

M2103 – Bases de la Programmation Orientée Objets



Java – Cours 9

Exceptions

Plan du Cours

- Traitement d'Erreurs avec Exceptions
- Exceptions Vérifiées et Non Vérifiées
- Lancement d'Exceptions
- Récupération d'Exceptions

Répondre à l'Echec

- Etape 1: Détecter le problème
- Etape 2: Rapporter le problème – pour que l'utilisateur sache que quelque chose d'inattendu s'est déroulé
- Etape 3: Traiter le problème de manière appropriée – le programme doit alors continuer normalement.

Jusqu'à Maintenant...

```
public boolean supprime (int cle)
{
    if (cle < 0)
    {
        System.out.println ("Élément n'existe pas");
        return false;
    }
    else ...
}
```

Solution Java

- Exception

- Événement se déroulant durant l'exécution d'un programme qui interrompt son cours normal
- Objet en Java (instance de la classe **Exception** ou sous-classe)

```
public Exception()
```

Construit une **Exception** sans message spécifié

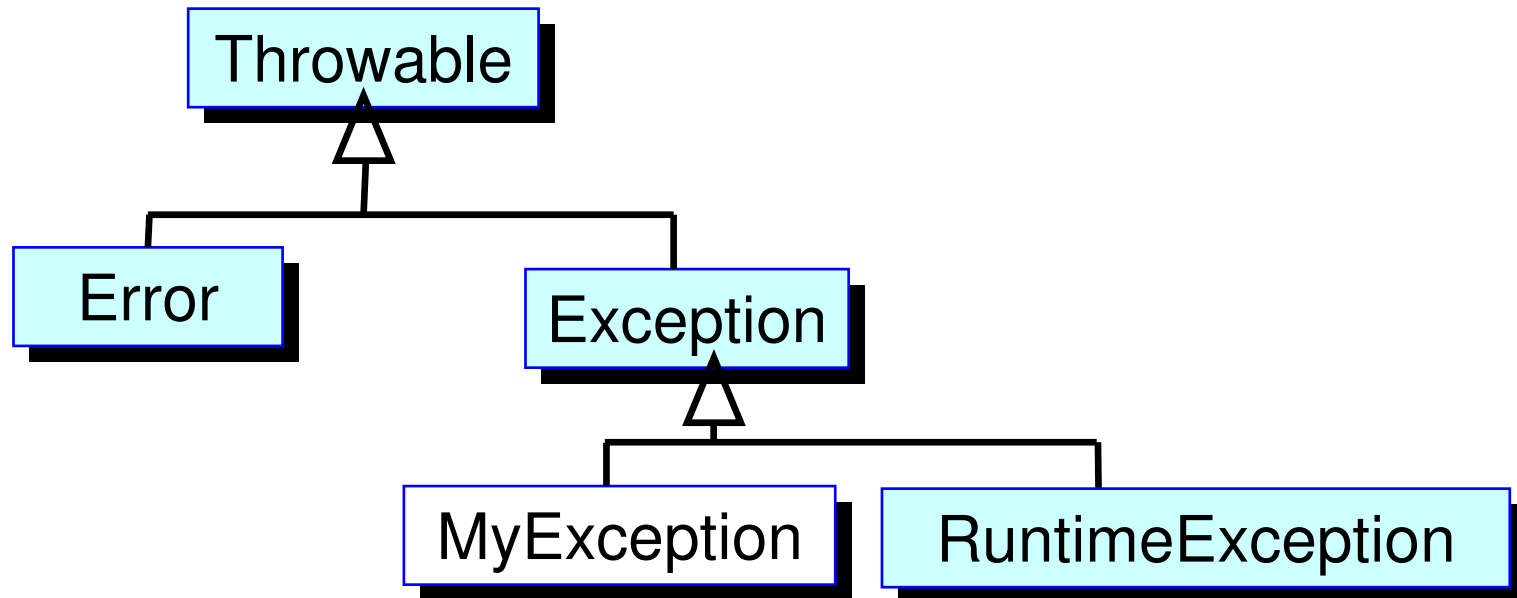
```
public Exception(String s)
```

Construit une **Exception** avec le message spécifié *s*

- Traitement d'Exception

- Mécanisme pour passer le contrôle à un “gestionnaire d'erreurs” dès qu'une erreur est détectée

Hiérarchie **Exception**



- **Throwable** unifie erreurs et exceptions
- Erreurs ne sont pas capturées contrairement aux exceptions
- Classes exceptions définies par l'utilisateur héritent de la classe **Exception**

Exceptions Vérifiées vs. Non Vérifiées (1)

Terminologie :

La plupart des exceptions sont des **exceptions vérifiées**. Elles sont déclarées dans une instruction **throws** de l'en-tête de la méthode, ou traitées dans le corps de la méthode.

Les exceptions non vérifiées peuvent se dérouler à n'importe quel moment et ne doivent pas être nécessairement déclarées.

