TRAVAUX PRATIQUES Initiation à la Programmation Orientée Objet en Java

DOCUMENTATION 3 - Exceptions en Java

Objet :

Utilisation du mécanisme d'exception du langage Java.

Définition :

Une exception, comme son nom l'indique sert à gérer des cas exceptionnels. La plupart du temps ces cas sont assimilés à des erreurs.
En Java il existe deux sortes d'exceptions : les exceptions de la Java Virtual Machine (*runtime exception*) et les exceptions de l'utilisateur (*user exception*). Les exceptions de la JVM sont générées par exemple lors d'un débordement de tableau, d'une tentative d'accès à travers une référence null, d'un problème réseau...
Les exceptions de l'utilisateur peuvent être créées par le programmeur pour gérer des cas de mauvaise utilisation d'un objet particulier.
Le *runtime* recherche « qui » peut récupérer l'exception lancée dans l'ensemble de la pile des appels. Si personne ne gère (catch) cette exception, c'est la JVM qui la récupère, l'affiche et arrête l'exécution du programme.

Créer une exception utilisateur :

Une exception est un objet comme un autre mais qui ne possède pas de méthodes spécifiques. Il existe 2 constructeurs pour les exceptions en Java :

```
Exception()
```
Construit une Exception sans message détaillé
```
Exception(String s)
```
Construit une Exception avec message détaillé

Lever une exception :

Les *runtime exception* sont levées automatiquement à l'exécution par la JVM.
Lorsque l'on détecte un cas particulier (par exemple une tentative de dépiler dans une pile vide), il est souhaitable de "lever" une *user exception.*

```
Exception pileVideException = new Exception("PileVide");
if (pileVide()) throw PileVideException;
```

ce que l'on peut écrire de manière condensée par

```
if (pileVide() throw new Exception("PileVide");
```

Transmettre une exception :

Lorsque le code d'une méthode contient au moins une clause throw on doit spécifier dans l'en-tête de la méthode que cette exception peut être lancée dans le corps de la méthode.
De même, si l'on appelle une méthode qui peut lancer une exception et que l'on ne souhaite pas la traiter on doit la transmettre. Exemple :

```
public Object depiler() throws Exception {
 ...
 if ( pileVide() ) throw new Exception("Dépilement de pile vide");
 ...;
}
```

TRAVAUX PRATIQUES Initiation à la Programmation Orientée Objet en Java

Récupérer une Exception :

Lorsque l'on sait traiter une Exception qui a été lancée par une méthode, on le fait grâce à la construction suivante :

```
try { bloc dans lequel une exception peut être lancée }
catch (Exception e) { bloc dans lequel on traite l'exception }
```

on "récupère" alors dans e l'exception lancée dans le bloc `try{...}`

`Exception` hérite de `Throwable` qui possède 2 méthodes très utiles :

```
String getMessage()
      rend le message d'erreur contenu dans l'exception
void printStackTrace()
      affiche le message d'erreur et le contenu de la pile d'exécution
```

Exemple d'utilisation :

```
try {element = p.depiler; }
catch (Exception e) { System.out.println(e.getMessage() ); }
```

En Java on doit appliquer la politique "*catch or throws*" de manière explicite (récupérer ou transmettre) pour toutes les exceptions de l'utilisateur. Pour les *runtime exception*, si on ne traite pas une exception, celle-ci est implicitement transmise sans devoir l'indiquer.

Définir des classes d'exceptions :

Bien que l'on puisse se débrouiller dans de petits programmes en utilisant directement Exception il est plus rationnel d'utiliser des classes dérivées de Exception. Java en fournit un grand nombre mais dans quelques cas vous pouvez aussi définir les vôtres soit en héritant directement de Exception soit d'une de ses classes dérivées (de préférence). L'utilisation d'exceptions de différentes classes permet en particulier de faire un traitement différent selon leur "type", on peut alors avoir plusieurs `catch` par `try`.

Exemple de définition d'exception :

```
class MonException extends Exception{
  public MonException(String s){super(s);}
  public MonException(){super();}
};
```

Par convention, il est habituel (*good practice*) de terminer le nom d'une exception par Exception.

Exemple de traitements multiples (inspiré du tutoriel java) :

```
readFile {
  try {
        open the file;
        read the file into memory;
        close the file;
        }
  catch (fileOpenFailed) {doSomething; }
  catch (readFailed)        { doSomethingElse; }
  catch (fileCloseFailed) { doSomethingElse; }
}
```

TRAVAUX PRATIQUES Initiation à la Programmation Orientée Objet en Java

Ce qu'il ne faut pas faire :

Pour le programmeur impatient, les Exceptions sont une contrainte désagréable. On trouve trop souvent dans les programmes :

```
try{ appels de méthodes pouvant générer des exception }
catch (Exception e);   ← bloc vide sans aucun traitement !!!
```

Le résultat est que s'il se produit une exception, il ne se passe rien de visible. Et donc il est quasi impossible de savoir ce qu'il s'est passé.

Pourquoi utiliser des exceptions :

Tout d'abord, quels sont les autres choix pour traiter un cas d'erreur ?

- rendre un code d'erreur est une alternative viable uniquement si l'appelant est capable de traiter le code d'erreur car la propagation de l'erreur n'est pas automatique,
- faire une sortie `System.out.print("erreur...")` n'est possible que si on a un écran, ... et quelqu'un de compétent devant, de plus, cela ne permet pas le traitement de l'erreur,
- terminer abruptement le programme par `System.exit()` ne permet pas de savoir où l'erreur s'est produite.

Dans tous les cas de traitements exceptionnels (erreurs), il est préférable d'utiliser les Exceptions.

Remarques :

On peut encapsuler un grand nombre d'appels de méthodes dans un seul bloc try mais cela signifie qu'à la première erreur on abandonne le traitement de l'ensemble du bloc ce qui est logique dans l'exemple précédent mais n'est pas toujours souhaitable.

Les exceptions ont un coût non négligeable à l'exécution. En effet le *run-time* doit mettre en place le mécanisme de récupération. Il ne faut donc pas non plus abuser des exceptions pour traiter des cas non exceptionnels qu'un simple `if` permet de résoudre.

Ce fichier se trouve sous `/home/clyde/d1/insa3/commun/java/`