# TD7 : Exceptions et Itérateurs

POO : Programmation Orientée Objet

Première année

## Rappel de cours sur les Exceptions

*The code in the* `finally` *block is executed under all circumstances, regardless of whether an exception occurs in the* `try` *block or is caught. Consider three possible cases :*

*If no exception arises in the* `try` *block,* `finalStatements` *is executed, and the next statement after the* `try` *statement is executed.*

*If one of the statements causes an exception in the* `try` *block that is caught in a* `catch` *block, the other statements in the* `try` *block are skipped, the* `catch` *block is executed, and the* `finally` *clause is executed. If the* `catch` *block does not rethrow an exception, the next statement after the* `try` *statement is executed. If it does, the exception is passed to the caller of this method.*

*If one of the statements causes an exception that is not caught in any* `catch` *block, the other statements in the* `try` *block are skipped, the* `finally` *clause is executed, and the exception is passed to the caller of this method.*

*The* `finally` *block executes even if there is a* `return` *statement prior to reaching the* `finally` *block.*

```
try {
  statements;
}
catch (TheException ex) {
  handling ex;
}
finally {
  finalStatements;
}
```

★ **Exercice 1:** On considère l'interface `Stack` et la classe `Value` suivantes :

```
package exo1;

public interface Stack {

    public boolean empty() ;

    public void push(Value v) ;

    public Value pop() throws EmptyStackException;

    public Value peek() throws EmptyStackException;
}
```

```
package exo1;

public class Value {
    private String name;
    private int value;

    public Value(String n, int v) {
        this.name = n;
        this.value = v;
    }
    public String toString() {
        return "<"+this.name+";"+this.value+">";
    }
}
```

- La classe `Value` décrit un couple <nom de la valeur, valeur entière stockée>,
- L'interface `Stack` décrit l'interface standard d'une pile de valeurs :
  - la méthode `boolean empty()` indique si la pile est vide ou non,
  - la méthode `void push(Value v)` empile une nouvelle valeur,
  - la méthode `Value pop()` retourne et dépile une valeur empilée,
  - la méthode `Value peek()` retourne une valeur empilée, sans la dépiler.
- Les méthodes `Value pop()` et `Value peek()` ne peuvent pas retourner de valeur quand elles sont appelées sur une pile vide. Dans ce cas, une exception `EmptyStackException` doit être levée.

▷ **Question 1:** Écrire la classe `EmptyStackException` héritant de la classe `Exception`.

▷ **Question 2:** Écrire une classe `LIFOStack` implémentant l'interface `Stack` qui réalise une pile du type *Last In, First Out* (la dernière valeur empilée est la première valeur à être dépilée). Pensez à lever une exception quand c'est nécessaire.

**Indication :** Comme structure interne de votre pile, vous utiliserez par exemple un champ particulier qui sera de classe `ArrayList` (il faut importer `java.util.ArrayList` pour pouvoir utiliser cette classe). Les méthodes dont vous aurez besoin sont les suivantes (rappel) :

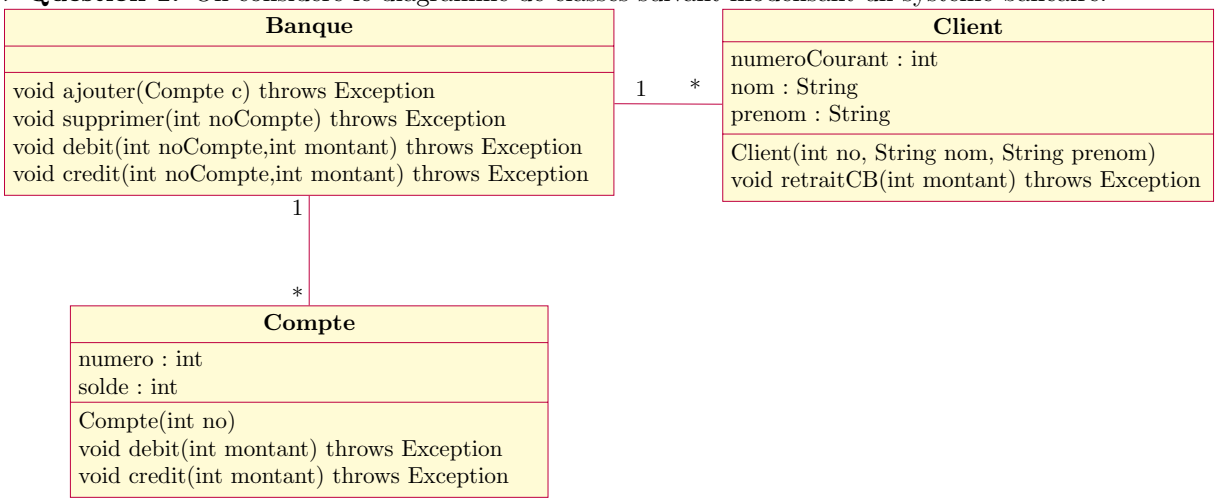
- connaître la taille de la collection : `coll.size()`
- ajouter un élément *elm* : `coll.add(`*elm*`)`

