Les exceptions

Mise à niveau Java

Y. Boichut & F. Moal

Fiabilité d’un logiciel

La fiabilité d’un logiciel concerne

- La robustesse : capacité du logiciel à fonctionner même en présence d’ évènements exceptionnels (mauvaises valeurs, etc)
- La correction : le fait que le logiciel fait ce qu’il doit faire quand il doit le faire

Fiabilité en Java

2 mécanismes facilitent une bonne fiabilité

- Les exceptions : pour la robustesse
- Les assertions : pour la correction (au programme dans le module “Java avancé”)

Erreurs / Exceptions

- On utilise les erreurs / exceptions pour traiter un fonctionnement anormal d'une partie du code (provoqué par une erreur ou un cas exceptionnel (le fameux "normalement, on ne devrait jamais arriver ici"))

- En Java, une erreur ne provoque pas un arrêt brutal du programme mais la création d'un objet *Exception* ou *Error*

Localisation du traitement exceptionnel

- Le traitement des événements exceptionnels se fait dans une zone spéciale appelée bloc “*catch*”

```
try {

        recuperationDesDevoirs.add(etudiantBrillant.rendreDevoir());

}

catch (MonChienAMangeMonOrdiException etudiant) {...

        System.err.println(“Chien du futur”};

}
```

Bloc try-catch

```
try {

        ...

}

catch (WTFException1 e) {...}

catch (WTFException2 e) {...}

catch (WTFException3 |WTFException4 e) {...}
```

- Multi-catch possible depuis JDK 7 mais les classes d'exceptions concernées ne doivent pas être dans la même "descendance" familiale
  - catch(NumberFormatException | Exception e){...} n'est pas compilable "Types in multi-catch must be disjoint"
- Importance de l'ordre des clauses "catch" - les plus générales à la fin

Dialecte

- **Lever** une exception : une exception est créée puis remontée à la méthode appelante
- **Traiter** une exception : l'exception est capturée et traitée dans la méthode courante
- **Remonter** une exception : l'erreur remonte vers la méthode appelante et n'est pas gérée dans la méthode courante
- **Exceptions contrôlées** : EOFException ou tout autre exception créée par le développeur
- **Exceptions non contrôlées** : OutOfBoundException, ...

Précisions pour les clauses “catch”

Les exceptions contrôlées spécifiées dans les clauses “catch” doivent être susceptibles d’être levées dans les instructions du bloc “try”

“Exception Blablabla never thrown in the corresponding try block”

Ce qui n’est absolument pas le cas pour les exceptions non contrôlées

Scénarios : exception levée dans un bloc try

Les instructions du bloc try suivant l'instruction responsable de la levée ne sont pas exécutées

- Si au moins une clause catch capture l'exception
  - la première clause catch appropriée est exécutée
  - le traitement reprend après le bloc try-catch
- Sinon
  - la méthode courante se termine immédiatement
  - l'exception est levée à la méthode appelante

Scénarios : aucune exception levée dans un bloc try

- le déroulement du bloc *try* s'effectue normalement et intégralement
- le programme se poursuit ensuite après les clauses *catch*

Scénarios : exception jamais traitée

- L’exception remonte jusqu’au *main*
- Le programme s’arrête immédiatement
- Le message associé à l’exception levée est affiché avec une description des méthodes traversées par l’exception
- Dans le cas d’un multi-thread, uniquement le thread concerné meurt

Que fait on dans un bloc catch ?

- Traduire l’exception levée en une autre exception de plus haut niveau dans le cadre d’une application multi-couches (cf IHM)
- Retourner une valeur particulière - exemple : retour de -1 lorsqu’on cherche l’indice d’une valeur dans un tableau
- Faire un traitement alternatif

