

Relatório

Projeto “RAMRaiders”

2º Semestre 24/25

Descrição

O projeto desenvolvido é uma versão moderna inspirada no clássico **Galaga**, um jogo de naves espaciais lançado originalmente em 1981. O objetivo central do jogador é **sobreviver a ondas sucessivas de inimigos** que descem em formação e atacam com projéteis, enquanto o jogador move-se horizontalmente na parte inferior do ecrã e dispara para eliminar os inimigos.

Padrões de Projeto

- 1) **Strategy Pattern:** Este padrão é fundamental para definir a família de algoritmos de comportamento dos inimigos. Permite encapsular diferentes estratégias de movimento (como movimento em círculo, de cima para baixo ou em zig-zag) e de ataque (como ataque teleguiado, kamikaze ou disparo linear) em classes separadas. A classe `EnemyBehavior` utilizaria estas estratégias, podendo alternar entre elas dinamicamente, o que facilita a criação de inimigos com comportamentos variados e a introdução de novas estratégias sem modificar o código central do inimigo.
- 2) **Observer Pattern:** Utilizado para criar uma relação de dependência entre os objetos do jogo, especificamente entre os inimigos e o jogador. Os inimigos (observadores) registam-se para serem notificados sobre mudanças de estado ou posição do jogador (sujeito). Isto permite que os inimigos reajam dinamicamente às ações do jogador, como ajustar a mira de um ataque teleguiado (`HomingShootAttack`) ou iniciar um movimento evasivo.

Características Principais

No que toca a controlos, o jogador controla uma **nave espacial** que se move **horizontalmente** ao longo da base da tela. A nave pode **disparar projéteis** verticais para destruir inimigos. Existe **limitação no número de projéteis ativos**, simulando o comportamento clássico de Galaga.

Em relação aos inimigos, os mesmos surgem em **formações organizadas**, movendo-se lateralmente e, por vezes, em ziguezague. Alguns inimigos **descem em voo kamikaze**, tentando colidir com o jogador ou atirar projéteis diretamente. Cada inimigo possui um **comportamento definido** pela sua classe **EnemyBehavior**, que pode incluir movimentos simples, ataques individuais ou ataques coordenados em grupo. Os Inimigos têm **estratégias de ataque trocáveis**, como:

- Ataque direto (disparo para a posição atual do jogador).
- Ataque em curva ou em padrão (ziguezague, forma de “V”, etc).
- Ataque em grupo (vários inimigos disparam ao mesmo tempo).

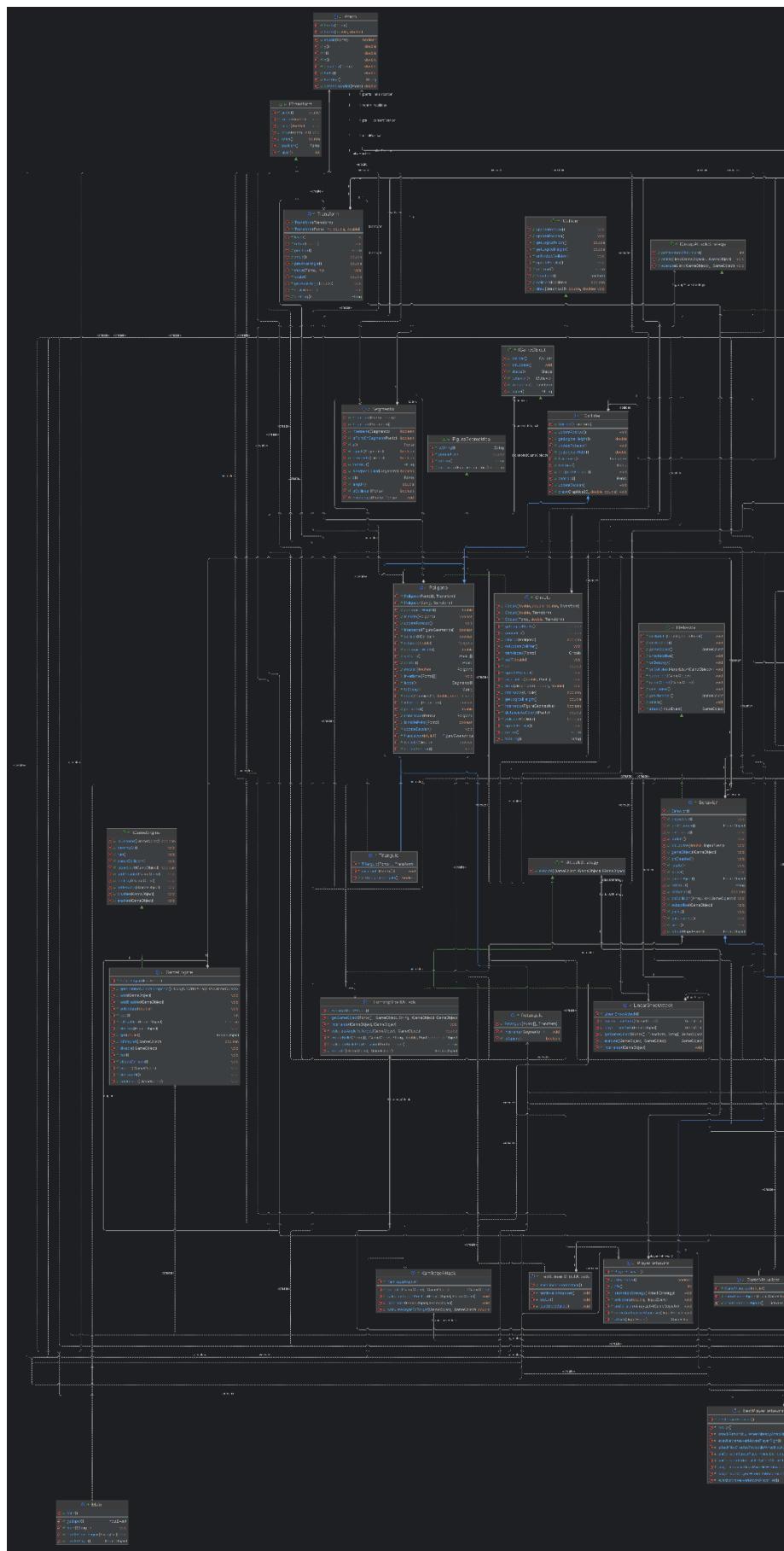
Em termos de colisões e danos, deu-se a utilização de colisores (circulares ou Poligonais) para deteção precisa de colisões. Quando um inimigo ou projétil colide com o jogador, o jogador perde uma vida e quando o projétil do jogador atinge um inimigo, o inimigo é destruído e pontos são atribuídos.

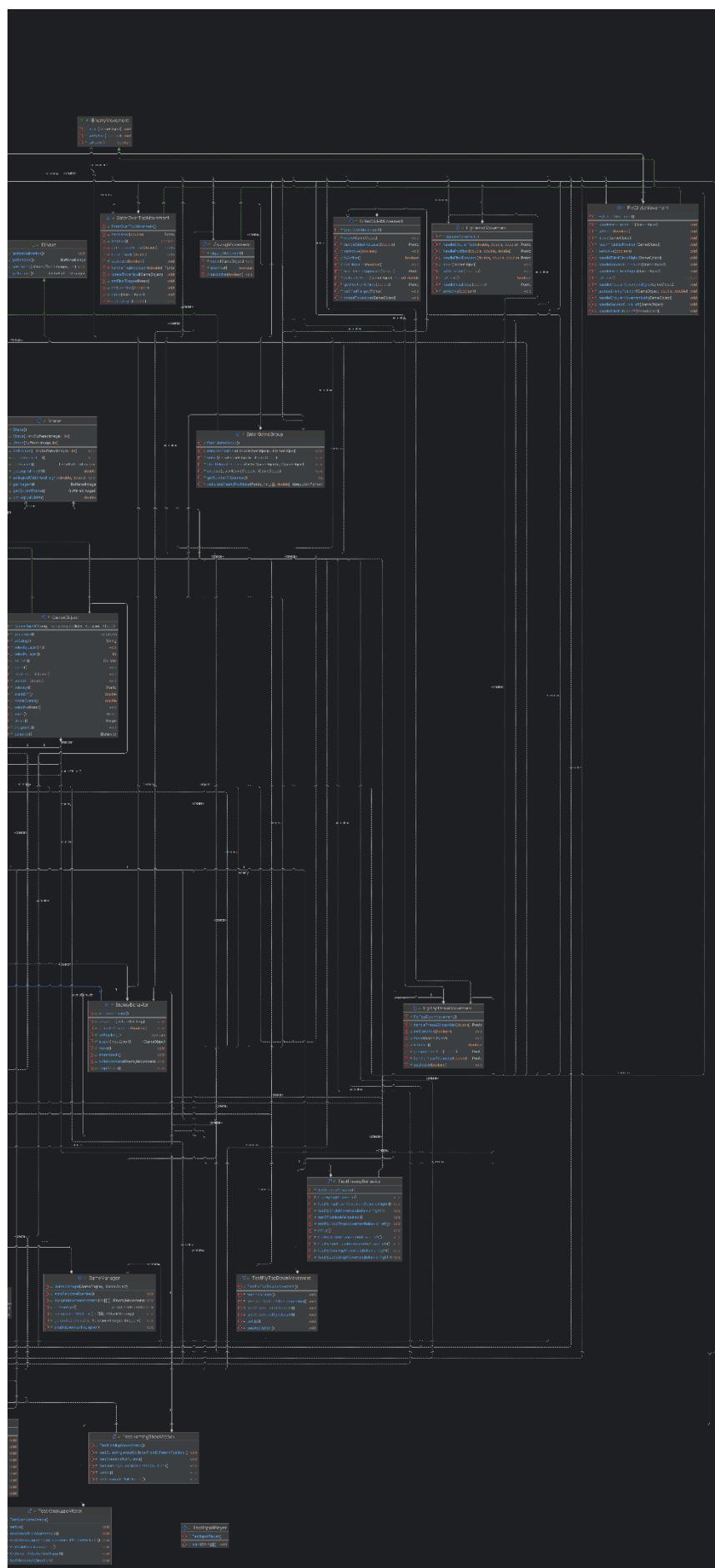
Os inimigos “observam” o jogador para reagir à sua posição:

- Alguns alinham os disparos à posição do jogador.
- Outros planeiam movimentos ou descidas em direção ao jogador.

A dificuldade aumenta com o tempo como por exemplo: Novos padrões de ataque são introduzidos, A frequência dos ataques aumenta e O número de inimigos por onda cresce.

Diagrama de Classes





[Diagrama UML – baixe aqui para maior qualidade](#)

JavaDoc

[JavaDoc RAMRaiders – baixe aqui](#)

Conclusão ou observação relevante

Implementou-se novos algoritmos para os inimigos e o jogador, assim contemplar uma ampla variedade de ataques e movimentos especiais. Alguns destes são:

- EnterOverTopMovement
- EnterSideMovement
- FlyCircleMovement
- LinearShootAttack
- HomingShootAttack

Criação da classe “GameManager”, esta gerênciaria os seguintes pontos:

- controlador central
- Sistema de camadas para renderização
- Coordenação de comportamentos e estratégias
- Integração com sistema de assets

Na hora de evoluir o “GameEngine” para o nosso jogo foi considerado o seguinte:

1. Thread-Safety: Uso de **CopyOnWriteArrayList** para garantir operações seguras em múltiplas threads
2. Modularização: Separação clara entre comportamentos, interfaces e gerenciamento do jogo
3. Uso de executores agendados para timing preciso
4. Cálculos com funções paramétricas para movimentação

Referências

[ScheduledExecutorService](#)

[Single ScheduledExecutorService instance vs Multiple ScheduledExecutorService instances](#)

[Is ScheduledExecutorService useful? - BlackBox.ai](#)

[Implementing Galaga Style Enemy Behavior](#)

[How to create a parametric function and what attributes I need to define one - ChatGPT](#)

[Assets galaga](#)

[Galaga – Nintendo NES system](#)

Testes Unitários

```

● ● ●

package test;

import game.*;
import game.behaviorItems.LinearShootAttack;

import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import geometry.Ponto;
import geometry.Retangulo;
import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

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

public class TestLinearShootAttack {
    private GameObject attacker;
    private LinearShootAttack shootAttack;
    private InputEvent inputEvent;

    @BeforeEach
    void setUp() {
        Ponto[] points = { new Ponto(2.0, 4), new Ponto(2.0, 0), new Ponto(0.0,
        0.0), new Ponto(0.0, 4.0) };
        Transform transform = new Transform(new Ponto(1.0, 2), 0, 0, 1);
        Retangulo rectangle = new Retangulo(points, transform);

        attacker = new GameObject(
            "TEST_ATTACKER",
            transform,
            rectangle,
            new Behavior(),
            new Shape());

        shootAttack = new LinearShootAttack();
        inputEvent = new InputEvent();
        inputEvent.setPlayerMove("ATTACK", true);
    }

    @Test
    void testShootBullet() {
        GameObject bullet = (GameObject) shootAttack.execute(attacker, null);
        GameObject bullet2 = (GameObject) shootAttack.execute(attacker, null);

        assertNotNull(bullet, "Bullet should not be null after attack execution");
        assertEquals("LINEAR_BULLET 0", bullet.name());
        assertEquals("LINEAR_BULLET 1", bullet2.name());
        assertEquals(1.0, bullet.transform().position().x(), "Bullet x position
        should be 1.0");

        // Test bullet movement
        Ponto initialPos = bullet.transform().position();
        bullet.onUpdate();
        assertNotEquals(initialPos, bullet.transform().position(), "Bullet should
        move after update");
    }

    @Test
    void testInvalidAttacker() {
        ByteArrayOutputStream outContent = new ByteArrayOutputStream();
        System.setOut(new PrintStream(outContent));

        assertThrows(IllegalArgumentException.class, () ->
        shootAttack.execute(null, null));
        assertEquals("LinearShootAttack:vi", outContent.toString().trim());

        assertThrows(IllegalArgumentException.class, () ->
        shootAttack.execute(null, attacker));
    }
}

