Tests des logiciels (Certification ISTQB)

Département Informatique

2023-2024

Mariem HAOUES

Maître-assistant, FSB

Université de Carthage

JUnit

TP 2

Tests unitaires et développement dirigé par les tests

Introduction

- JUnit est un framework open source de développement et d'exécution de tests unitaires pour le langage de programmation Java
- JUnit est disponible sur tous les Environnements de Développement Intégré (IDE) et donc utilisable directement pour tout projet Java
- JUnit est un framework mature pour permettre l'écriture et l'exécution de tests automatisés
- Créé par Kent Beck et Erich Gamma, JUnit est le projet de la série des xUnit connaissant le plus de succès
- JUnit 5, publié en 2017, utilise des fonctionnalités de Java 8 notamment les lambdas, les annotations répétées,
  - Les classes de JUnit 5 sont dans le package org.junit.*;
- Facilite :
  - écriture des programmes de tests unitaires (tests boite blanche)
  - exécution des test unitaires
  - l'exploitation des résultats de test

Introduction

Exemple introductif de cas de test

- Le programme suivant fournit des méthodes statiques pour faire des calculs arithmétiques d'addition et de division

```
public final class Calculator {
        public static int add (int operand1, int operand2 )
                {return operand1 + operand2;}
        public static int div (int operand1, int operand2 )
                {return operand1 / operand2;}
  }
```

- La classe de test suivante code un exemple de cas de test pour les deux opérations *add* et *div* sous la forme de deux méthodes de test unitaire

```
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
public final class CalculatorTest {
@Test
public void
testAdd() {assertEquals(3, Calculator.add(1, 2));}
@Test
public void
testDiv() {assertEquals(1, Calculator.div(3, 2));}
}
```

Les éléments de base de JUnit

- JUnit définit un certain nombre d'éléments pour simplifier l'écriture de tests, parmi lesquels les annotations et les assertions

Les annotations

La création de test avec JUnit utilise les annotations afin de rendre l'écriture du code moins verbeuse

Les annotations de JUnit

Annotation	Description
@Test **public void** should_do_when_case()	Marque une méthode comme contenant du code de test.
@Before **public void** setup()	Marque une méthode pour qu'elle s'exécute avant chaque méthode de test de la classe.
@After **public void** tearDown()	Marque une méthode pour qu'elle s'exécute après chaque méthode de test de la classe.
@BeforeClass **public static void** setupBeforeClass()	Marque une méthode statique pour qu'elle s'exécute avant la première méthode de test de la classe. Elle ne s'exécute qu'une seule fois par classe.
@AfterClass **public static void** tearDownAfterClass()	Marque une méthode statique pour qu'elle s'exécute après la dernière méthode de test de la classe. Elle ne s'exécute qu'une seule fois par classe.
@Test(expected=*Exception.class*) **public void** should_do_when_case()	Marque une méthode de test pour vérifier que la méthode testée lève bien l'exception attendue.
@Test(timeout=*100*) **public void** should_do_when_case()	Marque une méthode de test pour vérifier que la méthode testée s'exécute dans le temps imparti.

Les éléments de base de JUnit

Les assertion

Une assertion automatise le verdict d'un test unitaire pour décider si la méthode testée passe avec succès ou échoue le test. Les assertions Junit sont des méthodes statiques de la classe Assert. Chaque méthode existe en deux versions, avec ou sans un premier paramètre contenant le message qui sera affiché en cas d'erreurs.

Fonction	Description
assertTrue(*condition*)	Vérifie que la condition en paramètre est vraie.
assertFalse(*condition*)	Vérifie que la condition en paramètre est fausse.
assertEquals(*expected*, *actual*)	Teste si les valeurs sont égales au sens de la méthode `equals()`. Note : pour les tableaux, seule la référence est vérifiée pas le contenu.
assertArrayEquals(*expected*, *actual*)	Teste si les valeurs des éléments sont les mêmes dans les deux tableaux.
assertEquals(*expected*, *actual*, *tolerance*)	Vérifie l'égalité entre doubles ou flottants à une valeur de précision près.
assertNull(*object*)	Vérifie que l'objet en paramètre est bien « null ».
assertNotNull(*object*)	Vérifie que l'objet en paramètre n'est pas « null ».
assertSame(*expected*, *actual*)	Vérifie que les deux variables pointent vers le même objet en utilisant l'opérateur ==.
assertNotSame(*expected*, *actual*)	Vérifie que les deux variables ne pointent pas vers le même objet.

