Initiation aux frameworks : *JUnit*

Automatiser les tests unitaires avec JUnit, FestAssert et Mockito

Thomas Duchatelle (duchatelle.thomas@gmail.com)

Capgemini, pour Yves Rocher

January 24, 2013

Sommaire

Tests unitaires
Objectifs et définitions

Tests unitaires

Isoler une fonctionnalité ou un composant et tester son fonctionnement hors contexte.

Tests unitaires
Objectifs et définitions

Tests unitaires

Isoler une fonctionnalité ou un composant et tester son fonctionnement hors contexte.

Dans le cadre de la *Séparation des Préoccupations*

Une classe de test pour chaque brique logicielle, testée indépendamment des autres.

Intérêts des tests unitaires :

- tester tous les cas possibles d'une briques : passant et non-passant

Intérêts des tests unitaires :

- tester tous les cas possibles d'une briques : passant et non-passant
- assurer un non-régression sur les fonctionnalités testées, quelque soit le développeur

Intérêts des tests unitaires :

- tester tous les cas possibles d'une briques : passant et non-passant
- assurer un non-régression sur les fonctionnalités testées, quelque soit le développeur
- ne nécessite pas d'avoir fini l'application pour tester un composant

Bonne pratique

TDD : Test Driven Development

Développement Piloté par les Tests

Méthode de développement consistant à écrire les tests avant de développer le code.

Bonne pratique

TDD : Test Driven Development

Développement Piloté par les Tests

Méthode de développement consistant à écrire les tests avant de développer le code.

Nouveau développement :

1. écrire l'interface de la brique à développer
2. écrire les tests, à partir des spécifications
3. vérifier que les tests **ne** passent **pas**
4. développer la fonctionnalité
5. vérifier que les tests passent

Sommaire

Framework Junit

JUnit

JUnit est un *framework* exécutant les tests unitaires d'une application.

Framework Junit

JUnit

JUnit est un *framework* exécutant les tests unitaires d'une application.

Framework JUnit s'occupe de lister les tests à exécuter, les faire passer dans le contexte approprié et collecter les résultats afin d'en fournir un rapport.

Framework Junit

JUnit

JUnit est un *framework* exécutant les tests unitaires d'une application.

Framework JUnit s'occupe de lister les tests à exécuter, les faire passer dans le contexte approprié et collecter les résultats afin d'en fournir un rapport.

Maven et JUnit

Maven, outils de compilation, exécute les tests unitaires à chaque compilation. En cas d'échec, il ne produit pas le binaire.

Première classe de test

... avec Spring

```
1  @RunWith(SpringJUnit4ClassRunner.class)
2  @ContextConfiguration(locations = {
3    "classpath:spring/addressrepository-core.xml",
4    "classpath:spring/mocked-persistence.xml"
5  })
6  public class CalculatriceImplTest {
7
8    @Inject
9    private ICalculatrice calculatrice;
10
11   @Test
12   public void test_une_fonction() throws Exceptions {
13 // mon test
14   }
15 }
```

Première classe de test

... avec Spring

```
1  @RunWith(SpringJUnit4ClassRunner.class)
2  @ContextConfiguration(locations = {
3    "classpath:spring/addressrepository-core.xml",
4    "classpath:spring/mocked-persistence.xml"
5  })
6  public class CalculatriceImplTest {
7
8    @Inject
9    private ICalculatrice calculatrice;
10
11   @Test
12   public void test_une_fonction() throws Exceptions {
13 // mon test
14   }
15 }
```

- `@RunWith` : détermine l'outil à utiliser pour les tests. Ici une extension pour Spring.

Première classe de test

... avec Spring

```
1  @RunWith(SpringJUnit4ClassRunner.class)
2  @ContextConfiguration(locations = {
3    "classpath:spring/addressrepository-core.xml",
4    "classpath:spring/mocked-persistence.xml"
5  })
6  public class CalculatriceImplTest {
7
8    @Inject
9    private ICalculatrice calculatrice;
10
11   @Test
12   public void test_une_fonction() throws Exceptions {
13 // mon test
14   }
15 }
```

- `@RunWith` : détermine l'outil à utiliser pour les tests. Ici une extension pour Spring.
- `@ContextConfiguration` : liste des fichiers de configuration de Spring

Première classe de test

... avec Spring

```
1  @RunWith(SpringJUnit4ClassRunner.class)
2  @ContextConfiguration(locations = {
3    "classpath:spring/addressrepository-core.xml",
4    "classpath:spring/mocked-persistence.xml"
5  })
6  public class CalculatriceImplTest {
7
8    @Inject
9    private ICalculatrice calculatrice;
10
11   @Test
12   public void test_une_fonction() throws Exceptions {
13 // mon test
14   }
15 }
```

- `@RunWith` : détermine l'outil à utiliser pour les tests. Ici une extension pour Spring.
- `@ContextConfiguration` : liste des fichiers de configuration de Spring
- `@Test` : déclare la méthode comme un test à exécuter. La méthode doit-être publique, sans argument ni de retour.

Structure d'une méthode de test

Une méthode de test comporte 3 parties :

Structure d'une méthode de test

Une méthode de test comporte 3 parties :

1. création du jeu de données

Structure d'une méthode de test

Une méthode de test comporte 3 parties :

1. création du jeu de données
2. exécution du test

Structure d'une méthode de test

Une méthode de test comporte 3 parties :

1. création du jeu de données
2. exécution du test
3. vérification des résultats

Exemple simple

```
1   @Test
2   public void testAdd() {
3     // 1. Initilization
4     int a = 6;
5     int b = 12;
6
7     // 2. Execution
8     int result = calculatrice.add(a, b);
9
10    // 3. Verification / Assertion
11    assertThat(result).isEqualTo(18);
12  }
```

Sommaire

Fest Assert

Écrire les assertions dans un langage courant

Fest Assert

Outils facilitant l'écriture des assertions pour se rapprocher d'un langage courant.

Assertions : types basiques

```
1   // types primitifs
2   assertThat(12 - 9).isEqualTo(3).isGreaterThanOrEqualTo(3).isLessThan(4);
3
4   // String
5   assertThat("Bonjour monde !").isEqualToIgnoringCase("BONJOUR MONDE !").startWith("
        Bonjour").contains("mon");
6
7   // Instance / classe
8   assertThat(yoda).isInstanceOf(Jedi.class);
9   assertThat(frodo.getName()).isEqualTo("Frodo");
10  assertThat(frodo).isNotEqualTo(sauron);
11
12
13  assertThat(frodo).isIn(fellowshipOfTheRing);
14  assertThat(sauron).isNotIn(fellowshipOfTheRing);
15  
```

Assertions : collections

```
1  assertThat(frodo).isIn(fellowshipOfTheRing);
2  assertThat(sauron).isNotIn(fellowshipOfTheRing);
3
4  assertThat(fellowshipOfTheRing).hasSize(9)
5                                 .contains(frodo, sam)
6                                 .excludes(sauron);
```

Assertions : exceptions

```
1   try {
2     calculatrice.div(42, 0); // argggl !
3     // si ArithmeticException n'a pas ete levee, le test echoue avec le message :
4     // "Expected IndexOutOfBoundsException to be thrown"
5     failBecauseExceptionWasNotThrown(ArithmeticException.class);
6   } catch (Exception e) {
7     assertThat(e).isInstanceOf(ArithmeticException.class)
8                  .hasMessageContaining("by zero")
9                  .hasNoCause();
10  }
```

Sommaire

Sommaire