```

```

package test;

import game.*;
import game.behaviorItems.FlyCircleMovement;
import game.behaviorItems.FlyTopDownMovement;
import geometry.Poligono;
import geometry.Ponto;
import geometry.Retangulo;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class TestEnemyBehavior
{
    private EnemyBehavior enemyBehavior;
    private GameObject enemy;

    @BeforeEach
    void setUp()
    {
        Ponto[] pts = {new Ponto(2.0, 4), new Ponto(2.0, 0), new Ponto(0.0, 0.0), new Ponto(0.0, 4.0)};
        Transform transform = new Transform(new Ponto(1.0, 2), 1, 90, 1);
        Poligono polygon = new Retangulo(pts, transform);
        enemyBehavior = new EnemyBehavior();

        enemy = new GameObject(
            "TestEnemy",
            transform,
            polygon,
            enemyBehavior,
            new Shape());
        this.enemy.onInit();
        this.enemyBehavior.activateMovement(true);
    }

    @Test
    void testZigZagMovement()
    {
        // Test multiple frames to verify zigzag pattern
        double[] expectedX = new double[4];

        // Store initial position
        Ponto initialPos = enemy.transform().position();

        // Run movement for 4 frames and store x positions
        for (int i = 0; i < 4; i++)
        {
            enemyBehavior.move();

            expectedX[i] = enemy.transform().position().x();
        }

        // Verify zigzag pattern (alternating left and right movement)
        assertTrue(expectedX[1] != expectedX[0], "Position should change after first move");
        assertTrue(expectedX[2] != expectedX[1], "Position should change after second move");

        assertTrue(Math.abs(expectedX[3] - initialPos.x()) <= 0.5, "Movement should stay within amplitude bounds");
    }

    @Test
    void testYPositionMaintained()
    {
        double initialY = enemy.transform().position().y();
        // Move multiple times
        for (int i = 0; i < 10; i++)
        {
            enemyBehavior.move();
            assertEquals(initialY, enemy.transform().position().y(), "Y position should remain constant");
        }
    }

    @Test
    void testFlyCircleMovementBehaviorLeft()
    {
        FlyCircleMovement flyCircleMovement = new FlyCircleMovement();
        enemyBehavior.setMovement(flyCircleMovement);

        assertFalse(flyCircleMovement.isActive(), "Movement should not be active");

        flyCircleMovement.setActive(true);
        flyCircleMovement.setIsLeft(true);
        Ponto initialPosition = enemy.transform().position();

        for (int i = 0; i < 80; i++)
        {
            enemy.onUpdate();

            if(!flyCircleMovement.isActive())
                break;

            assertTrue(flyCircleMovement.isActive());
        }

        assertEquals(initialPosition.x(), enemy.transform().position().x(), 0.1);

        assertTrue(fly.isActive());
    }
}

```



The screenshot shows a Java code editor with three test methods for movement behaviors:

- testFlyCircleMovementBehaviorLeft()**: Tests the FlyCircleMovement behavior for left movement. It initializes a FlyCircleMovement object, sets it to active and left, and then loops 80 times to check if it remains active. It also asserts that the enemy's position does not change.
- testFlyCircleMovementBehaviorRight()**: Tests the FlyCircleMovement behavior for right movement. Similar to the left test, but the enemy moves to the right.
- testFlyTopDownMovementBehaviorRight()**: Tests the FlyTopDownMovement behavior for right movement. It initializes a FlyTopDownMovement object, sets it to active and right, and then loops 100 times to check if it remains active. It also asserts that the enemy's position does not change.

```

● ● ●

package test;

import game.*;
import game.behaviorItems.KamikazeAttack;
import game.objectsInterface.IGameObject;
import geometry.Ponto;
import geometry.Retangulo;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.io.ByteArrayOutputStream;
import java.io.PrintStream;

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

public class TestKamikazeAttack
{
    private GameObject attacker;
    private GameObject target;
    private KamikazeAttack kamikazeAttack;
    private Ponto[] points = {
        new Ponto(2.0, 4.0),
        new Ponto(2.0, 0.0),
        new Ponto(0.0, 0.0),
        new Ponto(0.0, 4.0)
    };

    @BeforeEach
    void setUp()
    {
        // Setup attacker
        Transform attackerTransform = new Transform(new Ponto(10.0, 10.0), 0, 0, 1);
        Retangulo attackerRect = new Retangulo(points, attackerTransform);
        attacker = new GameObject(
            "TEST_ATTACKER",
            attackerTransform,
            attackerRect,
            new EnemyBehavior(),
            new Shape()
        );

        // Setup target
        Transform targetTransform = new Transform(new Ponto(0.0, 0.0), 0, 0, 1);
        Retangulo targetRect = new Retangulo(points, targetTransform);
        target = new GameObject(
            "TEST_TARGET",
            targetTransform,
            targetRect,
            new PlayerBehavior(),
            new Shape()
        );

        kamikazeAttack = new KamikazeAttack();
        target.onInit();
        attacker.onInit();
    }

    @Test
    void testValidKamikazeAttack()
    {
        IGameObject result = kamikazeAttack.execute(attacker, target);
        assertNotNull(result, "Attack result should not be null");
        assertEquals(attacker, result, "Should return the modified attacker");

        Ponto velocity = ((GameObject)result).velocity();
        assertNotNull(velocity, "Velocity should be set");
        assertEquals(1.0, Math.sqrt(velocity.x() * velocity.x() + velocity.y() * velocity.y()), 0.001,
                    "Velocity magnitude should be 1.0");

        assertEquals(225, attacker.transform().angle());
    }

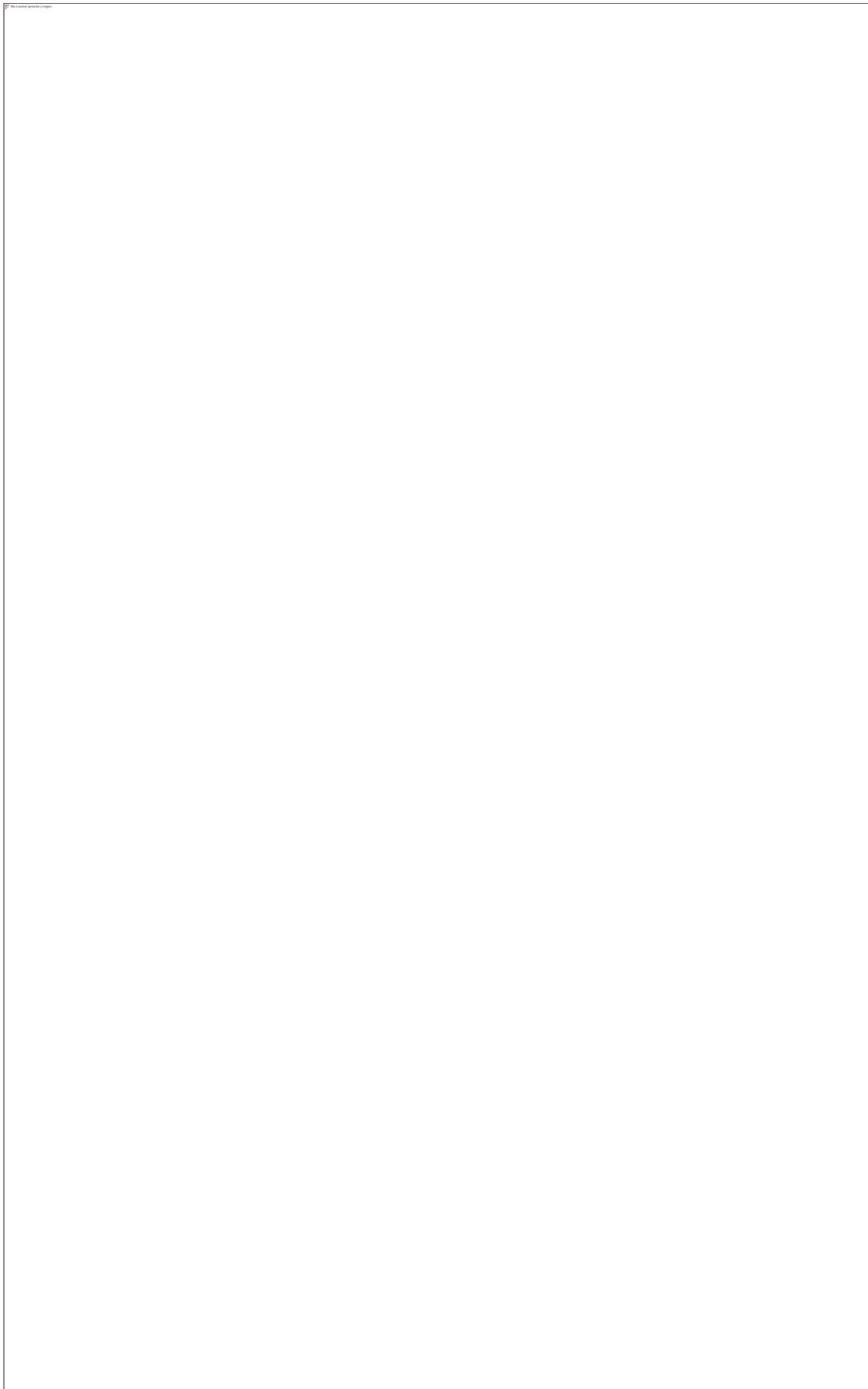
    @Test
    void testInvalidAttackerAndTarget()
    {
        ByteArrayOutputStream outContent = new ByteArrayOutputStream();
        System.setOut(new PrintStream(outContent));

        // Test null attacker
        assertThrows(IllegalArgumentException.class,
                    () -> kamikazeAttack.execute(null, target));
        assertEquals("KamikazeAttack:Vi", outContent.toString().trim());

        // Test null target
        assertThrows(IllegalArgumentException.class,
                    () -> kamikazeAttack.execute(attacker, null));

        // Test same object for attacker and target
        assertThrows(IllegalArgumentException.class,
                    () -> kamikazeAttack.execute(attacker, attacker));
    }
}

