Programmation orientée objet

Série 7

Joel Cavat / 2020

7.1 Exercice *(Pile d'entiers)*

Ecrivez une classe `IntegerArrayStack` qui hérite de l'interface `IntegerStack`. Utilisez une `ArrayList` pour simuler le comporterment de votre pile. Retournez l'exception (unchecked) `EmptyIntegerStack` lorsque c'est nécessaire.

Fournissez **le maximum** de méthodes concrètes (au moins quatre) dans l'interface.

```java
import java.util.Optional;

public interface IntegerStack {
  int pop(); // récupère et enlève le premier élément
  Optional<Integer> popOption();
  void ifHeadIsPresent(Consumer<Integer> consumer);
  void push(int i); // ajoute un élément
  int size(); // retourne la taille
  int peek(); // lit la valeur de tête en la conservant
  Optional<Integer> peekOption();
  boolean isEmpty();
}
```

Extrait d'utilisation:

```java
IntegerStack stack = new IntegerArrayStack();
stack.push(1);
stack.push(2);
stack.push(3);
stack.ifHeadIsPresent( v -> System.out.println("head: " + v) ); // head: 3
```

7.2 Exercice *(`ListInt` - `LinkedListInt`)*

Reprenez l'exercice déjà réalisé précédemment et ajoutez une nouvelle classe `LinkedListInt` qui implémente `ListInt`. Concrétisez cette liste en réalisant vous-même une liste chaînée.

L'illustration de la figure 1 représente l'état de la liste après avoir exécuté le code ci-dessous.

```
1 ListInt list = new LinkedListInt();
2 list.insert(12);
3 list.insertAll(99, 37);
```

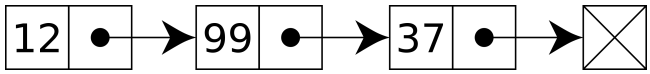

Figure 1: Liste chaînée - source: wikipedia.org

7.3 Exercice

Réalisez une méthode `test` qui prend en argument un `Supplier<Integer>` et retourne un `Optional<Integer>`. Cette méthode a pour but de déléguer une exécution et de retourner un optionel vide en cas de problème.

- si l'execution du supplier n'a pas levé d'exception, la valeur est encapsulée dans un Optional (lignes 1 et 2)
- retourne un Optional vide sinon (lignes 3 à 7)

Exemple d'utilisation:

```
1 test( () -> 3 ) // retourne Optional contenant 3
2 test( () -> 3 + 4 ) // retourne Optional contenant 7
3 test( () -> 3 / 0 ) // retourne Optional vide
4 test( () -> null ) // retourne Optional vide
5 test( () -> {
6   throw new RuntimeException()
7 }) // retourne un Optional vide
```

Remarques

1. Le `Supplier` est utilisé pour délégué une opération. Si la méthode `test` prenait un `int` au lieu d'un `Supplier<Intege>`, le problème surviendrait avant d'appeler la méthode, c'est-à-dire, lors de l'évaluation des arguments !
2. Notre méthode `test` fonctionne pour un `Supplier` et une éventuelle `RuntimeException` (ou héritant de). Trouvez un moyen, si le temps vous le permet, de le faire pour n'importe quelle exception héritant de `Exception`.

7.4 Exercice *(Account)*

WIP