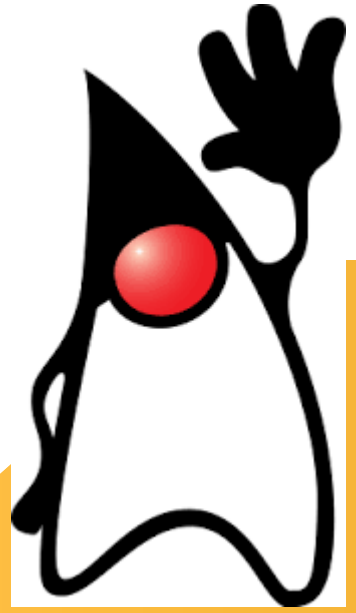
RÉPUBLIQUE TUNISIENNE
MESRS
Ministère de l'Enseignement Supérieur
et de la Recherche Scientifique

SFAX



Cours Programmation Orientée Objets Avancée

Préparé par :
Hedi HMANI, Manel BEN SALAH, Sameh HBAIEB TURKI

Année universitaire
2020-2021

Objectifs du cours

A la fin de ce cours, vous serez capable de:

- gérer les exceptions dans un programme java;
- maîtriser les collections JAVA;
- comprendre la notion de la programmation générique avec JAVA ;
- créer des interfaces graphiques personnalisées avec java;
- manipuler des BD relationnelles à l'aide de l'API JDBC; et
- créer des applications multithread pour éviter les inter-blocages.

Les pré-requis

Pour pouvoir suivre ce cours, vous devriez savoir :

- Créer des classes JAVA.
- Créer des instances d'objet à l'aide du mot clé new.
- Déclarer des variables primitives et des variables de référence JAVA.
- Déclarer des méthodes JAVA à l'aide de valeurs renvoyées et de paramètres.
- Déclarer et instancier des tableaux JAVA.
- Connaître les concepts fondamentaux relatifs aux bases de données et au langage SQL.

Chapitre 1
La gestion des exceptions

trycatch

Problématique

Exemple 1. On veut écrire une méthode qui calcule la division de a/b : comment pouvons-nous gérer l'erreur de division par 0.

Solution avec échec

```
public class testException {
public static void main (String []args)
{
int a = 15;
int b = 0;
System.out.println (a/b);
} }
```

L'exécution du programme génère le message d'erreur suivant :

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero
at testException.main(testException.java:9)
```

Solution avec succès

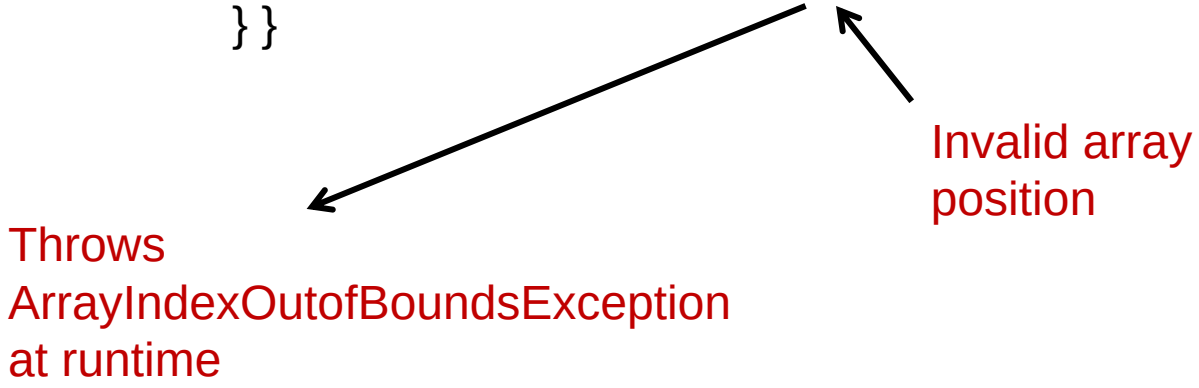
```
public class testException {
public static void main (String []args)
{
int a = 15;
int b = 0;
if (b==0)
   System.out.println ('Erreur: division par zéro');
else
    System.out.println (a/b);
} }
```

Le programme est exécuté avec succès et le message suivant sera affiché: Erreur: division par zéro

Problématique

Exemple 2. On veut accéder à une case du tableau qui n'existe pas (indice qui dépasse la longueur du tableau), que se passe-t-il ?

