

GRAU D'ENGINYERIA INFORMÀTICA

PROGRAMACIÓ II

Bloc 2:

Programació Orientada a Objectes (6)

Laura Igual

Departament de Matemàtica Aplicada i Anàlisi

Facultat de Matemàtiques

Universitat de Barcelona

Índex

- Excepcions
- Col·leccions
- Assercions

EXCEPCIONES

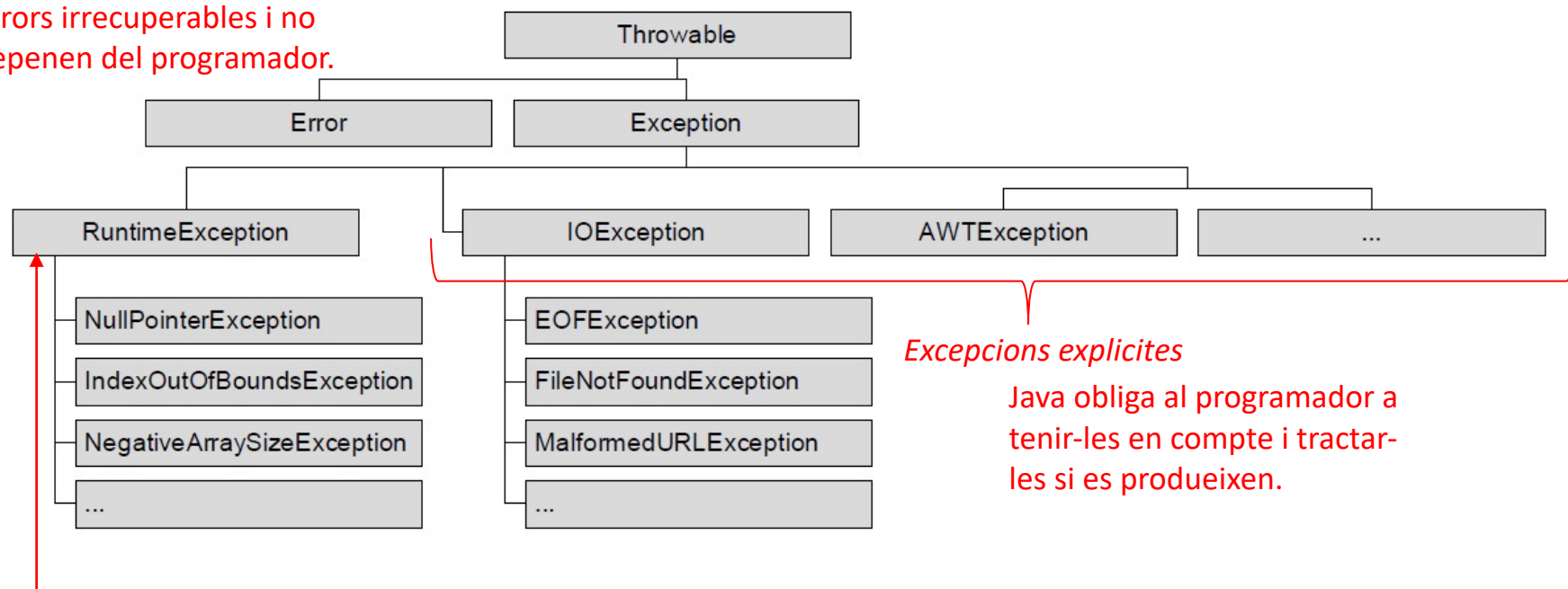
Excepcions

- Una excepció es pot definir com l'ocurrència d'un **esdeveniment inesperat durant l'execució** normal d'un programa.
- En els llenguatges orientats a objectes, **la gestió dels errors** es realitza a través d'excepcions.
- Atès que en **Java** tot són classes, necessitem una classe que ens representi l'excepció i un mecanisme del llenguatge que ens permeti capturar els errors i tractar-los en cas de necessitat.
- Existeixen dues famílies d'excepcions:
 - **Els errors**: problemes greus que es poden donar i que no val la pena controlar (falta de memòria, error de comunicacions, etc).
 - **Las excepcions** en si mateixes que són predictibles i que d'una manera raonable podem (i hem de) controlar.

