Programmation Objet Avancée - TP8

—o000o——o000o—

Exceptions et fichiers

Le but de ce TP est de manipuler et gérer les exceptions pour la création d'une pile (stack) et la gestion d'un système bancaire à travers un programme java. Une pile est une structure de données avec la politique LIFO (Last In First Out) ; le dernier élément inséré sera le premier à en sortir. De plus, vous devez apprendre comment gérer les fichiers à travers un programme java.

Attention Veuillez noter que le sujet **est volontairement moins directif** que les précédents. Vous avez désormais du recul en programmation mais en aussi en modélisation objet du fait du projet. Mettez cette expérience à profit pour faire des choix judicieux !

1 Stack et Value

Considérons l'interface `Stack` et la classe `Value` suivantes :

```java
public interface Stack {
    boolean empty();

    void push(Value value);

    Value pop() throws EmptyStackException;

    Value peek() throws EmptyStackException;
}
```

java/Stack.java

```java
public class Value {
    private final String name;
    private final int value;

    public Value(String name, int value) {
        this.name = name;
        this.value = value;
    }

    @Override
    public String toString() {
        return "<" + this.name + ";" + this.value + ">";
    }
}
```

java/Value.java

La classe `Value` décrit un couple `<nom de la valeur, valeur entière stockée>`.
L'interface `Stack` décrit l'interface standard d'une pile :

- La méthode `boolean empty()` indique si la pile est vide ou non
- La méthode `void push(Value value)` 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

1. Écrire la classe `EmptyStackException` héritant de la classe `Exception`
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 cela est nécessaire.

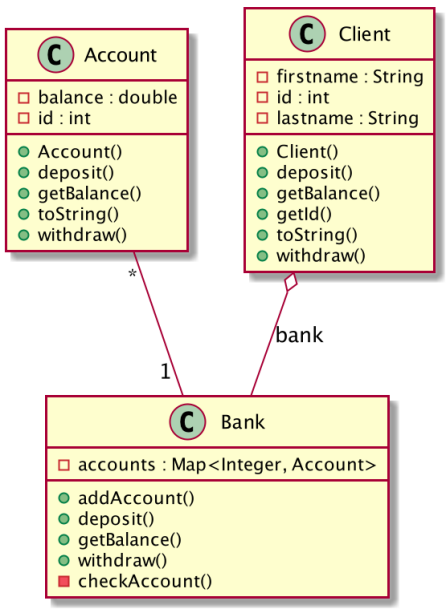
   Indication : La structure interne de votre pile se basera sur une `ArrayList`. Les méthodes dont vous aurez besoin sont les suivantes :

   - Connaître la taille de la collection : `size()`
   - Ajouter un élément : `add(E e)`
   - Retourner l'élément à la position "index" : `get(int index)`
   - Supprimer l'élément à la position "index" : `remove(int index)`
3. Écrire une classe de test. N'oubliez pas de capturer les exceptions.

2 Système bancaire

Nous considérons le diagramme de classes suivant modélisant un système bancaire.

Implémentez ce diagramme sachant que l'appel à la méthode `withdraw(double amount)` déclenche un appel à la méthode `withdraw(double amount)` de la classe `Bank` qui elle même fait appel à la méthode `withdraw(double amount)` de la classe `Account`. Dans le cas où le solde ne permettrait pas d'effectuer le retrait d'argent, une exception du type `InsufficientBalanceException` doit être levée et propagée.



⋆⋆ **Pour aller plus loin**, gérez également les exceptions `AccountNotFoundException` et `AccountArgumentException` (ex: le paramètre `amount <= 0`)

3 Parlons exceptions

[Question 1] Considérons que l'exécution `instruction2` lève une exception dans le bloc `try-catch` :

```
try {
    instruction1;
    instruction2;
    instruction3;
} catch (Exception1 e1) {
// ...
} catch (Exception2 e2) {
// ...
}

instruction4;
```

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

[Question 2] Considérons que l'exécution `instruction2` lève une exception dans le bloc `try-catch` :

```
try {
    instruction1;
    instruction2;
    instruction3;
} catch (Exception1 e1) {
// ...
} catch (Exception2 e2) {
// ...
} catch (Exception3 e3) {
    throw e3;
} finally {
    instruction4;
}

instruction5;
```

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

4 Les fichiers

4.1 API java.nio.file.*

Créez une classe `FileHelper` contenant la méthode suivante :

```
public static void replace(String src, String dst,
String searchString, String replacement)
```

Celle-ci prend en paramètres d'entrée 4 chaînes de caractères représentant respectivement les chemins des fichiers source et destination, la chaîne de caractères à modifier et sa valeur de remplacement.

Fichier original avec le mot "Imagine"	Fichier modifié avec le mot "Dream"
Imagine there's no heaven	**Dream** there's no heaven
It's easy if you try	It's easy if you try
No hell below us	No hell below us
Above us only sky	Above us only sky
Imagine all the people	**Dream** all the people
Living for today...	Living for today...

Pour cela, vous utiliserez les API suivantes :

- `java.nio.file.Paths`
- `java.nio.file.Path`
- `java.nio.file.Files`

4.2 Salle de classe et statistiques

On se propose d'obtenir des statistiques sur les élèves présents dans les salles de classe. Pour cela, chaque fichier (le nom du fichier correspond au numéro de la salle) contenu dans le répertoire "classroom" contient une liste d'élèves au format CSV (Comma-separated values).

Arborescence :

```
.
            classroom
                  E331
                  E335
```

Exemple de contenu d'un fichier :

```
1,Regine,Chassagne,44,Regine@arcadefire.com,Arcade Fire,Femme
2,Thom,York,52,thom@radiohead.com,Radiohead,Homme
```

Les statistiques devront être affichées sous cette forme :

```
E331 :: numberWomen=1, numberMen=1, total=2, avg=48.0
```

Implémentez les classes `Classroom` et `Student` et effectuez les modifications nécessaires dans la classe `FileHelper` pour répondre à cette problématique.

Nous nous intéressons désormais à sérialiser les objets.

4.3 Sérialisation

Sérialiser un objet consiste à le convertir en un tableau d'octets, que l'on peut ensuite écrire dans un fichier, envoyer sur un réseau au travers d'une socket, ... Pour cela, il suffit de passer tout objet qui implémente l'interface `Serializable` à une instance de `ObjectOutputStream` (et inversement, pour

désérialiser, vous devez utiliser `ObjectInputStream`).

Complétez la classe `FileHelper` pour lui ajouter 2 nouvelles méthodes :

- `void writeObject(String file, Object object)`
- `Object readObject(String file)`