ATIVIDADE:: Demais Dublês de Teste

A "Atividade: Testes com Mockito" solicita o seguinte:

- 3. Leia e teste os exemplos nas referências sobre o Mockito:
    - 1. indique ao menos três exemplos que considere mais notáveis;
    - 2. Comente as funcionalidades mais interessantes desse framework para criação de Mocks.

TestAccountServiceMockito :

Funcionalidades demonstradas no exemplo:

- @ExtendWith(MockitoExtension.class): Esta anotação é usada para integrar o Mockito com o ciclo de vida do JUnit 5.
- @Mock: Esta é uma anotação usada para injetar e criar automaticamente uma instância de mock para o campo anotado. No exemplo, ela é usada para criar o mockAccountManager.
- Mockito.lenient(): O exemplo utiliza este método. Isso sugere uma configuração que torna o mock "indulgente", o que geralmente significa que ele não falhará se houver configurações de*stub* (com when) que não foram utilizadas durante o teste.
- when(...): O exemplo mostra o início da sintaxe when(mockAccountManager.findAccountForUser("1")). Esta é a abordagem principal do Mockito para configurar o comportamento de um mock (conhecido como *stubbing*). Ela define o que deve acontecer (geralmente, qual valor retornar) quando um método específico do mock é invocado.

Exemplo 1 – Simulando dependência e retornando valores

Caso clássico: temos um UserService que depende de um UserRepository. Queremos testar o UserService sem precisar de banco de dados.

```
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
import org.mockito.Mockito;

class UserServiceTest {

  @Test
  void deveRetornarUsuarioQuandoIdExistir() {
    // 1. Criamos um mock do repositório (simula o banco)
    UserRepository userRepository = Mockito.mock(UserRepository.class);
```

```
        // 2. Definimos o comportamento do mock (stub)
        User fakeUser = new User(1, "Alice");
        when(userRepository.findById(1)).thenReturn(fakeUser);

        // 3. Injetamos o mock no service
        UserService service = new UserService(userRepository);

        // 4. Chamamos o método a ser testado
        User result = service.getUserById(1);

        // 5. Validamos o resultado
        assertEquals("Alice", result.getName());
    }
}
```

O que aconteceu:

- when(...).thenReturn(...) → simulou que o repositório retorna um objeto específico.
- Isso permite testar o UserService isolado, sem dependência de banco.

Exemplo 2 – Verificando interações

Agora queremos garantir que o UserService realmente chama o UserRepository como esperado.

```
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;

class UserServiceSaveTest {

    @Test
    void deveChamarSaveDoRepositorio() {
        // Mock do repositório
        UserRepository userRepository = mock(UserRepository.class);

        UserService service = new UserService(userRepository);

        User newUser = new User(2, "Bob");

        // Ação
```

```
        service.createUser(newUser);

        // Verificação de interação:
        verify(userRepository).save(newUser);
        // Podemos ainda verificar o número de chamadas:
        verify(userRepository, times(1)).save(newUser);
    }
}
```

O que aconteceu aqui:

- verify(...) garante que o método foi realmente chamado.
- Isso é útil em métodos void, onde não há retorno, mas queremos validar que uma ação ocorreu.

Exemplo 3 – Lançando exceções simuladas

Muitas vezes precisamos simular erros (ex.: falha de banco). Com Mockito podemos simular exceções para verificar se o código trata corretamente.

```
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

class UserServiceExceptionTest {

    @Test
    void deveLancarExcecaoQuandoRepositorioFalhar() {
        UserRepository userRepository = mock(UserRepository.class);

        // Simulando exceção no repositório
        when(userRepository.findById(99))
            .thenThrow(new RuntimeException("Erro no banco"));

        UserService service = new UserService(userRepository);

        // Valida que a exceção é propagada
        RuntimeException ex = assertThrows(
            RuntimeException.class,
```

```
            () -> service.getUserById(99)
        );

        assertEquals("Erro no banco", ex.getMessage());
    }
}
```

O que aconteceu aqui:

- thenThrow(...) simulou uma falha ao buscar o usuário.
- O teste garante que o UserService trata (ou repassa) a exceção como esperado.