EXEPCIONS (I)

Error Sintàctic -> El codi del programa no es correcte, el compilador mostra error i no compila.

Error Semàntic -> El codi és vàlid, el compilador no mostra error, però la instrucció és incorrecta en temps d'execució (ex: divisió per 0, posició incorrecta en un array, etc..)

Els errors semàntics, o errors que es produeixen durant l'execució d'un programa en JAVA s'anomenen **Excepcions**, per tant :

Una **excepció** és un **ERROR SEMÀNTIC** que es produeix en temps d'execució.

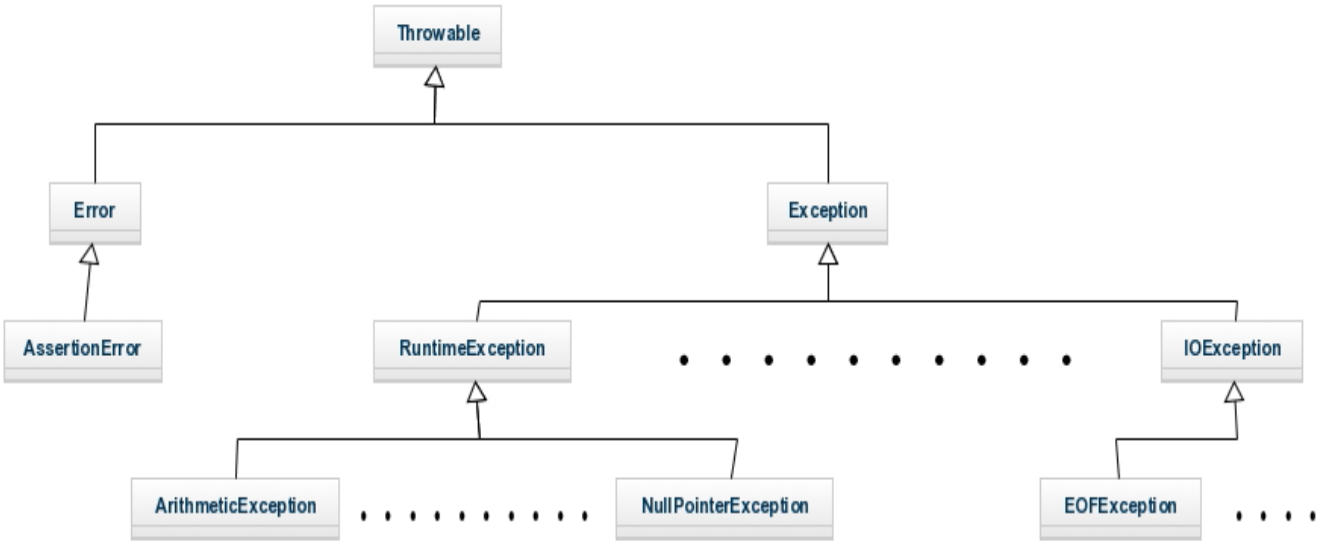
Entre els més comuns trobem:

- Dividir per 0.
- Accedir a una posició d'un array fora dels seus límits.
- Errors en la classe SCANNER (demanar un enter (nextInt()) i escriure una cadena de text)
- Un objecte és NULL i no ho pot ser
- **Accedir a un fitxer que no existeix o esta en un disc dur corrupte.**
- Etc.

Quan el compilador troba una excepció o error d'execució mostra un missatge d'error i finalitza l'execució del programa, i llança una excepció ("Throwing Exception").

Quan es produeix una excepció en Java es crea un OBJECTE de la classe **Exception** (les excepcions en Java són objectes) que ens proporcionarà informació sobre l'error i tots els mètodes necessaris per a obtindre eixa informació.

Estes classes tenen com a pare la classe ***Throwable***, per tant es manté una jerarquia en les excepcions (hi ha moltíssimes excepcions)



EXEMPLES D'EXEPCIONS

1.- Forcem la divisió per zero.