Exercice n° 1

L'objectif de la fonction **validateNumber**, est de vérifier si un nombre donné est un nombre premier ou non. Elle prend en entrée un entier **primeNumber** et retourne un booléen : **true** si le nombre est premier, **false** sinon. Pour cela, la fonction parcourt tous les nombres de 2 à la moitié du nombre donné et vérifie s'il existe un diviseur autre que 1 et le nombre lui-même. Si elle trouve un tel diviseur, elle retourne **false**, sinon elle retourne **true**. Un nombre premier est défini comme un nombre entier positif supérieur à 1 qui n'a pas d'autres diviseurs que 1 et lui-même.

```
package monprojet;
public class PrimeNumber {
public static Boolean validateNumber(final Integer primeNumber) {
for (int i = 2; i < (primeNumber / 2); i++)
     {
       if (primeNumber % i == 0) { return false; }
     }
return true;
  }
}
```

1. Proposez quatre cas de test pour la fonction **validateNumber**. Pour chaque cas de test, donnez l'objectif de test, les données d'entrée et les résultats attendus.
2. Rédigez ces quatre scénarios de test en utilisant **JUnit**.
3. Exécuter les cas de tests pour détecter le bug. Corriger le code de la fonction **validateNumber**.

Solution Exercice n° 1

Proposez quatre cas de test pour la fonction **validateNumber**. Pour chaque cas de test, donnez l'objectif de test, les données d'entrée et les résultats attendus.

Objectif de test: Vérifier si un nombre premier est correctement identifié comme tel.
Données d'entrée: Nombre premier, par exemple 7.
Résultat attendu: La fonction doit retourner **true**.

Objectif de test: Vérifier si un nombre non premier est correctement identifié comme tel.
Données d'entrée: Nombre non premier, par exemple 6.
Résultat attendu: La fonction doit retourner **false**.

Objectif de test: Vérifier si le nombre 1 est correctement identifié comme non premier.
Données d'entrée: Nombre 1.
Résultat attendu: La fonction doit retourner **false**.

Objectif de test: Vérifier si la fonction gère correctement les nombres négatifs en les considérant comme non premiers.
Données d'entrée: Nombre négatif, par exemple -10.
Résultat attendu: La fonction doit retourner **false**.

Solution Exercice n° 1

Rédigez ces quatre scénarios de test en utilisant JUnit.

```
package monprojet;

import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

public class PrimeNumberTest {

@BeforeEach
public void setUp() {
new PrimeNumber();
}

@Test
public void testValidatePrimeNumber() {
boolean result = PrimeNumber.validateNumber(7);
assertEquals(true, result);
}

@Test
public void testValidateNonPrimeNumber() {
boolean result = PrimeNumber.validateNumber(6);
assertEquals(false, result);
}
```

Solution Exercice n° 1

Rédigez ces quatre scénarios de test en utilisant JUnit.

```
@Test
public void testValidateNegativeNumber() {
boolean result = PrimeNumber.validateNumber(-10);
assertEquals(false, result);
}

@Test
public void testValidateOne() {
boolean result = PrimeNumber.validateNumber(1);
assertEquals(false, result);
}

@Test
public void testValidateZero() {
boolean result = PrimeNumber.validateNumber(0);
assertEquals(false, result);
}

}
```

Solution Exercice n° 1

Exécuter les cas de tests pour détecter le bug. Corriger le code de la fonction **validateNumber**.

```
package monprojet;

public class PrimeNumber
{
        public static Boolean validateNumber(final Integer primeNumber)
        {
          if (primeNumber <= 1) { return false; }
          for (int i = 2; i <= Math.sqrt(primeNumber); i++)
              {
                if (primeNumber % i == 0)
                   { return false; }
              }
          return true;
        }
}
```

Exercice n° 2

La fonction **findOccurrence** recherche la première occurrence d'une valeur cible dans un tableau d'entiers. Elle parcourt chaque élément du tableau et compare sa valeur à la valeur cible. Si la valeur cible est trouvée, le code retourne l'index de la première occurrence. Sinon, elle retourne -1 pour indiquer que la valeur cible n'a pas été trouvée dans le tableau.

```
package monprojet;

public class findFirstOccurrence
{
        public static int findOccurrence(int[] array, int target)
        {
          for (int i = 0; i < array.length; i++)
            {
              if (array[i] == target)
              return i;
            }
         return -1;
        }
}
```

1. Proposez quatre cas de test pour la fonction **findOccurrence**. Pour chaque cas de test, donnez l'objectif de test, les données d'entrée et les résultats attendus.
2. Rédigez ces quatre scénarios de test en utilisant **JUnit**.

Solution Exercice n° 2

La fonction **findOccurrence** recherche la première occurrence d'une valeur cible dans un tableau d'entiers. Elle parcourt chaque élément du tableau et compare sa valeur à la valeur cible. Si la valeur cible est trouvée, le code retourne l'index de la première occurrence. Sinon, elle retourne -1 pour indiquer que la valeur cible n'a pas été trouvée dans le tableau.

```
package monprojet;

public class findFirstOccurrence
{
        public static int findOccurrence(int[] array, int target)
        {
          for (int i = 0; i < array.length; i++)
            {
              if (array[i] == target)
              return i;
            }
         return -1;
        }
}
```

1. Proposez quatre cas de test pour la fonction **findOccurrence**. Pour chaque cas de test, donnez l'objectif de test, les données d'entrée et les résultats attendus.
2. Rédigez ces quatre scénarios de test en utilisant **JUnit**.