```
public class ArrayAccess {
   public static void main(String args[]) {
        String[] students = {"Shreya", "Joseph", null};
        System.out.println(students[5].length());
        } }
```



Problématique

Exemple 3. Trace d'exécution

Soit l'exemple suivant:

```
public class Trace {                                          //Line1
    public static void main(String args[]) {                  //Line2
            method1();                                        //Line3
      }                                                       //Line4
    public static void method1()    {                         //Line5
            method2();                                        //Line6
      }                                                       //Line7
    public static void method2()   {                          //Line8
        String[] students = {"Shreya", "Joseph"};             //Line9
        System.out.println(students[5]);                      //Line10
      }                                                       //Line11
}                                                             //Line12
```

Source de l'échec d'exécution du programme

Résultat

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBounds
Exception:
at Trace.method2(Trace.java:10)
at Trace.method1(Trace.java:6)
at Trace.main(Trace.java:3)
```

Comment gérer les exceptions qui sont levées dans un code JAVA ?
Comment créer des applications fiables ?

La fiabilité d’un logiciel en Java

La robustesse :

C’est la capacité du logiciel à fonctionner, même en présence **d’événements exceptionnels** tels que la saisie d’informations erronées par l’utilisateur

La correction :

C’est le fait que le logiciel fournit des résultats corrects en fonctionnant normalement

La fiabilité d'un logiciel en Java

2 mécanismes principaux assurent une bonne fiabilité en java :

les exceptions pour la **robustesse**

les assertions pour la **correction**

On peut utiliser également d'autres outils tels que: l'API de **journalisation** (**logging**); les outils de tests d'unités (comme JUnit) ou de tests fonctionnels et les débuggeurs

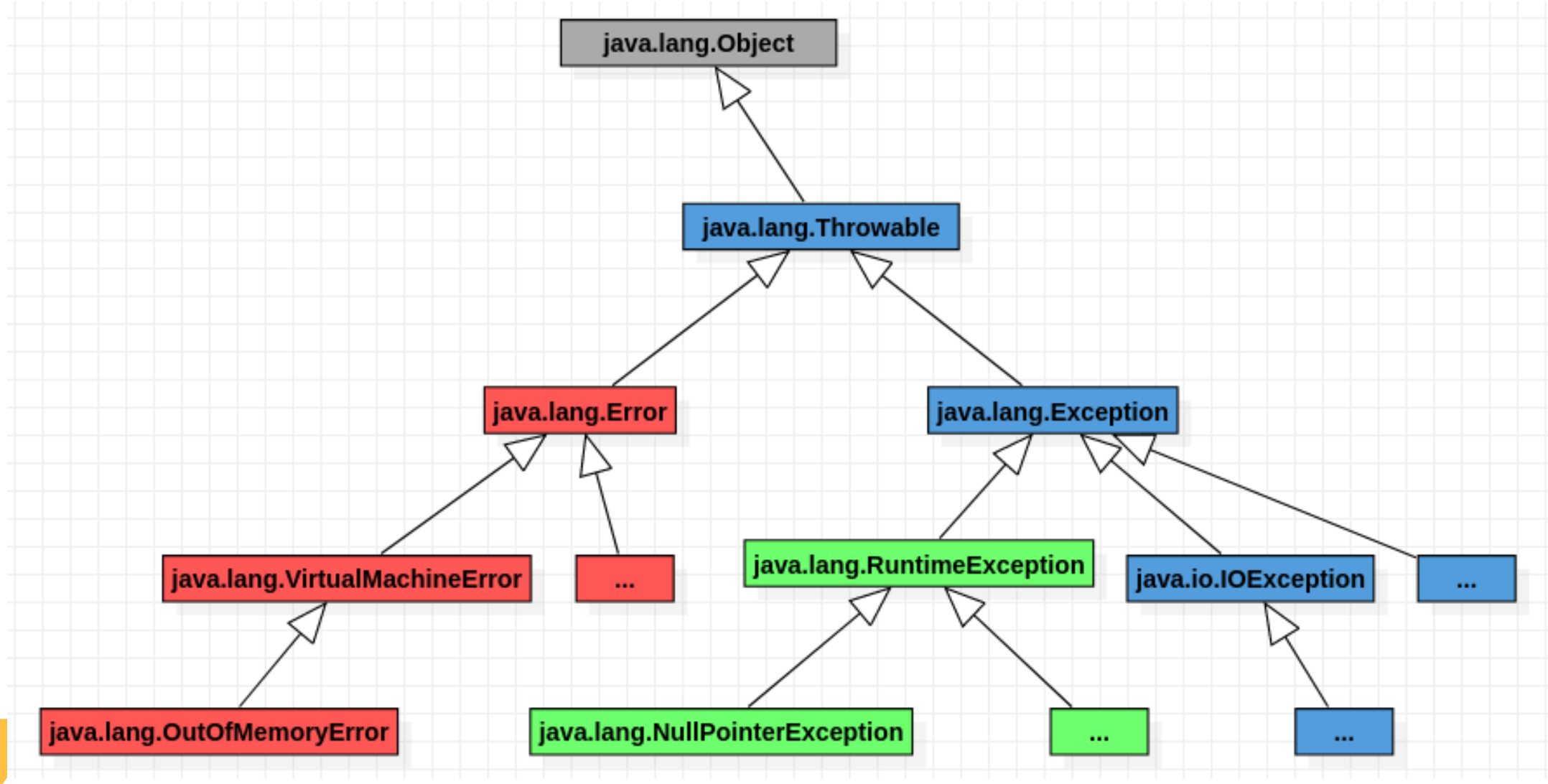
Qu’est ce qu’une exception en java ?