```



```
package test;
import game.*;
import game.objectsInterface.IGameObject;
import geometry.Ponto;
import geometry.Poligono;
import geometry.Retangulo;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import java.util.ArrayList;

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

public class TestPlayerBehavior
{
    private PlayerBehavior playerBehavior0;
    private GameObject gameObject0;

    private PlayerBehavior playerBehavior1;
    private GameObject gameObject1;

    private InputEvent inputEvent;

    @BeforeEach
    void setUp()
    {
        playerBehavior0 = new PlayerBehavior();

        Ponto[] pts = {new Ponto(2.0, 4), new Ponto(2.0, 0), new Ponto(0.0, 0.0), new Ponto(0.0, 4.0)};
        Transform transform = new Transform(new Ponto(1.0, 2), 0, 0, 1);
        Poligono polygon = new Retangulo(pts, transform);

        gameObject0 = new GameObject("PLAYER 0", transform, polygon, playerBehavior0, new Shape());
        gameObject0.onInit();

        playerBehavior1 = new PlayerBehavior();
        gameObject1 = new GameObject("PLAYER 1", transform, polygon, playerBehavior1, new Shape());
        gameObject1.onInit();

        inputEvent = new InputEvent();
    }

    @Test
    void attackNotCreatesProjectileWhenNotAttacking()
    {
        IGameObject result = playerBehavior0.attack(inputEvent);
        assertNull(result);
    }

    @Test
    void attackReturnsNullWhenAlreadyAttacking()
    {
        inputEvent.setPlayerMove("MOUSE_LEFT", true);
        IGameObject firstAttack = playerBehavior0.attack(inputEvent);
        assertNotNull(firstAttack);

        IGameObject secondAttack = playerBehavior0.attack(inputEvent);
        assertNull(secondAttack);
    }

    @Test
    void onCollisionReducesLifeByOneWhenHit()
    {
        ArrayList<IGameObject> collisions = new ArrayList<>();
        collisions.add(gameObject1);

        playerBehavior0.onCollision(collisions);
        assertEquals(2, playerBehavior0.life());
    }

    @Test
    void onCollisionMakesPlayerInvincibleTemporarily() throws InterruptedException
    {
        ArrayList<IGameObject> collisions = new ArrayList<>();
        collisions.add(gameObject1);

        playerBehavior0.onCollision(collisions);
        assertTrue(playerBehavior0.isInvincible());

        Thread.sleep(1500);
        assertFalse(playerBehavior0.isInvincible());
    }
}
```

```
  @Test
  void playerRemainsAliveWhenHitWithInvincibility() throws InterruptedException
  {
    ArrayList<IGameObject> collisions = new ArrayList<>();
    collisions.add(gameObject1);

    playerBehavior0.onCollision(collisions); // hit
    playerBehavior0.onCollision(collisions);

    Thread.sleep(1300);
    assertEquals(2, playerBehavior0.life());
    playerBehavior0.onCollision(collisions); // hit
    assertEquals(1, playerBehavior0.life());
    assertTrue(playerBehavior0.isEnabled());
  }

  @Test
  void playerIsDestroyedWhenLifeReachesZero() throws InterruptedException
  {
    ArrayList<IGameObject> collisions = new ArrayList<>();
    collisions.add(gameObject1);

    assertEquals(3, playerBehavior0.life());
    playerBehavior0.onCollision(collisions); // hit
    Thread.sleep(1300);
    assertEquals(2, playerBehavior0.life());
    playerBehavior0.onCollision(collisions); // hit
    Thread.sleep(1300);
    assertEquals(1, playerBehavior0.life());
    playerBehavior0.onCollision(collisions); // hit

    assertEquals(0, playerBehavior0.life());
    assertFalse(playerBehavior0.isEnabled());

  }

  @Test
  void evasiveManeuverMovesPlayerRight()
  {
    inputEvent.setPlayerMove("RIGHT", true);
    Ponto initialPosition = gameObject0.transform().position();

    playerBehavior0.evasiveManeuver(inputEvent);

    Ponto newPosition = gameObject0.transform().position();
    assertTrue(newPosition.x() > initialPosition.x());
    assertEquals(initialPosition.y(), newPosition.y());
  }

  @Test
  void evasiveManeuverMovesPlayerLeft() throws InterruptedException
  {
    inputEvent.setPlayerMove("LEFT", true);
    Ponto initialPosition = gameObject0.transform().position();

    playerBehavior0.evasiveManeuver(inputEvent);
    assertTrue(playerBehavior0.isInvincible());

    Ponto newPosition = gameObject0.transform().position();
    assertTrue(newPosition.x() < initialPosition.x());
    assertEquals(initialPosition.y(), newPosition.y());

    Thread.sleep(1200);
    assertFalse(playerBehavior0.isInvincible());
  }
}
```



```
    @Test
    void testScaleNegative() {
        GameObjects negativeScaleObject = new GameObjects(name, transform, collider, velocity,
        velocityLayer, rotateSpeed, -0.2);
        negativeScaleObject.scale();
        assertEquals(0.8, transform.scale());
    }

    @Test
    void testMove() {
        Ponto initialPosition = transform.position();
        gameObject.move();
        Ponto expectedPosition = new Ponto(initialPosition.x() + velocity.x(), initialPosition.y() +
        velocity.y());
        assertEquals(expectedPosition.x(), transform.position().x());
        assertEquals(expectedPosition.y(), transform.position().y());
    }

    @Test
    void testMoveZeroVelocity() {
        GameObjects stationaryObject = new GameObjects(name, transform, collider, new Ponto(0.0, 0.0),
        velocityLayer, rotateSpeed, scaleDiff);
        stationaryObject.move();
        assertEquals(transform.position().x(), 0.0);
        assertEquals(transform.position().y(), 0.0);
    }

    @Test
    void testUpdate() {
        Ponto initialPosition = transform.position();
        double initialScale = transform.scale();
        double initialAngle = transform.angle();

        gameObject.update();

        assertEquals(initialAngle + rotateSpeed, transform.angle());
        assertEquals(initialScale + scaleDiff, transform.scale());
        Ponto expectedPosition = new Ponto(initialPosition.x() + velocity.x(), initialPosition.y() +
        velocity.y());
        assertEquals(expectedPosition.x(), transform.position().x());
        assertEquals(expectedPosition.y(), transform.position().y());
    }

    @Test
    void testMultipleSequentialUpdates() {
        gameObject.update();
        gameObject.update();
        gameObject.update();

        assertEquals(30.0, transform.angle());
        assertEquals(1.3, transform.scale());
        assertEquals(3.0, transform.position().x());
    }

    @Test
    void testLayerMovement() {
        int initialLayer = transform.layer();
        gameObject.move();
        assertEquals(initialLayer + velocityLayer, transform.layer());
    }

    @Test
    void testMultipleUpdates() {
        Ponto initialPosition = gameObject.transform().position();
        gameObject.update();
        gameObject.update();
        Ponto finalPosition = gameObject.transform().position();
        assertNotEquals(initialPosition, finalPosition);
    }

    @Test
    void testToString() {
        String expected = name + "\n" + transform.toString() + "\n" + collider.toString();
        assertEquals(expected, gameObject.toString());
    }
}
```

```
package test;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import geometry.Ponto;
import game.Transform;

class TransformTest {

    private Transform transform;

    @BeforeEach
    void setUp() {
        transform = new Transform(new Ponto(1.0, 2.0), 1, 45.0, 1.0);
    }

    @Test
    void constructsTransformWithParameters() {
        assertNotNull(transform);
        assertEquals(new Ponto(1.0, 2.0), transform.position());
        assertEquals(1, transform.layer());
        assertEquals(45.0, transform.angle());
        assertEquals(1.0, transform.scale());
    }

    @Test
    void constructsTransformByCopying() {
        Transform copiedTransform = new Transform(transform);
        assertNotNull(copiedTransform);
        assertEquals(transform.position(), copiedTransform.position());
        assertEquals(transform.layer(), copiedTransform.layer());
        assertEquals(transform.angle(), copiedTransform.angle());
        assertEquals(transform.scale(), copiedTransform.scale());
    }

    @Test
    void movesTransform() {
        transform.move(new Ponto(2.0, 3.0), 1);
        assertEquals(new Ponto(3.0, 5.0), transform.position());
        assertEquals(2, transform.layer());
    }

    @Test
    void rotatesTransform() {
        transform.rotate(15.0);
        assertEquals(60.0, transform.angle());
    }

    @Test
    void scalesTransform() {
        transform.scale(2.0);
        assertEquals(3.0, transform.scale());
    }
}
```

```


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

import geometry.Circulo;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import geometry.Ponto;
import game.GameObjects;
import game.GameEngine;
import game.Transform;
import game.Collider;
import game.Layer;

class GameEngineTest {
    private GameEngine gameEngine;
    private GameObjects gameObject1;
    private GameObjects gameObject2;

    @BeforeEach
    void setUp() {
        gameEngine = new GameEngine();

        Transform transform1 = new Transform(new Ponto(1, 1), 0, 0, 1);
        Collider collider1 = new Circulo(transform1.position(), 1.0, transform1);
        gameObject1 = new GameObjects("Object1", transform1, collider1, new Ponto(1, 0), 0, 0, 0);

        Transform transform2 = new Transform(new Ponto(3, 3), 0, 0, 1);
        Collider collider2 = new Circulo(transform2.position(), 1.5, transform2);
        gameObject2 = new GameObjects("Object2", transform2, collider2, new Ponto(-1, 0), 0, 0, 0);
    }

    @Test
    void testRemoveGameObject() {
        gameEngine.add(gameObject1);
        gameEngine.destroy(gameObject1);
        assertEquals(0, gameEngine.size());
    }

    @Test
    void testUpdateGameObjects() {
        gameEngine.add(gameObject1);
        gameEngine.add(gameObject2);

        gameEngine.update();

        assertEquals(new Ponto(2, 1), gameObject1.transform().position());
        assertEquals(new Ponto(2, 3), gameObject2.transform().position());
    }

    @Test
    void testCollisionDetection() {
        gameEngine.add(gameObject1);
        gameEngine.add(gameObject2);

        gameObject2.transform().move(new Ponto(1, 1), 0); // Moving Object2 closer to Object1
        gameEngine.update();

        assertTrue(gameObject1.collider().colision(gameObject2.collider()));
    }

    @Test
    void testMultipleObjectCollisions() {
        Transform transform3 = new Transform(new Ponto(1.5, 1.5), 0, 0, 1);
        Collider collider3 = new Circulo(transform3.position(), 1.0, transform3);
        GameObjects gameObject3 = new GameObjects("Object3", transform3, collider3, new Ponto(0, 0), 0, 0, 0);

        gameEngine.add(gameObject1);
        gameEngine.add(gameObject2);
        gameEngine.add(gameObject3);

        gameEngine.update();
        assertTrue(gameObject1.collider().colision(gameObject3.collider()));
    }

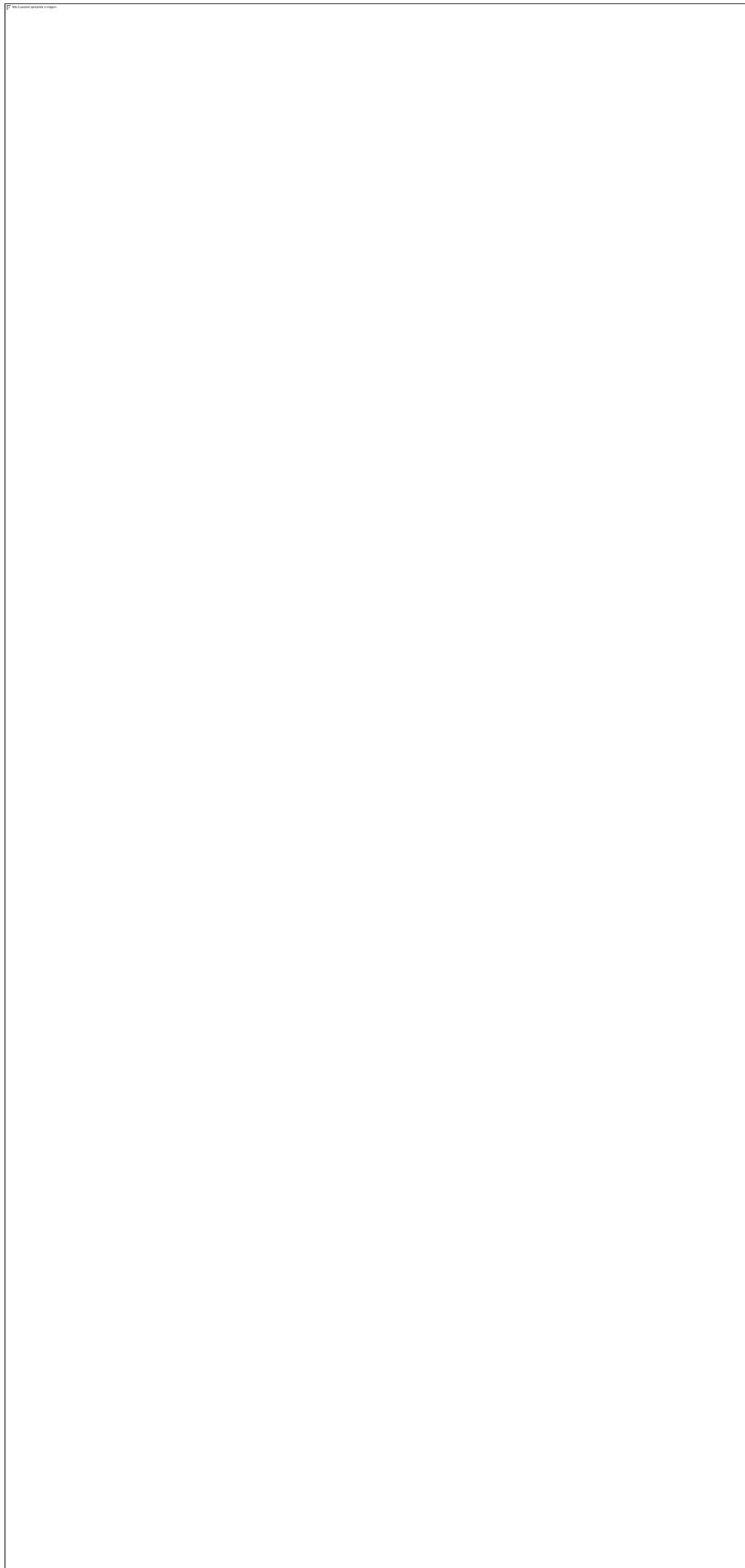
    @Test
    void testLayerBasedUpdates() {
        Transform t1 = new Transform(new Ponto(0, 0), 1, 0, 1);
        Transform t2 = new Transform(new Ponto(0, 0), 2, 0, 1);

        GameObjects obj1 = new GameObjects("Layer1", t1, new Circulo(t1.position(), 1.0, t1),
                                         new Ponto(1, 0), 1, 0, 0);
        GameObjects obj2 = new GameObjects("Layer2", t2, new Circulo(t2.position(), 1.0, t2),
                                         new Ponto(1, 0), 2, 0, 1);

        gameEngine.add(obj1);
        gameEngine.add(obj2);
        gameEngine.update();

        assertNotEquals(obj1.transform().position(), obj2.transform().position());
    }
}


