

19/5/2025

TP2

**Desenvolvimento de Serviços Web e Testes
com Java**

Professor(a): Bernardo Petry Prates

LINK GITHUB

<https://github.com/faculdade-infnet/IV-2-Java/tree/main/TP2>

1. EXERCÍCIO

- Implementação

```
© CalculadoraReembolso.java x
1 public class CalculadoraReembolso
  { 6 usages Samuel Hermany *
2     public double calcular(double valorConsulta, double
    percentualCobertura) { 4 usages Samuel Hermany *
3         return valorConsulta * (percentualCobertura / 100);
4     }
```

- Teste

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

public class CalculadoraReembolsoTest { new *
    private CalculadoraReembolso service; 5 usages
    private Paciente dummyPaciente; 1 usage

    // Setup
    @BeforeEach new *
    public void setUp() {
        service = new CalculadoraReembolso();
        dummyPaciente = new Paciente();
    }

    @Test new *
    @DisplayName("Exercício 01")
    public void testDeveRetornarValorOriginalComPercentualDeDesconto() {
        double actual = service.calcular( valorConsulta: 200, percentualCobertura: 70);

        assertEquals( expected: 140.0, actual, delta: 0.01);
    }
}
```

2. EXERCÍCIO

```
@Test new *
@DisplayName("Exercício 02 - Consulta com valor 0")
public void testDeveRetornarZeroQuandoConsultaZero() {
    double actual = service.calcular(valorConsulta: 0, percentualCobertura: 70);

    assertEquals(expected: 0.0, actual, delta: 0.01);
}

@Test new *
@DisplayName("Exercício 02 - Cobertura com valor 0")
public void testDeveRetornarZeroQuandoCoberturaZero() {
    double actual = service.calcular(valorConsulta: 200, percentualCobertura: 0);

    assertEquals(expected: 0.0, actual, delta: 0.01);
}

@Test new *
@DisplayName("Exercício 02 - Cobertura com valor 100")
public void testDeveRetornarValorInteiroQuandoCoberturaCem() {
    double actual = service.calcular(valorConsulta: 200, percentualCobertura: 100);

    assertEquals(expected: 200.0, actual, delta: 0.01);
}
```

3. EXERCÍCIO

- Implementação

```
© CalculadoraReembolso.java x
1 public class CalculadoraReembolso
2 { 6 usages Samuel Hermany *
3     public double calcular(double valorConsulta, double
4         percentualCobertura) { 4 usages Samuel Hermany *
5         return valorConsulta * (percentualCobertura / 100);
6     }
7 }
```

- Teste

```
CalculadoraReembolsoTest.java x
1  import org.junit.jupiter.api.*;
2
3  import static org.junit.jupiter.api.Assertions.*;
4
5  public class CalculadoraReembolsoTest { new *
6      private CalculadoraReembolso service; 5 usages
7      private Paciente dummyPaciente; 1 usage
8
9
10     // Setup
11     @BeforeEach new *
12     public void setUp() {
13         service = new CalculadoraReembolso();
14         dummyPaciente = new Paciente();
15     }
16
17     @Test new *
18     @DisplayName("Exercício 01")
19     public void testDeveRetornarValorOriginalComPercentualDeDesconto() {
20         double actual = service.calcular( valorConsulta: 200, percentualCobertura: 70);
21
22         assertEquals( expected: 140.0, actual, delta: 0.01);
23     }
24
25     @Test new *
26     @DisplayName("Exercício 02 - Consulta com valor 0")
27     public void testDeveRetornarZeroQuandoConsultaZero() {
28         double actual = service.calcular( valorConsulta: 0, percentualCobertura: 70);
29
30         assertEquals( expected: 0.0, actual, delta: 0.01);
31     }
32
33     @Test new *
34     @DisplayName("Exercício 02 - Cobertura com valor 0")
35     public void testDeveRetornarZeroQuandoCoberturaZero() {
36         double actual = service.calcular( valorConsulta: 200, percentualCobertura: 0);
37
38         assertEquals( expected: 0.0, actual, delta: 0.01);
39     }
40
41     @Test new *
42     @DisplayName("Exercício 02 - Cobertura com valor 100")
43     public void testDeveRetornarValorInteiroQuandoCoberturaCem() {
44         double actual = service.calcular( valorConsulta: 200,
45             percentualCobertura: 100);
46
47         assertEquals( expected: 200.0, actual, delta: 0.01);
48     }
49 }
```

4. EXERCÍCIO

- Implementação

```
CalculadoraReembolso.java x
1 public class CalculadoraReembolso {
5
6     public double calcular(double valorConsulta, double percentualCobertura,
7         Paciente paciente) {
8         return valorConsulta * (percentualCobertura / 100);
9     }
9
```

- Teste