Définition

- Une exception est un événement, se produisant lors de l'exécution d'un programme, qui **interrompt** l'enchainement normal des instructions.

- Les exceptions/erreurs sont utilisées pour traiter un fonctionnement anormal d'une partie d'un code (provoqué par une erreur ou un cas exceptionnel) comme par exemple:
  - Mauvaise gestion des classes: accès hors tableau,…
  - Entrée utilisateurs non valides: effacer un fichier inexistants, Division par 0, Référence null, Mauvais URL ..
  - Liées aux périphériques: manque de papier dans l'imprimante,…
  - Limitation physique: disque plein, Plus de mémoire libre …
  - ….

Les classes d’exception



Hiérarchie des classes d’exception

Les classes d'exception

Deux classes dérivent de Throwable: Error et Exception

1. Une *Error* est une exception qui indique des problèmes graves : une application ne résout pas ce genre de problème. Une méthode n'a pas à déclarer dans sa clause *throws* les *Error* qui ne seraient pas capturées.

2. Une *Exception* en revanche peut être capturée et traitée par l'application, ou apparaître dans la liste des exceptions levées par la méthode.

L’objet Exception

Exemple:

```
public class Principale {
    private static Scanner scanner = new Scanner(System.in);
    public static void main(String[] args) {
        int n = 0;
        System.out.println("Saisir un entier");
        n = scanner.nextInt();
        System.out.println("Valeur saisie="+n);
    }
}
```

```
Saisir un entier
e
Exception in thread "main" java.util.InputMismatchException
        at java.util.Scanner.throwFor(Unknown Source)
        at java.util.Scanner.next(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at java.util.Scanner.nextInt(Unknown Source)
        at Principale.main(Principale.java:8)
```

```
public class Principale {
    private static Scanner scanner = new Scanne
    public static void main(String[] args) {
        int n = 0;
        System.out.println("Saisir un entier");
        try {
            n = scanner.nextInt();
            System.out.println("Valeur saisie="+n);
        }
        catch (Exception e) {
            System.out.println("ERREUR");
        }
    }
}
```

```
Saisir un entier
e
ERREUR
```

L’objet Exception

- Lorsqu’une erreur se produit dans une méthode, celle-ci crée un objet Exception qui contient des informations sur l’erreur survenue:

- Le type de l’erreur: objet null, dépassement de tableau, etc.
- La trace d’exécution du programme: la stack trace

- L’action de créer un objet Exception et de le passer au système d’exécution est appelée **lever/ lancer** une exception(throwing an exception)

L’objet Exception

Quelques méthodes que l'on peut utiliser avec les objets exception :

`getMessage()`	:	retourne un `String` contenant le message (texte) associé à l'exception
`toString()`	:	retourne un `String` contenant le nom de l'exception suivi du message associé
`printStackTrace()`	:	affiche sur la console de sortie le nom de l'exception, le message associé ainsi que l'état de la pile des appels qui ont conduit au traitement de l'exception *(Stack-Trace)*
`getStackTrace()`	:	retourne les informations de la *Stack-Trace* sous forme d'un tableau de `StackTraceElement`

