JUnit em Java

Exemplos Práticos e Boas Práticas para Testes Eficazes

JUnit é uma biblioteca de testes essencial para o desenvolvimento de software em Java, permitindo a criação de testes automatizados que asseguram a funcionalidade e a qualidade do código.

A implementação de testes, especialmente em projetos complexos desenvolvidos com frameworks como Spring Boot e JSP, é crucial para garantir que novas alterações não introduzam bugs.

Este artigo explora uma ampla gama de exemplos práticos do JUnit, apresentando 50 casos de teste comentados em alto nível para auxiliar desenvolvedores a adotar boas práticas na implementação de testes automatizados.

Exemplos Didáticos de JUnit

1. Teste Simples de Soma

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

public class CalculadoraTest {

    @Test
    public void testSoma() {
        Calculadora calc = new Calculadora();
        // Verifica se 2 + 3 resulta em 5
        assertEquals(5, calc.soma(2, 3), "A soma deve ser 5");
    }
}
```

2. Teste de Subtração

```
@Test
public void testSubtracao() {
    Calculadora calc = new Calculadora();
    // Verifica se 5 - 3 resulta em 2
    assertEquals(2, calc.subtracao(5, 3), "A subtração deve ser 2");
}
```

3. Teste de Multiplicação

```
@Test
```

```
public void testMultiplicacao() {
    Calculadora calc = new Calculadora();
    // Verifica se 4 * 2 resulta em 8
    assertEquals(8, calc.multiplicacao(4, 2), "A multiplicação deve ser 8");
}
```

4. Teste de Divisão

```
@Test
public void testDivisao() {
    Calculadora calc = new Calculadora();
    // Verifica se 10 / 2 resulta em 5
    assertEquals(5, calc.divisao(10, 2), "A divisão deve ser 5");
}
```

5. Teste de Exceção (Divisão por Zero)

```
@Test
public void testDivisaoPorZero() {
    Calculadora calc = new Calculadora();
    // Verifica se uma exceção é lançada ao dividir por zero
    assertThrows(ArithmeticException.class, () -> calc.divisao(10, 0));
}
```

6. Teste de Inicialização com @BeforeEach

```
import org.junit.jupiter.api.BeforeEach;

public class UsuarioServiceTest {

    private UsuarioService usuarioService;

    @BeforeEach
    void setUp() {
        // Inicializa o serviço antes de cada teste
        usuarioService = new UsuarioService();
    }

    @Test
    public void testCadastrarUsuario() {
        Usuario usuario = new Usuario("Maria");
        usuarioService.cadastrar(usuario);
        // Verifica se o usuário foi adicionado
        assertNotNull(usuarioService.buscarPorNome("Maria"));
    }
}
```

7. Teste com @ParameterizedTest

```
import org.junit.jupiter.params.ParameterizedTest;
```

```
import org.junit.jupiter.params.provider.ValueSource;

public class CalculadoraTest {

    @ParameterizedTest
    @ValueSource(ints = {2, 4, 6})
    void testPar(int numero) {
        // Verifica se o número é par
        assertTrue(numero % 2 == 0, numero + " deve ser par");
    }
}
```

8. Mocking com Mockito

```
import static org.mockito.Mockito.*;
import org.mockito.InjectMocks;
import org.mockito.Mock;
import org.mockito.MockitoAnnotations;

public class UsuarioServiceTest {

    @Mock
    private UsuarioRepository usuarioRepository;

    @InjectMocks
    private UsuarioService usuarioService;

    @BeforeEach
    void setUp() {
        MockitoAnnotations.openMocks(this);
    }

    @Test
    void testBuscarUsuarioPorId() {
        Usuario usuario = new Usuario(1L, "João");
        when(usuarioRepository.findById(1L)).thenReturn(Optional.of(usuario));
        // Verifica se o nome do usuário encontrado é "João"
        assertEquals("João", usuarioService.buscarUsuarioPorId(1L).getNome());
    }
}
```

9. Teste de Integração com Spring Boot

```
import org.junit.jupiter.api.Test;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.boot.test.autoconfigure.web.servlet.AutoConfigureMockMvc;
import org.springframework.boot.test.context.SpringBootTest;

@SpringBootTest
@AutoConfigureMockMvc
public class UsuarioControllerTest {

    @Autowired
    private MockMvc mockMvc;

    @Test
    public void testGetUsuario() throws Exception {
        mockMvc.perform(get("/usuarios/1"))
                .andExpect(status().isOk())
```

```
            .andExpect(jsonPath("$.nome").value("João"));
    }
}
```

10. Teste de CRUD em um Controlador