Solution Exercice n° 2

Proposez deux cas de test pour la fonction **findOccurrence**. Pour chaque cas de test, donnez l'objectif de test, les données d'entrée et les résultats attendus.

Objectif de test: Vérifier si la fonction trouve la première occurrence de la valeur cible dans le tableau lorsque la valeur cible est présente.
Données d'entrée: Tableau [1, 3, 5, 7, 5, 9] et valeur cible 5.
Résultat attendu: L'index 2 (correspondant à la première occurrence de 5 dans le tableau).

Objectif de test: Vérifier si la fonction retourne -1 lorsque la valeur cible n'est pas présente dans le tableau.
Données d'entrée: Tableau [2, 4, 6, 8, 10] et valeur cible 7.
Résultat attendu: -1 (la valeur cible 7 n'est pas présente dans le tableau).

Solution Exercice n° 2

Rédigez ces deux scénarios de test en utilisant **JUnit**.

```
package monprojet;
import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.Test;
public class findFirstOccurrenceTest
{
	@Test
	public void testFindFirstOccurrence_FirstOccurrencePresent()
	{
		int[] array = {1, 3, 5, 7, 5, 9};
		int target = 5;
		int expected = 2;
		int actual = findFirstOccurrence.findOccurrence(array, target);
		assertEquals(expected, actual);
	}

	@Test
	public void testFindFirstOccurrence_TargetNotPresent()
	{
		int[] array = {2, 4, 6, 8, 10};
		int target = 7;
		int expected = -1;
		int actual = findFirstOccurrence.findOccurrence(array, target);
		assertEquals(expected, actual);
	}
}
```

Exercice n° 3

La fonction **validateForm** de la classe **FormValidator** prend trois paramètres : firstName, lastName, et email, qui représentent respectivement le prénom, le nom de famille, et l'e-mail d'un formulaire. Elle vérifie si ces champs sont vides ou non. Si l'un des champs est vide, elle renvoie **false**, indiquant que le formulaire est invalide. Ensuite, elle vérifie si l'e-mail est valide en appelant la méthode **isValidEmail**. Si l'e-mail n'est pas valide, la fonction renvoie également **false**. Sinon, elle renvoie **true**. La méthode **isValidEmail** vérifie si l'e-mail contient le caractère "@" pour déterminer si l'e-mail est valide.

```
package monprojet;
public class FormValidator
{
public boolean validateForm(String firstName, String lastName, String email)
  {
     if (firstName.isEmpty() || lastName.isEmpty() || email.isEmpty())
          { return false; // Au moins l'un des champs est vide}
     if (!isValidEmail(email))
          { return false; // L'e-mail n'est pas valide }
     // Autres règles de validation...
     return true; // Tout est valide
  }
private boolean isValidEmail(String email)
  { return email.contains("@"); // Implémentation de la validation de l'e-mail }
}
```

Solution Exercice n° 3

Proposez trois cas de test pour la fonction **FormValidator**. Pour chaque cas de test, donnez l'objectif de test, les données d'entrée et les résultats attendus.

Objectif : Vérifier si la fonction renvoie **true** lorsque tous les champs du formulaire sont remplis correctement.
Données d'entrée : Prénom = "John", Nom de famille = "Doe", Email = "john@example.com".
Résultat attendu : La fonction doit renvoyer **true**.

Objectif : Vérifier si la fonction renvoie false lorsque l'un des champs du formulaire est vide.
Données d'entrée : Prénom = "", Nom de famille = "Doe", Email = "john@example.com".
Résultat attendu : La fonction doit renvoyer **false**.

Objectif : Vérifier si la fonction renvoie false lorsque l'e-mail n'est pas valide.
Données d'entrée : Prénom = "John", Nom de famille = "Doe", Email = "johnexample.com".
Résultat attendu : La fonction doit renvoyer **false**.

Solution Exercice n° 3

Rédigez ces trois scénarios de test en utilisant Junit

```
package monprojet;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
public class FormValidatorTest
{
@Test
public void testValidateForm_ValidInput_ReturnsTrue()
 {
     FormValidator validator = new FormValidator();
     assertTrue(validator.validateForm("John", "Doe", "john@example.com"));
 }
@Test
public void testValidateForm_EmptyField_ReturnsFalse()
 {
     FormValidator validator = new FormValidator();
     assertFalse(validator.validateForm("", "Doe", "john@example.com"));
 }
@Test
public void testValidateForm_InvalidEmail_ReturnsFalse()
 {
     FormValidator validator = new FormValidator();
     assertFalse(validator.validateForm("John", "Doe", "johnexample.com"));
 }
}
```