ÉTS
Le génie pour l'industrie

LOG121
Conception orientée objet

Tests unitaires: une introduction

Enseignante: Souad Hadjres

Plan

- Tests en génie logiciel
- Tests unitaires
  - Qu'est ce que c'est, pourquoi et quand?
- Cadres de développement pour les tests
  - Exemple de JUnit

Tests en génie logiciel

- Technique empirique qui vise à
  - collecter un certain nombre d'informations pour évaluer la conformité d'une implémentation à la spécification et aux attentes du client
    - Le fonctionnement et la qualité de l'implémentation sous certaines conditions
  - détecter et corriger les erreurs dans l'implémentation

Tests en génie logiciel

- Faire des tests à différents niveaux
  - Unitaire: test d'une classe/ d'un composant
  - D'intégration: vérifier l'interface et l'interaction entre deux composants différents
    - L'intégration doit se faire de façon incrémentale
  - Système: l'ensemble du système par rapport aux exigences

…

Plan

- Tests en génie logiciel
- Tests unitaires
  - Qu'est ce que c'est, pourquoi et quand?
- Cadres de développement pour les tests
  - Exemple de JUnit

Tests unitaires

- Un test unitaire se concentre sur une unité du code source
- En orienté objet, l'unité individuelle est la classe
- Pourquoi est-ce important de fournir un jeu de tests lorsqu'on implémente une classe?

Tests unitaires

- Objectifs
  - Prouver que la classe fonctionne correctement
  - Re-tester la classe à chaque fois qu'elle est modifiée (tests de régression)
  - Réduire le temps que l'on passe à déboguer
  - Permettre la réalisation de tests de façon régulière
  - Faciliter l'intégration plus tard avec le reste de l'application

Tests unitaires

- Quand est-ce qu'on écrit nos tests?
  - Les tests doivent être développés en même temps que la classe
    - Avant, pendant et après l'implémentation de la classe
  - Souvent cela dépend du processus adopté de développement
    - Dans le processus agile, on parle du développement dirigé par les tests (TDD)

Plan

- Tests en génie logiciel
- Tests unitaires
  - Qu'est ce que c'est, pourquoi et quand?
- Cadres de développement pour les tests
  - Exemple de JUnit

Les cadres de développement pour les tests unitaires

- Il existe plusieurs cadres de développements qui facilitent l'écriture de tests unitaires
  - Exemples: JUnit, CUnit
- Rappel: Un cadre de développement « *framework* » est une librairie de classes qui facilite le développement d'applications
  - Il fournit une fonctionnalité générique que l'on peut étendre pour implémenter une fonctionnalité plus spécifique
  - Il fournit une interface bien définie (API)

JUnit pour les tests unitaires

- Pour tester une classe avec JUnit, on doit créer une classe de tests
  - Chaque cas de test doit être placé dans une méthode
- Pour créer des tests en JUnit
  - On créé une classe qui regroupe les tests JUnit
    - Recommandation: Le nom d'une classe de test est le nom de la classe à tester suivi par « Test », ex: MaClasseTest pour tester MaClasse
  - Un test JUunit est une méthode de test
    - Une méthode de test est précédée du tag **@Test**
    - On utilise des « assertions » dans le corps de cette méthode pour comparer le résultat retourné par une méthode par rapport au résultat attendu

JUnit pour les tests unitaires

- Un cas de test est organisé comme suit

```
@Test
public void test_Add()
{
   Addition addition = new Addition();
   int X1 = 20;
   int X2 = 40;
   int somme = addition.add(X1, X2);
   assertTrue(somme == 60);
}
```

Ici la méthode teste la méthode Add de la classe Addition

Créer les conditions correspondant à l'état avant le test

Exécuter la méthode qu'on veut tester

Vérifier que l'état après exécution correspond à l'état attendu (assertions)

JUnit pour les tests unitaires

- Un simple exemple: une classe et sa classe de test

```
public class Addition {
  public int add(int x, int y){
    return x + y;
  }
}
```

```
import monPackage.Addition;
import org.junit.Test;
import static org.junit.Assert.assertTrue;

public class AdditionTest {
        @Test
        public void testAdd() {
                Addition addition = new Addition();
                int x1 = 40;
                int y1 = 20;
                int sum = addition.add(x1, y1);
                assertTrue(sum == 60);
        }
}
```

Note: Cet exemple a été créé avec la version 4 de JUnit

JUnit pour les tests unitaires

- On peut aussi créer une suite de tests pour exécuter un ensemble de tests

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
import org.junit.runners.Suite.SuiteClasses;

@RunWith(Suite.class)
@SuiteClasses({ AdditionTest.class, MultiplicationTest.class})
public class AllTests {
}
```

- Exécuter des tests dans votre code

```
import org.junit.runner.JUnitCore;
import org.junit.runner.Result;
import org.junit.runner.notification.Failure;

public class MaClasseTestRunner {
    public static void main(String[] args) {
        Result result = JUnitCore.runClasses(AdditionTest.class);
        for (Failure failure : result.getFailures()) {
            System.out.println(failure.toString());
        }
    }
}
```

Note: Cet exemple a été créé avec la version 4 de JUnit