École Polytechnique de Montréal

Département de génie informatique et génie logiciel

LOG3430 - Méthodes de test et de validation du logiciel

Guide pour l'utilisation de JUnit 5 et JaCoCo avec Eclipse

Préparé par :

Hiba Bagane

Automne 2018

JUnit 5

Description

JUnit est l'un des frameworks de tests unitaires les plus populaires pour Java. JUnit a été initialement développé par Erich Gamma, l'un des quatre auteurs "*Gang of Four*" du livre *Design Patterns* et Kent Beck, créateur de la méthode *eXtreme Programming*.

Installation

JUnit est intégré à l'environnement de développement Eclipse. Les versions récentes de l'IDE offrent un choix entre JUnit 3, 4 et 5 lors de la création d'un nouveau *Test Case*.

Création d'une classe de test JUnit 5 avec Eclipse

1 - Créer un nouveau package qui contiendra toutes vos classes de test.

2 - Ajoutez un nouveau JUnit Test Case.

3 - Sélectionner New JUnit Jupiter test (JUnit 5)

4 - Les méthodes de setUp et tearDown peuvent être générées automatiquement si sélectionnées.

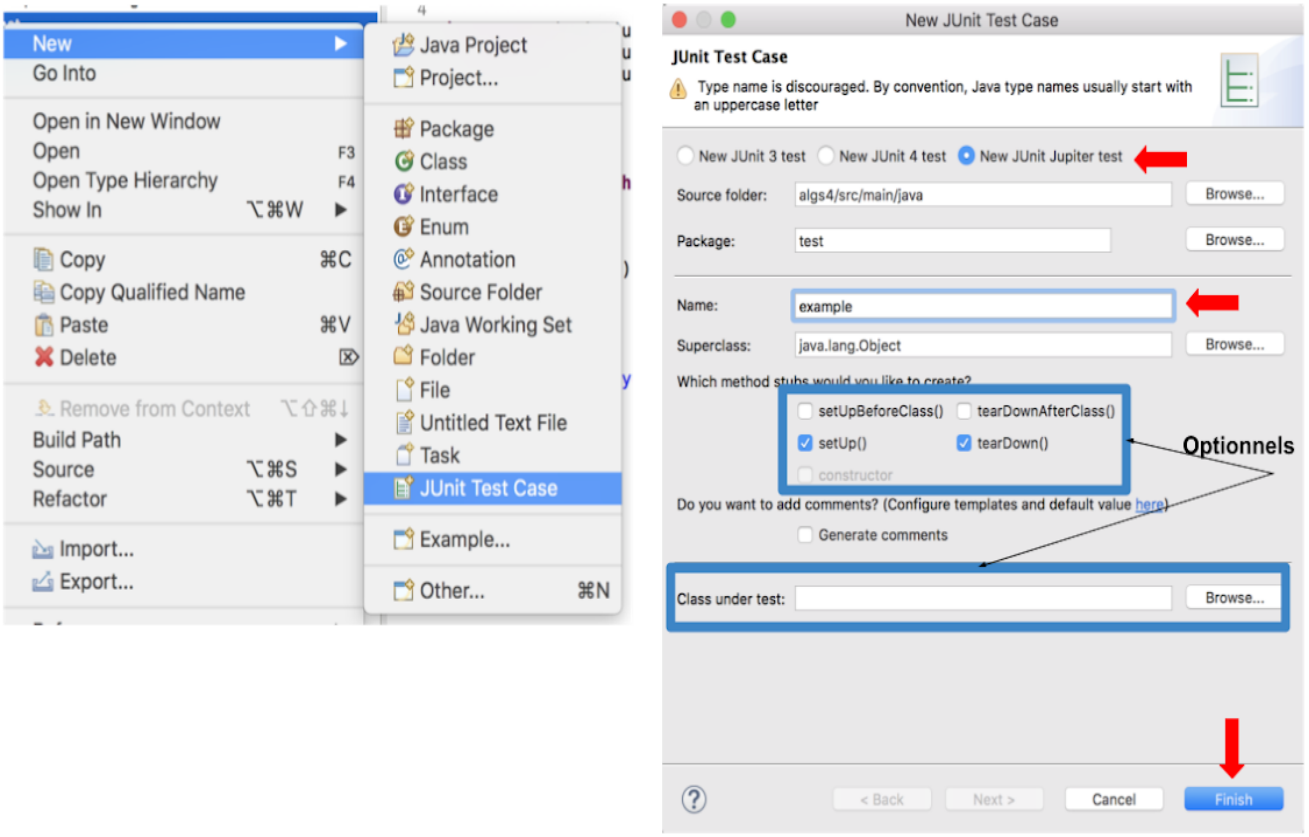

Figure 1 : Création d'une classe de test avec Eclipse

La classe de test générée est la suivante :

```java
import static org.junit.jupiter.api.Assertions.*;

import org.junit.jupiter.api.AfterEach;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

class Example {

    @BeforeEach
    void setUp() throws Exception {
    }

    @AfterEach
    void tearDown() throws Exception {
    }

    @Test
    void test() {
        fail("Not yet implemented");
    }

}
```