– lire l'élément à la position *pos* : `coll.get(idx)`
– retirer l'élément à la position *pos* : `coll.remove(idx)`

▷ **Question 3:** Écrire une classe `TestStack` qui crée une instance de la classe `LIFOStack`, qui empile une valeur et essaye de la dépiler. N'oubliez pas de capturer les exceptions qui peuvent être levées.

## ★ Exercice 2:

▷ **Question 1:** On considère le diagramme de classes suivant modélisant un système bancaire.




Implanter ce diagramme sachant que l'appel à la méthode `retraitCB()` déclenche un appel à la méthode `debit()` de la classe `Banque` qui elle même fait appel à la méthode `debit()` de la classe `Compte`. Dans le cas où le solde ne permettrait pas d'effectuer le retrait d'argent, une exception du type `PasAssezArgentException` doit être levée et propagée.

## ★ Exercice 3:

▷ **Question 1:** On considère que l'exécution `instruction2` lève une exception dans le bloc `try-catch`

```
try {
  instruction1;
  instruction2;
  instruction3;
} catch (Exception1 ex1) {
  // ...
} catch (Exception2 ex2) {
  // ...
}
instruction4;
```

(a) L'instruction `instruction3` sera-t-elle exécutée ?
(b) Si l'exception n'est pas attrapée, l'instruction `instruction4` sera-t-elle exécutée ?
(c) Si l'exception est attrapée dans un des blocs `catch`, l'instruction `instruction4` sera-t-elle exécutée ?
(d) Si l'exception est passée à la méthode appelante, l'instruction `instruction4` sera-t-elle exécutée ?

▷ **Question 2:** On considère que l'exécution `instruction2` lève une exception dans le bloc `try-catch`

```
try {
  instruction1;
  instruction2;
  instruction3;
} catch (Exception1 ex1) {
  // ...
} catch (Exception2 ex2) {
  // ...
} catch (Exception3 ex3) {
  throw ex3;
} finally {
  instruction4;
}
instruction5;
```

(a) L'instruction `instruction5` sera-t-elle exécutée si l'exception n'est pas attrapée ?
(b) Si l'exception levée est de type `Exception3`, l'instruction `instruction4` sera-t-elle exécutée ? l'instruction `instruction5` sera-t-elle exécutée ?

★ **Exercice 4:** On considère l'interface Java suivante :

```
package exo4;

public interface IterateurTabInt {
    public abstract int suivant();
    public abstract int indiceDuSuivant();
    public abstract boolean aUnSuivant();
}
```

On veut écrire une classe d'itérateurs qui parcourent uniquement les cases contenant un nombre pair dans un tableau quelconque d'entiers. Cette classe se nommera `IterateursDesPairs` et implantera l'interface `IterateurTabInt`.

Le sens du parcours est celui des indices croissants. La référence du tableau à parcourir est un attribut noté `tab` et est transmise lors de l'instanciation de l'itérateur. L'indice du tableau dont la valeur est retournée par la méthode `suivant()` est noté `pos`. Cet indice est mis à jour la première fois à l'instanciation, et, les fois suivantes, lors des appels à la méthode `suivant()`. Il n'y a plus de suivant quand cet indice est égal à la longueur du tableau.

Les premières déclarations de la classe sont donc :

```
    private int[] tab;
    private int pos;

    public IterateurDesPairs(int[] tab) {
        this.tab = tab;

        // à compléter
    }
```

▷ **Question 1:** Écrivez le code de la classe `IterateurDesPairs`.

▷ **Question 2:** Complétez le corps du constructeur de la classe `Test` pour avoir à l'exécution les affichages suivants (format : indice de la valeur entière paire → entier pair )

```
test1 : 1->2,2->6,4->8,7->12,8->14,
test2 : 0->2,
test3 :
```

```
package exo4;

public class TestIterateur {
    public TestIterateur(String message, int[] t) {
        IterateurDesPairs i = new IterateurDesPairs(t);
        System.out.print(message);


/******************/
/* A VOUS DE JOUER */
/******************/

    }

    public static void main(String[] args) {
        new TestIterateur("test1 : ", new int[] {1,2,6,7,8,9,11,12,14,13,5});
        new TestIterateur("test2 : ", new int[] {2,79});
        new TestIterateur("test3 : ", new int[] {});
    }
}
```