Engenharia de Software

19/5/2025

TP2

Desenvolvimento de Serviços Web e Testes com Java

Professor(a): Bernardo Petry Prates

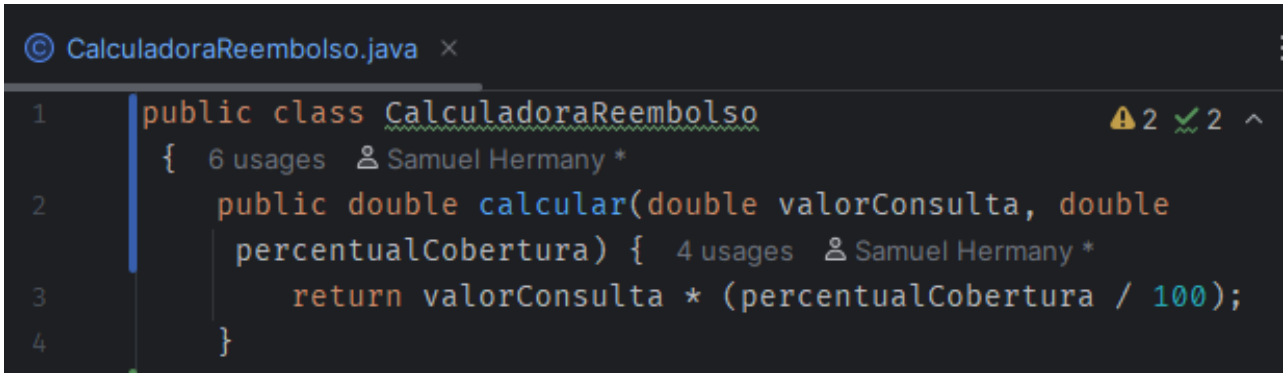
Samuel Hermany
INSTITUTO INFNET

LINK GITHUB

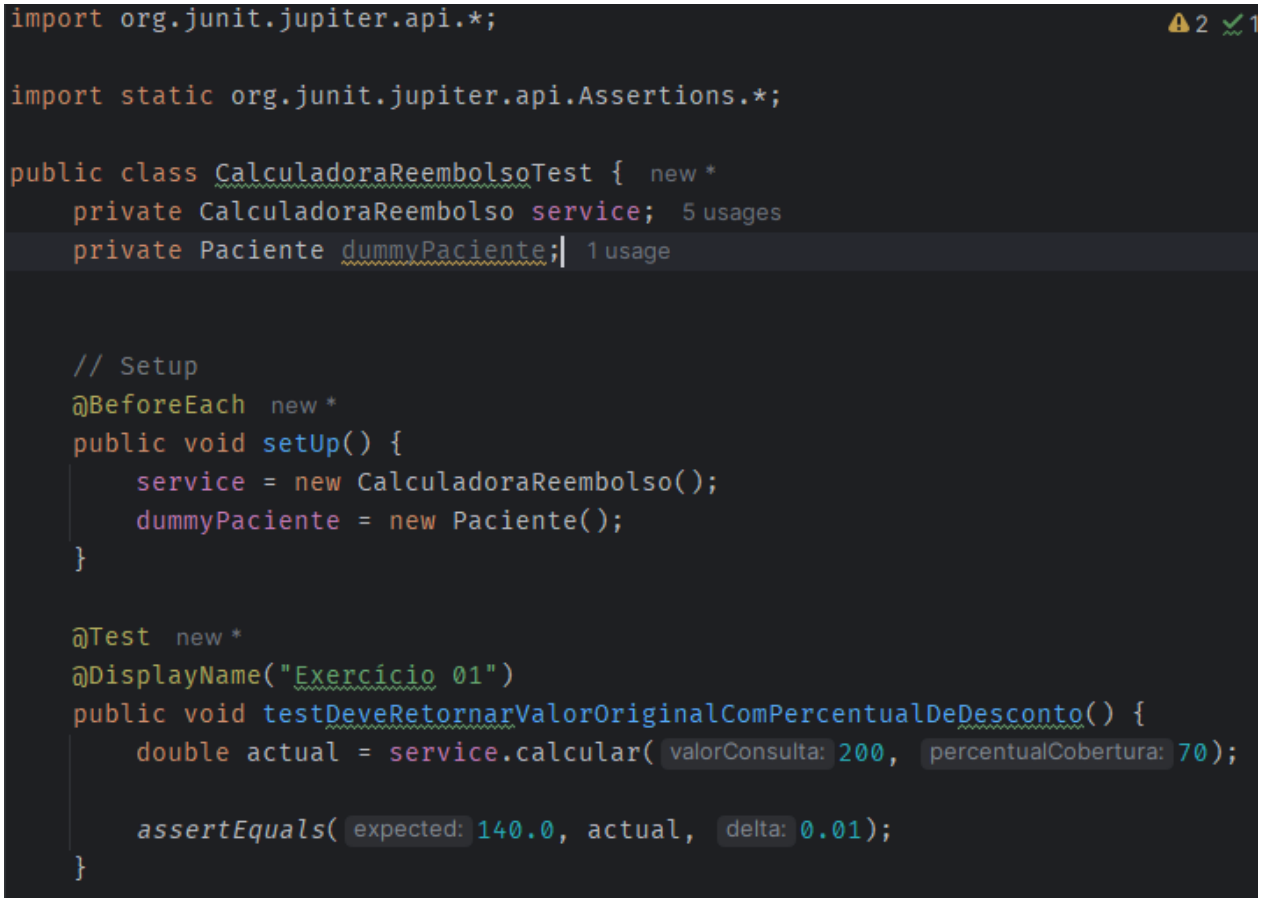
https://github.com/faculdade-infnet/IV-2-Java/tree/main/TP2