```





```

// Test intersection methods
@Test
void testIntersecta() {
    // Test circle-circle intersection
    Circulo c1 = new Circulo(0.0, 0.0, 5.0, transform);
    Circulo c2 = new Circulo(3.0, 3.0, 3.0, transform);
    assertTrue(c1.intersecta(c2));

    // Test circle-polygon intersection
    assertTrue(circulo.intersecta(poligono));

    // Test polygon-polygon intersection
    Ponto[] pts2 = new Ponto[] {
        new Ponto(5.0, 5.0),
        new Ponto(15.0, 5.0),
        new Ponto(15.0, 15.0),
        new Ponto(5.0, 15.0)
    };
    Poligono p2 = new Poligono(pts2, transform);
    assertTrue(poligono.intersecta(p2));

    // Test non-intersecting shapes
    Circulo c3 = new Circulo(20.0, 20.0, 2.0, transform);
    assertFalse(circulo.intersecta(c3));
    assertFalse(poligono.intersecta(c3));
}

// Test isInside methods
@Test
void testIsInside() {
    // Test circle inside polygon
    Circulo c = new Circulo(5.0, 5.0, 2.0, transform);
    assertTrue(poligono.isInside(c));

    // Test polygon inside circle
    Ponto[] smallPts = new Ponto[] {
        new Ponto(2.0, 2.0),
        new Ponto(3.0, 2.0),
        new Ponto(3.0, 3.0),
        new Ponto(2.0, 3.0)
    };
    Poligono smallPoly = new Poligono(smallPts, transform);
    assertTrue(circulo.isInside(smallPoly));

    // Test circle with center at polygon center
    Circulo c2 = new Circulo(5.0, 5.0, 4.0, transform);
    assertTrue(poligono.isInside(c2));
}

// Test update methods
@Test
void testUpdate() {
    // Test circle update
    transform.move(new Ponto(10.0, 10.0), 1);
    transform.scale(2.0);
    circulo.update();
    assertEquals(10.0, circulo.centro().x());
    assertEquals(10.0, circulo.centro().y());
    assertEquals(10.0, circulo.r());

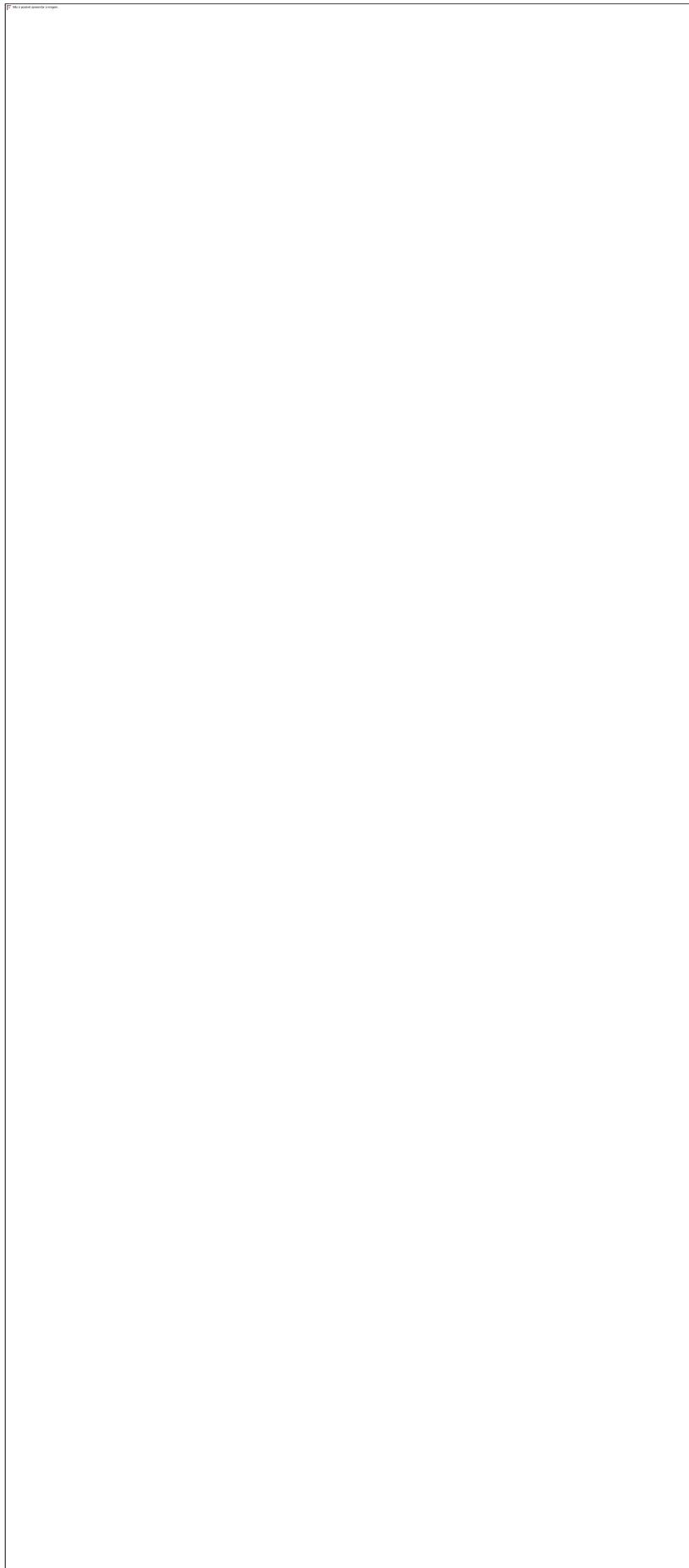
    // Test polygon update
    transform.move(new Ponto(20.0, 20.0), 1);
    transform.scale(1.5);
    transform.rotate(45.0);
    poligono.update();
    assertEquals(20.0, poligono.centro().x());
    assertEquals(20.0, poligono.centro().y());
}

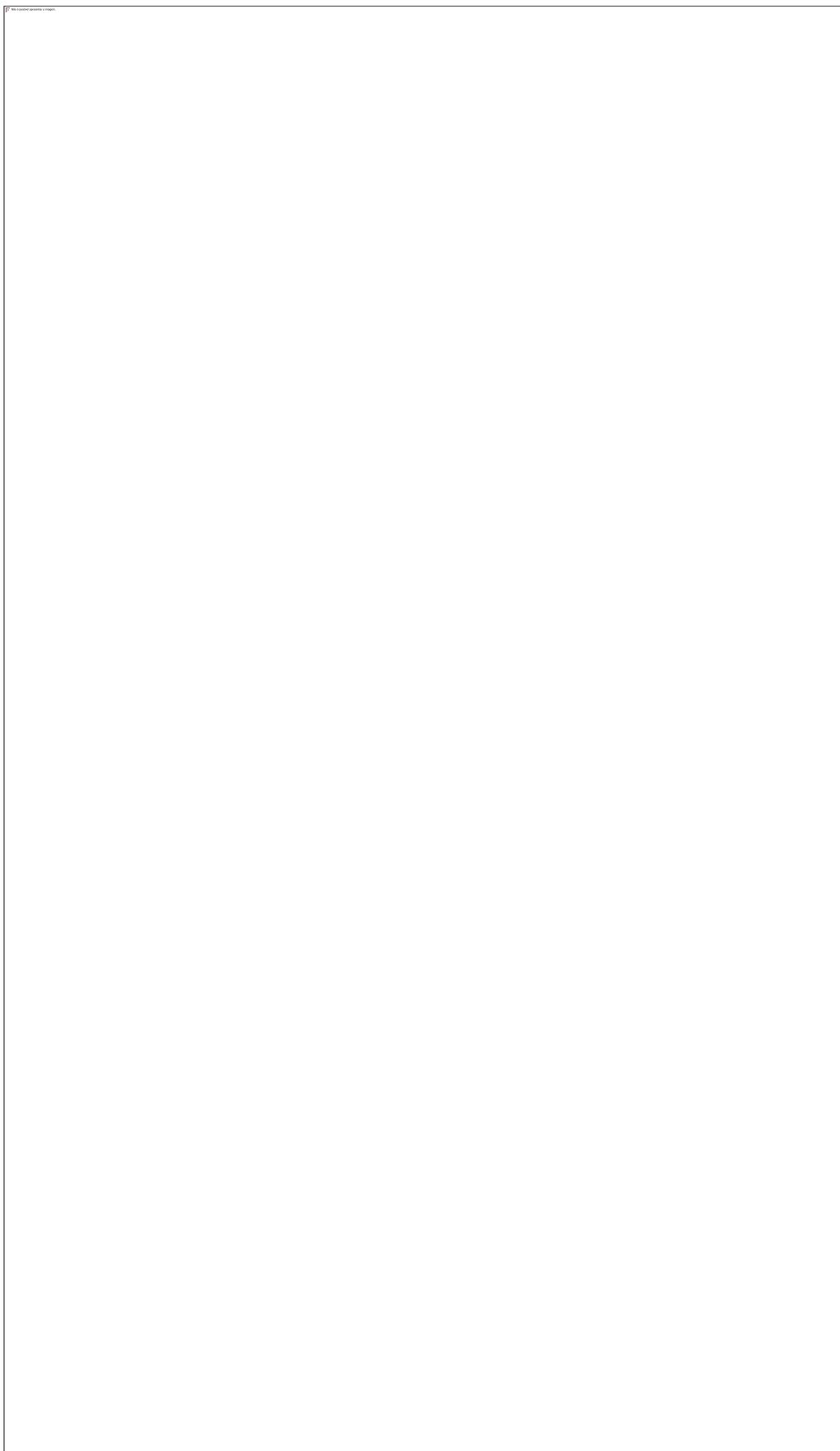
// Test translation methods
@Test
void testTranslacao() {
    // Test circle translation
    Ponto newCenter = new Ponto(10.0, 10.0);
    Circulo c = circulo.translacao(newCenter);
    assertEquals(10.0, c.centro().x());
    assertEquals(10.0, c.centro().y());

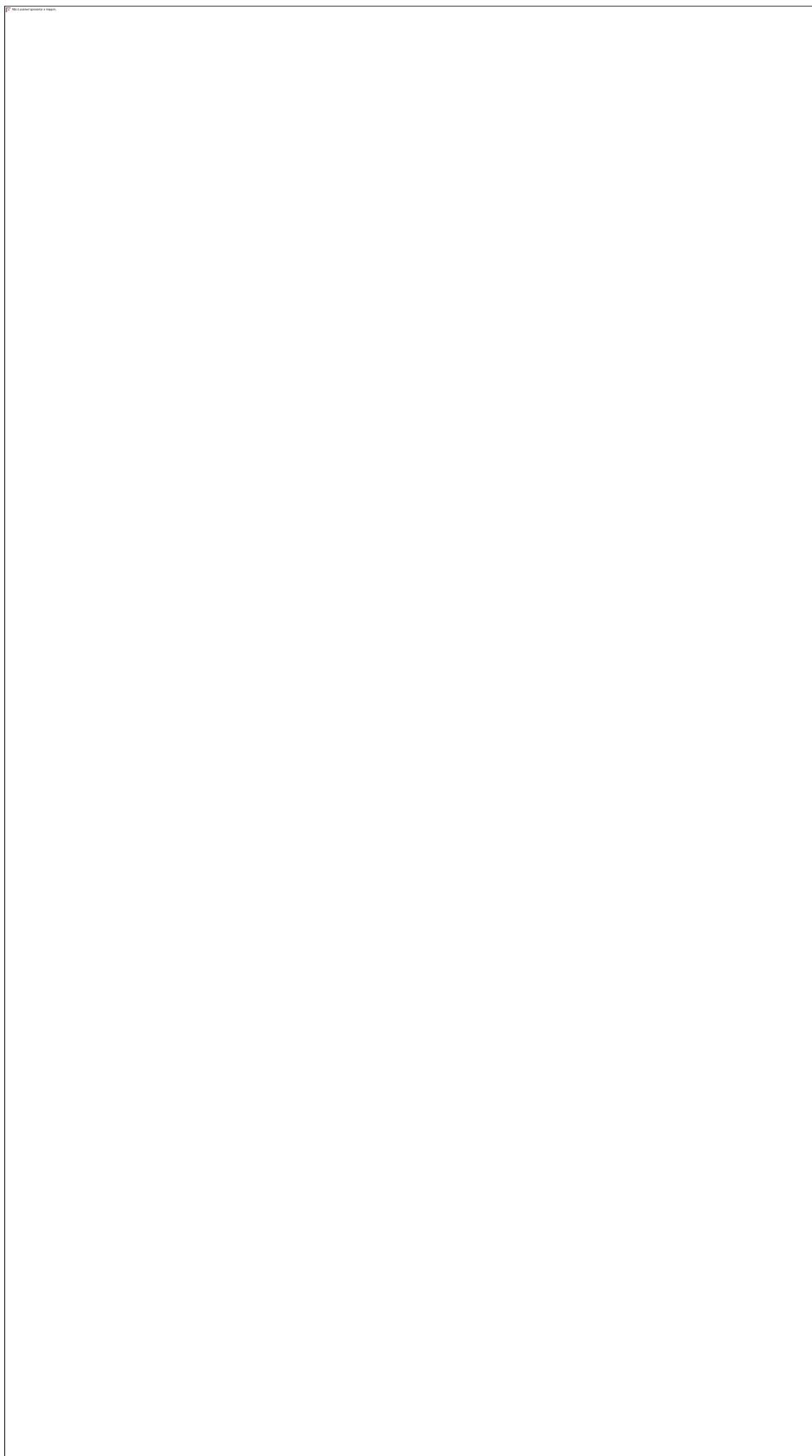
    // Test polygon translation
    Poligono p = poligono.translacao(newCenter);
    assertEquals(10.0, p.centro().x());
    assertEquals(10.0, p.centro().y());

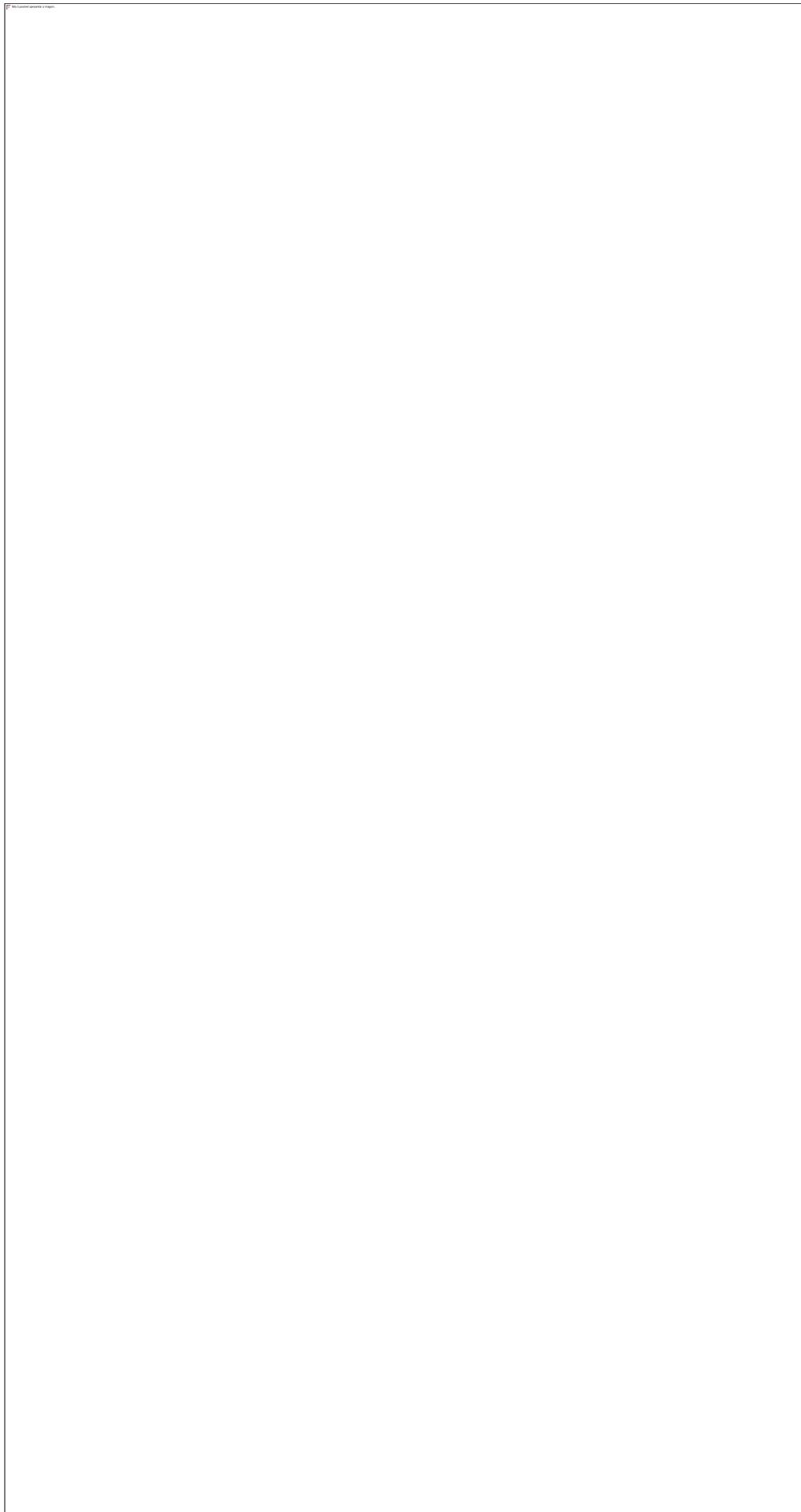
    // Test translation with negative coordinates
    Ponto negCenter = new Ponto(-5.0, -5.0);
    Circulo c2 = circulo.translacao(negCenter);
    assertEquals(-5.0, c2.centro().x());
    assertEquals(-5.0, c2.centro().y());
}

```











The screenshot shows a Java code editor with a dark theme. The code is a test class named `SegmentoTest` located in the `test` package. The class contains several test methods using the `@Test` annotation from the `org.junit.jupiter.api` package. The tests cover various aspects of the `Segmento` class, including constructor validation, length calculation, perpendicularity checks, and intersection detection.

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

public class SegmentoTest {

    @Test
    void constructorWithValidPoints() {
        Ponto p1 = new Ponto(1, 2);
        Ponto p2 = new Ponto(3, 4);
        Segmento seg = new Segmento(p1, p2);
        assertEquals(p1, seg.a());
        assertEquals(p2, seg.b());
    }

    @Test
    void constructorWithEqualPoints() {
        Ponto p1 = new Ponto(1, 2);
        assertThrows(IllegalArgumentException.class, () -> new Segmento(p1, p1));
    }

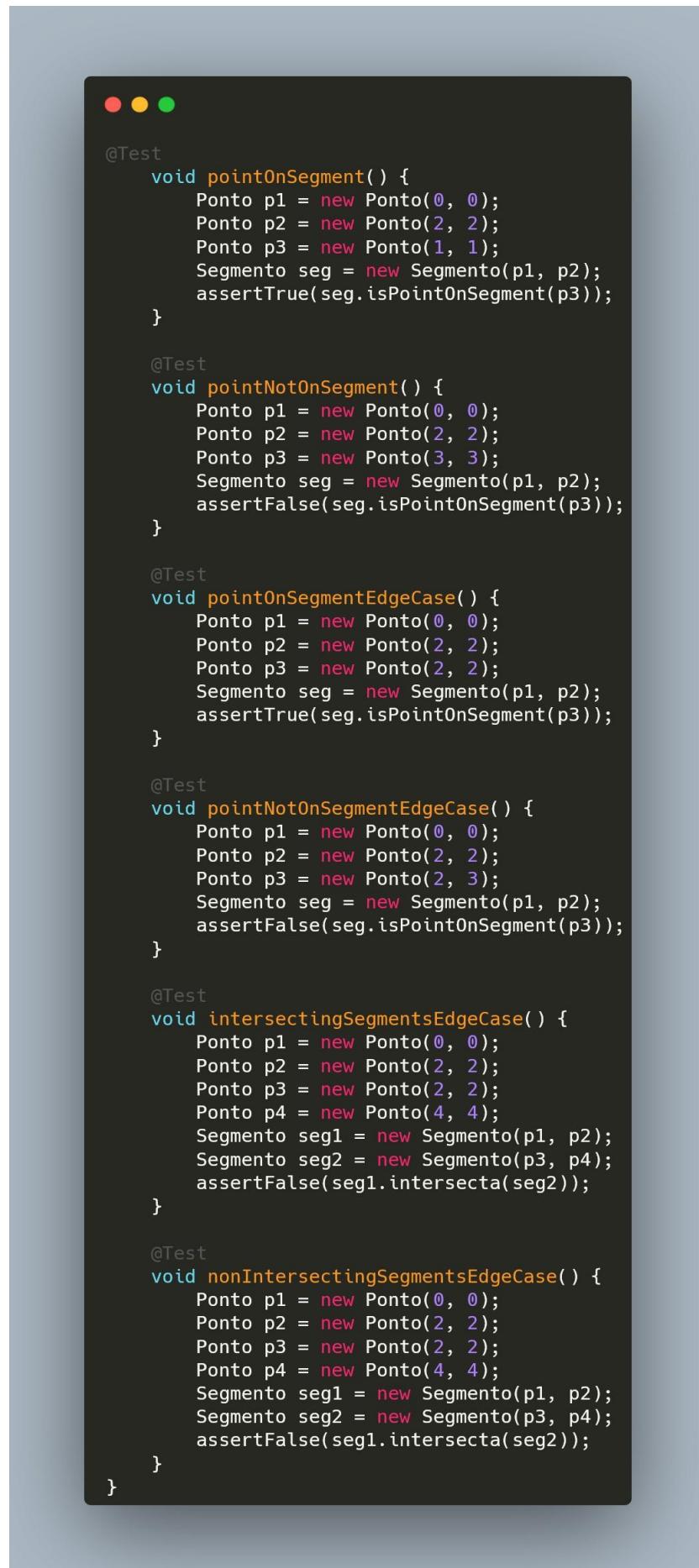
    @Test
    void lengthCalculation() {
        Ponto p1 = new Ponto(0, 0);
        Ponto p2 = new Ponto(3, 4);
        Segmento seg = new Segmento(p1, p2);
        assertEquals(5, seg.length(), 0.001);
    }

