



UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

Escola Tècnica Superior d'Enginyeria Informàtica



Tema 3. Elements de la POO: Herència i Tractament d'Excepcions

Programació (PRG)

Curs 2019/20

Departament de Sistemes Informàtics i Computació



Continguts

Duració: 3 sessions

- Conceptes de la POO
 - Herència
 - La jerarquia de classes en Java
 - L'herència a la documentació de l'API de Java
- Tractament d'excepcions en Java
 - La jerarquia Throwable
 - Excepcions d'usuari
 - Instrucció try- catch- finally
 - Instrucció throws
 - Instrucció throw

- Pràctiques relacionades:

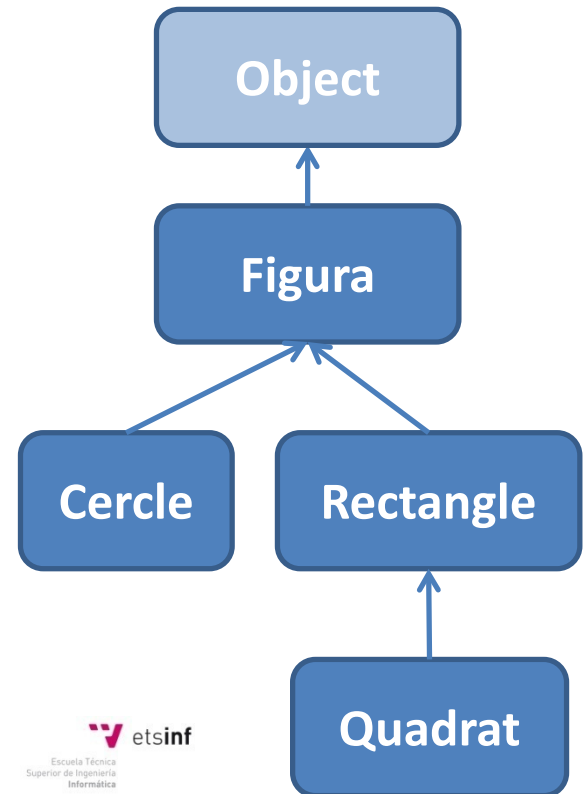
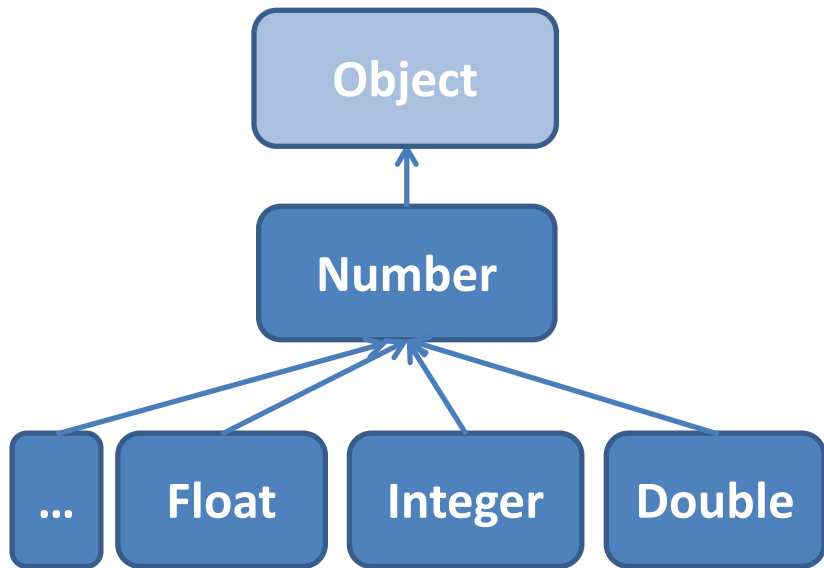
PL 4. Tractament d'excepcions i fitxers. (3 sessions)



- **Descarrega** (del Tema 3 de PoliformaT) els fitxers **exemplesT3.jar** i **exercicisT3.jar** en una carpeta **PRG/Tema 3** dins del teu **disc W**
- Des de l'opció **Projecte** de **BlueJ**, usa l'opció **Open ZIP/JAR...** per tal d'obrir-los com projectes **BlueJ** i prepara't per usar-los.
- Cadascun dels exemples i exercicis que se't proposen en aquest tema es correspon amb un dels paquets continguts als projectes **exemplesT3** i **exercicisT3**, respectivament.

Introducció a l'herència

- L'**herència** és el mecanisme que proporcionen els llenguatges de programació orientats a objectes per reutilitzar el disseny de classes ja existents per definir noves classes.
- Permet modelar de forma intuïtiva una relació de tipus **ÉS UN**, definint una **jerarquia de classes**.
- Les classes poden, d'aquesta manera, utilitzar atributs i mètodes que hagen segut definits en classes que estiguen per damunt d'elles a la jerarquia.
- Exemples:



Sobre la necessitat de l'herència

- Es demana construir la classe **Estudiant** amb els atributs *nom*, *dni* i *crèdits* matriculats. Es disposa de la classe **Persona** ja implementada.

```
public class Persona {  
    private String nom;  
    private int dni;  
    public Persona(String n, int d) {  
        this.nom = n; this.dni = d;  
    }  
    public String getNom() { return this.nom; }  
    public int getDni() { return this.dni; }  
    public String toString() {  
        return "Nom: " + this.nom + " DNI: " + this.dni; }  
}
```

- Opcions possibles:
 - **Inapropiada:** Ignorar la classe **Persona** i construir la classe **Estudiant** amb tres atributs (*dni*, *nom* i *crèdits*). Es repeteix la declaració d'atributs i mètodes ja feta a la classe **Persona**.
 - **Apropiada:** Usar l'herència per definir la classe **Estudiant** en base a la classe **Persona**.

Construint una subclasse

- Construcció de la classe **derivada** Estudiant a partir de la classe **base** Persona.
 - La classe Estudiant hereta (pot usar) tots els atributs i tots els mètodes no privats de Persona i pot afegir atributs i mètodes propis.

```
public class Persona {  
    protected String nom;  
    protected int dni;  
    public Persona(String n, int d) {  
        this.nom = n; this.dni = d;  
    }  
    public String getNom() { return this.nom; }  
    public int getDni() { return this.dni; }  
    public String toString() {  
        return "Nom: " + this.nom + " DNI: " + this.dni; }  
}
```



BlueJ: exemplesT3 [personaEstudiant]

protected indica que l'atribut és privat excepte per a les classes derivades i les classes que formen part del mateix paquet.

```
public class Estudiant extends Persona {  
    private int credits;  
    public Estudiant(String nom, int dni) {  
        super(nom, dni); this.credits = 60;  
    }  
    public int getCredits() { return this.credits; }  
}
```

Persona

extends

Estudiant

Ha de ser la primera instrucció del constructor de la classe derivada. Si no es crida explícitament al constructor de la classe base, el compilador insereix una crida al constructor sense arguments (`super();`) que, si en la classe base no existeix, provoca un error de compilació.

Construint una subclasse

The screenshot shows the BlueJ IDE interface with a project named 'exemplesT3' and a package named 'personaEstudiant'. The class hierarchy diagram shows 'Persona' as the superclass, with 'Estudiant' and 'Prof' as subclasses. 'TestEstudiantPersona' is a test class. The 'TestEstudiantPersona' class is currently selected, and the 'Create object' button is highlighted. Below the class hierarchy, there are two buttons: 'persona1: Persona' and 'estudian1: Estudiant'. The 'estudian1: Estudiant' button is selected, and the 'Create object' button is highlighted.

Two object inspection windows are open:

persona1 : Persona

protected String nom	"Lluïsa Garcia"	Inspect
protected int dni	23456678	Get

Show static fields Close

estudian1 : Estudiant

private int credits	60	Inspect
protected String nom	"Joan Lopez"	Get
protected int dni	10987653	

Show static fields Close

Modificadors d'accés

package unPaquetExemple;

```
public class A {  
    public int i1;  
    protected int i2;  
    int i3;  
    private int i4;  
    ...  
}
```

```
public class B {  
    pot accedir a i1  
    pot accedir a i2  
    pot accedir a i3  
    no pot accedir a i4  
    ...  
}
```

```
public class C  
    extends A {  
    pot accedir a i1  
    pot accedir a i2  
    pot accedir a i3  
    no pot accedir a i4  
    ...  
}
```

```
public class D  
    extends A {  
    pot accedir a i1  
    pot accedir a i2  
    no pot accedir a i3  
    no pot accedir a i4  
    ...  
}
```

```
public class E {  
    pot accedir a i1  
    no pot accedir a i2  
    no pot accedir a i3  
    no pot accedir a i4  
    ...  
}
```

Es suposa que a les classes B, C, D i E no hi ha definit cap atribut d'instància amb identificador i1, i2, i3 o i4.

Les mateixes regles que s'apliquen ací a atributs d'instància es poden aplicar també a mètodes.

“**pot accedir**” significa que pot accedir directament (per nom). Per exemple, des de les classes C i D es pot accedir a i1 amb i1 (this.i1 o super.i1). Des de les classes B i E, creant un objecte de tipus A, objA.i1.

A i4 s'accedirà fent ús d'un consultor public (definit només en A): des de les classes C i D, getI4() (this.getI4() o super.getI4()) i des de les classes B i E, creant un objecte de tipus A, objA.getI4().

Utilitzant una subclasse



BlueJ: exemplesT3 [personaEstudiant]

```
public class TestEstudiantPersona {  
    private TestEstudiantPersona() { }  
    public static void main(String[] args) {  
        Persona p = new Persona("Lluïsa Garcia", 23456678);  
        System.out.println("Persona: " + p);    // p.toString()  
        Estudiant e = new Estudiant("Joan Lopez", 10987653);  
        System.out.println("Estudiant: " + e.getNom() + " "  
            + e.getDni() + ": " + e.getCredits() + " crèdits");  
    }  
}
```

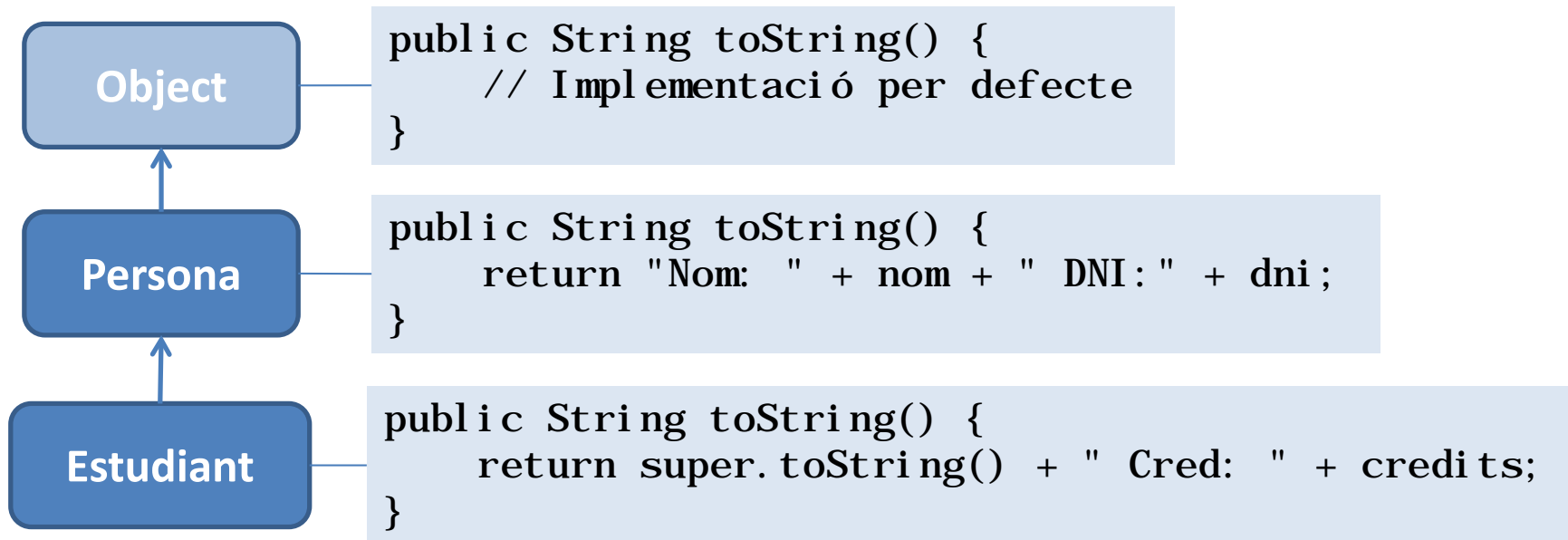
- Es pot invocar als mètodes declarats en Persona des d'un objecte Estudiant ja que Estudiant els hereta de Persona.
 - La classe Estudiant pot accedir directament (per nom) als atributs i mètodes no privats de la classe Persona.
- Mostra per pantalla:
Persona: Nom: Lluïsa Garcia DNI: 23456678
Estudiant: Joan Lopez 10987653: 60 crèdits