```
@Test
public void testCriarUsuario() throws Exception {
    Usuario usuario = new Usuario("Maria");

    mockMvc.perform(post("/usuarios")
            .contentType(MediaType.APPLICATION_JSON)
            .content("{\"nome\": \"Maria\"}"))
            .andExpect(status().isCreated());
}
```

11. Teste de Endpoints com Cabeçalhos

java

```
@Test
public void testGetUsuarioComCabecalho() throws Exception {
    mockMvc.perform(get("/usuarios/1")
            .header("Authorization", "Bearer token"))
            .andExpect(status().isOk());
}
```

12. Teste de Resposta com JSON

```
@Test
public void testUsuarioResponse() throws Exception {
    mockMvc.perform(get("/usuarios/1"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("$.nome").value("João"))
            .andExpect(jsonPath("$.id").value(1));
}
```

13. Teste de Exceção em Controlador

```
@Test
public void testUsuarioNaoEncontrado() throws Exception {
    mockMvc.perform(get("/usuarios/999"))
            .andExpect(status().isNotFound());
}
```

14. Teste de Serviço com Condições

```
@Test
public void testBuscarUsuarioAtivo() {
    Usuario usuario = new Usuario("João", true);
    usuarioService.cadastrar(usuario);

    List<Usuario> ativos = usuarioService.buscarAtivos();
    assertTrue(ativos.contains(usuario), "Deve conter o usuário ativo");
}
```

15. Teste de Vários Cenários com @Nested

```
import org.junit.jupiter.api.Nested;

@Nested
class QuandoUsuarioAtivo {

    @Test
    void testBuscarUsuarioAtivo() {
        // Teste específico para usuário ativo
    }

    @Test
    void testCadastrarUsuarioAtivo() {
        // Teste específico para cadastrar usuário ativo
    }
}
```

16. Teste de Conexão com o Banco de Dados

```
@Test
@Transactional
public void testUsuarioPersistidoNoBanco() {
    Usuario usuario = new Usuario("Maria");
    usuarioService.cadastrar(usuario);
    assertNotNull(usuarioService.buscarPorNome("Maria"));
}
```

17. Teste de Implementação de Interface

```
public class UsuarioServiceImpl implements UsuarioService {
    @Override
    public void cadastrar(Usuario usuario) {
        // Implementação do método
    }
}

@Test
public void testCadastrarUsuarioImplementacao() {
    UsuarioService usuarioService = new UsuarioServiceImpl();
    Usuario usuario = new Usuario("Ana");
    usuarioService.cadastrar(usuario);
    // Verifica se o usuário foi cadastrado corretamente
}
```

18. Teste de Performance

```
@Test
public void testPerformanceCadastro() {
    long startTime = System.currentTimeMillis();

    for (int i = 0; i < 1000; i++) {
        usuarioService.cadastrar(new Usuario("Usuario" + i));
    }

    long endTime = System.currentTimeMillis();
    // Verifica se o tempo de execução é aceitável
    assertTrue(endTime - startTime < 1000, "Cadastro deve ser rápido");
}
```

19. Teste de Segurança de Dados

```
@Test
public void testSegurancaDados() {
    // Testa se os dados sensíveis estão criptografados
    String senhaCriptografada = usuarioService.criptografarSenha("minhaSenha");
    assertNotEquals("minhaSenha", senhaCriptografada);
}
```

20. Teste de Validação de Dados de Entrada

```
@Test
public void testUsuarioNomeNaoVazio() {
    Usuario usuario = new Usuario("");
    // Verifica se a exceção é lançada ao tentar cadastrar um usuário com nome vazio
    assertThrows(IllegalArgumentException.class, () -> usuarioService.cadastrar(usuario));
}
```

21. Teste de Múltiplos Cenários com @MethodSource

```
@ParameterizedTest
@MethodSource("providesDadosParaTesteSoma")
void testSomaComDiferentesValores(int a, int b, int resultadoEsperado) {
    assertEquals(resultadoEsperado, Calculadora.soma(a, b));
}

static Stream<Arguments> providesDadosParaTesteSoma() {
    return Stream.of(
            Arguments.of(1, 2, 3),
            Arguments.of(2, 3, 5),
            Arguments.of(5, 5, 10)
    );
}
```

22. Teste de Manipulação de Exceções

```
@Test
public void testManipularExcecao() {
    try {
        usuarioService.cadastrar(null);
    } catch (IllegalArgumentException e) {
        // Verifica se a exceção é a esperada
        assertEquals("Usuário não pode ser nulo", e.getMessage());
    }
}
```

23. Teste de Estado de Objeto

