

Criando testes unitários em Java utilizando JUnit

O que aprenderão ?



- ▣ Como criar testes unitários do ZERO.
- ▣ Isolar o método de teste de dependências externas.
- ▣ Aplicar o básico de TDD.
- ▣ Criar builders de objetos para centralizar a criação de entidades.
- ▣ Breve introdução de como utilizar Mocks e PowerMocks.

Roteiro

1. Informações básicas do minicurso.
2. Visão geral: testes unitários.
3. Trabalhando com JUnit.
4. Introdução à TDD.
5. Usando Data Builders.
6. Trabalhando com Mocks.
7. Trabalhando com PowerMocks.
8. Conclusão

1.

Informações básicas do minicurso



Requisitos básicos:

- Conhecimento básico em: **Java**
- Ter instalado alguma IDE como: **Eclipse**
- O minicurso está disponível no seguinte link:
<https://github.com/francieleap/minicurso-junit>
- A documentação do JUnit está disponível no seguinte link: <https://junit.org/>

2.

**Visão geral:
testes unitários.**

O que são testes unitários?

- O **Teste Unitário** é uma modalidade de teste que é implementado com base no **menor elemento testável** (unidades) do software.
- Em **linguagens orientadas a objetos**, essa menor parte do código pode ser um **método** de uma classe.
- Etapas básicas para criação de um teste unitário:

CENÁRIO

AÇÃO

VALIDAÇÃO

Qual a importância do uso de testes unitários ?



- Evitar efeito borboleta:

“Uma coisa tão Simples, quanto o bater de asas de uma borboleta, pode causar um tufão do outro lado do mundo”

- Vantagens do uso de testes unitários:
 - Permitem maior cobertura de teste.
 - Previnem regressão.
 - Incentivam o refactoring.
 - Evitam longas sessões de debug.
 - Servem como documentação.

Aula-01: Testando sem usar framework

Importe no eclipse o projeto maven inicial do minicurso:

<https://github.com/francieleap/minicurso-junit/tree/master/Aula-01>

Adicionar este trecho:

```
29 public static void main(String[] args) {
30
31     //Cenário
32
33     LocacaoService service = new LocacaoService();
34     Usuario usuario = new Usuario("Usuário 01");
35     Filme filme = new Filme("Filme", 10, 5.0);
36
37     //Ação
38
39     Locacao locacao = service.alugarFilme(usuario, filme);
40
41     //Verificação
42
43     System.out.println(locacao.getValor() == 5);
44     System.out.println(DataUtils.isMesmaData(locacao.getDataLocacao(), new Date()));
45     System.out.println(DataUtils.isMesmaData(locacao.getDataRetorno(), DataUtils.adicionarDias(new Date(), 1)));
46 }
```

3.

**Trabalhando
com JUnit**

Alguns frameworks de testes unitários



LuaUnit

JUnit



Conhecendo o framework JUnit



- O **JUnit** é um framework de testes escrito por Erich Gamma e Kent Beck que facilita a implementação **de unidades de teste em Java**.
- JUnit é **open source** e oferece um conjunto de classes permitindo a fácil integração e execução regular de testes durante o processo de desenvolvimento
- Permite a **criação rápida de código de teste**.
- Checa os resultados dos testes e fornece uma **resposta imediata**.
- Pode ser utilizado da **linha de comando ou integrado** em IDE, e.x., Eclipse.

**Vamos para o
código ...**

Aula-02: Testando usando JUnit

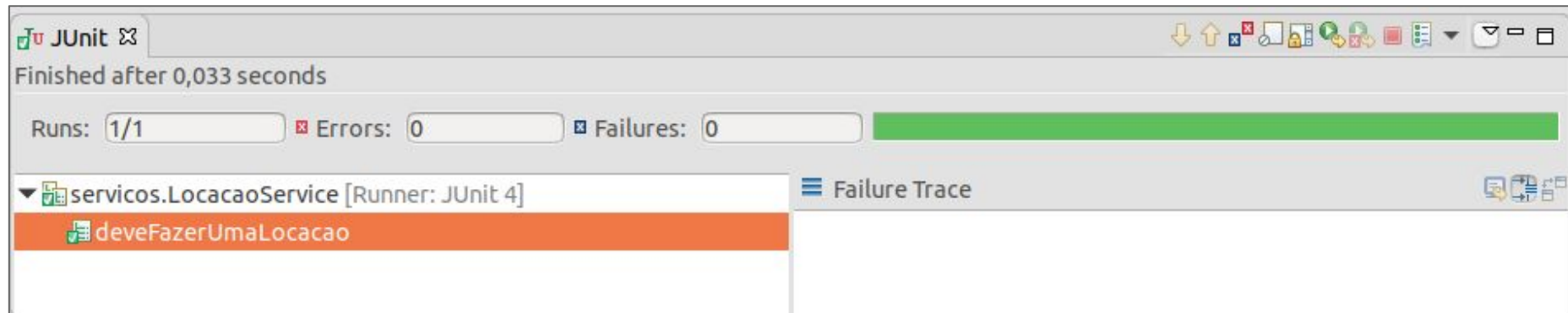
Importando JUnit 4.12:

```
1 <project xmlns="http://maven.apache.org/POM/4.0.0"
2     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
3     xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
4     <modelVersion>4.0.0</modelVersion>
5     <groupId>br.com</groupId>
6     <artifactId>aula-02</artifactId>
7     <version>0.0.1-SNAPSHOT</version>
8
9     <dependencies>
10         <dependency>
11             <groupId>junit</groupId>
12             <artifactId>junit</artifactId>
13             <version>4.12</version>
14         </dependency>
15     </dependencies>
16 </project>
```

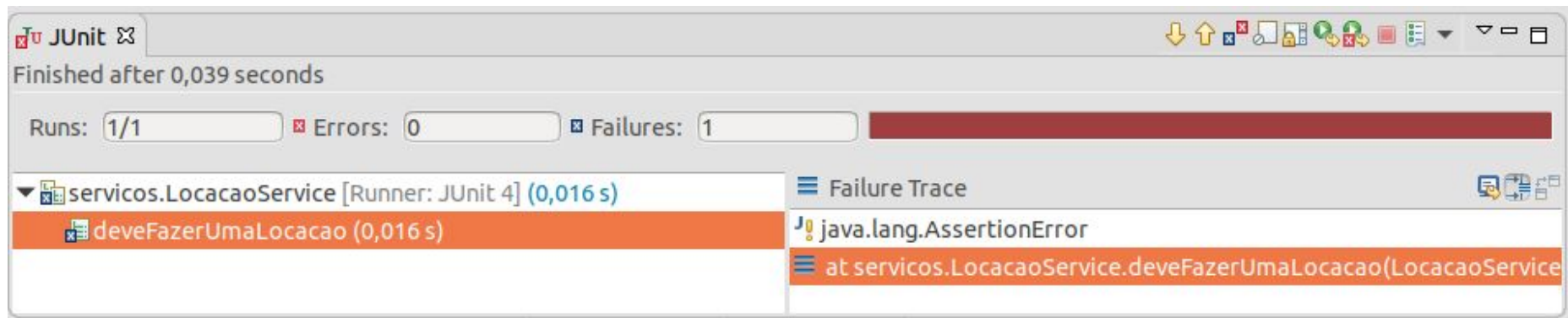
<https://github.com/francieleap/minicurso-junit/tree/master/Aula-02>

Aula-02: Testando usando JUnit

Teste com sucesso:



Teste com erro:



Aula-02: Testando usando JUnit

Adicionando primeiro teste:

```
@Test
public void deveFazerUmaLocacao() {

    //Cenário
    LocacaoService service = new LocacaoService();
    Usuario usuario = new Usuario("Usuário 01");
    Filme filme = new Filme("Filme", 10, 5.0);

    //Ação
    Locacao locacao = service.alugarFilme(usuario, filme);

    //Verificação
    Assert.assertTrue(locacao.getValor() == 5);
    Assert.assertTrue(DataUtils.isMesmaData(locacao.getDataLocacao(), new Date()));
    Assert.assertTrue(DataUtils.isMesmaData(locacao.getDataRetorno(), DataUtils.adicionarDias(new Date(), 1)));

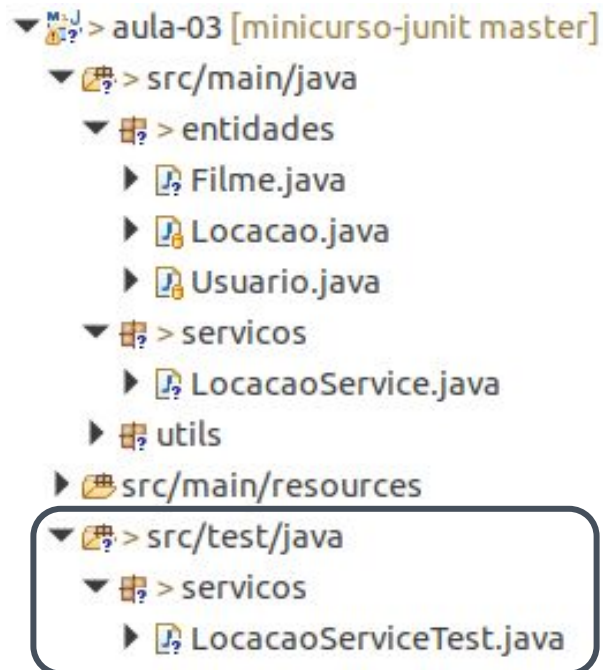
}
```

<https://github.com/francieleap/minicurso-junit/tree/master/Aula-02>

Aula-03: Organização dos arquivos de teste

Convenção:

- Uma classe de teste por classe a ser testada.
- Nome da classe de teste = NomeDaClasse + Test



```
4 import java.util.Date;
5
6 import org.junit.Assert;
7 import org.junit.Test;
8
9 import entidades.Filme;
10 import entidades.Locacao;
11 import entidades.Usuario;
12 import servicos.LocacaoService;
13 import utils.DataUtils;
14
15 public class LocacaoServiceTest {
16
17     @Test
18     public void deveFazerUmaLocacao() {
19
20         //Cenário
21     }
```

Aula-04: Trabalhando com Assertivas



Alguns exemplos de assertivas:

- `AssertTrue(condicao);`
- `AssertFalse(condicao);`
- `AssertEquals(valor esperado, valor atual);`
- `AssertNotEquals(valor esperado, valor atual);`
- `AssertArrayEquals(array esperado, array atual);`
- `AssertNull(objeto);`
- `AssertNotNull(objeto);`
- `AssertSame(objeto esperado, objeto atual);`
- `AssertNotSame(objeto esperado, objeto atual);`
- `AssertThat(atual, matcher);`
- `Fail();`

Aula-04: Trabalhando com Assertivas

Exemplos:

```
@Test
public void testeAssertivas() {

    Assert.assertTrue(1 == 1 );
    Assert.assertFalse(1 == 2);

    Assert.assertEquals(1, 1);
    Assert.assertEquals("teste", "teste");

    int numerosPares[] = new int[3];
    numerosPares[0] = 2;
    numerosPares[1] = 4;
    numerosPares[2] = 6;

    int copiaNumerosPares[] = new int[3];
    copiaNumerosPares[0] = 2;
    copiaNumerosPares[1] = 4;
    copiaNumerosPares[2] = 6;

    Assert.assertArrayEquals(numerosPares, copiaNumerosPares);

    Assert.assertEquals(Math.PI, 3.14, 0.01);
```

Aula-04: Trabalhando com Assertivas

Exemplos:

```
//Trabalhando com objetos
```

```
Usuario usuario1 = new Usuario("Usuario 1");  
Usuario usuario2 = new Usuario("Usuario 1");
```

```
Assert.assertEquals(usuario1, usuario2);
```

**Qual o resultado
desse teste ?**

SUCESSO OU ERRO

Aula-04: Trabalhando com Assertivas

Exemplos:

```
//Trabalhando com objetos
```

```
Usuario usuario1 = new Usuario("Usuario 1");  
Usuario usuario2 = new Usuario("Usuario 1");  
  
Assert.assertEquals(usuario1, usuario2);
```

Qual o resultado
desse teste ?

SUCESSO OU ERRO

A diagram consisting of six arrows pointing towards the word "ERRO" in the text "SUCESSO OU ERRO". The arrows originate from the top, bottom, and sides, converging on the word "ERRO".

Como não foi implementado o método **equals** na classe Usuario o assertEquals compara os objetos a nível de **instância**.

Aula-04: Trabalhando com Assertivas

Exemplos:

```
21 @Override
22 public boolean equals(Object obj) {
23     if (this == obj)
24         return true;
25     if (obj == null)
26         return false;
27     if (getClass() != obj.getClass())
28         return false;
29     Usuario other = (Usuario) obj;
30     if (nome == null) {
31         if (other.nome != null)
32             return false;
33     } else if (!nome.equals(other.nome))
34         return false;
35     return true;
36 }
37
```

Solução

Aula-04: Trabalhando com Assertivas



Exemplos:

```
//Trabalhando com objetos

Usuario usuario1 = new Usuario("Usuario 1");
Usuario usuario2 = new Usuario("Usuario 1");

Assert.assertEquals(usuario1, usuario2);

Usuario usuario3 = usuario2;

Assert.assertSame(usuario2, usuario3);

Usuario usuario4 = null;

Assert.assertNull(usuario4);
```


Aula-04: Trabalhando com Assertivas

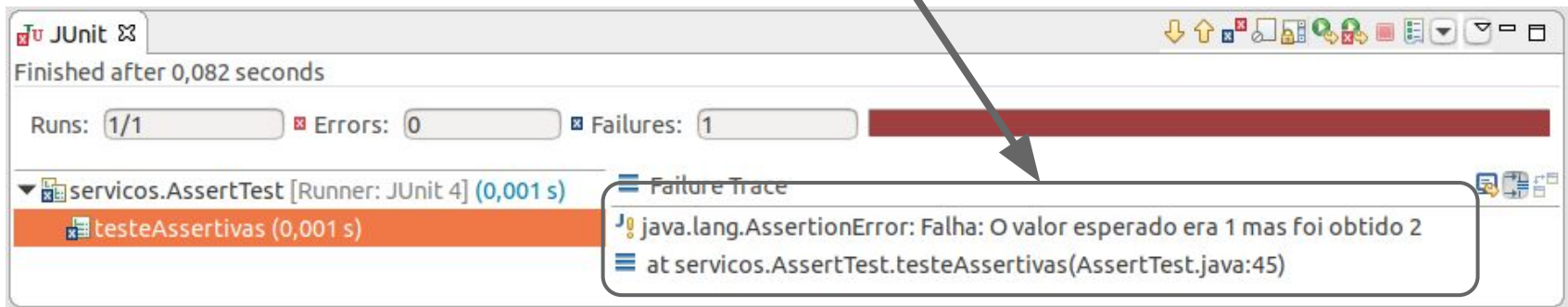
Exemplos:

```
//Customizando mensagem de exceção
```

```
int a = 1; int b=2;
```

```
String mensagem = "Falha: O valor esperado era %d mas foi obtido %d";
```

```
Assert.assertTrue(String.format(mensagem, a, b), a == b);
```



Aula-04: Trabalhando com Assertivas



Exemplos:

```
//Trabalhando com fail()

try {
    // faz um teste que deveria dar exception...
    Assert.assertTrue((2/0) == 1);
    Assert.fail();

} catch (Exception e) {
    Assert.assertTrue(true);
}
```

A ideia do **fail()** é ser usado para interromper a execução quando a linha em que ele é usado jamais deveria ter sido alcançada.

<https://github.com/francieleap/minicurso-junit/tree/master/Aula-04>

Aula-05: Trabalhando com AssertThat e Hamcrest



- O método **AssertThat**(valor atual, matcher) oferece uma maneira melhor de escrever asserções.
- O **Hamcrest** é um framework que possibilita a criação de regras de verificação(**matchers**) de forma declarativa.
- Um matcher Hamcrest é um objeto que:
 - Reporta se um dado objeto satisfaz um determinado critério;
 - Pode descrever este critério; e
 - É capaz de descrever porque um objeto não satisfaz um determinado critério.

Aula-05: Trabalhando com AssertThat e Hamcrest

Porque usar AssertThat?

Legibilidade:

```
assertThat(actual, is(equalTo(expected)));
```

Melhores mensagens de erro:

```
assertThat(actual, containsString(expected));  
java.lang.AssertionError:  
Expected: a string containing "abc" got: "def"
```

Tipo de segurança:

```
assertEquals("abc", 123);  
//compila mais falha
```

```
assertThat(123, is("abc"));  
//não compila
```

Flexibilidade:

```
assertThat("test", anyOf(is("test2"), containsString("te")));  
  
assertThat("test", anyOf(is("test2"), containsString("ca")));  
java.lang.AssertionError: Expected: (is "test2" or a string containing "ca") got:  
"test"
```

Aula-05: Trabalhando com AssertThat e Hamcrest



Alguns exemplos de matchers:

- `CoreMatchers.is();`
- `CoreMatchers.any();`
- `CoreMatchers.describeAs();`
- `CoreMatchers.allOf();`
- `CoreMatchers.anyOf();`
- `CoreMatchers.not();`
- `CoreMatchers.equalTo();`
- `CoreMatchers.instanceOf();`
- `CoreMatchers.notNullValue();`
- `CoreMatchers.nullValue();`
- `CoreMatchers.sameInstance();`

<https://github.com/francieleap/minicurso-junit/tree/master/Aula-05>

Aula-05: Trabalhando com AssertThat e Hamcrest



Exemplos:

```
@Test
public void testeAssertivaThat() {

    Assert.assertThat("123", is("123"));

    Assert.assertThat(123, any(Integer.class));

    Assert.assertThat(123, describedAs("Inteiro igual a %0", equalTo(123), 123));

    Assert.assertThat("123", allOf(isA(String.class), equalTo("123")));

    Assert.assertThat("123", anyOf(isA(String.class), equalTo("111")));

    Usuario usuario = new Usuario();

    Assert.assertThat(usuario, instanceof(Usuario.class));

    String texto = "texto";
    Assert.assertThat(texto, notNullValue(String.class));

    Assert.assertThat(usuario, sameInstance(usuario));
}
```

Aula-05: Trabalhando com AssertThat e Hamcrest

Exemplos:

```
3 import org.hamcrest.Description;
4 import org.hamcrest.TypeSafeMatcher;
5
6 public class CustomMatcher extends TypeSafeMatcher<String> {
7
8     private String letter;
9
10    private CustomMatcher(String letter) {
11        this.letter = letter;
12    }
13
14    public void describeTo(Description description) {
15        description.appendValue("Esperava uma palavra que começa com " + this.letter);
16    }
17
18    @Override
19    protected boolean matchesSafely(String item) {
20        String letra = String.valueOf(item.charAt(0));
21        return letra.equals(this.letter);
22    }
23
24    public static CustomMatcher startWithLetter(String letter) {
25        return new CustomMatcher(letter);
26    }
27 }
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43 //Customizando matchers
44 Assert.assertThat("Aluno", CustomMatcher.startWithLetter("A"));
```

Aula-06: Formas de dividir um teste

Convenção:

- Uma assertiva para cada método de teste. Dessa forma o teste não vai parar caso algum dê erro.

```
@Test
public void deveChecarValorLocacao() {

    //Cenário
    LocacaoService service = new LocacaoService();
    Usuario usuario = new Usuario("Usuário 01");
    Filme filme = new Filme("Filme", 10, 5.0);

    //Ação
    Locacao locacao = service.alugarFilme(usuario, filme);

    //Verificação
    Assert.assertTrue(locacao.getValor() == 5);
}
```

```
@Test
public void deveChecarDataLocacao() {

    //Cenário
    LocacaoService service = new LocacaoService();
    Usuario usuario = new Usuario("Usuário 01");
    Filme filme = new Filme("Filme", 10, 5.0);

    //Ação
    Locacao locacao = service.alugarFilme(usuario, filme);

    //Verificação
    Assert.assertTrue(DataUtils.
        isMesmaData(locacao.getDataLocacao(), new Date()));
}
```


Aula-07: Tratamento de exceções

Nova regra:

