Les tests unitaires

1j – 7h

BERNIER Allan

03 2023

I. Introduction aux tests unitaires (durée: 1 heure)

- A. Définition des tests unitaires
- B. Pourquoi les tests unitaires sont importants?
- C. Types de tests unitaires D. Exemples de frameworks de tests unitaires

II. JUnit 5 (durée: 2 heures)

- A. Présentation de JUnit 5
- B. Annotation @Test
- C. Annotation @BeforeEach et @AfterEach
- D. Assertion API
- E. Les autres annotations JUnit 5 (@BeforeAll, @AfterAll, @Disabled, @Nested, @DisplayName, ...)
- F. Les paramètres de tests avec JUnit 5 G. Gestion des exceptions

III. TDD (Test-Driven Development) (durée: 2 heures)

- A. Qu'est-ce que TDD?
- B. Les trois étapes du TDD
- C. Les avantages et les inconvénients du TDD
- D. Les pratiques TDD

IV. Pratique des tests unitaires et de TDD avec JUnit 5 (durée: 2 heures)

- A. Exemples de tests unitaires avec JUnit 5
- B. Comment écrire des tests avec TDD?
- C. Implémentation de code avec TDD
- D. Intégration des tests unitaires dans un projet

V. Conclusion (durée: 10 minutes)

- A. Résumé du cours
- B. Prochaines étapes pour continuer à apprendre et pratiquer

I. Introduction aux tests unitaires (durée: 1 heure)

A. Définition des tests unitaires

Les tests unitaires sont des tests automatisés qui permettent de vérifier le bon fonctionnement d'une partie précise du code, appelée "unité".

Une unité peut être une méthode, une fonction ou une classe.

L'objectif des tests unitaires est de s'assurer que chaque unité fonctionne correctement en toutes circonstances et de détecter rapidement les erreurs.

B. Pourquoi les tests unitaires sont importants?

Les tests unitaires permettent d'identifier les erreurs dès le début du développement, ce qui réduit le coût de correction des erreurs dans les phases ultérieures du cycle de vie du logiciel.

Les tests unitaires facilitent la maintenance du code en réduisant le risque d'introduire des bugs lors de l'ajout de nouvelles fonctionnalités ou de la refonte du code existant.

Les tests unitaires aident les développeurs à mieux comprendre le code et à concevoir des interfaces claires pour leurs classes et leurs méthodes.

C. Types de tests unitaires

Les tests unitaires peuvent être divisés en plusieurs catégories en fonction de leur objectif :

Les tests de validation : vérifient que le comportement de l'unité est conforme aux spécifications.

Les tests d'erreur : vérifient que l'unité renvoie les erreurs attendues dans les cas où cela est approprié.

Les tests de performance : vérifient que l'unité fonctionne efficacement en termes de temps d'exécution et d'utilisation de la mémoire.

D. Exemples de frameworks de tests unitaires

JUnit : un framework de test unitaire pour Java.

NUnit : un framework de test unitaire pour .NET.

PHPUnit : un framework de test unitaire pour PHP.

Jasmine : un framework de test unitaire pour JavaScript.

PyUnit : un framework de test unitaire pour Python.

II. JUnit 5 (durée: 2 heures)

A. Présentation de JUnit 5

JUnit 5 est le dernier framework de test unitaire pour Java, sorti en 2017.

Il est constitué de plusieurs modules tels que JUnit Platform, JUnit Jupiter et JUnit Vintage.

JUnit 5 permet d'écrire des tests plus efficacement et plus facilement que les versions précédentes de JUnit.

B. Annotation @Test

L'annotation @Test est utilisée pour indiquer qu'une méthode est un test.

Elle permet d'exécuter la méthode pour vérifier qu'elle fonctionne correctement.

La méthode annotée @Test doit retourner void et ne doit prendre aucun argument.

C. Annotation @BeforeEach et @AfterEach

Les annotations @BeforeEach et @AfterEach permettent d'exécuter du code avant et après chaque méthode de test.

@BeforeEach est utilisé pour initialiser des objets ou des variables avant chaque test.

@AfterEach est utilisé pour nettoyer les ressources utilisées par les tests.

D. Assertion API

L'API d'assertion de JUnit 5 permet de vérifier si une condition est vraie ou fausse.

Les méthodes d'assertion peuvent être utilisées pour vérifier des valeurs, des exceptions, des tableaux, des collections, etc.

Les assertions aident à détecter rapidement les erreurs dans le code.

E. Les autres annotations JUnit 5

@BeforeAll : permet d'exécuter du code avant tous les tests dans la classe.

@AfterAll : permet d'exécuter du code après tous les tests dans la classe.

@Disabled : permet de désactiver un test.

@Nested : permet de créer des classes internes pour mieux organiser les tests.

@DisplayName : permet de spécifier un nom personnalisé pour un test.

