




# Gestion des erreurs



LP IEM

## Les exceptions : gestion des erreurs

-  Lorsqu'une erreur se produit à l'intérieur d'une méthode, la méthode crée un objet qu'elle passe au système. Cet objet est une exception. Elle contient des informations sur l'erreur et possède un type qui permet d'identifier la nature de l'erreur
  - ⇒ la méthode "lance une exception"
-  Le système remonte la pile des appels de fonctions à la recherche d'un bloc de code traitant cette exception
-  Si aucun bloc n'est trouvé le programme se termine et la machine virtuelle Java affiche un message sur la console

## ❑ Les exceptions : gestion des erreurs

❑ Il existe 3 types d'exceptions

- ❑ Les **exceptions vérifiées** (*checked exceptions*) : conditions exceptionnelles qu'une application bien conçue doit anticiper et dont elle doit récupérer (ex : saisie d'un nom de fichier inexistant par l'utilisateur,...).
- ❑ Les **exceptions non vérifiées** (sous classes de **RuntimeException**) : conditions exceptionnelles internes à l'application et que l'application ne peut généralement pas résoudre. Ce sont généralement des bugs, erreurs de logique, mauvaise utilisation d'une API.
- ❑ Les **erreurs** (sous classes d'**Error**) : conditions exceptionnelles externes à l'application et que l'application ne peut généralement pas résoudre (ex : erreur de lecture de fichier due à un secteur défectueux). L'application peut traiter ces exceptions mais ce n'est pas une obligation.

## Les exceptions : gestion des erreurs

### Les 3 types d'exceptions

#### Exemples d'exceptions non vérifiées

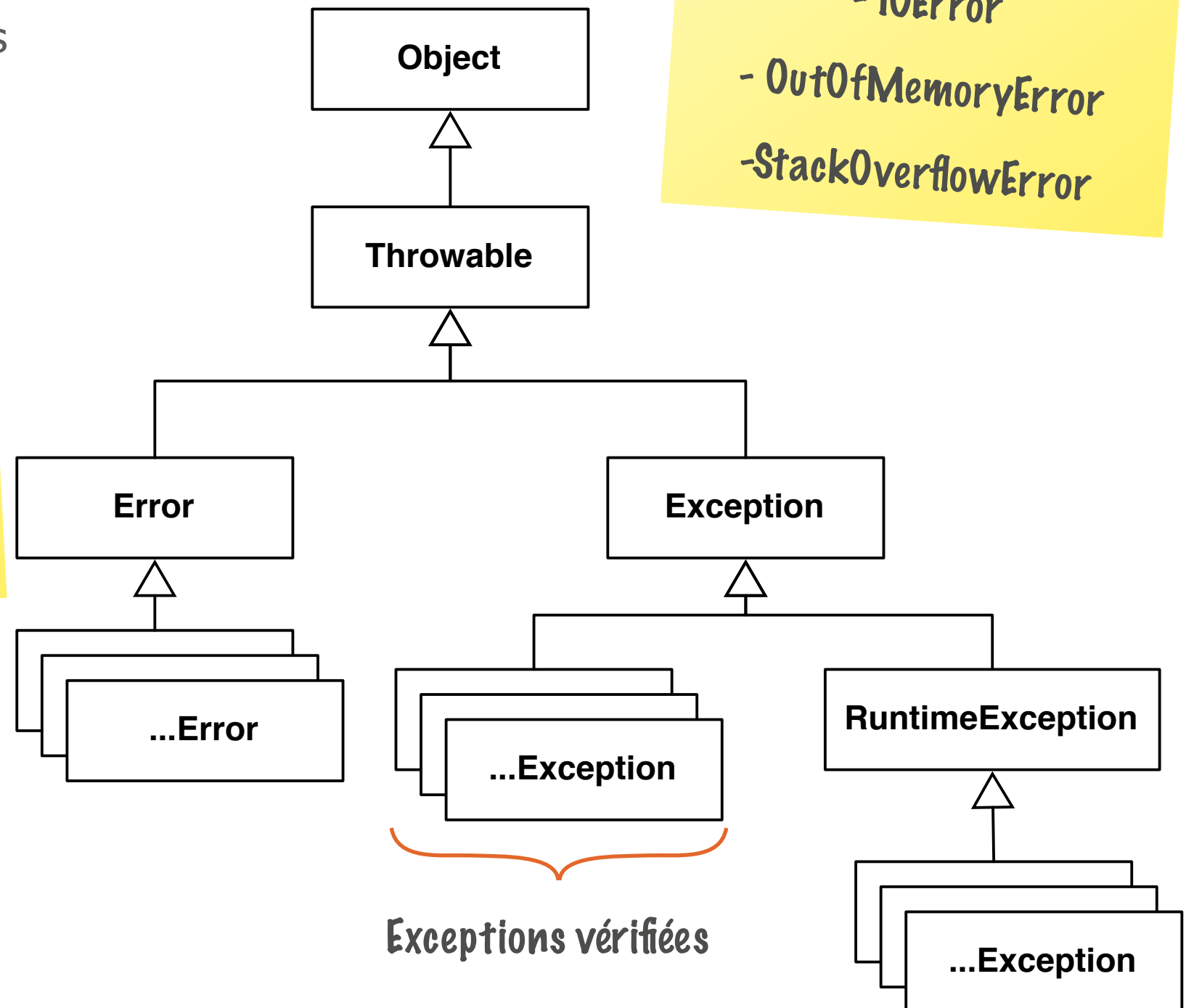
- ArithmeticException  
(division par zéro)
- NullPointerException
- IndexOutOfBoundsException
- IllegalArgumentException