```
@Test
public void testEstadoDoUsuario() {
    Usuario usuario = new Usuario("Carlos", false);
    usuario.ativar();
    assertTrue(usuario.isAtivo(), "Usuário deve estar ativo após ativação");
}
```

24. Teste de Métodos Estáticos

```
@Test
public void testCalculoEstatistico() {
    double resultado = Estatisticas.media(new int[]{1, 2, 3, 4, 5});
    assertEquals(3.0, resultado, "A média deve ser 3.0");
}
```

25. Teste de Inicialização de Configurações

```
import org.junit.jupiter.api.BeforeAll;

public class ConfiguracaoTest {

    @BeforeAll
    static void inicializarConfiguracao() {
        // Configurações globais antes de todos os testes
    }

    @Test
    public void testConfiguracaoValida() {
        assertTrue(Configuracao.isValida(), "A configuração deve ser válida");
    }
}
```

26. Teste de Resposta de API com MockRestServiceServer

```
@Autowired
private MockRestServiceServer server;

@Test
public void testChamadaApiExterna() {
    server.expect(requestTo("/api/external"))
          .andRespond(withSuccess("{\"result\": \"success\"}", MediaType.APPLICATION_JSON));

    String resposta = usuarioService.chamarApiExterna();
    assertEquals("success", resposta);
}
```

27. Teste de Comportamento de Método

```
java

@Test
public void testComportamentoMetodo() {
    Usuario usuario = new Usuario("Ana");
    usuarioService.cadastrar(usuario);

    verify(usuarioRepository, times(1)).save(usuario);
}
```

28. Teste de Reações a Mudanças de Estado

```
@Test
public void testReacaoMudancaEstado() {
    Usuario usuario = new Usuario("Beatriz", false);
    usuario.ativar();
    assertTrue(usuario.isAtivo(), "Usuário deve estar ativo após ativação");
}
```

29. Teste de Segurança de Dados

```
@Test
public void testSegurancaDados() {
    // Testa se os dados sensíveis estão criptografados
    String senhaCriptografada = usuarioService.criptografarSenha("minhaSenha");
    assertNotEquals("minhaSenha", senhaCriptografada);
}
```

30. Teste de Integração com Múltiplas Dependências

```
@Test
public void testIntegracaoComServicosExternos() {
    Usuario usuario = new Usuario("Fernando");
    usuarioService.cadastrar(usuario);

    // Verifica se o usuário foi cadastrado em um sistema externo
    assertTrue(externalService.usuarioCadastrado(usuario.getId()));
}
```

31. Teste de Validação de Emai

```
@Test
public void testEmailValido() {
    String email = "usuario@exemplo.com";
    assertTrue(EmailUtils.ehValido(email), "O email deve ser válido");
}
```

32. Teste de Cache

```
@Test
public void testCacheUsuario() {
    Usuario usuario = usuarioService.buscarPorId(1L);
    Usuario cachedUsuario = usuarioService.buscarPorId(1L); // Deve usar o cache
    assertSame(usuario, cachedUsuario, "Deve retornar o mesmo objeto do cache");
}
```

33. Teste de Mensagens de Erro

```
@Test
public void testMensagemErroQuandoUsuarioNaoEncontrado() {
    Exception exception = assertThrows(UsuarioNaoEncontradoException.class, () -> {
        usuarioService.buscarPorId(999L);
    });
    assertEquals("Usuário não encontrado", exception.getMessage());
}
```

34. Teste de Repositorio com Spring Data

```
@Test
public void testSalvarUsuario() {
    Usuario usuario = new Usuario("Lucas");
    usuarioRepository.save(usuario);
    assertNotNull(usuario.getId(), "O ID do usuário não deve ser nulo após salvar");
}
```

35. Teste de Atualização de Usuário

```
@Test
public void testAtualizarUsuario() {
    Usuario usuario = new Usuario("Gabriel");
    usuarioRepository.save(usuario);
    usuario.setNome("Gabriel Atualizado");
    usuarioService.atualizar(usuario);
    assertEquals("Gabriel Atualizado", usuarioService.buscarPorId(usuario.getId()).getNome());
}
```

36. Teste de Exclusão de Usuário

```
@Test
public void testExcluirUsuario() {
    Usuario usuario = new Usuario("Pedro");
    usuarioRepository.save(usuario);
    usuarioService.excluir(usuario.getId());
    assertThrows(UsuarioNaoEncontradoException.class, () ->
usuarioService.buscarPorId(usuario.getId()));
}
```

37. Teste de Dados Não Nulos

