Université Abou Beckr Belkaid
Master 2 Génie Logiciel
Année Universitaire 2021/2022

Examen Final

—

validation et Vérification logiciel

—

Remarques :

- Les documents ne sont pas autorisées, ainsi que les appareils électroniques (PC, Tablette, téléphone..).
- La lisibilité et la clarté de vos réponses et de votre code sont très importantes. Une réponse pas claire ne sera pas prise en compte lors de la correction.

Questions de cours (5 points)

1,5 1 Donner les différences entre les testes en boites noires et en boites blanches.
1,5 2 Donner la définition des classes d'équivalence.
2 3 Quels sont les cas d'utilisation des testes en isolation ?

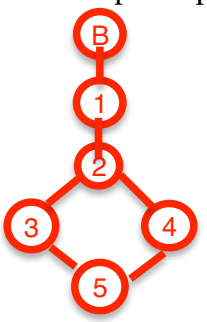
Exercice 1 :(5 points)

On considère la spécification suivante, donnée sous la forme d'une table de vérité liant les entrées du programme x, y (entiers relatifs) et la valeur de la sortie z.

$x < 0$	$y < 0$	Output(z)
true	true	foo1(x,y)
true	false	foo2(x,y)
false	true	foo2(x,y)
false	false	foo1(x,y)

Le but est de vérifier si les spécifications données sont satisfaites par le programme suivant :

```
int x,y,z;                  B
input(x,y);                 1
if (x < 0 & y < 0)          2
  z = foo1(x,y);            3
else
  z = foo2(x,y);            4
output(z);                  5
```



Question :

1 1 Donner le graphe de contrôle du programme.
1 2 Définir des jeux de teste permettant d'obtenir le critère "toutes les instructions".
1 3 Donner des jeux de teste permettant d'obtenir le critère "toutes les branches".
2 4 Est ce que les critères choisis permettent de vérifier si le programme satisfait les spécifications du tableau ? Si oui, justifier. Si non, donner une solution.

La suite de tests $\{t_1 : (x, y) = (-3, -2); t_2 : (x, y) = (-4, 2)\}$ est adéquate sur P_1 pour le critère de couverture des décisions. La sortie pour t_1 est $z = foo1(x, y)$, et pour t_2 on obtient $z = foo2(x, y)$ ce qui est correct par rapport à la spécification (respectivement première et deuxième ligne de la table de vérité). En revanche la suite de tests n'est pas adéquate pour le critère de couverture des conditions, puisqu'on mesure $T = 0.5$ (la condition $x < 0$ n'est pas couverte, puisqu'elle s'évalue à vrai pour t_1 et t_2).

En revanche, le test $t_3 : (x, y) = (3, 4)$ permet de révéler un défaut dans l'implantation : en effet P_1 retourne $z = foo2(x, y)$ alors que la quatrième ligne de la table de vérité indique qu'il faut obtenir $z = foo1(x, y)$. En outre, la suite de tests $\{t_1, t_2, t_3\}$ est adéquate pour les critères « toutes les décisions » et « toutes les conditions ». La troisième ligne de la table de vérité de la spécification n'est couverte par aucun de ces tests.

Exercice 2 : (10 points)

Écrire la classe *Rationnel* qui modélise les rationnels des mathématiciens : ce sont les quotients de la forme p/q où p et q sont des entiers relatifs (entiers avec un signe). Cette classe est donnée comme suite :

```
public class Rationnel {
    private int num;
    private int den;
    public Rationnel(int num, int den) {
     this.num = num;
     this.den = den;
     }

    public int getNum() {
        return num;
    }

    public int getDen() {
        return den;
    }

    public Rationnel addition(Rationnel r2) {
        int denumerateurResult = den * r2.getDen();
        int numerateurResult = num*r2.getDen() + r2.getNum()*den;
        return new Rationnel(numerateurResult, denumerateurResult).simplifier();
    }

    public Rationnel soustraction(Rationnel r2) {
        int denumerateurResult = den * r2.getDen();
        int numerateurResult = num*r2.getDen() - r2.getNum()*den;
        return new Rationnel(numerateurResult, denumerateurResult).simplifier();
    }

    private static int pgcd(int a, int b) {
         if (a < 0) a = -a;
         if (b < 0) b = -b;
         if (a == b) return a;
         else if (a > b) return pgcd (a-b, b);
         else if (b > a) return pgcd (a, b - a);
         return 1;
         }
    private Rationnel simplifier() {
         int lePGCD = pgcd (getNum(), getDen());
         return (new Rationnel (getNum()/lePGCD, getDen()/lePGCD));
         }

    @Override
    public String toString() {
        return "(" + num + "/" + den + ")";
     }
}
```