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

        int resultat;
        int numerador=10;
        int denominador=0;

        System.out.println("\nABANS DE LA DIVISIÓ\n");

        resultat=numerador/denominador;

        System.out.println("\nDESPRÉS DE LA DIVISIÓ\n");
        System.out.println("\n El resultat de la divisió és: " + resultat)
    }
}
```

Eixida per pantalla:

```
ABANS DE LA DIVISIÓ

Exception in thread "main" java.lang.ArithmeticException: / by zero
        at exemples.DivZero.main(DivZero.java:12)
```

La màquina virtual de JAVA **detecta un error en temps d'execució** i crea un **objecte** de la classe ***java.lang.ArithmeticException***, i com el mètode on es produeix l'excepció no pot tractar-la acaba el programa en la fila 12.

2- Forcem error en conversió de tipus

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

    String cadena1="123";
    String cadena2="Hola";
    int num1, num2;

    num1=Integer.parseInt(cadena1);
    System.out.println("\nConversió de cadena1 a numero Enter: " + cadena1);

    num2=Integer.parseInt(cadena2);
    System.out.println("\nConversió de cadena1 a numero Enter: " + cadena2);

    }
}
```

Eixida per pantalla:

```
Conversió de cadena1 a numero Enter: 123
Exception in thread "main" java.lang.NumberFormatException: For input string: "Hola"
        at java.lang.NumberFormatException.forInputString(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at java.lang.Integer.parseInt(Unknown Source)
        at exemples.ConversioTipus.main(ConversioTipus.java:13)
```

Com **cadena2 no té un format adequat** ("Hola" no representa un número vàlid), el mètode Integer.parseInt (...) no pot convertir-la a un valor de tipus int i llança l'excepció ***NumberFormatException.*** La màquina virtual Java finalitza el programa a la línia 13 i mostra per pantalla la informació sobre l'excepció que s'ha produït.

3.- Forcem els límits d'un vector.

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

        int vector[]={1,2,3,4};
        System.out.println("\nABANS DEL FOR");

        for(int i=0;i<=5;i++){

            System.out.println("v["+i+"]= "+vector[i]);
        }

        System.out.println("\nDESPRES DEL FOR");
    }
}
```

Eixida per pantalla:

```
ABANS DEL FOR
v[0]= 1
v[1]= 2
v[2]= 3
v[3]= 4
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 4
        at exemples.PosicioArray.main(PosicioArray.java:11)
```

Com intentem accedir a la posició 5 del vector i sols té 4 es produeix una excepció de tipus **ArrayIndexOutOfBoundsException**. La màquina virtual de java finalitza el programa a la línia 11 i mostra el missatge d'error sobre l'excepció que s'ha produït.

Per sort, Java ens permet realitzar un control de les excepcions per a que el nostre programa no es pare de manera inesperada, i continuï la seua execució per això contem amb l'estructura **"try-catch-finally"**.

GESTIONAR EXEPCIONS (TRY-CATCH-FINALLY)

En Java es poden gestionar excepcions utilitzant tres mecanismes anomenats **gestors d'excepcions**.
Existeixen tres i funcionen conjuntament:

• Bloc **try (intentar):** codi que podria llançar una excepció.
• Bloc **catch (capturar):** codi que manejarà l'excepció si és llançada.
• Bloc **finally (finalment**): codi que s'executa tant si hi ha excepció com si no.
Un **gestor d'excepcions** és un bloc de codi encarregat de tractar les excepcions per intentar recuperar-se de la decisió i evitar que l'excepció sigui llançada descontroladament fins al **main**() i acabe amb l'execució del programa.

Sintàxis:

```
try {
        // Instruccions que podrien llançar una exepció
}
catch (TipusExcepció nomVariable) {

        // Instruccions que s'executen quant 'try' llança una excepció.
}
finally {

        // Instrucciones que s'executen tant si hi ha exepció com no (SEMPRE)
}
```

- **El bloc try intentarà executar el codi**. Si es produeix una excepció s'abandona aquest bloc (no s'executaran les altres instruccions de l'try) i es saltarà a el bloc catch. Lògicament, si en el try no es produeix cap excepció el bloc catch s'ignora.
- **El bloc catch capturarà les excepcions de l'tipus TipusExepció**, evitant que siga llançada al mètode que ens va cridar. Ací hem d'escriure les instruccions que siguen necessàries per gestionar l'error. Poden especificar diversos blocs catch per a diferents tipus d'excepcions.
- **El bloc finally és opcional** i s'executarà tant si s'ha llançat una excepció com si no.

Exemple Divisió per Zero:

```
Public class DivZero {

        public static void main(String args[]) {
                int resultat;
                int numerador=10;
                int denominador=0;

                try {

                        resultat = numerador / denominador;
                        System.out.println("L'execució no aplegarà fins ací");
                }
                catch (ArithmeticException exepcio1) {

                        System.out.println("Has intentat dividir per 0, DATIL!!!");
                }

                finally {

                        System.out.println("Finally és opcional");
                }
                System.out.println("Fi del programa");
        }
}
```

