

🔑 Objectifs

- Écriture de tests unitaires avec JUnit
- Vérification du taux de couverture de test avec Jacoco

1 - Présentation

Comme vous savez le faire en Python avec les «asserts», il est également possible de réaliser des tests unitaires en Java avec l'outil JUnit. Ce support va donc vous détailler comment intégrer et utiliser l'outil JUnit afin de mener à bien votre campagne de tests.

Comme une bonne nouvelle n'arrive jamais seule, pour aller plus loin, nous verrons aussi comment mesurer la couverture de tests d'un projet JAVA en utilisant là encore un outil tel que JaCoCo (Java Code Coverage Library). C'est un outil populaire qui permet de générer des rapports de couverture de code, ce qui vous aide à identifier les parties du code qui ne sont pas couvertes par vos tests. Ce document vous guidera étape par étape pour intégrer JaCoCo dans votre projet JAVA.

Ce document s'appuie sur le projet JAVA du Pendu (TP n°5 IHM) qui n'utilise pas MAVEN ou GRADLE.

2 – Prérequis : une arborescence digne de ce nom



- **doc** : contient la JavaDoc
- **img** : contient les images du Pendu
- **lib** : contiendra les différentes bibliothèques à venir
- **out** : contient les fichiers .class générés lors de la compilation
- **report** : généré lors de l'exécution de JaCoCo, il est inexistant au départ de la création de votre projet.
- **src** : contient les sources :
 - de votre projet (src/main/java/com/cdal) : les fichiers .java doivent importer le package
`package main.java.com.cdal;`
 - de vos tests (src/test/java/com/cdal) : les fichiers .java doivent importer le package
`package test.java.com.cdal;`

On pourrait séparer Modèle / Vue / Contrôleur au sein de ces répertoires...

3 – Préparation du projet :

3.1 Téléchargez les archives ZIP de JaCoCo à partir du site officiel :

<https://github.com/jacoco/jacoco/releases/>

3.2 Extrayez les fichiers JAR nécessaires (jacocoagent.jar et jacococli.jar)

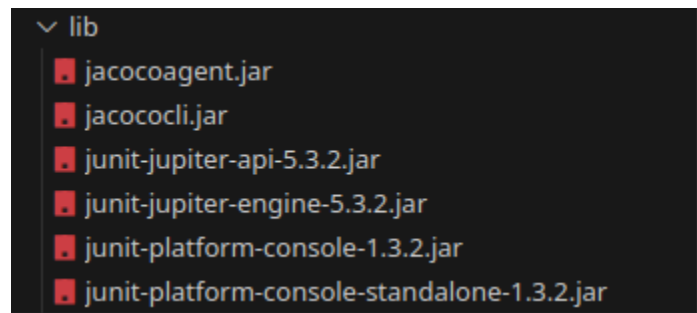
3.3 Déplacer ces fichiers JAR dans le répertoire lib/ de votre projet

3.4 Téléchargez les fichiers JAR de JUnit5 à partir du site officiel ou bien allez faire un tour sur votre machine usr/share/java/ et vous y trouverez votre bonheur !!

3.5 Copiez les fichiers JAR suivants :

- junit-jupiter-api-5.3.2.jar
- junit-jupiter-engine-5.3.2.jar
- junit-platform-console-1.3.2.jar
- junit-platform-console-standalone-1.3.2.jar

3.6 Coller ces fichiers JAR dans le répertoire lib/ de votre projet qui devrait ressembler à cela



4 – Écriture des tests

Je désire tester la classe MotMystere du projet Pendu dont voici la structure à gauche. Nous avons donc pas mal de tests à écrire pour valider le constructeur, les getters, les setters et les fonctions métiers de notre modèle.

