Série C : Exceptions

Table des matières

Série C : Exceptions.....	1
Exercice C.1: Extraits du tutoriel Java de Oracle.....	1
Exercice C.2: Apprendre à connaître les exceptions.....	4
Exercice C.3: Fraction.....	7
Exercice C.4: FractionList.....	7
Exercice C.5: Cryptor.....	7

Exercice C.1: Extraits du tutoriel Java de Oracle

Question 1:

Is the following code legal?

```
try {  
    ...  
} finally {  
    ...  
}
```

Question 2:

What exception types can be caught by the following handler?

```
catch (Exception e) {  
  
}
```

What is wrong with using this type of exception handler?

Question 3:

Is there anything wrong with this exception handler as written? Will this code compile?

```
try {  
  
} catch (Exception e) {  
  
} catch (ArithmeticException a) {  
  
}
```

Question 4:

Match each situation in the first list with an item in the second list.

- a) `ArrayList<String> sl; String s = sl.get(0) ;`
 - b) The JVM starts running your program, but the JVM can't find the Java platform classes. (The Java platform classes reside in `classes.zip` or `rt.jar`.)
 - c) A program is reading a stream and reaches the end of stream marker.
 - d) Before closing the stream and after reaching the end of stream marker, a program tries to read the stream again.
-
- 1. error
 - 2. checked exception
 - 3. compile error
 - 4. no exception

Exercice C.2: Apprendre à connaître les exceptions

Expérimentez avec les exceptions, et notez chaque fois vos constatations, les messages obtenus et vos conclusions :

- Créez une classe avec une méthode lançant une exception obligatoire (**checked exception**), p.ex. une `IOException`.
- Traitez les exceptions uniquement par **throws** sans utiliser **try catch**.
- Appelez la méthode critique par une autre méthode dans la même classe
- Appelez la méthode critique par une autre méthode dans une autre classe.
- Ajoutez une troisième classe appelant la méthode critique de la deuxième classe. Faites l'appel dans la méthode `main(...)`.
- Que remarquez-vous si la classe 'supérieure' (dans la hiérarchie des appels) ne traite pas l'exception, mais effectue aussi un **throws** ?
- Traitez les exceptions par **try catch** aux différents niveaux. Enlevez les instruction **throws** superflues. Faites afficher un '**stack trace**' dans le cas d'une erreur.
- Refaites le même exercice avec une exception du type `RuntimeException`. (p.ex. `ArithmeticException`).
- Entourez l'instruction critique et les appels des méthodes critiques par des affichages p.ex : `"callCriticalMethod2: before call"`, `"callCriticalMethod2: after call"`. Ajoutez un affichage à la fin des méthodes. Faites des essais en plaçant **try-catch** aux différents niveaux (dans les différentes méthodes). Que constatez-vous ?
- Ajoutez des blocs `try ... finally` à toutes les méthodes, désactivez les blocs `catch` et observez la suite des traitements.
- Est-ce que les deux blocs suivants réagissent de la même façon ? Motivez votre réponse !

```
(A)    try {
        <bloc d'instructions 1>
    }
    catch (...) {
        <bloc d'instructions 2>
    }
    finally {
        <bloc d'instructions 3>
    }
}
```

```
(B)    try {
        <bloc d'instructions 1>
    }
    catch (...) {
        <bloc d'instructions 2>
    }
    <bloc d'instructions 3>
```

Exercice C.3: Fraction

Même si dans le cours, nous avons défini une exception du type **DivisionByZeroException**, pour le cas où le dénominateur est zéro, ce n'est pas la meilleure manière de procéder, car dans ce cas c'est trop tard : la fraction a déjà été initialisée avec la valeur indésirable.

Mieux vaut éviter ce cas dès le départ :

1. Reprenez votre classe **Fraction** et lancez une exception du type **IllegalArgumentException** dès qu'on essaie d'affecter une valeur nulle au dénominateur. Enlevez tous les autres tests sur une valeur nulle du dénominateur. Affichez un message d'erreur informatif.
2. Testez l'exception, en Unimozer.
3. Développez un programme (en Unimozer ou en NetBeans) pour tester votre classe. Interceptez l'erreur et affichez son message à l'écran.
4. Après avoir ajouté une méthode **divideBy(Fraction)** à votre classe, un autre cas de figure se présente : Définissez un nouveau type d'exception **DivisionByZeroException** (dérivé de **ArithmeticException**) que vous provoquez dans ce cas.
5. Testez les deux exceptions, en Unimozer, puis développez une application pour tester toutes les opérations et les exceptions qu'elles peuvent provoquer. Interceptez les deux exceptions avec deux blocs **try ... catch**.

Exercice C.4: FractionList

Copiez l'exercice précédent et élargissez la copie comme suit :

L'application **FractionList** sait produire au hasard des fractions avec des valeurs aléatoires positives ou négatives (p.ex. entre -5 et 5) comme numérateur et comme dénominateur. La liste des fractions est à représenter à l'écran.

Si le générateur de nombres aléatoires produit un dénominateur zéro, alors la fraction n'est pas produite et elle n'est pas ajoutée à la liste. Utilisez **try-catch(-finally)** et pas de structure **if**. A la fin, le nombre de fractions dans la liste doit correspondre au nombre demandé.

Affichez aussi le nombre de fractions incorrectes survenues.

Exercice C.5: Cryptor

Reprenez la classe **Cryptor**:

1. Déterminez les cas où il peut s'avérer nécessaire de produire une exception. Testez votre classe pour ce cas. Quel type d'exception est approprié ?
2. Définissez un type d'exception individualisé pour ce type d'erreur.
3. Testez les exceptions avec une classe qui intercepte l'exception et réagit en conséquence.
4. Définissez un message par défaut pour le nouveau type d'exception.