JAVA
Les exceptions

LICENCE 3 Sciences pour l'Ingénieur Parcours Informatique

Les exceptions

- Comment faire remonter une erreur en Java ?
  - A la compilation : moment idéal pour repérer les erreurs
  - Si aucune erreur à la compilation : à l'exécution. Comment anticiper les erreurs ?
- Comment gérer une erreur quand elle arrive ?
  - Résoudre le problème tout de suite (si c'est possible) pour continuer l'exécution du code
  - Transmettre l'erreur (l'exception) à la méthode appelante en espérant qu'elle puisse trouver une solution et réparer l'erreur
    -> **C'est la propagation d'exception**

Différents types d'erreurs

- Fautes de saisie de l’utilisateur
  - URL obsolète...
- Erreurs du matériel
  - Une imprimante déconnectée ou qui manque de papier (au milieu d’une impression)
- Contraintes physiques
  - Disque plein ou manque de mémoire
- Erreurs de fichier :
  - inexistant
  - contenant une information incorrecte
  - interdit en écriture, lecture...
- Connexion réseau défaillante
- Index de tableau invalide
- Emploi d’une référence à un objet non initialisée

Avantage des exceptions

- Le compilateur vous force à prendre en compte les erreurs
- Méthode de travail permettant de séparer le code d'application et le code de gestion d'erreurs
- En cas d'erreur :
  - Revenir à un état défini et offrir à l’utilisateur la possibilité d’exécuter d’autres commandes
  - Permettre à l’utilisateur de sauvegarder son travail et sortir normalement du programme

Exemple de code générant une exception

```
class DivByZero {
	public static void main(String args[]){
		System.out.println(3/0);
		System.out.println(“Pls. print me.”);
	}
}
```

- Génère l'exception suivante :

```
Exception in thread "main"
java.lang.ArithmeticException: / by zero at
DivByZero.main(DivByZero.java:3)
```

- Mode de gestion automatique des Exceptions
  - Assuré par la Machine virtuelle
  - Ecrit une description de l’exception
  - Imprime la pile d’appel
  - Cause l’arrêt du programme

Comment savoir qu'une méthode génère un exception

- On regarde la doc !

javax.xml.stream

Interfaces

EventFilter
Location
StreamFilter
XMLEventReader
XMLEventWriter
XMLReporter
XMLResolver
XMLStreamConstants
XMLStreamReader
XMLStreamWriter

Classes

XMLEventFactory
XMLInputFactory
XMLOutputFactory

Exceptions

XMLStreamException

Errors

FactoryConfigurationError

Capturer et traiter une exception

- Si une méthode est susceptible de propager une exception, il faut l'insérer dans un bloc **try**
- Pour traiter l'erreur, déclarer un bloc **catch**

```
try{
      // Code susceptible de lever une exception
      // Code « normal »
} catch (XXXException1 e1){
      // Code de traitement de l'exception e1
} catch (XXXException2 e2){
      // Code de traitement de l'exception e2
}
```

Exemple de code

```
public static void main(String args[]){
	int n;
	try {
		n = Integer.parseInt(args[0]);
	}
	catch(NumberFormatException e) {
		System.err.println("Mauvais nombre passé en
					paramètre");
	}
}
```

Traiter toutes les exceptions en une seule fois

- Faire un catch sur un objet du type Exception qui est la classe mère de toutes les exceptions

```
catch (Exception e){
     System.out.println("Capture d'une Exception"+e);
}
```

- Le traitement d'erreur est alors beaucoup moins fin...
- À placer en dernier de la liste des Exceptions capturées

Que faire pour lever une exception depuis votre code, méthode ou constructeur ?

- Bien réfléchir au type d'exception
- Créer une nouvelle instance de l'exception choisie
- La lancer dans la pile de propagation des exceptions
- Signaler dans l'en-tête que votre code est susceptible de propager cette(ces) exception(s)

Déclencher (lever) une Exception via throw

- Pour lever une exception il faut
  - Créer une nouvelle instance de l'exception que l'on a choisi
  - Puis la lancer dans la pile d'appel via le mot clé throw

```
if (t==null) throw new NullPointerException();
```

- On peut passer un message en paramètre du constructeur d'exception pour rendre le code plus explicite

```
if (t==null)
    throw new NullPointerException("t==null");
```

- Il y a toujours deux constructeurs pour toutes les exceptions

Signaler que votre méthode propage via throws

- Lorsque vous ne savez pas comment traiter une ou des exception(s) susceptible(s) d'être lancée(s) par du code invoqué dans une de vos méthode vous pouvez simplement vous faire intermédiaire et propager cette ou ces exception(s)
- `typeRetour nomMéthode(...) throws XXXException, YYYException{...}`
- L'exception est alors propagée dans la pile d'appel et voyage jusqu'à trouver une méthode susceptible de traiter l'erreur
- Toutes les exceptions propagées doivent être mentionnées

Re-propager une exception capturée

- Parfois vous pouvez avoir besoin de re-propager une exception que vous avez attrapée, en particulier si vous avez utilisé une référence à Exception

```
catch(Exception e){
	System.out.println("Une exception était propagée");
	throw e;
}
```

- Cette exception est propagée dans la pile d'appel supérieure (les éventuels autres catch sont ignorés)