Sobreescritura de mètodes

- Qualsevol mètode **no privat** de la classe **base** que es definisca de nou en la classe **derivada** es **sobreescriu**:
- **Sobreescritura completa**:
 - Per a tal cosa, definim en la classe **derivada** un mètode:
 - Amb el mateix perfil que en la classe **base** (nom, llista de paràmetres i tipus del resultat).
 - Amb un codi nou a propòsit per a la classe **derivada**.
- **Sobreescritura parcial**:
 - Quan només es vol canviar parcialment el comportament del mètode de la classe **base**. S'utilitza **super** per tal d'invocar el mètode de la classe **base**.
- Qualsevol classe Java hereta de la classe predefinida **Object**, entre altres, els mètodes:
 - `public String toString()`
 - `public boolean equals(Object o)`
- Sol ser habitual la seua sobreescritura.

Sobreescritura de mètodes

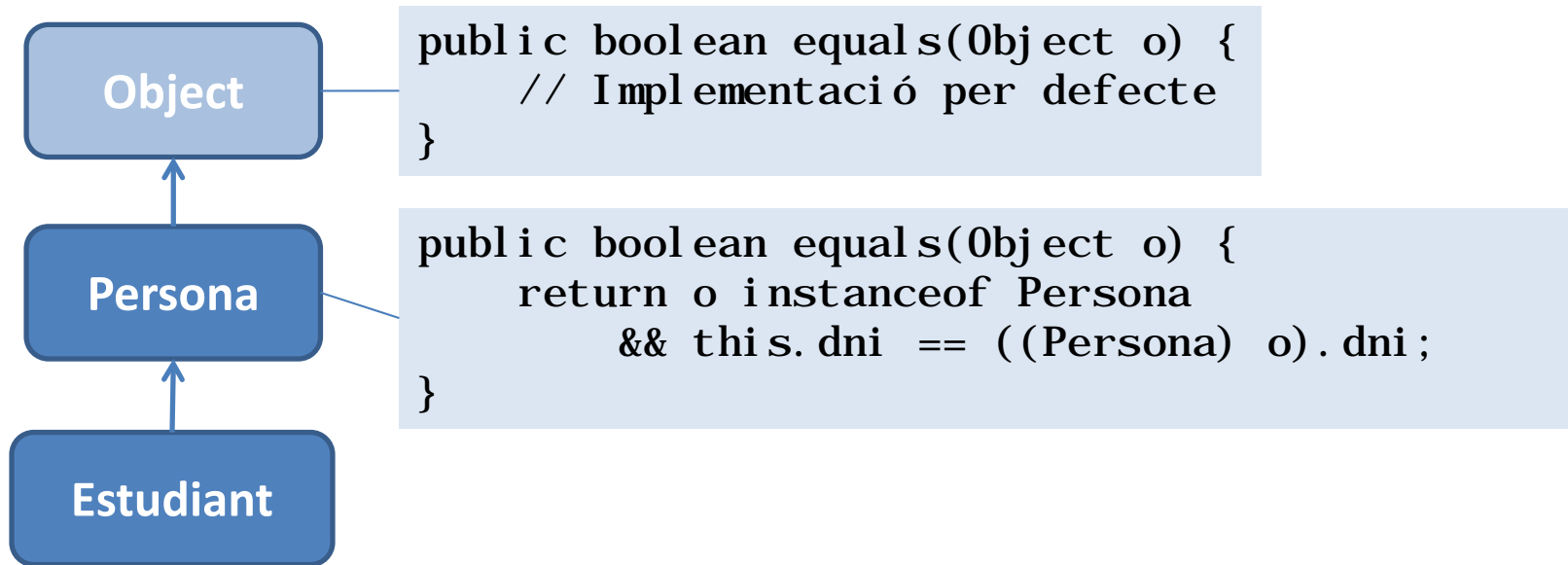
- Exemple: sobreescritura del mètode `toString()` d'`Object`



- La classe `Persona` sobreescriu completament el mètode que hereta d'`Object`. La classe `Estudiant`, sobreescriu parcialment el mètode que hereta de `Persona`.

Sobreescritura de mètodes

- Exemple: sobreescritura del mètode `equals(Object o)` d'`Object`



- La classe `Persona` sobreescriu completament el mètode que hereta d'`Object`. La classe `Estudiant` hereta el mètode `equals` sobreescrit en `Persona`.

Sobreescritura de mètodes

```
public class TestEstudiantPersona2 {  
    private TestEstudiantPersona2() { }  
    public static void main(String[] args) {  
        Persona p = new Persona("Lluïsa Garcia", 23456678);  
        Estudiant e = new Estudiant("Joan Lopez", 10987653);  
        System.out.println("Persona: " + p);    // p.toString()  
        System.out.println("Estudiant: " + e);  // e.toString()  
        System.out.println(p.equals(e));  
    }  
}
```

de Persona

d'Estudiant

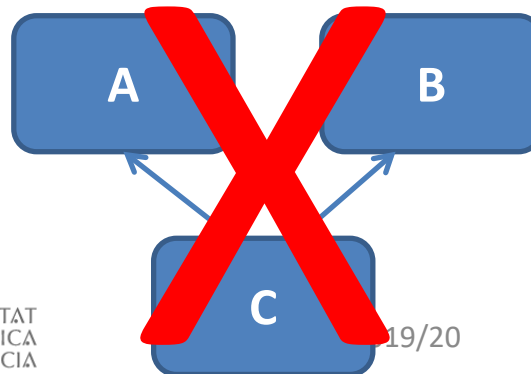
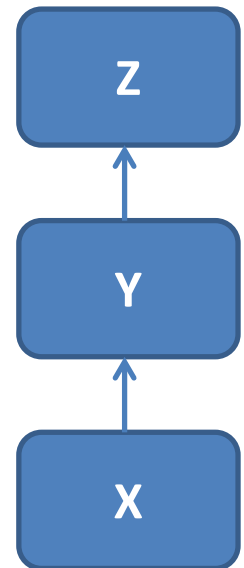
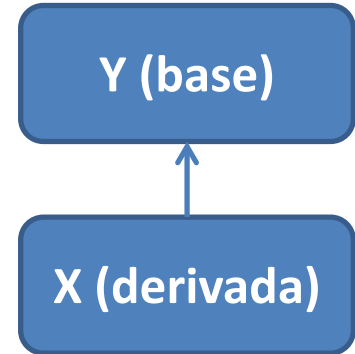
- El programa mostra per pantalla:

Persona: Nom: Lluïsa Garcia DNI: 23456678

Estudiant: Nom: Joan Lopez DNI: 10987653 Cred: 60
false

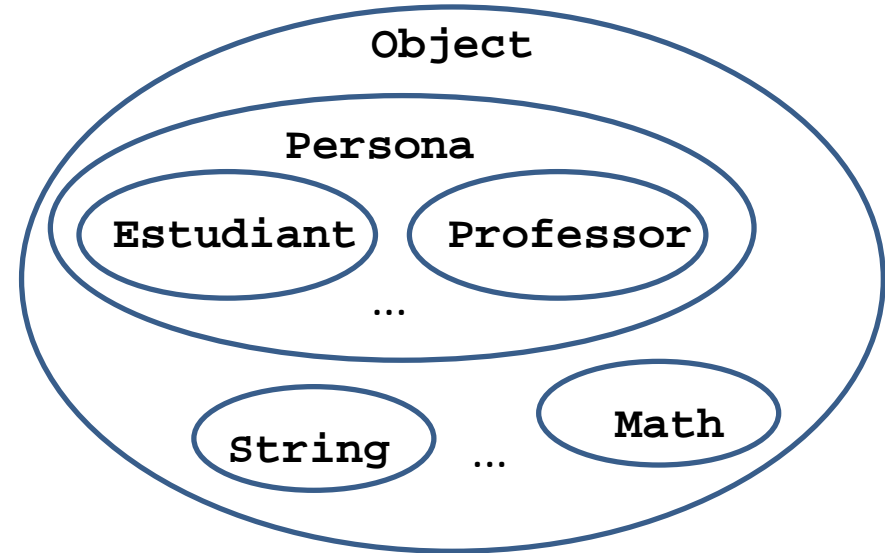
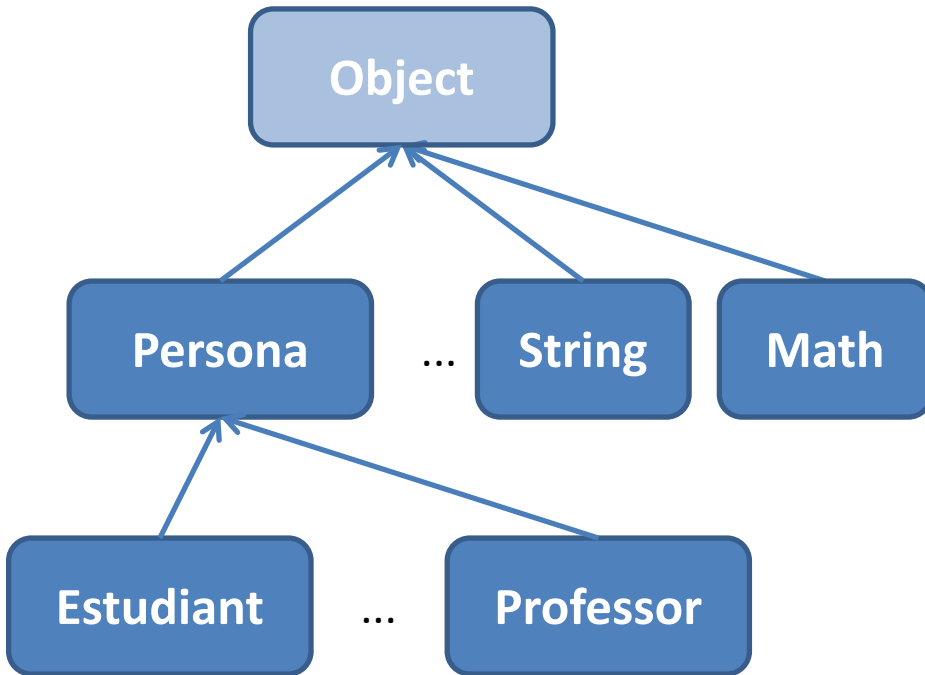
Jerarquia de classes

- Si X **ÉS UN(A)** Y,
 - Es diu que la classe derivada X és una variació de la classe base Y.
 - Es diu que X i Y formen una **jerarquia**, on la classe X és una **subclasse** (o classe **derivada**) de Y i Y és una **superclasse** (o classe **pare** o **base**) de X.
 - La relació és **transitiva**: si X **ÉS UN(A)** Y i Y **ÉS UN(A)** Z, aleshores X **ÉS UN(A)** Z.
 - X pot referenciar directament (per nom) tots els atributs i mètodes que no siguin privats en Y.
 - X és una classe completament nova i independent (els canvis en X no afecten a Y).
- Java no soporta herència múltiple.
 - Una classe només pot heretar com a màxim d'un altra.