Voici un exemple de tests avec le fichier MotMystereTest.java qui se trouvera dans src/test/java/com/cdal.

```

▼ STRUCTURE
  {} main.java.com.cdal
  ▼ MotMystere
    [ ] FACILE
    [ ] MOYEN
    [ ] DIFFICILE
    [ ] EXPERT
    [ ] motATrouver
    [ ] niveau
    [ ] motCrypte
    [ ] lettresEssayees
    [ ] nbLettresRestantes
    [ ] nbEssais
    [ ] nbErreursRestantes
    [ ] nbErreursMax
    [ ] dict
    [ ] MotMystere(String, int, int)
    [ ] MotMystere(String, int, int, int, int)
    [ ] initMotMystere(String, int, int) : void
    [ ] getMotATrouver() : String
    [ ] getNiveau() : int
    [ ] setMotATrouver(String) : void
    [ ] setMotATrouver() : void
    [ ] setNiveau(int) : void
    [ ] getMotCrypte() : String
    [ ] getLettresEssayees() : Set<String>
    [ ] getNbLettresRestantes() : int
    [ ] getNbEssais() : int
    [ ] getNbErreursMax() : int
    [ ] getNbErreursRestantes() : int
    [ ] perdu() : boolean
    [ ] gagne() : boolean
    [ ] essaiLettre(char) : int
    [ ] toString() : String
  
```

```

package test.java.com.cdal;

import static org.junit.jupiter.api.Assertions.assertEquals;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import main.java.com.cdal.MotMystere;

public class MotMystereTest {

    private MotMystere mot;

    @BeforeEach
    public void setUp() {
        mot = new MotMystere(motATrouver:"TESTABILITE", MotMystere.FACILE, nbErreursMax:10);
    }

    @Test
    public void testgetMotATrouver() {
        assertEquals("TESTABILITE", mot.getMotATrouver());
    }

    @Test
    public void testgetNiveau() {
        assertEquals(0, mot.getNiveau());
    }

    @Test
    public void testgetNbEssais() {
        assertEquals(0, mot.getNbEssais());
    }

    @Test
    public void testgetNbErreurMax() {
        assertEquals(10, mot.getNbErreursMax());
    }

    @Test
    public void testgetLettresRestantes() {
        assertEquals(9, mot.getNbLettresRestantes());
    }
}
  
```

5 – Explication de texte

Nous allons faire un petit focus sur le fichier MotMystereTest.java

Je passe volontiers les différents imports nécessaire à la bonne compilation de ce fichier. Notez toutefois que le fichier de test est dans un package et que la classe à tester est dans un autre !

Focalisons-nous sur le contenu de la classe MotMystereTest:

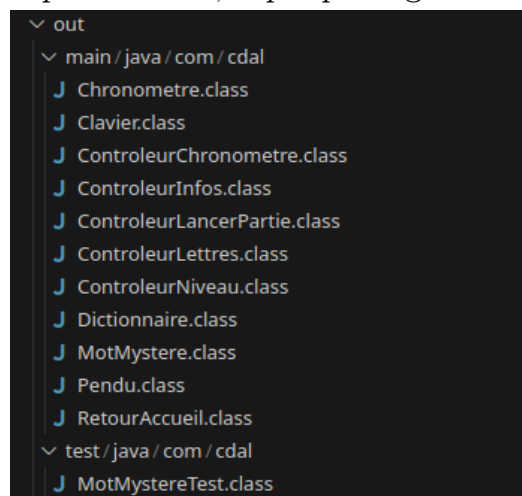
- Elle a un attribut privé de type MotMystere: cela tombe bien, c'est la classe à tester !
- Elle possède une méthode publique **setUp()** précédée par une annotation **@BeforeEach**: autrement dit, cette fonction publique **setUp()** sera appelée avant chaque test unitaire. Nous aurons donc le mot TESTABILITE à trouver en mode facile avec 10 erreurs autorisées
- Elle possède plusieurs méthodes publiques précédées par l'annotation **@Test** : ce sont nos tests unitaires. Par convention, les méthodes commencent par le mot test + le nom de la fonction à tester. Mais cela vous avez l'habitude... L'appel à la fonction **assertEquals** permet de vérifier le résultat attendu et le résultat obtenu par l'appel de la fonction à tester, appliqué à notre attribut privé de type MotMystere