#### Exemples d'exceptions vérifiées

- SocketException
- FileNotFoundException

#### Exemples d'erreurs

- IOException
- OutOfMemoryError
- StackOverflowError



## ☐ Les exceptions : gestion des erreurs

### ☐ Syntaxe

☐ Lors de l'appel d'une méthode pouvant lancer une exception vérifiée, il faut soit :

☐ la gérer (**try/catch**)

☐ la re-déclarer (clause **throws** dans la déclaration de la méthode)

☐ Gestion de l'exception (**try/catch**)

Chemin "normal" {

```

try {
    UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
} catch (Exception e) {
    e.printStackTrace();
}
    
```

Si une exception se produit on affiche le contenu de la pile des appels (pas nécessairement la meilleure solution en production)

## ☐ Les exceptions : gestion des erreurs

### ☐ Syntaxe

#### ☐ Redéclaration de l'exception (**throws**)

- ☐ La méthode englobante doit redéclarer toutes les exceptions que peuvent lancer les méthodes qu'elle appelle
- ☐ La gestion est donc reportée au niveau supérieur (ici c'est la JVM qui devra gérer)

```
public static void main(String[] args) throws ClassNotFoundException,  
InstantiationException, IllegalAccessException, UnsupportedLookAndFeelException {  
  
    //...  
    UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());  
    //...
```

## ❑ Les exceptions : gestion des erreurs

### ❑ Syntaxe

#### ❑ Gestion de l'exception (**try/catch**)

- ❑ dans l'exemple précédent on avait regroupé toutes les exceptions dans un seul bloc catch (en exploitant le polymorphisme)
- ❑ On peut affiner le traitement en fonction de la nature de l'exception

```
try {  
    UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());  
} catch (ClassNotFoundException e) {  
    //Faire qqchose en rapport avec cette exception  
}  
catch (InstantiationException e) {  
    //Faire qqchose en rapport avec cette exception  
}  
catch (IllegalAccessException e) {  
    //Faire qqchose en rapport avec cette exception  
}  
catch (UnsupportedLookAndFeelException e) {  
    //Faire qqchose en rapport avec cette exception  
}
```

## ❑ Les exceptions : gestion des erreurs

### ❑ Exemples

#### ❑ Importer le fichier `fr.lpiem.erreurs.Calcul`

❑ Observer l'utilisation des exceptions pour vérifier le domaine de variation des arguments

❑ Est-ce une exception vérifiée ?

#### ❑ Importer le fichier `fr.lpiem.erreurs.CalculAvecTryCatch`

❑ Tester le programme et provoquer une exception

❑ Quelle méthode provoque l'exception "attrapée" ?

❑ Est-ce une exception vérifiée ?



## ❑ Les exceptions : gestion des erreurs

### ❑ Exemples

- ❑ Importer le fichier `fr.lpiem.erreurs.CalculAvecTryCatchTouteException`
- ❑ Analyser la gestion des exceptions (blocs catch) et tester le programme
- ❑ Que fait la méthode `getMessage()` ?
- ❑ Dans quelle classe est-elle définie ?
- ❑ Où le message est-il initialisé ?
- ❑ Tester les différentes exceptions

## ☐ Les exceptions : gestion des erreurs

### ☐ Syntaxe

- ☐ Le mot clé **finally** permet de fournir un bloc de traitement exécuté systématiquement qu'il y ait eu une exception ou non
- ☐ Ce bloc permet de faire le ménage quand le traitement a été interrompu par une exception (fermeture de fichiers, connexions réseaux,...)

### ☐ Exemple

- ☐ Importer le fichier `fr.lpiem.erreurs.CalculAvecTryCatchFinally`
  - ☐ Vérifier que le bloc `finally` est exécuté dans tous les cas de figure

## ☐ Les exceptions : gestion des erreurs

### ☐ try-with-resources

#### ☐ Depuis Java 7

☐ Permet de simplifier la fermeture des ressources en cas d'exception, en ne fermant que les ressources dont l'ouverture est effectivement allée à bout (distinction entre exceptions à l'ouverture de la ressource et exception pendant l'utilisation de la ressource)

☐ Les classes compatibles implémentent l'interface Autocloseable

☐ Pour mesurer l'ampleur du travail réalisé par cette construction, il faut analyser le code généré par le compilateur java  
<https://romain.vernoux.fr/a-try-with-resources-equivalent-for-java-6-and-android/>

