Residência em Software

Módulo Programação JAVA (Avançado)

MÊS 01

INSTITUIÇÃO EXECUTORA

cepedi UESC

COORDENADORA

MCTI FUTURO Softex

APOIO

MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO

GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

Importância de escrever testes

- Quality Assurance
- Regression Testing
- Documentation
- Code Maintainability

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

Testes Unitários

Possibilitam verificar a exatidão de componentes de unidades individuais ou módulos do software.

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

Para que servem os Testes Unitários?

- Detectar defeitos precocemente
- Garantir a correção do código
- Facilitar a refatoração
- Promover a modularidade
- Promover a reutilização
- Documentação executável

Escrevendo testes usando **JUnit e Mockito**

JUnit

JUnit é uma estrutura de teste Open Source para Java que depende fortemente de anotações para executar e gerenciar nossos testes.

Ele define uma interface estável e poderosa entre o JUnit e seus clientes, como ferramentas de compilação.

Escrevendo testes usando **JUnit e Mockito**

Novas anotações - JUnit 5 (Júpiter)

@TestFactory: denota um método que é uma fábrica de testes para testes dinâmicos
@DisplayName: define um nome de exibição personalizado para uma classe de teste ou um método de teste
@Nested: indica que a classe anotada é uma classe de teste aninhada e não estática
@Tag: declara tags para testes de filtragem
@ExtendWith: registra extensões personalizadas

Escrevendo testes usando **JUnit e Mockito**

Novas anotações - JUnit 5 (Júpiter)

@BeforeEach: denota que o método anotado será executado antes de cada método de teste (previamente @Before)
@AfterEach: denota que o método anotado será executado após cada método de teste (previamente @After)
@BeforeAll: denota que o método anotado será executado antes de todos os métodos de teste na classe atual (anteriormente @BeforeClass)
@AfterAll: indica que o método anotado será executado após todos os métodos de teste na classe atual (anteriormente @AfterClass)
@Disabled: desabilita uma classe ou método de teste (anteriormente @Ignore)

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

JUnit(dependências maven)

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.9.2</version>
    <scope>test</scope>
</dependency>
```

Arquivo: pom.xml

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

@BeforeAll e @BeforeEach

```
@BeforeAll
static void setup() {
    log.info("@BeforeAll - executes once before all test methods in this class");
}

@BeforeEach
void init() {
    log.info("@BeforeEach - executes before each test method in this class");
}
```

Escrevendo testes usando **JUnit e Mockito**

@DisplayName e @Disabled

```
@DisplayName("Single test successful")
@Test
void testSingleSuccessTest() {
    log.info("Success");
}

@Test
@Disabled("Not implemented yet")
void testShowSomething() {
}
```

Escrevendo testes usando **JUnit e Mockito**

@AfterEach e @AfterAll

```
@AfterEach
void tearDown() {
    log.info("@AfterEach - executed after each test method.");
}

@AfterAll
static void done() {
    log.info("@AfterAll - executed after all test methods.");
}
```

Escrevendo testes usando **JUnit e Mockito**

Teste de exceção

```java
@Test
void shouldThrowException() {
    Throwable exception = assertThrows(UnsupportedOperationException.class, () -> {
      throw new UnsupportedOperationException("Not supported");
    });
    assertEquals("Not supported", exception.getMessage());
}