1. EXERCÍCIO

- Implementação



- Teste



2. EXERCÍCIO

```java
@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

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

- Teste

CalculadoraReembolsoTest.java

```java
import org.junit.jupiter.api.*;

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

public class CalculadoraReembolsoTest {
    private CalculadoraReembolso service;
    private Paciente dummyPaciente;


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

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

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

    @Test
    @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
    @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
    @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);
    }
}
```

4. EXERCÍCIO

- Implementação

CalculadoraReembolso.java

```java
public class CalculadoraReembolso {

    public double calcular(double valorConsulta, double percentualCobertura,
     Paciente paciente) {
        return valorConsulta * (percentualCobertura / 100);
    }

}
```

- Teste

CalculadoraReembolsoTest.java

```java
public class CalculadoraReembolsoTest {  Samuel Hermany *
    private CalculadoraReembolso service;  5 usages
    private Paciente dummyPaciente;  5 usages

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

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

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

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

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

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

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

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

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

5. EXERCÍCIO

HistoricoConsultas.java

```java
import java.util.List;

public interface HistoricoConsultas {  1 usage
    Change signature
    void adicionarConsulta(Consulta consulta);

    List<Consulta> listarConsultas();  no usages
}
```

FakeHistoricoConsultas.java

```java
import java.util.*;

public class FakeHistoricoConsultas implements HistoricoConsultas {  no u
    private final List<Consulta> historicoConsultas = new ArrayList<>();

    @Override  no usages  new *
    public void adicionarConsulta(Consulta consulta) {
        historicoConsultas.add(consulta);
    }

    @Override  no usages  Samuel Hermany
    public List<Consulta> listarConsultas() {
        return new ArrayList<>(historicoConsultas);
    }
}
```

6. EXERCÍCIO

- Interface

PlanoSaude.java

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

- Implementação

```java
public class PlanoSaudeBasico implements PlanoSaude {  12 usages  new *
    @Override  2 usages  new *
    public double getPercentualCobertura(double percentualCobertura) {
        return percentualCobertura / 100;
    }
}
```

- Teste

```java
import org.junit.jupiter.api.*;

import static org.junit.jupiter.api.Assertions.assertEquals;

public class StubPlanoSaudeTest {  Samuel Hermany *
    private CalculadoraReembolso service;  3 usages
    private PlanoSaudeBasico planoStub;  3 usages

    // Setup
    @BeforeEach  Samuel Hermany *
    public void setUp() {
        service = new CalculadoraReembolso();
        planoStub = new PlanoSaudeBasico();
    }

    @Test  Samuel Hermany *
    @DisplayName("Exercício 06 - Stub - Plano com 50% de cobertura")
    void testPlano50Porcento() {
        double resultado = service.calcular( valorConsulta: 200, planoStub);
        assertEquals( expected: 140.0, resultado,  delta: 0.01);
    }

    @Test  Samuel Hermany *
    @DisplayName("Exercício 06 - Stub - Plano com 80% de cobertura")
    void testPlanoComOitentaPorcento() {
        double resultado = service.calcular( valorConsulta: 200, planoStub);
        assertEquals( expected: 140.0, resultado,  delta: 0.01);
    }
}
```