- Não deve alugar filme sem estoque.

```
public Locacao alugarFilme(Usuario usuario, Filme filme) throws Exception {  
    //Validação filme sem estoque  
  
    if (filme.getEstoque() == 0) {  
        throw new Exception("Filme sem estoque.");  
    }  
  
    Locacao locacao = new Locacao();  
    locacao.setFilme(filme);  
    locacao.setUsuario(usuario);  
    locacao.setDataLocacao(new Date());  
    locacao.setValor(filme.getPrecoLocacao());  
  
    //Entrega no dia seguinte  
    Date dataEntrega = new Date();  
    dataEntrega = adicionarDias(dataEntrega, 1);  
    locacao.setDataRetorno(dataEntrega);  
  
    return locacao;  
}
```

```
@Test  
public void deveChecarValorLocacao() {  
  
    //Cenário  
    LocacaoService service = new LocacaoService();  
    Usuario usuario = new Usuario("Usuário 01");  
    Filme filme = new Filme("Filme", 10, 5.0);  
  
    //Ação  
    Locacao locacao;  
    try {  
        locacao = service.alugarFilme(usuario, filme);  
  
        //Verificação  
        Assert.assertTrue(locacao.getValor() == 5);  
    } catch (Exception e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
}
```


Aula-07: Tratamento de exceções

Exemplos:

```
@Test
public void deveChecarValorLocacao() {

    //Cenário
    LocacaoService service = new LocacaoService();
    Usuario usuario = new Usuario("Usuário 01");
    Filme filme = new Filme("Filme", 0, 5.0);

    //Ação
    Locacao locacao;
    try {
        locacao = service.alugarFilme(usuario, filme);

        //Verificação
        Assert.assertTrue(locacao.getValor() == 5);

    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    }

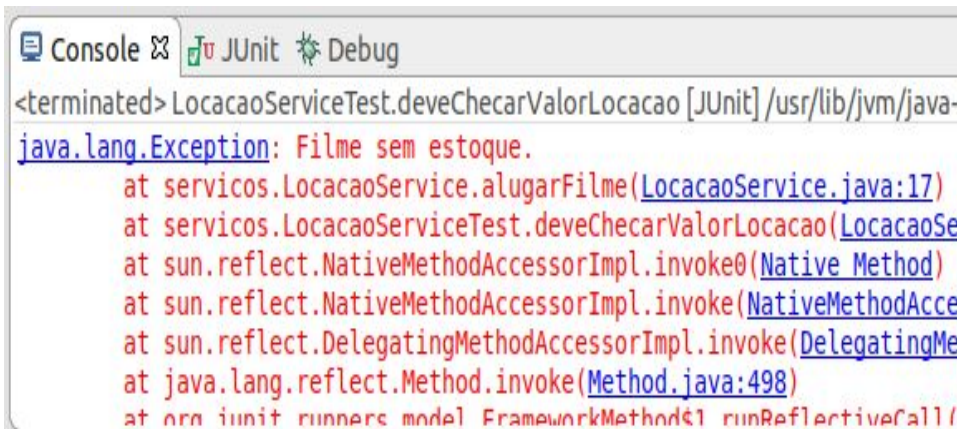
}
```

**Qual o resultado
desse teste ao
zerar o estoque?**

SUCESSO OU ERRO

Aula-07: Tratamento de exceções

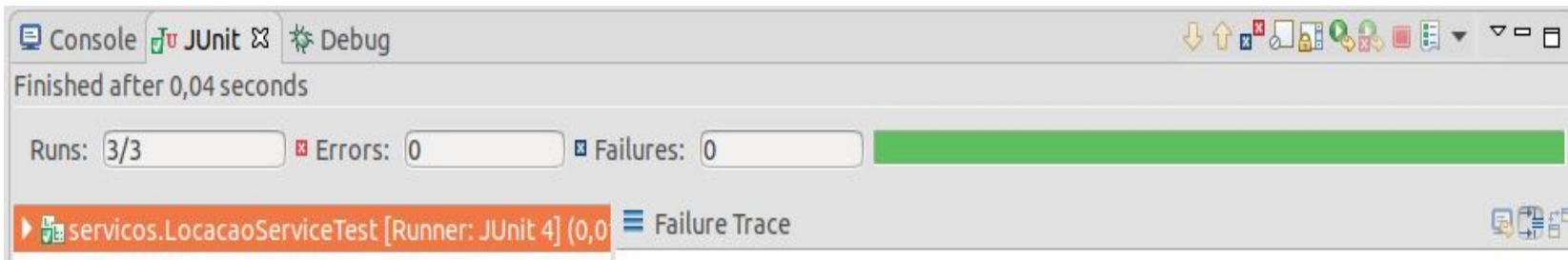
Exemplos:



```
<terminated> LocacaoServiceTest.deveChecarValorLocacao [JUnit] /usr/lib/jvm/java-  
java.lang.Exception: Filme sem estoque.  
    at servicos.LocacaoService.alugarFilme(LocacaoService.java:17)  
    at servicos.LocacaoServiceTest.deveChecarValorLocacao(LocacaoSe  
    at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)  
    at sun.reflect.NativeMethodAccessorImpl.invoke(NativeMethodAcce  
    at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMe  
    at java.lang.reflect.Method.invoke(Method.java:498)  
    at org.junit.runners.model.FrameworkMethod$1.runReflectiveCall(/
```

Qual o resultado
desse teste ao
zerar o estoque?

SUCESSO OU ERRO



```
Console JUnit Debug  
Finished after 0,04 seconds  
Runs: 3/3 Errors: 0 Failures: 0  
servicos.LocacaoServiceTest [Runner: JUnit 4] (0,0 Failure Trace
```

Aula-07: Tratamento de exceções

Exemplos:

```
//Tratamento exceção não esperada
@Test
public void deveChecarValorLocacao() {
    //Cenário
    LocacaoService service = new LocacaoService();
    Usuario usuario = new Usuario("Usuário 01");
    Filme filme = new Filme("Filme", 0, 5.0);

    //Ação
    Locacao locacao;
    try {
        locacao = service.alugarFilme(usuario, filme);

        //Verificação
        Assert.assertTrue(locacao.getValor() == 5);
    } catch (Exception e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
        Assert.fail();
    }
}
```

Exceção não esperada!

Solução 1

Aula-07: Tratamento de exceções

Exemplos:

```
//Tratamento de exceção não esperada.
@Test
public void deveChecarValorLocacao() throws Exception {

    //Cenário
    LocacaoService service = new LocacaoService();
    Usuario usuario = new Usuario("Usuário 01");
    Filme filme = new Filme("Filme", 0, 5.0);

    //Ação
    Locacao locacao = service.alugarFilme(usuario, filme);

    //Verificação
    Assert.assertTrue(locacao.getValor() == 5);
}
```

Exceção não esperada!

Solução 2

Console JUnit Debug

Finished after 0,047 seconds

Runs: 3/3 Errors: 1 Failures: 0

Failure Trace

java.lang.Exception: Filme sem estoque.
at services.LocacaoService.alugarFilme(LocacaoService.java:17)
at services.LocacaoServiceTest.deveChecarValorLocacao(LocacaoServiceTest.java:17)

deveChecarDataRetornoLocacao (0,009 s)
deveChecarDataLocacao (0,000 s)
deveChecarValorLocacao (0,007 s)

Aula-07: Tratamento de exceções

Exemplos:

```
//Tratamento de exceção esperada.
@Test(expected=Exception.class)
public void deveChecarFilmeSemEstoque() throws Exception {

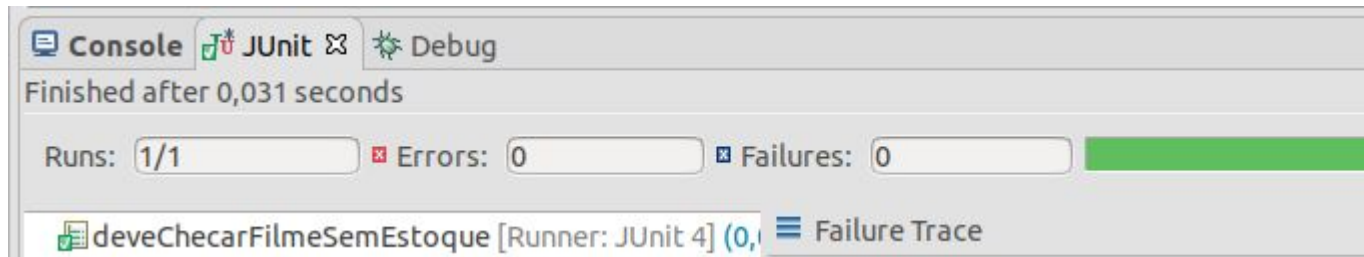
    //Cenário
    LocacaoService service = new LocacaoService();
    Usuario usuario = new Usuario("Usuário 01");
    Filme filme = new Filme("Filme", 0, 5.0);

    //Ação
    Locacao locacao = service.alugarFilme(usuario, filme);

    //Verificação
    Assert.assertTrue(locacao.getValor() == 5);
}
```

Exceção esperada!

Solução 1



Aula-07: Tratamento de exceções

Exemplos:

Exceção esperada!

```
//Tratamento de exceção esperada.
@Test
public void deveChecarFilmeSemEstoque_() {

    //Cenário
    LocacaoService service = new LocacaoService();
    Usuario usuario = new Usuario("Usuário 01");
    Filme filme = new Filme("Filme", 0, 5.0);

    //Ação
    Locacao locacao;
    try {
        locacao = service.alugarFilme(usuario, filme);

        //Verificação
        Assert.assertTrue(locacao.getValor() == 5);

        Assert.fail("Deveria ter lançado uma exceção!");
    } catch (Exception e) {

        Assert.assertThat(e.getMessage(), CoreMatchers.is("Filme sem estoque."));
    }
}
```

Solução 2

Aula-08: Usando as anotações Before e After

Exemplos:

```
public class LocacaoServiceTest {  
  
    LocacaoService service ;  
  
    @Before  
    public void inicializa() {  
        System.out.println("@Before");  
        service = new LocacaoService();  
    }  
  
    @After  
    public void encerra() {  
        System.out.println("@After");  
    }  
  
    @BeforeClass  
    public static void inicializaClasse() {  
        System.out.println("@BeforeClass");  
    }  
  
    @AfterClass  
    public static void encerraClasse() {  
        System.out.println("@AfterClass!");  
    }  
}
```

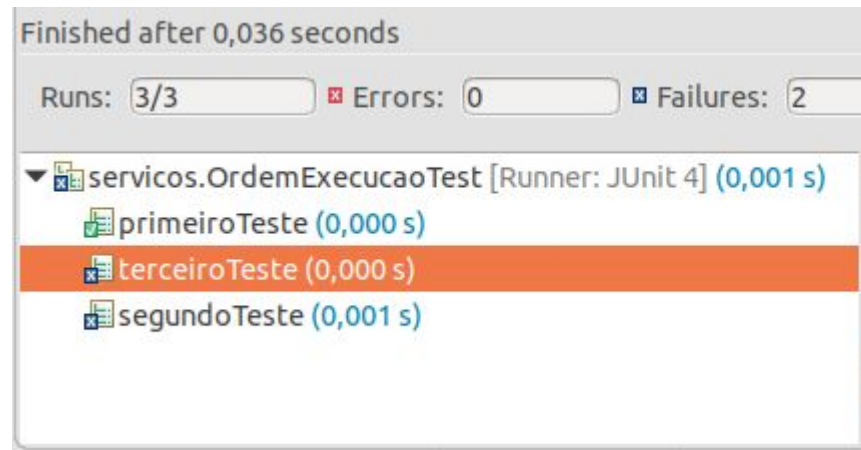


Console JUnit Debug
<terminated> LocacaoServiceTest (5)
@BeforeClass
@Before
@After
@Before
@After
@Before
@After
@Before
@After
@Before
@After
@AfterClass!

