Session 1, 2021

Paradigmes Émergents de la Programmation Programmation Avancée

Examen, 45', Feuille A4 autorisée (à rendre)

Exercice 1

Considérons le code du listing 1:

```
// Listing 1: Code
import java.io.*;
class Moyenne {
  public double valeur () throws FileNotFoundException, I0Exception (
  int nb = 0:
  double somme = 0:
  try (BufferedReader in = new BufferedReader(new FileReader ("exemplel.txt"))) {
    String ligne = null;
    while ((ligne = in. readLine()) != nulL) {
      String[] morceaux = ligne.split(" ");
      nb++;
      somme += Double. parseDouble (morceaux[1]);
    }
    return somme / nb;
  }

  public static void main(String[] args) throws Exception {
    System.out.println(new Moyenne().valeur()) ;
  }
}
```

1. Que fait ce programme ? On donnera un exemple de contenu pour le fichier `exemple.txt`.
2. Indiquer ses limites et les évolutions qu'il serait souhaitable de lui apporter.

Exercice2: `checkedList`

La classe utilitaire `Collections` définit la méthode `checkedList` qui s'utilise de la manière suivante (en considérant que `maListe` et `autre` sont déclarées du même type `List`):

```
autre = Collections.checkedList (maListe, Point.class);
```

Le deuxième paramètre est le type attendu pour les éléments de la liste. Les méthodes qui ajoutent ou remplacent un élément de la liste lèvent l'exception `ClassCastException` si le nouvel élément n'est pas compatible avec le type attendu.

Il existe une telle méthode pour chaque structure de données : `checkedSet`, `checkedMap`...

1. Quel est le patron de conception utilisé pour implanter de telles méthodes ? On donnera le diagramme de classe qui correspond à ce patron et on utilisera du pseudo-code pour en expliquer le comportement.
2. L'introspection pourrait être utilisée pour implanter ces différentes méthodes du *framework* des collections. Expliquer comment ceci serait fait en Java.
3. Les implantations de ces méthodes dans le *framework* des collections n'utilisent pas l'introspection. Quelles raisons peut-on avancer ?

Exercice 3

On considère le code du listing 2:

```
// Listing 2: Code
import java.util.Observable;

public aspect CompteObserver (
  declare parents : CompteSimple extends Observable;

  private void CompteSimple.avertir(double montant) {
    this.setChanged();
    this.notifyObservers(montant);
  }

  pointcut changeSolde (CompteSimple cs, double m) :
    target(cs)
    && args (m)
    && (call (void CompteSimple.crediter (double))
        || call (void CompteSimple.debiter (double)));

  after (CompteSimple cs, double montant) : changeSolde(cs, montant) {
    String methodName = thisJoinPoint.getSignature().getName();
    int facteur = (methodName. equals ("debiter")) ? -1 : 1;
    cs.avertir(facteur * montant);
    System.out.println("Notification sur " + cs);
  }
}
```

1. Quel est le langage utilisé ?
2. Expliquer les éléments qui apparaissent sur ce listing.
3. Dessiner le diagramme de classe de l'application qui s'exécutera.

Exercice 4

Les langages Java et Python proposent la notation `@xxx`.

1. Expliquer l'objectif de cette notation.
2. Donner au moins un exemple de cette notation pour chaque langage, Java et Python.
3. Comparer la mise en œvre de cette notation dans les langages Java et Python.