```
Has intentat dividir per 0, DATIL!!! java.lang.ArithmeticException: / by zero
Fi del programa
```

Al intentar dividir per zero es llança automàticament una **ArithmeticException**.

Com que això pasa dins de l'bloc **try**, l'execució de el programa passa al primer bloc **catch** perquè coincideix amb el tipus d'excepció produïda **(ArithmeticException)**.

S'executarà el codi del bloc catch i després el programa continuarà amb normalitat.

Clàusula CATCH

L'objectiu d'una clàusula catch és resoldre la condició excepcional perquè el programa puga continuar com si l'error mai hagués passat.

- **Clàusules catch múltiples.** Es poden especificar diverses clàusules catch, tantes com vulguem, perquè cadascuna capture un tipus diferent d'excepció.

```
try {
        //     instruccions que poden produir diferents tipus d'Excepcions
}
catch (TipoExcepción1 e1) {
        // instruccions per manejar un TipoExcepción1
}
catch (TipoExcepción2 e2) {
        // instruccions per manejar un TipoExcepción2
}
...
}
catch (TipoExcepciónN eN) {
        // instruccions per manejar un TipoExcepciónN
}
finally { // opcional
        // instruccions que s'executaran tant si hi ha excepció com si no
}
```

Quan es llança una excepció dins del **try**, es comprova cada sentència **catch** en ordre i s'executa la primera que coincideix el tipus

Els altres blocs **catch** seran ignorats.

Després s'executarà el bloc **finally** (si s'ha definit) i el programa continuarà la seva execució després del bloc **try-catch-finally.**

Si el tipus excepció produïda no coincideix amb cap dels **catch**, llavors l'excepció serà llançada al mètode que ens va cridar. Y si no pot resoldre l'excepció, donarà error.

Exemple:

```
import java.util.Scanner;

public class DivZero {

    public static void main(String args[]) {

        Scanner in = new Scanner(System.in);
        int resultat;
        int numerador=10;
        int denominador=0;
        int posicio;
        int[] vector = {1,2,3};

        try {

            System.out.println("Introdueix el numerador");
            numerador=in.nextInt();

            System.out.println("Introdueix el denominador");
            denominador=in.nextInt();

            resultat = numerador / denominador;

            System.out.println("La divisió és " + resultat);

            System.out.println("Introdueix la posició del vector a consultar");
            posicio = in.nextInt();

            System.out.println("l'element és " + vector[posicio]);
        }
        catch (ArithmeticException exepcio) {

            System.out.println("Has intentat dividir per 0, DATIL!!!" + exepcio);
        }
        catch (ArrayIndexOutOfBoundsException exepcio) {

            System.out.println("T'has passat de tamany del vector, XAA!!! COMPTA BÉ!!!!" + exepcio);
        }

            System.out.println("Fí del programa");
    }
```

Poden passar tres coses :

• El **try** s'executa sense excepcions, s'ignoren els **catch** i s'imprimeix "Fi del programa".

• Es produeix **l'excepció de divisió per zero** l'execució salta al primer **catch**, i es mostra el missatge "Has intentat dividir per 0" i després "Fi del programa".

• Es produeix l'excepció de sobrepassar el vector salta al segon **catch**, i mostra el missatge "T'has passat de tamany del vector" i "Fi del programa".

EIXIDES:

```
Introdueix el numerador
1
Introdueix el denominador
0
Has intentat dividir per 0, DATIL!!! java.lang.ArithmeticException: / by zero
Fí del programa
```

```
Introdueix el numerador
4
Introdueix el denominador
2
La divisió és 2
Introdueix la posició del vector a consultar
5
El tamany del vector és 3!! XAA!!! COMPTA BÉ!!!! java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3
Fí del programa
```

- **El bloc catch només capturarà excepcions del tipus indicat**. Si es produeix una excepció diferent no la capturarà. I retornarà l'excepció al mètode que la produeix. **Si no sap resoldre-la donarà error**.