    @Test
    void perpendicularSegments() {
        Ponto p1 = new Ponto(0, 0);
        Ponto p2 = new Ponto(1, 0);
        Ponto p3 = new Ponto(0, 0);
        Ponto p4 = new Ponto(0, 1);
        Segmento seg1 = new Segmento(p1, p2);
        Segmento seg2 = new Segmento(p3, p4);
        assertTrue(seg1.isPerpendicular(seg2));
    }

    @Test
    void nonPerpendicularSegments() {
        Ponto p1 = new Ponto(0, 0);
        Ponto p2 = new Ponto(1, 1);
        Ponto p3 = new Ponto(0, 0);
        Ponto p4 = new Ponto(1, 0);
        Segmento seg1 = new Segmento(p1, p2);
        Segmento seg2 = new Segmento(p3, p4);
        assertFalse(seg1.isPerpendicular(seg2));
    }

    @Test
    void intersectingSegments() {
        Ponto p1 = new Ponto(0, 0);
        Ponto p2 = new Ponto(2, 2);
        Ponto p3 = new Ponto(0, 2);
        Ponto p4 = new Ponto(2, 0);
        Segmento seg1 = new Segmento(p1, p2);
        Segmento seg2 = new Segmento(p3, p4);
        assertTrue(seg1.intersecta(seg2));
    }

    @Test
    void nonIntersectingSegments() {
        Ponto p1 = new Ponto(0, 0);
        Ponto p2 = new Ponto(1, 1);
        Ponto p3 = new Ponto(2, 2);
        Ponto p4 = new Ponto(3, 3);
        Segmento seg1 = new Segmento(p1, p2);
        Segmento seg2 = new Segmento(p3, p4);
        assertFalse(seg1.intersecta(seg2));
    }
}
```



The screenshot shows a Java code editor with several test methods for a `Segmento` class. The code uses `Ponto` and `Segmento` objects to perform assertions about point containment and segment intersection.

```
@Test
void pointOnSegment() {
    Ponto p1 = new Ponto(0, 0);
    Ponto p2 = new Ponto(2, 2);
    Ponto p3 = new Ponto(1, 1);
    Segmento seg = new Segmento(p1, p2);
    assertTrue(seg.isPointOnSegment(p3));
}

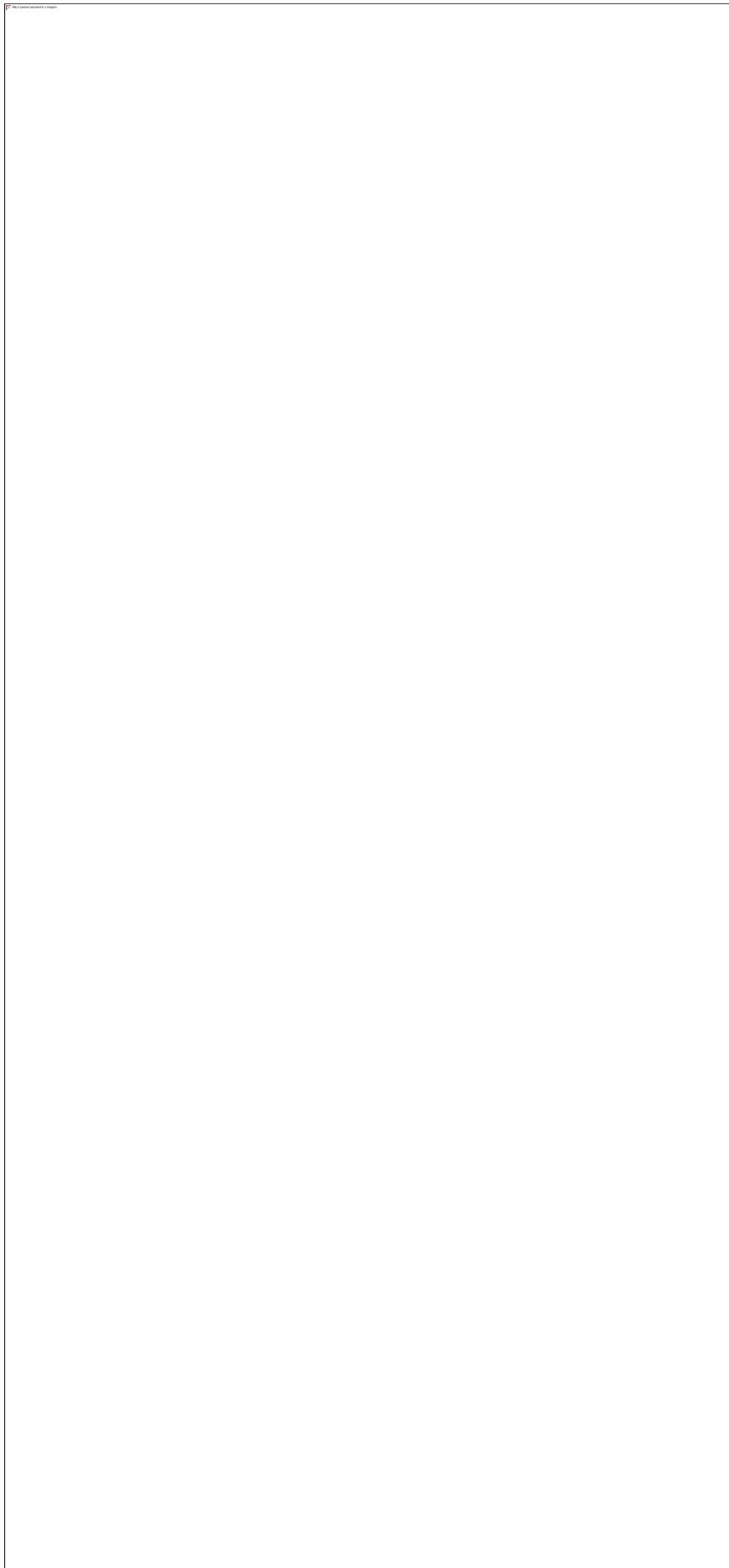
@Test
void pointNotOnSegment() {
    Ponto p1 = new Ponto(0, 0);
    Ponto p2 = new Ponto(2, 2);
    Ponto p3 = new Ponto(3, 3);
    Segmento seg = new Segmento(p1, p2);
    assertFalse(seg.isPointOnSegment(p3));
}

@Test
void pointOnSegmentEdgeCase() {
    Ponto p1 = new Ponto(0, 0);
    Ponto p2 = new Ponto(2, 2);
    Ponto p3 = new Ponto(2, 2);
    Segmento seg = new Segmento(p1, p2);
    assertTrue(seg.isPointOnSegment(p3));
}

@Test
void pointNotOnSegmentEdgeCase() {
    Ponto p1 = new Ponto(0, 0);
    Ponto p2 = new Ponto(2, 2);
    Ponto p3 = new Ponto(2, 3);
    Segmento seg = new Segmento(p1, p2);
    assertFalse(seg.isPointOnSegment(p3));
}

@Test
void intersectingSegmentsEdgeCase() {
    Ponto p1 = new Ponto(0, 0);
    Ponto p2 = new Ponto(2, 2);
    Ponto p3 = new Ponto(2, 2);
    Ponto p4 = new Ponto(4, 4);
    Segmento seg1 = new Segmento(p1, p2);
    Segmento seg2 = new Segmento(p3, p4);
    assertFalse(seg1.intersecta(seg2));
}

@Test
void nonIntersectingSegmentsEdgeCase() {
    Ponto p1 = new Ponto(0, 0);
    Ponto p2 = new Ponto(2, 2);
    Ponto p3 = new Ponto(2, 2);
    Ponto p4 = new Ponto(4, 4);
    Segmento seg1 = new Segmento(p1, p2);
    Segmento seg2 = new Segmento(p3, p4);
    assertFalse(seg1.intersecta(seg2));
}
```



```
assertThrows(IllegalArgumentException.class, () -> {
    new Poligono(new Ponto[]{new Ponto(0.0, 0.0), new Ponto(1.0, 1.0)}, transform);
});

// Test invalid string constructor
assertThrows(IllegalArgumentException.class, () -> {
    new Poligono("", transform);
});

assertThrows(IllegalArgumentException.class, () -> {
    new Poligono("0.0 0.0", transform);
});

assertThrows(IllegalArgumentException.class, () -> {
    new Poligono("0.0 0.0 1.0 1.0", transform);
});

assertThrows(NumberFormatException.class, () -> {
    new Poligono("invalid string", transform);
});
}

// Access Method Tests
@Test
void testLados() {
    Segmento[] lados = square.lados();
    assertEquals(4, lados.length);
    assertEquals(4.0, lados[0].length());
    assertEquals(4.0, lados[1].length());
    assertEquals(4.0, lados[2].length());
    assertEquals(4.0, lados[3].length());
}

@Test
void testVertices() {
    Ponto[] vertices = square.vertices();
    assertEquals(4, vertices.length);
    assertEquals(0.0, vertices[0].x());
    assertEquals(0.0, vertices[0].y());
    assertEquals(4.0, vertices[1].x());
    assertEquals(0.0, vertices[1].y());
}

@Test
void testToString() {
    String expected = "0.0 0.0 4.0 0.0 4.0 4.0 0.0 4.0";
    assertEquals(expected, square.toString());
}

@Test
void testPerimetro() {
    assertEquals(16.0, square.perimetro());
    assertEquals(12.0, triangle.perimetro());
    assertTrue(pentagon.perimetro() > 0);
}

@Test
void testCentro() {
    Ponto squareCentro = square.centro();
    assertEquals(2.0, squareCentro.x());
    assertEquals(2.0, squareCentro.y());

    Ponto triangleCentro = triangle.centro();
    assertEquals(1.0, triangleCentro.x());
    assertEquals(1.33, triangleCentro.y(), 0.01);

    Ponto pentagonCentro = pentagon.centro();
    assertEquals(2.0, pentagonCentro.x());
    assertEquals(2.2, pentagonCentro.y(), 0.1);
}
```

```
// Transformation Tests
@Test
void testTranslacao() {
    // Test translacao with dx, dy
    Poligono translated = (Poligono) square.translacao(2, 3);
    assertEquals(2.0, translated.vertices()[0].x());
    assertEquals(3.0, translated.vertices()[0].y());

    // Test translacao with new center
    Ponto newCenter = new Ponto(5.0, 5.0);
    Poligono translatedToCenter = square.translacao(newCenter);
    assertEquals(5.0, translatedToCenter.centro().x());
    assertEquals(5.0, translatedToCenter.centro().y());

    // Test zero translation
    assertSame(square, square.translacao(0, 0));
}

@Test
void testRotacao() {
    // Test 90-degree rotation
    Poligono rotated90 = square.rotacao(90.0);
    assertEquals(2.0, rotated90.centro().x());
    assertEquals(2.0, rotated90.centro().y());

    // Test 180-degree rotation
    Poligono rotated180 = square.rotacao(180.0);
    assertEquals(2.0, rotated180.centro().x());
    assertEquals(2.0, rotated180.centro().y());

    // Test 360-degree rotation
    Poligono rotated360 = square.rotacao(360.0);
    assertEquals(0.0, rotated360.vertices()[0].x());
    assertEquals(0.0, rotated360.vertices()[0].y());
}

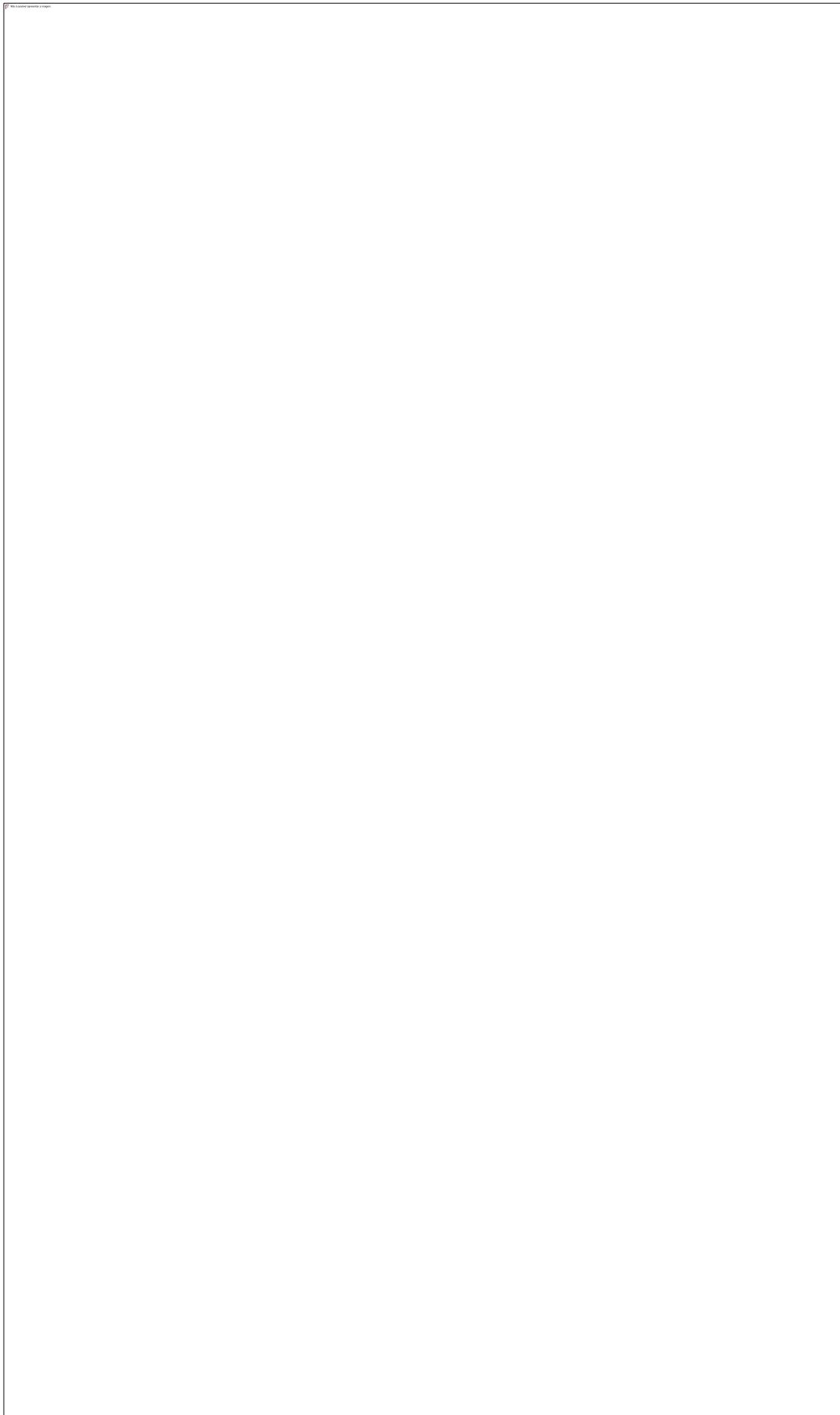
@Test
void testEscalar() {
    // Test scale up
    Poligono scaledUp = square.escalar(2.0);
    assertEquals(8.0, scaledUp.perimetro());