Figure 2 : Classe de test générée avec JUnit 5 et Eclipse

Exécution d'un test dans Eclipse

À partir du menu du fichier contenant votre classe de test ou le package qui contient toutes vos classes de test, choisir Run As -> JUnit Test.

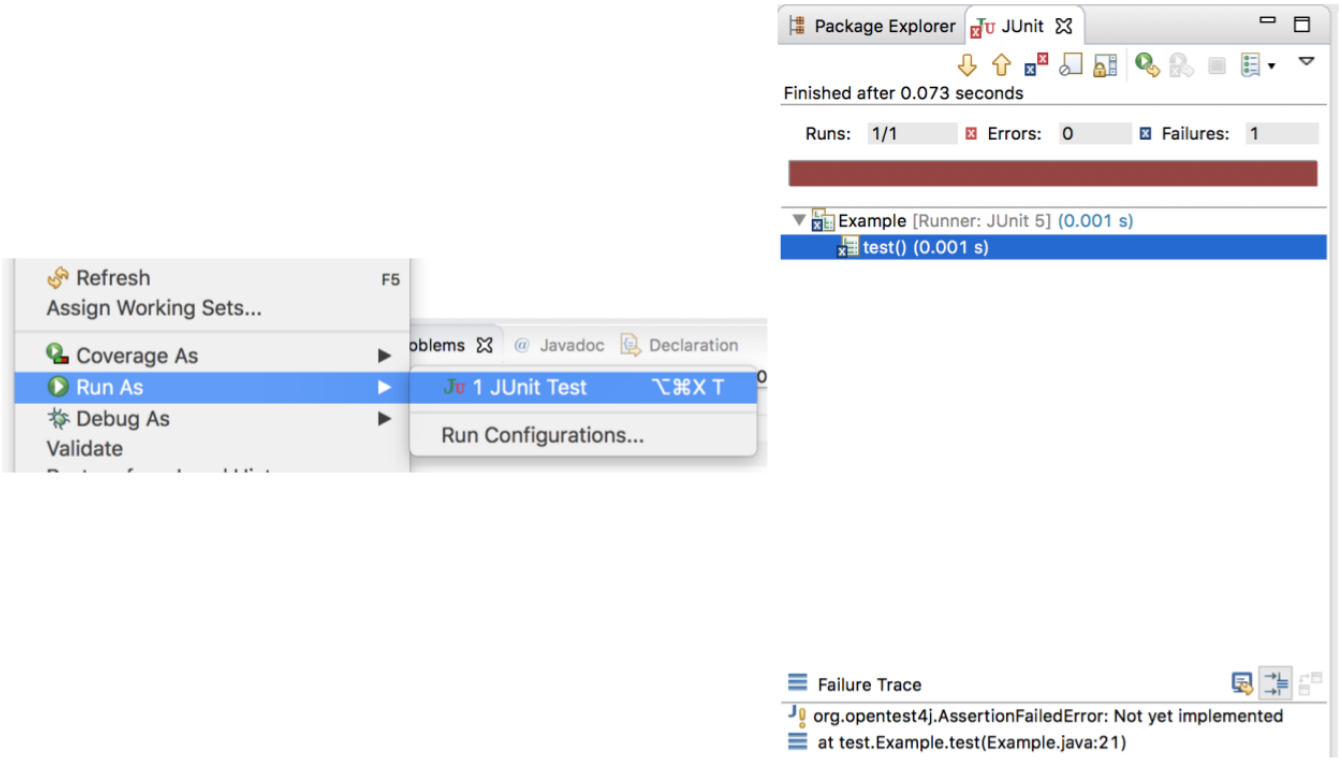

Figure 3 : Exécution de tests dans Eclipse

Documentation

Tableau 1 : Exemples d'annotations JUnit 5

Annotation	Description
@BeforeEach	Exécutée avant chaque méthode de test. Elle permet de préparer l'environnement de test (initialisation, lecture des données etc…)
@AfterEach	Exécutée après chaque méthode de test. Elle permet de nettoyer l'environnement de test (suppression des données temporaires, restauration des données par défaut etc …)
@BeforeClass	Exécutée une seule fois avant toutes les méthodes de test. Cette méthode peut donc être utilisée pour des initialisations communes à toutes les méthodes de test.
@AfterClass	Exécutée une seule fois après toutes les méthodes de test. Cette méthode peut donc être utilisée pour des activités de nettoyage.
@Ignore	Ignore la méthode de test correspondante durant l'exécution des tests. Elle est utile lorsque le code à tester a changé mais que le test n'a pas encore été mis à jour.
@Test	Indique que la méthode est une méthode de test.

Note: Toutes les annotations doivent être suivis par la méthode correspondante.

Tableau 2 : Exemples d'assertions JUnit 5

Assertion	Description
`fail(String message)`	Force l'échec d'un test avec le message donné.
`assertEquals(double expected, double actual, String message)`	Vérifie que la valeur attendue et la valeur actuelle sont égales.
`assertEquals(float expected, float actual, String message)`	Vérifie que la valeur attendue et la valeur actuelle sont égales.
`assertEquals(int expected, int actual, String message)`	Vérifie que la valeur attendue et la valeur actuelle sont égales.
`assertEquals(long expected, long actual, String message)`	Vérifie que la valeur attendue et la valeur actuelle sont égales.
