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

Examen de Rattrapage

—

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 la définition des testes d'intégration.

1,5 2 Quels sont les rôles du graphe de flot de contrôle?

2 3 Donner la définitions des étapes du cycle rouge-vert-refactor dans le développement dirigé par les testes.

Exercice 1 :(5 points)

Soit le programme suivant :

```
int x,y,z;
input(x,y);
if (x < 0 | y < 0)   /* (&) devient (|) */
  z = foo1(x,y);
else
  z = foo2(x,y);
output(z);
```

Question **:**

1 1 Donner la définition du critère "toutes les décisions". (voir le cours)

1 2 Donner la définition du critère "toutes les conditions-décisions".(voir le cours)

3 Soit les deux jeux de tests suivants :

$$S_1 = \{(x,y) = (-3,2); (x,y) = (4,2)\}$$

$$S_2 = \{(x,y) = (-3,2); (x,y) = (4,-2)\}$$

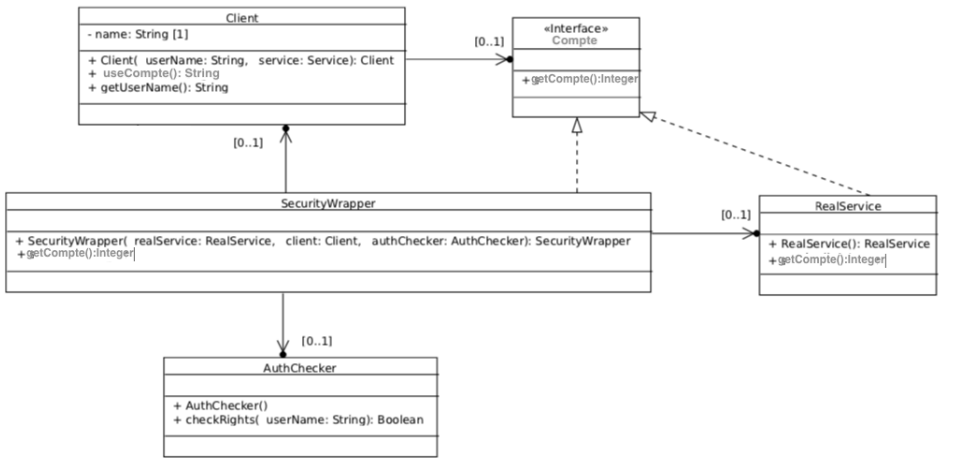
Pour chaque jeux de tests vérifier à quelles critères il répond "toutes les décisions" ou "toutes les conditions-décisions" en justifiant.

1,5 On vérifie que S_1 est adéquate pour le critère « toutes les décisions », mais pas pour le critère « toutes les conditions » : $y < 0$ restant toujours vrai.

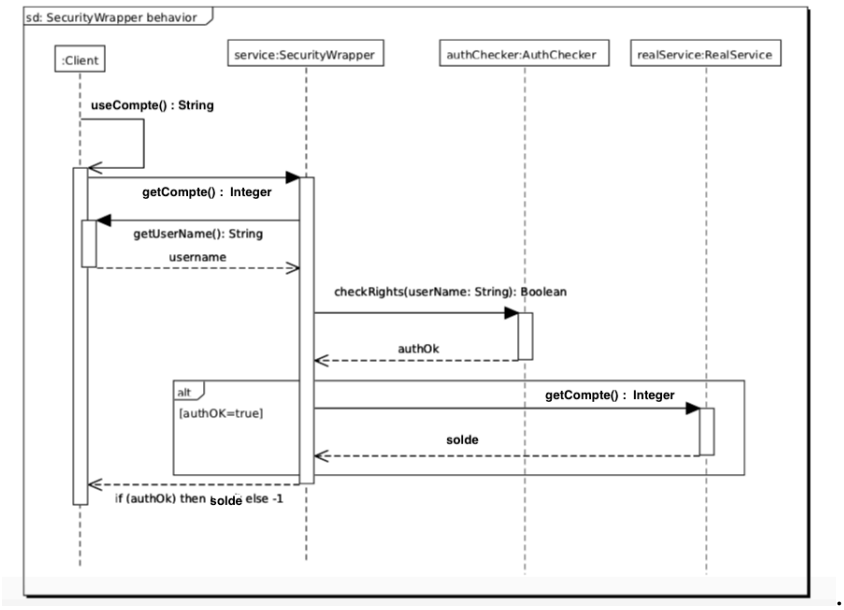
1,5 S_2 est adéquate pour le critère « toutes les conditions », mais pas pour « toutes les décisions » : l'un de $x < 0$ ou $y < 0$ étant toujours évalué à vrai, seule la première branche de la décision est prise.

Exercice 2 : (10 points)

Un système est mis en place afin de sécuriser l'accès à un service, ce système est modélisé comme suite :



La méthode *useCompte()* fait appel à la méthode *getCompte()* et retourne le nom du client suivi de son solde si le client à un compte et "Compte inexistant" sinon. Afin d'accéder au compte un client devra passer la barrière de sécurité en s'authentifiant. Le diagramme de séquence correspondant est donné comme suite :



Questions :

Nous souhaitons tester le comportement de *getCompte()*, du point de vue du *Client* et cela en utilisant les tests d'integration.

4

1 Proposer une stratégie d'integration liée à ce système, en justifiant votre choix.
Pour choisir une stratégie d'integration il faut construire le graphe de dépendance de test puis choisir un ordre d'integration.

2 En utilisant Mockito et en se basant sur le diagramme de séquence donné, donner l'implémentation de la classe des tests associés, et cela en testant :
— la classe Client en isolation.

```
public class CompteTest(){
    private Compte mockCompte;
    private AuthChecker mockChecker;
    private RealService mockRealCompte;
    // Création du mock de la persistance et injection dans l'instance de la classe à tester
    @Before
    public void setUp() throws Exception {
        mockCompte = mock(SecurityWrapper.class);
        mockChecker = mock(AuthChecker.class);
        mockRealCompte = mock(RealService.class);
    }
    /** Les tests de la classe en isolation */
@Test
    public final void testCompteOK(){
     when(mockCompte.getCompte()).thenReturn(100000);
     Client c = new Client("Amine",mockCompte);
    String s= c.useCompte();
        assertEquals("Amine 100000", s);
    }
@Test
    public final void testCompteKO(){
     when(mockCompte.getCompte()).thenReturn(-1);
     Client c = new Client("Amine",mockCompte);

    String s= c.useCompte();
        assertEquals("Compte inexistant", s);
    }
```

3

— les interactions entre classes.

```
    /**
       Les tests de l'interaction entre classes
     */
@Test
    public final void testCompteInteraction(){

     Client c = new Client("Amine",mockCompte);
     InOrder ordreExecution1 = inOrder(mockCompte,mockChecker);
     InOrder ordreExecution2 = inOrder(mockCompte,mockRealCompte);
     InOrder ordreExecution3 = inOrder(mockChecker,mockRealCompte);

    String s= c.useCompte();
    verify(mockCompte).getCompte();

    ordreExecution1.verify(mockCompte).getCompte();
    ordreExecution1.verify(mockChecker).checkRights();

    ordreExecution2.verify(mockCompte).getCompte();
    ordreExecution2.verify(mockRealCompte).getCompte();

    ordreExecution3.verify(mockChecker).checkRights();
    ordreExecution3.verify(mockRealCompte).getCompte();

    }
```

3