R6.A.06

TP3 :

Exercice 1 : Anti-pattern Spaghetti code

L'énoncé de cet exercice est extrait de l'ouvrage :

W.J.Brown & al., Anti-patterns refactoring Software, Architectures and Projects in Crisis, Wiley.

Vous trouverez sur la zone moodle :

- le chapitre Software development AntiPattern avec la description de cet anti-patten
- le fragment de code du chapitre (la classe Showcase)

A faire dans un compte rendu détaillé de TP :

1. Lire la description de l'anti-pattern (jusqu'à known exceptions et l'exemple)
2. Identifier les principaux symptômes du fragment de code (à rédiger)
3. Lire la partie *refactored solution*
4. Essayer au mieux de suivre les conseils (page 3) pour découper et rende plus lisible le code de cette classe (rédiger les étapes).

Déposer votre compte-rendu et le code obtenu.

Exercice 2 :

Ces exercices utilisent des recommandations sur l'écriture de programmes Java.

Le livre « Effective Java » de Joshua Bloch a été conçu pour nous aider à utiliser le plus efficacement possible le langage de programmation Java et ses bibliothèques fondamentales. Il se compose de 68 items, chacun véhiculant une règle (pratique bénéfique).

2.1 Trouver le bug ?

item 46 : Prefer for-each loops to traditional for loops

```
// Can you spot the bug ?
enum Suit { CLUB,DIAMOND,HEART, SPADE}
enum Rank { ACE, DEUCE, THREE, FOUR, FIVE,SIX, SEVEN, EIGHT,NINE, TEN, JACK,
QUEEN, KING}
...
Collection <Suit> suits = Arrays.asList(Suit.values());
Collection <Rank> ranks = Arrays.asList(Rank.values());

List<Card> deck = new ArrayList<Card>();
for (Iterator<Suit> i = suits.iterator(); i.hasNext();)
    for (Iterator<Rank> j = ranks.iterator(); j.hasNext();)
        deck.add(new Card(i.next(), j.next()));


// Same bug, different symptom!
Enum Face { ONE, TWO, THREE, FOUR, FIVE,SIX}
...
Collection<Face> faces = Arrays.asList(Face.values());

for (Iterator<Face> i = faces.iterator(); i.hasNext();)
     for (Iterator<Face> j = faces.iterator(); j.hasNext();)
         System.out.println(i.next()+ " "+ j.next());
```

A faire :

Expliquer le bug. Pour vous aider vous pouvez écrire une classe Card et voir ce que fait ce code.

Ré-écrire le code en suivant le conseil « item 46 ».

2.2 Tableau ou collections vides

Item 43 : Return empty arrays or collections, not nulls

On écrit souvent des méthodes qui ressemblent à :

```
private final List<Cheese> cheesInStock= … ;
/**
* @return an array containing all of the cheeses in the shop,
* or null if no cheeses are available for purchase.
*/
public Cheese[] getCheeses() {
   if (cheesesInStock.size() == 0)
       return null;
...
}
```

Ensuite, cela demande au client de gérer le retour d'une valeur nulle.

```
Cheese[] cheeses = shop.getCheeses();
  if (cheeses != null &&
          Arrays.asList(shop.getCheeses()).contains(Cheese.STILTON))
     System.out.println("Jolly good, just the thing.");
```

au lieu de :

```
if (Arrays.asList(shop.getCheeses()).contains(Cheese.STILTON))
   System.out.println("Jolly good, just the thing.");
```

A faire :

Proposer 2 solutions pour appliquer la règle item 43 :

- une solution utilisant une constante pour un tableau vide
- et une solution utilisant la classe `Collections` pour une collection vide (ré-éecrire le code ci-dessus avec une liste au lieu d'un tableau pour avoir une collection vide à gérer)
- rendre vos propositions de code.

Exercice 3 : Réflexion

La classe Class et le package java.lang.reflect permettent de faire de l'introspection.

L'introception est la capacité d'un programme à examiner son propre état.

La java.lang.Class<T> permet d'obtenir des informations sur les classes

T type de la classe modélisée par cet objet Class

Par exemple le type de String.class est Class<String>

Le package java.lang.reflect fournit des classes et des interfaces pour obtenir des informations réflexives sur les classe : Field, Method, Parameter, Constructor ….

Le fichier « Reflection-java.pdf » vous donne des explications l'introspection en Java et va vous aider à faire les exercices.

1) Enumérer les caractéristiques d'une classe

Ecrire une petit programme qui permet d'obtenir les caractéristiques d'une classe. Soigner votre développement et l'impression des résultats

- la superclasse
- tous les types d'interfaces que la classe implémente
- le package de la classe
- les noms et types des attributs
- Les noms, types de paramètres et types retour de toutes les méthodes.

Vous pourrez explorer des classes de l'API Java ou des classes que vous écrits pour d'autres cours.

2) Métriques quantitatives sur les informations d'une classe

Implémenter une classe qui permet de donner des valeurs quantitatives sur les informations d'une classe. Pour cela faire un classe d'utilitaires dont les méthodes seront les métriques.

Classiquement on cherche à calculer :

- nombre de méthodes
- nombre de variables d'instance
- nombre de variables de classes (static)
- profondeur dans l'arbre d'héritage

Vous pourrez explorer les informations pour des classes de l'API java .

Rendre :

- le code de vos classes et au moins une trace d'exécution