Testes Unitários com JUnit 5 e Mockito

Testes unitários são uma parte fundamental do desenvolvimento de software, garantindo a qualidade e a confiabilidade do código. Eles consistem em testar as menores unidades de código de forma isolada, como métodos ou classes, para verificar se funcionam conforme o esperado.

# O que são Testes Unitários?

Testes unitários são automações que validam se uma parte específica e isolada de um software (a "unidade") funciona corretamente. O objetivo é testar cada unidade de código (um método, uma função, uma classe) independentemente das outras, para identificar e corrigir erros precocemente no ciclo de desenvolvimento. Isso ajuda a reduzir o número de bugs em produção e facilita a manutenção do código.

## Como eles são implementados?

A implementação de testes unitários geralmente segue o padrão "Arrange-Act-Assert" (AAA):

* **Arrange (Preparar):** Configura o ambiente de teste, inicializando objetos, dados e dependências necessários para o teste.
* **Act (Agir):** Executa a unidade de código que está sendo testada.
* **Assert (Verificar):** Valida o resultado da execução, comparando o comportamento real com o comportamento esperado.

# JUnit 5

JUnit 5 é o framework de testes mais popular para Java. Ele oferece uma API robusta para escrever e executar testes unitários de forma eficaz.

## Principais anotações do JUnit 5:

* @Test: Marca um método como um método de teste.
* @DisplayName: Fornece um nome mais legível para o teste.
* @BeforeEach: Executa um método antes de cada teste.
* @AfterEach: Executa um método depois de cada teste.
* @BeforeAll: Executa um método uma vez antes de todos os testes na classe.
* @AfterAll: Executa um método uma vez depois de todos os testes na classe.
* @Disabled: Desabilita um teste ou classe de teste.

# Mockito

Mockito é um framework de mocking para Java. Ele permite criar objetos "mock" (simulados) de dependências externas para isolar a unidade de código que está sendo testada. Isso é crucial para testes unitários, pois garante que o teste se concentre apenas no comportamento da unidade em questão, sem se preocupar com a lógica interna das suas dependências.

## Principais anotações do Mockito:

* @Mock: Cria um objeto mock.
* @InjectMocks: Injeta os mocks criados nos objetos que estão sendo testados.
* when().thenReturn(): Define o comportamento de um método mockado.
* verify(): Verifica se um método de um mock foi chamado.

# Utilizando JUnit 5 e Mockito em Conjunto

A combinação de JUnit 5 e Mockito é poderosa para a escrita de testes unitários abrangentes. O JUnit 5 fornece a estrutura para os testes, enquanto o Mockito permite controlar o comportamento das dependências, isolando a unidade sob teste.

## Exemplo de implementação em Java:

Considere uma classe UserService que depende de um UserRepository para acessar dados de usuários. Segue o link de um gist para o exemplo proposto:

<https://gist.github.com/AFSFerreira/5f7c2df69e16106ecd6acf73867a3854>

1. @ExtendWith(MockitoExtension.class): Integra o Mockito ao JUnit 5, permitindo o uso das anotações @Mock e @InjectMocks.
2. @Mock private UserRepository userRepository;: Cria uma instância mock do UserRepository.
3. @InjectMocks private UserService userService;: Cria uma instância de UserService e injeta automaticamente o userRepository mockado nela.
4. when(userRepository.findById(userId)).thenReturn(mockUser);: Esta é a parte essencial do Mockito. Estamos dizendo ao mock que, quando o método findById for chamado com o userId específico, ele deve retornar o mockUser que criamos. Isso simula o comportamento do banco de dados ou de uma API externa sem realmente acessá-los.
5. assertEquals e assertNull: São métodos de asserção do JUnit 5 que verificam se o resultado obtido é o esperado.

Ao executar esses testes, você estará verificando o comportamento da classe UserService de forma isolada, sem depender de uma implementação real do UserRepository (que poderia, por exemplo, exigir um banco de dados). Isso torna os testes mais rápidos, confiáveis e independentes.

Para mais informações, você pode consultar as documentações oficiais:

* JUnit 5: File
* Mockito: File