**Exemplos de teste:**

**Comparando Doubles:**

|  |
| --- |
| **import** org.junit.Assert;  **import** org.junit.Test;  **import** br.com.caelum.leilao.dominio.Lance;  **import** br.com.caelum.leilao.dominio.Leilao;  **import** br.com.caelum.leilao.dominio.Usuario;  **import** br.com.caelum.leilao.servico.Avaliador;  **public** **class** AvaliadorTest {    @Test  **public** **void** deveEntenderLancesEmOrdemCrescente() {//o método precisa ser public, não pode ser static e não retorna nada  // cenario: 3 lances em ordem crescente  Usuario joao = **new** Usuario("João");  Usuario jose = **new** Usuario("José");  Usuario maria = **new** Usuario("Maria");  Leilao leilao = **new** Leilao("Playstation 3 Novo");  leilao.propoe(**new** Lance(joao, 250.0));  leilao.propoe(**new** Lance(jose, 300.0));  leilao.propoe(**new** Lance(maria, 400.0));  // parte 2: acao  Avaliador leiloeiro = **new** Avaliador();  leiloeiro.avalia(leilao);    //parte 3: validação  // exibindo a saida - a classe Assert tem vários métodos que nos ajudam a conferir o resultado calculado  **double** maiorEsperado = 400;  **double** menorEsperado = 250;  /\*\*  \* AssertEquals  \* primeiro o valor esperado, depois o método que vai ser testado e trazer o valor para comparação e depois um delta,  \* devido o double ter problemas para arredondar, esta é a margem que ele pode deixar "passar"  \*/  Assert.*assertEquals*(maiorEsperado, leiloeiro.getMaiorLance(), 0.00001);  Assert.*assertEquals*(menorEsperado, leiloeiro.getMenorLance(), 0.00001);  }  } |

**Dificuldades nos testes de unidade**

Desenvolvedores que estão aprendendo a testar geralmente sentem dificuldades no momento de levantar e escrever cenários para o teste.

Lembre-se que um teste automatizado é muito parecido com um teste manual. Do mesmo jeito que você pensa no cenário de um teste manual (por exemplo, visitar a página de cadastro, preencher o campo CPF com "123", clicar no botão, e etc), você faz no automatizado.

Foque-se na classe que você está testando. Pense sobre o que você espera dela. Como ela deve funcionar? Se você passar tais parâmetros para ela, como ela deve reagir?

**TDD – Test Drive Development**

A ideia focal é escrever o teste antes do código do produção com a validação da regra de negócio para ver ele falhar, depois escrever o código em produção da maneira mais simples possível para ver ele passar no teste, depois que passou fica livre para você refatorar o código e melhorar o código, pois caso erre em algo o teste falhará.

O ideal é garantir que toda regra de negócio seja testada, mesmo que simples.

**Entendendo o @Before**

O método Before é executado antes de cada teste da classe.

**Conhecendo o @After**

Ao contrário do @Before, métodos anotados com @After são executados após a execução do método de teste.

Utilizamos métodos @After quando nossos testes consomem recursos que precisam ser finalizados. Exemplos podem ser testes que acessam banco de dados, abrem arquivos, abrem sockets, e etc.

(Apesar desses testes não serem mais considerados testes de unidade, afinal eles falam com outros sistemas, desenvolvedores utilizam JUnit para escrever testes de integração.

**@BeforeClass e @AfterClass**

Métodos anotados com @BeforeClass são executados apenas uma vez, antes de todos os métodos de teste.

O método anotado com @AfterClass, por sua vez, é executado uma vez, após a execução do último método de teste da classe.

Eles podem ser bastante úteis quando temos algum recurso que precisa ser inicializado apenas uma vez e que pode ser consumido por todos os métodos de teste sem a necessidade de ser reinicializado.

**Vantagens de testes de qualidade**

Nosso código de teste é altamente acoplado ao nosso código de produção. Isso significa que uma mudança no código de produção pode impactar profundamente em nosso código de testes.

Se não cuidarmos dos nossos testes, uma simples mudança pode impactar em MUITAS mudanças no código de testes.

É por isso que neste capítulo discutimos métodos auxiliares e test data builders. Todas elas são maneiras para fazer com que nosso código de testes evolua mais facilmente.