Classification des exceptions

En Java, il existe 2 catégories d'exceptions :

1. **Les exceptions prédéfinies:** Offerts dans l'API Java : sous classes de Exception ou de RuntimeException

2. **Les exceptions personnalisées (construites par l'utilisateur):** Classes définies par le développeur pour gérer des exceptions spécifiques. Il doit les étendre de Exception ou de RuntimeException

Exemples d’exceptions prédéfinies

- **ArithmeticException** Division entière par zéro. Note : la division réelle par zéro ne lève pas d’exception, mais son résultat est INFINITY.
- **NullPointerException** : utilisation de length ou accès à un case d’un tableau valant null (c’est à dire non encore créé par un new).
- **ArrayIndexOutOfBoundsException** : accès à une case inexistante dans un tableau.
- **StringIndexOutOfBoundsException** : accès au i eme caractère d’un chaîne de caractères de taille inférieure à i.
- **NegativeArraySizeException** : création d’un tableau de taille négative.
- **NumberFormatException** : erreur lors de la conversion d’un chaîne de caractères en nombre.

Exemples d’exceptions personnalisées

- ***TempsException*** *: Pour une horloge, l’heure doit être entre 0 et 23, les minutes et les secondes entre 0 et 59*
- ***NoteException*** *: La note doit être entre 0 et 20*
- *…*

Interception vs propagation

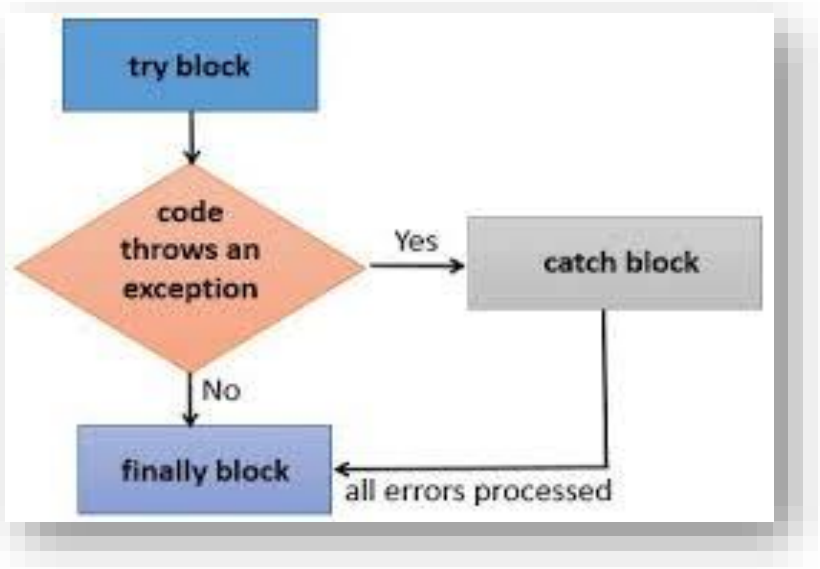
Si une méthode génère une exception ou appelle une autre méthode qui génère une exception, il existe 2 façons de gérer cette exception :

- **intercepter** (lever, capturer, attraper) et traiter l'exception avec **try-catch**
- **propager** (lancer, déclencher, diffuser) l'exception avec **throw** et **throws**

Interception avec le bloc try-catch

- Le bloc try-catch sert à capturer une exception dans une méthode
- Chaque bloc d'instructions comportant l'appel d'une méthode susceptible de lancer un objet d'exception doit être surveillé (bloc try)
- Chaque bloc surveillé est suivi par des blocs qui captent et traitent les objets d'exception à raison d'un bloc de traitement par type d'exception attendue (bloc catch)
- Un bloc finally est optionnel. Il permet au programmeur de définir un ensemble d'instructions qui est toujours exécuté, que l'exception soit levée ou non, capturée ou non.

Interception avec le bloc try-catch



01

try
marquer les endroits réceptifs aux erreurs (« attraper » l'objet lancé)

02

catch
leur associer un gestionnaire d'exception

03

finally
Gérer le reste du code

Interception avec le bloc try-catch

```
try
{
// suite d'instructions
}
catch (Exception1 e1)
{
// traitement 1 de l'exception e1
}
catch (Exception2 e2)
{
// traitement 2 de l'exception e2
}
finally
{
// traitement 3
}
```

L'ordre des clauses catch est important !

