

GRAU D'ENGINYERIA INFORMÀTICA

PROGRAMACIÓ II

CURS 12-13

Bloc 2:

Programació Orientada a Objectes (5)

Laura Igual

Departament de Matemàtica Aplicada i Anàlisi

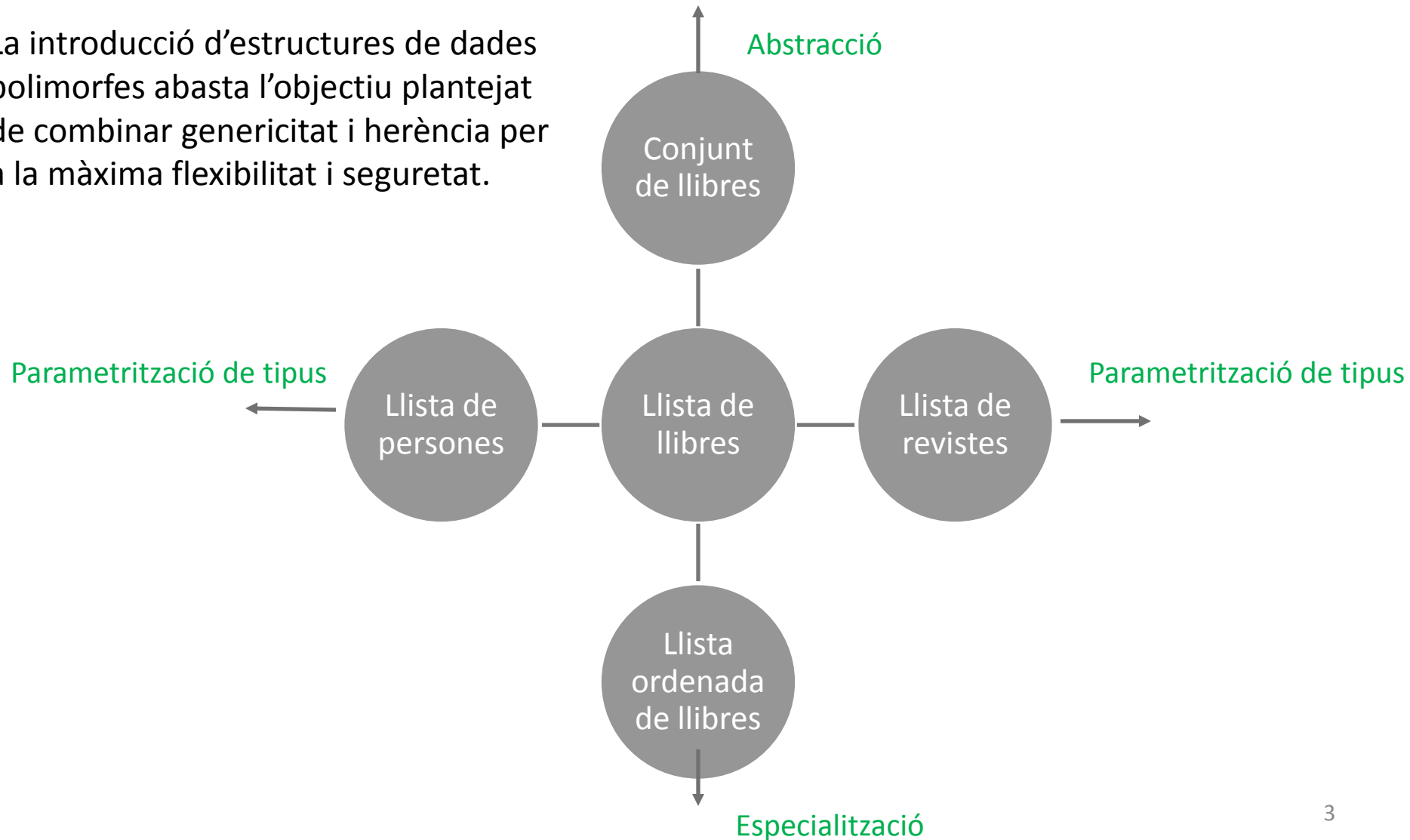
Facultat de Matemàtiques

Universitat de Barcelona

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.

