A3P/IPO 2023/2024

Cours 7

Sommaire

1. Concept d'exception, et en Java
2. La hiérarchie et les 3 sortes
3. Traitement d'une exception (**`try`** / **`catch`**)
4. Non-traitement d'une exception (**`throws`**)
5. La clause **`finally`**
6. Lancement d'une exception (**`throw`**)
7. Création de nouvelles exceptions
8. Bonus : les assertions
9. La méthode **`main`** et la ligne de commande

1. & 2. Qu'est-ce qu'une exception ?

- **1. Conceptuellement :** Un évènement inhabituel survenant pendant le déroulement de votre programme, et susceptible de l'interrompre.

- **2. En Java :** Une instance (un objet) de la classe `Exception`
  ou d'une de ses sous-classes.

3. Plus de technique

- **3. Quand/comment est créé cet objet ?**

  - Quand la JVM détecte une erreur
    ou quand une méthode décide de lancer une exception après avoir testé une certaine condition.

  - Par exemple, si on regarde le code de la méthode `Integer.parseInt`, on verra un test vérifiant que chaque caractère de la `String` est bien un chiffre ; dès qu'elle trouve autre chose, cette méthode lance une `NumberFormatException`.

4. Avantages

- **4. Avantages attendus du système des exceptions**

➢ 4.1. séparer les instructions normales du traitement des erreurs

➢ 4.2. propager l'exception à la méthode appelante, etc.

➢ 4.3. différencier et regrouper des exceptions différentes grâce aux hiérarchies de classes (héritant de Exception)

5.1 & 5.2

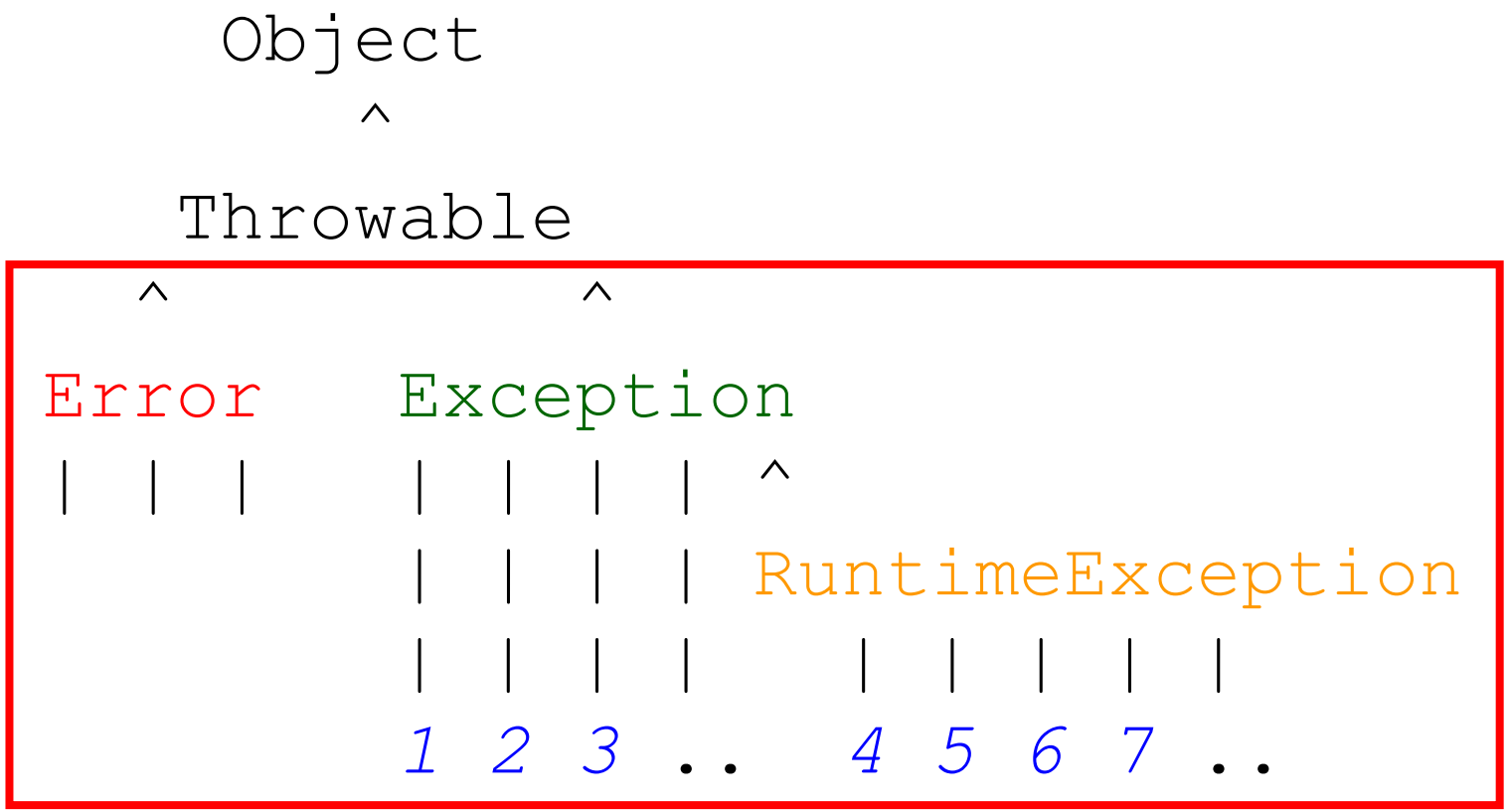
- **5.1. Error** est un type d'erreur système "grave"; on ne peut rien faire, donc :
inutile d'essayer de les attraper et de les traiter.
- (exemples: `AssertionEr`, `VirtualMachineEr`, `OutOfMemoryEr`, `StackOverflowEr`)

