Les tests unitaires, souvent désignés par le terme "unit testing," sont une pratique de test logiciel qui consiste à évaluer individuellement des composants logiciels, appelés "unités," pour s'assurer qu'ils fonctionnent correctement de manière isolée. Une "unité" dans ce contexte peut être une fonction, une méthode, une classe ou un module de code.

Voici les caractéristiques clés des tests unitaires :

Isolation : Les tests unitaires sont conçus pour être exécutés de manière isolée, ce qui signifie qu'ils ne dépendent pas d'autres parties du code. Les dépendances externes sont généralement remplacées par des objets factices ou des mocks pour s'assurer que les tests évaluent uniquement l'unité en question.

Automatisation : Les tests unitaires sont automatisés, ce qui signifie qu'ils peuvent être exécutés régulièrement sans intervention manuelle. Cela permet de détecter rapidement les régressions ou les problèmes de code.

Répétabilité : Les tests unitaires devraient donner les mêmes résultats à chaque exécution, à condition que le code testé et l'environnement de test restent inchangés. Cela garantit la reproductibilité des tests.

Rapidité : Les tests unitaires doivent être rapides à exécuter, car ils sont souvent exécutés fréquemment. Cette rapidité permet une validation continue du code pendant le développement.

Indépendance du contexte : Les tests unitaires ne doivent pas dépendre de la configuration de l'environnement, des bases de données, des services externes, etc. Ils sont conçus pour être portables et indépendants du contexte.

Le "mocking" (ou "simulation" en français) est une technique utilisée en développement de logiciels et en test logiciel pour créer des objets factices ou des mocks (simulations) qui imitent le comportement des composants réels du logiciel, généralement dans le but de les isoler lors de tests unitaires. Cette technique est couramment utilisée pour simuler le comportement de dépendances externes, telles que des services, des bases de données ou d'autres composants, afin de tester une unité de code de manière isolée.

Voici quelques points clés liés au mocking :

Isolation des dépendances : Le mocking permet d'isoler l'unité de code que vous souhaitez tester en remplaçant ses dépendances réelles par des mocks. Cela garantit que les tests se concentrent uniquement sur le code de l'unité en question.

Contrôle du comportement : Les mocks permettent de définir un comportement spécifique pour les méthodes ou les fonctions des dépendances simulées. Vous pouvez indiquer comment les mocks doivent répondre aux appels de méthodes, quels résultats renvoyer, etc.

Réduction de la complexité : En utilisant le mocking, vous pouvez éviter de devoir configurer des dépendances complexes, telles que des bases de données ou des services externes,

lors de l'exécution de tests unitaires. Cela simplifie la mise en place des tests et accélère leur exécution.

Répétabilité : Les mocks assurent la reproductibilité des tests, car vous pouvez prédéfinir les réponses attendues des dépendances simulées, garantissant ainsi que les tests unitaires produiront des résultats cohérents.

Facilitation des tests d'erreurs : Les mocks permettent de simuler des conditions d'erreur ou des situations difficiles à reproduire avec de vraies dépendances, ce qui facilite les tests d'erreurs et les scénarios exceptionnels.

Les bibliothèques de mock, telles que Mockito (pour Java), Moq (pour .NET), sinon (pour JavaScript), et d'autres, offrent des outils pour créer et gérer des mocks de manière efficace. Ces outils simplifient la création de mocks et la définition de comportements simulés, permettant ainsi aux développeurs de concevoir des tests unitaires complets et précis pour leurs logiciels.

JUnit :
JUnit est un framework de test unitaire pour Java, bien qu'il existe également des versions pour d'autres langages de programmation. Il permet aux développeurs de créer des tests unitaires pour vérifier le comportement attendu des parties individuelles de leur code source. Les tests JUnit sont écrits sous forme de méthodes de test annotées, et JUnit fournit des assertions pour vérifier si les résultats correspondent aux attentes.

Mockito :
Mockito est une bibliothèque Java populaire utilisée pour la création de mock objects (objets factices) dans les tests. Les objets factices, ou "mocks", sont utilisés pour simuler le comportement de composants réels, tels que des dépendances externes, afin d'isoler l'unité de code testée. Mockito facilite la création de mocks et permet de définir des comportements simulés pour ces mocks.

Tests Doubles :
Les "tests doubles" sont un concept général en programmation et en test logiciel. Ils sont utilisés pour remplacer de vrais composants par des implémentations factices ou simulées dans le but de simplifier le processus de test. Il existe plusieurs types de tests doubles, dont voici quelques-uns :

Dummy : Un objet "dummy" est utilisé pour remplir un paramètre sans être réellement utilisé dans le test. Il ne fait que satisfaire la signature de la méthode.

Stub : Un "stub" est un remplacement qui fournit des réponses prédéfinies aux appels de méthodes. Il est utilisé pour contrôler le comportement d'une dépendance dans un test.

Spy : Un "spy" est un objet réel qui est surveillé pendant le test. Vous pouvez vérifier comment il est utilisé, notamment combien de fois certaines méthodes sont appelées.

Mock : Un "mock" est un objet qui enregistre les appels de méthodes et leurs paramètres, vous permettant de vérifier si les appels attendus ont été effectués. Mockito est une bibliothèque populaire pour créer des mocks.

Fake : Un "fake" est une implémentation simplifiée d'une dépendance réelle qui peut être utilisée à des fins de test, mais qui peut avoir des comportements différents de la vraie implémentation.