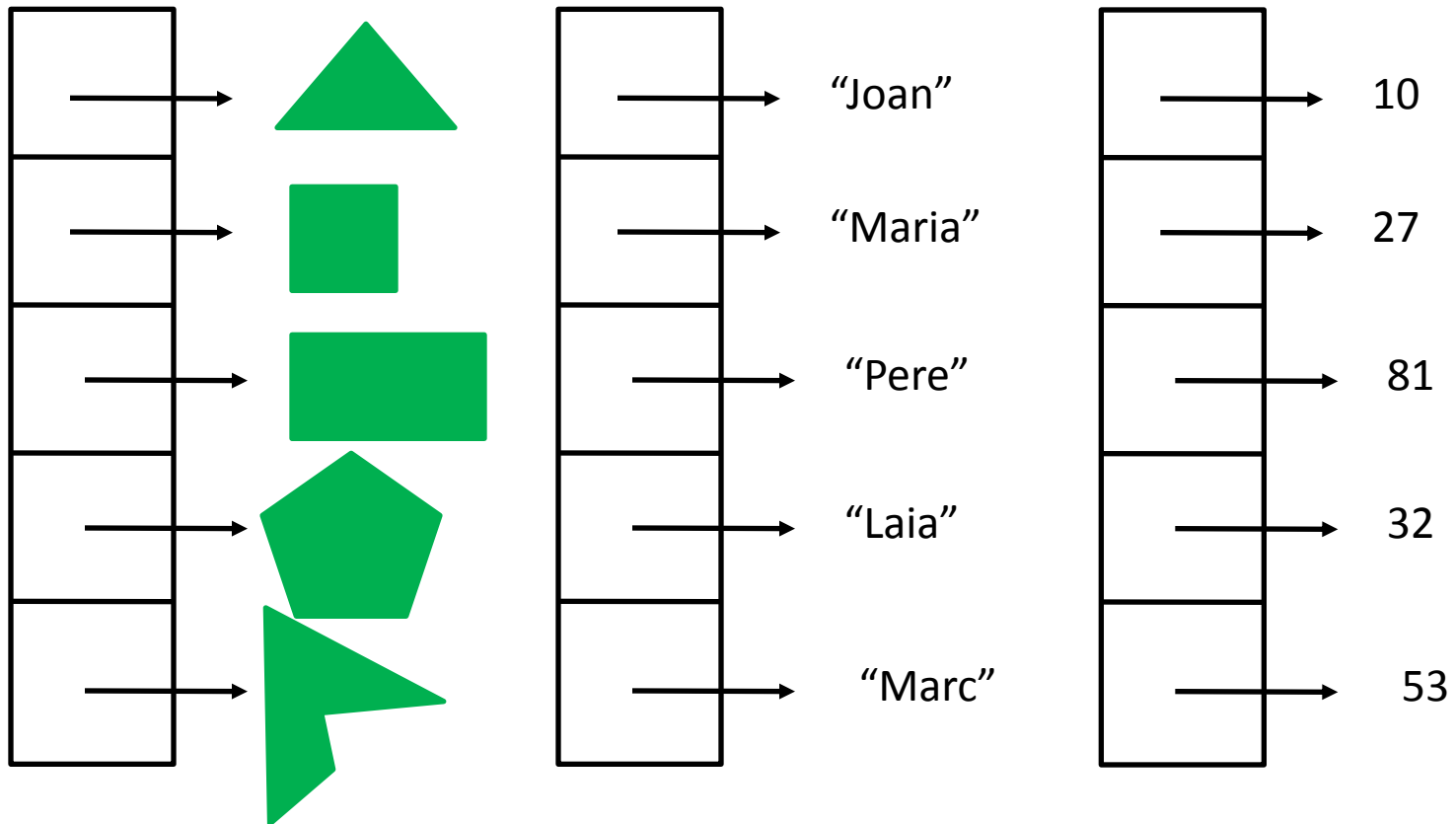
- **Exemple:**

- 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**).