```
public class DivZero {

    public static void main(String args[]) {

        Scanner in = new Scanner(System.in);
        int resultat;
        int numerador=10;
        int denominador=0;
        int posicio;
        int[] vector = {1,2,3};

        try {

            System.out.println("Introdueix el numerador");
            numerador=in.nextInt();

            System.out.println("Introdueix el denominador");
            denominador=in.nextInt();

            resultat = numerador / denominador;

            System.out.println("La divisió és " + resultat);

        }

        catch (ArrayIndexOutOfBoundsException exepcio) {

            System.out.println("El tamany del vector és 3!! XAA!!! COMPTA BÉ!!!! " + exepcio);
        }

            System.out.println("Fí del programa");
    }
}
```

Es produeix una excepció de tipus ***ArithmeticException DivZero***

Sols hem gestionat una excepció de tipus ***ArrayIndexOfBoundsException***.

La clàusula catch no pot gestionar l'excepció i es produeix un error que finalitza el programa

```
Introdueix el numerador
4
Introdueix el denominador
0
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Exemples.DivZero.main(DivZero.java:23)
```

- **El bloc catch també capturarà excepcions heretades del tipus indicat**. Per exemple, **catch** (**ArithmeticException e**) capturarà qualsevol tipus d'excepció que herete de **"ArithmeticException**".

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

    Scanner in = new Scanner(System.in);
    int resultat;
    int numerador=10;
    int denominador=0;
    int posicio;
    int[] vector = {1,2,3};

    try {

        System.out.println("Introdueix el numerador");
        numerador=in.nextInt();

        System.out.println("Introdueix el denominador");
        denominador=in.nextInt();

        resultat = numerador / denominador;

        System.out.println("La divisió és " + resultat);

        System.out.println("Introdueix la posició del vector a consultar");
        posicio = in.nextInt();

        System.out.println("l'element és " + vector[posicio]);
    }

    catch (Exception exepcio) {

        System.out.println("El tamany del vector és 3!! XAA!!! COMPTA BÉ!!!! " + exepcio);
    }

        System.out.println("Fí del programa");
}
```

El cas més general és:
catch (Exception nomExepcio**)**

Capturarà **TOTES** les excepcions ja que en Java totes hereten de la classe **Exception**.

No és una bona idea, hem d'utilitzar excepcions **més pròximes al tipus d'error** previst, perquè **NO PODREM SABER** quin tipus d'error ens ha donat, i no podrem gestionar com cal l'excepció.

Això si, el programa no es pararà, i continuarà amb possibles errors en les

EIXIDES:

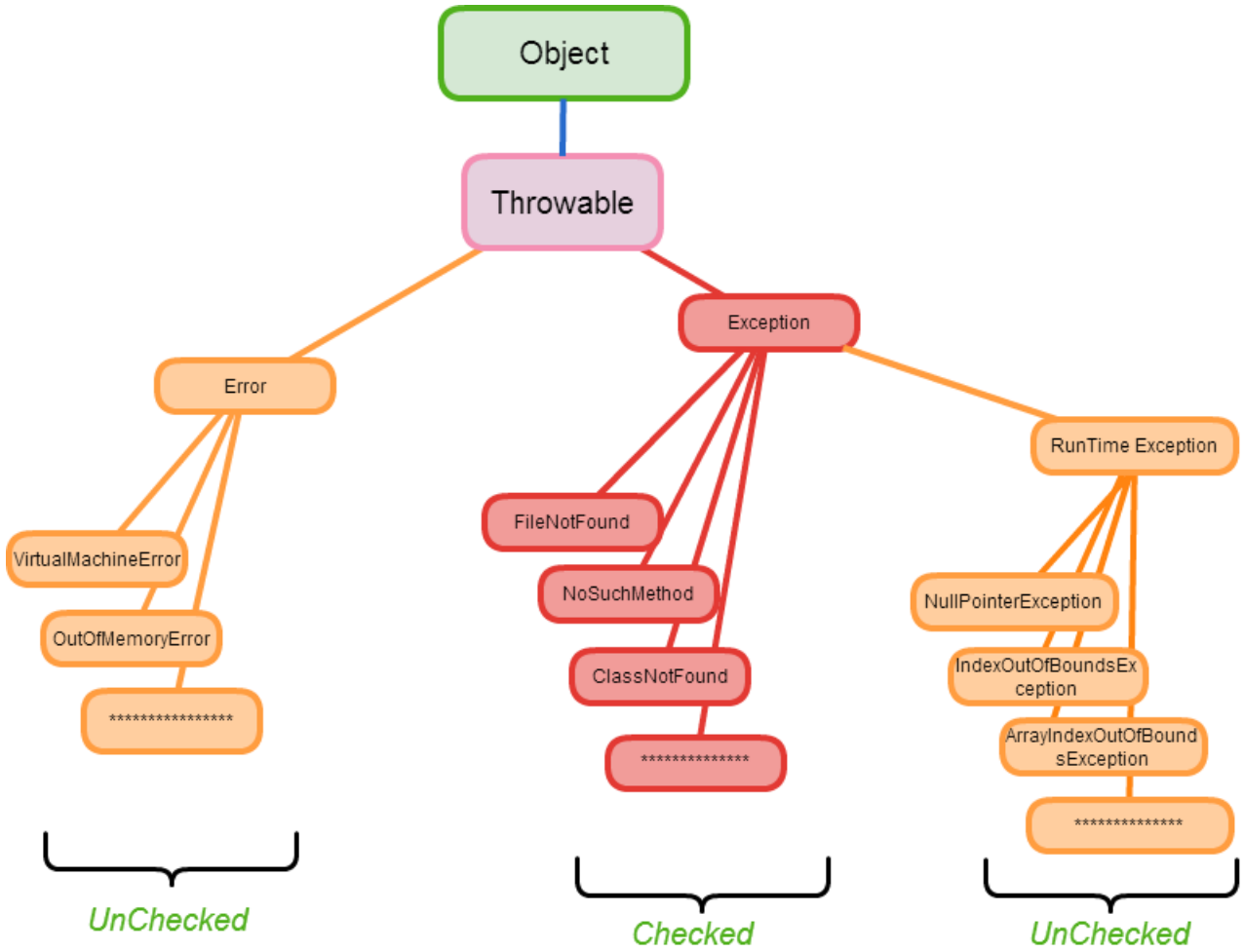
```
Introdueix el numerador
1
Introdueix el denominador
0
El tamany del vector és 3!! XAA!!! COMPTA BÉ!!!! java.lang.ArithmeticException: / by zero
Fí del programa
```

