FACULTE DES SCIENCES AIN CHOCK
UNIVERSITE HASSAN II DE CASABLANCA

Département: Mathématiques & Informatique

Gestion des exceptions

Licence Professionnelle : Développement Informatique

Pr: Youssef Ouassit

Gestion des exceptions

1. Le mécanisme des exceptions en Java (try, catch, finally).
2. Les classes d'exceptions (Exception, RuntimeException).
3. Lever des exceptions (throw) et créer des exceptions personnalisées.
4. Différence entre les exceptions vérifiées et non vérifiées.

Gestion des exceptions

Plan du chapitre :

- Introduction
- Hiérarchie des exceptions
- Traitement des exceptions
  - Interception d'exceptions: bloc *try – catch – finally*
  - Lancement (génération) par une méthode: *throws* et *throw*

Introduction

La gestion des exceptions en Java

Définition :

Une exception est un événement qui perturbe le flux normal d'un programme.

Une exception peut être causée par une erreur de programmation (par exemple, division par zéro) ou une erreur externe (par exemple, un fichier non trouvé).

Gestion des exceptions

Introduction

La gestion des exceptions en Java

Pourquoi gérer les Exceptions ?

Sécurité du programme : Éviter l'arrêt brutal du programme en cas d'erreur.

Améliorer l'expérience utilisateur : Fournir des messages d'erreur utiles.

Maintenance du code : Aider à repérer les erreurs et anomalies de manière efficace.

Gestion des exceptions

Introduction

La gestion des exceptions en Java

Types d'exceptions :

- **Exceptions vérifiées (Checked Exceptions)** : Doivent être explicitement gérées (par exemple, IOException, SQLException).

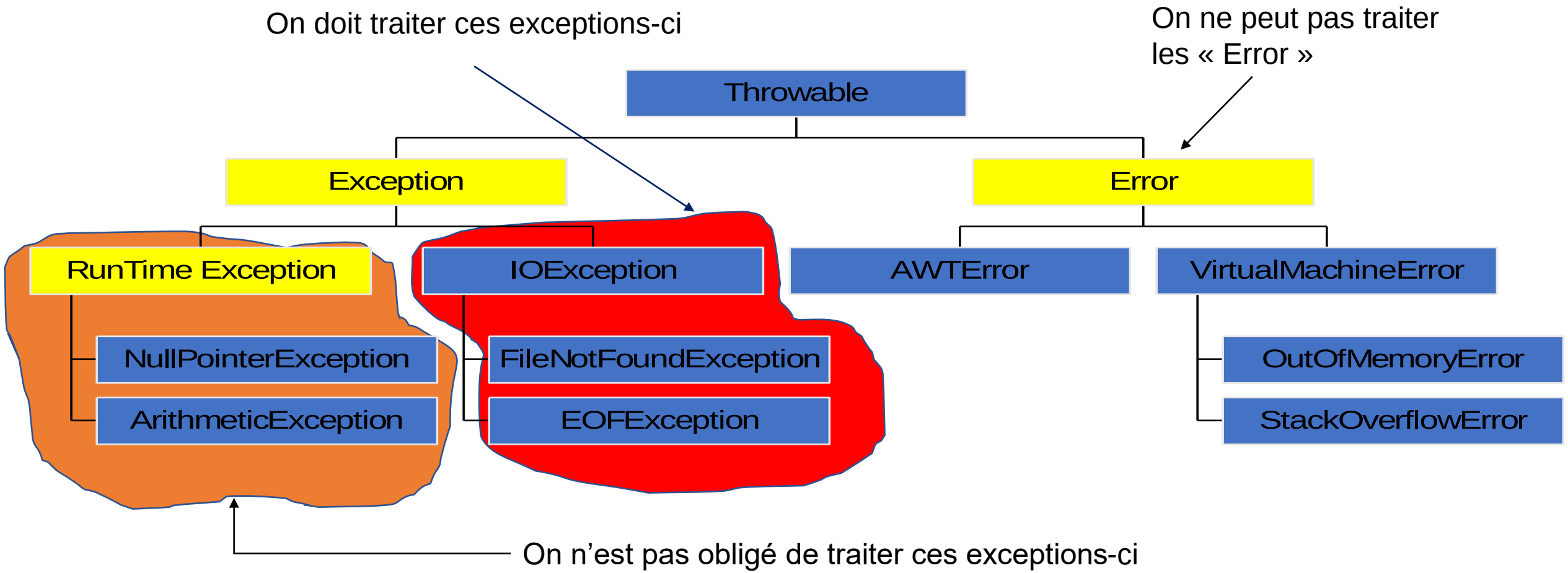
- **Exceptions non vérifiées (Unchecked Exceptions)** : Héritent de RuntimeException, elles ne sont pas obligatoires à gérer (par exemple, NullPointerException, ArithmeticException).