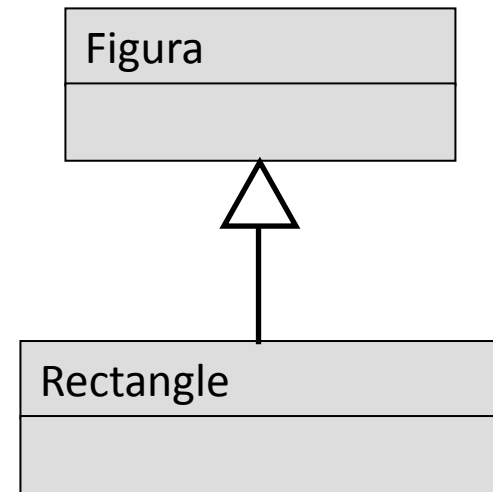


Informació de classes en temps d'execució (Recordatori)

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

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



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:

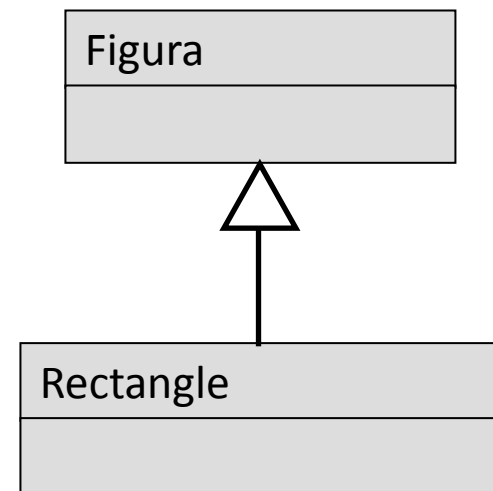
Upcasting

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



Downcasting

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



Informació de classes en temps d'execució

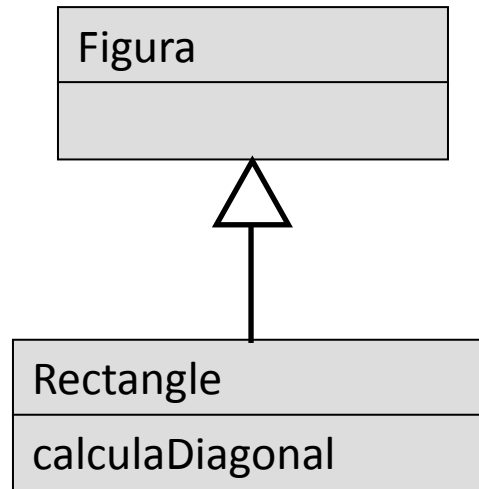
- Un casting permet forçar la connexió a la referència
- Un intent de casting impossible generarà una excepció ***ClassCastException*** en temps d'execució
- Possibles accions:
 - Podem capturar aquesta excepció per a determinar si l'objecte apuntat per la referència és del tipus esperat o no, realitzant 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

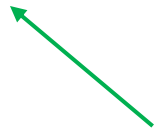
Exemple: diagonal màxima

- 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 unAltreInterfícies.Gos
persegueix