- **5.2. Exception** est un type d'erreur qui peut être traitée; **on a le choix** de dire qu'une méthode la lance **ou bien** de l'attraper et la traiter à l'intérieur.
- (*catch or specify*, checked exceptions, outside JVM)
(exemples: `CloneNotSupportedEx`, `AWTEx`, `IOEx`, `EOFEx`, `FileNotFoundEx`)

5.3 `RuntimeException`

- **5.4. RuntimeException** est un type particulier d'exception qui correspond, en général, à une erreur du programmeur
- on n'a donc pas obligation ni de la déclarer ni de l'attraper et traiter. Par contre, on a obligation de corriger son programme pour qu'elle n'arrive pas !
- (unchecked exceptions, inside JVM) (exemples: `ArithmeticEx`, `ClassCastEx`, `NegativeArraySizeEx`, `NullPointerEx`, `IllegalArgumentEx`, `NumberFormatEx`, `IndexOutOfBoundsEx`)

5.4. Hiérarchie dans le JDK



- *Les classes 1 2 3 4 5 ... correspondent aux exceptions déjà définies dans les nombreuses classes du JDK.*

6. Que faire des exceptions lancées par le JDK ?

- // 6.1. Exemple

```
try {
   i1; i2; i3; i4; i5;
}
catch ( final UneClasseException pObjetException ) {
   traitement
}
```

- S'il y a par exemple **5 instructions** dans le `try`,
  et que `i2` lance l'exception attendue,
  le programme sort immédiatement du bloc `try`
  (**`i3 i4 i5`** ne seront jamais exécutées),
  et exécute le traitement prévu dans le bloc `catch`
  (correspondant au type de l'exception survenue)
- (voir méthodes `getStackTrace()`, `printStackTrace()`, `getMessage()`, `getLocalizedMessage()`, `toString()`)

6.2 & 6.3 & 6.4 Types d'exceptions

- **6.2.** S'il y a **plusieurs exceptions possibles**, on peut mettre successivement plusieurs blocs `catch` après un seul bloc `try`. Attention ! l'ordre est important : du particulier vers le plus général
  Java 7 : `( final TypEx1 | TypEx2 pObjEx )`
- **6.3.** Si un traitement est commun à plusieurs exceptions héritant d'une même classe, écrire :
  `catch ( final ClasseMereException pObjetException )`
- **6.4.** On sait quelle classe d'exception peut survenir (et pourquoi) en regardant la javadoc :
  Throws: `UneClasseException` when …

6.5 & 6.6 `throws`

- **6.5.** Si une méthode appelée lance une exception dont la classe **n'hérite pas de `RuntimeException`**,
  soit il faut entourer l'appel d'un `try/catch`,
  soit il faut déclarer que cette méthode lance l'exception
- ==> **`throws`** `UneClasseException`
  à la fin de la signature
  (plusieurs possibles, séparées par des virgules)

- **6.6.** On peut traiter partiellement, puis relancer l'exception dans le `catch` : **`throw`** `pObjetException;`
- ==> **`throws`** `UneClasseException`
  (malgré le `try/catch( UneClasseException )` !)

6.7 `finally`

- **6.7.** Si un traitement doit être exécuté qu'une exception soit lancée ou pas, quelle que soit la manière de sortir de la méthode, ajouter après le `try/catch` un bloc
- ```
  finally {
    traitement
  }
  ```
- (si fin normale du `try` ou du `catch`, ou `return` dans `try` ou `catch`, ou exception imprévue)
- Classe de démonstration `FinallyDemo` (à comprendre en détail absolument !)
  ```

7. Lancer volontairement des exceptions du JDK

- **7.1.** Exemple :

```
if (mauvais paramètre)
   throw new UneClasseException("...");
```

- ==> `throws UneClasseException`

- **7.2.** Ne pas rattraper une exception dans la même méthode que celle où on l'a lancée :

```
try { if (! condition)
            throw new UneClasseException();
      i1; }
catch ( final UneClasseException pE ) { i2; }
```

- `if (condition) { i1; } else { i2; }`

8. Créer ses propres exceptions

8.1. Créer une classe qui `extends Exception` ou une de ses sous-classes

8.2. Redéfinir éventuellement `getMessage()`
(visible aussi dans `toString()`)

8.3. Ajouter éventuellement des attributs (donc constructeur) pour stocker de l'information, voire des méthodes (accesseurs, ...)

9. Bonus : Les assertions

- **9.1.** Programmation par contrat, sécurité : pré-condition, post-condition, pas d'invariant

- **9.2.** `assert condition : message;`
- équivalent à `if (! condition) throw new AssertionError(message);`
- sauf que le compilateur peut retirer automatiquement les assertions « en production »

- ***9.3.*** *Pour ne pas vérifier les assertions, supprimer* `-ea` *à la compilation (*`bluej.defs` *ou* `javac`*)*

10. Ligne de commande

- **Lancer un programme java depuis la ligne de commande :**

- `java MaClasse`

  ou bien

- `java MaClasse` $\texttt{mot}_0$ `...` $\texttt{mot}_{N-1}$

- **Différence avec le C :**
  `exécutable` $\texttt{mot}_1$ … $\texttt{mot}_N$

11. Méthode `main`

- **Quelle méthode de MaClasse est alors appelée ?**
- **La méthode qui a une signature précise :**

- L'appel provient de l'extérieur de la classe
- L'appel se fait directement sur la classe
- Elle ne retourne rien ==> procédure
- Son nom doit être imposé
- On veut pouvoir passer des arguments sur la ligne de commande (0 ou 1 ou plusieurs)

- `public static void main( final String[] pArgs )`

12. Développement sans BlueJ

- **`javac MaClasse.java`**

ou

- **`javac *.java`**

et aussi

- **`javadoc ...`**