Exceptions

Université Française d'Égypte

Version O 0.9 – 17/9/12

Richard Grin

Fiabilité d'un logiciel

- Robustesse : fonctionne correctement, même en présence d'événements exceptionnels
- Correction : donne les résultats corrects lorsqu'il fonctionne normalement

R. Grin

Exceptions

2

Fiabilité en Java

- 2 mécanismes principaux facilitent une bonne fiabilité :
 - les exceptions pour la robustesse
 - les assertions pour la correction

R. Grin

Exceptions

3

Multi-catch

- Depuis le JDK 7 il est possible d'attraper plusieurs types d'exceptions dans un bloc `catch` :

```
try {
    ...
}
catch(Type1Exception | Type2Exception e) {
    ...
}
```

R. Grin

Fiabilité

4

Mécanisme de traitement des exceptions

Exemples de traitements dans un bloc `catch`

- Fixer le problème et réessayer le traitement qui a provoqué le passage au bloc `catch`
- Faire un traitement alternatif
- Faire un traitement partiel du problème et relancer (`throw`) la même exception, ou une autre
- Retourner (`return`) une valeur particulière
- Sortir de l'application avec `System.exit`

R. Grin

Exceptions

5

R. Grin

Exceptions

6

Laisser remonter une exception

- Plus une méthode est éloignée de la méthode `main` dans la pile d'exécution, moins elle a une vision globale de l'application
- Une méthode peut laisser remonter une exception si elle ne sait pas comment la traiter, en espérant qu'une méthode appelante en saura assez pour la traiter

R. Grin

Exceptions

7

Erreur fréquente à éviter

- **A NE PAS FAIRE !** *Pourquoi ?*
 - attraper une exception
 - afficher un message du type « Erreur ! » dans le bloc `catch`
- Le minimum à faire : appeler la méthode `printStackTrace`

R. Grin

Exceptions

8

Clause `finally`

R. Grin

Exceptions

9

Traitement général des exceptions

```
try {
    . . .
}
catch (ClasseException1 e) {
    . . .
}
catch (ClasseException2 e) {
    . . .
}
finally {
    . . .
}
```

Possible d'avoir
`try - finally` sans `catch`

R. Grin

Exceptions

10

`try` avec ressources

R. Grin

Fiabilité

11

`try` avec ressources

- Nouvelle syntaxe du JDK 7
- Facilite la fermeture automatique des ressources : flots, fichiers, connexion sur une base de données,...

R. Grin

Fiabilité

12

Exemple avec **try** avec ressources

```
try(
  InputStream in = new FileInputStream(f1);
  OutputStream out = new FileOutputStream(f2)){
  byte[] buf = new byte[8192];
  int n;
  while ((n = in.read(buf)) >= 0) {
    out.write(buf, 0, n);
  }
}
```

R. Grin

Fiabilité

13

Avantages

- Les ressources sont fermées à la sortie du bloc **try**, quoi qu'il arrive
- Le code est bien plus lisible (surtout si plusieurs ressources doivent être fermées) qu'avec un **try - finally** « normal »

R. Grin

Fiabilité

14

Interface `java.lang.AutoCloseable`

- Les classes qui peuvent être gérées par un **try** avec ressources doivent l'implémenter
- Contient la méthode `void close()`

R. Grin

Fiabilité

15

Exception « contrôlée » (1)

- Exception qui hérite de la classe **Exception** mais pas de **RuntimeException**
- Le compilateur vérifie que les méthodes utilisent correctement cette exception
- Toute méthode qui peut lancer une exception contrôlée, **doit** le déclarer dans sa déclaration

```
int m() throws TrucException {
  . . .
}
```

R. Grin

Exceptions

16

Exceptions « contrôlées » (2)

- Soit une méthode **m2** avec une en-tête « **throws TrucException** »
- Si une méthode **m1** fait un appel à **m2**
 - soit **m1** entoure l'appel de **m2** avec un bloc « **try - catch(TrucException)** »
 - soit **m1** déclare qu'elle peut lancer une **TrucException**

R. Grin

Exceptions

17

Quand lever une exception

- Le plus près possible de l'endroit du code à l'origine du problème
- Par exemple, si une exception est due à un paramètre qui n'a pas une valeur correcte, il faut lancer l'exception au début de la méthode

R. Grin

Fiabilité

18

Chaînage de **Throwable**

- Il est aussi fréquent d'avoir un constructeur qui prend un **Throwable** en paramètre
- Ainsi, une API peut transformer une exception de bas niveau en exception de plus haut niveau, tout en gardant la cause première de l'exception (obtenue par `getCause()`)

R. Grin

Exceptions

19

Exemple d'une API sur les factures

- `FactureException(String message, Throwable cause) {`
`super(message, cause);`
`}`
 - Appel du constructeur de **Exception**
- `try {`
`. . .`
`catch(IOException ex) {`
 - Où se trouve ce code ?`throw new FactureException(`
`"Facture de " + client + " pas trouvée",`
`ex);`
`}`

R. Grin

Exceptions

20

Choix des exceptions à lancer

R. Grin

Exceptions

21

Error

- Réservé aux erreurs qui surviennent dans le fonctionnement de la JVM
- Par exemple `OutOfMemoryError`
- Ne devrait jamais arriver
- Éviter de lancer une **Error**

R. Grin

Exceptions

22

RuntimeException

- Bon choix si le problème ne peut être résolu par une des méthodes appelantes
- Inutile alors d'alourdir le code de ces méthodes avec des `catch` ou des `throws`
- Dû à une erreur indépendante du code qui l'a lancée (passage d'un mauvais paramètre,...)

R. Grin

Exceptions

23

Traitement des exceptions non contrôlées

- Laisser remonter l'exception jusqu'au début de l'action lancée par l'utilisateur
- La classe qui attrape l'exception
 - l'enregistre comme événement inattendu (*logging*)
 - stoppe proprement l'action qui a provoqué le problème (remet les choses en place)
 - prévient l'utilisateur s'il le faut

R. Grin

Exceptions

24

Exceptions contrôlées

- Pour les cas peu fréquents, mais pas inattendus
- Correspondent à des scénarios envisagés par le développeur
- Pour les problèmes qui peuvent être résolus (au moins partiellement) par une des méthodes de la pile d'exécution
- Par exemple, si une étagère est pleine, le code qui a voulu ajouter un livre pourra choisir une autre étagère

R. Grin

Exceptions

25

Exceptions du JDK ou nouveau type d'exception ?

- Si une exception du JDK convient, il vaut mieux l'utiliser car ces exceptions sont bien connues des développeurs

R. Grin

Exceptions

26