He extret el animal del tipus:class unAltreInterfícies.Gat
esgarrapa

Framework Col·leccions

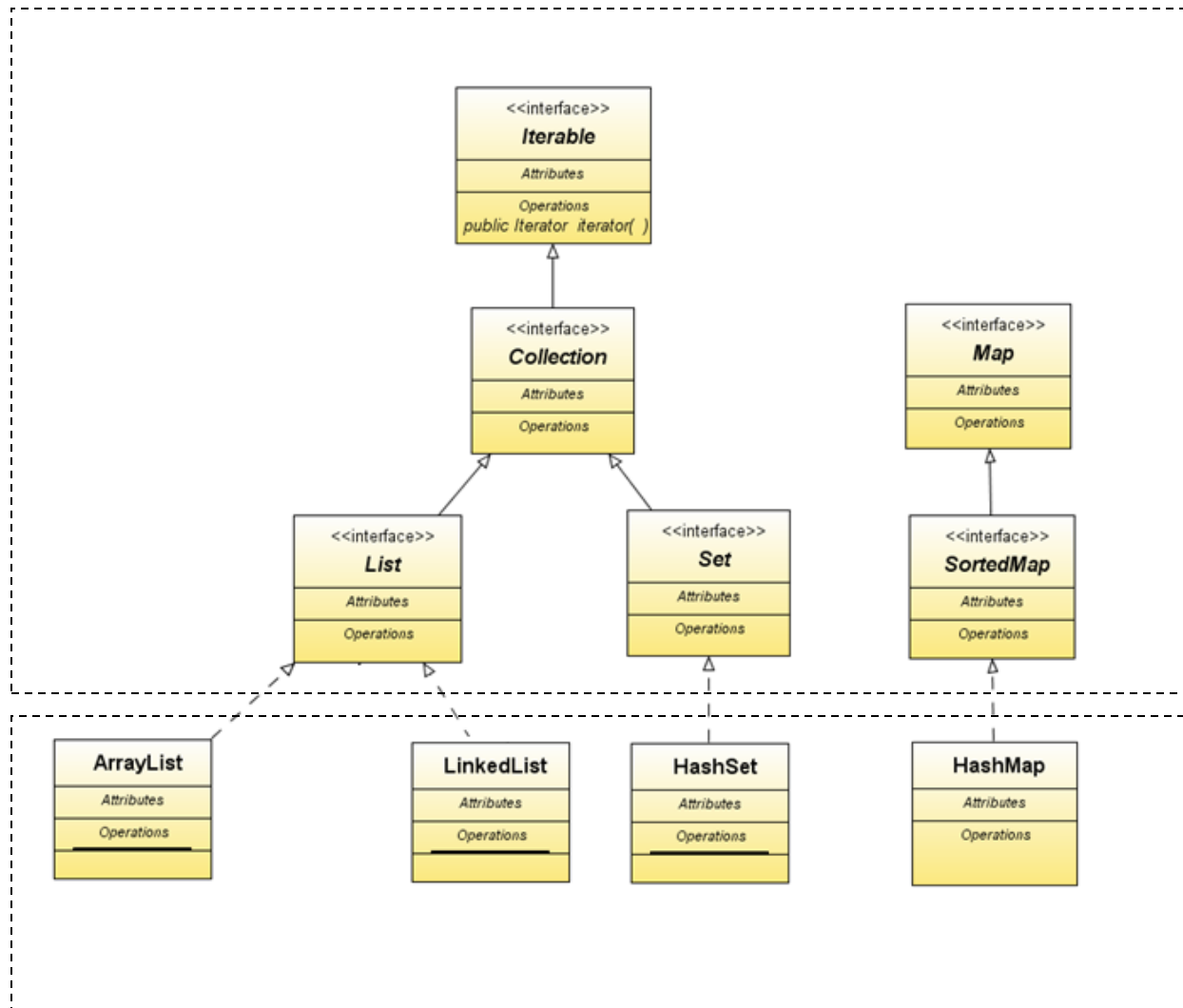
- Una **col·lecció** és un objecte que agrupa múltiples elements en una única unitat.
- Normalment representen elements d'informació dins d'un grup natural, com
 - una bústia de correu (una col·lecció de correus),
 - un directori (una col·lecció de fitxers),
 - una guia telefònica (una associació entre 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,

Diagrama de classes simplificat

Interfaces



Implementations

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`

- 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`

```
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`

- 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 propis 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);  
    }  
}
```


Example: iterators

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

Exemple (1)


...

```
ArrayList integerList = new ArrayList();
```

```
integerList.add( new Integer(1));
```

```
integerList.add( new Integer(2));
```

Els elements de la lista són de tipus Object.



```
Iterator listIterator = integerList.iterator();
```


```
while(listIterator.hasNext()) {
```

```
    Integer item = (Integer) listIterator.next();
```

```
}
```

...

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");
```

No hi ha cap
restricció dels
elements



