JAVA
FONDAMENTAUX

Présenté par :

Xavier TABUTEAU

Qu’est ce que JUnit ?

JUnit est un framework open source servant à la création et l’exécution de tests unitaire automatisables. Ce type de tests est appelé tests unitaires de non-régression. L’intérêt de de ce framework est de s’assurer que le code est toujours bon si à un moment donné on doit le modifier.
Depuis la version 5 son nom à changé en Jupiter.

Le site de JUnit est : http://junit.org

Les tests se font à l’aide de classes qui sont situé dans un package de test au même niveau que les classes du projet. Les méthodes testées de la classe du projet doivent être suffixer par « test » dans la classe de test. Les tests doivent être indépendant les uns des autres. La classe de test a le nom de la classe à tester suffixé par « Test ».

JUnit peut être utilisé avec les différents type de projet.

Exemple de type de projet Java :

Apache Ant («Another Neat Tool») est une bibliothèque Java utilisée pour automatiser les processus de construction pour les applications Java.
Apache Maven est un outil de gestion des dépendances et d'automatisation de construction, principalement utilisé pour les applications Java.

Obtenir JUnit 5

Avec Eclipse :

Créer un projet, puis aller dans les propriétés du projet. Choisir « Java Build Path » dans la partie gauche de la fenêtre et sélectionner l'onglet librairies dans la partie de droite. Sélectionner ClassPath (ou ModulePath si on utilise les modules). Cliquer sur « Add librairies » et choisir JUNIT5. cliquer sur « Apply And Close ». JUnit5 est ajouter au projet.

Avec Netbeans :

Pour utiliser JUnit avec un projet « Java with ANT » il n'y a rien de spécial à faire.
Pour utiliser JUnit avec un projet « Java with Maven » il faut aller dans le fichier « pom.xml » situé à la racine du projet et l'éditer afin de modifié son contenu. Selon les IDE ce fichier n'est pas affiché au même endroit dans l'arborescence du projet.

Ce fichier permet d'importer les dépendances et les plugins nécessaire à l'exécution de JUnit 5.

Obtenir JUnit 5

Contenu nécessaire du fichier « pom.xml » :
(avant <dependencies> se sont les infos sur votre projet)

```
 <dependencies>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-api</artifactId>
            <version>5.10.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-params</artifactId>
            <version>5.10.1</version>
            <scope>test</scope>
        </dependency>
        <dependency>
            <groupId>org.junit.jupiter</groupId>
            <artifactId>junit-jupiter-engine</artifactId>
            <version>5.10.1</version>
            <scope>test</scope>
        </dependency>
```

Obtenir JUnit 5


```
        <dependency>
                <groupId>org.junit.platform</groupId>
                <artifactId>junit-platform-suite</artifactId>
                <version>1.10.1</version>
                <scope>test</scope>
        </dependency>
        <dependency>
                <groupId>org.junit.platform</groupId>
                <artifactId>junit-platform-console-standalone</artifactId>
                <version>1.10.1</version>
                <scope>test</scope>
        </dependency>
        <dependency>
                <groupId>org.junit.platform</groupId>
                <artifactId>junit-platform-reporting</artifactId>
                <version>1.10.1</version>
                <scope>test</scope>
        </dependency>
</dependencies>
```

Obtenir JUnit 5


```
 <build>
        <plugins>
            <plugin>
                <artifactId>maven-surefire-plugin</artifactId>
                <version>3.2.5</version>
            </plugin>
        </plugins>
    </build>
    <properties>
        <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
        <exec.mainClass>prj.junit5_maven.JUnit5_maven</exec.mainClass>
        <maven.compiler.release>21</maven.compiler.release>
    </properties>
</project>
```

Les tests unitaires

Les tests unitaires sont des tests permettant de tester des petites unités d'un programme. Ils sont souvent utilisés pour tester des méthodes.
Ce sont des tests qui doivent s'exécuter rapidement sans interaction avec une personne et être automatisable car il faut les exécuter dès qu'il y a un changement dans le code source.

JUnit est un ensemble de bibliothèques de tests unitaires. Historiquement, JUnit 4 a été la version la plus utilisée dans les projets, mais JUnit 5, disponible depuis fin 2017 (10 ans après Junit 4), est désormais la référence pour les nouveaux développements. Les principales versions de JUnit diffèrent dans leur fonctionnement et ne sont pas totalement compatibles entre elles.

Une fois un projet créer et une classe écrite en Java, on peut demander à l'IDE de créer une classe de test faisant référence a la classe du projet. On créer une classe de test par classe de projet.

Les tests unitaires se font à l'aide d'annotations et de méthodes de test prédéfinies commençant toutes par « assert ». Ces annotations sont nombreuses. Il est possible de les consulter sur le site officiel de JUnit. Les tests se font uniquement dans la classe de test. On ne touche pas au code en développement.

https://junit.org/junit5/docs/current/api/index.html

Les tests unitaires

Pour faire une méthode de test on utilise l'annotation @Test au dessus du nom de la méthode dans la classe de test.