6 – Compilation et lancement des tests

6.1 Assurez vous que les fichiers sources et les fichiers de tests sont compilés. Par exemple, pour compiler avec **javac** depuis le répertoire Pendu, en précisant les options nécessaire pour **javafx**, les **librairies** nécessaires, les fichiers **sources** et les fichiers de **tests** :

```
javac --module-path /usr/share/openjfx/lib/ --add-modules javafx.controls -d out -cp lib/junit-jupiter-api-5.3.2.jar:lib/junit-jupiter-engine-5.3.2.jar:lib/junit-platform-console-standalone-1.3.2.jar src/main/java/com/cdal/*.java src/test/java/com/cdal/*.java
```

Ceci va créer les fichiers .class dans le répertoire out., et par package s'il vous plaît!!



6.2 Utilisez JaCoCo pour instrumenter les tests et collecter les données de couverture, depuis le répertoire Pendu avec la commande suivante:

```
java --module-path /usr/share/openjfx/lib/ --add-modules javafx.controls
-javaagent:lib/jacocoagent.jar=destfile=jacoco.exec -cp out:lib/junit-jupiter-api-5.3.2.jar:lib/junit-jupiter-
engine-5.3.2.jar:lib/junit-platform-console-standalone-1.3.2.jar
org.junit.platform.console.ConsoleLauncher --scan-class-path --class-path out
```

Quelques explications :

- `-javaagent:lib/jacocoagent.jar=destfile=jacoco.exec` : Utilise JaCoCo agent pour collecter les données de couverture et les enregistre dans jacoco.exec.
- `-cp out:lib/junit-platform-console-standalone-1.3.2.jar` : Définit le classpath pour inclure les classes compilées et le JAR de JUnit contenant ConsoleLauncher.
- `org.junit.platform.console.ConsoleLauncher --class-path out --scan-class-path` : Exécute les tests en scannant le classpath pour les classes de test.

En plus de la génération du fichier jacoco.exec à la racine du répertoire Pendu, le terminal nous affiche un joli résultat, du moins en apparence...

```
JUnit Jupiter ✓
└─ MotMystereTest ✓
   └─ testgetNbErreurMax() ✓
      └─ testgetMotATrouver() ✓
         └─ testgetNiveau() ✓
            └─ testgetLettresRestantes() ✓
               └─ testgetNbEssais() ✓
JUnit Vintage ✓

Test run finished after 216 ms
[ 3 containers found ]
[ 0 containers skipped ]
[ 3 containers started ]
[ 0 containers aborted ]
[ 3 containers successful ]
[ 0 containers failed ]
[ 5 tests found ]
[ 0 tests skipped ]
[ 5 tests started ]
[ 0 tests aborted ]
[ 5 tests successful ]
[ 0 tests failed ]
```

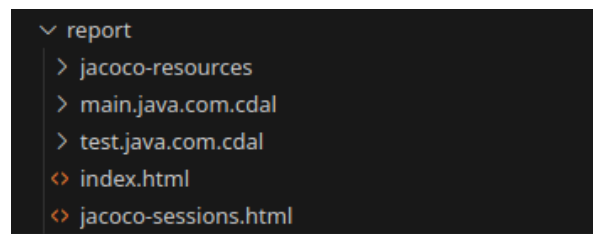
7 – Couverture des tests

Du moins en apparence...Humm... Pourquoi tant de haine ?
Générons le rapport de couverture pour mieux comprendre !




Utilisez JaCoCo pour analyser le fichier jacoco.exec et générer un rapport de couverture avec la commande suivante depuis le répertoire Pendu :

```
java -jar lib/jacococli.jar report jacoco.exec --classfiles out --sourcefiles src/main/java --html report
```

Cela génère le répertoire report avec tout un contenu intéressant.