Jerarquia de classes

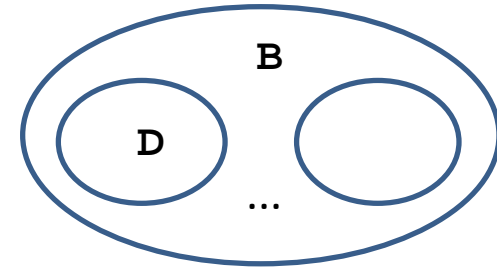
- La relació entre classes base-derivada estableix una ordenació o *jerarquia* que té en el cim a la classe `Object`.



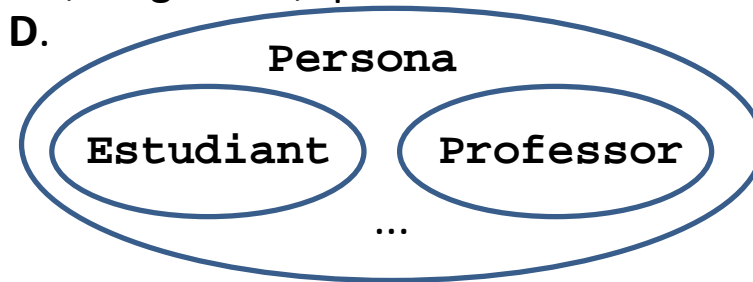
```
Estudiant e = new Estudiant("Joan Lopez", 10987653);  
double n = e.getCredits();  
Persona p = e; // Tot objecte Estudiant és un objecte Persona  
e = p; <-- Error de compilació  
Error: incompatible types: Persona cannot be converted to Estudiant
```

Jerarquia de classes

- **Compatibilitat de tipus en presència d'herència:** dos tipus són compatibles si pertanyen a la mateixa línia de la jerarquia en sentit descendent. Sigui **B** la classe base i **D** una classe derivada, açò és, $D \subseteq B$, en general:
 - si **d** és una variable declarada de tipus **D** (o un objecte creat de tipus **D**), **d** és també de tipus **B**;
 - a una variable de tipus **B**, se li pot assignar **d**;
 - a **d** se li poden aplicar els mètodes de **B** (heretats per **D**).



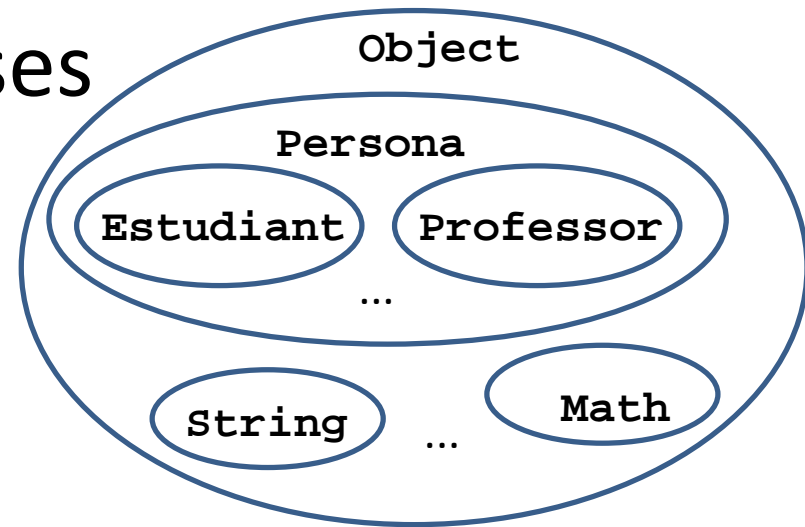
- El contrari no es compleix sempre: no es pot assegurar, en general, que una variable de tipus **B** estiga referenciant a un objecte de tipus **D**.
- En l'exemple anterior:



```
Estudiant e = new Estudiant("Joan Lopez", 10987653);  
double n = e.getCredits();  
Persona p = e; // Tot objecte Estudiant és un objecte Persona  
n = p.getCredits(); <-- Error de compilació  
                        Error: cannot find symbol - method getCredits()  
e = p; <-- Error de compilació  
        Error: incompatible types: Persona cannot be converted to Estudiant
```


Jerarquia de classes

- Es pot forçar al compilador mitjançant un **càsting** si el programador preveu que l'objecte referenciat va a ser del tipus inferior. Però, si s'equivoca, es produeix l'error d'execució **ClassCastException**.



```
Estudiant e = new Estudiant("Joan Lopez", 10987653);
Persona p = e;
e = (Estudiant) p; // Sense el càsting, error de compilació
Object o1 = p, o2 = "Hola";
Estudiant e2 = (Estudiant) o1;
Estudiant e3 = (Estudiant) o2; <-- Error d'execució
    java.lang.ClassCastException: java.lang.String cannot be cast to Estudiant
```

- L'operador **instanceof** permet verificar si un objecte és d'una classe determinada.

```
Estudiant e = new Estudiant("Joan Lopez", 10987653);
Persona p = e;
e = (Estudiant) p; // Sense el càsting, error de compilació
Object o1 = p, o2 = "Hola";
Estudiant e2 = null, e3 = null;
if (o1 instanceof Estudiant) { e2 = (Estudiant) o1; } // e2 != null
if (o2 instanceof Estudiant) { e3 = (Estudiant) o2; } // e3 == null
```

Exemple

- La classe ArrayPersones del paquet personaEstudiant crea un array de estudiants i professors i mostra per pantalla els departaments dels professors.

```
public class ArrayPersones {
    /** No hi ha objectes d'aquesta classe. */
    private ArrayPersones() { }

    public static void main(String[] args) {
        Estudiant e1 = new Estudiant("Manel Font", 33445566);
        Estudiant e2 = new Estudiant("Lluís Sales", 44226655);
        Professor pr1 = new Professor("Víctor Ferragut", 11223344, "idm");
        Professor pr2 = new Professor("Andreu Pla", 44332288, "dsic");
        Professor pr3 = new Professor("Mireia Albiol", 99432885, "disca");
        Persona[] a = {pr1, e1, pr2, e2, pr3};

        for (int i = 0; i < a.length; i++) {
            if (a[i] instanceof Professor) {
                System.out.println(((Professor) a[i]).getDepartament());
                // Si no es fa el càsting, error de compilació
            }
            // perquè a[i] és de tipus Persona

            //System.out.println(((Professor) a[i]).getDepartament());
            // És necessari comprovar que a[i] és de tipus Professor
            // Sense el if, error d'execució:
            // java.lang.ClassCastException: Estudiant cannot be cast to Professor
        }
    }
}
```

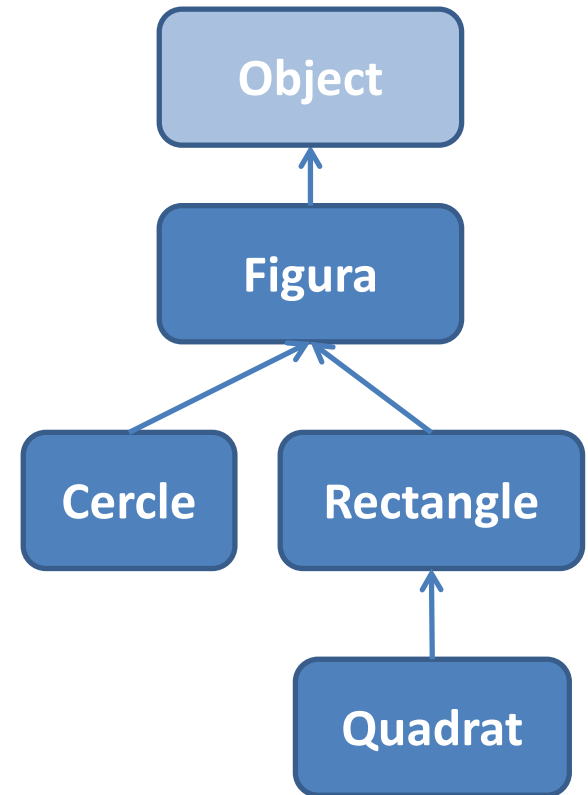
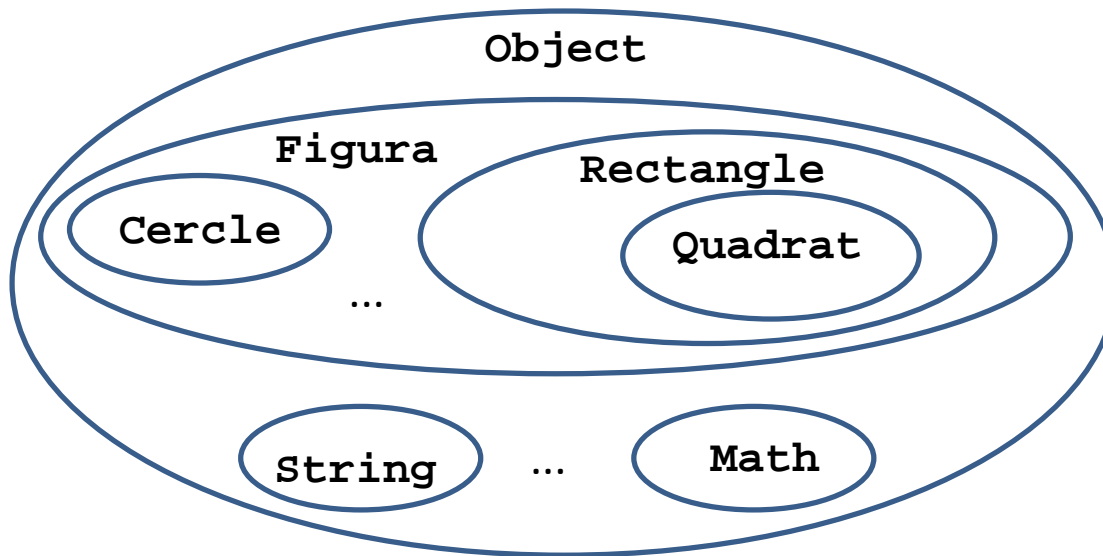
BlueJ: exemplesT3 [personaEstudiant]

Jerarquia de classes

- **EXEMPLE** de jerarquia de classes:
 - Una **Fi gura** **ÉS** UN **Obj ect**
 - Un **Cercl e** **ÉS** UNA **Fi gura**
 - Un **Rect angl e** **ÉS** UNA **Fi gura**
 - Un **Quadrat** **ÉS** UN **Rectangle**
(amb base i altura iguals)



BlueJ: exemplesT3 [figures]



L'herència a la documentació de Java

- Extracte de la documentació de la classe `Number` a l'API Java.

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Number.html>

java.lang

Class Number

La classe `Number` hereta d'`Object` (com totes)

java.lang.Object

java.lang.Number

All Implemented Interfaces:

Serializable

Direct Known Subclasses:

AtomicInteger, AtomicLong, BigDecimal, BigInteger, Byte, Double, Float, Integer, Long, Short

Capçalera de la classe `Number`

Subclasses directes de la classe `Number`

```
public abstract class Number
extends Object
implements Serializable
```

The abstract class `Number` is the superclass of classes `BigDecimal`, `BigInteger`, `Double`, `Float`, `Integer`, `Long`, and `Short`.