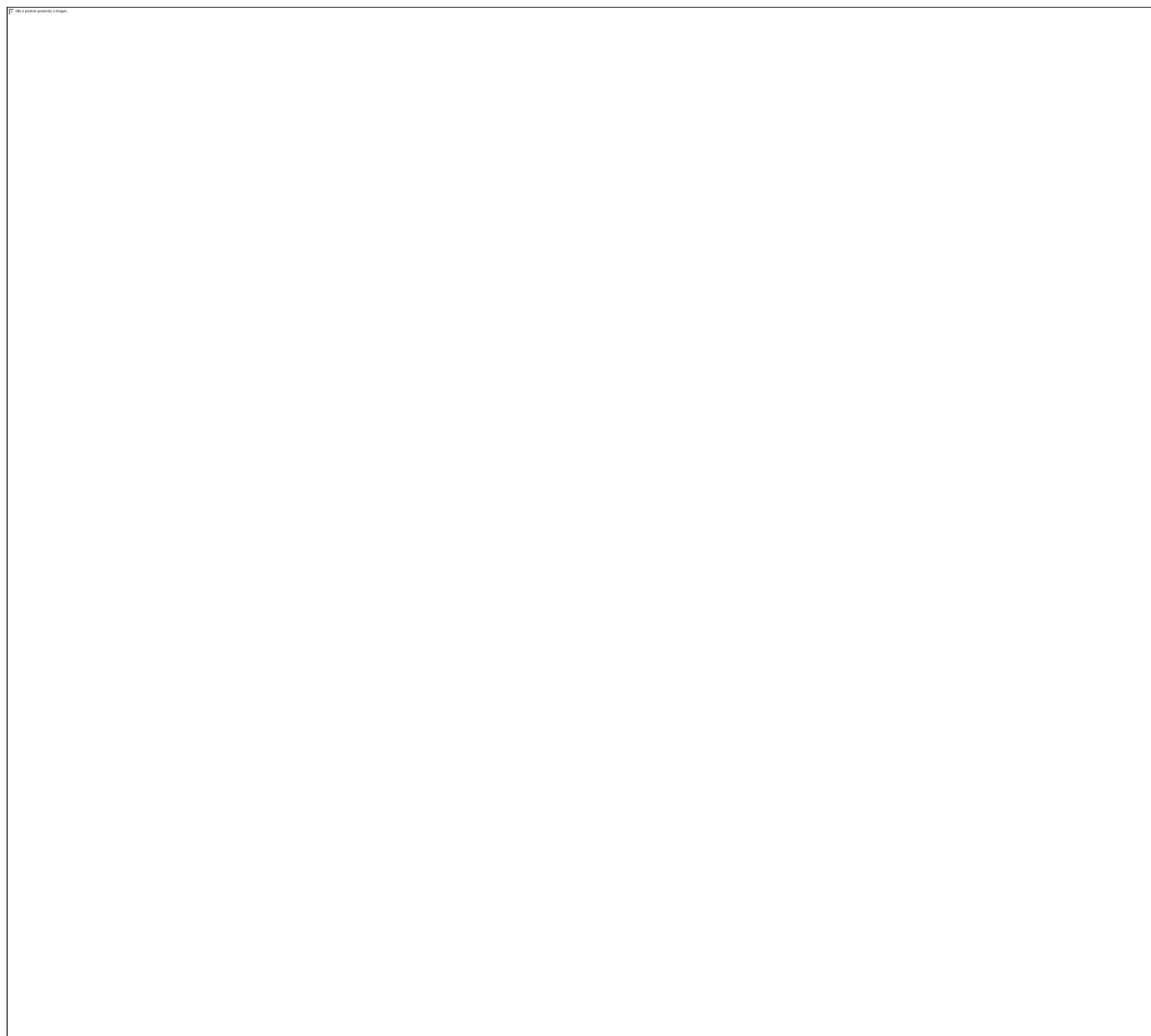
    // Test scale down
    Poligono scaledDown = square.escalar(0.5);
    assertEquals(4.0, scaledDown.perimetro());

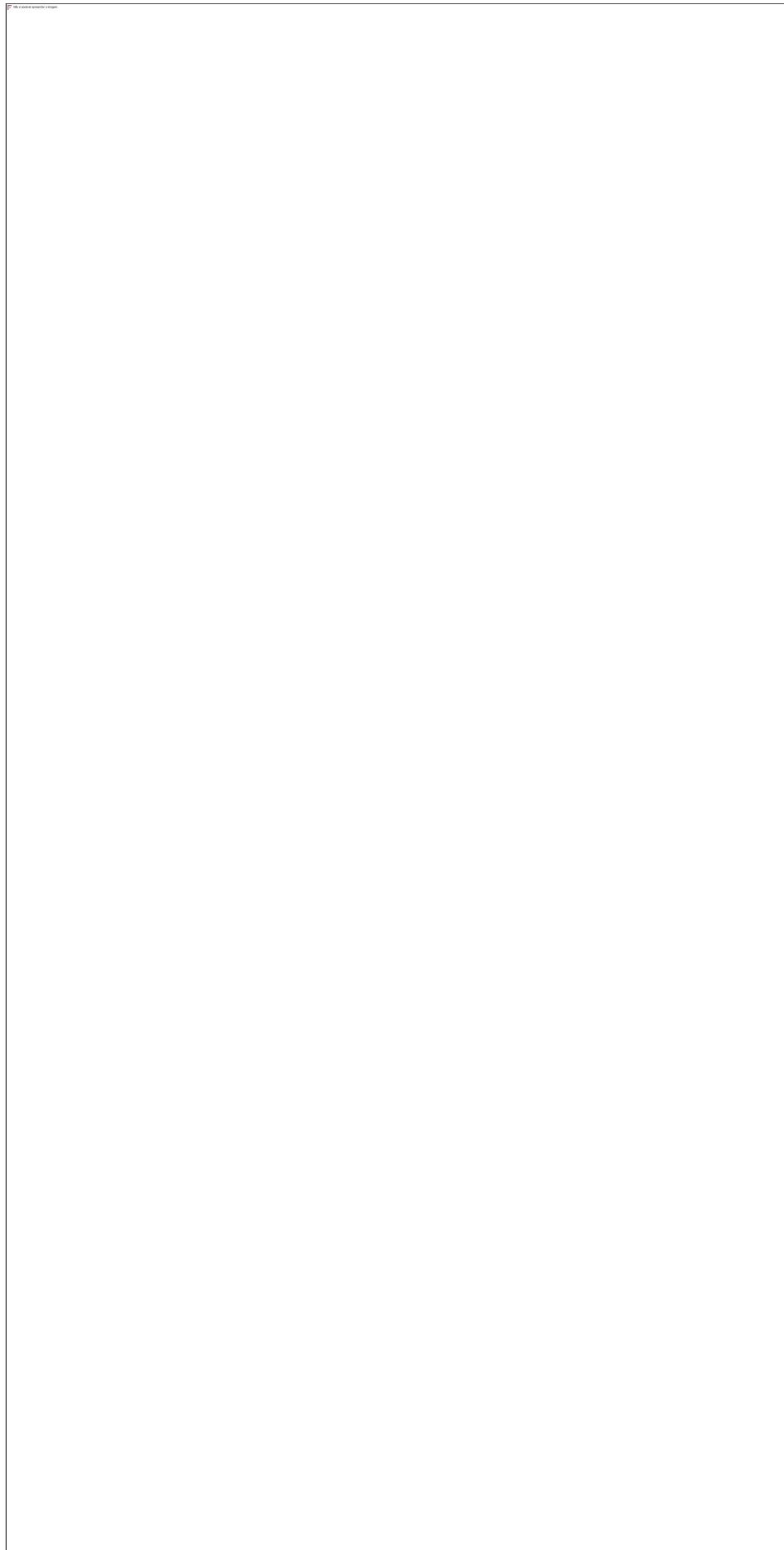
    // Test scale with 1.0
    assertSame(square, square.escalar(1.0));
}