A l'intérieur de la méthode de test on utilise des méthodes de la classe Assertion de JUnit pour faire les vérifications voulues.

Il existe une méthode particulière nommée fail(x) qui provoque l'échec de la méthode de test. X correspond au message à renvoyé dans l'IDE.
Pour désactiver une méthode de test on utilise l'annotation @Disabled(x). X est un message retourné à l'IDE.

Quelques méthodes courantes de vérification de la classe Assertions :

assertEquals(x, y) : x = y.
assertNotEquals(x, y) : x != y.
assertNull(x) : x est null.
assertNotNull(x) : x n'est pas null.
assertTrue(x) : l'expression x est vraie.
assertFalse(x) : l'expression x est fausse.
assertSame(x, y) : les deux objets sont égaux.
assertNotSame(x, y) : les deux objets ne sont pas égaux.
assertArrayEquals(x, y) : les deux tableaux sont égaux.

Les tests unitaires paramétrés

Il est possible de faire des tests paramétrés. Pour cela on utilise l'annotation @ParameterizedTest et @ValueSource qui prend un paramètre de type tableau de int, double, long ou string.

Syntaxe :

```
@ParameterizedTest
@ValueSource(ints = { 1, 2, 3, 18, …})
```

Il est aussi possible de passer par l'annotation @EnumSource. Elle accepte en paramètre « value » une classe de type enum . Il y a deux autres paramètres optionnels « names » pour indiquer des éléments et « mode » permettant de choisir le comportement.
mode prend 4 valeurs possibles.

INCLUDE (par défaut)	: Tests pour les valeurs de la liste passer dans « names ».
EXCLUDE	: Tests pour les valeurs de l'enum qui ne sont pas dans « names ».
MATCH_ALL	: Ne fournir que les éléments de l'énumération dont le nom correspond aux motifs fournis dans « names ».
MATCH_ANY	: Ne fournir que les éléments de l'énumération dont le nom correspond à un des motifs fournis dans l'attribut « names ».

Il existe d'autres sources, telle que @MethodeSource, @ArgumentsSource, …

Ordonner les tests unitaires

Il existe des annotations pour ordonner des tests (fixtures) :

@BeforeAll : exécute le test avant tous les autres.
@AfterAll : exécute le test après tous les autres.
@BeforeEach : s'exécute avant chaque test.
@AfterEach : s'exécute après chaque test.

Il est possible aussi de choisir dans quel ordre on veut exécuter les méthodes de test à l'aide de @Order(valeur). Les exécutions des tests se feront dans l'ordre ascendant de la valeur passer à @Order.

Pour que @Order fonctionne il faut aussi positionner l'annotation @TestMethodOrder (OrderAnnotation.class) au dessus de la classe de test.

Il est aussi possible de répéter n fois un test.
Il suffit de remplacer @Test par @RepeatedTest(n).

Les suites de tests

JUnit 5 permet de faire des suites de test qui sont une agrégation de multiples classes de tests qui pourrons être exécuté ensemble. Les classes peuvent provenir de package diffèrent mais elles doivent respecter les règles de visibilité.

Pour créer une suite de test on utilise l'annotation @Suite sur la classe qui fera la suite de tests. Il y a une série d'annotation utilisable pour ajouter ou enlever des classes de test unitaires à la classe de série de tests.

@ExcludeClassNamePatterns / @IncludeClassNamePatterns :
Précise avec une expression régulière le nom des classes à inclure ou exclure de la suite.

@ExcludePackages / @IncludePackages :
Exclut ou inclut les packages pour la suite des tests.

@ExcludeTags / @IncludeTags :
Précise des tags à exclure ou inclure pour les tests.

@SelectClasses : Indique quelles classes utilisées pour la suite de tests.
@SelectPackages : Indique quels packages utilisés pour les suites de tests.
@SuiteDisplayName : Affiche un nom personnalisé pour la classe de suite de tests sur le rapport généré par l'IDE.

Couverture de code

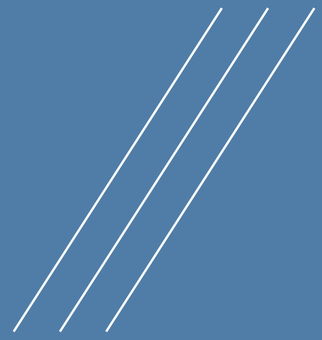
La couverture de code est une mesure utilisée pour décrire le taux de code source exécuté d'un programme quand une suite de test est lancée.

La couverture de code est intégrée à Eclipse.

Pour vérifier si c'est configurer, aller dans Help – Install New Software. Cliquer sur le bouton manage et vérifier que la ligne contenant « https://update.eclemma.org/ » est cochée.

Ensuite on fera bouton droit de la souris sur le fichier à tester avec JUnit, choisir « Coverage As » puis « JUnit Test ». Le test JUnit s'effectue et la couverture de code aussi. JUnit donne le résultat dans l'onglet console tandis que la couverture de code s'affichera dans l'onglet « Coverage ».

JAVA
FONDAMENTAUX



Présenté par
Xavier TABUTEAU