Question :

1 Donner une classe de teste Junit pour tester les deux méthodes addition et soustraction.

```
class RationnelTest {

    private Rationnel r1;
    private Rationnel r2;
     @Before
    public void init() {
     r1 = new Rationnel(1, 2);
     r2 = new Rationnel(1, 3);
     }
     @Test
1   public void testAddition() {
     Rationnel r3 = r1.addition(r2);
     Rationnel resultat = new Rationnel(5,6);
     assertEquals(r3, resultat);
     System.out.println(resultat);
     }
     @Test
    public void testSoustraction() {
     Rationnel r3 = r1.soustraction(r2);
1    Rationnel resultat = new Rationnel(1,6);
     assertEquals(resultat,r3);
     System.out.println(resultat);
     }
}
```

1 2 Afin d'exécuter la classe de teste, il faut ajouter une méthode à la classe Rationnel. C'est la méthode *Equal()* entre deux nombres rationnel afin de pouvoir exécuter les méthodes *assertEqual()* de Junit.

2 3 Donner l'implémentation de cette méthode.

```java
@Override
public boolean equals(Object obj) {
 if (this == obj)
 return true;
 if (obj == null)
 return false;
 if (getClass() != obj.getClass())
 return false;
 Rationnel other = (Rationnel) obj;
 if (den * other.getNum() != num * getDen())
 return false;
 else {
 return true;
 }
 }
```

Exercice 3 : (5 points)

Dans cet exercice, la classe *Portefeuille* représente un Portefeuille boursier. Et l'interface *Bourse* définit la méthode permettant d'obtenir le cours d'une action.

```java
public class Portefeuille {

       protected final Bourse bourse;
        private Map<String, Integer> actions = new HashMap<String, Integer>();

        public Portefeuille(Bourse bourse) {
            this.bourse = bourse;
        }

        public void add(String action, int nombre) {
            actions.put(action, nombre + actions.getOrDefault(action, 0));
        }

        public double getValeur() {
            return actions.entrySet().stream().map(e -> bourse.getCours(e.getKey()) * e.getValue()).reduce(0.0, Double:
        }

}

public interface Bourse {
    double getCours(String action);
}
```

***Question* :**

1 Nous souhaitons tester la classe *Portefeuille*, quelle méthode est privilégier et pourquoi? nous utili-
2 sons un stub pour un teste en isolation car nous n'avons pas l'implémentation de l'interface *Bourse* en utilisant le framework Mockito.

2 donner l'implémentation de la classe teste.

3

```java
public class PortefeuilleMockTest {

    Bourse bourse = mock(Bourse.class);

    @Test
    public void doitCalculerLaValeurDunPortefeuilleVide() {
        // Given

        Portefeuille portefeuille = new Portefeuille(bourse);

        // When
        double valeur = portefeuille.getValeur();

        // Then
        assertEquals(0, valeur);
    }

    @Test
    public void doitCalculerLaValeur() {
        // Given
        when(bourse.getCours(anyString()))
                .thenAnswer(i -> Double.valueOf(i.getArgument(0,
                String.class).length()));

        Portefeuille portefeuille = new Portefeuille(bourse);
        portefeuille.add("AccorHotels", 1);
        portefeuille.add("Air", 1);
        portefeuille.add("Airbus", 1);
        portefeuille.add("ArcelorMittal", 1);
        portefeuille.add("Atos", 1);
        portefeuille.add("Axa", 1);

        // When
        double valeur = portefeuille.getValeur();

        // Then
        assertEquals(40, valeur);
    }
```