// Intersection and Containment Tests
@Test
void testIsInside() {
    // Test polygon inside polygon
    Ponto[] smallSquare = new Ponto[] {
        new Ponto(1.0, 1.0),
        new Ponto(2.0, 1.0),
        new Ponto(2.0, 2.0),
        new Ponto(1.0, 2.0)
    };
    Poligono smallSquarePoligono = new Poligono(smallSquare, transform);
    assertTrue(square.isInside(smallSquarePoligono));
}
```

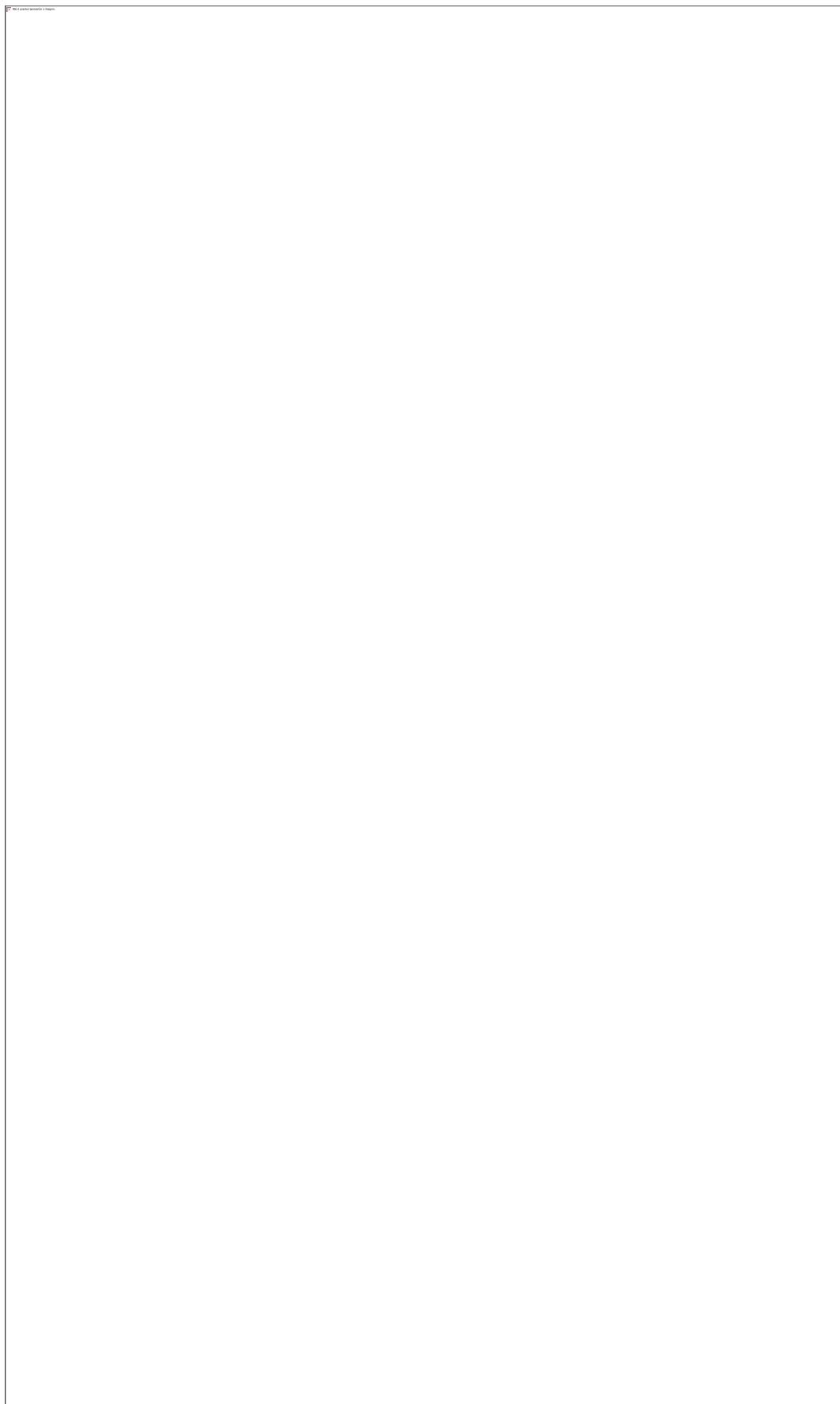


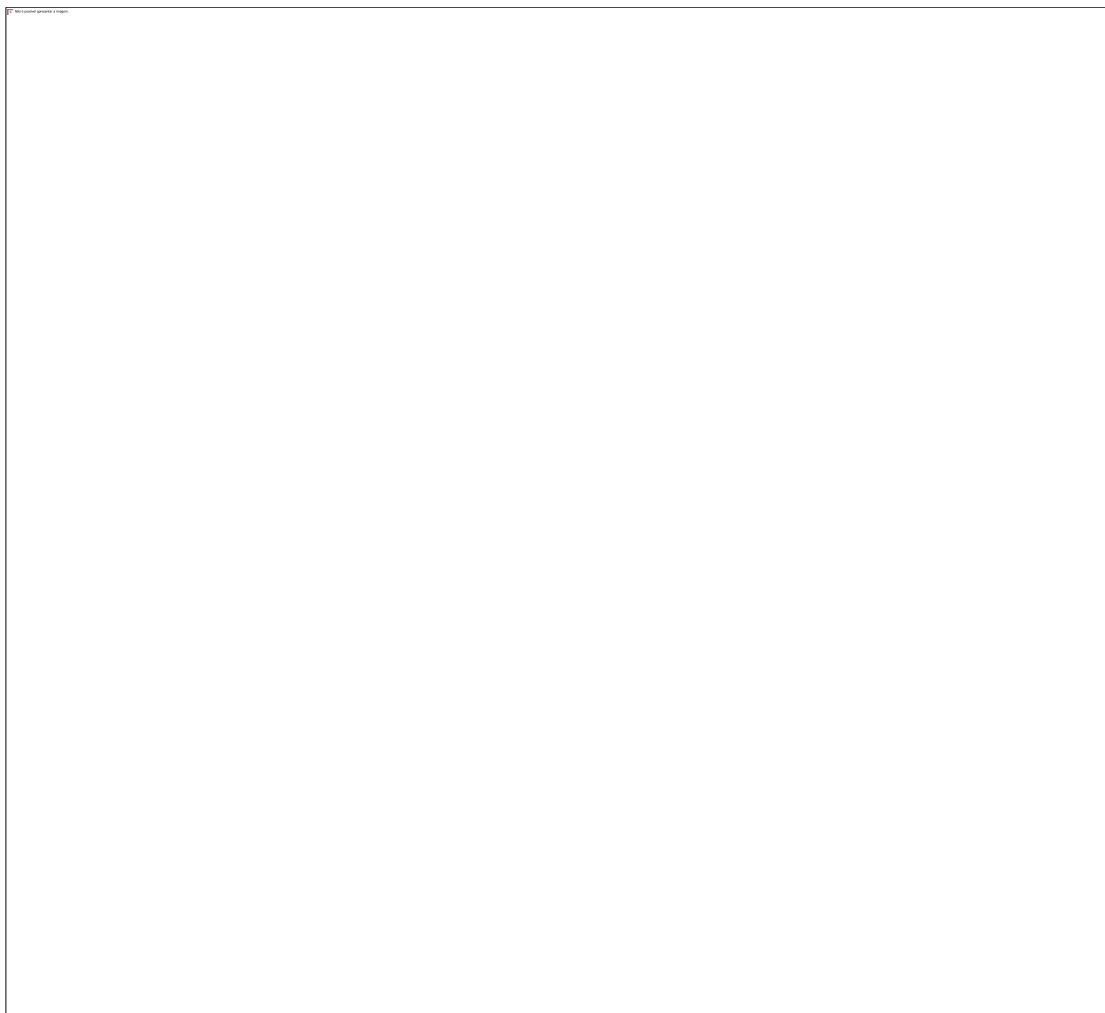


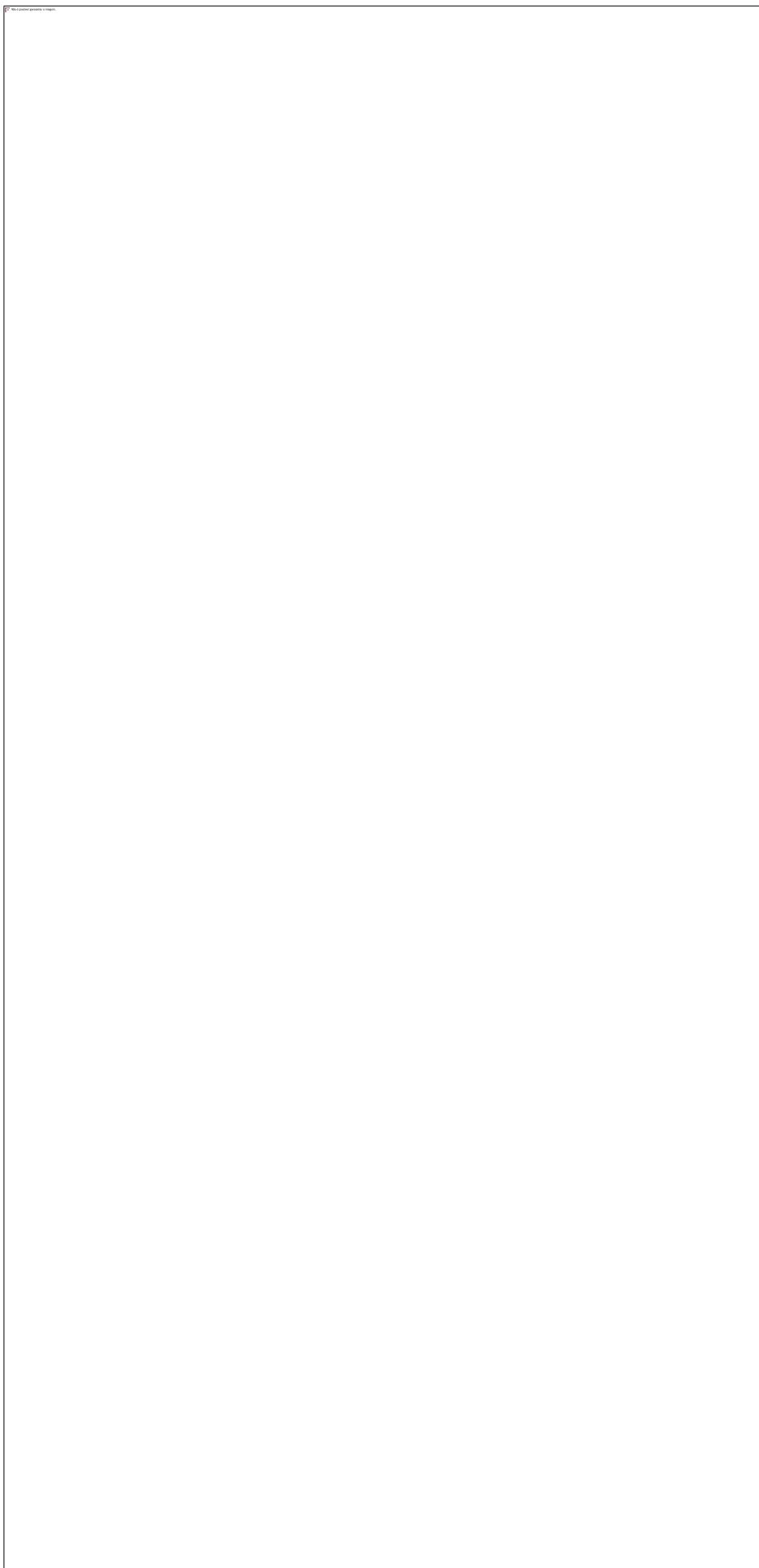


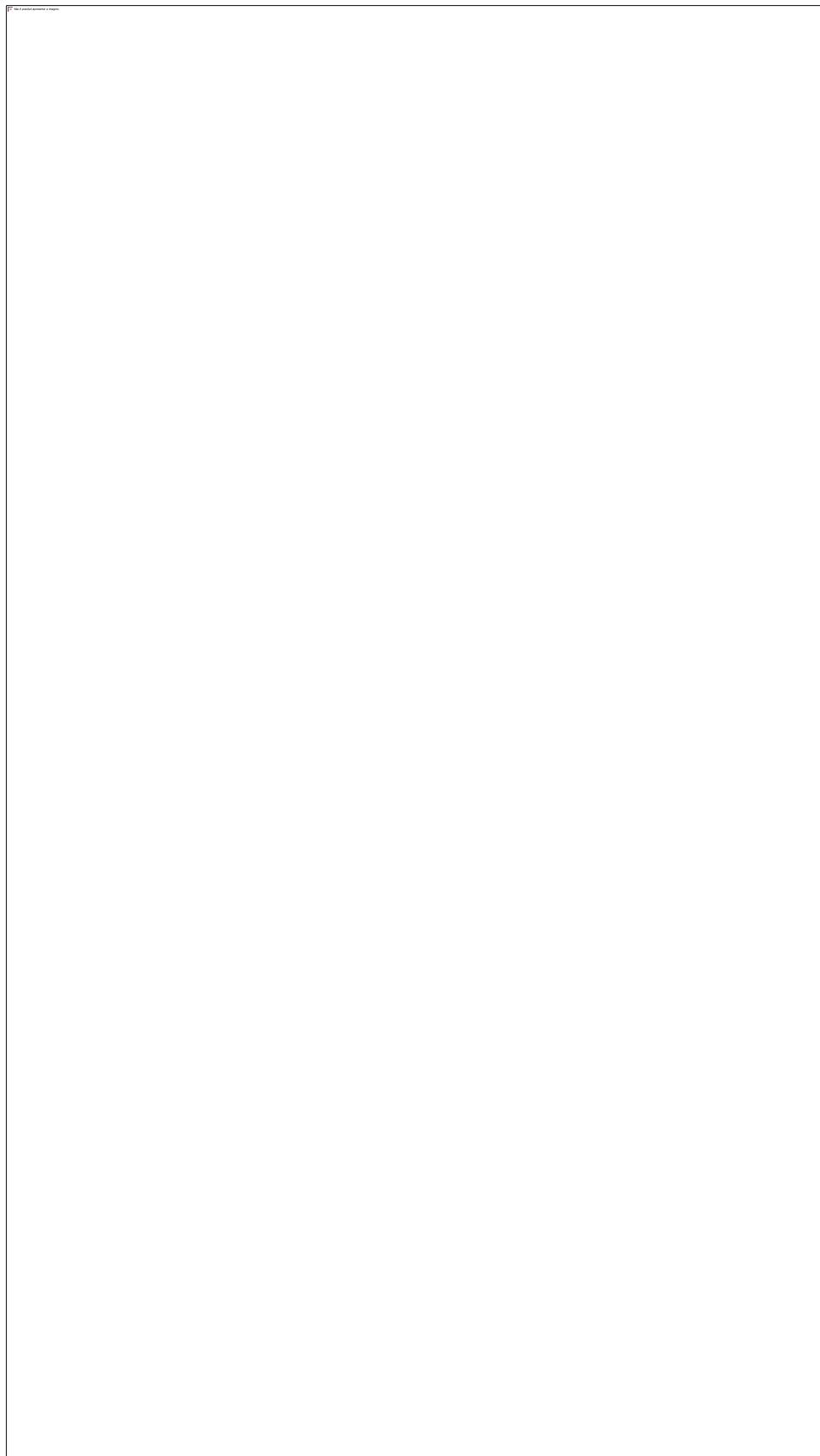


```
● ● ●  
  
@Test  
void testPerimetro() {  
    Ponto p1 = new Ponto(0.0, 0.0);  
    Ponto p2 = new Ponto(0.0, 2.0);  
    Ponto p3 = new Ponto(3.0, 2.0);  
    Ponto p4 = new Ponto(3.0, 0.0);  
    Ponto[] pontos = {p1, p2, p3, p4};  
    Retangulo ret = new Retangulo(pontos, transform);  
    assertEquals(10.0, ret.perimetro());  
}  
  
@Test  
void testCentro() {  
    Ponto p1 = new Ponto(0.0, 0.0);  
    Ponto p2 = new Ponto(0.0, 2.0);  
    Ponto p3 = new Ponto(3.0, 2.0);  
    Ponto p4 = new Ponto(3.0, 0.0);  
    Ponto[] pontos = {p1, p2, p3, p4};  
    Retangulo ret = new Retangulo(pontos, transform);  
    Ponto centro = ret.centro();  
    assertEquals(1.5, centro.x());  
    assertEquals(1.0, centro.y());  
}  
}
```











```
@Test
void testFlyLoopDropMovementBehaviorLeft()
{
    FlyLassoMovement fly = new FlyLassoMovement();
    enemyBehavior.setMovement(fly);
    assertFalse(fly.isActive(), "Movement should not be active");
    fly.setActive(true);
    fly.setDirection(false);

    for (int i = 0; i < 150; i++)
    {
        enemy.onUpdate();

        if(!fly.isActive())
            break;

        assertTrue(fly.isActive());
    }
}

@Test
void testFlySideLassoMovementBehaviorLeft()
{
    EnterSideMovement fly = new EnterSideMovement();
    enemyBehavior.setMovement(fly);
    assertFalse(fly.isActive(), "Movement should not be active");
    fly.setActive(true);

    fly.setFinalTarget(new Ponto(10,12));

    for (int i = 0; i < 150; i++)
    {
        enemy.onUpdate();

        if(!fly.isActive())
            break;

        assertTrue(fly.isActive());
    }
}

@Test
void testFlyOverTopMovementBehaviorRight()
{
    EnterOverTopMovement fly = new EnterOverTopMovement();
    enemyBehavior.setMovement(fly);
    assertFalse(fly.isActive(), "Movement should not be active");
    fly.setDirection(false);
    Ponto target = new Ponto(8,2);
    fly.setFinalTarget(target);

    fly.setActive(true);

    for (int i = 0; i < 300; i++)
    {
        System.out.println(enemy.transform().position());
        enemy.onUpdate();

        if(!fly.isActive())
            break;

        assertTrue(fly.isActive());
    }

    assertEquals(target.x(),enemy.transform().position().x(),0.1);
    assertEquals(target.y(),enemy.transform().position().y(),0.1);
}
```

```

package test;

import core.*;
import core.behaviorItems.FlyTopDownMovement;
import geometry.Polygon;
import geometry.Ponto;
import geometry.Retangulo;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class TestFlyTopDownMovement
{
    private FlyTopDownMovement movement;
    private GameObject enemy;

    @BeforeEach
    void setup()
    {
        Ponto[] pts = {new Ponto(2.0, 4), new Ponto(2.0, 0),
                      new Ponto(0.0, 0.0), new Ponto(0.0, 4.0)};
        Transform transform = new Transform(new Ponto(1.0, 2), 1, 90, 1);
        Polygon polygon = new Retangulo(pts, transform);
        movement = new FlyTopDownMovement();

        enemy = new GameObject(
            "TestEnemy",
            transform,
            polygon,
            new EnemyBehavior(),
            new Shape()
        );
        enemy.onInit();
    }

    @Test
    void testInitialState()
    {
        assertFalse(movement.isActive(), "Movement should be inactive initially");
    }

    @Test
    void testActivation()
    {
        movement.setActive(true);
        assertTrue(movement.isActive(), "Movement should be active after activation");

        movement.setActive(false);
        assertFalse(movement.isActive(), "Movement should be inactive after deactivation");
    }

    @Test
    void testMovementLeftToRight()
    {
        movement.setActive(true);
        movement.setDirection(true);
        Ponto initialPos = enemy.transform().position();

        movement.move(enemy);
        enemy.onUpdate();

        Ponto newPos = enemy.transform().position();
        assertTrue(newPos.x() > initialPos.x(), "Should move right");
        assertTrue(newPos.y() < initialPos.y(), "Should move down");
    }

    @Test
    void testMovementRightToLeft()
    {
        movement.setActive(true);
        movement.setDirection(false);
        Ponto initialPos = enemy.transform().position();

        movement.move(enemy);
        enemy.onUpdate();

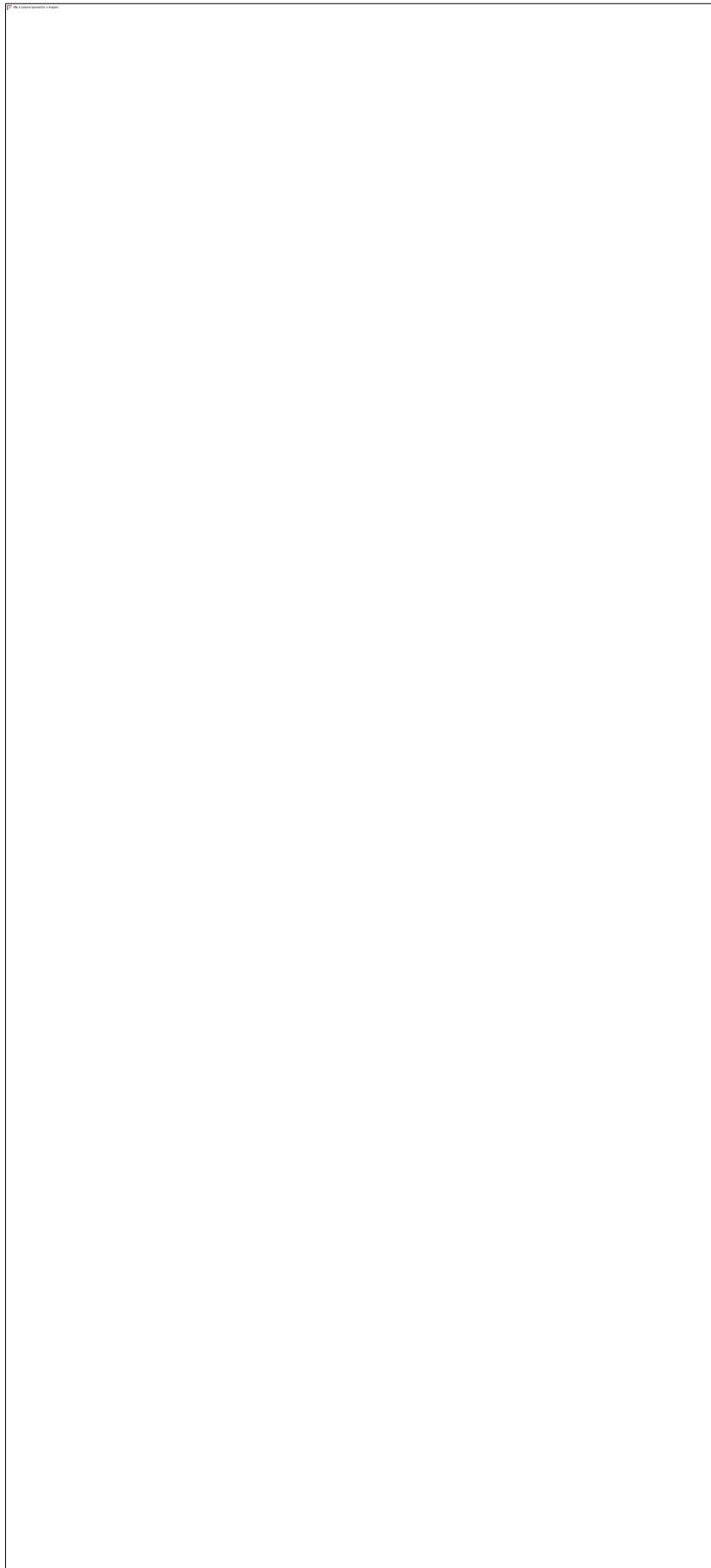
        Ponto newPos = enemy.transform().position();
        assertTrue(newPos.x() < initialPos.x(), "Should move left");
        assertTrue(newPos.y() < initialPos.y(), "Should move down");
    }