Excepcions Estàndard de Java

- Jerarquia de classes derivades de Throwable:

Errors irrecuperables i no
depenen del programador.



Excepcions explícites

Java obliga al programador a
tenir-les en compte i tractar-
les si es produeixen.

*Són excepcions molt freqüents,
relacionades amb errors de programació.
Es poden anomenar excepcions implícites.*

Excepcions Estàndard de Java

- Les classes derivades d'Exception poden pertanyer a diferents packages de Java: java.lang (Throwable, Exception, RuntimeException, ...), java.io (EOFException, FileNotFoundException,...), etc.
- Al heretar de **Throwable** tots els tipus d'excepcions poden utilitzar els mètodes següents:
 - String ***getMessage()***: Extreu el missatge associat amb l'excepció.
 - String ***toString()***: Torna un String que descriu l'excepció.
 - void ***printStackTrace()***: Indica el mètode on es va llançar l'excepció.

Creació d'una excepció

- En Java, qualsevol excepció ha d'heretar de la classe `Exception`, que ja tenim definida i implementada en l'API.

```
public class MyException extends Exception{  
    ...  
}
```

- La pròpia classe `Exception` ja té un atribut de tipus `String` per a emmagatzemar el missatge que donarà.
 - Es poden definir missatges personalitzats que informin de la mateixa manera sobre el mateix error

Creació d'una excepció

MyException.java

```
public class MyException extends Exception{  
    public static String MyMessage = "Escriu aquí el que necessites"; // Constant  
  
    public MyException(){                                // Constructor per defecte  
        super();  
    }  
    public MyException(String message){                  // Constructor amb missatge  
        super(message);  
    }  
}
```


Llançament d'excepcions

- Un cop tenim definida una classe que representa les nostres excepcions, caldrà poder-les utilitzar per a indicar les situacions anòmales que es produeixen durant l'execució del codi.
- Es llança una excepció amb la sentència **throw** seguida de l'objecte Exception creat:

```
throw new MyException("Descripció de l'error");
```

O l'equivalent:

```
MyException me =new MyException("Descripció de l'error");  
throw me;
```

- L'execució del mètode en què s'ha llançat l'excepció queda interromput
- Immediatament retornem al lloc on es va realitzar la crida (com amb un return)

Capturar excepcions

- Hi ha certs mètodes que llancen excepcions, si no es té en compte es produirà un error de compilació.
- El programador haurà de fer una d'aquestes dues coses:
 1. Re-llançar l'excepció: **throws**
 2. Captura/Gestió de l'excepció: bloc **try ... catch**

Tractament d'excepcions

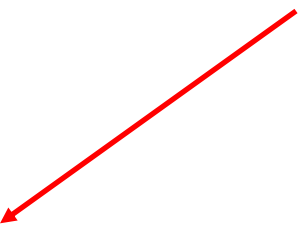
- Cal tenir constància del lloc en què es poden produir.
- Hem d'incloure dintre d'un bloc **try ... catch** el conjunt d'instruccions que poden generar una excepció.

```
try {  
    // Codi que podria llançar una excepció  
}  
catch (ExceptionType exceptionVar) {  
    // Codi que gestiona una excepció  
}
```

Tractament d'excepcions

```
try {  
    // Codi que podria llançar una excepció  
}catch (ExceptionType1 exceptionVar1) {  
    // Codi que gestiona una excepció  
}catch (ExceptionType2 exceptionVar2) {  
    // Codi que gestiona una excepció  
}finally {  
    // Codi que s'executa sempre, sigui quina sigui l'excepció  
    // o si no hi ha. Inclús si en el bloc try hi ha un return  
}
```

Es poden incloure
tants try com siguin
necessaris, per
tractar diferents
tipus d'excepcions



Bloc opcional




- Cal tenir en compte que hi ha excepcions més genèriques que altres
 - per tant, cal posar els blocs “catch” des del que tracta l'excepció més específica fins al que tracta l'excepció més genèrica, en cas contrari sempre s'entrarà al més genèric.
- Si s'utilitza una superclasse podem tractar **un grup d'excepcions** (totes les que deriven d'ella).