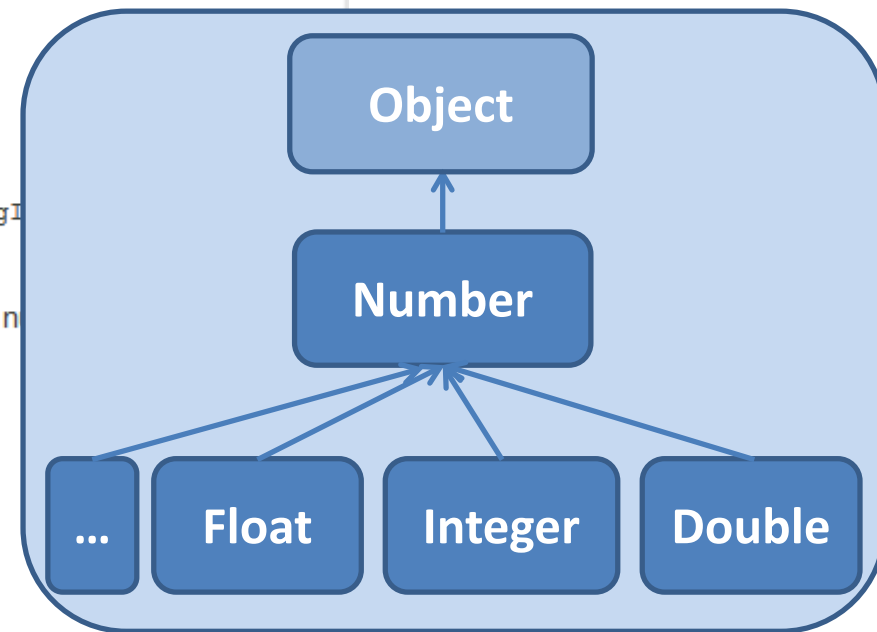
Subclasses of `Number` must provide methods to convert the represented number to byte, double, float, int, long, and short.

Since:

JDK1.0

See Also:

Byte, Double, Float, Integer, Long, Short, Serialized Form



L'herència a la documentació de Java

- Extracte de la documentació de la classe `Integer` a l'API Java.

java.lang

Class `Integer`

java.lang.Object

java.lang.Number

java.lang.Integer

Jerarquia de la classe `Integer` (hereta de `Number` que a la seua vegada hereta d'`Object`)

All Implemented Interfaces:

`Serializable`, `Comparable<Integer>`

```
public final class Integer
extends Number
implements Comparable<Integer>
```

Capçalera de la classe `Integer`

The `Integer` class wraps a value of the primitive type `int` in an object. An object of type `Integer` contains a single field whose type is `int`.

In addition, this class provides several methods for converting an `int` to a `String` and a `String` to an `int`, as well as other constants and methods useful when dealing with an `int`.

Implementation note: The implementations of the "bit twiddling" methods (such as `highestOneBit` and `numberOfTrailingZeros`) are based on material from Henry S. Warren, Jr.'s *Hacker's Delight*, (Addison Wesley, 2002).

Since:

JDK1.0

See Also:

`Serialized Form`

L'herència a la documentació de Java

- Extracte de la documentació de l'exemple.

OVERVIEW PACKAGE **CLASS** TREE INDEX HELP

ALL CLASSES SEARCH

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package personaEstudiant

Class Persona

java.lang.Object
personaEstudiant.Persona

Direct Known Subclasses:
Estudiant, Professor

public class **Persona**
extends Object

Classe Persona: classe tipus de dades per representar una persona amb

Version:
Curs 2019/20

Author:
PRG

OVERVIEW PACKAGE **CLASS** TREE INDEX HELP

ALL CLASSES SEARCH

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD

Package personaEstudiant

Class Estudiant

java.lang.Object
personaEstudiant.Persona
personaEstudiant.Estudiant

public class **Estudiant**
extends Persona

Classe Estudiant: classe tipus de dades per representar un estudiant amb el seu nom, el seu dni i els credits dels que esta matriculat.

Version:
Curs 2019/20

Author:
PRG

Field Summary

Fields inherited from class personaEstudiant.Persona
dni, nom

Constructor Summary

Constructors

Constructor	Description
Estudiant(String nom, int dni)	Crea un objecte Estudiant amb nom i dni donats i matriculat de 60 crèdits.

Method Summary

All Methods **Instance Methods** **Concrete Methods**

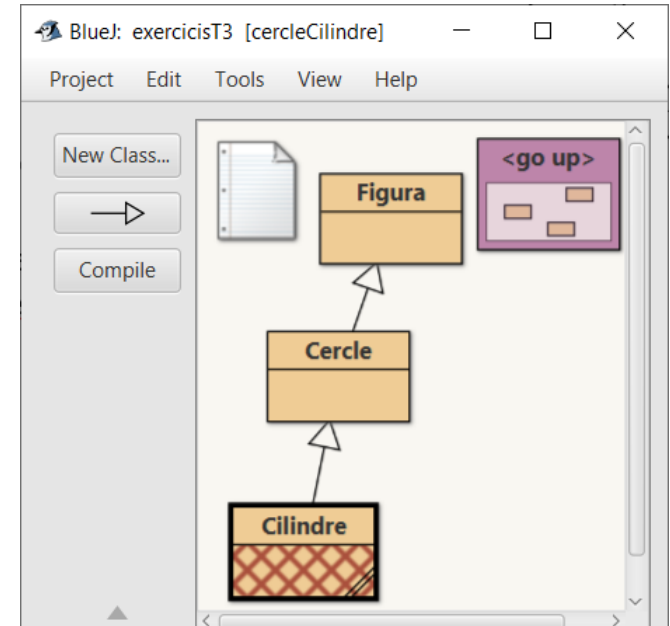
Modifier and Type	Method	Description
int	getCredits()	Torna el nombre de crèdits dels que l'estudiant està matriculat.
String	toString()	Torna un String amb les dades de l'estudiant.

Methods inherited from class personaEstudiant.Persona
equals, getDni, getNom

Methods inherited from class java.lang.Object
clone, getClass, hashCode, notify, notifyAll, wait, wait, wait

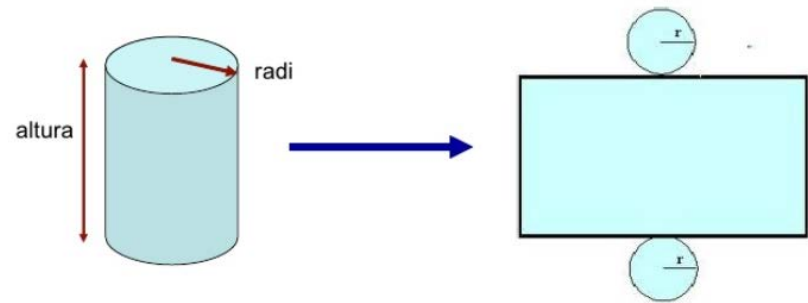
Exercici: classe Cilindre

1. A la classe **Cercle**, quin modificador de visibilitat hauria de tenir l'atribut `radi` perquè siga directament accessible des de la classe **Cilindre** i afavorisca el principi d'ocultació d'informació?
2. De les següents implementacions del constructor de **Cilindre**, una d'elles és incorrecta, quina? Per què?
 - a) `super(radi Base, color);`
`super.tipus = "Cilindre";`
`altura = alt;`
 - b) `super.color = color;`
`super.radi = radi Base;`
`super.tipus = "Cilindre";`
`altura = alt;`
3. A la classe **Cilindre**, modifica les implementacions dels mètodes `getArea()`, `getVolum()` i `toString()` per afavorir el principi de reutilització de software.



$$\text{Àrea cercle} = \pi r^2$$

$$\begin{aligned} \text{Àrea rectangle} &= \text{Base} \cdot \text{altura} = \text{longitud} \\ \text{circumferència} \cdot \text{altura} &= 2\pi r \cdot h \end{aligned}$$



$$\text{Àrea total} = 2 \text{ cercles} + 1 \text{ rectangle}$$

$$\text{Volum} = \text{Abase} \cdot \text{altura cilindre}$$

on Abase = àrea d'un cercle

Exercici: Actors i Pel·lícules - Errors

BlueJ: exercicisT3 [actorPelicles]

- Corregeix el codi de les classes **Persona**, **Actor** i **Pel·lícules** per tal que el mètode **mostrarRepartiment** siga correcte sintàctica i semànticament:

```
public class Persona {  
    private String nom;  
    public Persona(String n) {  
        this.nom = n;  
    }  
}
```

```
public class Actor extends Persona {  
    private String pelicula;  
    public Actor(String n, String p) {  
        this.nom = n;  
        this.pelicula = p;  
    }  
}
```

```
public class Pel·lícules {  
    private Pel·lícules() { }  
    public static void mostrarRepartiment(Actor[] llista, String pelicula){  
        for (int i = 0; i < llista.length; i++) {  
            if (llista[i].pelicula == pelicula) {  
                System.out.println(llista[i].toString());  
            }  
        }  
    }  
    public static void main(String[] args) { ... }  
}
```



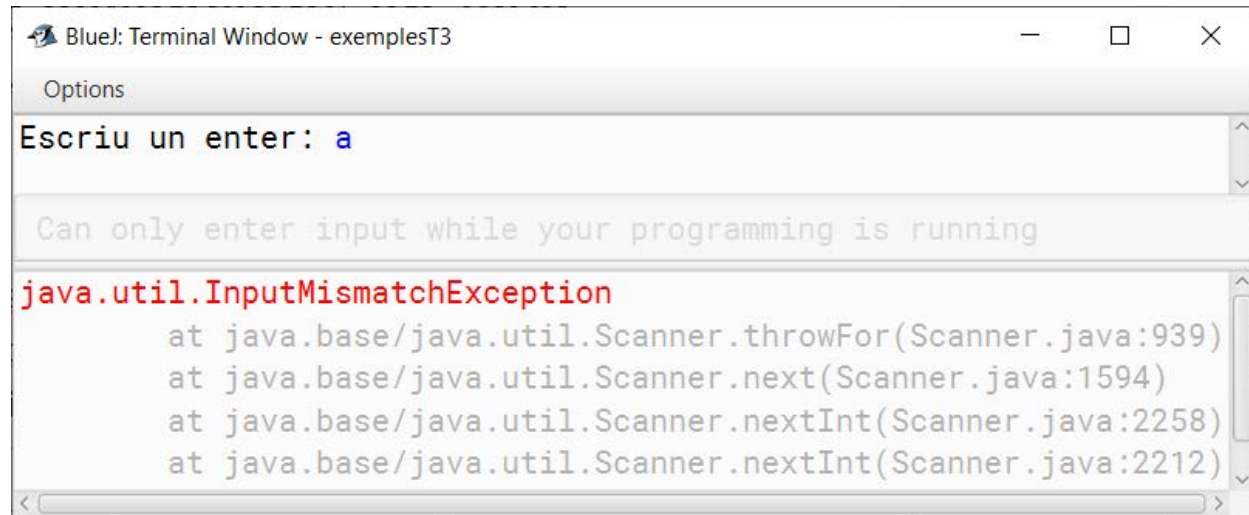
Exercici

poli **[formaT]** PRG >  **EXÀMENS** 1B - T3. Qüestionari: herència

Tractament d'excepcions en Java

- Què ocorreix quan es produeix un error inesperat, una excepció al comportament d'un programa que no és fàcil preveure?
- Per exemple, demanar a l'usuari un número enter des de teclat i que escriga una lletra.

```
import java.util.Scanner;  
Scanner teclat = new Scanner(System.in);  
System.out.print("Escriu un enter: ");  
int n = teclat.nextInt();  
Exception: java.util.InputMismatchException (null)
```