- Exceptions non vérifiées en Java sont implémentées par les classes **RuntimeException** et **Error**

Exceptions Vérifiées vs. Non Vérifiées (2)

- Java définit un certain nombre d'exceptions, vérifiées et non vérifiées, qui sont spécifiées dans l'API Java (à consulter)
- Exceptions Vérifiées
 - Détaillées dans la suite
 - Exemple : `IOException`
- Exceptions Non Vérifiées
 - Rencontrées lorsque
 - Situation pour laquelle le programmeur n'a pas de contrôle
 - Erreur de programmation
 - Exceptions lancées par la machine virtuelle Java
 - Exemples
 - `InputMismatchException`
 - `NullPointerException`
 - `ArrayIndexOutOfBoundsException`
 - `ClassCastException`
 - `IllegalArgumentException`

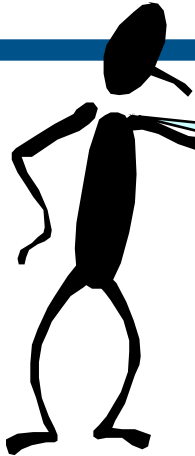
Lancement d'Exceptions

Méthodes peuvent lancer leur propre exception vérifiée pour signaler un problème

```
public void supprime (int numElt) throws Exception
{
    if (numElt < 0 || numElt > compteur)
    {
        throw new Exception();
    }
    ...
}
```

Une méthode lançant une exception vérifiée doit la déclarer dans son en-tête

Création d'Exceptions Sur Mesure



Exceptions sur mesure créées
en héritant de la classe **Exception**

```
class NombreHorsLimiteException extends Exception
{
    /**
     * Création d'une nouvelle exception avec le nombre
     * illégal comme paramètre
     */
    NombrehorsLimiteException (int nombre)
    {
        super("Le nombre " + nombre + " est hors limite");
    }
}
```

Lancement d'Exceptions Sur Mesure

```
public void supprime (int numElt)
    throws NombreHorsLimiteException
{
    if (numElt < 0 || numElt > compteur)
    {
        throw new NombreHorsLimiteException(numElt);
    }
    ...
}
```

Traitement d'Exceptions

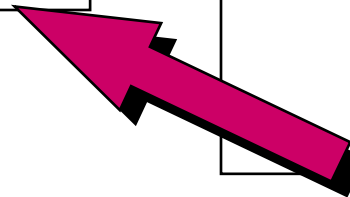
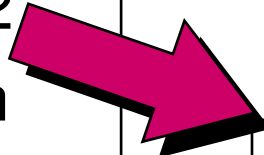
- Souvent, on ne veut pas seulement signaler un problème, mais également le traiter. Le programme ne doit pas terminer son exécution sur un échec !
- On peut capturer l'exception et :
 - La traiter
 - La lancer une nouvelle fois pour un traitement ultérieur

méthode1:

appelle méthode2
capture exception
la traite ou la relance

méthode2:

détecte erreur
lance exception



Les Mots-clé **try** et **catch**

Lors de l'appel d'une méthode générant potentiellement une exception, on la place dans un bloc **try**

```
try
{
    //code générant potentiellement une exception
    maStructure.supprime (numElt) ;
}
```

Quand une exception est lancée dans un bloc **try**, le programme recherche un bloc **catch** correspondant

```
catch (NombreHorsLimiteException probleme)
{
    // traitement approprié
}
```

Synthèse

```
...  
int numElt = getEntreeUtilisateur();  
try  
{  
    structure.supprime(numElt);  
}  
catch (NombreHorsLimiteException probleme)  
{  
    System.out.println("Erreur : " + probleme);  
}  
...
```

Capturer Différentes Exceptions

```
...  
int numElt = getEntreeUtilisateur();  
try  
{  
    structure.supprime(numElt);  
}  
catch (NombreHorsLimiteException probleme)  
{  
    System.out.println("Erreur : " + probleme);  
}  
  
catch (Exception probleme)  
{  
    System.out.println("Erreur : " + probleme);  
}  
...
```

Points Importants à Souligner

- Possibilité d'écrire autant de blocs **catch** que voulu
- Eviter d'écrire des blocs **catch** génériques – il faut cerner l'erreur de manière la plus spécifique possible
- Lorsqu'une exception est lancée, le contrôle de la situation est cédé au bloc **catch** approprié, puis à l'instruction suivant le bloc **try**
- Les exceptions peuvent être relancées de manière à être traitées à un autre endroit du programme.

```
catch (NombreHorsLimiteException probleme)
{
    System.out.println("problème à traiter ailleurs");
    throw probleme;
}
```


Le Mot-Clé **finally**

- Quand une exception est capturée, le contrôle est cédé au bloc **catch** correspondant et le code qui se trouve encore dans le bloc **try** n'est pas exécuté. On ne veut pas toujours ceci !
- L'instruction **finally** optionnelle est toujours exécutée après un bloc **try**, indépendamment de ce qui s'est déroulé précédemment dans le bloc **try**
- Possibilité d'utiliser des blocs **finally** pour 'nettoyer' après une opération

try et finally

```
public void uneMethode ()
{
    ....
    try
    {
        // code
    }
    catch
    {
        ....
    }
    finally
    {
        // 'nettoyage'
    }
    ....
}
```

toujours exécuté !



Exemple

```
...  
int numElt = getEntreeUtilisateur();  
try  
{  
    structure.supprime(numElt);  
}  
catch (NombreHorsLimiteException probleme)  
{  
    System.out.println("Erreur : " + probleme);  
}  
catch (Exception probleme)  
{  
    System.out.println("Erreur : " + probleme);  
}  
  
finally  
{  
    System.out.println("Fin Programme");  
}  
...  
...
```