## ❑ Les exceptions : gestion des erreurs

### ❑ try-with-resources - exemple

```
public class EchoServer {  
  
    public static void main(String[] args) {  
  
        try (ServerSocket serverSocket = new ServerSocket(50007, 1, null);  
             Socket clientSocket = serverSocket.accept();  
             PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);  
             BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));  
             ) {  
            System.out.println("Connected by " + clientSocket.getInetAddress());  
            out.println(in.readLine());  
            clientSocket.close();  
            serverSocket.close();  
        } catch (Exception e) {  
            e.printStackTrace(); //Gestion des erreurs  
        }  
    }  
}
```

## ❑ Exceptions ou assertions

- ❑ Les méthodes publiques doivent être protégées par des exceptions (documentées) contre des arguments invalides

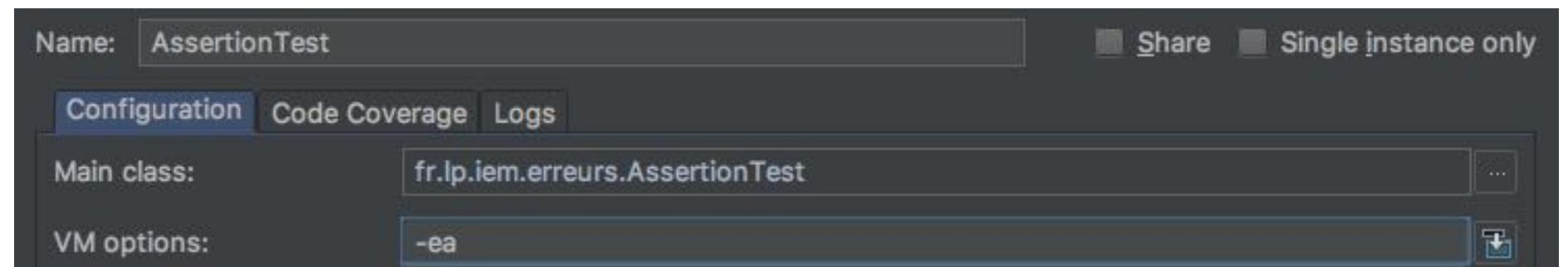
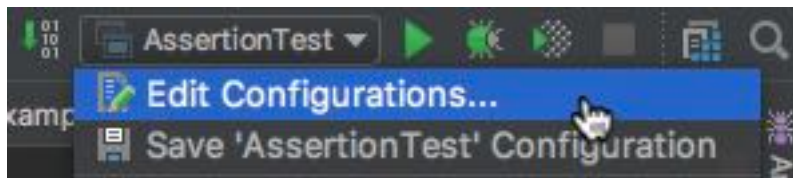
```
/**
 * Returns a BigInteger whose value is (this mod m). This method
 * differs from the remainder method in that it always returns a
 * non-negative BigInteger.
 *
 * @param m the modulus, which must be positive
 * @return this mod m
 * @throws ArithmeticException if m is less than or equal to 0 */
public BigInteger mod(BigInteger m) { if (m.signum() <= 0)
    throw new ArithmeticException("Modulus <= 0: " + m);
    //... Do the computation
}
```

## ☐ Exceptions ou assertions

- ☐ Pour les méthodes non exportées, l'auteur du package maîtrise les circonstances dans lesquelles les méthodes sont appelées
- ☐ Il est quand même nécessaire, à des fins de debug et de documentation de tester le bon respect des domaines de validité des arguments d'une méthode : plages de valeurs, non nullité d'un argument, toute autre hypothèse
- ☐ Solution élégante : utiliser les assertions
  - ☐ Les assertions sont activables
    - ☐ A la différence des exceptions, elles peuvent être retirées du code de production pour améliorer les performances
    - ☐ Il faut passer l'option -ea à la machine virtuelle java pour que les assertions soient prises en compte
    - ☐ Lorsqu'elles sont activées, le non respect d'une condition déclenche une AssertionError

## ☐ Assertions

- ☐ Activation des assertions sous IntelliJ Idea



Ajouter l'option à la machine virtuelle Java



## ☐ Assertions

- ☐ Utiliser l'instruction `assert` suivi d'une condition booléenne

```
// Private helper function for a recursive sort
private static void sort(long a[], int offset, int length) {
    assert a != null;
    assert offset >= 0 && offset <= a.length;
    assert length >= 0 && length <= a.length - offset;
    //... Do the computation
}
```



## ☐ Assertions

- ☐ Utiliser l'instruction assert suivi d'une condition booléenne

```
public class AssertionTest {  
    static class Joueur {  
        int age;  
  
        void jouerAuMonopoly() {  
            assert (age >= 7 && age <= 77);  
            System.out.println("J'achète un hôtel rue de la Paix");  
        }  
    }  
  
    public static void main(String[] args) {  
        Joueur donald = new Joueur();  
        donald.age = 8;  
        donald.jouerAuMonopoly();  
  
        Joueur gontran = new Joueur();  
        gontran.age = 6;  
        gontran.jouerAuMonopoly();  
    }  
}
```