Tractament d'excepcions

- Si no volem tractar l'excepció en un mètode, però volem que el mètode que l'ha cridat tingui consciència de que s'ha generat:

```
void metode(String nom) throws Exception1 {  
    ...  
}
```

Es declaren les
excepcions que es
poden produir.



Exemple del Reproductor:

Volem que s'imprimeixen tots els missatges d'error en la classe ReproductorUB2 que pertany a la vista, llavors, delegarem la gestió de les excepcions al mètode que crida al mètode on es llança l'excepció:

```
public void addFitxer(File fitxer) throws ReproException {  
  
}
```


Tractament d'excepcions

Exemple:

```
private static void createException() throws MyException {  
    throw new MyException(MyException.MyPersonalMessage);  
}
```

```
try {  
    ....  
    createException();  
}  
catch (MyException m) {  
    System.out.println(m.getMessage());  
}
```

Es declaren les
excepcions que es
poden produir.



Exemple

```
try {  
    ...  
} catch (FileNotFoundException e) {  
    // S'ocupa de l'excepció FileNotFoundException donant un avís:  
    System.err.println("FileNotFoundException: " + e.getMessage());  
}  
catch (IOException e) {  
    System.err.println("Caught IOException: " + e.getMessage());  
}
```

Tornen el
missatges d'error



Example 1

MyException.java

```
package examplesjava;

class MyException extends Exception{
    public MyException(){
        super();    // constructor per defecte
    }
    public MyException(String s){
        super(s);    // constructor amb missatge
    }
}
```

Llanzadora.java

```
package examplesjava;

// Aquesta classe llançarà una excepció
public class Llanzadora {
    void llanzaSiNegatiu( int param ) throws MyException {
        if ( param < 0 ){
            throw new MyException( "Numero negatiu" );
        }
    }
}
```


Exemple 1

TestExcepcions .java

```
package examplesjava;
import java.io.FileInputStream;
import java.io.IOException;
public class TestExcepcions{
    public static void main( String[] args ) {
        Llanzadora llanza = new Llanzadora();
        FileInputStream entrada = null;
        int leo;
        try {
            entrada = new FileInputStream( "fitx.txt" );
            leo = entrada.read();
            while ( leo != -1 ){
                llanza.llanzaSiNegatiu( leo );
            }
            entrada.close();
            System.out.println( "Tot ha anat bé" );
        } catch ( MyException e ){ // Personalitzada
            System.out.println( "Excepcio: " + e.getMessage() );
        } catch ( IOException e ){ // Estàndard
            System.out.println( "Excepcio: " + e.getMessage() );
        } finally {
            if ( entrada != null )
                try {
                    entrada.close(); // Sempre queda tancat
                } catch ( Exception e ) {
                    System.out.println( "Excepcio: " + e.getMessage() );
                }
            System.out.println( "Fitxer tancat." );
        }
    }
}
```

Torna el següent byte de dades, o
-1 si s'ha arribat al final del fitxer.

Example 2

Classes:

- Atribut.java
- MyClass.java
- TestMyClass.java

Example 2

Atribut.java

```
package examplesjava;

import java.io.Serializable;

public class Atribut implements Serializable{
    private int x;

    // Constructor:
    Atribut(int x) {
        this.x=x;
    }
    // Accesosors d'escriptura i lectura:
    public void setx(int x){
        this.x = x;
    }
    public int getx(){
        return this.x;
    }

    public String toString(){
        String retorn;
        retorn = "valor de x: " + this.x;
        return retorn;
    }
}
```

Exemple 2

MyClass .java

```
package examplesjava;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.io.Serializable;
import java.util.Scanner;
import java.util.logging.Level;
import java.util.logging.Logger;

public class MyClass implements Serializable{
    // attributs de la classe MyClass
    private Atribut atribut;

    public MyClass(){
        atribut = new Atribut(0);
    }
    public MyClass(int val){
        atribut = new Atribut(val);
    }
}
```