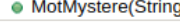
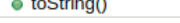
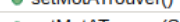

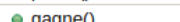

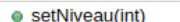

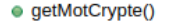

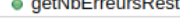
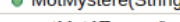
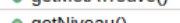
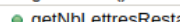
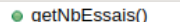
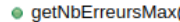


Rendez vous sur la page index.html ! Pas brillant n'est ce pas avec 8 % de couverture de tests

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
main.java.com.cdal		8 %		10 %	88	97	264	292	54	62	9	11
test.java.com.cdal		100 %	n/a	n/a	0	7	0	13	0	7	0	1
Total	1 265 of 1 431	11 %	57 of 64	10 %	88	104	264	305	54	69	9	12

Si vous cliquez sur le détail du package main.java.com.cdal (votre package quoi!), puis sur la classe MotMystere, vous verrez que vous avez encore beaucoup de travail !! Les deux premières colonnes sont vos indicateurs privilégiés.

MotMystere

Source file "main/java/com/cdal/MotMystere.java" was not found during generation of report.

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
essaiLettre(char)		0 %		0 %	5	5	13	13	1	1
initMotMystere(String, int, int)		76 %		58 %	5	7	5	23	0	1
MotMystere(String, int, int, int, int)		0 %	n/a	n/a	1	1	5	5	1	1
toString()		0 %	n/a	n/a	1	1	1	1	1	1
setMotATrouver()		0 %	n/a	n/a	1	1	2	2	1	1
setMotATrouver(String)		0 %	n/a	n/a	1	1	2	2	1	1
perdu()		0 %		0 %	2	2	1	1	1	1
gagne()		0 %		0 %	2	2	1	1	1	1
setNiveau(int)		0 %	n/a	n/a	1	1	2	2	1	1
getMotCrypte()		0 %	n/a	n/a	1	1	1	1	1	1
getLettresEssayees()		0 %	n/a	n/a	1	1	1	1	1	1
getNbErreursRestants()		0 %	n/a	n/a	1	1	1	1	1	1
MotMystere(String, int, int)		100 %	n/a	n/a	0	1	0	3	0	1
getMotATrouve()		100 %	n/a	n/a	0	1	0	1	0	1
getNiveau()		100 %	n/a	n/a	0	1	0	1	0	1
getNbLettresRestantes()		100 %	n/a	n/a	0	1	0	1	0	1
getNbEssais()		100 %	n/a	n/a	0	1	0	1	0	1
getNbErreursMax()		100 %	n/a	n/a	0	1	0	1	0	1
Total	169 of 284	40 %	17 of 24	29 %	22	30	35	61	11	18

8 – Conclusion







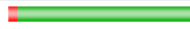
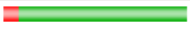

















Un taux de couverture de tests de 80 % est le minimum.

Alors, développez vos projets, codez vos tests, compilez, testez et collectez les données de couverture tout au long de la durée de vie du projet.

Vous avez les commandes, alors automatisez tout cela dans un joli script et présentez vos résultats.

Tous les moyens sont bons pour montrer votre rigueur de développement et vos bonnes pratiques.

Bon j'y retourne, j'ai encore un peu de travail. Le modèle est bien testé à la fin de la rédaction de ce document. Mais il me reste les contrôleurs et les vues et c'est un autre sujet...

 MotMystere	 82 %	 87 %		
Element	Missed Instructions	Cov.	Missed Branches	Cov.
● MotMystere(String, int, int, int, int)		0 %		n/a
● toString()		0 %		n/a
● setMotATrouver()		0 %		n/a
● initMotMystere(String, int, int)		95 %		91 %
● perdu()		71 %		50 %
● essaiLettre(char)		100 %		87 %
● MotMystere(String, int, int)		100 %		n/a
● setMotATrouver(String)		100 %		n/a
● gagne()		100 %		100 %
● setNiveau(int)		100 %		n/a
● getMotATrouve()		100 %		n/a
● getNiveau()		100 %		n/a
● getMotCrypte()		100 %		n/a
● getLettresEssayees()		100 %		n/a
● getNbLettresRestantes()		100 %		n/a
● getNbEssais()		100 %		n/a
● getNbErreursMax()		100 %		n/a
● getNbErreursRestants()		100 %		n/a
Total	50 of 284	82 %	3 of 24	87 %