Invoquer une méthode propageant des exceptions

- Vous voulez dans votre code (méthode truc) utiliser la méthode machin dont la signature est la suivante :
  - `public machin(int i) throws TestException`
- Elle est donc susceptible de lever une exception de type **TestException**
- Si vous l'utilisez sans prendre en compte les exceptions vous obtiendrez aun message d'erreur à la compilation
- Pour éviter cela vous avez trois possibilités
  - 1. Traiter l'exception
  - 2. Ne pas traiter l'exception et la propager en espérant une méthode appelante saura quoi faire
  - 3. Traiter partiellement l'exception et la re- propager pour qu'une méthode appelante puisse terminer le rattrapage

1. Traiter l'exception

- Si vous savez comment traiter l’anomalie
  - Utilisez un bloc **try/catch** dans lequel est implémenté un code de traitement d’erreur
- Exemple : `public machin(int i) throws TestException`

```
public truc(int i, MaClasseTest m){
	int monI;
	try{
		monI = m.machin(i);
	}catch(TestException e){
		// Faire quelque chose
	}
}
```

2. Propager l'exception

- Si vous ne savez pas comment traiter l’erreur, vous propagez simplement l’exception et une autre méthode fera le travail
  - La méthode doit signaler la propagation d’erreur via throws
- Exemple : `public machin(int i) throws TestException`

```
public truc(int i, MaClasseTest m) throws TestException{
        int monI = m.machin(i);
}
```

Traiter l'exception et la re-propager

- Vous savez en partie ce qu'il faut faire en cas d'erreur mais ne pouvez pas tout faire, vous traitez l'exception à votre manière et vous la re-propagez, une méthode appelante finira le travail
  - Bloc try/catch, avec à la fin du catch la propagation de l'erreur via throw
  - La méthode appelante doit également signaler la propagation de l'erreur via throws
- Exemple : `public machin(int i) throws TestException`

```
public truc(int i, MaClasseTest m) throws TestException{
	int monI;
	try{
		monI = m.machin(i);
	}catch(TestException e){
		//faire quelquechose...
		throw e;
		//et relancer l'exception
	}
}
```

Les méthodes de la classe Exception

- Elle hérite de Throwable un ensemble de méthodes
  - `String getMessage(), String getLocalizedMessage()`
    - Retourne le message d'erreur associé à l'exception (dans la langue locale pour Localized...)
  - `String toString()`
  - `void printStackTrace(),`
  - `void printStackTrace(PrintStream)`
    - Écriture de la pile d'appels des méthodes ayant menées au problème

Finally

- Lorsque votre code lance une exception, il interrompt le code restant dans votre méthode et sort de celle-ci.
- Cela peut poser problème lorsqu'elle a allouer une ressource locale (fichier...)
- La ressource doit être désallouer
- Le bloc optionnel finally est exécuté dans tous les cas...
- Exemple :

```
try{
      // Code susceptible de lever une exception
} catch {
      // Code de traitement des exceptions
} finally {
      // Code exécuté quoiqu'il arrive
}
```

- Il est possible d'avoir un bloc try/finally sans catch

Finally

- Ce bloc est toujours exécuté
  - Même si un des **catch** précédents se termine par un **return** ou lance une exception
- Un seul cas de non exécution : si on a utilisé dans un **catch** un `System.exit()` qui sort directement de l'application
- Si une des instructions du bloc **try** produit une exception
  - Si un **catch** attrape cette exception il est exécuté et ensuite la main passe au bloc **finally** qui est à son tour exécuté
  - Si l'exception n'est pas attrapée (aucun **catch** correspondant) le bloc **finally** est tout de même exécuté et l'exception est ensuite propagée vers la méthode appelante

Le bloc try-with

- On peut utiliser **try** avec un argument pour initialiser une variable.
- Cette variable doit être d'un type qui implémente l'interface **AutoCloseable** ou **Closeable**.
  - Ces interfaces ne déclarent qu'une seule méthode : **close**
- Exemple :

```
static String readFirstLineFromFile(String path) throws IOException {
       try (BufferedReader br = new BufferedReader(new FileReader(path))){
       }
}
```

- Correspond à :

```
static String readFirstLineFromFileWithFinallyBlock(String path)
                                                  throws IOException {
       BufferedReader br = new BufferedReader(new FileReader(path));
       try { return br.readLine(); }
       finally { if (br != null) br.close();
}
```

Créer vos propres classes d'exception

- Quand et pourquoi créer ses propres classes d’exceptions ?
  - Vous rencontrez un problème qui n’est pas décrit de façon satisfaisante par les classes d’exceptions Java prédéfinies
  - Vous voulez adapter des exceptions Java pré-existantes pour mieux gérer des erreurs susceptibles d’être générées par vos classes
    - L’information transportée par les classes standard est insuffisante
- Faire dériver sa classe de la classe **Exception** ou d’une de ses classes filles
- Nommer votre classe en respectant la forme **XXXException**
- Définir deux constructeurs minimum :
  - Constructeur vide
  - Constructeur à un argument de type **String** pour permettre l’affichage d’un message (affiché par la méthode **toString** de la classe **Throwable**)

Syntaxe des Exceptions personnalisées

```
class MonException extends Exception{
    public MonException(){
    }

    public MonException(String message){
        super(message);
    }
}
```