Continuació MyClass.java

```
public void guardarInfo(){
    Scanner sc = new Scanner(System.in);
    System.out.println("\n Quin és el nom del fitxer on es guardarà:");
    String fileNameOut = sc.nextLine();
    FileOutputStream fout=null;
    ObjectOutputStream oos=null;
    try {
        fout = new FileOutputStream(fileNameOut);
    } catch (FileNotFoundException ex) {
        Logger.getLogger(MyClass.class.getName()).log(Level.SEVERE, null, ex);
    }
    try {
        oos = new ObjectOutputStream(fout);
    } catch (IOException ex) {
        Logger.getLogger(MyClass.class.getName()).log(Level.SEVERE, null, ex);
    }
    try {
        oos.writeObject(this);
    } catch (IOException ex) {
        Logger.getLogger(MyClass.class.getName()).log(Level.SEVERE, null, ex);
    }
    try {
        fout.close();
    } catch (IOException ex) {
        Logger.getLogger(MyClass.class.getName()).log(Level.SEVERE, null, ex);
    }
}
```

Continuació MyClass.java

```
public static MyClass carregarInfo(){
    Scanner sc = new Scanner(System.in);
    System.out.println("\n Quin és el nom del fitxer d'on es carregarà:");
    String nomFitxer = sc.nextLine();
    FileInputStream fin = null;
    ObjectInputStream ois = null;
    MyClass myClass=null;
    try {
        fin = new FileInputStream(nomFitxer);
    } catch (FileNotFoundException ex) {
        Logger.getLogger(MyClass.class.getName()).log(Level.SEVERE, null, ex);
    }
    try {
        ois = new ObjectInputStream(fin);
    } catch (IOException ex) {
        Logger.getLogger(MyClass.class.getName()).log(Level.SEVERE, null, ex);
    }
    try {
        myClass = (MyClass) ois.readObject();
    } catch (IOException ex) {
        Logger.getLogger(MyClass.class.getName()).log(Level.SEVERE, null, ex);
    } catch (ClassNotFoundException ex) {
        Logger.getLogger(MyClass.class.getName()).log(Level.SEVERE, null, ex);
    }
    try {
        fin.close();
    } catch (IOException ex) {
        Logger.getLogger(MyClass.class.getName()).log(Level.SEVERE, null, ex);
    }
    return myClass;
}
```

Continuació MyClass .java

```
// Accesos d'escriptura i lectura:
public void setAtribut(Atribut atribut){
    this.atribut = atribut;
}
public Atribut getAtribut(){
    return this.atribut;
}
public void setAtribut(int i) {
    assert this.atribut != null;
    this.atribut.setx(i);
}
// Sobreescrivim el mètode toString:
public String toString(){
    String retorn;
    retorn = "valor de l'atribut: " + this.atribut;
    return retorn;
}
} // Fi de la classe MyClass
```

TestMyClass .java

```
public class TestMyClass {
    public static void main(String[] args){
        MyClass myClass = new MyClass();
        System.out.println("L'objecte myClass és: " + myClass);
        System.out.println("L'objecte myClass té el codi: " + myClass.hashCode());

        // Guardem l'objecte a disc:
        myClass.guardarInfo();

        // Modifiquem els valors de l'atribut
        myClass.setAtribut(2);
        System.out.println("L'objecte myClass és: " + myClass);
        System.out.println("L'objecte myClass té el codi: " + myClass.hashCode());

        // Després d'haver modificat els valors de myClass (aquí podriem inclús haver eliminat aquest
        objecte),
        // volem recuperar l'objecte d'inici que havíem guardat a disc:
        // Carreguem l'objecte de disc:
        myClass = MyClass.carregarInfo();
        System.out.println("L'objecte myClass té el codi: " + myClass.hashCode());
        System.out.println("L'objecte myClass és: " + myClass);

    }
} // Fi de la classe TestMyClass
```

Exercici: Penseu quina serà la sortida per pantalla.