F. Les paramètres de tests avec JUnit 5

JUnit 5 permet de passer des paramètres à une méthode de test en utilisant l'annotation @ParameterizedTest.

Les paramètres peuvent être fournis en utilisant différentes sources telles que des tableaux, des fichiers CSV ou des méthodes dédiées.

G. Gestion des exceptions

JUnit 5 permet de tester les exceptions en utilisant l'annotation @Test et en spécifiant l'exception attendue.

Il est possible de spécifier un message personnalisé pour l'exception.

III. TDD (Test-Driven Development) (durée: 2 heures)

A. Qu'est-ce que le TDD ?

Le TDD est une approche de développement de logiciels qui consiste à écrire d'abord les tests unitaires avant d'écrire le code de production.

Cette approche aide à garantir que le code est testé de manière exhaustive dès le début du processus de développement.

B. Les étapes du TDD

Les étapes du TDD sont les suivantes : écrire un test, exécuter le test et le faire échouer, écrire le code de production, exécuter les tests et les faire passer, puis refactoriser le code.

C. Les avantages du TDD

Le TDD permet d'écrire du code de qualité supérieure en détectant les erreurs dès le début du processus de développement.

Il encourage une meilleure conception du code et une meilleure couverture de test.

Le TDD permet également de réduire les coûts de maintenance et de débogage.

D. Les défis du TDD

Le TDD peut être difficile à mettre en place dans certaines organisations qui ont une culture de développement traditionnelle.

Il peut également être difficile d'écrire des tests exhaustifs pour tous les cas d'utilisation.

Enfin, le TDD peut prendre plus de temps que la méthode traditionnelle de développement.

E. Les meilleures pratiques pour le TDD

Écrire des tests simples qui se concentrent sur un seul aspect de la fonctionnalité.

Écrire des tests suffisamment détaillés pour détecter les erreurs dès le début du processus de développement.

Écrire des tests qui sont indépendants les uns des autres et ne dépendent pas d'autres tests.

Refactoriser le code souvent pour éviter les répétitions et les duplications de code.

F. Les outils pour le TDD

Il existe de nombreux outils disponibles pour faciliter le TDD, tels que JUnit, Mockito et EasyMock.

Les outils de couverture de code peuvent aider à vérifier la couverture de test.

Les outils d'intégration continue peuvent aider à automatiser le processus de construction et de test.

IV. Pratique des tests unitaires et de TDD avec JUnit 5 (durée: 2 heures)

A. Configuration de l'environnement de développement

Installer les outils nécessaires tels qu'un IDE (IntelliJ, Eclipse, etc.) et JUnit 5.

B. Écriture de tests unitaires avec JUnit 5

Expliquer la syntaxe de base de JUnit 5 pour les tests unitaires tels que @Test, @BeforeEach, @AfterEach, @BeforeAll et @AfterAll.

Exemples de tests unitaires pour des cas d'utilisation simples.

C. TDD avec JUnit 5

Écrire des tests unitaires avant le code de production pour une fonctionnalité simple.

Écrire le code de production pour faire passer les tests.

Répéter le processus pour plusieurs fonctionnalités.

D. Tests unitaires avancés avec JUnit 5

Expliquer comment utiliser les assertions avancées telles que assertThrows et assertAll.

Expliquer comment utiliser les annotations telles que @DisplayName et @Disabled.

E. Tests unitaires pour des applications réelles

Écrire des tests unitaires pour une application Java réelle en utilisant JUnit 5.

Couvrir différentes parties de l'application avec des tests unitaires.

F. Tests unitaires automatisés

Expliquer comment automatiser les tests unitaires à l'aide d'outils d'intégration continue tels que Jenkins ou Travis CI.

V. Conclusion (durée: 10 minutes)

Dans cette formation, nous avons vu l'importance des tests unitaires et du TDD dans le développement de logiciels de qualité. Nous avons examiné les bases de JUnit 5, les techniques de TDD et comment écrire des tests unitaires pour des applications Java réelles. Nous avons également discuté de l'automatisation des tests unitaires à l'aide d'outils d'intégration continue tels que Jenkins ou Travis CI.

En conclusion, les tests unitaires et le TDD sont des compétences essentielles pour tout développeur de logiciels. Ils permettent de garantir que le code fonctionne comme prévu et qu'il est facilement maintenable. En outre, l'automatisation des tests unitaires permet d'assurer la qualité du code tout au long du développement.

Nous espérons que cette formation vous a donné les compétences et les connaissances nécessaires pour écrire des tests unitaires efficaces et mettre en pratique le TDD dans vos projets de développement de logiciels. N'oubliez pas que l'écriture de tests unitaires est un processus continu qui doit être intégré dans votre cycle de développement de logiciels pour garantir la qualité et la fiabilité du code.