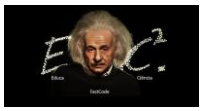


Mockito Essencial

30 Boas Práticas com Exemplos para Testes em Java

Mockito é uma das bibliotecas mais populares para criação de testes unitários em Java. Com ele, podemos simular comportamentos de dependências e focar o teste exclusivamente na lógica da unidade sob teste.

Neste artigo, apresento 30 boas práticas e formas eficazes de utilizar o Mockito para escrever testes mais limpos, legíveis e robustos.

1. Use `@Mock` para declarar dependências simuladas

```
@Mock
private Service service;
```

2. Use `@InjectMocks` para injetar automaticamente os mocks

```
@InjectMocks
private Controller controller;
```

3. Inicialize os mocks com `MockitoAnnotations.openMocks(this)` ou `@ExtendWith`

JUnit 5:

```
@ExtendWith(MockitoExtension.class)
class ControllerTest {}
```

4. Defina comportamentos com `when(...).thenReturn(...)`

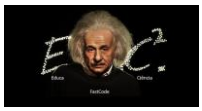
```
when(service.getData()).thenReturn("mockado");
```

5. Verifique interações com `verify(...)`

```
verify(repository).save(entity);
```

6. Use `verify(..., times(n))` para checar quantidade de chamadas

```
verify(service, times(2)).execute();
```



7. Use verifyNoMoreInteractions(...)

verifyNoMoreInteractions(service);

8. Substitua comportamentos reais com doReturn(...).when(...)

doReturn("valor").when(service).metodo();

9. Simule exceções com doThrow(...).when(...)

doThrow(new RuntimeException()).when(service).delete(id);

10. Use ArgumentCaptor para capturar argumentos

ArgumentCaptor<Entidade> captor = ArgumentCaptor.forClass(Entidade.class);
verify(repo).save(captor.capture());
assertEquals("nome esperado", captor.getValue().getNome());

11. Simule chamadas consecutivas com diferentes retornos

when(service.get()).thenReturn("um", "dois", "tres");

12. Use any() para argumentos genéricos

when(repo.findById(any())).thenReturn(Optional.of(entity));

13. Combine eq() e any() para correspondência parcial

when(service.process(eq("nome"), anyInt())).thenReturn(true);

14. Use lenient() para mocks que não precisam ser validados

lenient().when(service.metodo()).thenReturn("ok");

15. Use spy() para testar parcialmente um objeto real

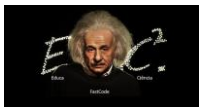
List<String> spyList = spy(new ArrayList<>());
doReturn("mockado").when(spyList).get(0);

16. Use @Spy com @InjectMocks para objetos parcialmente reais

@Spy
private List<String> nomes = new ArrayList<>();

17. Use @Captor para capturar argumentos sem repetir forClass(...)

@Captor
ArgumentCaptor<Usuario> captor;



18. Nomeie seus mocks de forma clara

```
@Mock
private EmailService emailService;
```

19. Evite usar mocks em excesso

Evite:

```
@Mock
String texto;
```

20. Não simule classes simples como `String` ou `List`

Utilize instâncias reais quando apropriado.

21. Use `InOrder` para verificar a ordem das chamadas

```
InOrder inOrder = inOrder(service1, service2);
inOrder.verify(service1).start();
inOrder.verify(service2).execute();
```

22. Use `thenAnswer(...)` para comportamento dinâmico

```
when(service.transform(anyString())).thenAnswer(invocation -> {
    String input = invocation.getArgument(0);
    return input.toUpperCase();
});
```

23. Evite usar `reset()` a menos que seja essencial

```
reset(service); // use com cautela
```

24. Use `Mockito.mockStatic(...)` para simular métodos estáticos

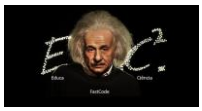
```
try (MockedStatic<Utils> mocked = mockStatic(Utils.class)) {
    mocked.when(Utils::calcular).thenReturn(42);
}
```

25. Use `@ExtendWith(MockitoExtension.class)` com JUnit 5

```
@ExtendWith(MockitoExtension.class)
class MeuTeste {}
```

26. Evite `@RunWith(MockitoJUnitRunner.class)` com JUnit 5

Prefira `@ExtendWith(MockitoExtension.class)`.



27. Não misture mocks com dependências reais sem critério

Mantenha o escopo de teste isolado.

28. Simule apenas o necessário

Evite configuração de comportamento não utilizado:

```
when(service.metodoInexistente()).thenReturn("x");
```

29. Prefira nomes de testes descritivos

```
@Test
void deveRetornarErroQuandoUsuarioNaoExiste() {}
```

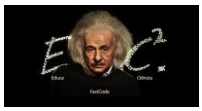
30. Organize seus mocks por contexto de teste

Agrupe-os em classes internas ou por cenário de caso de uso.

Exemplos práticos adicionais

A. Teste com @Mock e @InjectMocks

```
@ExtendWith(MockitoExtension.class)
class UserServiceTest {

    @Mock
    private UserRepository userRepository;

    @InjectMocks
    private UserService userService;

    @Test
    void deveSalvarUsuario() {
        User user = new User("João");
        when(userRepository.save(any())).thenReturn(user);

        User result = userService.salvar(user);

        assertEquals("João", result.getNome());
        verify(userRepository).save(user);
    }
}
```



B. Uso de ArgumentCaptor

```
@Test
void deveCapturarArgumentoSalvo() {
    User user = new User("Maria");
    userService.salvar(user);

    ArgumentCaptor<User> captor = ArgumentCaptor.forClass(User.class);
    verify(userRepository).save(captor.capture());

    assertEquals("Maria", captor.getValue().getNome());
}
```

C. Simulação de exceção

```
@Test
void deveLancarExcecaoAoDeletar() {
    doThrow(new IllegalArgumentException()).when(userRepository).deleteById(1L);

    assertThrows(IllegalArgumentException.class, () -> userService.excluir(1L));
}
```

D. Verificação de ordem de chamadas

```
@Test
void deveChamarServicosEmOrdem() {
    userService.executarProcesso();

    InOrder inOrder = inOrder(auditoriaService, processamentoService);
    inOrder.verify(auditoriaService).registrarInicio();
    inOrder.verify(processamentoService).processar();
}
```

Enfim...

Adotar boas práticas com Mockito é essencial para garantir testes unitários confiáveis, fáceis de manter e que realmente validem o comportamento da lógica de negócio.

Mais do que simplesmente simular dependências, o uso consciente de anotações, capturas, verificadores e organização dos testes permite extrair o máximo da ferramenta.

Em um contexto de engenharia de software profissional, o Mockito se torna uma ponte fundamental entre qualidade, segurança e produtividade.

A comunidade EducaCiência FastCode incentiva o uso de padrões e ferramentas modernas para acelerar o desenvolvimento com qualidade.

Esperamos que este guia sirva como referência para você aprimorar seus testes e projetos Java.

EducaCiência FastCode para a comunidade