Initiation aux frameworks : *JUnit*

Automatiser les tests unitaires avec JUnit, FestAssert et Mockito

Thomas Duchatelle (duchatelle.thomas@gmail.com)

Capgemini, pour Yves Rocher

February 13, 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 :

- peuvent être exécutés automatiquement (intégration continue, maven)

Intérêts des tests unitaires :

- peuvent être exécutés automatiquement (intégration continue, maven)
- tester tous les cas possibles d'une briques : passant et non-passant

Intérêts des tests unitaires :

- peuvent être exécutés automatiquement (intégration continue, maven)
- tester tous les cas possibles d'une briques : passant et non-passant
- assurer la non-régression sur les fonctionnalités testées, quelque soit le développeur

Intérêts des tests unitaires :

- peuvent être exécutés automatiquement (intégration continue, maven)
- tester tous les cas possibles d'une briques : passant et non-passant
- assurer la 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.

Correctifs

Avant de corriger le code, reproduire l'erreur en test unitaire !

Bonne pratique

TDD : Test Driven Development

Développement Piloté par les Tests

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

Correctifs

Avant de corriger le code, reproduire l'erreur en test unitaire !

Correctif :

1. ajouter un test mettant en évidence le bug. *Il ne doit pas passer.*

Bonne pratique

TDD : Test Driven Development

Développement Piloté par les Tests

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

Correctifs

Avant de corriger le code, reproduire l'erreur en test unitaire !

Correctif :

1. ajouter un test mettant en évidence le bug. *Il ne doit pas passer.*
2. développer le correctif

Bonne pratique

TDD : Test Driven Development

Développement Piloté par les Tests

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

Correctifs

Avant de corriger le code, reproduire l'erreur en test unitaire !

Correctif :

1. ajouter un test mettant en évidence le bug. *Il ne doit pas passer.*
2. développer le correctif
3. vérifier que les tests passent (nouveau + non régression)

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. implémenter la fonctionnalité
5. vérifier que les tests passent

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. implémenter la fonctionnalité
5. vérifier que les tests passent

Évolution :

1. modifier / compléter les tests unitaires
2. vérifier que les tests **ne** passent **pas**
3. développer l'évolution
4. 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.

- liste les tests à exécuter
- les exécute dans le contexte approprié
- collecte les résultats afin d'en fournir un rapport.

Framework Junit

JUnit

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

- liste les tests à exécuter
- les exécute dans le contexte approprié
- collecte 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/employeerepository-context.xml"
4  })
5  public class CalculatriceImplTest {
6
7    @Inject
8    private ICalculatrice calculatrice;
9
10   @Test
11   public void test_une_fonction() throws Exceptions {
12 // mon test
13   }
14 }
```

Première classe de test

... avec Spring

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

- `@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/employeerepository-context.xml"
4  })
5  public class CalculatriceImplTest {
6
7    @Inject
8    private ICalculatrice calculatrice;
9
10   @Test
11   public void test_une_fonction() throws Exceptions {
12 // mon test
13   }
14 }
```

- `@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/employeerepository-context.xml"
4  })
5  public class CalculatriceImplTest {
6
7    @Inject
8    private ICalculatrice calculatrice;
9
10   @Test
11   public void test_une_fonction() throws Exceptions {
12 // mon test
13   }
14 }
```

- `@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 :

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.

Fest Assert

Écrire les assertions dans un langage courant

Fest Assert

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

Assertion Condition qui doit être vérifiée pour continuer. Si elle ne l'est pas, le test s'interrompt et il est en échec (*failure*).

Assertions : types basiques

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

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);
7
8
9  assertThat(extractProperty("age", Integer.class).from(fellowshipOfTheRing)).contains
       (35, 17);
```

Assertions : exceptions

```
1   try {
2     calculatrice.div(42, 0); // argggl !
3
4     // si ArithmeticException n'a pas ete levee, le test echoue avec le message :
5     // "Expected IndexOutOfBoundsException to be thrown"
6     failBecauseExceptionWasNotThrown(ArithmeticException.class);
7
8   } catch (Exception e) {
9     assertThat(e).isInstanceOf(ArithmeticException.class)
10                  .hasMessageContaining("by zero")
11                  .hasNoCause();
12  }
```