```
CalculadoraReembolsoTest.java x
5 >> public class CalculadoraReembolsoTest { @ Samuel Hermany *
6     private CalculadoraReembolso service; 5 usages
7     private Paciente dummyPaciente; 5 usages
8
9     // Setup
10    @BeforeEach @ Samuel Hermany
11    public void setUp() {
12        service = new CalculadoraReembolso();
13        dummyPaciente = new Paciente();
14    }
15
16    @Test @ Samuel Hermany
17    @DisplayName("Exercício 01")
18    > public void testDeveRetornarValorOriginalComPercentualDeDesconto() {
19        double actual = service.calcular(valorConsulta: 200, percentualCobertura: 70, dummyPaciente);
20
21        assertEquals(expected: 140.0, actual, delta: 0.01);
22    }
23
24    @Test @ Samuel Hermany
25    @DisplayName("Exercício 02 - Consulta com valor 0")
26    > public void testDeveRetornarZeroQuandoConsultaZero() {
27        double actual = service.calcular(valorConsulta: 0, percentualCobertura: 70, dummyPaciente);
28
29        assertEquals(expected: 0.0, actual, delta: 0.01);
30    }
31
32    @Test @ Samuel Hermany
33    @DisplayName("Exercício 02 - Cobertura com valor 0")
34    > public void testDeveRetornarZeroQuandoCoberturaZero() {
35        double actual = service.calcular(valorConsulta: 200, percentualCobertura: 0, dummyPaciente);
36
37        assertEquals(expected: 0.0, actual, delta: 0.01);
38    }
39
```

```
40    @Test @ Samuel Hermany
41    @DisplayName("Exercício 02 - Cobertura com valor 100")
42    > public void testDeveRetornarValorInteiroQuandoCoberturaCem() {
43        double actual = service.calcular(valorConsulta: 200, percentualCobertura: 100, dummyPaciente);
44
45        assertEquals(expected: 200.0, actual, delta: 0.01);
46    }
47 }
```

5. EXERCÍCIO

```
HistoricoConsultas.java x
1 import java.util.List;
2
3 public interface HistoricoConsultas { 1 usage
4     void adicionarConsulta(Consulta consulta);
5
6     List<Consulta> listarConsultas(); no usages
7 }
```

```
FakeHistoricoConsultas.java x
1 import java.util.*;
2
3 public class FakeHistoricoConsultas implements HistoricoConsultas { no u
4     private final List<Consulta> historicoConsultas = new ArrayList<>();
5
6     @Override no usages new *
7     public void adicionarConsulta(Consulta consulta) {
8         historicoConsultas.add(consulta);
9     }
10
11     @Override no usages Samuel Hermany
12     public List<Consulta> listarConsultas() {
13         return new ArrayList<>(historicoConsultas);
14     }
15 }
```

6. EXERCÍCIO

- Interface

```
PlanoSaude.java x
1 public interface PlanoSaude { 1 usage 1 implementation Samuel Hermany
2     double getPercentualCobertura(double percentualCobertura);
3 }
```

- Implementação

```
PlanoSaudeBasico.java x
1 public class PlanoSaudeBasico implements PlanoSaude { 12 usages new *
2     @Override 2 usages new *
3     public double getPercentualCobertura(double percentualCobertura) {
4         return percentualCobertura / 100;
5     }
6 }
```

- Teste

```
StubPlanoSaudeTest.java x
1 import org.junit.jupiter.api.*;
2
3 import static org.junit.jupiter.api.Assertions.assertEquals;
4
5 public class StubPlanoSaudeTest { 2 Samuel Hermany *
6     private CalculadoraReembolso service; 3 usages
7     private PlanoSaudeBasico planoStub; 3 usages
8
9     // Setup
10    @BeforeEach 2 Samuel Hermany *
11    public void setUp() {
12        service = new CalculadoraReembolso();
13        planoStub = new PlanoSaudeBasico();
14    }
15
16    @Test 2 Samuel Hermany *
17    @DisplayName("Exercício 06 - Stub - Plano com 50% de cobertura")
18    void testPlano50Porcento() {
19        double resultado = service.calcular(valorConsulta: 200, planoStub);
20        assertEquals(expected: 140.0, resultado, delta: 0.01);
21    }
22
23    @Test 2 Samuel Hermany *
24    @DisplayName("Exercício 06 - Stub - Plano com 80% de cobertura")
25    void testPlanoComOitentaPorcento() {
26        double resultado = service.calcular(valorConsulta: 200, planoStub);
27        assertEquals(expected: 140.0, resultado, delta: 0.01);
28    }
29 }
```

7. EXERCÍCIO

- Interface