```
@Test
public void testDadosNaoNulos() {
    Usuario usuario = new Usuario("Clara", null);
    // Verifica se a exceção é lançada ao tentar cadastrar um usuário com dados nulos
    assertThrows(IllegalArgumentException.class, () -> usuarioService.cadastrar(usuario));
}
```

38. Teste de Resposta a Vários Usuários

```
@Test
public void testBuscarTodosUsuarios() {
    List<Usuario> usuarios = usuarioService.buscarTodos();
    assertFalse(usuarios.isEmpty(), "A lista de usuários não deve estar vazia");
}
```

39. Teste de Método Privado com Refletindo

```
@Test
public void testMetodoPrivado() throws Exception {
    Method metodoPrivado = UsuarioService.class.getDeclaredMethod("metodoPrivado");
    metodoPrivado.setAccessible(true);
    String resultado = (String) metodoPrivado.invoke(usuarioService);
    assertEquals("resultado esperado", resultado);
}
```

40. Teste de Validação com @Valid

```
@Test
public void testValidacaoUsuario() {
    Usuario usuario = new Usuario("", "senha");
    // Verifica se a exceção é lançada devido a falha na validação
    assertThrows(MethodArgumentNotValidException.class, () ->
usuarioService.cadastrar(usuario));
}
```

41. Teste de Conexão a API com RestTemplate

```
@Test
public void testChamadaApiComRestTemplate() {
    String response = restTemplate.getForObject("/api/external", String.class);
    assertNotNull(response, "A resposta da API não deve ser nula");
}
```

42. Teste de Ordenação

```
@Test
public void testOrdenacaoUsuarios() {
    List<Usuario> usuarios = usuarioService.buscarTodos();
    usuarios.sort(Comparator.comparing(Usuario::getNome));
    // Verifica se os usuários estão ordenados
    assertEquals("Ana", usuarios.get(0).getNome());
}
```

43. Teste de Filtragem

```
@Test
public void testFiltrarUsuariosAtivos() {
    List<Usuario> ativos = usuarioService.buscarAtivos();
    for (Usuario usuario : ativos) {
        assertTrue(usuario.isAtivo(), "Todos os usuários devem estar ativos");
    }
}
```

44. Teste de Lógica Complexa

```
@Test
public void testLogicaComplexa() {
    // Implementa lógica complexa para o teste
    boolean resultado = usuarioService.calcularLogicaComplexa();
    assertTrue(resultado, "A lógica complexa deve retornar verdadeiro");
}
```

45. Teste de Padrões de Projeto (Singleton)

```
@Test
public void testSingleton() {
    UsuarioService instancia1 = UsuarioService.getInstancia();
    UsuarioService instancia2 = UsuarioService.getInstancia();
    assertSame(instancia1, instancia2, "As instâncias devem ser iguais");
}
```

46. Teste de Retorno de Código HTTP

```
@Test
public void testRetornoHttp() throws Exception {
    mockMvc.perform(get("/usuarios/1"))
           .andExpect(status().isOk());
}
```

47. Teste de Propriedades de Configuração

```
@Test
public void testPropriedadesConfiguracao() {
    String valor = environment.getProperty("minha.propriedade");
    assertEquals("valorEsperado", valor, "A propriedade deve ter o valor esperado");
}
```

48. Teste de Injeção de Dependência

```
@Autowired
private UsuarioService usuarioService;

@Test
public void testInjecaoDependencia() {
    assertNotNull(usuarioService, "O serviço não deve ser nulo");
}
```

49. Teste de Componentes com @ComponentTest

```
@Test
@ComponentTest
public void testComponente() {
    assertTrue(componente.isAtivo(), "O componente deve estar ativo");
}
```

50. Teste de API com MockMvc e @AutoConfigureMockMvc

```
@Test
@AutoConfigureMockMvc
public void testApiUsuarios() throws Exception {
    mockMvc.perform(get("/api/usuarios"))
            .andExpect(status().isOk())
            .andExpect(content().contentType(MediaType.APPLICATION_JSON));
}
```

Conclusão

A utilização do JUnit em conjunto com boas práticas de teste é fundamental para garantir a qualidade e a manutenibilidade do código em projetos Java, especialmente em ambientes corporativos. Este artigo apresentou uma variedade de exemplos que abrangem desde testes simples de lógica até integrações complexas com bancos de dados e APIs. A implementação de testes não é apenas uma prática recomendada, mas uma necessidade para o desenvolvimento de software robusto.

Referências

- Documentação Oficial do JUnit 5
- Spring Boot Testing
- Mockito Documentation
- Java API Documentation
- Java Persistence API (JPA)

EducaCiência FastCode para a comunidade.