Assertions : objets

Écrire ses propres assertions

```
1  // Objectifs :
2  assertThat(employee).isHiredBy(yvesRocher)
3                      .hasEmail("foo.bar@yrnet.com")
4                      .hasEmailDomain("yrnet.com")
5                      .isRA()
6                      .isRaOf("rc", "vpci");
```

Assertions : objets

```
1  public class EmployeeAssertion extends AbstractAssert<EmployeeAssertion, Employee> {
2
3    /** Constructeur obligatoire */
4    public EmployeeAssertion(Employee actual) {
5      super(actual, EmployeeAssertion.class);
6    }
7
8    public EmployeeAssertion isRa() {
9      if(actual.getManagedApplications().isEmpty()) {
10       throw new AssertionError("Employee is not RA");
11     }
12
13     return this;
14   }
15
16   public EmployeeAssertion isHiredBy(Enterprise expected) {
17     if( expected != null && ! expected.isEquals(actual.getEnterprise())) {
18       throw new AssertionError("Expected enterprise to be " + expected + ", but was "
               + actual.getEnterprise());
19     }
20
21     return this;
22   }
23 }
```

Assertions : objets

```
1  public class EmployeeAssertion extends AbstractAssert<EmployeeAssertion, Employee> {
2
3    /** Constructeur obligatoire */
4    public EmployeeAssertion(Employee actual) {
5      super(actual, EmployeeAssertion.class);
6    }
7
8    public EmployeeAssertion isRa() {
9      if(actual.getManagedApplications().isEmpty()) {
10       throw new AssertionError("Employee is not RA");
11     }
12
13     return this;
14   }
15
16   public EmployeeAssertion isHiredBy(Enterprise expected) {
17     if( expected != null && ! expected.isEquals(actual.getEnterprise())) {
18       throw new AssertionError("Expected enterprise to be " + expected + ", but was "
               + actual.getEnterprise());
19     }
20
21     return this;
22   }
23 }
```

```
1  public static EmployeeAssertion assertThat(Employee employee) {
2    return new EmployeeAssertion(employee);
3  }
```

Sommaire

Qu'est-ce qu'un *Mock* ?

Mock Bouchon dont le comportement peut-être décrit pour chaque test et où les appels peuvent être contrôlés.

Qu'est-ce qu'un *Mock* ?

Mock Bouchon dont le comportement peut-être décrit pour chaque test et où les appels peuvent être contrôlés.

Mocker (terme non officiel) Remplacer les dépendances d'une brique applicative par des *mocks*.

Mockito

"A mocking framework that tastes really good."

Mockito

Mockito est un framework qui isole une brique, et vérifie son comportement vis à vis de ses dépendances. Il propose une API pour valider les appels, et diriger le comportement des autres briques.

Mocker une brique applicative

```
1   public class EmployeeManagerImplTest {
2
3     /** Classe mockee */
4     @InjectMock
5     private EmployeeManagerImpl employeeManager;
6
7     /** Mock du generateur d'email */
8     @Mock
9     private IEmailGenerator emailGenerator;
10
11    /** Mock de la couche de persistance */
12    @Mock
13    private IEmployeeDAO employeeDao;
14
15    /** Espionne/Controle les appels au generateur de matricule */
16    @Inject
17    @Spy
18    private IEmployeeNumberGenerator employeeNumberGenerator;
19  }
```