FRAMEWORK COL·LECCIONS

Framework Col·leccions

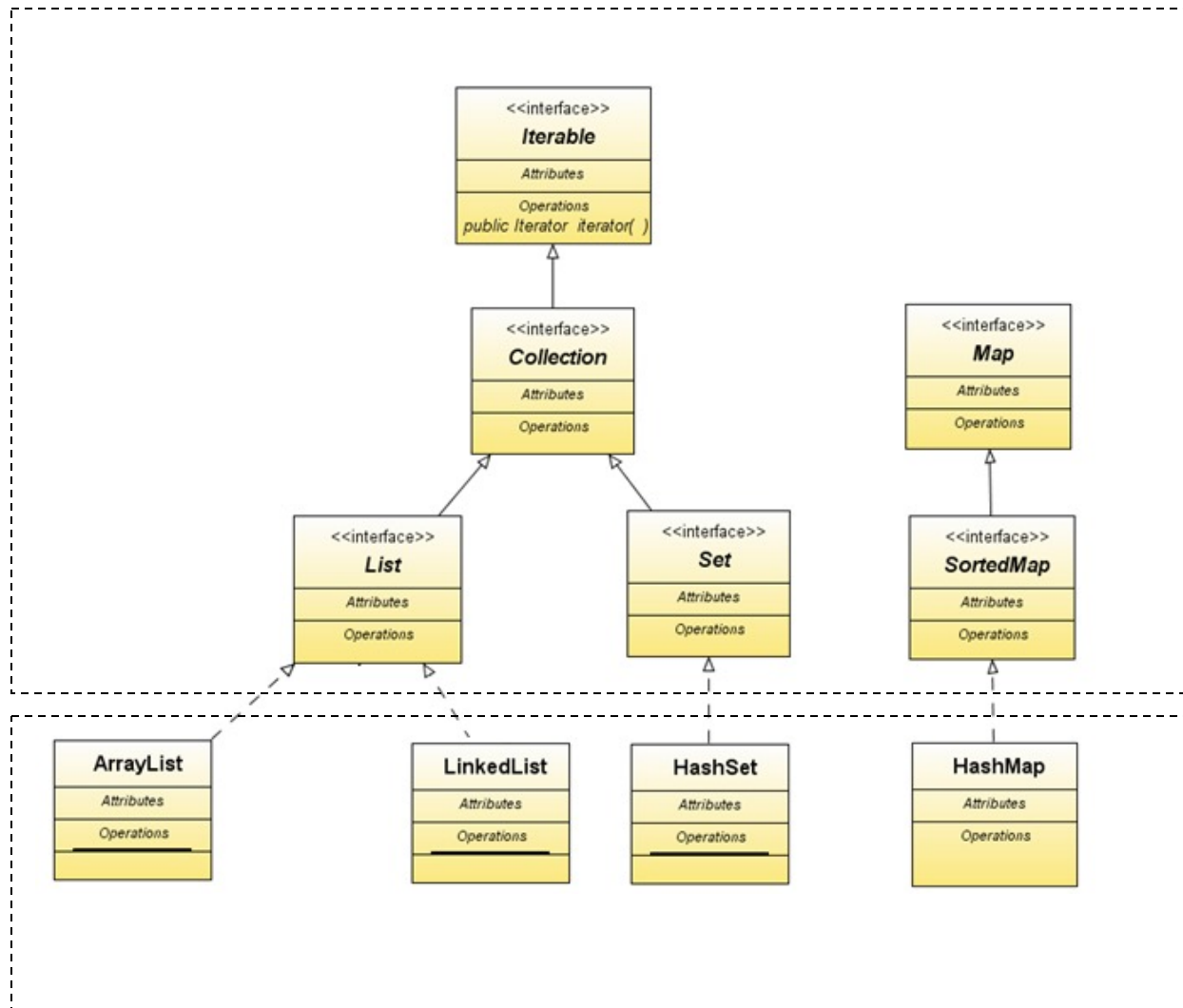
- Una **col·lecció** és un objecte que agrupa múltiples elements en una única unitat.
- Normalment representen elements d'informació dins d'un grup natural, com
 - una bústia de correu (una col·lecció de correus),
 - un directori (una col·lecció de fitxers),
 - una guia telefònica (una associació entre 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 és la que més utilitzem a les pràctiques, hi ha altres col·leccions útils:
 - LinkedList – llista enllaçada
 - HashMap – mapa hash