```
Iterator listIterator = integerList.iterator();
```


```
while(listIterator.hasNext()) {
```

```
    Integer item = (Integer) listIterator.next();
```

```
}
```

```
...
```

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

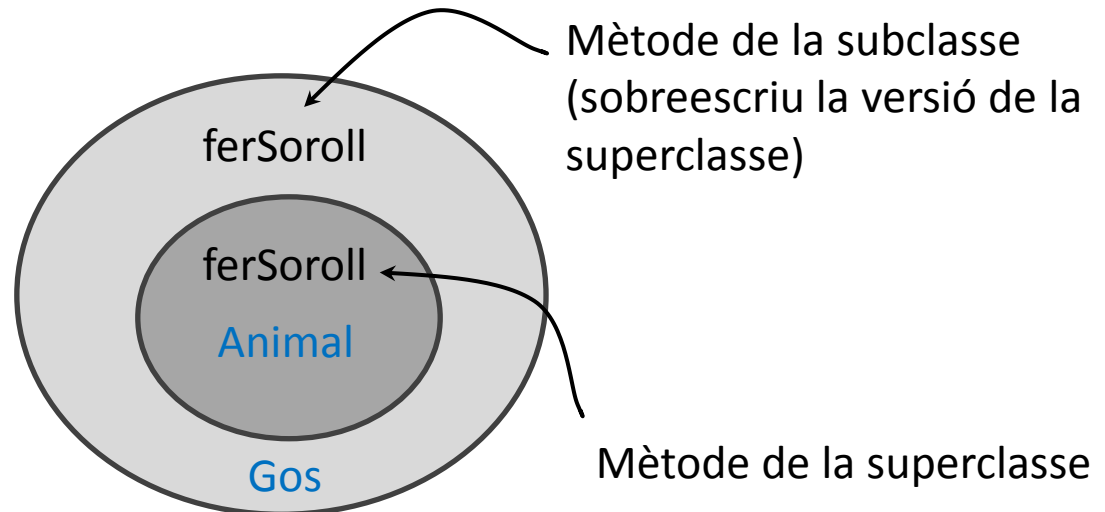
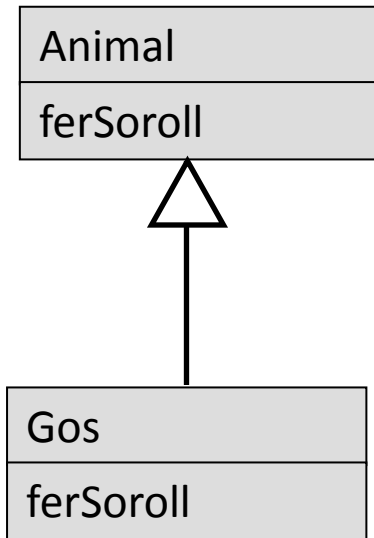
- Si no utilitzem polimorfisme tindríem que saber en temps de disseny quins animals tindràs exactament.
- Per tant dissenyes una classes que sigui lo suficientment genèrica com per a que accepti qualsevol tipus d'animal sense necessitat de saber quin tipus d'animal és: `ArrayList<Animal>`

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:

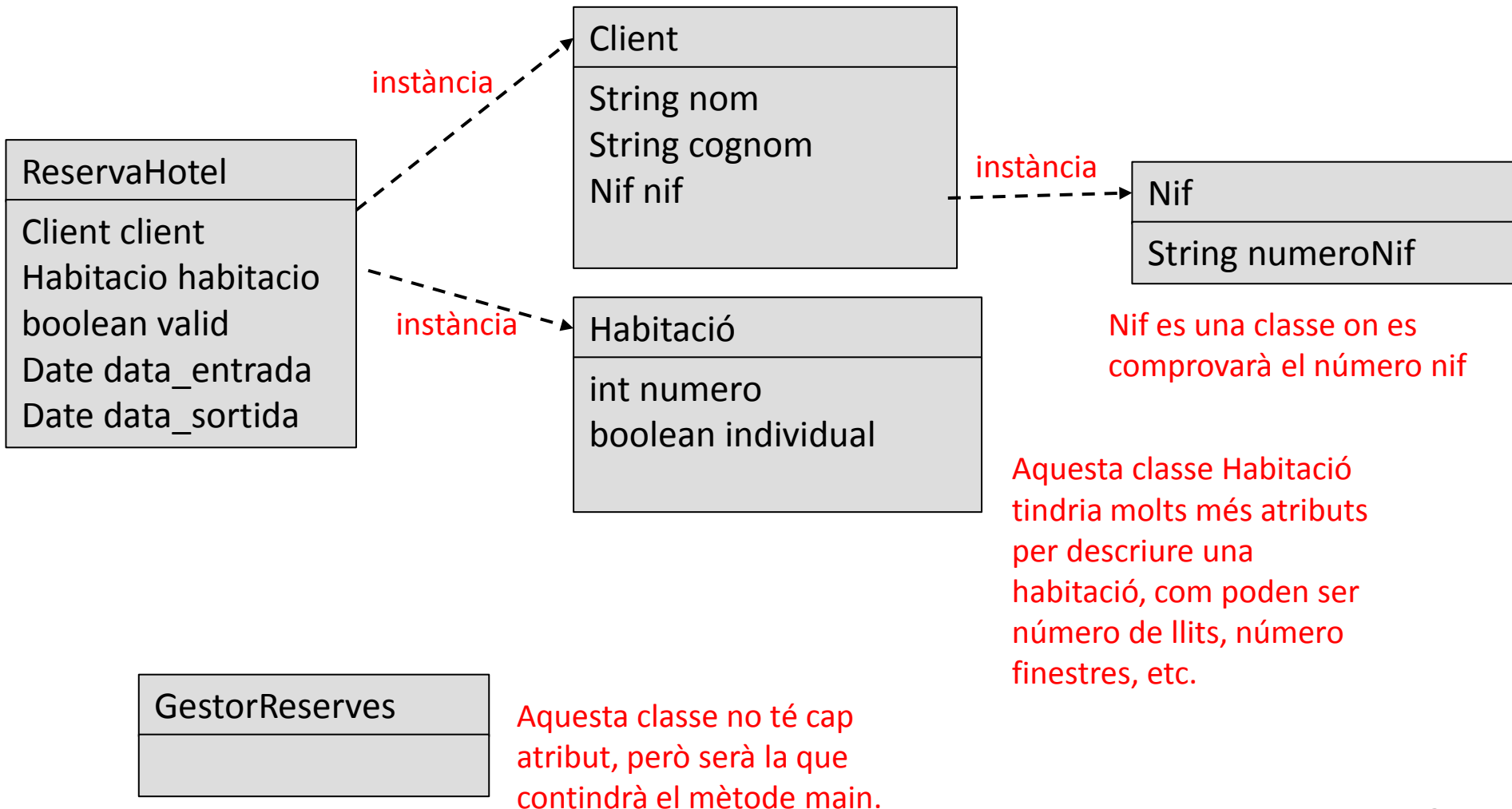
Una referència a un objecte de la subclasse (**Gos**) sempre cridarà a la versió de la subclasse del mètode sobreescrit. Això és el polimorfisme. Però el codi de la subclasse pot cridar `super.ferSoroll()` per invocar la versió de la superclasse.

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.

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();  
    }  
}
```

```
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 ...  
}
```