Assertions

Vérifier que les dépendances soient correctement appelées

```
1   @Test
2   public void test_create_new_employee() {
3     // CREATION DU JEU DE TEST
4     when(emailGenerator.generateEmail(any(Employee.class))).thenReturn("ironman@stark-
          enterprise.us");
5     when(employeeNumberGenerator.generateNumber()).thenReturn(42, 43, 44);
6
7     // EXECUTION
8     Employee employee = new Employee("Tony", "Stark");
9     employee.setEnterprise("Stark");
10
11    employeeManager.createNewEmployee(employee);
12
13    // ASSERTIONS
14    // mon client est complet
15    assertThat(employee).hasFirstnameAndLastName("TONY", "STARK")
16                        .isHiredBy("stark")
17                        .hasEmail("ironman@stark-enterprise.us")
18                        .hasNumber(42);
19
20    // les generateurs ont ete correctement appeles et l'employe a ete sauvegarde
21    verify(emailGenerator).generateEmail(employee);
22    verify(employeeNumberGenerator).generateNumber();
23    verify(employeeDao).save(employee);
24
25    verifyNoMoreInteractions(employeeDao, employeeNumberGenerator, emailGenerator);
26  }
```

Sommaire

Ce qu'il faut retenir...

Définitions

Tests unitaires Isole une brique applicative pour valider qu'elle remplisse son rôle, indépendamment de ses dépendances.

Ce qu'il faut retenir...

Définitions

Tests unitaires Isole une brique applicative pour valider qu'elle remplisse son rôle, indépendamment de ses dépendances.

TDD *Développement Piloté par les Tests* : commencer à écrire les tests avant de développer la fonctionnalité

Ce qu'il faut retenir...

Définitions

Tests unitaires Isole une brique applicative pour valider qu'elle remplisse son rôle, indépendamment de ses dépendances.

TDD *Développement Piloté par les Tests* : commencer à écrire les tests avant de développer la fonctionnalité

JUNIT Framework exécutant les tests unitaires

Ce qu'il faut retenir...
Définitions

Tests unitaires Isole une brique applicative pour valider qu'elle remplisse son rôle, indépendamment de ses dépendances.

TDD *Développement Piloté par les Tests* : commencer à écrire les tests avant de développer la fonctionnalité

JUNIT Framework exécutant les tests unitaires

FestAssert Facilite l'écriture des assertions

Ce qu'il faut retenir...
Définitions

Tests unitaires Isole une brique applicative pour valider qu'elle remplisse son rôle, indépendamment de ses dépendances.

TDD *Développement Piloté par les Tests* : commencer à écrire les tests avant de développer la fonctionnalité

JUNIT Framework exécutant les tests unitaires

FestAssert Facilite l'écriture des assertions

Mockito Facilite l'isolation des briques applicatives

Ce qu'il faut retenir...

Annotations

Annotations à retenir :

`@Test` Déclare un test sur une méthode public sans paramètres ni retour

Ce qu'il faut retenir...
Annotations

Annotations à retenir :

`@Test` Déclare un test sur une méthode public sans paramètres ni retour

`@InjectMocks` Remplace les dépendances par des *Mocks*

Ce qu'il faut retenir...
Annotations

Annotations à retenir :

`@Test` Déclare un test sur une méthode public sans paramètres ni retour

`@InjectMocks` Remplace les dépendances par des *Mocks*

`@Mock` Déclare un attribut comme étant un mock à injecter via l'annotation `@InjectMocks`

Ce qu'il faut retenir...
Annotations

Annotations à retenir :

`@Test` Déclare un test sur une méthode public sans paramètres ni retour

`@InjectMocks` Remplace les dépendances par des *Mocks*

`@Mock` Déclare un attribut comme étant un mock à injecter via l'annotation `@InjectMocks`

`@Spy` Espionne les appels effectués sur une brique applicative ; donne la possibilité de l'injecter comme un Mock

Ce qu'il faut retenir...
Methodes

Méthodes à retenir :

`assertThat` Début d'une assertion sur un objet (donné en paramètre)

Ce qu'il faut retenir...
Methodes

Méthodes à retenir :

`assertThat` Début d'une assertion sur un objet (donné en paramètre)

`when` Défini le comportement d'un *Mock*

Ce qu'il faut retenir...
Methodes

Méthodes à retenir :

`assertThat` Début d'une assertion sur un objet (donné en paramètre)

`when` Défini le comportement d'un *Mock*

`verify` Vérifie les appels qui ont été effectués sur un *Mock*.

Fin

Merci, des questions ?