7. Exercício

- Interface

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

- Implementação AuditoriaSpy

AuditoriaSpy.java

```java
public class AuditoriaSpy implements Auditoria {  2 usages  new *
    private boolean foiChamado = false;  2 usages

    @Override  1 usage  new *
    public void registrarConsulta(Paciente paciente, double valor) {
        foiChamado = true;
    }

    public boolean foiChamado() {  no usages  new *
        return foiChamado;
    }
}
```

- Ajuste no método CalculadoraReembolso

CalculadoraReembolso.java

```java
public class CalculadoraReembolso {  8 usages  Samuel Hermany *
    private Auditoria auditoria;  3 usages
    private AutorizadorReembolso autorizador;  1 usage

    public CalculadoraReembolso() {  4 usages  Samuel Hermany
    }

    public CalculadoraReembolso(Auditoria auditoria) {  3 usages  Samuel Hermany
        this.auditoria = auditoria;
    }

    public CalculadoraReembolso(Auditoria auditoria, AutorizadorReembolso autorizador) {
        this.auditoria = auditoria;
        this.autorizador = autorizador;
    }

    public double calcular(double valorConsulta, PlanoSaude plano) {  2 usages  Samuel Herma
        return valorConsulta * (plano.getPercentualCobertura() / 100);
    }

    public double calcular(Consulta consulta, Paciente paciente, PlanoSaude plano) {  1 usa
        double reembolso = consulta.getValor() * (plano.getPercentualCobertura() / 100);

        // Define o valor de auditoria como TRUE, garantindo que ele foi chamado
        auditoria.registrarConsulta(consulta);
        return reembolso;
    }
}
```

- Teste

```java
import ...

public class AuditoriaSpyTest {  Samuel Hermany *
    @Test  Samuel Hermany *
    @DisplayName("Exercício 07")
    public void testDeveChamarAuditoriaAoRegistrarConsulta() {
        AuditoriaSpy auditoriaSpy = new AuditoriaSpy();
        CalculadoraReembolso service = new CalculadoraReembolso(auditoriaSpy);
        Consulta consulta = new Consulta( valor: 200.0);
        Paciente paciente = new Paciente();
        PlanoSaudeCobertura50PorCento plano = new PlanoSaudeCobertura50PorCento();

        service.calcular(consulta, paciente, plano);

        assertTrue(auditoriaSpy.foiChamado(), message: "Esperado que o método
         registrarConsulta tenha sido chamado.");
    }
}
```

8. EXERCÍCIO

- Interface

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

- Método

```java
public class CalculadoraReembolso {  12 usages  Samuel Hermany *
    private Auditoria auditoria;  3 usages
    private AutorizadorReembolso autorizador;  2 usages

    public CalculadoraReembolso(Auditoria auditoria, AutorizadorReembolso
     autorizador) {  4 usages  new *
        this.auditoria = auditoria;
        this.autorizador = autorizador;
    }

    public double calcular(Consulta consulta, Paciente paciente, PlanoSaudeBasico
     plano) {  3 usages  Samuel Hermany *
        if (!autorizador.autorizar(consulta, paciente)) {
            throw new RuntimeException("Reembolso não autorizado");
        }

        double reembolso = consulta.getValor() * (plano.getPercentualCobertura(70)
          / 100);

        // Define o valor de auditoria como TRUE, garantindo que ele foi chamado
        auditoria.registrarConsulta(consulta);
        return reembolso;
    }
}
```