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 list= 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");
```

`map.put(clau, valor)`

`map.get (clau)` retorna el valor corresponent

```
// 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).
- Codi per contar el número de números diferents en una serie.

```
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 numbers detected:");
        System.out.println(m);
    }
}
```

put(clau, valor)

? : operador condicional ternari
test ? expression1 : expression2

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

Col·leccions i iteradors

- 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'últim objecte (el més recent) retornat pel mètode `next`

Col·leccions i iteradors

```
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();
}
```

- 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 parametrizats.

```
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

(Continua implementació classe CreaColeccioHomogenea)

```
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: iteradors

```
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!

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();  
}
```

ASSERTIONS

Assercions

- La tècnica d'assercions consisteix en sembrar el codi de tests d'integritat (de suposicions), de forma que si alguna cosa no és normal, ho detectarem el més aviat possible per sí mateix, en lloc de tenir que estar traçant marxa enrere les causes que han portat a un error.
- En Java s'escriu de forma compacta:
`assert estem_com_volem;`
- Un assert és una condició que s'ha de complir per a considerar que la nostra classe provada funciona de la manera esperada.

Assercions

- En Java les assercions tenen dues formes:

1. La forma simple

`assert Expressio1 ;`

on *Expressio*₁ és una expressió booleana.

Quan el sistema corre l'asserció, s'avalua *Expressio*₁ i si és false llançarà un error d'asserció (AssertionError) sense missatge de detall.

2. L'altra forma és:

`assert Expression1 : Expression2 ;`

on:

*Expression*₁ és una expressió booleana.

*Expression*₂ és una expressió que té un valor. (No pot ser una invocació d'un mètode que està declarat com a void.)

Exemple 1

```
public class AssertionExample{  
    public static void main(String[] args) {  
        int x = -15;  
  
        // Comprovem que el valor de x és positiu  
        assert x > 0  
  
        System.out.println("Valor positiu x: " + x);  
    }  
}
```

Resultat:

Exception in thread "main" java.lang.AssertionError

Exemple 1

```
public class AssertionExample{  
    public static void main(String[] args) {  
        int x = -15;  
  
        // Comprovem que el valor de x és positiu  
        assert x > 0 : "El valor ha de ser positiu";  
  
        System.out.println("Valor positiu x: " + x);  
    }  
}
```

Resultat:

Exception in thread "main" java.lang.AssertionError: El valor ha de ser positiu

S'afegeix el missatge
de sortida



Exemple 2

```
public class AssertionExample{  
    public static void main(String [] args) {  
        int i = 10;  
        assert i > 100 : missatgeError();  
        System.out.println("Això és un exemple");  
    }  
    public static String missatgeError() {  
        return "La variable està fora de rang";  
    }  
}
```

A pràctiques

Es podrien utilitzar:

- Per comprovar que s'ha creat correctament el model i el controlador:

```
...  
assert controlador!=null;  
...
```

- Per verificar que quan es passa un fitxer a la llista aquest no sigui null

```
...  
assert fitxer!=null;  
correcte = dades.afegirFitxer(biblio, fitxer);  
...
```

Netbeans

- Per activar els asserts anar a
- Properties of the project → Run → afegir a les opcions de la Virtual Machine (VM):
-ea o -enableassertions
- Provar el següent codi:

```
public static void main(String[] args) {  
    int a = 1;  
    assert(a==0): "valor incorrecte";  
}
```

Referències

- Bertrand Meyer, “**Construcción de software orientado a objetos**”, Prentice Hall, 1998.
- “**Thinking in Java**” Bruce Eckel.
- Bert Bates, Kathy Sierra. **Head First Java**. O’Reilly Media, 2005.