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

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

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
resea sense assignar
habitació i client ho
podem fer així.

Exemple

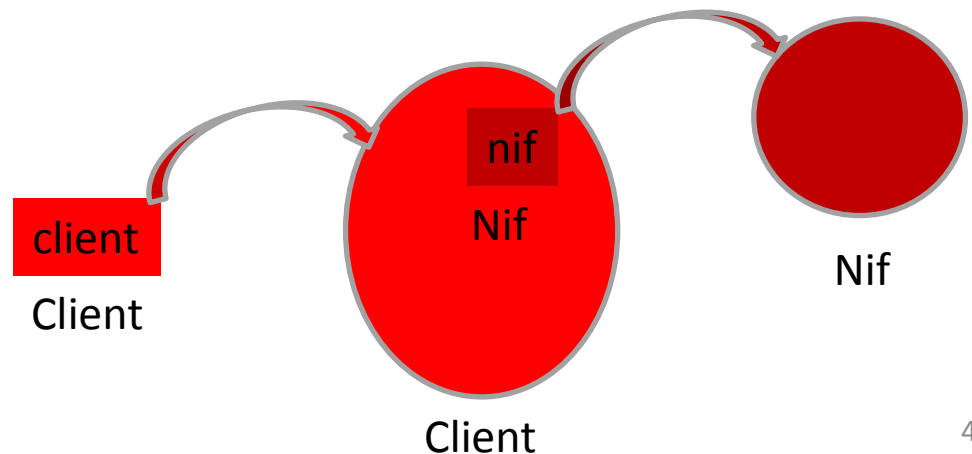
```
public class Client{  
    String nom;  
    String cognom;  
    Nif nif;  
  
    // constructor de la classe:  
    public Client(String nif){  
        nif = new Nif(nif);  
    }  
    // mètodes d'accés i d'escriptura de la classe:  
    public void setNom(String nom){  
        this.nom=nom;  
    }  
    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

- Bertrand Meyer, “**Construcción de software orientado a objetos**”, Prentice Hall, 1998.
- “Software Architecture and UML” de Grady Booch (Rational Software). Presentació P. Letelier.
- Bert Bates, Kathy Sierra. **Head First Java**. O’Reilly Media, 2005.