



ENSAH



جامعة عبد الملك السعدي  
Université Abdelmalek Essadi

# Gestion des exceptions

## Langage Java

**2<sup>ème</sup> année Génie Informatique**

**Ecole Nationale des Sciences Appliquées – Al Hoceima**

Prof A. Bahri  
abahri@uae.ac.ma

A.U 2020/2021

# Exception

- ❑ Une exception est littéralement un «événement exceptionnel».
- ❑ Un événement exceptionnel est une situation «anormale» du programme, une erreur, à un certain moment (***une division par zéro provoque une exception de type `arithmeticException`***).
- ❑ Java utilise les exceptions pour :
  - Indiquer les problèmes d'exécution
  - Indiquer un événement particulier qui mérite un traitement spécial

# Exemple

```
class DivParZero{  
    public static void main (String argv[] ) {  
        int val=0;val= 1997/val;  
        System.out.println("Fin du programme");  
    }  
}
```


Exécution de ce programme , on obtient l'affichage suivant : (message par défaut)  
**java.lang.ArithmeticException: / by zero**  
**at DivParZero.main(DivParZero.java:6)**

On distingue:

- le nom complet de l'exception qui a été *levée*,
- un message précisant la cause de cette erreur est envoyé par l'objet de type exception (/by zero),
- l'indication de la classe, de la méthode et du numéro de ligne où s'est produite cette exception.

# Obligation de capture ou de transfert

- ❑ Il existe trois catégories d'exceptions:
  - Les erreurs (problèmes de la JVM): correspondent à des exceptions qu'il est rare d'attraper.
  - Les exceptions non vérifiées (Runtime): que l'on peut rattraper mais que l'on **n'est pas obligé**.
  - Les exceptions vérifiées (la majorité): que **l'on est obligé** d'attraper (**try/catch**) ou de dire que la méthode appelante devra s'en occuper (**throws**).

- 
- ❑ Java oblige au traitement des exceptions vérifiées. Soit :
    - On doit trouver un gestionnaire
    - On doit transférer l'exception au-dessus

# Comment on gère les exceptions

❑ Une exception est un objet (instance) envoyé par la machine virtuelle lorsqu'une certaine condition d'erreur apparaît.

❑ On met le code qui appelle la fonction où l'erreur peut se produire «sous contrôle», dans une structure **try {}**  
**catch {} finally {}**

# Traitements des exceptions :

- ❑ **try** : contient le code qui peut produire une ou plusieurs exceptions
- ❑ **catch** : contient le code destiné à gérer les exceptions
- ❑ **finally** (facultative) : contient le code qui est toujours exécuté avant qu'une méthode ne se termine (que l'exception soit levée ou non)

# Bloc try

- ❑ Contient le code qui peut être à l'origine de l'exception
- ❑ Syntaxe :

```
try{
```

```
//code pouvant lever une ou plusieurs exceptions
```

```
}
```

# Bloc catch

- ❑ Contient le code chargé de gérer une exception d'un type donné
- ❑ Doit suivre immédiatement le bloc **try**
- ❑ Syntaxe :  
**catch(TypeException objet){**  
**//code chargé de gérer**  
**l'exception**  
**}**
- ❑ Un même bloc **try** peut avoir plusieurs blocs **catch**
- ❑ Permet de capturer une exception que l'on veut traiter
- ❑ La pire des choses à faire :
  - **catch** (Exception e) { }



# Exemple

```
Class TestException{  
    public static void main(String[] arg){  
        int i=10,j=...;  
        String x= "rrrr";  
        try{  
            System.out.println("i="+ i+" et j="+j);  
            i=Integer.parseInt(x);  
            System.out.println(i/j);  
            System.out.println("Fin du bloc try ");  
        }  
        catch(ArithmeticException e){  
            System.out.println("Erreur : division par 0 ");  
        }  
        System.out.println("Après le bloc try");  
    }  
}
```

# Bloc catch multiple

- ❑ Si un bloc **try** peut lever plusieurs types d'exception, on peut insérer plusieurs blocs **catch** pour les gérer

- ❑ Syntaxe :

```
try{  
    //code pouvant lever une ou plusieurs exceptions  
}  
catch(TypeError objet){  
    //code chargé de gérer l'exception  
}  
catch(TypeError objet){  
    //code chargé de gérer l'exception  
}  
....
```

# Exemple

```
try{  
    A=b/c;  
}  
catch(ArithmeticException e){  
    //code chargé de gérer l'exception  
}  
catch(Exception objet){  
    //code chargé de gérer l'exception  
}
```

# Exception

## Remarques :

- ❑ Si on doit capturer des exceptions de différents types pour un bloc try, l'ordre des blocs catch est très important.
- ❑ Lorsqu'une exception est levée, elle sera capturée par le 1<sup>er</sup> bloc catch dont le type de paramètre est identique à celui de l'exception, où dont le type est une superclasse du type de l'exception
- ❑ Le type Exception permet la capture de tout type d'exception ➡ Un seul catch est exécuté en cas d'erreur dans un bloc try

# Exemple

```
try{  
    //code pouvant lever une ou plusieurs exceptions  
}  
catch(Exception e){  
    //code chargé de gérer l'exception  
}  
catch(ArithmeticException e){  
    //code chargé de gérer l'exception  
}
```

Les exceptions du type '[ArithmeticException](#)' n'arriveront jamais à atteindre le second bloc catch, car elles seront toujours captées par le 1<sup>er</sup>.