```
Auditoria.java x
1 public interface Auditoria { 3 usages 1 implementation new *
2     void registrarConsulta(Paciente paciente, double valor);
3 }
```

- Implementação AuditoriaSpy

```
AuditoriaSpy.java x
1 public class AuditoriaSpy implements Auditoria { 2 usages new *
2     private boolean foiChamado = false; 2 usages
3
4     @Override 1 usage new *
5     public void registrarConsulta(Paciente paciente, double valor) {
6         foiChamado = true;
7     }
8
9     public boolean foiChamado() { no usages new *
10        return foiChamado;
11    }
12 }
```

- Ajuste no método CalculadoraReembolso

```
CalculadoraReembolso.java x
1 public class CalculadoraReembolso { 8 usages Samuel Hermany *
10     private Auditoria auditoria; 3 usages
11     private AutorizadorReembolso autorizador; 1 usage
12
13     public CalculadoraReembolso() { 4 usages Samuel Hermany
14     }
15
16     public CalculadoraReembolso(Auditoria auditoria) { 3 usages Samuel Hermany
17         this.auditoria = auditoria;
18     }
19
20     public CalculadoraReembolso(Auditoria auditoria, AutorizadorReembolso autorizador) {
21         this.auditoria = auditoria;
22         this.autorizador = autorizador;
23     }
24
25     @ public double calcular(double valorConsulta, PlanoSaude plano) { 2 usages Samuel Herma
26         return valorConsulta * (plano.getPercentualCobertura() / 100);
27     }
28
29     @ public double calcular(Consulta consulta, Paciente paciente, PlanoSaude plano) { 1 usa
30         double reembolso = consulta.getValor() * (plano.getPercentualCobertura() / 100);
31
32         // Define o valor de auditoria como TRUE, garantindo que ele foi chamado
33         auditoria.registrarConsulta(consulta);
34         return reembolso;
35     }
36 }
```


- Teste

```
AuditoriaSpyTest.java x
1  > import ...
4
5  public class AuditoriaSpyTest {  Samuel Hermany *
6      @Test  Samuel Hermany *
7      @DisplayName("Exercício 07")
8      public void testDeveChamarAuditoriaAoRegistrarConsulta() {
9          AuditoriaSpy auditoriaSpy = new AuditoriaSpy();
10         CalculadoraReembolso service = new CalculadoraReembolso(auditoriaSpy);
11         Consulta consulta = new Consulta( valor: 200.0);
12         Paciente paciente = new Paciente();
13         PlanoSaudeCobertura50PorCento plano = new PlanoSaudeCobertura50PorCento();
14
15         service.calcular(consulta, paciente, plano);
16
17         assertTrue(auditoriaSpy.foiChamado(), message: "Esperado que o método
18             registrarConsulta tenha sido chamado.");
19     }
}
```

8. EXERCÍCIO

- Interface

```
AutorizadorReembolso.java x
1  public interface AutorizadorReembolso { 4 usages new *
2      boolean autorizar(Consulta consulta, Paciente paciente);
3  }
```

- Método

```
CalculadoraReembolso.java x
1  public class CalculadoraReembolso { 12 usages Samuel Hermany *
22
23      private Auditoria auditoria; 3 usages
24      private AutorizadorReembolso autorizador; 2 usages
25
26      public CalculadoraReembolso(Auditoria auditoria, AutorizadorReembolso
27          autorizador) { 4 usages new *
28          this.auditoria = auditoria;
29          this.autorizador = autorizador;
30      }
31
32      public double calcular(Consulta consulta, Paciente paciente, PlanoSaudeBasico
33          plano) { 3 usages Samuel Hermany *
34          if (!autorizador.autorizar(consulta, paciente)) {
35              throw new RuntimeException("Reembolso não autorizado");
36          }
37
38          double reembolso = consulta.getValor() * (plano.getPercentualCobertura(70)
39              / 100);
40
41          // Define o valor de auditoria como TRUE, garantindo que ele foi chamado
42          auditoria.registrarConsulta(consulta);
43          return reembolso;
44      }
45  }
```

- Teste