Aula-09: Ordem de execução dos testes

Exemplos:

```
7 public class OrdemExecucaoTest {
8
9     public static int contador = 0;
10
11     @Test
12     public void primeiroTeste() {
13         contador = contador + 1;
14         Assert.assertThat(contador, CoreMatchers.is(1));
15     }
16
17     @Test
18     public void segundoTeste() {
19         contador = contador + 1;
20         Assert.assertThat(contador, CoreMatchers.is(2));
21     }
22
23     @Test
24     public void terceiroTeste() {
25         contador = contador + 1;
26         Assert.assertThat(contador, CoreMatchers.is(3));
27     }
28
29 }
```



Aula-09: Ordem de execução dos testes

Exemplos:

```
9 @FixMethodOrder(MethodSorters.NAME_ASCENDING)
10 public class OrdemExecucaoTest {
11
12     public static int contador = 0;
13
14     @Test
15     public void teste1() {
16         contador = contador + 1;
17         Assert.assertThat(contador, CoreMatchers.is(1));
18     }
19
20     @Test
21     public void teste2() {
22         contador = contador + 1;
23         Assert.assertThat(contador, CoreMatchers.is(2));
24     }
25
26     @Test
27     public void teste3() {
28         contador = contador + 1;
29         Assert.assertThat(contador, CoreMatchers.is(3));
30     }
31
32 }
```

Solução

@FixMethodOrder
(MethodSorters.NAME_ASCENDING)

Finished after 0,029 seconds

Runs: 3/3 Errors: 0 Failures: 0

▼ servicicos.OrdemExecucaoTest [Runner: JUnit 4] (0,000 s) Failure Tr

- teste1 (0,000 s)
- teste2 (0,000 s)
- teste3 (0,000 s)

4.

Introdução à TDD

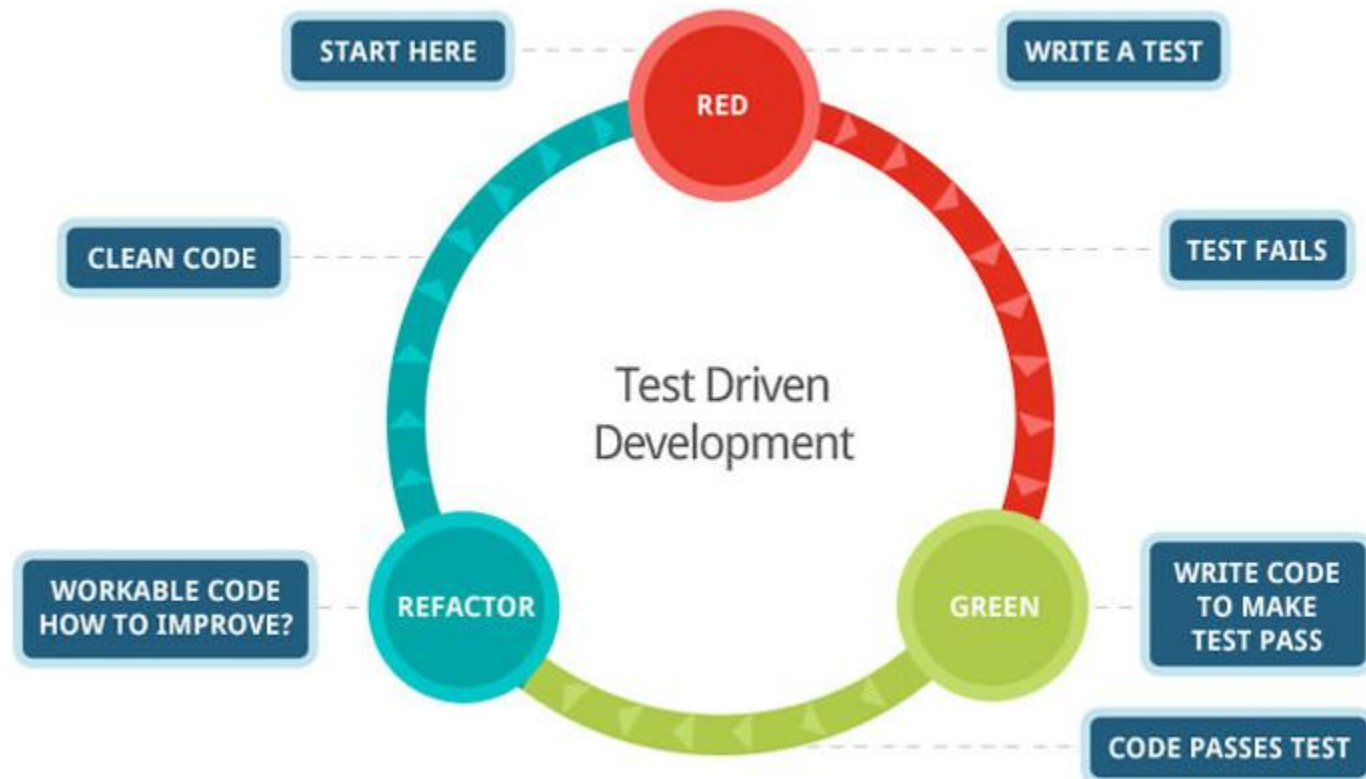
Introdução à TDD



- **Test Driven Development (TDD)** ou desenvolvimento guiado por testes é uma técnica de desenvolvimento de software que se relaciona com o conceito de **verificação** e **validação**.
- A ideia é bem simples: escreva os testes antes mesmo de escrever o código de produção.

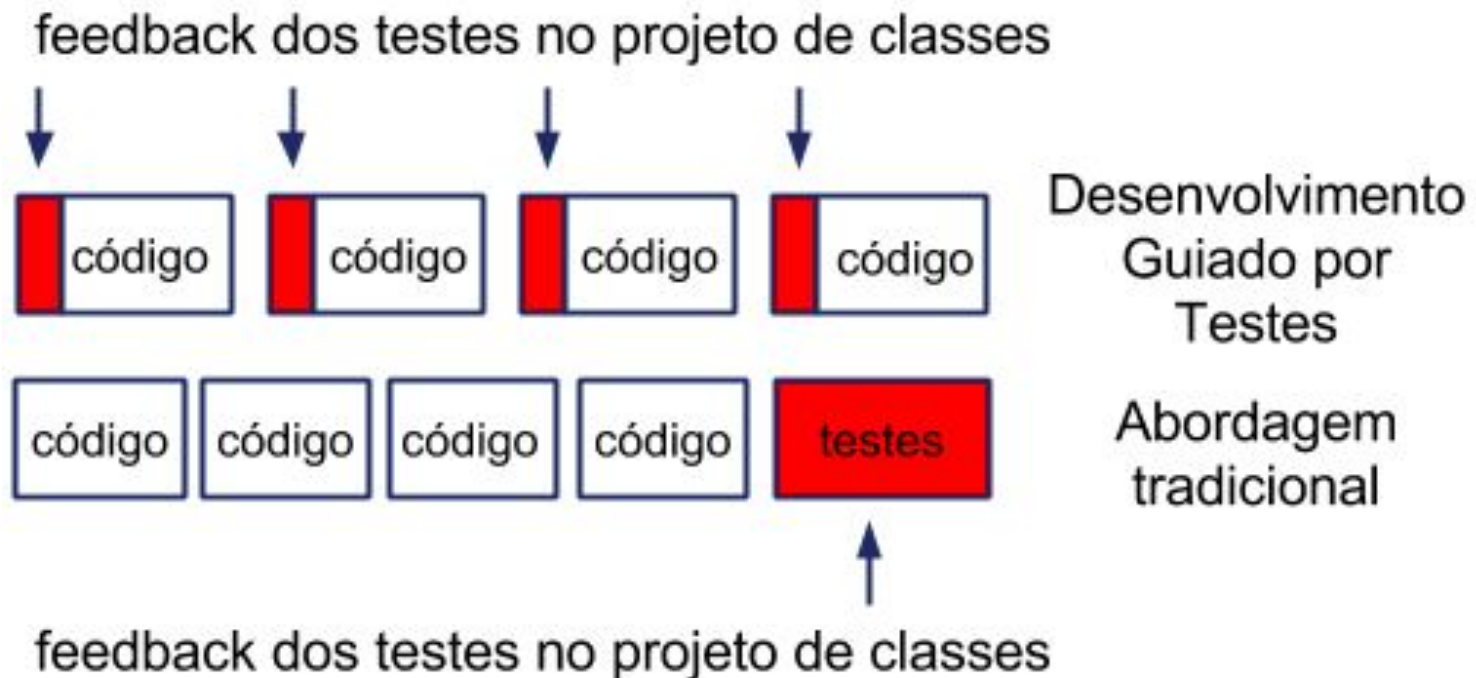
Introdução à TDD

- O ciclo do TDD é conhecido como: RED-GREEN-REFACTOR.



Introdução à TDD

- Qual a diferença entre fazer TDD e escrever o teste depois ?



Introdução à TDD

■ Alguns desafios:

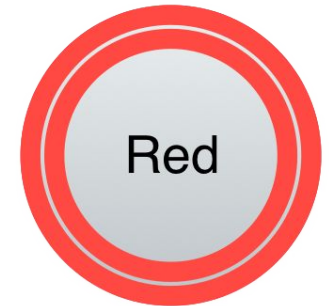
- Resistência do programador em adotar esta prática.
- A curva de aprendizagem é um pouco extensa.
- Para linguagens que não possuam frameworks o TDD pode se tornar pesado e difícil.
- O TDD é difícil de ser implementado em códigos legados.