- **Errors :** On ne peut pas les traiter.

Gestion des exceptions

Introduction

Hiérarchie des exceptions



Traitement des exceptions

Principe

- Le traitement des exceptions contient deux aspects:
  - L'interception des exceptions
    - Utilisation du bloc *try – catch – finally* pour récupérer les exceptions
    - Et réaliser les actions nécessaires
  - Le lancement (la génération) d'exceptions
    - Automatiquement par l'environnement run-time ou la machine virtuelle pour certaines exceptions prédéfinies par Java
    - Explicitement par le développeur dans une méthode avec « throws » et « throw » (en tout cas pour les exceptions créées par le développeur

Traitement des exceptions

Interception par bloc try – catch :

```
try
{
  // quelques actions potentiellement risquées
}
catch(SomeException se)
{
  // que faire si une exception de ce type survient
}
```

Traitement des exceptions

Interception de plusieurs types d'exceptions

```
try
{
  // quelques actions potentiellement risquées
}
catch(SomeException se)
{
  // que faire si une exception de ce type survient
}
catch(Exception e)
{
  // que faire si une exception d'un autre type survient
}
```

Traitement des exceptions

Le block finally (1/2)

```
try
{
  // quelques actions potentiellement risquées
}
catch(SomeException se)
{
  // que faire si une exception de ce type survient
}
catch(Exception e)
{
  // que faire si une exception d'un autre type survient
}
finally
{
  // toujours faire ceci, quelle que soit l'exception
}
```

Traitement des exceptions

Exemple :

Implémentation

```
public ExampleException() {
  for(int i=0;i<3;i++) {
    test[i]=Math.log(i);
  }
  try {
    for(int i=0;i<4;i++) {
      System.out.println("log("+i+") = "+test[i]);
    }
  } catch(ArrayIndexOutOfBoundsException ae) {
     System.out.println("Arrivé à la fin du tableau");
  }
  System.out.println("Continuer le constructeur");
}
```

Traitement des exceptions

Lancement avec les mots-clés throws et throw (1/4)

- Si une exception peut survenir, mais que la méthode n'est pas censée la traiter elle-même, il faut en « lancer » une instance
- Il faut préciser que la méthode peut lancer ces exceptions
  ➔ ajouter une clause throws à la signature de la méthode

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

- Peut lancer une IOException ➔ doit être attrapée
- Peut lancer une ArrayIndexOutOfBoundsException

```
public void writeList() throws IOException,
                    ArrayIndexOutOfBoundsException {
```

Traitement des exceptions

Lancement avec les mots-clés throws et throw (2/4)

- Une méthode avec une propriété *throws* peut lancer des exceptions avec *throw*
- Toute exception ou erreur lancée pendant l'exécution provient d'un throw
- Fonctionne avec les exceptions qui héritent de Throwable (la classe de base)
- 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»);
        …
}
```

Traitement des exceptions

Lancement avec les mots-clés throws et throw (3/4)

- Une fois l'exception lancée avec throw, il faut soit l'attraper (avec catch), soit la re-lancer. Si vous la re-lancez, votre méthode doit le déclarer

```
public void connectMe(String serverName)  throws ServerTimedOutException {
   boolean success;
   int portToConnect = 80;
   success = open(serverName, portToConnect);
   if (!success) {
     throw new ServerTimedOutException("Connection impossible", 80);
  }
}
```

```
public void connectMe(String serverName)  {
   boolean success;
   int portToConnect = 80;
   success = open(serverName, portToConnect);
   if (!success) {
     try{  throw new ServerTimedOutException("Connection impossible ", 80);
     } catch(ServerTimedOutException stoe){ }
  }
}
```

Traitement des exceptions

Lancement avec les mots-clés throws et throw (4/4)

- Si la méthode à redéfinir lance une exception
  - ➔On ne peut pas lancer une « autre » exception
  - ➔Mais on peut utiliser l'héritage