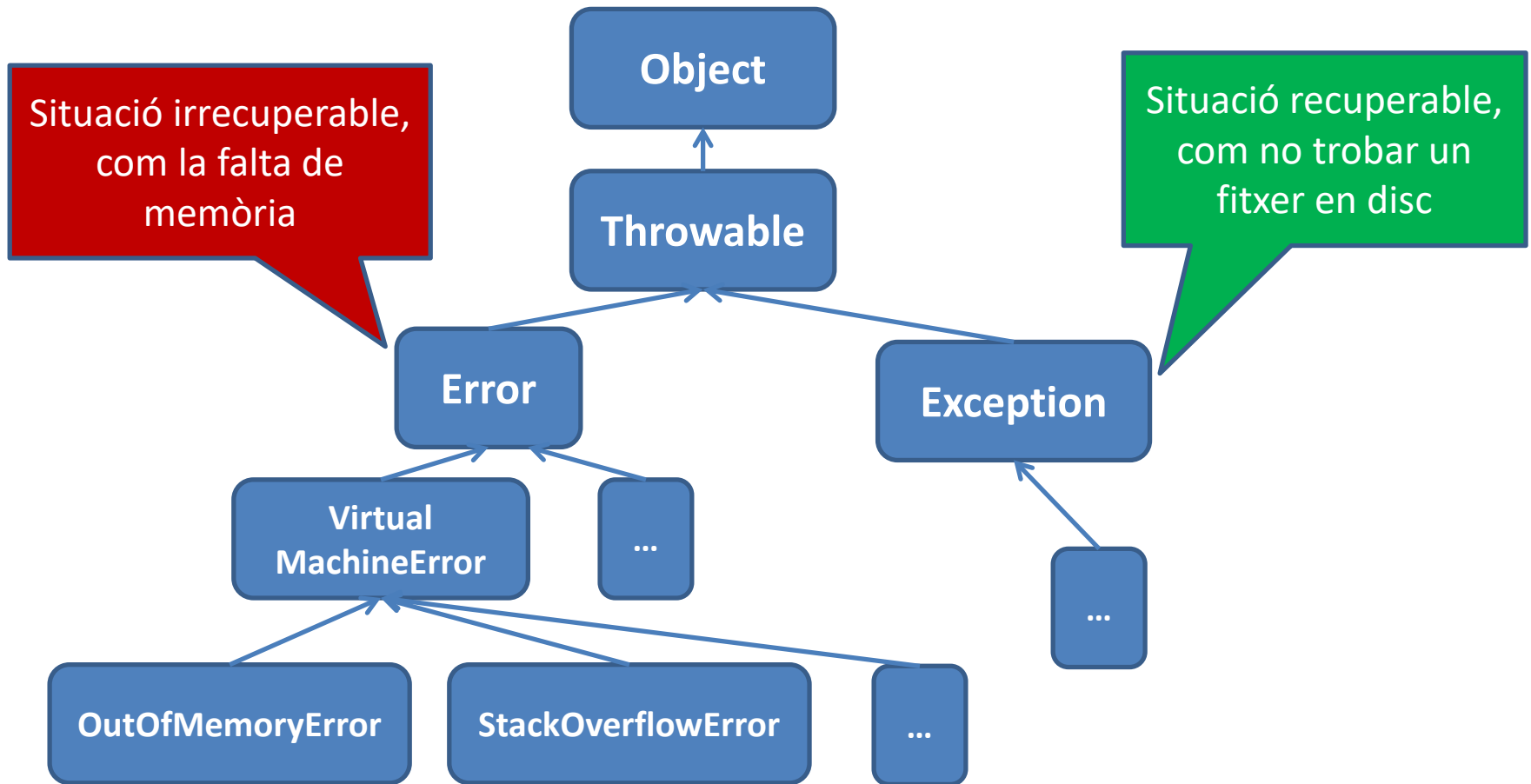


```
BlueJ: Terminal Window - exemplesT3  
Options  
Escriu un enter: a  
Can only enter input while your programming is running  
java.util.InputMismatchException  
    at java.base/java.util.Scanner.throwFor(Scanner.java:939)  
    at java.base/java.util.Scanner.next(Scanner.java:1594)  
    at java.base/java.util.Scanner.nextInt(Scanner.java:2258)  
    at java.base/java.util.Scanner.nextInt(Scanner.java:2212)
```

- Java permet evitar que un error com aquest no acabe necessàriament avortant l'execució del programa.

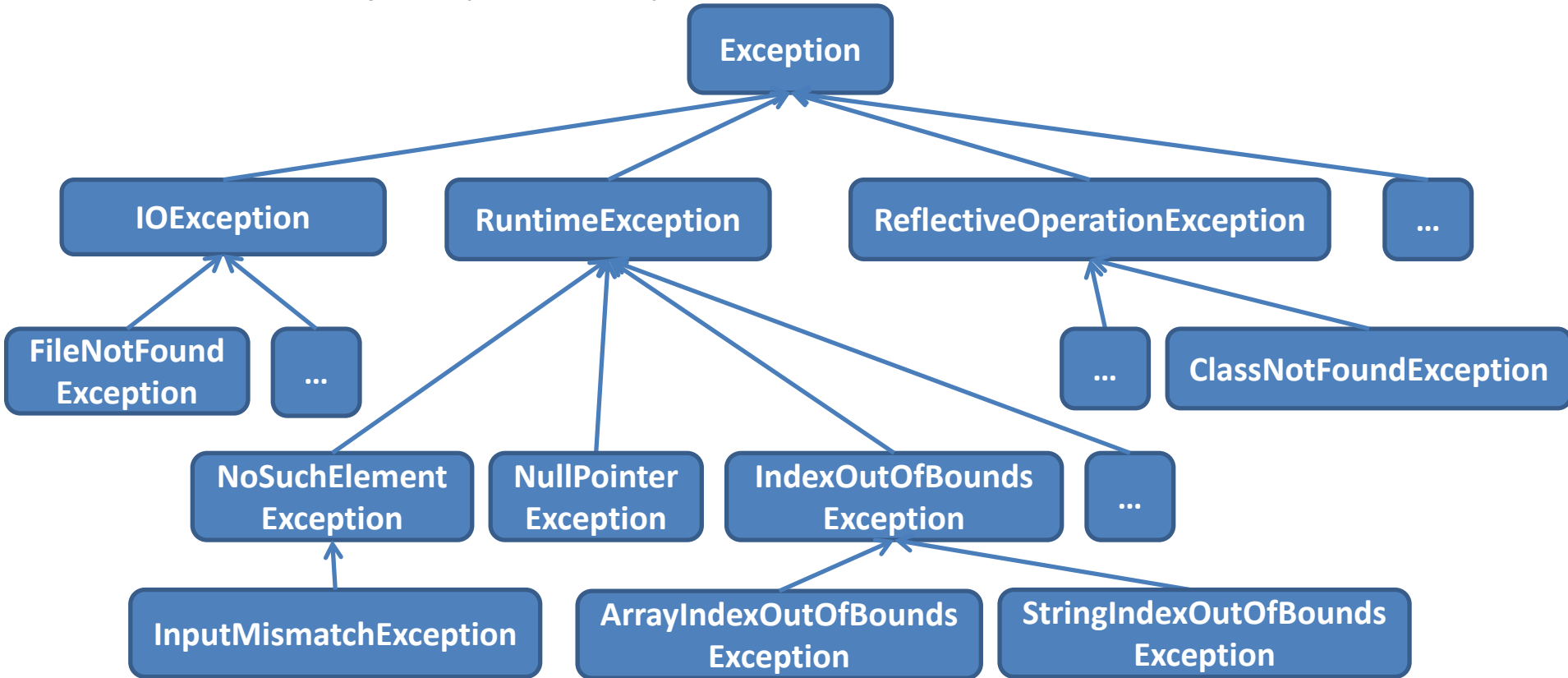
La jerarquia Throwable

- Java inclou una jerarquia per representar els errors d'execució com objectes de subclasses de Throwable.



La jerarquia d' excepcions en Java

- Extracte de la jerarquia d' excepcions en Java.



- Aquest tema es centra en la classe `Exception` i les seues derivades, en concret:
 - `IOException`, per a les excepcions d'E/S de dades (com `FileNotFoundException`).
 - `RuntimeException` que representa les excepcions d'errors de programació (per exemple, `ArrayIndexOutOfBoundsException`).

La jerarquia d'excepcions en Java

<https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Exception.html>

```
public class Exception
    extends Throwable
```

The class `Exception` and its subclasses are a form of `Throwable` that indicates conditions that a reasonable application might want to catch.

The class `Exception` and any subclasses that are not also subclasses of `RuntimeException` are *checked exceptions*. Checked exceptions need to be declared in a method or constructor's `throws` clause if they can be thrown by the execution of the method or constructor and propagate outside the method or constructor boundary.

Since:

1.0

See Also:

Error, Serialized Form

See The Java™ Language Specification:

11.2 Compile-Time Checking of Exceptions

Constructor Summary

Constructors

Modifier	Constructor	Description
	<code>Exception()</code>	Constructs a new exception with null as its detail message.
	<code>Exception(String message)</code>	Constructs a new exception with the specified detail message.

...

Method Summary

Methods declared in class `java.lang.Throwable`

`addSuppressed`, `fillInStackTrace`, `getCause`, `getLocalizedMessage`, `getMessage`, `getStackTrace`, `getSuppressed`, `initCause`, `printStackTrace`, `printStackTrace`, `printStackTrace`, `setStackTrace`, `toString`

Excepcions d'usuari

- També existeix la possibilitat de definir excepcions d'usuari, és a dir, per a representar errors de la lògica de l'aplicació que el programador vullga gestionar.
 - Per exemple, es defineix un menú amb opcions entre 0 i 5. Es podria definir una excepció d'usuari (és a dir, dins dels paquets que conformen l'aplicació de l'usuari) anomenada `OpcioForaDeRangException` de la següent manera:

```
public class OpcioForaDeRangException extends Exception {  
  
    public OpcioForaDeRangException() { super(); }  
  
    public OpcioForaDeRangException(String msg) { super(msg); }  
  
}
```

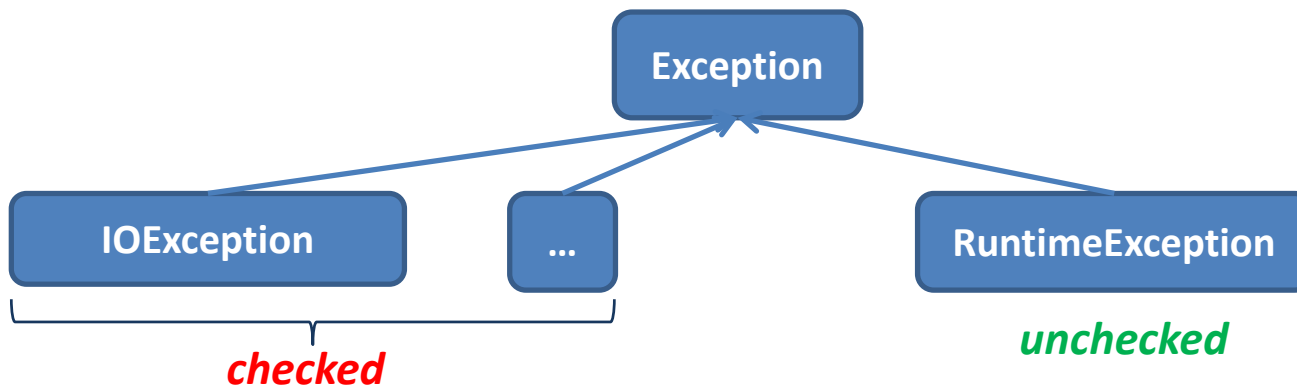
- Les excepcions d'usuari són subclasses d'`Exception` i poden incloure un missatge descriptiu de la situació que va conduir a l'error.