**Vamos para o
código ...**

Aula-10: Test Driven Development - TDD



Exemplo: Vamos criar a CalculadoraTest.



```
7 public class CalculadoraTest {
8
9     @Test
10     public void deveSomarDoisNumeros() {
11         //cenário
12         int a = 5;
13         int b = 3;
14
15         Calculadora calculadora = new Calculadora();
16
17         //ação
18         int resultado = calculadora.somar(a,b);
19
20         //verificacao
21         Assert.assertThat(resultado, CoreMatchers.is(8));
22
23     }
24 }
```

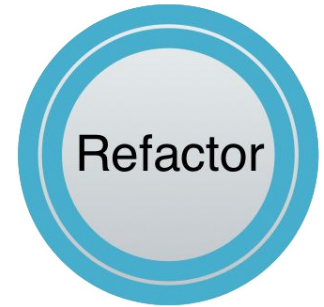

Aula-10: Test Driven Development - TDD

Exemplo: Vamos criar a Calculadora.



```
3 public class Calculadora {  
4  
5     public int somar(int a, int b) {  
6  
7         return a + b;  
8     }  
9  
10 }
```

Aula-10: Test Driven Development - TDD



Exemplo: Vamos refatorar o código.

```
8 public class CalculadoraTest {
9
10     Calculadora calculadora;
11
12     @Before
13     public void inicializa() {
14         calculadora = new Calculadora();
15     }
16
17     @Test
18     public void deveSomarDoisNumeros() {
19         //cenário
20         int a = 5;
21         int b = 3;
22
23         //ação
24         int resultado = calculadora.somar(a,b);
25
26         //verificacao
27         Assert.assertThat(resultado, CoreMatchers.is(8));
28     }
29 }
30 }
```

Aula-10: Test Driven Development - TDD

A red circle with a thick red border and a light gray center, containing the word "Red" in black text.

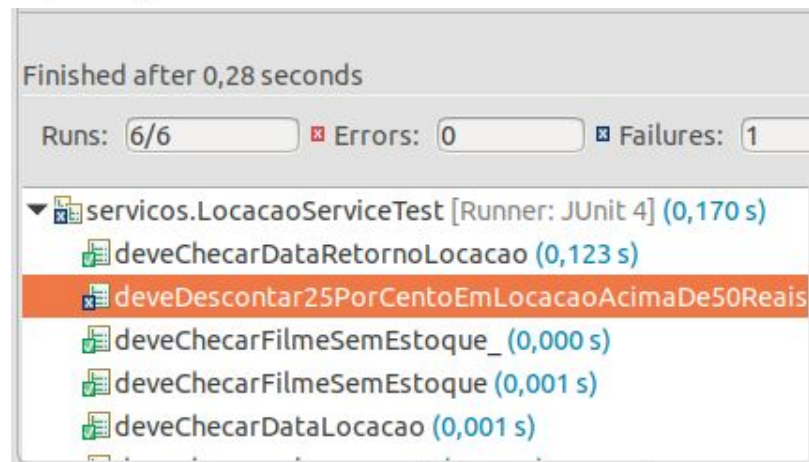
Nova regra:

- Locação acima de 50 reais deve receber 25% de desconto.

```
@Test
public void deveDescontar25PorCentoEmLocacaoAcimaDe50Reais() throws Exception {
    //Cenário
    Usuario usuario = new Usuario("Usuário 01");
    Filme filme = new Filme("Filme", 10, 60.0);

    //Ação
    Locacao locacao = service.alugarFilme(usuario, filme);

    //Verificação
    Assert.assertThat(locacao.getValor(), CoreMatchers.is(45.0));
}
```



Aula-10: Test Driven Development - TDD



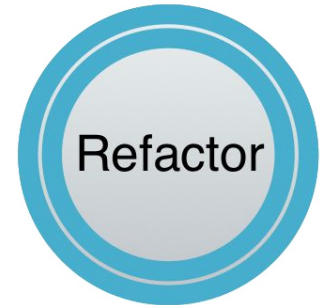
```
public Locacao alugarFilme(Usuario usuario, Filme filme) throws Exception {  
    //Validação filme sem estoque  
  
    if (filme.getEstoque() == 0) {  
        throw new Exception("Filme sem estoque.");  
    }  
  
    Locacao locacao = new Locacao();  
    locacao.setFilme(filme);  
    locacao.setUsuario(usuario);  
    locacao.setDataLocacao(new Date());  
  
    if (filme.getPrecoLocacao() > 50.0) {  
        Double desconto = filme.getPrecoLocacao() * 0.25;  
        locacao.setValor(filme.getPrecoLocacao() - desconto);  
    } else {  
        locacao.setValor(filme.getPrecoLocacao());  
    }  
  
    //Entrega no dia seguinte  
    Date dataEntrega = new Date();  
    dataEntrega = adicionarDias(dataEntrega, 1);  
    locacao.setDataRetorno(dataEntrega);  
  
    return locacao;  
}
```

Finished after 0,117 seconds

Runs: 6/6 Errors: 0 Failures: 0

▼ servicos.LocacaoServiceTest [Runner: JUnit 4] (0,001 s)
 ✓ deveChecarDataRetornoLocacao (0,000 s)
 ✓ deveDescontar25PorCentoEmLocacaoAcimaDe50Reais
 ✓ deveChecarFilmeSemEstoque_ (0,000 s)
 ✓ deveChecarFilmeSemEstoque (0,001 s)
 ✓ deveChecarDataLocacao (0,000 s)

Aula-10: Test Driven Development - TDD



```
public Locacao alugarFilme(Usuario usuario, Filme filme) throws Exception {
    //Validação filme sem estoque

    if (filme.getEstoque() == 0) {
        throw new Exception("Filme sem estoque.");
    }

    Locacao locacao = new Locacao();
    locacao.setFilme(filme);
    locacao.setUsuario(usuario);
    locacao.setDataLocacao(new Date());
    locacao.setValor(calculaValorLocacao(filme.getPrecoLocacao()));

    //Entrega no dia seguinte
    Date dataEntrega = new Date();
    dataEntrega = adicionarDias(dataEntrega, 1);
    locacao.setDataRetorno(dataEntrega);

    return locacao;
}

private Double calculaValorLocacao(Double precoFilme) {

    if (precoFilme > 50) {
        Double desconto = precoFilme * 0.25;
        return precoFilme-desconto;
    }

    return precoFilme;
}
```

Finished after 0,117 seconds

Runs: 6/6 Errors: 0 Failures: 0

▼ servicos.LocacaoServiceTest [Runner: JUnit 4] (0,001 s)

- deveChecarDataRetornoLocacao (0,000 s)
- deveDescontar25PorCentoEmLocacaoAcimaDe50Reais**
- deveChecarFilmeSemEstoque_ (0,000 s)
- deveChecarFilmeSemEstoque (0,001 s)
- deveChecarDataLocacao (0,000 s)

5.

Usando Data Builders

Usando Data Builders

- **Builder**, é um **padrão de projeto** de software criacional que permite a separação da construção de um objeto complexo da sua representação.
- Este padrão permite que o mesmo processo de construção do objeto possa criar **diferentes representações**.
- No contexto de teste unitário, o padrão **Data Builder** é usado para criar dados de teste de forma automatizada que facilitam a leitura dos testes de unidade.

Usando Data Builders



- O padrão pode ser implementado da seguinte maneira:
 1. Para cada classe de domínio, cria-se uma **classe Builder correspondente**.
 2. No construtor da classe Builder, **inicialize** cada propriedade com um **valor default**.
 3. Para cada atributo da classe adiciona-se um **método precedido com With ou Com**, em português, que altere a propriedade e retorne o próprio Builder.
 4. E por fim, adicione um **método build()** que retorne uma nova instância da classe de domínio com os valores passados.

Usando Data Builders

■ Exemplo:

```
User aUser = new User();  
aUser.setName("John");  
aUser.setPassword("42abc");
```



```
User aUser = UserBuiler.aUser()  
    .withName("John")  
    .withPassword("42abc")  
    .build();
```

**Vamos para o
código ...**

Aula-11: Usando Data Builders

Exemplo:

```
1 package builders;                                     //Cenário
2
3 import entidades.Usuario;                             Usuario usuario = UsuarioBuilder.umUsuario().build();
4
5 public class UsuarioBuilder {
6
7     private Usuario usuario;
8
9     private UsuarioBuilder() {};
10
11     public static UsuarioBuilder umUsuario() {
12
13         UsuarioBuilder builder = new UsuarioBuilder();
14         builder.usuario = new Usuario();
15         builder.usuario.setNome("Usuario 1");
16         return builder;
17     }
18
19     public Usuario build() {
20         return usuario;
21     }
22 }
```

Aula-11: Usando Data Builders

Exemplo:

```
5 public class FilmeBuilder {
6
7     private Filme filme;
8     private FilmeBuilder() {};
9
10    public static FilmeBuilder umFilme() {
11
12        FilmeBuilder builder = new FilmeBuilder();
13        builder.filme = new Filme();
14        builder.filme.setNome("A freira");
15        builder.filme.setEstoque(10);
16        builder.filme.setPrecoLocacao(5.0);
17        return builder;
18    }
19
20    public FilmeBuilder comEstoque(Integer valor) {
21        filme.setEstoque(valor);
22        return this;
23    }
24
25    public FilmeBuilder comPrecoLocacao(Double valor) {
26        filme.setPrecoLocacao(valor);
27        return this;
28    }
29
30    public Filme build() {
31        return filme;
32    }
33 }
```

//Cenário

```
Filme filme = FilmeBuilder.umFilme().comEstoque(0).build();
```

Aula-11: Usando Data Builders

Exemplo:

//Cenário

Locacao locacao = LocacaoBuilder.umaLocacao().build();