`assertSame(Object expected, Object actual, String message)`	Vérifie que l'objet attendu et l'objet actuel sont des références au même objet.
`assertNotSame(Object unexpected, Object actual, String message)`	Vérifie que l'objet attendu et l'objet actuel ne sont pas des références au même objet.
`assertFalse(boolean condition, String message)`	Vérifie que la condition fournie est fausse.
`assertTrue(boolean condition, String`	Vérifie que la condition fournie est vraie.

`message)`	
`assertNotNull(Object actual, String message)`	Vérifie que l'objet fourni n'est pas nul.
`assertNull(Object actual, String message)`	Vérifie que l'objet fourni est nul.
`assertThrows(Class<T> expectedType, Executable executable,String message)`	Vérifie que `executable` lance une exception de type `expectedType`.
`assertDoesNotThrow(Executable executable, String message)`	Vérifie que `executable` ne lance pas une exception.

Note : Les messages dans les assertions doivent être expressifs afin d'aider à l'identification et à la résolution de problèmes.

Exceptions

Voici un exemple d'un test qui vérifie qu'une méthode lance une exception de type `UnsupportedOperationException`.Avec cette syntaxe, on évite de trainer des bloc try catch dans nos tests.

```
@Test
void shouldThrowExceptionExample() {
    Throwable exception = assertThrows(
         UnsupportedOperationException.class,
         () -> { throwsAnUnsupportedOperationException()});
    });
    assertEquals(exception.getMessage(), "Not supported");
}
```

La méthode ou la fonction sous test doit être appelée dans une lambda, sinon l'exception fera échouer le test. Les expressions Lambda ont été introduites à partir de Java 8. Pour plus d'informations, consultez le lien suivant: https://docs.oracle.com/javase/tutorial/java/javaOO/lambdaexpressions.html#syntax

Description

JaCoCo est un outil gratuit pour Eclipse, qui permet de calculer la couverture du code Java par les tests et générer des rapports de couverture.