Tractament d'excepcions en Java

- Què fer amb una excepció una vegada s'ha produït?

Les excepcions poden ser *llançades*, *propagades* i/o *capturades*.

En realitat alguns tipus d'excepcions **HAN DE** ser propagades o capturades i altres **PODEN** ser propagades o capturades. En concret, **totes les que deriven directament d'Exception (excepte les RuntimeException) inclouen les d'usuari (que no deriven de RuntimeException)** s'anomenen excepcions **checked** o comprovades i sempre **s'han de** propagar o capturar, mentre que les derivades de **RuntimeException** s'anomenen **unchecked** o no comprovades i **es poden** capturar o propagar o simplement deixar que avorten el programa.



Tractament d'excepcions en Java

- Què fer amb una excepció una vegada s'ha produït?

Una excepció es **llança** quan es produeix el fet que la provoca, per exemple, al tractar d'obrir un fitxer que no existeix. No obstant, també poden ser creades i llançades voluntariament:

- Suposem que l'usuari introdueix un valor al programa fora de l'interval esperat (per exemple, un número major o igual que 60 o menor a zero en la variable `mi nuts`). Per tal de llançar l'excepció s'utilitza **throw**.

```
Scanner t = new Scanner(System.in);
System.out.print("Mi nuts? ");
int mi nuts = t.nextInt();
if (mi nuts < 0 || mi nuts >= 60) {
    throw new InputMismatchException("Val or incorrecte");
}
```


Tractament d'excepcions en Java

- I si l'excepció es produeix DINS d'un mètode?

Si ens interessa que siga el propi mètode el que gestione l'excepció, es pot **capturar**. En cas contrari, es pot **propagar** cap al mètode que el va invocar:

Suposem que el programa principal invoca al mètode `escriuComp` i dintre es produeix una excepció:

```
public static void main(String[] args) {
    int talla = ...; int[] elArray = new int[talla];
    ...
    Scanner t = new Scanner(System.in);
    System.out.print("Índex (entre 0 i " + (talla - 1) + "? ");
    int n = t.nextInt();
    escriuComp(n, elArray);
    ...
}

// si l'usuari escriu un enter < 0 o >= elArray.length
// es produeix l'excepció ArrayIndexOutOfBoundsException
public static void escriuComp(int i, int[] a) {
    System.out.println(a[i]);
}
```

Tractament d'excepcions en Java

- La gestió per defecte de l'excepció (açò és, no fer res) provoca la finalització abrupta del programa.
- No obstant, es podria fer que el mètode *capturara* l'excepció i continuara funcionant fins i tot amb dades que provocarien un error.
- Java proporciona un mecanisme que permet gestionar les excepcions que poden ocórrer en un bloc de codi. Es tracta de la instrucció `try-catch` (ó `try-catch-finally`).

Captura d'excepcions

```
try {  
    // codi on poden produir-se  
    // les excepcions e1, e2, ...  
} catch (ExcepcioTipus1 e1) {  
    // codi de gestió de e1  
} catch (ExcepcioTipus2 e2) {  
    // codi de gestió de e2  
...  
} finally {  
    // codi que s'executa sempre,  
    // tant si el try es completa  
    // com si no  
}
```

finally: conté instruccions que s'executen tant si es completa el **try** com si no:

- És opcional. S'utilitza per si es produeixen excepcions que no són comprovades ni capturades i que avortarien el programa però que, tot i així, cal fer determinades accions abans d'acabar. Per exemple, guardar les dades i tancar un fitxer en disc.

try: conté les instruccions on es poden produir un o més tipus d'excepcions:

- Si es produeix una excepció, el fluxe d'execució bota directament al **catch** d'aquesta excepció (o al d'una de les seues superclasses).
- Si no s'en produeix cap, s'executa normalment.

catch: conté les instruccions a realitzar si es produeix una excepció:

- Ha d'haver al menys un per cada **try** (si no hi ha **finally**).
- Indica quin tipus d'excepció es captura.
- Si hi ha més d'una excepció possible, hi haurà més d'un bloc **catch**.
- Per darrere d'un bloc **catch (ExTipus e)**, no pot apareixer un bloc **catch (ExTipus' e)** tal que **ExTipus' \subseteq ExTipus** (error de compilació).
- Com totes les excepcions són subclasses d'**Exception**, sempre es pot realitzar un **try-catch** general d'**Exception** i capturar qualsevol tipus d'excepció, però no és recomanable, ja que es capturarien les que es sap que es poden produir però també totes amb les quals no es contava, conduint a errors d'execució de difícil detecció.

A partir de la versió 7.0 de Java, un únic bloc **catch** pot utilitzar-se per a capturar més d'un tipus d'excepció, separant-les per una barra vertical (**|**), sempre que no hi haja cap que siga subclasse d'una altra

Tractament d'excepcions en Java

- Suposem que es vol que siga el mètode `escriuComp` el que resolga el problema sense avortar l'execució del programa. En aquest cas, s'afegeix un `try-catch` per capturar la possible excepció i permetre que continue l'execució del programa. Es diu que l'excepció es capturada localment.

```
// si el paràmetre i és < 0 o >= elArray.length
// es produeix l'excepció ArrayIndexOutOfBoundsException
public static void escriuComp(int i, int[] a) {
    try {
        System.out.println(a[i]);
    } catch (ArrayIndexOutOfBoundsException e) {
        System.out.println("error, s'ha produït l'excepció " + e
            + " el valor " + i + " no és vàlid");
    }
}
```

Tractament d'excepcions en Java

- Java també ofereix un mecanisme per a que l'excepció no tinga que ser tractada exactament on es produeix.
- En l'exemple presentat, la dada errònia havia estat introduïda al programa principal, però l'excepció es va produir dins d'un mètode.
 - No seria més raonable que es corregira l'error allí on s'ha produït? Per exemple, per poder tornar a demanar una dada vàlida.
- En Java es poden afegir clàusules en la capçalera dels mètodes per a que **propaguen** les excepcions que es produïxquen al seu interior.
 - L'execució del mètode s'avortarà, però la del programa pot continuar si es captura l'excepció en un mètode anterior en la pila de crides.
- La clàusula que s'afegeix a la capçalera és: **throws**.
- Si l'excepció és *checked*, obligatòriament el mètode tindrà un `throws` en la seua capçalera o la capturarà amb `try/catch`.

Tractament d'excepcions en Java

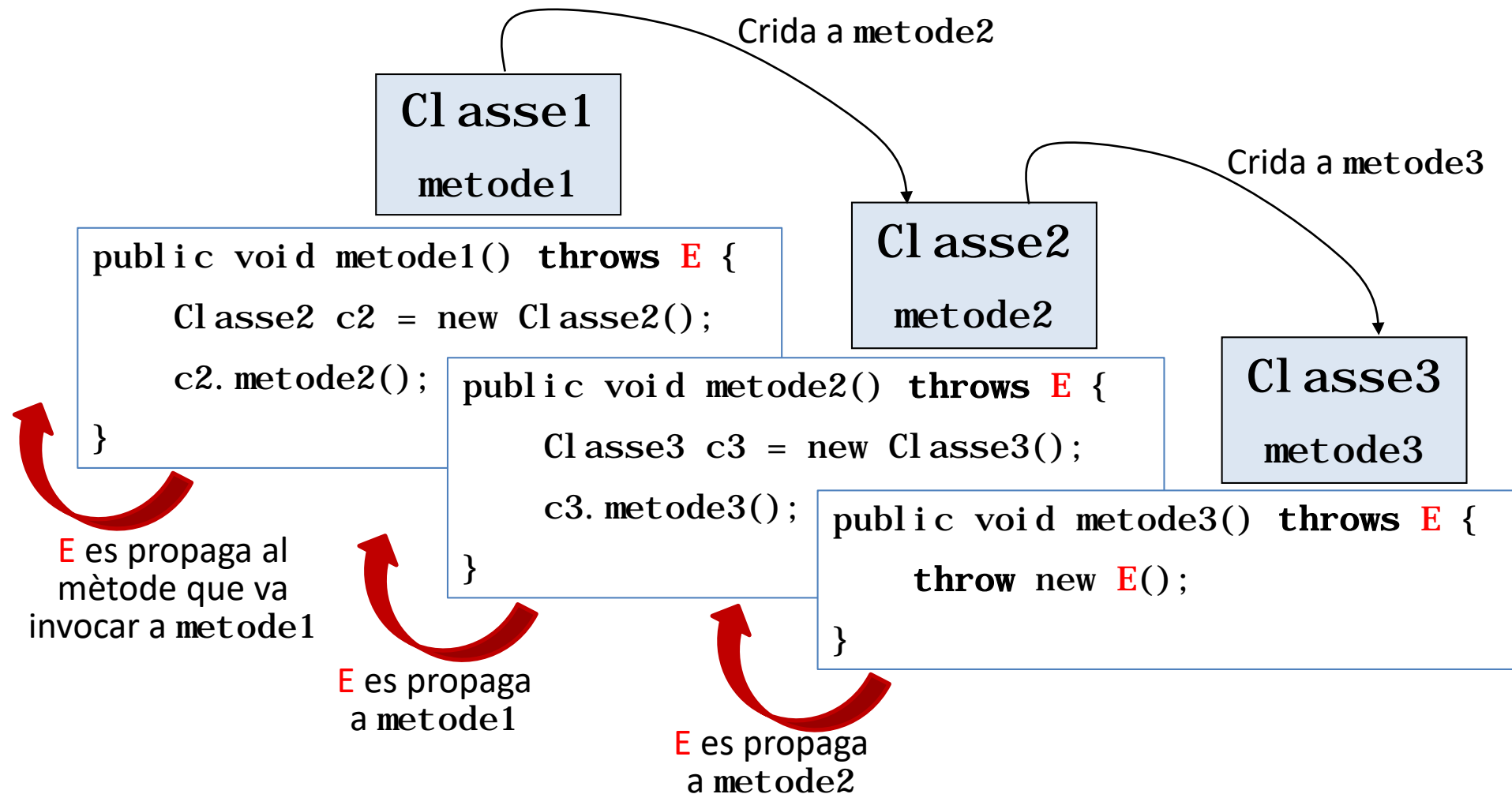
- Suposem que volem que el mètode `escriuComp` notifique l'índex erroni al programa per tal que aquest torne a demanar la dada a l'usuari.

```
public static void escriuComp(int i, int[] a)
    throws ArrayIndexOutOfBoundsException {
    System.out.println(a[i]);
}

public static void main(String[] args) {
    int talla = ...; int[] elArray = new int[talla];
    ...
    Scanner t = new Scanner(System.in);
    int n = 0; boolean lecturaCorrecta = false;
    do {
        try {
            System.out.print("Índex (entre 0 i " + (talla - 1) + "? ");
            n = t.nextInt();
            escriuComp(n, elArray);
            lecturaCorrecta = true;
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("error, " + n + " no és vàlid, prova de nou.");
        }
    } while (!lecturaCorrecta);
    ...
}
```

Propagació d'excepcions en Java

- Quin serà l'ordre de propagació de l'excepció **E** llançada en `metode3`?