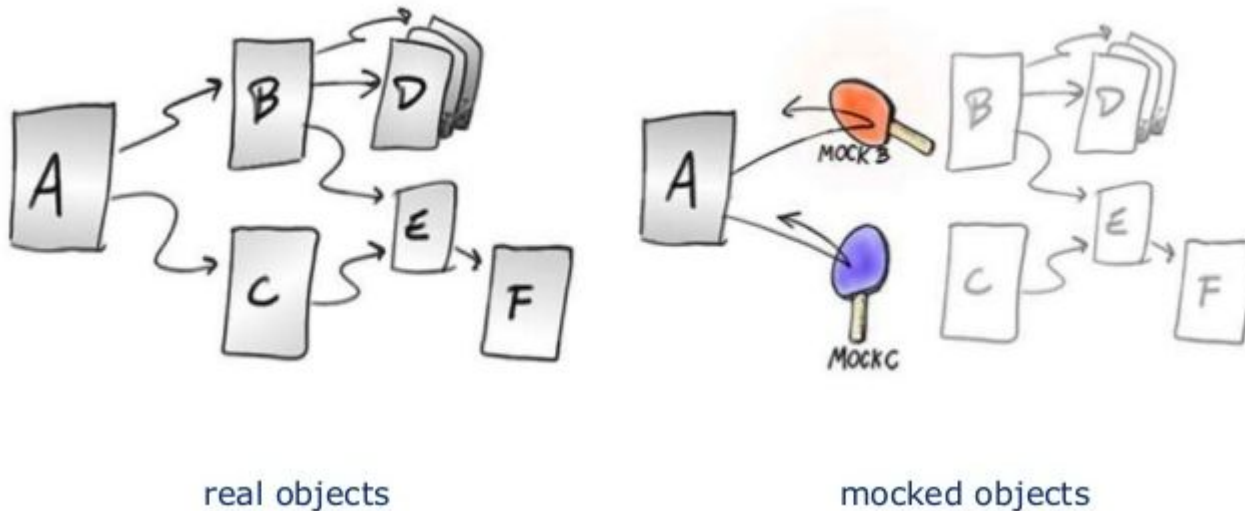
```
8 public class LocacaoBuilder {
9
10     private Locacao locacao;
11     private LocacaoBuilder() {};
12
13     public static LocacaoBuilder umaLocacao() {
14
15         LocacaoBuilder builder = new LocacaoBuilder();
16         builder.locacao.setDataLocacao(new Date());
17         builder.locacao.setDataRetorno(DataUtils.adicionarDias(new Date(), 7));
18         builder.locacao.setUsuario(UsuarioBuilder.umUsuario().build());
19         builder.locacao.setFilme(FilmeBuilder.umFilme().build());
20         builder.locacao.setValor(30.00);
21         return builder;
22     }
23
24     public LocacaoBuilder comDataRetorno(Date data) {
25         locacao.setDataRetorno(data);
26         return this;
27     }
28
29     public Locacao build() {
30         return locacao;
31     }
32
33 }
```

6.

Trabalhando com Mocks

Trabalhando com Mocks

- **Objetos Mock**, do inglês Mock object em desenvolvimento de software são objetos que **simulam o comportamento de objetos reais** de forma controlada.



Trabalhando com Mocks



- **Pode-se utilizar objetos mocks quando o objeto:**
 - Gera resultados não determinísticos. (e.x. a hora ou temperatura atual);
 - Tem estados que são difíceis de criar ou reproduzir (e.x. erro de comunicação da rede);
 - É lento (e.x. um banco de dados completo que precisa ser inicializado antes do teste);
 - Ainda não existe ou pode ter comportamento alterado;

**Vamos para o
código ...**

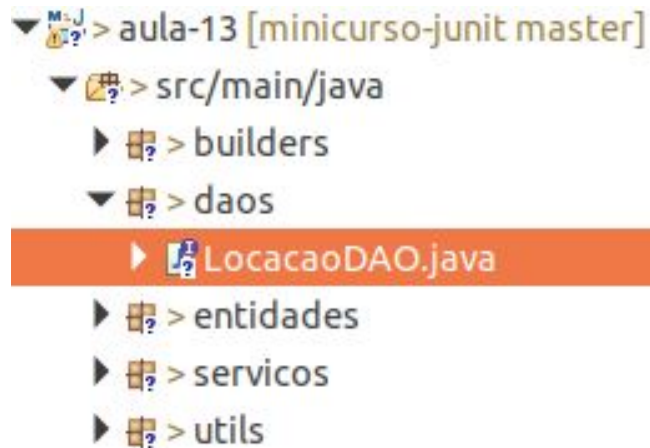
Aula-13: Trabalhando com Mocks

Importando Mockito 1.10.19 :

```
9  <dependencies>
10  <dependency>
11      <groupId>junit</groupId>
12      <artifactId>junit</artifactId>
13      <version>4.12</version>
14  </dependency>
15  <dependency>
16      <groupId>org.mockito</groupId>
17      <artifactId>mockito-all</artifactId>
18      <version>1.10.19</version>
19  </dependency>
20 </dependencies>
```

Aula-13: Trabalhando com Mocks

Exemplo: Cria-se uma interface que simula o comportamento esperado.



```
1 package daos;  
2  
3 import entidades.Locacao;  
4  
5 public interface LocacaoDAO {  
6     public void salvar(Locacao locacao);  
7 }
```

Aula-13: Trabalhando com Mocks

Exemplo: Altera classe LocacaoService

```
12 public class LocacaoService {
13
14     private LocacaoDAO dao;
15
16     public void setDao(LocacaoDAO dao) {
17         this.dao = dao;
18     }
19
20     public Locacao alugarFilme(Usuario usuario, Filme filme) throws Exception {
21         //Validação filme sem estoque
22
23         if (filme.getEstoque() == 0) {
24             throw new Exception("Filme sem estoque.");
25         }
26
27         Locacao locacao = new Locacao();
28         locacao.setFilme(filme);
29         locacao.setUsuario(usuario);
30         locacao.setDataLocacao(new Date());
31         locacao.setValor(calculaValorLocacao(filme.getPrecoLocacao()));
32
33         //Entrega no dia seguinte
34         Date dataEntrega = new Date();
35         dataEntrega = adicionarDias(dataEntrega, 1);
36         locacao.setDataRetorno(dataEntrega);
37
38         //Salvando a locação
39         dao.salvar(locacao);
40
41         return locacao;
42     }
```

Aula-13: Trabalhando com Mocks

Exemplo: Altera classe LocacaoServiceTest

```
23 public class LocacaoServiceTest {
24     LocacaoService service ;
25
26     @Before
27     public void inicializa() {
28
29         System.out.println("@Before");
30
31         service = new LocacaoService();
32         LocacaoDAO dao = Mockito.mock(LocacaoDAO.class);
33         service.setDao(dao);
34     }
35 }
```

O Mockito vai criar uma cópia da estrutura dessa classe com uma implementação vazia e com retornos padrões em todos os métodos

Aula-13: Trabalhando com Mocks

Exemplo: Altera classe LocacaoServiceTest

```
24 public class LocacaoServiceTest {
25
26     private LocacaoService service ;
27
28     @Mock
29     private LocacaoDAO dao;
30
31     @Before
32     public void inicializa() {
33
34         System.out.println("@Before");
35         MockitoAnnotations.initMocks(this);
36
37         service = new LocacaoService();
38         service.setDao(dao);
39     }
```

O Mock pode ser feito tanto com classes concretas, abstratas e interfaces.

Aula-13: Trabalhando com Mocks

Exemplo: Altera classe LocacaoServiceTest

```
25 public class LocacaoServiceTest {
26
27     @InjectMocks
28     private LocacaoService service ;
29
30     @Mock
31     private LocacaoDAO dao;
32
33     @Mock
34     private SPCService spcService;
35
36     @Mock
37     private EmailService emailService;
38
39     @Before
40     public void inicializa() {
41
42         System.out.println("@Before");
43         MockitoAnnotations.initMocks(this);
44     }
```

Aula-13: Trabalhando com Mocks

Gravando expectativas

- Nova Regra: Não deve alugar filme para caloteiros.

```
1 package servicos;  
2  
3 import entidades.Usuario;  
4  
5 public interface SPCService {  
6  
7     public boolean possuiNegativacao(Usuario usuario);  
8  
9 }
```


Aula-13: Trabalhando com Mocks

Gravando expectativas

```
12 public class LocacaoService {
13
14     private LocacaoDAO dao;
15     private SPCService spcService;
16
17     public void setDao(LocacaoDAO dao) {
18         this.dao = dao;
19     }
20
21     public void setSpcService(SPCService spc) {
22         this.spcService = spc;
23     }
24
25     public Locacao alugarFilme(Usuario usuario, Filme filme) throws Exception {
26         //Validação filme sem estoque
27
28         if (filme.getEstoque() == 0) {
29             throw new Exception("Filme sem estoque.");
30         }
31
32         if (spcService.possuiNegativacao(usuario)) {
33             throw new Exception("Usuário Negativado.");
34         }
35     }
36 }
```

Aula-13: Trabalhando com Mocks

Gravando expectativas

```
24 public class LocacaoServiceTest {
25
26     private LocacaoService service ;
27
28     @Mock
29     private LocacaoDAO dao;
30
31     @Mock
32     private SPCService spcService;
33
34     @Before
35     public void inicializa() {
36
37         System.out.println("@Before");
38         MockitoAnnotations.initMocks(this);
39
40         service = new LocacaoService();
41         service.setDao(dao);
42         service.setSpcService(spcService);
43     }
44 }
```

Aula-13: Trabalhando com Mocks

Gravando expectativas

```
175     @Test(expected=Exception.class)
176     public void naoDeveAlugarFilmeParaUsuarioNegativado() throws Exception {
177         //cenario
178         Usuario usuario = UsuarioBuilder.umUsuario().build();
179         Filme filme = FilmeBuilder.umFilme().build();
180
181         Mockito.when(spcService.possuiNegativacao(usuario)).thenReturn(true);
182
183         //acao
184         service.alugarFilme(usuario, filme);
185
186     }
```

Aula-13: Trabalhando com Mocks



Gravando expectativas:

- `Mockito.when(methodCall).thenReturn();`
- `Mockito.when(methodCall).thenCallRealMethod();`
- `Mockito.when(methodCall).thenThrow(throwableClasses);`
- `Mockito.when(methodCall).wait(timeout);`
- `Mockito.doNothing().when(mock).methodCall();`

Aula-13: Trabalhando com Mocks

Verificando comportamentos

```
1 package servicos;  
2  
3 import entidades.Usuario;  
4  
5 public interface EmailService {  
6  
7     public boolean enviaEmailUsuarioNegativado(Usuario usuario);  
8  
9 }
```