Installation

1 - Allez au Marketplace d'Eclipse

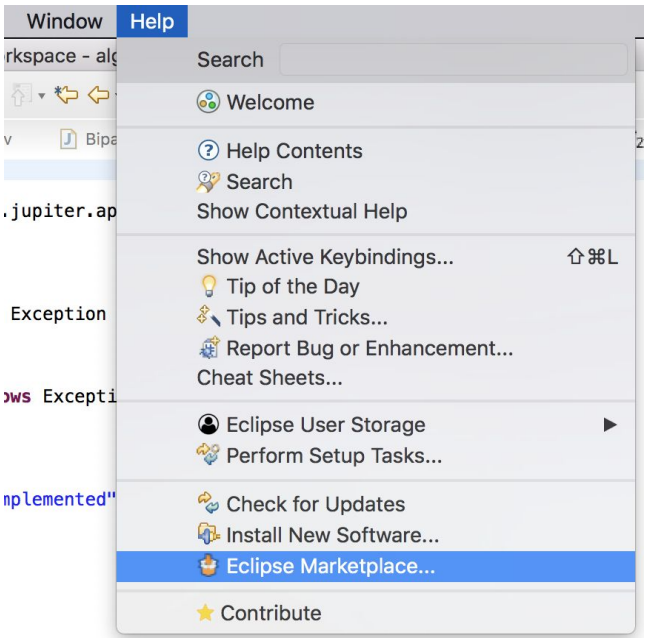

Figure 4 : Menu d'aide Eclipse

2 - Cherchez EclEmma ou eclemma, ensuite cliquez sur le bouton d'installation

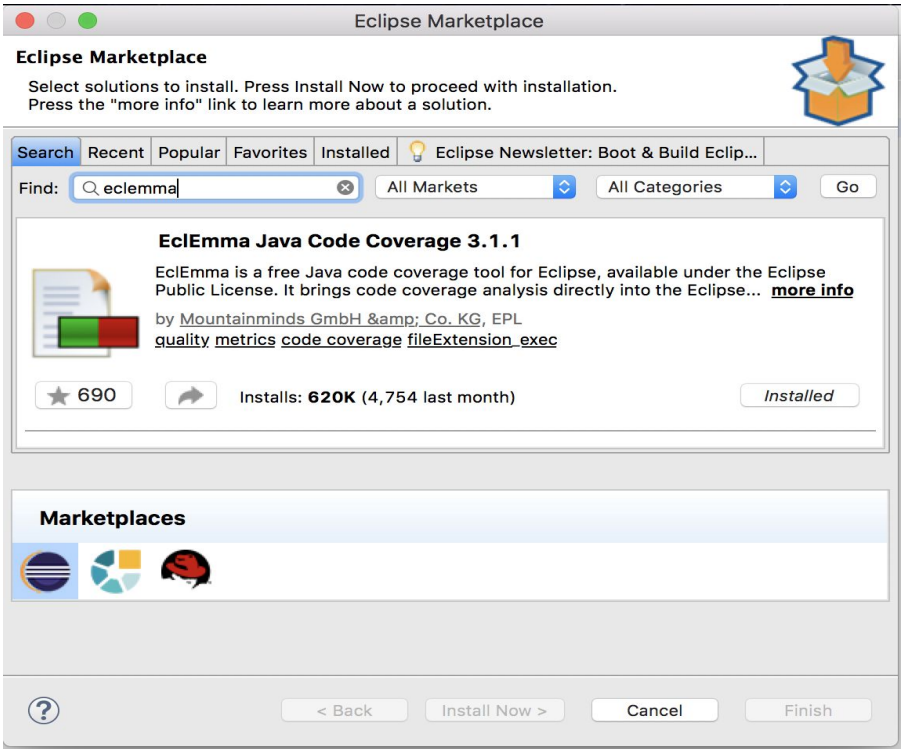

Figure 5 : Recherche dans Eclipse Marketplace

Utilisation

Sélectionner le fichier test ou votre package qui contient toutes les classes de test ensuite à partir du menu choisir Coverage As -> JUnit Test.

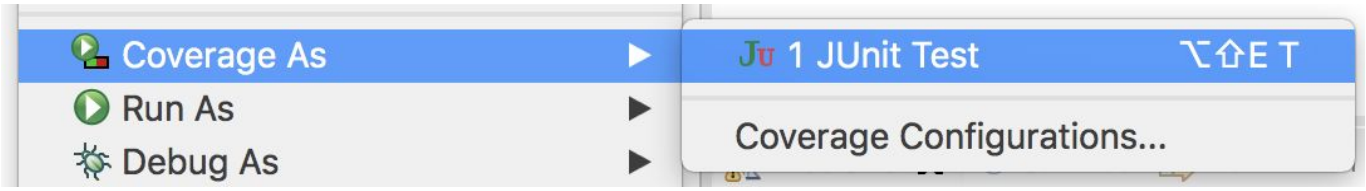

Figure 6 : Menu JaCoCo

Couleurs et indications :

```
package com.example.jacoco;

public class Rectangle {
    private int x;
    private int y;
    private int width;
    private int height;

    public Rectangle(int x, int y, int width, int height) {
        if (width <= 0 || height <= 0)
            throw new IllegalArgumentException("Dimensions are not positive");

        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }

    public boolean intersects(Rectangle other) {
        if (x + width <= other.x)
            return false;
        if (x >= other.x + other.width)
            return false;
        return (y + height > other.y && y < other.y + other.height);
    }
```

Figure 7 : Exemple de la couverture d'une classe avec JaCoCo

- **Rouge :** Instruction non couverte par les tests, c'est-à-dire toutes les branches de l'instruction ne sont pas parcourues par les tests.
- **Vert :** Instruction totalement couverte, c'est-à-dire toutes les branches de l'instruction sont parcourues par les tests.
- **Jaune :** Instruction partiellement couverte, c'est-à-dire certaines branches de l'instruction sont parcourues par les tests.