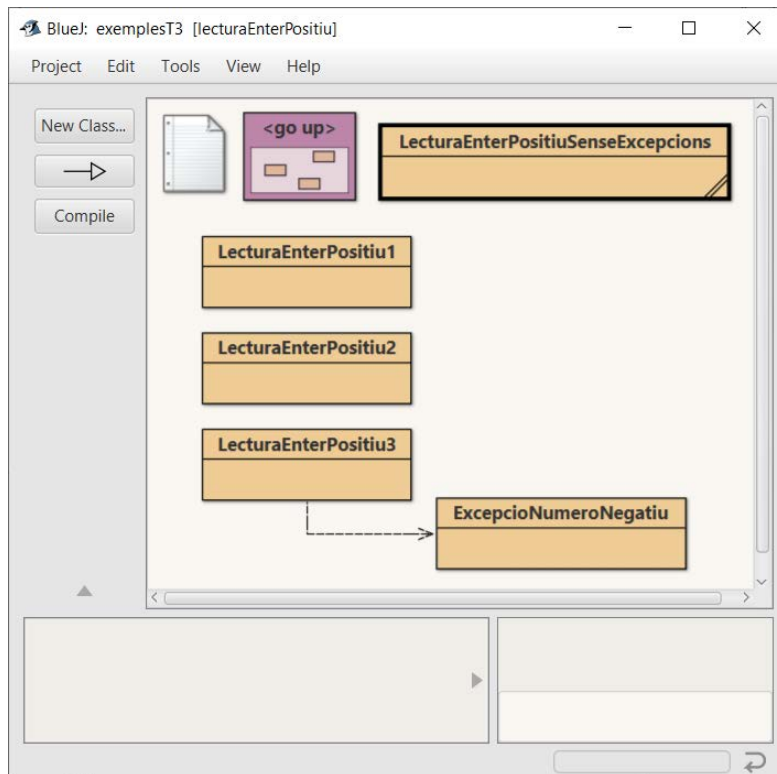
Tractament d'excepcions en Java

- Amb la clàusula `throws` es pot propagar l'excepció per tota la pila de crides, situant el tractament on es vullga.
- Pot haver més d'una excepció (separades per comes) en una mateixa clàusula `throws`.
- No confondre la clàusula `throws` per propagar l'excepció amb `throw` per llançar una nova excepció. (Recordar que hi ha que tenir-la creada, ja que és un objecte).
- Les excepcions *checked* sempre han de ser propagades o capturades.
- De la mateixa manera que amb la resta d'estructures de control, les instruccions `try-catch-finally` poden niuar-se o situar-se dintre de (o contenir) bucles, conditionals, etc.

Exemple: paquet LecturaEnterPositiu

- Al paquet LecturaEnterPositiu hi ha diferents classes que permeten fer la lectura d'un enter positiu des de teclat sense fer cap gestió de les excepcions que poden ocórrer o fent aquesta gestió de diverses maneres.

[https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Scanner.html#nextInt\(\)](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/util/Scanner.html#nextInt())



nextInt

```
public int nextInt()
```

Scans the next token of the input as an int.

An invocation of this method of the form `nextInt()` behaves in exactly the same way as the invocation `nextInt(radix)`, where `radix` is the default radix of this scanner.

Returns:

the int scanned from the input

Throws:

InputMismatchException - the next token does not match the *Integer* regular expression, or is out of range

NoSuchElementException - if input is exhausted

IllegalStateException - if this scanner is closed

Package java.util

Class InputMismatchException

java.lang.Object

java.lang.Throwable

java.lang.Exception

java.lang.RuntimeException

java.util.NoSuchElementException

java.util.InputMismatchException

Exemple: classe ExampleTryCatchFor del paquet al tres

- En el `main` de la classe `ExampleTryCatchFor` del paquet `al tres` és un exemple que mostra la diferència en l'ús d'un bloc `try-catch` dins i fora d'una instrucció `for`. Les dades s'han de passar com arguments al mètode `main`.

[https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Double.html#parseDouble\(java.lang.String\)](https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/Double.html#parseDouble(java.lang.String))

```
public static void main(String[] args) {
    System.out.println("Primera versio: for que inclou try-catch");
    System.out.println("-----");
    for (int i = 0; i < args.length; i++) {
        try {
            double valor = Double.parseDouble(args[i]);
            System.out.println(args[i] + " es un numero");
        } catch (NumberFormatException e) {
            System.out.println(args[i] + " no es un numero");
        }
    }

    System.out.println("\nSegona versio: try-catch que inclou for");
    System.out.println("-----");
    int i = 0;
    try {
        for ( ; i < args.length; i++) {
            double valor = Double.parseDouble(args[i]);
            System.out.println(args[i] + " es un numero");
        }
    } catch (NumberFormatException e) {
        System.out.println(args[i] + " no es un numero");
    }
}
```

parseDouble

`public static double parseDouble(String s) throws NumberFormatException`

Returns a new double initialized to the value represented by the specified String, as performed by the `valueOf` method of class `Double`.

Parameters:

`s` - the string to be parsed.

Returns:

the double value represented by the string argument.

Throws:

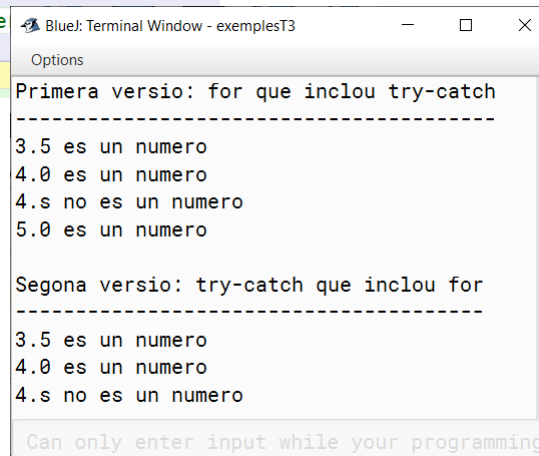
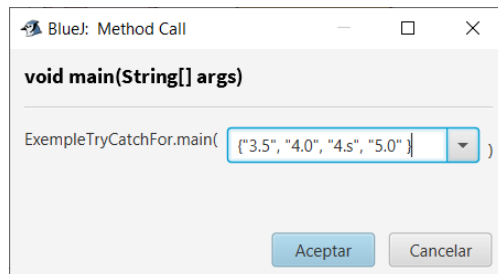
`NullPointerException` - if the string is null

`NumberFormatException` - if the string does not contain a parsable double.

Package `java.lang`

Class `NumberFormatException`

`java.lang.Object`
`java.lang.Throwable`
`java.lang.Exception`
`java.lang.RuntimeException`
`java.lang.IllegalArgumentException`
`java.lang.NumberFormatException`



Exercici: BonoMetro amb Excepcions



Blue: exercicisT3 [bonoMetro]

- La classe **BonoMetro** permet representar el títol de transport amb el que se pot viatjar al metro.
 - Cada bonometro té un nombre de tiquets disponibles i tots els bonometros comparteixen una quantitat per defecte de recàrrega que és 10.
 - Un bonometro es pot carregar amb un nombre donat de viatges o per defecte; es pot consultar el nombre de viatges disponible i es pot recarregar.
 - El mètode **pi car** comprova si queden viatges, en aquest cas actualitza el nombre de tiquets i torna el missatge "Bono amb xxx tiquets". Si no queden viatges torna el missatge "Bono esgotat. Recàrrega' l JA! "
- Modifica el mètode **pi car** per tal que quan no queden viatges llance l'excepció **Sal doEsgotatExcepti on**.
 - Revisa la implementació de l'excepció d'usuari **Sal doEsgotatExcepti on**. Fixa't que és una subclasse d'**Excepti on** i, per tant, ha de tractar-se com una excepció *checked*.
 - Completa la classe **TestBonoMetro** per tal que verifiqui la funcionalitat prèviament implementada.

Exercici: Carnet per Punts

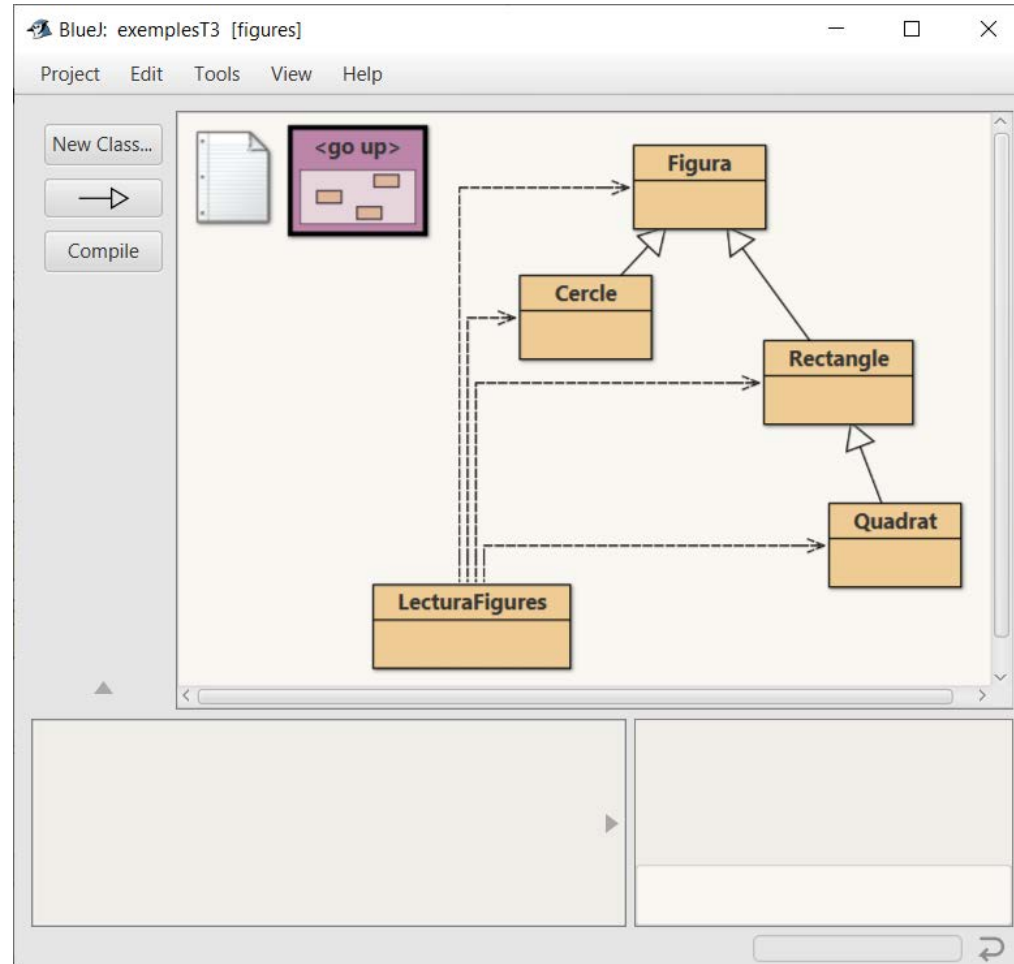


BlueJ: exercicisT3 [carnetPerPunts]

- La classe **CarnetDeConduir** permet representar un carnet de conduir amb un crèdit inicial de 12 punts que es va perdent a mesura que es cometeixen infraccions. Un saldo de zero punts o negatiu implica una retirada immediata del carnet de conduir.
 - La classe **DGT** permet aplicar la penalització.
1. Revisa l'excepció *comprovada* d'usuari **RetiradaImmediataCarnet**.
 2. Modifica el disseny actual del mètode **llevarPunts** de la classe **CarnetDeConduir** per tal que, quan el saldo de punts d'un carnet de conduir siga negatiu o zero després de la penalització, llance l'excepció **RetiradaImmediataCarnet**.
 3. Modifica el mètode **multar** de la classe **DGT** per a que mostre un missatge d'error per pantalla si la penalització de punts comporta la retirada immediata del carnet.