En les dos eixides, tant si fem la divisió per 0 o posem un valor fora dels límits del vector, capturem la mateixa excepció. Mostra el mateix error, per tant no estem gestionant bé les excepcions,

```
Introdueix el numerador
4
Introdueix el denominador
2
La divisió és 2
Introdueix la posició del vector a consultar
5
El tamany del vector és 3!! XAA!!! COMPTA BÉ!!!! java.lang.ArrayIndexOutOfBoundsException: Index 5 out of bounds for length 3
Fí del programa
```

Jerarquia d'Excepcions en JAVA

La classe **Exception** hereta de Throwable, i totes les excepcions hereten de **Exception**.



Les Excepcions poden ser **comprovades i no comprovades**:

• **Excepcions comprovades**: aquelles que Java **comprova** durant la **compilació**, abans de l'execució del programa.

• **Excepcions no comprovades**: aquelles que Java no pot comprovar durant la compilació i **es produiran durant l'execució** de el programa.

Les **excepcions comprovades** definides en java.lang són:

Excepción	Significado
ClassNotFoundException	No se ha encontrado la clase.
CloneNotSupportedException	Intento de duplicado de un objeto que no implementa la interfaz clonable.
IllegalAccessException	Se ha denegado el acceso a una clase.
InstantiationException	Intento de crear un objeto de una clase abstracta o interfaz.
InterruptedException	Hilo interrumpido por otro hilo.
NoSuchFieldException	El campo solicitado no existe.
NoSuchMethodException	El método solicitado no existe.

Les **subclasses** de RuntimeException **no comprovades** són:

Excepción	Significado
AritmeticException	Error aritmético como división entre cero.
ArrayIndexOutOfBoundsException	Índice de la matriz fuera de su límite.
ArrayStoreException	Asignación a una matriz de tipo incompatible.
ClassCastException	Conversión invalida.
IllegalArgumentException	Uso inválido de un argumento al llamar a un método.
IllegalMonitorStateException	Operación de monitor inválida, como esperar un hilo no bloqueado.
IllegalStateException	El entorno o aplicación están en un estado incorrecto.
IllegalThreadStateException	La operación solicitada es incompatible con ell estado actual del hilo.
IndexOutOfBoundException	Algún tipo de índice está fuera de su rango o de su límite.
NegativeArraySizeException	La matriz tiene un tamaño negativo.
NullPointerException	Uso incorrecto de una referencia NULL.
NumberFormatException	Conversión incorrecta de una cadena a un formato numérico.
SecurityException	Intento de violación de seguridad.
StringIndexOutBounds	Intento de sobrepasar el límite de una cadena.
TypeNotPresentException	Tipo no encontrado.
UnsupportedOperationException	Operación no admitida.