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

Exercice 1 : unmodifiableList

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

```
...
autre = Collections.unmodifiableList(maListe);
...
```

Si on essaie d'utiliser une opération de modification sur `autre` (`set`, `add`, `remove`, etc.) une exception est levée et `maListe` n'est pas modifiée.

Il existe une telle méthode pour chaque structure de données : `unmodifiableSet`, `unmodifiableMap`...

Il existe aussi d'autres méthodes qui fonctionnent sur le même principe comme `checkedList` et ses homologues qui vérifient que l'élément que l'on ajoute dans une liste est bien du type attendu.

1.1. 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.

1.2. L'introspection pourrait être utilisée pour implanter ces différentes méthodes du *framework* des collections. Expliquer comment ceci serait fait en Java.

1.3. Les implantations de ces méthodes dans le *framework* des collections n'utilisent pas l'introspection. Quelles raisons peut-on avancer ?

Exercice 2 : Contexte de test avec pytest

Le programme suivant est un exemple de programme de test pytest. Le `@pytest.fixture` placé devant les fonctions `set1` et `set2` indique que ces fonctions correspondent à un contexte de test : si une fonction de test (fonction dont le nom commence par test) a un paramètre de même nom qu'un contexte de test, il sera automatiquement initialisé par `pytest` avec le résultat de la fonction de même nom.

```
import pytest

@pytest.fixture
def set1():
    return {4, 5, 6, 7, 6 , 5, 4}

@pytest.fixture
def set2():
    return set(range(1, 6))

def test_taille(set1):
    assert len(set1) == 4

def test_intersection(set1, set2):
    assert set1 & set2 == {4, 5}
```

2.1. Comment s'appellent les `@xxx` en Python et à quoi correspondent-ils dans le langage Python ?
2.2. Expliquer, sans donner le code Python correspondant, comment pourraient être implantés en Python les contextes de test (`pytest.fixture`).
2.3. Le langage Java propose une notation équivalente (par exemple `@Override`, `@Test`) ? Comment ceci est nommé en Java et comment est-ce fait techniquement ?

Exercice 3 : Inversion de contrôle

Expliquer en quoi consiste l'inversion de contrôle. Prendre un exemple concret.

Exercice 4 : Programmation par aspect

On considère le diagramme de classe de la figure 1 qui décrit des expressions simplifiées. Les losanges (qui correspondent en général à une relation n-aire en UML) sont utilisés pour que plantuml produise un placement plus lisible des éléments. Le code correspondant n'est pas donné mais peut se déduire facilement du diagramme de classe.

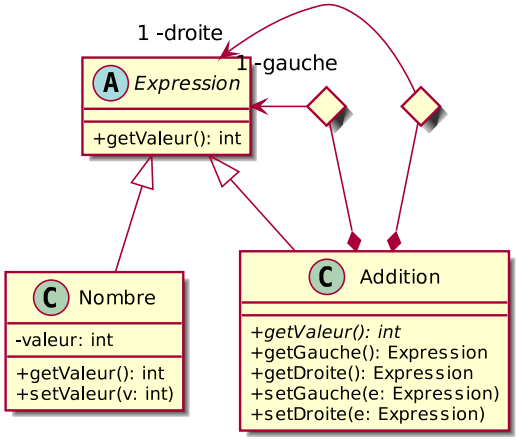

FIGURE 1 – Diagramme de classe des expressions

4.1. Expliquer les principaux éléments du langage AspectJ qui apparaissent sur le listing 1. On répondra directement sur le listing. Il est inutile de donner plusieurs fois la même explication.
4.2. Que fait le code du listing 1 ?
4.3. Quels sont les avantages et les inconvénients de la programmation par aspect.

```aspectj
public aspect Caching {
    private int Expression.cache;
    private boolean Expression.cacheValide = false;
    private Expression Expression.parent = null;

    public int Expression.getCache()           { return cache; }
    public void Expression.setCache(int cache) { this.cache = cache; }
    public boolean Expression.estCacheValide() { return cacheValide; }
    public void Expression.validerCache()      { this.cacheValide = true; }

    public void Expression.invaliderCache() {
        cacheValide = false;
        if (this.getParent() != null) {
            this.getParent().invaliderCache();
    }    }

    public Expression Expression.getParent()            { return parent; }
    public void Expression.setParent(Expression parent) { this.parent = parent; }

    pointcut change(Expression exp):
        target(exp) && (
            call(public void Nombre.setValeur(int)) ||
            call(public void Addition.setGauche(Expression)) ||
            call(public void Addition.setDroite(Expression))
        );

    after(Expression exp):change(exp) {
        exp.invaliderCache();
    }

    pointcut evalue(Expression exp):
        target(exp) && call(public int Expression.getValeur());

    int around(Expression exp):evalue(exp) {
        if (!exp.estCacheValide()) {
            int result = proceed(exp);
            exp.setCache(result);
            exp.validerCache();
        }
        return exp.getCache();
    }

    pointcut creationAddition(Addition exp):
        this(exp) && execution(Addition.new(Expression,Expression));

    after(Addition exp) : creationAddition(exp) {
        exp.getGauche().setParent(exp);
        exp.getDroite().setParent(exp);
}   }
```

Listing 1: Code AspectJ