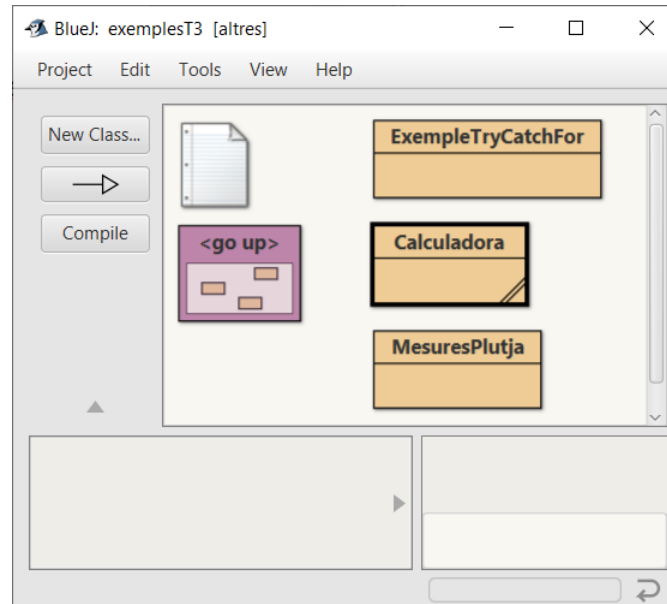
Example: classe LecturaFigures del paquet figures

- La classe LecturaFigures del paquet figures és un altre exemple de validació de dades llegides des de teclat i gestió de les excepcions que es poden produir.



Exemple: classe Calculadora del paquet al tres

- La classe Calculadora del paquet al tres representa una calculadora d'enters on les operacions permeses són la suma, la diferència, el producte i la divisió.
- Els errors en la introducció des de teclat dels operands num1 i num2 i l'operador op es gestionen capturant l'excepció InputMismatchException.
- Si op no és un operador vàlid, es llança InputMismatchException.
- Es captura també l'excepció ArithmeticException que és produïx quan num2 és 0 en la divisió num1 / num2.
- Fixa't en el try- catch- finally inclòs en el finally que gestiona la possible InputMismatchException en la resposta de l'usuari per acabar o no l'execució.



Exercici: Dia d'una data



BlueJ: exercicisT3 [utilsData]

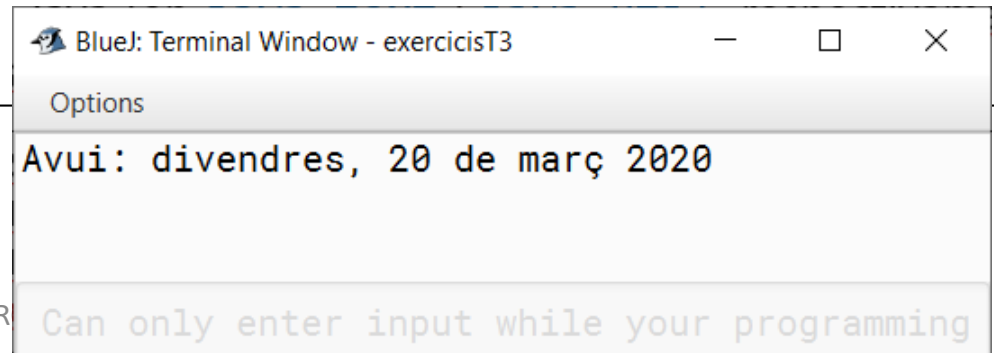
- La classe **Di aSetmanaAvui** és un exemple d'ús de les classes **SimpleDateFormat** i **Date** de l'API de Java (en **java.text** i **java.util**, respectivament) per saber a quin dia de la setmana correspon la data actual.

```
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;
public class Di aSetmanaAvui {
    /** No hi ha objectes d'aquesta classe */
    private Di aSetmanaAvui () { }

    public static void main(String[] args) {
        SimpleDateFormat fmt = new SimpleDateFormat("EEEE, dd MMM yyyy",
                                                    new Locale("ca", "ES"));


        Date avui = new Date();
        String avuiAmbDia = fmt.format(avui);
        System.out.println("Avui: " + avuiAmbDia);
    }
}
```

- Si s'executa el 20/3/2020...



Exercici: Dia d'una data



 BlueJ: exercicisT3 [utilsData]

- Per conèixer el dia de la setmana de qualsevol data donada en el format dd/mm/yyyy (p.e., 08/03/2019), es pot utilitzar el mètode `parse` de la classe `SimpleDateFormat`, que pot llançar l'excepció comprovada `ParseException`:

```
String data = "08/02/2019";
SimpleDateFormat fmt = new SimpleDateFormat("dd/MM/yyyy",
                                           new Locale("ca", "ES"));

Date dataD = fmt.parse(data);
fmt.applyPattern("EEEE, dd MMM yyyy");
String dataDAmbDia = fmt.format(dataD);
System.out.println(dataDAmbDia);
```

parse

```
public Date parse(String source)
    throws ParseException
```

Parses text from the beginning of the given string to produce a date. The method may not use the entire text of the given string.

See the `parse(String, ParsePosition)` method for more information on date parsing.

Parameters:

`source` - A String whose beginning should be parsed.

Returns:

A Date parsed from the string.

Throws:

`ParseException` - if the beginning of the specified string cannot be parsed.

Class ParseException

```
java.lang.Object
    java.lang.Throwable
        java.lang.Exception
            java.text.ParseException
```

All Implemented Interfaces:

`Serializable`

```
public class ParseException
    extends Exception
```

Signals that an error has been reached unexpectedly while parsing.

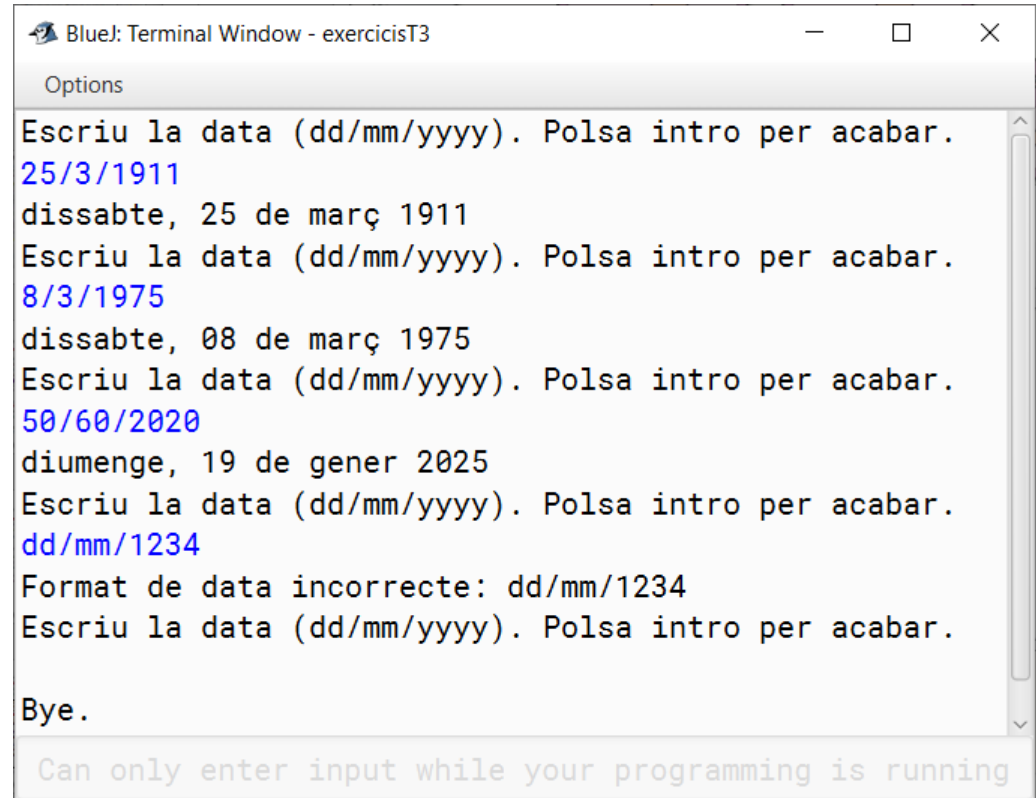
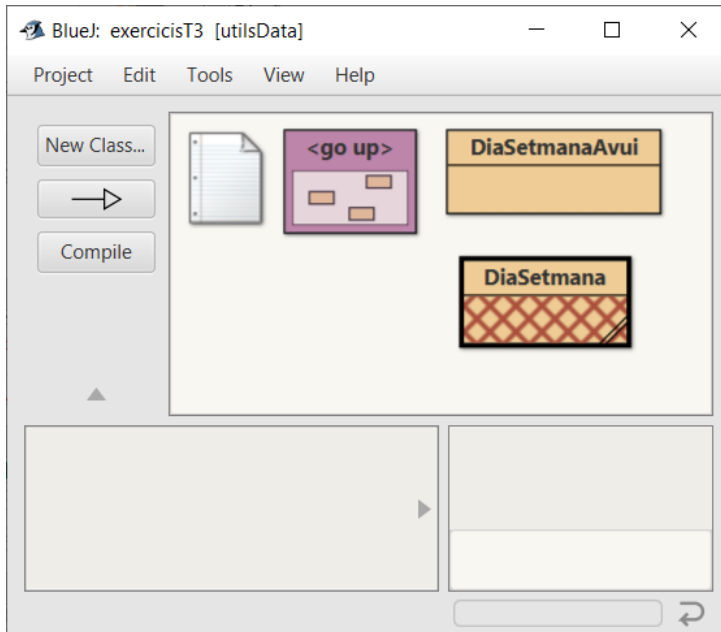
Super

Exercici: Dia d'una data



BlueJ: exercicisT3 [utilsData]

- Completa el codi de la classe **Di aSetmana** per tal de:
 1. Corregir l'error de compilació.
 2. Permetre que l'usuari pugui introduir per teclat totes les dates que desitge.
 3. Si alguna de les dates que escriu no s'ajusta al format establert, ha d'informar de l'error.
 4. El programa ha d'acabar quan l'usuari introduísca la **String** buida.



UNIVERSITAT
POLITÉCNICA
DE VALÈNCIA

Exercici: Transferència de fitxers



BlueJ: exercicisT3 [transferirFitxers]

- La classe **CopyViaFTP** permet realitzar la transferència d'un arxiu a un altra màquina mitjançant FTP. Quan, pel motiu que siga, aquesta transferència no es pot fer, llança l'excepció comprovada d'usuari **UnableToTransferException**.

```
public class CopyViaFTP {  
    public static void copyTo(String hostName, String localFilePath)  
        throws UnableToTransferException {  
        ...  
    }  
}
```

- Completa el `main` de la classe **TestCopyViaFTP** per tal que faça la transferència del fitxer `/tmp/data` a la màquina `fileserver.upv.es`. En cas d'error, l'operació s'haurà de reintentar un màxim de 3 vegades i indicar a l'usuari el número d'intent.
- Al final del codi del mètode `copyTo`, per fer proves, s'ha afegit la instrucció:
`throw new UnableToTransferException("no es pot fer la transferència");`
simulant que no es pot fer la transferència.

Exercici

poli **[formaT]** PRG >  **EXÀMENS** 1B - T3. Qüestionari: excepcions