UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

JAVA CLEAN CODE Partie 4

Développement TDD

- Rappel des frameworks java pour les Test.
- Les paradigmes du TDD.
- Les bonnes pratiques du TDD.

Développement TDD

Rappel des frameworks java pour les Test

- En Java, plusieurs frameworks sont disponibles pour vous aider à créer et à gérer des tests unitaires, d'intégration et fonctionnels.
- **JUnit** : C'est le framework de test le plus populaire pour Java. Il est utile pour les tests unitaires et peut être intégré à des outils d'automatisation de construction comme Maven et Gradle.
- **Mockito** : Il s'agit d'un framework de simulation (mocking) populaire en Java. Mockito est souvent utilisé en combinaison avec JUnit. Il vous permet de créer et de gérer des objets fictifs (mocks) pour simuler le comportement de classes et d'interfaces complexes.

Développement TDD

Les paradigmes du TDD

- Le Test Driven Development (TDD), ou Développement Dirigé par les Tests, est une méthode de développement de logiciel qui encourage l'écriture de tests avant l'écriture du code de production. En Java, spécifiquement avec le framework Spring, le TDD suit généralement ce processus :
- **Écrire un test** : Vous commencez par écrire un test pour la nouvelle fonctionnalité que vous voulez implémenter. À ce stade, le test échouera, car vous n'avez pas encore écrit le code de production.
- **Exécuter tous les tests et voir si le nouveau test échoue** : Ceci est fait pour s'assurer que le nouveau test ne passe pas par accident. C'est une étape importante pour valider que le test est bien écrit et teste la bonne chose.
- **Écrire le code de production** : Vous écrivez maintenant le code qui fera passer le test. À ce stade, vous ne vous concentrez que sur le fait de faire passer le test et non sur la propreté ou l'efficacité du code.

Développement TDD

Les paradigmes du TDD

- **Exécuter les tests** : Vous exécutez maintenant tous les tests pour vous assurer que le nouveau code de production passe le nouveau test et que tous les autres tests passent toujours.
- **Refactoriser le code** : Si les tests passent, vous pouvez alors refactoriser le code de production pour l'améliorer tout en gardant les mêmes fonctionnalités. L'objectif est d'améliorer la structure du code sans changer son comportement. Après la refactorisation, vous exécutez à nouveau les tests pour vous assurer qu'ils passent toujours.

Développement TDD

Les bonnes pratiques du TDD

- **Comprendre les exigences** : Avant d'écrire vos tests, vous devez avoir une compréhension claire de ce que le code doit réaliser. Cela comprend la compréhension des exigences fonctionnelles et non fonctionnelles.
- **Garder les tests simples** : Chaque test doit être indépendant et tester une seule fonctionnalité ou un seul comportement. Cela signifie que vous devriez vous efforcer de garder vos tests aussi simples et spécifiques que possible.
- **Ne pas anticiper les fonctionnalités futures** : Lorsque vous écrivez vos tests, concentrez-vous sur la fonctionnalité actuellement requise, pas sur ce que vous pensez qu'elle pourrait être requise à l'avenir. C'est ce qu'on appelle le principe YAGNI (You Aren't Gonna Need It).
- **Écrire le code juste nécessaire pour passer le test**: Après avoir écrit un test, écrivez le code de production minimum nécessaire pour passer le test. Ne vous inquiétez pas de la perfection à ce stade, il suffit que le code passe le test.

Développement TDD

Les bonnes pratiques du TDD

- **Refactorisation régulière** : Une fois que votre code passe les tests, prenez le temps de le refactoriser. Cela signifie le rendre plus lisible, le simplifier, éliminer les duplications, etc. N'oubliez pas de réexécuter vos tests après chaque refactorisation pour vous assurer que rien n'a été brisé.
- **Faire des tests automatisés** : Les tests doivent être automatisés afin qu'ils puissent être exécutés à chaque modification du code. Cela vous aidera à attraper les bugs tôt et facilitera l'intégration continue.
- **Mise en place des doubles de test (test doubles)** : Utilisez des bouchons (stubs), des mocks et des objets factices (dummy objects) pour isoler le code que vous testez. Cela vous permet de concentrer vos tests sur le code que vous écrivez, plutôt que sur ses dépendances.
- **Exécuter les tests régulièrement** : Les tests doivent être exécutés régulièrement, idéalement à chaque changement du code, pour s'assurer que tout fonctionne toujours comme prévu.

Développement TDD

Les bonnes pratiques du TDD

- **Maintenir une bonne couverture de test** : Toutes les parties du code doivent être testées. Vous devriez viser une haute couverture de code par les tests, bien que 100% ne soit pas toujours réaliste ou nécessaire.
- **Tester aux différents niveaux** : Il ne suffit pas de faire des tests unitaires, pensez également aux tests d'intégration, aux tests système, aux tests d'acceptation, etc

Développement TDD

FIRST

F - **Fast (Rapide)** : Les tests doivent être rapides, car cela encourage les développeurs à les exécuter fréquemment, ce qui facilite la détection précoce des problèmes.
I - **Independent (Indépendant)** : Les tests doivent être indépendants les uns des autres, ce qui signifie qu'ils ne doivent pas avoir de dépendances mutuelles. Cela permet d'isoler les problèmes plus facilement et d'améliorer la maintenance.
R - **Repeatable (Reproductible)** : Les tests doivent être reproductibles dans n'importe quel environnement. Ils ne doivent pas dépendre de conditions externes instables ou d'états changeants, afin de garantir des résultats cohérents.
S - **Self-validating (Auto-vérifiable)** : Les tests doivent s'auto-vérifier et retourner un résultat clair (passé/échoué) sans nécessiter de vérification manuelle. Cela facilite l'intégration des tests dans les processus d'intégration continue et d'automatisation.
T - **Timely (Opportun)** : Les tests doivent être écrits en même temps que le code, voire avant. Les retards dans l'écriture des tests peuvent entraîner des problèmes de qualité et de maintenabilité.

Développement TDD

TP

- On souhaite développer une classe Frame, qui représente une Frame dans le jeu du bowling, en utilisant les TDD.
- Les tests pour réaliser la classe Frame du jeu de bowling doivent couvrir les scénarios suivants:
  - S'il s'agit d'une série standard (round 1 par exemple)
  - Le premier lancer d'une série doit augmenter le score de la série
  - Le second lancer d'une série doit augmenter le score de cette série
  - En cas de strike, il ne doit pas être possible de lancer de nouveau au cours de cette même série.
  - En cas de lancers standards, il ne doit pas être possible de lancer plus de 2 fois

Développement TDD

TP

- S'il s'agit d'une série finale (dernier round)
  - En cas de strike, il doit être possible de lancer une nouvelle fois au cours d'une série
  - En cas de strike puis de lancer, le score est censé augmenter en accord avec le résultat du lancer.
  - En cas de strike puis d'un lancer, il doit être possible de lancer une nouvelle fois
  - En cas de strike puis de lancer, le score est censé augmenter en accord avec le résultat
  - En cas de spare, il doit être possible de lancer une nouvelle fois au cours d'une série
  - En cas de spare puis de lancer, le score est censé augmenter en accord avec le résultat du lancer.
  - En cas de lancers standards, il ne doit pas être possible de lancer plus de 4 fois