Générer les rapport de couvertures en format HTML

1 - Sélectionner votre projet dans la fenêtre Coverage ensuite choisir Export Session

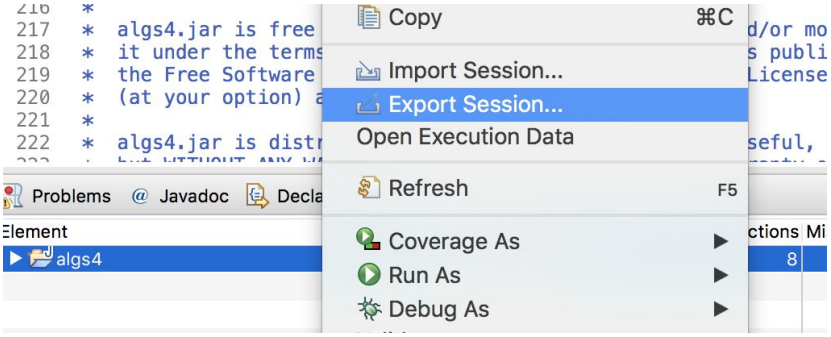

Figure 8 : Menu de génération des rapports de couverture avec JaCoCo

2 - Choisir le format HTML ainsi qu'un emplacement pour les fichiers qui seront générés

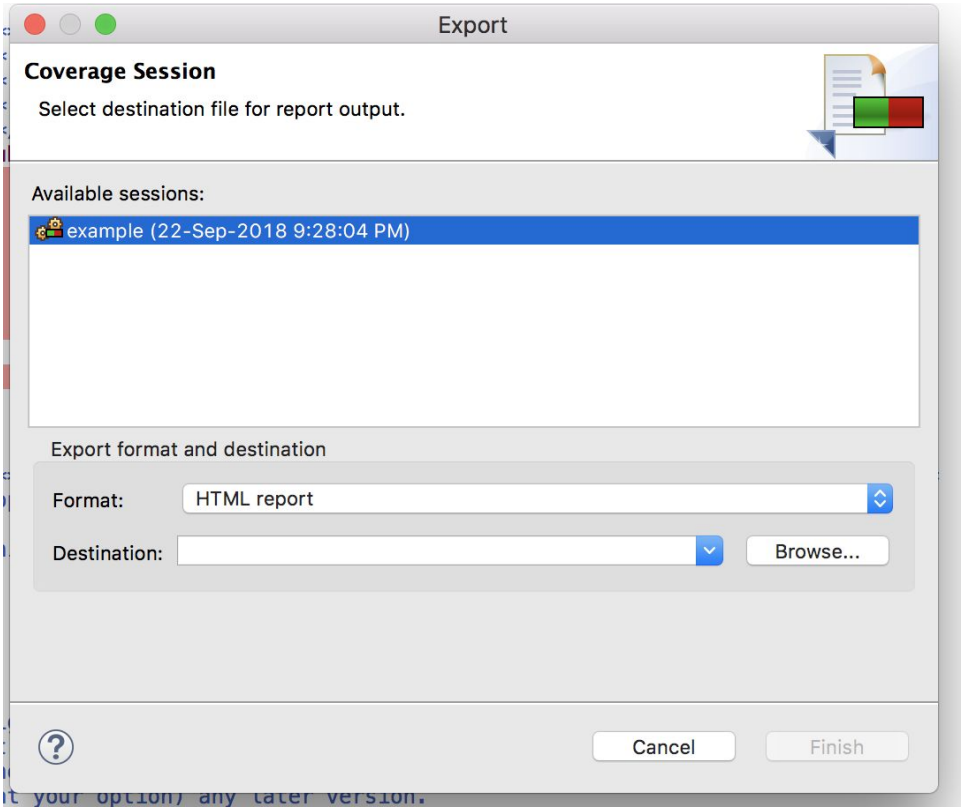

Figure 8 : Génération des rapports de couverture avec JaCoCo

Références :

JUnit 5 User Guide. JUnit. Disponible : https://junit.org/junit5/docs/current/user-guide/#writing-tests-assertions