130; i++) {
         x++;
         y++;
         drawPanel.repaint();
         try {
           Thread.sleep(5);
         } catch(Exception ex) { }
      }
   }// close go() method
```

Mètode main

L'acció està aquí:

Es repeteix 130 vegades

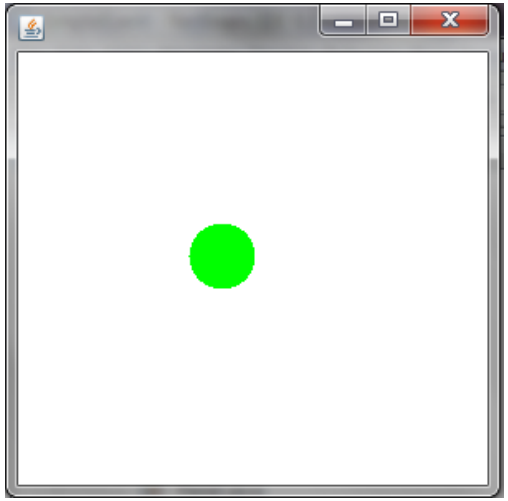
Li diu al panel que es torni a pintar

Espera 5 milisegons

```
class PanelDeDibuix extends JPanel {

   public void paintComponent(Graphics g) {
      g.setColor(Color.white);
      g.fillRect(0,0,this.getWidth(), this.getHeight());

      g.setColor(Color.green);
      g.fillOval(x,y,40,40);

   }
  } // close inner class
} // close outer class
```

El blanc és el primer color seleccionat

El verd és el segon



Exemple 9: Animació simple

- Implementeu una interfície gràfica que dins d'una finestra de tamany 500x300 pixels dibuixa un rectangle blau d'amplada 500 i alçada 250 i el va fent més petit fins que desapareix.

Exemple 9: Una altra animació

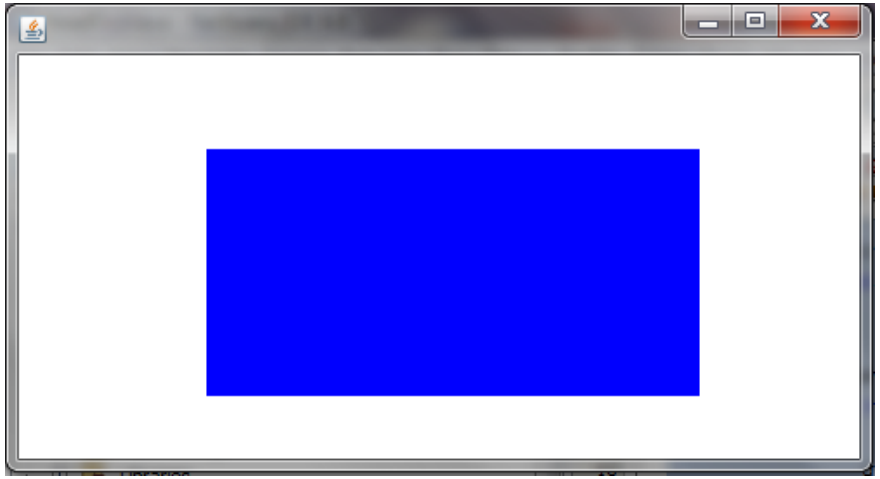
```
import javax.swing.*;
import java.awt.*;
public class Animate {
   private static final int XSIZE = 500;
   private static final int YSIZE = 300;
   int x = 1;
   int y = 1;
```

Mètode main

```
   public static void main (String[] args) {
      Animate gui = new Animate ();
      gui.go();
   }
   public void go() {
      JFrame frame = new JFrame();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      PanelDeDibuix panelDibuix = new PanelDeDibuix();
      frame.getContentPane().add(panelDibuix);
      frame.setSize(XSIZE,YSIZE);
      frame.setVisible(true);
      for (int i = 0; i < 125; i++) {
         panelDibuix.repaint();
         try {
           Thread.sleep(50);
         } catch(Exception ex) { }
         x=x+2;
         y++;
      }
   }// close go() method
```

El rectangle és dues vegades més ample que alt

```
   class PanelDeDibuix extends JPanel {
         public void paintComponent(Graphics g ) {
               g.setColor(Color.white);
               g.fillRect(0,0,XSIZE,YSIZE);
               System.out.println("x,y" + x +" , " +y);
               g.setColor(Color.blue);
               g.fillRect(x,y,500-x*2,250-y*2);
         }
      }// Fi classe PanelDeDibuix
   } // Fi classe Animate
```



Exemple 10: animació amb el ratolí

- Implementeu una interfície gràfica de tamany 400x200 pixels on al apretar amb el ratolí sobre ella apareixen al costat els valors de les coordenades de la posició del ratolí en aquest moment.

RatonSwing Animation

```
public class RatonSwing{
   int x,y;
   JFrame frame;
   PanelDeDibujo panelDeDibujo;
```

Mètode main

```
   public static void main (String[] args) {
      RatonSwing r = new RatonSwing();
      r.go();
   }

   private void go(){
      frame = new JFrame("Jugando con el ratón ...");
      frame.addWindowListener(new GestorFinestra());

      panelDeDibujo = new PanelDeDibujo();
      frame.getContentPane().add(panelDeDibujo,BorderLayout.CENTER);
      frame.addMouseListener(new GestorRatoli());

      frame.setBounds(250,200,400,200);
      frame.setVisible(true);
   }
```

Continua en la següent pàgina….

```
class GestorRatoli extends MouseAdapter{
    @Override
    public void mousePressed(MouseEvent e) {
        x = e.getX();
        y = e.getY();
        frame.repaint();
    }
}
```

Classe interna que hereta de la classe MouseAdapter i gestiona l'event de prémer el ratolí reescrivint el mètode mousePressed.

Crida al mètode paint de la classe PanelDeDibujo.

```
class PanelDeDibujo extends JPanel {
     PanelDeDibujo(){
        setBackground(Color.white);
     }
    public void paint(Graphics g) {
        g.drawString(x+","+y,x,y);
    }
}
} Final classe RatonSwing
```

Classe interna que hereta de Jpanel i reescriu el mètode paint

```
class GestorFinestra extends WindowAdapter{
    @Override
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
}
```

Classe que hereta de WindowAdapteri reescriu el mètode windowClosing.

Referències del bloc 3

- Llibre "**Head First Java**", Kathy Sierra & Bert Bates.
- *Apunts: "***Aprenda Java como si estuviera en Primero**" (Universidad de Navarra): http://www.tecnun.es/asignaturas/Informat1/ayudainf/aprendainf/Java/Java2.pdf
- "**Creating a GUI with JFC/Swing**" (The Swing Tutorial) http://java.sun.com/docs/books/tutorial/uiswing/
- The Swing Connection http://java.sun.com/javase/technologies/desktop/articles.jsp
- Components: http://download.oracle.com/javase/tutorial/uiswing/components/