

Tests unitaires en boîte blanche

1.1 Ecriture de test

Dans ce tp il s'agit de l'utilisation d'IntelliJ et JUnit, conjointement avec Gradle pour l'automatisation de la production logicielle. En utilisant JUnit, des classes de test peuvent être créées, où les cas de test sont des méthodes annotées avec `@Test`. Des annotations supplémentaires telles que `@BeforeAll`, `@AfterAll`, `@BeforeEach`, et `@AfterEach` sont disponibles pour gérer les initialisations et les cas particuliers avant et après l'exécution des tests.

1.2 Utilisation de Junit

- Lorsqu'on lance la méthode de test **testGetterImaginary** le test se finit bien sans erreur.
- Et lorsqu'on lance l'ensemble des test de la classe **ComplexTest**, on constate qu'il y a certains tests qui ont échoué.

Pour que l'ensemble de test se termine sans erreur, on peut modifier la classe de test en ajoutant **@Disabled** juste avant la méthode qu'on veut ignorer. J'ai donc ajouté **@Disabled** aux test qui ont posé problème: **testZeroTrue**, **testProductReal**, **testProductImaginary**.

Pour tester **@BeforeAll**, **@AfterAll**, **@BeforeEach** et **@AfterEach**, j'ai rajouté des instructions d'impressions **System.out.print** visible dans la classe de test **ComplexTest**.

Afin de simplifier les tests on peut mutualiser le code de création des objets en l'insérant dans le code de **@BeforeEach**. Ces modifications sont disponibles dans le code en regardant **les lignes commentées**

On peut utiliser l'outil gradle pour lancer tous les tests. Avec la commande **gradle test** en ligne de commande on a **16 tests exécutés, 3 échecs et 1 ignorés** et ceci exécuté en **4 secondes**.

Partie Complex

Testons maintenant la méthode **inverse** de la classe Complex.

Pour cela ajoutons les tests: **testIsRealPartOfInverseIsCorrect** qui assure que la partie réelle de l'inverse est celle attendue,

testIsImaginaryPartOfInverseIsCorrect qui certifie que la partie imaginaire donnée par la méthode inverse est réellement correcte et **testInverseException** qui lève une exception lorsque le nombre dont on veut l'inverse est nul.




Ensuite, corrigeons le code de la méthode **product**: c'est la ligne commentée à ce niveau dans le code. On teste maintenant cette méthode avec deux tests qui échouaient à savoir: **testProductReal** et **testProductImaginary**. Se référer aux lignes commentées dans le code pour avoir la correction de ces tests.

Passons à **infinite**. pour que l'exécution ne termine pas, il suffit de rajouter une boucle **while (true)** dans le code de **infinite**. Lorsqu'on test cette méthode, on constate que le test ne finit pas. J'ai donc rajouté une ligne de code pour que le test échoue si infinite ne termine pas au bout de 100ms. Et le test se termine bien.

Calcul de la couverture par JaCoCo

1. Les couleurs concernant JaCoCo sont associées aux résultats de la couverture de code lors de l'exécution des tests.
La couleur **verte** est une couleur positive qui montre que le code associé a été exécuté lors des tests et qu'il a été pris en compte dans la couverture de code.
Le **rouge** indique qu'il y a des parties du code qui n'ont pas été exécutées lors des tests.
Le **jaune** signifie qu'une partie du code a été exécutée et une autre non.
2. Les taux de couverture d'une classe représentent à quel point les tests ont couvert la classe. Le taux de couverture des branches représentent les conditions qui ont été exécutées parmi toutes les conditions.

Partie Palindrome:

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
● Palindrome()		0 %	n/a		1 1	1 1	1 1
● isPalindrome(String)		100 %		100 %	0 4	0 7	0 1
Total	3 of 35	91 %	0 of 6	100 %	1 5	1 8	1 2

Selon le fichier html généré, on voit bien que la méthode isPalindrome est testée complètement à 100% pour les branches et les instructions. Donc il est possible de couvrir toutes les instructions de la classe Palindrome. Lorsqu'on enlève certains tests en utilisant **@Disabled** on remarque que la couverture des tests diminue.

Partie PartialCoverage:

J'ai créé une classe de test pour la seule méthode de partial de partial coverage. On constate avec le fichier html généré, on ne peut pas couvrir toutes les instructions de cette classe. Ceci s'explique par le fait qu'on ne rentre pas dans la boucle car i est toujours supérieur a y. Avec n'importe quelle valeur de x et y le résultat est toujours **zéro**. Cette méthode n'a pas tellement de sens. Il est impossible de couvrir toutes les instructions de cette classe.