Aula-13: Trabalhando com Mocks

Verificando comportamentos

```
12 public class LocacaoService {
13
14     private LocacaoDAO dao;
15     private SPCService spcService;
16     private EmailService emailService;
17
18     public void setDao(LocacaoDAO dao) {
19         this.dao = dao;
20     }
21
22     public void setSpcService(SPCService spc) {
23         this.spcService = spc;
24     }
25
26     public void setEmailService(EmailService email) {
27         this.emailService = email;
28     }
29 }
```

Aula-13: Trabalhando com Mocks

Verificando comportamentos

```
30 public Locacao alugarFilme(Usuario usuario, Filme filme) throws Exception {
31     //Validação filme sem estoque
32
33     if (filme.getEstoque() == 0) {
34         throw new Exception("Filme sem estoque.");
35     }
36
37     if (spcService.possuiNegativacao(usuario)) {
38
39         emailService.enviaEmailUsuarioNegativado(usuario);
40
41         throw new Exception("Usuário Negativado.");
42     }
43 }
```


Aula-13: Trabalhando com Mocks

Verificando comportamentos

```
192- @Test(expected=Exception.class)
193- public void deveEnviarEmailParaUsuarioNegativado() throws Exception {
194-     //cenario
195-     Usuario usuario = UsuarioBuilder.umUsuario().build();
196-     Filme filme = FilmeBuilder.umFilme().build();
197-
198-     Mockito.when(spcService.possuiNegativacao(usuario)).thenReturn(true);
199-
200-     //acao
201-     service.alugarFilme(usuario, filme);
202-
203-     //Verificação
204-     Mockito.verify(emailService).enviaEmailUsuarioNegativado(usuario);
205-
206- }
```


Aula-13: Trabalhando com Mocks

Verificando comportamentos

```
208 @Test
209 public void naoDeveEnviarEmailParaUsuarioNaoNegativado() throws Exception {
210     //cenario
211     Usuario usuario = UsuarioBuilder.umUsuario().build();
212     Filme filme = FilmeBuilder.umFilme().build();
213
214     //acao
215     service.alugarFilme(usuario, filme);
216
217     //Verificação
218     Mockito.verify(emailService, Mockito.never()).enviaEmailUsuarioNegativado(usuario);
219
220 }
```

Aula-13: Trabalhando com Mocks

Capturando argumentos

Pode-se utilizar a anotação @Captor

```
224 @Test
225 public void deveAlugarFilmeEscolhido() throws Exception {
226
227     //cenario
228     Usuario usuario = UsuarioBuilder.umUsuario().build();
229     Filme filme = FilmeBuilder.umFilme().comNome("Se beber nao case").build();
230
231     //acao
232     service.alugarFilme(usuario, filme);
233
234     ArgumentCaptor<Locacao> argCaptLocacao = ArgumentCaptor.forClass(Locacao.class);
235     Mockito.verify(dao).salvar(argCaptLocacao.capture());
236
237     Locacao locacaoRetornada = argCaptLocacao.getValue();
238
239     assertThat(locacaoRetornada.getFilme().getNome(), CoreMatchers.is("Se beber nao case"));
240 }
```

Aula-13: Trabalhando com Mocks

Capturando argumentos

```
53 @Test
54 public void deveChecarValorParametros() {
55     //cenário
56     int a = 9;
57     int b = 3;
58
59     //acao
60     Calculadora calculdora = Mockito.mock(Calculadora.class);
61     ArgumentCaptor<Integer> argCaptInt = ArgumentCaptor.forClass(Integer.class);
62
63     Mockito.when(calculdora.somar(argCaptInt.capture(), argCaptInt.capture())).thenReturn(12)
64
65     int resultado = calculdora.somar(a, b);
66
67     assertThat(resultado, CoreMatchers.is(12));
68     assertThat(argCaptInt.getAllValues().get(0), CoreMatchers.is(a));
69     assertThat(argCaptInt.getAllValues().get(1), CoreMatchers.is(b));
70 }
71
```

Aula-13: Trabalhando com Mocks

Capturando argumentos

```
72 @Test
73 public void deveChecarValorParametrosMetodoVoid() {
74
75     //cenário
76     int a = 9;
77
78     Calculadora calculdora = Mockito.mock(Calculadora.class);
79     ArgumentCaptor<Integer> argCaptInt = ArgumentCaptor.forClass(Integer.class);
80
81     //acao
82     Mockito.doNothing().when(calculdora).imprime(argCaptInt.capture());
83
84     calculdora.imprime(a);
85
86     //verificacao
87     assertThat(argCaptInt.getValue(), CoreMatchers.is(a));
88 }
```

Aula-13: Trabalhando com Mocks

■ Usando a anotação @Spy:

- O spy é um objeto que “engole” uma instância real do tipo “espionado”, de modo que podemos utilizar esse objeto com o seu comportamento verdadeiro.
- O spy também pode ter seus métodos configurados para devolver respostas pré-fabricadas.
- Quando os parâmetros da expectativa são diferentes dos parâmetros passado na chamada do método o spy executa o método.
- O spy não funciona com interfaces apenas com classes concretas.

Aula-13: Trabalhando com Mocks

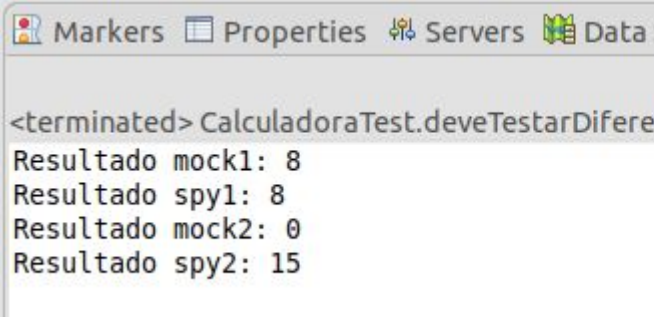
Usando o Spy

```
19 public class CalculadoraTest {
20
21     Calculadora calculadora;
22
23     @Spy
24     Calculadora calculadoraSpy;
25     @Mock
26     Calculadora calculadoraMock;
27
28     @Before
29     public void inicializa() {
30         calculadora = new Calculadora();
31         MockitoAnnotations.initMocks(this);
32     }
33 }
```


Aula-13: Trabalhando com Mocks

Usando a anotação @Spy

```
99 @Test
100 public void deveTestarDiferencaMockSpy() {
101     //cenário
102     int a = 5;
103     int b = 3;
104     int c = 10;
105
106     Mockito.when(calculadoraMock.somar(a, b)).thenReturn(8);
107     Mockito.when(calculadoraSpy.somar(a, b)).thenReturn(8);
108
109     //ação
110     int resultadoM1 = calculadoraMock.somar(a,b);
111     int resultadoS1 = calculadoraSpy.somar(a,b);
112
113     int resultadoM2 = calculadoraMock.somar(a,c);
114     int resultadoS2 = calculadoraSpy.somar(a,c);
115     //verificacao
116
117     System.out.println("Resultado mock1: " + resultadoM1);
118     System.out.println("Resultado spy1: " + resultadoS1);
119     System.out.println("Resultado mock2: " + resultadoM2);
120     System.out.println("Resultado spy2: " + resultadoS2);
121 }
```



```
<terminated> CalculadoraTest.deveTestarDifere
Resultado mock1: 8
Resultado spy1: 8
Resultado mock2: 0
Resultado spy2: 15
```

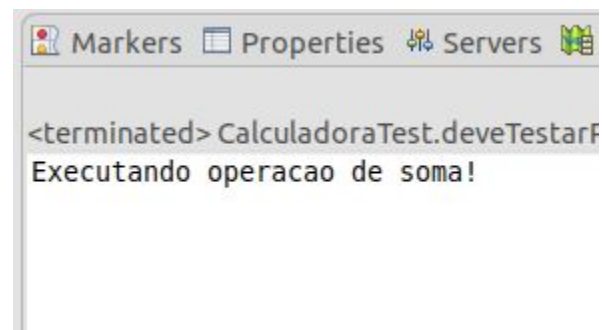
Aula-13: Trabalhando com Mocks

Problema do uso do Spy

```
3 public class Calculadora {
4
5     public int somar(int a, int b) {
6
7         System.out.println("Executando operacao de soma!");
8
9         return a + b;
10    }
124 @Test
125 public void deveTestarProblemaSpy() {
126     //cenário
127     int a = 5;
128     int b = 3;
129
130     Mockito.when(calculadoraSpy.somar(a, b)).thenReturn(8);
131
132     //ação
133     int resultadoS1 = calculadoraSpy.somar(a,b);
134
135     //verificacao
136
137     assertThat(resultadoS1, CoreMatchers.is(8));
138
139 }
```

Ao gravar a expectativa no spy ele executa o método real ao utilizar o Mockito.when() devido questões de precedência no java.

Esse problema não acontece para os mocks.



Aula-13: Trabalhando com Mocks

Problema do uso do Spy

SOLUÇÃO

Alterar a ordem de precedência informado o retorno quando executar o método soma.

```
124 @Test
125 public void deveTestarProblemaSpy() {
126     //cenário
127     int a = 5;
128     int b = 3;
129
130     //Mockito.when(calculadoraSpy.somar(a, b)).thenReturn(8);
131     Mockito.doReturn(8).when(calculadoraSpy).somar(a, b);
132
133     //ação
134     int resultadoS1 = calculadoraSpy.somar(a,b);
135
136     //verificacao
137
138     assertThat(resultadoS1, CoreMatchers.is(8));
139 }
```

Aula-13: Trabalhando com Mocks



▣ Limitações Mocks:

- ▣ Mock do construtor de um objeto
- ▣ Mock métodos estáticos
- ▣ Mock métodos privados

7.

**Trabalhando
com
PowerMock**

Trabalhando com PoweMock



- O **PowerMock** é um framework que estende outras bibliotecas “simuladas”, como o EasyMock, oferecendo recursos mais poderosos.
- O PowerMock usa um classloader personalizado e manipulação de bytecode para permitir a simulação de métodos estáticos, construtores, classes e métodos final, métodos privado e muito mais.
- Todos os usos exigem `@RunWith(PowerMockRunner.class)` e `@PrepareForTest` anotados no nível da classe.