➔ Erreur

[exception java.lang.ArithmeticException has already been caught](#)

# Résumé

- ❑ Si un bloc `try{}` peut gérer plusieurs types d'exception, l'ordre est important
- ❑ Il faut commencer avec la sous **classe la plus basse** et en progressant vers la superclasse.

# Exemples d'exceptions prédéfinies

Nom de la classe	Condition de génération
ArithmeticException	Une condition arithmétique non valide (division par 0, par exemple)
IndexOutOfBoundsException	Un index d'un tableau qui se trouve en dehors des limites
NegativeArraySizeException	Un tableau avec une dimension négative
NullPointerException	Utilisation d'une variable objet contenant la valeur null

# Exemples d'exceptions prédéfinies

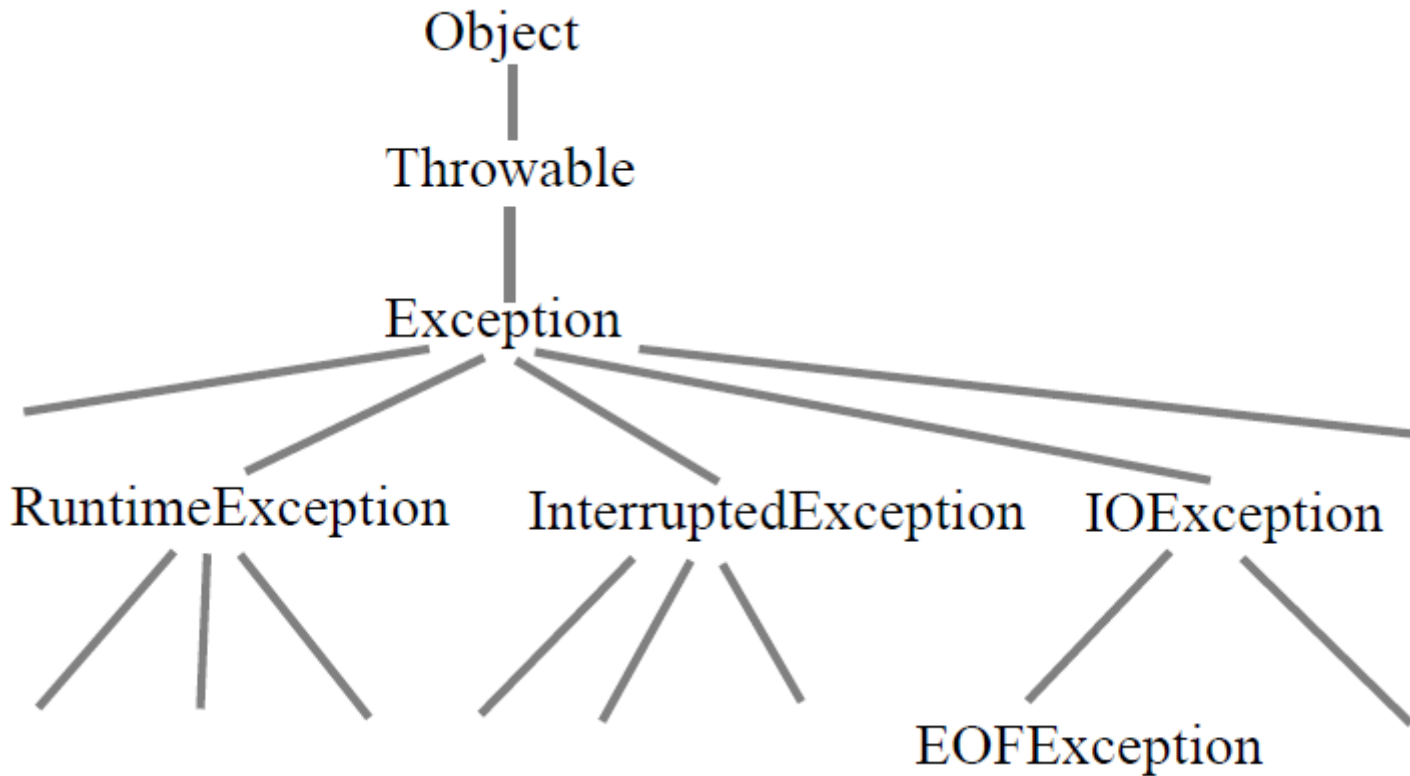
Nom de la classe	Condition de génération
NumberFormatException	Une condition de conversion au format nombre non valide
ClassNotFoundException	Référence à une classe inexistante
SQLException	Erreur dans une requête SQL
IOException	Exception d'entrée/sortie