```
CalculadoraReembolsoMockitoTest.java x
6 public class CalculadoraReembolsoMockitoTest { new *
7     @BeforeEach new *
8     public void setUp() {
9         auditoriaSpy = new AuditoriaSpy();
10        autorizadorMock = mock(AutorizadorReembolso.class);
11        consulta = new Consulta( valor: 200.0);
12        paciente = new Paciente();
13        plano = new PlanoSaudeBasico();
14    }
15
16    @Test new *
17    @DisplayName("Exercício 08 - Nao autorizado")
18    public void testDeveLancarExcecaoQuandoNaoAutorizado() {
19        // Arrange
20        Auditoria auditoriaMock = mock(Auditoria.class);
21        HistoricoConsultas historicoMock = mock(HistoricoConsultas.class);
22        AutorizadorReembolso autorizadorMock = mock(AutorizadorReembolso.class);
23
24        when(autorizadorMock.autorizar(any(Consulta.class), any(Paciente.class))).thenReturn(false);
25        CalculadoraReembolso service = new CalculadoraReembolso(auditoriaMock, autorizadorMock);
26
27        double reembolso = service.calcular(consulta, paciente, plano);
28        // O teste passa se a exceção ReembolsoNaoAutorizadoException for lançada
29    }
30
31    @Test new *
32    @DisplayName("Exercício 08 - Autorizado")
33    public void testDeveCalcularQuandoAutorizado() {
34        // Arrange
35        Auditoria auditoriaMock = mock(Auditoria.class);
36        AutorizadorReembolso autorizadorMock = mock(AutorizadorReembolso.class);
37
38        when(autorizadorMock.autorizar(any(Consulta.class), any(Paciente.class))).thenReturn(true);
39        CalculadoraReembolso service = new CalculadoraReembolso(auditoriaMock, autorizadorMock);
40
41        double reembolso = service.calcular(consulta, paciente, plano);
42
43        assertEquals( expected: 140.0, reembolso, delta: 0.001);
44        verify(autorizadorMock).autorizar(consulta, paciente);
45    }
46 }
```

9. EXERCÍCIO

- Helpers

```
ConsultaTestHelper.java x
1 package helpers;
2
3 import org.Consulta;
4
5 public class ConsultaTestHelper { 4 usages new *
6     @ public static Consulta criarConsultaPadrao() { 2 usages
7         return new Consulta( valor: 200.0);
8     }
9
10    @ public static Consulta criarConsulta(double valor) {
11        return new Consulta(valor);
12    }
13 }
```

- Teste

```
CalcularReembolsoComHelpersTest.java x
1  > import ...
7
8  public class CalcularReembolsoComHelpersTest { new *
9      private CalculadoraReembolso service; 4 usages
10     private Auditoria auditoriaMock; 2 usages
11     private AutorizadorReembolso autorizadorMock; 3 usages
12     private Paciente dummyPaciente; 4 usages
13
14     @BeforeEach new *
15     public void setup() {
16         auditoriaMock = mock(Auditoria.class);
17         autorizadorMock = mock(AutorizadorReembolso.class);
18         when(autorizadorMock.autorizar(any(), any())).thenReturn(true);
19
20         service = new CalculadoraReembolso(auditoriaMock, autorizadorMock);
21         dummyPaciente = new Paciente();
22     }
23
24     @Test new *
25     @DisplayName("Exercício 9 - Cálculo de reembolso básico")
26     public void deveCalcularReembolsoCorretamente() {
27         // Arrange
28         Consulta consulta = ConsultaTestHelper.criarConsulta(valor: 0);
29         PlanoSaudeStub plano = new PlanoSaudeStub(percentualCobertura: 70);
30
31         double reembolso = service.calcular(consulta, dummyPaciente, plano);
32
33         assertEquals(expected: 0.0, reembolso, delta: 0.001);
34     }
35
36     @Test new *
37     @DisplayName("Exercício 9 - Cobertura com valor 0")
38     public void testDeveRetornarZeroQuandoCoberturaZero() {
39         // Consulta com valor 200
40         Consulta consulta = ConsultaTestHelper.criarConsultaPadrao();
41         PlanoSaudeStub plano = new PlanoSaudeStub(percentualCobertura: 0);
42
43         double reembolso = service.calcular(consulta, dummyPaciente, plano);
44
45         assertEquals(expected: 0.0, reembolso, delta: 0.01);
46     }
47
48     @Test new *
49     @DisplayName("Exercício 9 - Cobertura com valor 100")
50     public void testDeveRetornarValorInteiroQuandoCoberturaCem() {
51         // Arrange
52         Consulta consulta = ConsultaTestHelper.criarConsultaPadrao(); // Con
53         PlanoSaudeStub plano = new PlanoSaudeStub(percentualCobertura: 100); // F
54
55         double reembolso = service.calcular(consulta, dummyPaciente, plano);
56
57         assertEquals(expected: 200.0, reembolso, delta: 0.01);
58     }
59 }
```

10. EXERCÍCIO

11. EXERCÍCIO

12. EXERCÍCIO