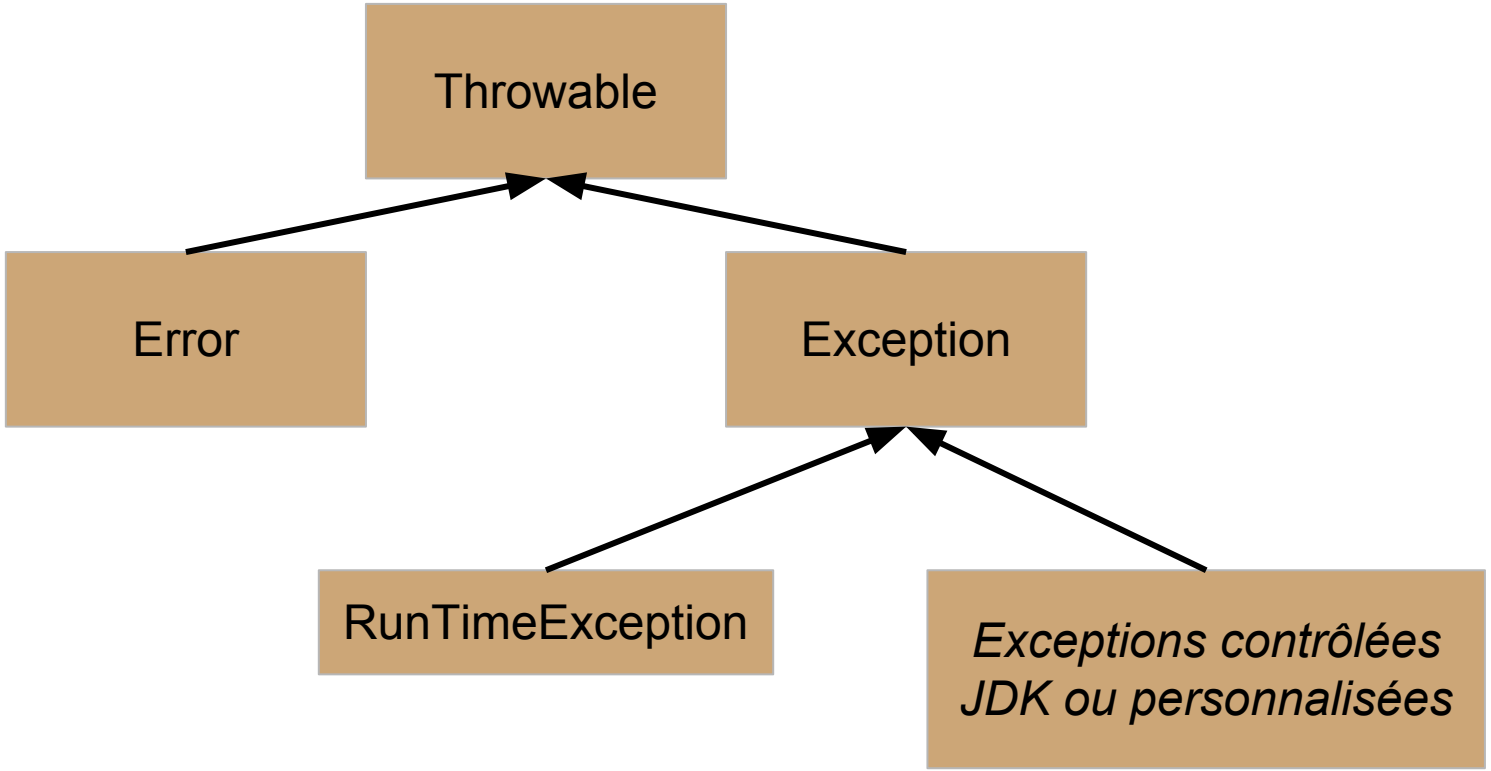
Traitement d’une exception

1. Traiter complètement l’exception dans la méthode concernée
2. Laisser remonter l’exception
3. Traiter partiellement l’exception, puis lever une nouvelle exception

Pourquoi laisser remonter une exception ?

- Plus une méthode est éloignée du *main* moins elle a une connaissance globale de l'application
- Elle peut donc laisser une méthode mieux placée pour traiter l'exception

Hiérarchie des exceptions / errors



Hiérarchie des exceptions / erros

- RunTimeException
  - NullPointerException
  - ClassCastException
  - ArrayIndexOutOfBoundsException
- Exceptions contrôlées de la JDK
  - FileNotFoundException
  - EOFException
  - ClassNotFoundException

Méthode susceptible de lever une exception

- Utilisation du mot clé *throws* dans le profil de la méthode
- *public Devoir rendreDevoir()* ***throws*** *MonChienAMangeMonOrdiException*

Créer ses exceptions

1. Créer une classe étendant la classe Exception
    - par convention le nom de la classe doit se terminer par Exception
2. Créer deux constructeurs
    - Un sans paramètre
    - Un avec une chaîne de caractères en paramètres
3. Les deux constructeurs appelleront systématiquement le constructeur de la classe mère avec l'instruction *super*

Lever une exception

- Utilisation de l’instruction *throw*
- throw new MonChienAMangeMonOrdiException();
- La méthode concernée doit être déclarée comme susceptible de lever cette même exception

Mise en application

Live démo - développement d’une application d’un prof collectant les devoirs de ses élèves

Savoir lire un message d'erreur……

- Le message commence par le message d'erreur associé à l'erreur qui a provoqué l'affichage
- On a ensuite la pile des méthodes traversées en attente de résultat depuis la méthode *main* pour arriver à l'erreur
- Les méthodes les plus récemment invoquées sont en premier, la méthode *main* est en dernier
    - A chaque étape on a le numéro de la ligne en cause