TD Méthodes et Outils
pour La Conception Logicielle

Charbel Daoud

L'objectif de ce TD est de manipuler les technologies de construction, de test et d'intégration continue au sein d'un projet Java. Pour cela Maven, Junit5, Mockito ainsi que Github Actions seront utilisés. Le code a été volontairement buggé, vous devrez donc trouver ces bugs à l'aide de vos tests unitaires et fonctionnels.

Un projet simulant une machine à café sera la base de ce TD, celui- ci est disponible ici: https://github.com/NinjaCoder8/cours-qualite-2025

La quasi-totalité des méthodes de ce projet a été commentée par de la Javadoc afin de s'en servir comme "spécification" pour écrire les tests unitaires de ces méthodes. Vous trouverez à la racine du projet 2 diagrammes UML, générés à partir du code afin que vous ayez la vision de l'architecture du projet.

Mise en place de l'environnement de développement:

1. Disposer d'un JDK18 ou supérieur et de Maven sur votre machine
2. Faire un fork sur votre GitHub du projet de la machine à café
3. Cloner ce projet dans votre IDE préféré (de préférence IntelliJ IDEA)
4. Ouvrir un terminal à la racine du projet puis exécuter un: *mvn clean install*. Cette étape peut également être réalisée à l'aide du Maven installé dans votre IDE. (la sortie risque d'être "build failed", c'est normal des tests unitaires échoués)
5. Déployer votre projet sur Github en utilisant Github Actions

I - Refactoring

La première étape consiste à refactorer le code. En effet, le projet actuel se présente sous la forme d'un seul et même module avec des fonctionnalités souvent éloignées. Le package "storage" est relativement différent du package "machine". À des fins de maintenabilité, il est donc bon de créer un nouveau module Maven en y migrant l'ensemble du package "storage" à l'intérieur. Ce refactoring va impliquer un certain nombre de choses: faire attention aux dépendances entre les modules, regarder s'il est possible de factoriser des dépendances mutuelles entre les deux modules, etc. Une fois ce refactoring effectué, n'oubliez pas de lancer la commande "mvn clean install". Si le build Maven est un succès, alors committer et pusher sur GitHub. Le build devrait se lancer automatiquement sur GitHub.

II- Tests Unitaires et Couverture de Code

Nous allons maintenant créer des tests unitaires avec JUnit 5, puis examiner la couverture de notre code à l'aide de JaCoCo. Pour rappel, JaCoCo est un outil permettant d'examiner sous différents angles (lignes, méthodes et classes) la couverture du code par des tests unitaires. Un taux de couverture de code idéal par classes se situe aux alentours de 80%. Le fait de tester les getters et les setters est assez débattu. Créer des tests pour ces derniers vous permet d'augmenter le taux de couverture, mais pour un coût non négligeable et une utilité limitée. Cela est d'autant plus vrai que ceux-ci sont souvent générés avec un IDE et que donc les erreurs sont minimisées.

Nous souhaitons ici couvrir le maximum de méthodes des classes situées dans le package "machine" en excluant les getters et setters. Pour cela, il vous faudra utiliser la documentation JUnit 5 et Mockito, ainsi que les tests déjà réalisés, afin de vous en inspirer. La Javadoc créée au-dessus des méthodes est à utiliser comme "spécification" des méthodes afin d'écrire vos tests et de détecter ou non la présence de bugs.

Lors de la conception de vos tests unitaires, n'oubliez pas de tester les cas nominaux, mais aussi d'essayer les non nominaux (valeurs hors plages, valeurs null, génération d'exceptions, etc.) afin de couvrir l'ensemble des branches conditionnelles. Dans le cas où vous trouverez des bugs, effectuez les corrections nécessaires dans le code de la machine à café afin que vos tests passent au vert.

III- Tests Fonctionnels avec Cucumber

Cucumber est un framework de tests fonctionnels, c'est-à-dire des tests permettant de vérifier les fonctionnalités de l'application du point de vue de l'utilisateur. Cucumber permet de rédiger des tests en langage naturel à l'aide de fichiers *.feature* et d'une syntaxe Gherkin. Cette approche de test est connue sous le nom de Développement Piloté par le Comportement (BDD).

Pour cela, trois éléments sont indispensables :

- Un fichier .feature décrivant le test
- Un fichier de test intermédiaire permettant de lier l'exécution du test au fichier .feature décrivant le test. Ce fichier est également appelé "glue" en test logiciel.
- Un fichier implémentant les étapes du test fonctionnel

Accédez au dossier "test/java/fr.imt.coffee.machine" du module Maven de la machine à café. Vous y trouverez les deux fichiers de test permettant de lancer et exécuter les tests Cucumber.

Dans le fichier “CoffeeMachineCucumberFunctionalTest.java”, veuillez supprimer l'annotation *@Ignore* qui permet d'ignorer les tests fonctionnels lors de la phase de test de l'application par Maven.

Lancez ensuite la commande "mvn clean install" et observez les résultats. Si votre construction est un "success" et qu'il n'y a pas d'erreurs d'assertions remontées par Cucumber, il y a de fortes chances que vous ayez réussi à corriger un bug grâce à vos tests unitaires. Dans le cas contraire, il vous faudra identifier et résoudre ce bug.

Essayez d'implémenter un ou plusieurs scénarios de test dans le fichier "make a coffee.feature" avec Cucumber. Vous pouvez, par exemple, tenter de tester un cas très simple, à savoir créer une machine à café, la brancher, puis vérifier que la machine est correctement raccordée au réseau électrique. Ensuite, vous pourrez créer un scénario de test plus complexe, comme tester la machine à expresso.

Essayez d'implémenter un ou plusieurs scénarios de test dans le fichier "make a coffee.feature" avec Cucumber. Par exemple, testez un cas très simple où vous créez une machine à café, la branchez, puis vérifiez que la machine est correctement raccordée au réseau électrique. Ensuite, vous pouvez créer un scénario de test plus complexe, comme tester la machine à expresso.

Documentation

- Mockito: https://javadoc.io/doc/org.mockito/mockito-core/latest/org/mockito/Mockito.html
- JUnit 5: https://junit.org/junit5/docs/current/user-guide/
- Hamcrest: http://hamcrest.org/JavaHamcrest/
- Cucumber: https://docs.cucumber.io/
- Maven: https://maven.apache.org/guides/index.html
- Github Actions: https://docs.github.com/en/actions

Commandes de survie Maven

- *mvn clean install:* compile, test, package
- *mvn clean test:* compile, test
- *mvn clean install -DskipTests=True*: fait un mvn clean install sans exécuter les tests
- *mvn javadoc:javadoc*: génère la javadoc
- *mvn clean site:* compile, test, package puis génère la javadoc