Les blocs catch doivent être présentés selon la hiérarchie des classes d'exception en question: du type dérivé vers le type général (sinon il y aura une erreur de compilation)

Interception avec le bloc try-catch

Exemple:

```
public class Principale {
    private static Scanner scanner new Scanner(System.in);
    public static void main(String[] args) {
        System.out.println("Saisir un entier");
        try {
            int b = scanner.nextInt();
            int c = 100/b;
            System.out.println("Partie entière de 100/b="+c);
        }
        catch (NumberFormatException e1) {
            System.out.println("Il faut un nombre entier !");
        }
        catch (ArithmeticException e2) {
            System.out.println ("Parti vers l'infini !");
        }
        finally {
            System.out.println("Passage obligé !");
        }
    }
}
```

Lancement avec throws et throw

- Si une exception peut survenir, mais que la méthode n'est pas censée la traiter elle-même, il faut lancer (propager) cette exception
- Ajouter le mot-clé **throws** à la signature de la méthode

```
public void writeList() throws IOException {
  PrintWriter out = new PrintWriter(new FileWriter("OutFile.txt"));
  for (int i = 0; i < vector.size(); i++)
       out.println("Valeur = " + vector.elementAt(i));
}
```

Lancement avec throws et throw

- Une méthode avec une propriété throws peut lancer des exceptions avec throw
- Le développeur peut créer de nouvelles classes d'exceptions et les lancer avec throw

```
class MyException extends Exception {
  MyException(String msg){
    System.out.println("MyException lancee, msg =" + msg);
  }
}

void someMethod(boolean flag)throws MyException {
       if(!flag) throw new MyException(“someMethod”);
       …
}
```

Lancement avec throws et throw

- Une même méthode peut tout à fait "laisser remonter" plusieurs types d'exceptions (séparés par des ,).

```
public void writeList() throws IOException, ArrayIndexOutOfBoundsException {
  PrintWriter out = new PrintWriter(new FileWriter("OutFile.txt"));
  for (int i = 0; i < vector.size(); i++)
      out.println("Valeur = " + vector.elementAt(i));
}
```

- Une méthode doit traiter ou "laisser remonter" toutes les exceptions qui peuvent être générées dans les méthodes qu'elle appelle.

Exemple

Etape 1: Définition de la classe d’exception

```
Class TempsException extends Exception
{ public String message;
  public TempsException ( String m)
   { message = m;}
}
```

Suite de l’exemple

Etape 2: Propagation de l’exception

```
Public Class Temps
{ private int heures, minutes, secondes;
  public Temps (int h, int m, int s) throwsTempsException
  { if (h>=0 && h<=23)
      heures = h;
    else
      throw new TempsException(« heure erronée »);
```

Suite de l'exemple

Etape 2: Propagation de l'exception

```
Public Class Temps
{ private int heures, minutes, secondes;
  public Temps (int h, int m, int s) throwsTempsException
  { if (h>=0 && h<=23)
      heures = h;
    else
      throw new TempsException(« heure erronée »);
    if (m>=0 &&m<=59)
        minutes= m;
      else
        throw new TempsException(« minute erronée »);
    if (s>=0 && s<=59)
        secondes= s;
      else
        throw new TempsException(« seconde erronée »);
    } }
```

Suite de l’exemple

Etape 3: Capture et traitement d’une exception

```
private static Scanner scanner=new Scanner(System.in);
public static void main (String args [])
  { int h, m, s;
     try
       { h= scanner.nextInt();
         m = scanner.nextInt();
         s = scanner.nextInt();
         Temps t = new Temps (h, m, s);
} catch (TempsException e)
   {System.out.println (« Temps invalide: » + e.message);}}
```

Intérêts des exceptions

- Augmenter la lisibilité du code en séparant les instructions qui traitent le cas normal des instructions nécessaires au traitement des erreurs ou des événements exceptionnels.
- Permettre (voire imposer) la déclaration explicite des exceptions qu'une méthode peut lever (fait partie de la signature).

- Forcer le programmeur à prendre en compte (traiter ou propager) les cas exceptionnels (erreurs ou autres événements) qui sont déclarés dans les méthodes qu'il invoque.

Des Questions ?