LES EXCEPTIONS JAVA

Ludovic Liétard

LES EXCEPTIONS (1)

- Une exception correspond à un événement anormal ou inattendu
- L'événement doit être peu souhaitable ou peu probable
- Une exception est une interruption logicielle
- Exemple : division par zéro, lire un caractère au lieu d'un entier

LES EXCEPTIONS (2)

- Le concepteur d'une classe peut alors prévoir une instruction du genre

> Si la situation est « telle situation » alors
> lancer une exception instance de
> ExceptionTelleSituation

- La classe ExceptionTelleSituation est à concevoir. Elle étend la classe `java.lang.Exception`

LES EXCEPTIONS (3)

- Exemple :

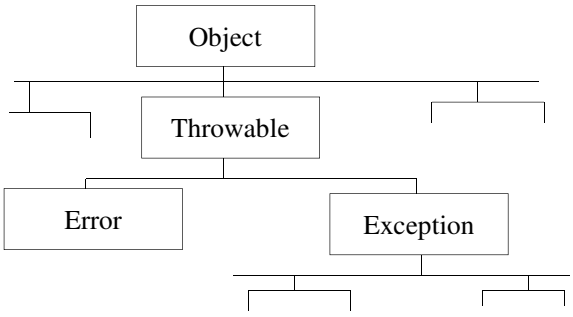
La méthode de classe `parseInt(String s)` de la classe `Integer` est prédéfinie et elle convertit l'instance de chaîne s en entier. Son code contient l'instruction :

Si s n'est pas de format entier alors lancer une exception instance de `NumberFormatException`

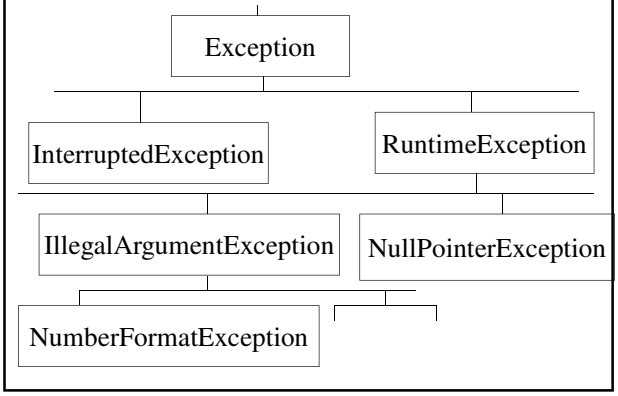
Cette exception correspond à un événement qui est une erreur. On verra qu'il est possible de traiter cette exception, ce qui permet de continuer le programme.

LES EXCEPTIONS (4)

- Hiérarchie des classes



LES EXCEPTIONS (5)



PROPAGATION

- Une exception qui est lancée se propage de la manière suivante :

Contexte : l'exception est lancée par une méthode 1,

Cette méthode 1 est appelée par une méthode 2. La méthode 2 est appelée par une méthode 3, et ainsi de suite jusqu'à la méthode main…

Propagation : l'exception se propage, tant qu'elle n'est pas traitée :

méthode1 ⇒ méthode 2 ⇒ méthode 3 ⇒ … ⇒ main puis arrêt de l'exécution

PROPAGATION

- Il faut rajouter:

  … throws NomException {

- Dans l'entête de la méthode qui propage NomException.

TRAITER UNE EXCEPTION (1)

- Pour qu'une exception soit traitée, il faut que la méthode qui la déclenche ou la propage soit dans un bloc `try` et que l'exception soit prévue dans un bloc `catch` associé

```
try {       appel à la méthode 1;
            appel à la méthode 2;
            appel à la méthode 3;
 }

catch (NomException e){
            <appels de méthodes>}
```

TRAITER UNE EXCEPTION (2)

- Si la méthode 2 déclenche une exception, le code qui reste à exécuter dans le bloc `try` est ignoré puis :
- Si l'exception est une instance de exception1 (ou exception1 en est une superclasse) alors la section <appels de méthodes > est effectuée. Le programme reprend normalement avec le code qui suit le bloc catch
- Sinon l'exception est propagée

DEFINIR UNE EXCEPTION (1)

- Il faut étendre la classe `EXCEPTION` ou une de ses sous-classes
- La classe étendue contiendra :

  Un (ou plusieurs) constructeurs
  Éventuellement la redéfinition
  de la méthode `toString`

LANCER UNE EXCEPTION (1)

- Le mot réservé `throw` appliqué à une instance d'exception est utilisé pour lancer une exception

```
if <condition> then
throw
new NomException();
```

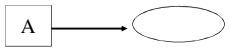
Utiliser un bloc finally

- Un bloc `finally` suit un bloc `try` est est toujours exécuté, peu importe la manière dont on est sorti du bloc `try` (des blocs `catch` peuvent s'intercaler)

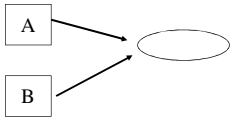
```
try {       <appels à des méthodes>
 }

catch (exception1 e){
            <appels à des méthodes>}

finally {<appels à des méthodes>
 }
```

CLONAGE ET DUPLICATION D'OBJET (1)

- La duplication superficielle d'un objet : les références sont copiées

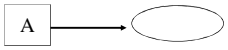


Après la copie superficielle de A sur B (affectation B=A) :

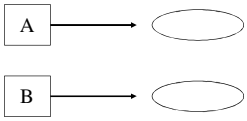


CLONAGE ET DUPLICATION D'OBJET (2)

- La duplication en profondeur d'un objet : les contenus sont copiés



Après la copie en profondeur de A sur B :



CLONAGE ET DUPLICATION D'OBJET (3)

- La méthode `clone()` de la classe `Object` délivre une duplication superficielle de l'objet concerné :

```
protected Object clone() throws
            CloneNotSupportedException;
```

- L'objet doit implémenter l'interface `Cloneable`, sinon l'exception `CloneNotSupportedException` est lancée