- Teste

```java
public class CalculadoraReembolsoMockitoTest {
    @BeforeEach
    public void setUp() {
        auditoriaSpy = new AuditoriaSpy();
        autorizadorMock = mock(AutorizadorReembolso.class);
        consulta = new Consulta( valor: 200.0);
        paciente = new Paciente();
        plano = new PlanoSaudeBasico();
    }


    @Test
    @DisplayName("Exercício 08 - Nao autorizado")
    public void testDeveLancarExcecaoQuandoNaoAutorizado() {
        // Arrange
        Auditoria auditoriaMock = mock(Auditoria.class);
        HistoricoConsultas historicoMock = mock(HistoricoConsultas.class);
        AutorizadorReembolso autorizadorMock = mock(AutorizadorReembolso.class);

        when(autorizadorMock.autorizar(any(Consulta.class), any(Paciente.class))).thenReturn( t: false);
        CalculadoraReembolso service = new CalculadoraReembolso(auditoriaMock, autorizadorMock);

        double reembolso = service.calcular(consulta, paciente, plano);
        // O teste passa se a exceção ReembolsoNaoAutorizadoException for lançada
    }

    @Test
    @DisplayName("Exercício 08 - Autorizado")
    public void testDeveCalcularQuandoAutorizado() {
        // Arrange
        Auditoria auditoriaMock = mock(Auditoria.class);
        AutorizadorReembolso autorizadorMock = mock(AutorizadorReembolso.class);

        when(autorizadorMock.autorizar(any(Consulta.class), any(Paciente.class))).thenReturn( t: true);
        CalculadoraReembolso service = new CalculadoraReembolso(auditoriaMock, autorizadorMock);

        double reembolso = service.calcular(consulta, paciente, plano);

        assertEquals( expected: 140.0, reembolso,  delta: 0.001);
        verify(autorizadorMock).autorizar(consulta, paciente);
    }
}
```

9. Exercício

- Helpers

```java
package helpers;

import org.Consulta;

public class ConsultaTestHelper {
    public static Consulta criarConsultaPadrao() {
        return new Consulta( valor: 200.0);
    }

    public static Consulta criarConsulta(double valor) {
        return new Consulta(valor);
    }
}
```

- Teste

CalcularReembolsoComHelpersTest.java

```java
import ...

public class CalcularReembolsoComHelpersTest {  new *
    private CalculadoraReembolso service;  4 usages
    private Auditoria auditoriaMock;  2 usages
    private AutorizadorReembolso autorizadorMock;  3 usages
    private Paciente dummyPaciente;  4 usages

    @BeforeEach  new *
    public void setup() {
        auditoriaMock = mock(Auditoria.class);
        autorizadorMock = mock(AutorizadorReembolso.class);
        when(autorizadorMock.autorizar(any(), any())).thenReturn( t: true);

        service = new CalculadoraReembolso(auditoriaMock, autorizadorMock);
        dummyPaciente = new Paciente();
    }

    @Test  new *
    @DisplayName("Exercício 9 - Cálculo de reembolso básico")
    public void deveCalcularReembolsoCorretamente() {
        // Arrange
        Consulta consulta = ConsultaTestHelper.criarConsulta( valor: 0);
        PlanoSaudeStub plano = new PlanoSaudeStub( percentualCobertura: 70);

        double reembolso = service.calcular(consulta, dummyPaciente, plano);

        assertEquals( expected: 0.0, reembolso,  delta: 0.001);
    }

    @Test  new *
    @DisplayName("Exercício 9 - Cobertura com valor 0")
    public void testDeveRetornarZeroQuandoCoberturaZero() {
        // Consulta com valor 200
        Consulta consulta = ConsultaTestHelper.criarConsultaPadrao();
        PlanoSaudeStub plano = new PlanoSaudeStub( percentualCobertura: 0);

        double reembolso = service.calcular(consulta, dummyPaciente, plano);

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

    @Test  new *
    @DisplayName("Exercício 9 - Cobertura com valor 100")
    public void testDeveRetornarValorInteiroQuandoCoberturaCem() {
        // Arrange
        Consulta consulta = ConsultaTestHelper.criarConsultaPadrao(); // Con
        PlanoSaudeStub plano = new PlanoSaudeStub( percentualCobertura: 100); //

        double reembolso = service.calcular(consulta, dummyPaciente, plano);

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

10. Exercício

11. Exercício

12. Exercício