@Test
void assertThrowsException() {
    String str = null;
    assertThrows(IllegalArgumentException.class, () -> {
      Integer.valueOf(str);
    });
}
```

Escrevendo testes usando **JUnit e Mockito**

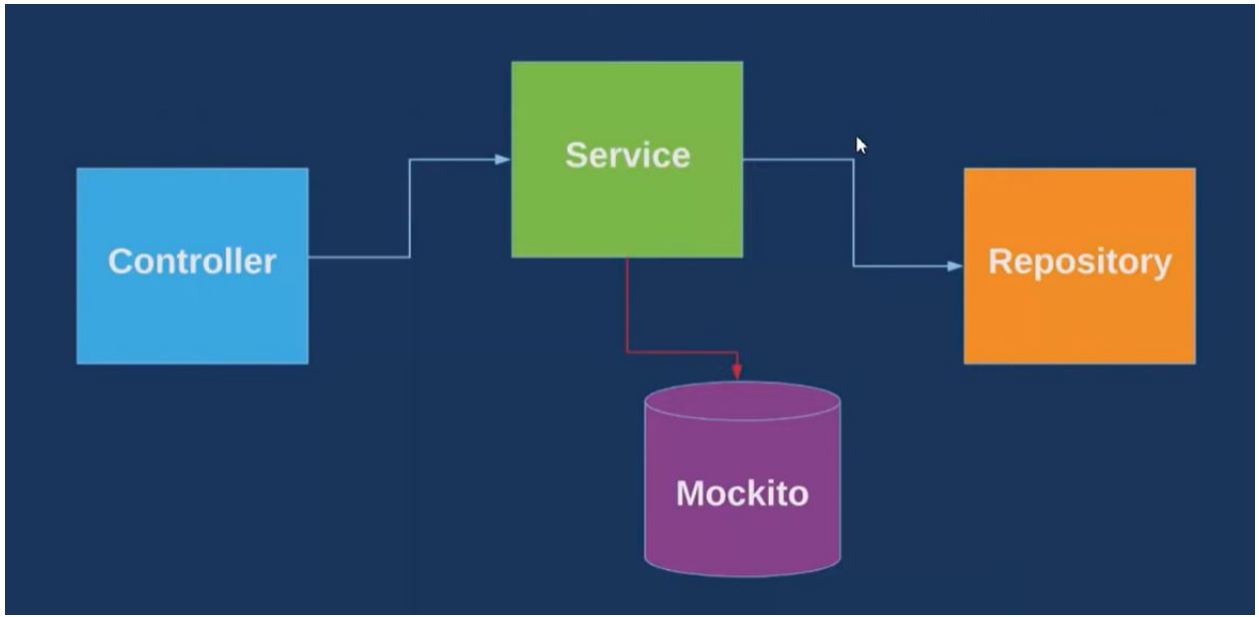
O que é o Mockito?

O Mockito é um framework de testes unitários para instanciar classes e controlar o comportamento dos métodos. Isso é chamado de mock, ao mockar a dependência de uma classe, eu faço com que a classe que estou testando pense estar invocando o método realmente.

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

Como funciona?



Escrevendo testes usando **JUnit e Mockito**

Anotações do Mockito

Mock: cria uma instância de uma classe, porém Mockada. Se você chamar um método ele não irá chamar o método real, exceto se você queira.
Spy: cria uma instância de uma classe, que você pode mockar ou chamar os métodos reais. É uma alternativa ao InjectMocks, quando é preciso mockar métodos da própria classe que esta sendo testada.
InjectMocks: criar uma instância e injeta as dependências necessárias, que estão anotadas com @Mock.

Escrevendo testes usando **JUnit e Mockito**

Anotações do Mockito

Verify: verifica a quantidade de vezes e quais parâmetros utilizados para acessar um determinado método.
When: Após um mock ser criado, você pode direcionar um retorno para um metodo dado um parâmetro de entrada.
Given: Mesmo propósito que o when, porém é utilizado para BDD. Fazendo parte do BDDMockito.

Residência em Software

Escrevendo testes usando **JUnit e Mockito**

Mockito (dependências maven)

```
<dependency>
    <groupId>org.mockito</groupId>
    <artifactId>mockito-junit-jupiter</artifactId>
    <version>5.3.1</version>
    <scope>test</scope>
</dependency>
```

Arquivo: pom.xml

Residência em Software

PP009

Exemplos de uso:

- https://gist.github.com/rog3r/49de1e3fdbc15a960876cda534f1bffc
- https://gist.github.com/rog3r/04e9976957f1e286262f37653c9e5ce4

Residência em Software

PP009

Orientações:

- Instale as dependências necessárias, e
- Altere suas classes para usar o JUnit5 e o Mockito.

Residência em Software

Contato

rogerio.jesus@cepedi.org.br

https://moodle.residenciatic18.cepedi.org.br/