# Bloc finally

- ❑ Quelque soit la façon dont on est sorti du "bloc **try**", les instructions contenues dans le "bloc **finally**" sont exécutées
- ❑ Un "bloc **finally**" est en général utilisé pour effectuer des nettoyages (fermer des fichiers, libérer des ressources...).

# Hiérarchie des classes d'exceptions



# Exceptions définies par l'utilisateur

- ❑ Si on veut créer notre propre exception, il faut étendre la classe **java.lang.Exception** ou d'une sous classe
- ❑ Ne pas hériter de **RuntimeException** ou **d'Error**
- ❑ Convention : Inclure **Exception** dans le nom
- ❑ L'utilisation se fait classiquement par un **try/catch/finally**
- ❑ La classe étendue ne contient en général pas d'autre champ qu'un (ou plusieurs) **constructeur(s)** et éventuellement une redéfinition de la méthode **toString**.
- ❑ Lors du lancement de l'exception, (à l'aide du mot réservé **throw**), on crée une instance de la classe définie (instanciation nécessaire).

# Exemple (création d'une exception)

```
class MyException extends Exception {  
    public MyException() {}  
    public MyException(String s) {  
        super(s);  
    }  
    public String toString() {  
        return("Aucune note n'est valide");  
    }  
}
```

# Exceptions définies par l'utilisateur

- ❑ On indique qu'une méthode `m` peut générer une exception `MyException` par le mot clé `throws` (obligatoire pour les exceptions sous contrôle)

```
void m () throws MyException
{
    ...
}
```

- ❑ On peut déclencher une exception grâce au mot clé `throw`
- ❑ Exemple:

```
if (x<0)
{
    throw new IllegalArgumentException ("x doit etre positif");
    // ne mettre aucune instruction en dessous du throw
}
```

# Exemple 1 (Exceptions définies par l'utilisateur)

```
public class Pile{
    private int table [] ;
    private int hauteur = 0 ;
    public Pile () { table = new int [3] ; }
    public Pile (int h) { table = new int [h] ; }
    public void insertValue (int valeur) throws
    PileException{
        if (hauteur == table.length) throw new
        PileException ("Pile pleine") ;
        else table [hauteur++] = valeur ;
    }
    public int removeValue () throws
    PileException{
        if (hauteur == 0) throw new
        PileException ("Pile vide") ;
        else return table [--hauteur] ;
    }
}
```

```
public class PileException extends Exception {
    public PileException(String m)
    {
        super (m) ;
    }
}
```

Utilisation:

```
Pile pile = new Pile () ;
try {
    System.out.println (pile.removeValue()) ;
} catch (PileException e){
    System.out.println (e.getMessage()) ;
}
```

# Exemple2 (Exceptions définies par l'utilisateur)

```
class TestException {  
    public static void main(String[] liste) throws MonException{  
        int somme=0,entier, nbNotes=0;  
        int i;  
        for (i=0;i < liste.length;i++) {  
            try {  
                entier=Integer.parseInt(liste[i]);  
                somme+=entier; nbNotes++;  
            } catch (NumberFormatException e) {  
                System.out.println("La " +(i+1)+ " eme note n'est pas entiere");  
            }  
        }  
        if (nbNotes==0) throw new MonException();  
        System.out.println(somme/nbNotes);  
    }  
}
```

```
class MonException extends Exception {  
    public MonException() {}  
    public MonException(String s) {  
        super(s);  
    }  
    public String toString() {  
        return("Aucune note n'est valide");  
    }  
}
```

# Le bon usage des exceptions

- ❑ Les exceptions doivent rester ... exceptionnelles
- ❑ Ce n'est pas une nouvelle structure de contrôle en plus des if, while, for, switch
- ❑ Pour éclaircir les traitements d'erreur du genre:

```
if (erreur) <trait-erreur>
    else {...
        if (erreur) <trait-erreur>
            else {...
                if (erreur) <trait-erreur>
                    else {...
                        (suite)
                    }
                }
            }
        }
    }
```

- ❑ Eviter de capter les erreurs graves non récupérables (NullPointerException, ...)



# Exercice 1

1. Écrire un programme Java qui permet de calculer la moyenne de plusieurs notes passées en arguments. Distinguer le cas où le format d'une note n'est pas valide à l'aide des exceptions
2. Refaire la question précédente en introduisant plusieurs valeurs à partir du clavier