**Vamos para o
código ...**

Aula-14: Trabalhando com PowerMock

Importando PowerMock 1.6.6 :

```
20- <dependency>
21     <groupId>org.powermock</groupId>
22     <artifactId>powermock-api-mockito</artifactId>
23     <version>1.6.6</version>
24 </dependency>
25- <dependency>
26     <groupId>org.powermock</groupId>
27     <artifactId>powermock-module-junit4</artifactId>
28     <version>1.6.6</version>
29 </dependency>
30 </dependencies>
31
```

Aula-14: Trabalhando com PowerMock

Mockando Construtores

```
39 @RunWith(PowerMockRunner.class)
40 @PrepareForTest(LocacaoService.class)
41 public class LocacaoServiceTest {
42
43     @InjectMocks
44     private LocacaoService service ;
45
46     @Mock
47     private LocacaoDAO dao;
48
49     @Mock
50     private SPCService spcService;
51
52     @Mock
53     private EmailService emailService;
54
55     @Captor
56     private ArgumentCaptor<Locacao> argCaptLocacao;
57 }
```

Aula-14: Trabalhando com PowerMock

Mockando Construtores

```
256- @Test
257- public void deveMockarConstrutorDate() throws Exception {
258-
259-     //cenario
260-     Usuario usuario = UsuarioBuilder.umUsuario().build();
261-     Filme filme = FilmeBuilder.umFilme().comNome("Se beber nao case").build();
262-
263-     SimpleDateFormat format = new SimpleDateFormat("dd/MM/yyyy");
264-     Date dataEsperada = format.parse("09/08/1996");
265-
266-     Mockito.whenNew(Date.class).withNoArguments().thenReturn(dataEsperada);
267-
268-     //acao
269-     service.alugarFilme(usuario, filme);
270-
271-     //verificacao
272-
273-     Mockito.verify(dao).salvar(argCaptLocacao.capture());
274-     assertThat(argCaptLocacao.getValue().getDataLocacao(), CoreMatchers.is(dataEsperada));
275-
276- }
```


Aula-14: Trabalhando com PowerMock

Mockando Construtores - com parâmetros

```
49 public Locacao alugarFilmeGratis(Usuario usuario, Filme filme) throws Exception {  
50     //Validação filme sem estoque  
51  
52     if (filme.getEstoque() == 0) {  
53         throw new Exception("Filme sem estoque.");  
54     }  
55  
56     Locacao locacao = new Locacao(usuario, filme);  
57     locacao.setDataLocacao(new Date());  
58     locacao.setValor(0.00);  
59  
60     //Entrega no dia seguinte  
61     Date dataEntrega = new Date();  
62     dataEntrega = adicionarDias(dataEntrega, 1);  
63     locacao.setDataRetorno(dataEntrega);  
64  
65     //Salvando a locação  
66     dao.salvar(locacao);  
67  
68     return locacao;  
69 }
```

Aula-14: Trabalhando com PowerMock

Mockando Construtores - com parâmetros

```
278- @Test
279- public void deveMockarConstrutorLocacaoComParametros() throws Exception {
280-
281-     //cenario
282-     Usuario usuario = UsuarioBuilder.umUsuario().comNome("Ezequiel").build();
283-     Filme filme = FilmeBuilder.umFilme().comNome("Se beber nao case").build();
284-
285-     Usuario usuario2 = UsuarioBuilder.umUsuario().comNome("Carlos").build();
286-     Filme filme2 = FilmeBuilder.umFilme().comNome("Quarteto Fantastico").build();
287-
288-     Locacao locacaoEsperada = new Locacao(usuario, filme);
289-
290-     PowerMockito.whenNew(Locacao.class)
291-         .withArguments(Mockito.any(Usuario.class), Mockito.any(Filme.class))
292-         .thenReturn(locacaoEsperada);
293-
294-     //acao
295-     service.alugarFilmeGratis(usuario2, filme2);
296-
297-     //verificacao
298-
299-     Mockito.verify(dao).salvar(argCaptLocacao.capture());
300-
301-     assertThat(argCaptLocacao.getValue().getUsuario().getNome(), CoreMatchers.is("Ezequiel"));
302-     assertThat(argCaptLocacao.getValue().getFilme().getNome(), CoreMatchers.is("Se beber nao case"));
303- }
```

Aula-14: Trabalhando com PowerMock

Mockando Métodos Estáticos

```
81- public static void aplica10DescontoValorLocacao(Locacao locacao) {
82-     Double desconto = locacao.getValor() * 0.1;
83-     Double novoValor = locacao.getValor() - desconto;
84-     locacao.setValor(novoValor);
85- }
--

306- @Test
307- public void deveMockarMetodoEstaticoVoid() {
308-     //cenario
309-
310-     Locacao locacao = LocacaoBuilder.umaLocacao().comValor(500.00).build();
311-
312-     PowerMockito.mockStatic(LocacaoService.class);
313-     PowerMockito.doNothing().when(LocacaoService.class);
314-
315-     //acao
316-     LocacaoService.aplica10DescontoValorLocacao(locacao);
317-
318-     //Verificacao
319-
320-     assertThat(locacao.getValor(), CoreMatchers.is(500.00));
321-
322- }
```

Aula-14: Trabalhando com PowerMock

Mockando Métodos Estáticos

```
87 public static Double get10DescontoValorLocacao(Locacao locacao) {
88     Double desconto = locacao.getValor() * 0.1;
89     Double novoValor = locacao.getValor() - desconto;
90     return novoValor;
91 }

324 @Test
325 public void deveMockarMetodoEstaticoComRetorno() {
326
327     //cenario
328     Locacao locacao = LocacaoBuilder.umaLocacao().comValor(500.00).build();
329
330     Mockito.mockStatic(LocacaoService.class);
331     Mockito.when(LocacaoService.get10DescontoValorLocacao(Mockito.any(Locacao.class)))
332         .thenReturn(600.00);
333
334     //acao
335     Double novoValorLocacao = LocacaoService.get10DescontoValorLocacao(locacao);
336
337     //Verificacao
338
339     assertThat(novoValorLocacao, CoreMatchers.is(600.00));
340 }
```

Aula-14: Trabalhando com PowerMock

- Mockando Métodos Estáticos
 - Verificando comportamento

```
324 @Test
325 public void deveMockarMetodoEstaticoComRetorno() {
326
327     //cenario
328     Locacao locacao = LocacaoBuilder.umaLocacao().comValor(500.00).build();
329
330     PowerMockito.mockStatic(LocacaoService.class);
331     PowerMockito.when(LocacaoService.get10DescontoValorLocacao(Mockito.any(Locacao.class)))
332         .thenReturn(600.00);
333
334     //acao
335     Double novoValorLocacao = LocacaoService.get10DescontoValorLocacao(locacao);
336
337     //Verificacao
338
339     assertThat(novoValorLocacao, CoreMatchers.is(600.00));
340
341     PowerMockito.verifyStatic();
342     LocacaoService.get10DescontoValorLocacao(locacao);
343 }
```


Aula-14: Trabalhando com PowerMock

Mockando Métodos Privados

```
347 @Test
348 public void deveMockarMetodoPrivado() throws Exception {
349
350     //cenario
351     Usuario usuario = UsuarioBuilder.umUsuario().build();
352     Filme filme = FilmeBuilder.umFilme().comPrecoLocacao(100.00).build();
353     service = PowerMockito.spy(service);
354
355     PowerMockito.doReturn(400.00).when(service, "calculaValorLocacao", Mockito.any(Double.class));
356
357     //acao
358     Locacao locacao = service.alugarFilme(usuario, filme);
359
360     //verificacao
361     assertThat(locacao.getValor(), CoreMatchers.is(400.0));
362 }
```

Aula-14: Trabalhando com PowerMock

- Mockando Métodos Privados
 - Verificando comportamento

```
347 @Test
348 public void deveMockarMetodoPrivado() throws Exception {
349
350     //cenario
351     Usuario usuario = UsuarioBuilder.umUsuario().build();
352     Filme filme = FilmeBuilder.umFilme().comPrecoLocacao(100.00).build();
353     service = PowerMockito.spy(service);
354
355     PowerMockito.doReturn(400.00).when(service, "calculaValorLocacao", Mockito.any(Double.class));
356
357     //acao
358     Locacao locacao = service.alugarFilme(usuario, filme);
359
360     //verificacao
361     assertThat(locacao.getValor(), CoreMatchers.is(400.0));
362
363     PowerMockito.verifyPrivate(service).invoke("calculaValorLocacao", Mockito.any(Double.class));
364 }
365
```

Aula-14: Trabalhando com PowerMock

■ Testando Métodos Privados

Utiliza biblioteca **Whitebox** do Powermock

```
366 @Test
367 public void deveTestarMetodoPrivado() throws Exception {
368
369     //cenario
370     Double valorLocacao = 100.00;
371
372     Double valorRetornado = org.powermock.reflect.Whitebox
373         .invokeMethod(service, "calculaValorLocacao", valorLocacao);
374     //acao
375
376     assertThat(valorRetornado, CoreMatchers.is(75.0));
377 }
```


Aula-14: Trabalhando com PowerMock

■ Testando Construtores

Utiliza biblioteca **Whitebox** do Powermock

```
379 @Test
380 public void deveTestarConstrutor() throws Exception {
381
382     //cenario
383     Usuario usuario = UsuarioBuilder.umUsuario().comNome("Franciele").build();
384     Filme filme = FilmeBuilder.umFilme().comNome("Minha mae eh uma peca").build();
385
386     Locacao locacao = org.powermock.reflect.Whitebox
387         .invokeConstructor(Locacao.class, usuario, filme);
388
389     //acao
390     assertThat(locacao.getUsuario().getNome(), CoreMatchers.is("Franciele"));
391     assertThat(locacao.getFilme().getNome(), CoreMatchers.is("Minha mae eh uma peca"));
392 }
```

Aula-14: Trabalhando com PowerMock



- **Pontos negativos do PowerMock:**
 - Diminui cobertura dos testes
 - Aumenta o tempo de execução dos testes

8.

Conclusão

Conclusão



- Neste minicurso foi apresentado o uso de testes unitários na linguagem Java utilizando Junit.
- Foi apresentado as vantagens do uso do framework bem como o uso do mesmo.
- Foi também apresentada a técnica de desenvolvimento TDD e o padrão Data Builder.
- Por fim foi apresentado o uso de Mocks para simular objetos reais para teste.
- Com isso foi possível perceber as vantagens e abrangência dos testes unitários para melhorar a qualidade do software.

Fim ...

Obrigada!

Alguma dúvida?



/franciferreiraap



/franciele-ferreira



/francieleap



/franciferreiraap