Labo POO avec Java *TP2*

Piles, exceptions

Semestre d'hiver 2017/2018 – filière ITI

Objectifs

Ce laboratoire répond aux objectifs suivants:

1) Mise en œuvre d'une pile
2) Introduction aux modificateurs d'accès `private` et `public`
3) Levée, capture et traitement d'une exception

Reddition

Le TP est à terminer pour le 17 octobre 2016 (2 séances)

I. Descriptif

On aborde la mise en œuvre d'objets de stockage, l'écriture d'une classe, les notions de modificateurs d'accès (`private`, `public`) et la gestion des exceptions.

II. Enoncé

Il s'agit d'implémenter une classe générique `Pile` en utilisant la classe générique `ArrayList<E>` du paquetage `java.util` (voir l'API Java pour la description) afin de stocker les valeurs empilées.

La classe `Pile` comprendra le champs privé `container` du type `ArrayList` et les méthodes publiques `empiler()`, `depiler()`, `sommet()` et `estVide()` ainsi qu'un constructeur sans paramètres permettant d'initialiser le champ `container`. En réalité, l'implémentation correcte de la classe générique `Pile` devrait se faire par héritage. Le programme à réaliser (`PileTest.java`) comprendra la classe `Pile` et une classe publique `PileTest` constituée uniquement de la méthode `main` pour effectuer des tests de votre implémentation de pile.

La méthode `depiler()` lèvera une exception avec le message d'erreur "`Pile vide`" si elle est appelée alors que la pile est vide. La méthode `main()` de `PileTest` capturera et traitera les exceptions levées par `depiler()`.

La classe `Pile` sera réutilisée au prochain travail pratique pour implémenter une calculatrice qui évalue des expressions postfixes. Elle sera modifiée pour en faire une classe dérivée. La gestion des exceptions sera étendue aux erreurs de saisie au clavier.

III. Implémentations

Etape A : architecture du programme

Il s'agit juste d'implémenter la classe `Pile` et de la tester, la gestion des erreurs étant laissée pour l'étape suivante.

Programme du TP2

```
import java.util.ArrayList;

class Pile<T> {
   //champ à déclarer

   //constructeur sans paramètre
   public Pile() {
       //à completer
   }

   //autres méthodes à définir
}

public class PileTest {
   public static void main(String[] args) {
       Pile<Double> maPile = new Pile<Double>();
       // à completer
   }
}
```

A cette étape, ignorez la partie concernant les exceptions. Veillez à ce que le champ `container` ait le modificateur d'accès `private` et les méthodes le modificateur d'accès `public`.

QUESTIONS

1. Que signifient les modificateurs d'accès `private` et `public` ?
2. Peut-on aussi déclarer `public` la classe `Pile` ?
3. Que veut dire l'écriture `<T>` dans la définition d'une classe?
4. Qu'est-ce qu'un constructeur et comment est-il appelé ?

Etape B : lever une exception

Pour lever une exception, utiliser l'instruction `throw`, comme par exemple :

```
throw new RuntimeException("Pile vide!");
```

Les exceptions sont des objets qui appartiennent à la hiérarchie `Exception`, dans laquelle on trouve la classe `RuntimeException`. Cette dernière est utilisée couramment pour générer des exceptions qui peuvent être ignorées par le programme. On parle alors d'exception non-contrôlée. Les autres types d'exceptions, dites contrôlées, doivent être obligatoirement:

- soit récupérées par le programmeur,
- soit propagées de manière explicite par le programmeur.

Nous reviendrons plus précisément par la suite sur les exceptions.

A noter qu'il est possible de créer un nouveau type d'exception au moyen d'une classe:

```
class PileVideException extends RuntimeException {
   public PileVideException() {
      // Invocation du superconstructeur
      // c.-à-d. le constructeur de la classe parente
      super("Pile vide!");
   }
}
```

Une telle exception est levée avec l'instruction :

```
throw new PileVideException();
```

Quand une exception est susceptible d'être levée dans un bloc d'instructions, on entoure celui-ci par une clause :

```
try {
  //bloc d'instructions testé
} catch(ClasseException1 e1) {
  //traitement de l'exception e1
} catch(ClasseException2 e2) {
  //traitement de l'exception e2
} catch(ClasseException3 e3) {
  //traitement de l'exception e3
}
```

Il est déconseillé de laisser le bloc de traitement d'une exception vide. Il faut au moins envoyer à l'objet exception concerné, le message :

```
e.printStackTrace();
```

Rajouter dans la méthode `depiler()`, la levée d'une exception accompagnée du texte `"Pile vide!"`. Dans la méthode `main()`, capturez et traitez les exceptions potentiellement levées par des appels à `depiler()`.

Une méthode propage une exception si son entête contient la déclaration :

```
throws ClasseException
```

Par exemple, si la méthode `main()` doit propager une exception d'entrée-sortie, son entête est :

```
public static void main(String[] args) throws IOException
```

QUESTIONS

1. Que fait le message `printStackTrace()` sur une exception ?
2. Pourquoi capturer des exceptions autres que celles de la classe `Exception` ?