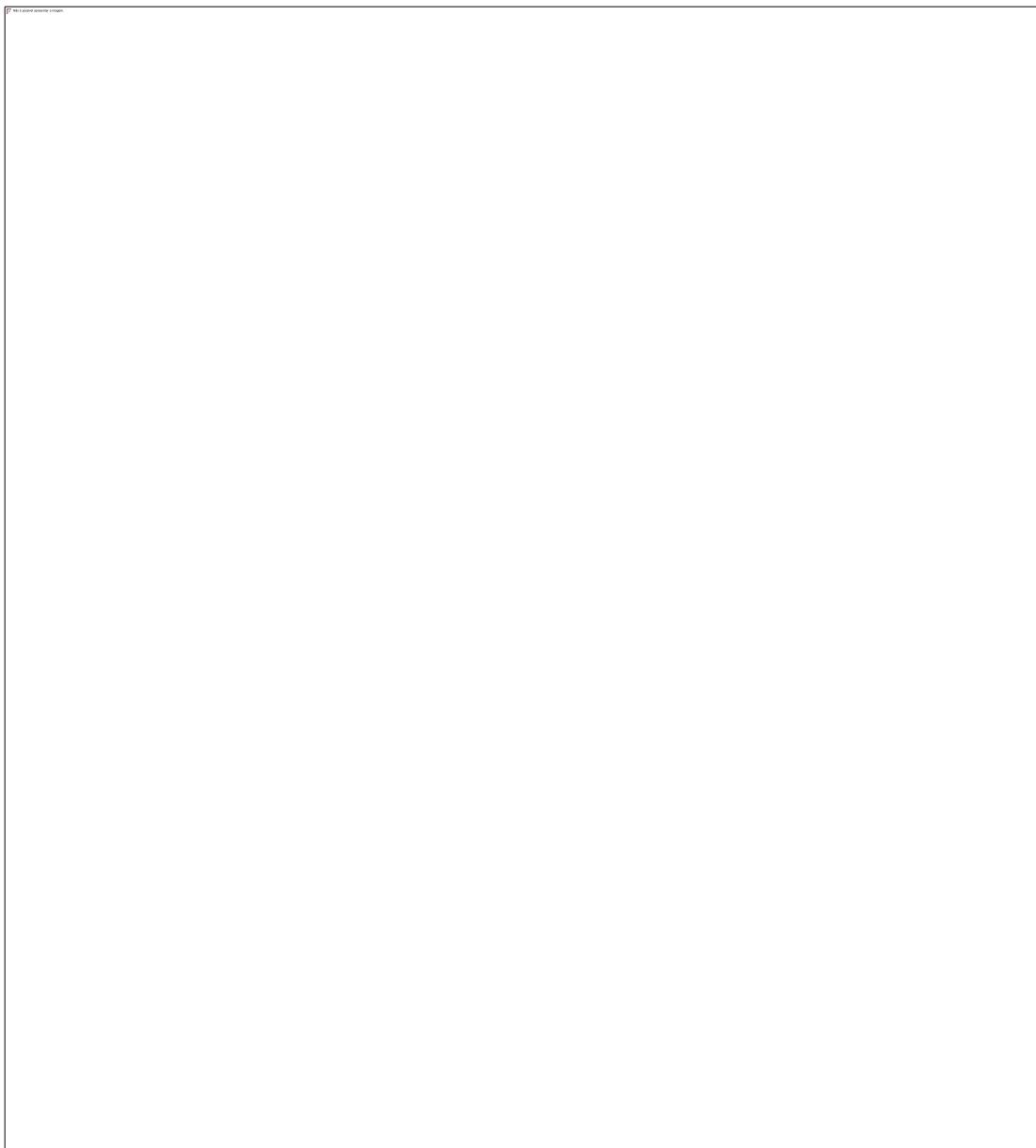
    @Test
    void testDeactivationAfterCompletion()
    {
        movement.setActive(true);

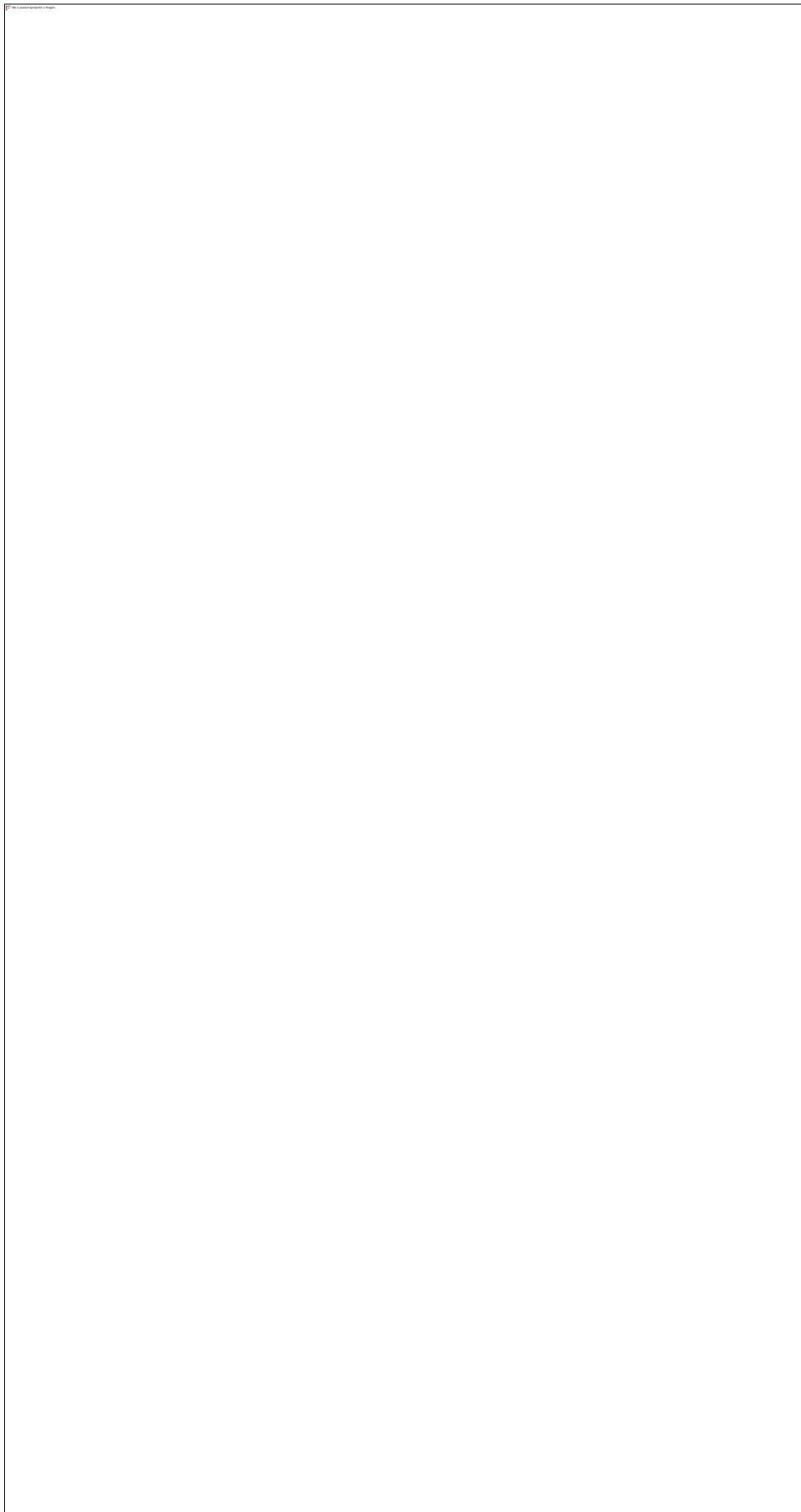
        // Run movement for more than 2 time units
        for(int i = 0; i < 150; i++)
        {
            movement.move(enemy);
            enemy.onUpdate();
        }

        assertFalse(movement.isActive(), "Movement should deactivate after completion");
    }
}

```







```

  @Test
  void testMovementDirection()
  {
    IGameObject result = kamikazeAttack.execute(attacker, target);
    Ponto velocity = ((GameObject)result).velocity();

    // Calculate expected angle to target
    double dx = target.transform().position().x() - attacker.transform().position().x();
    double dy = target.transform().position().y() - attacker.transform().position().y();
    double expectedAngle = Math.atan2(dy, dx);

    // Calculate actual angle from velocity
    double actualAngle = Math.atan2(velocity.y(), velocity.x());

    assertEquals(expectedAngle, actualAngle, 0.001,
                "Velocity direction should match angle to target");
  }

  @Test
  void testAttackAngleAndCollisionFromDifferentPosition()
  {
    Transform attackerTransform = new Transform(new Ponto(4.0, 1.0), 0.180, 1);
    Retangulo attackerRect = new Retangulo(new Ponto[] {
      new Ponto(4.0, 2.0),
      new Ponto(4.0, 0.0),
      new Ponto(0.0, 0.0),
      new Ponto(0.0, 2.0)
    }, attackerTransform);
    attacker = new GameObject("TEST_ATTACKER", attackerTransform, attackerRect, new
    EnemyBehavior(), new Shape());
    attacker.onInit();

    Transform targetTransform = new Transform(new Ponto(20.0, 2.0), 0, 0, 1);
    Retangulo targetRect = new Retangulo(points, targetTransform);
    target = new GameObject("TEST_TARGET", targetTransform, targetRect, new PlayerBehavior(), new
    Shape());
    target.onInit();

    double angle = attacker.transform().angle();
    kamikazeAttack.execute(attacker, target);

    assertNotEquals(angle, target.transform().angle(), 0.0, "Angle should be different");
    for (int i = 0; i < 14; i++)
      attacker.onUpdate();
    assertTrue(attacker.collider().colision(target.collider()));
  }

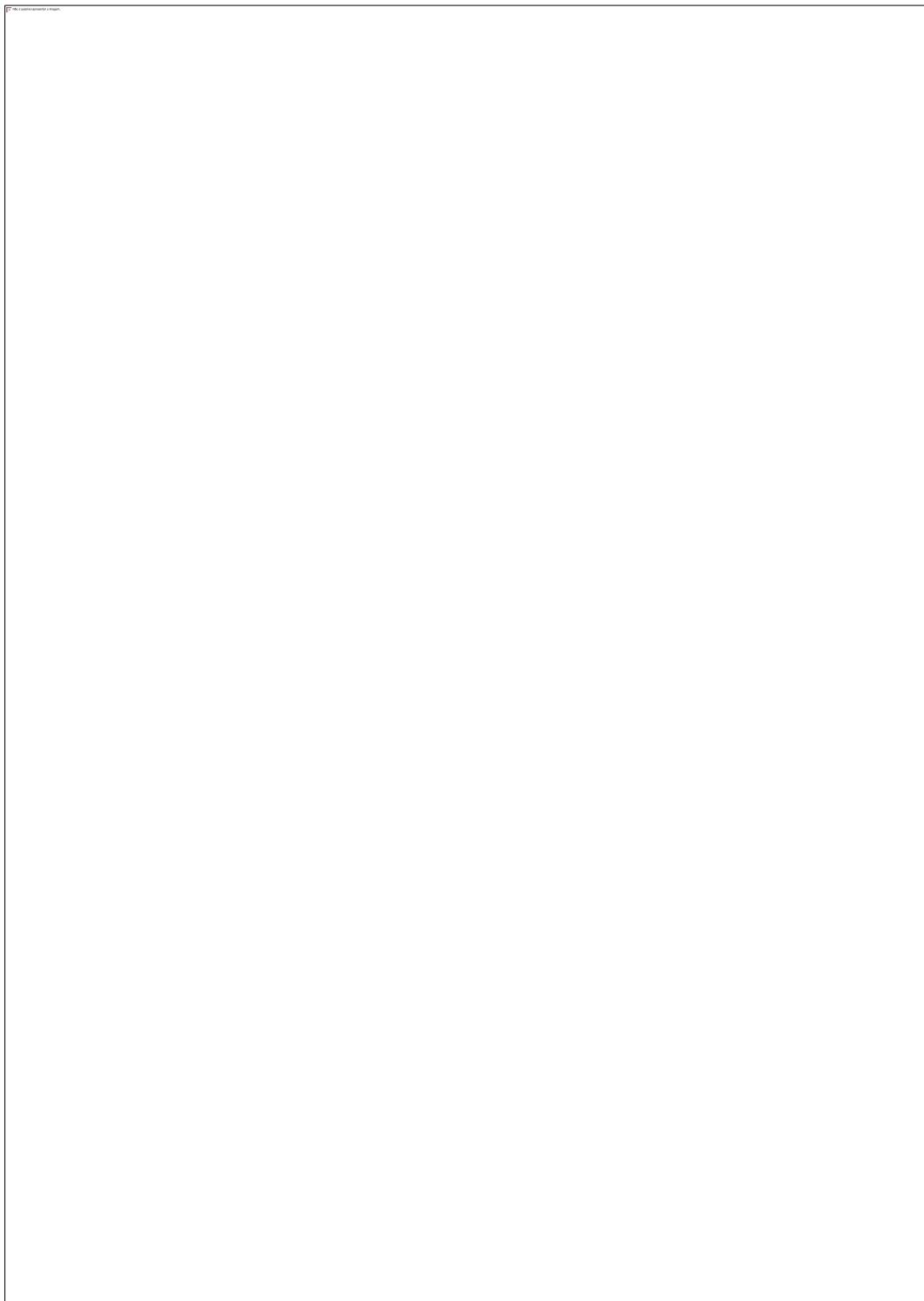
  @Test
  void testAttackFromSameAngle()
  {
    Transform attackerTransform = new Transform(new Ponto(0.0, 6.0), 0.270, 1);
    Retangulo attackerRect = new Retangulo(new Ponto[] {
      new Ponto(4.0, 2.0),
      new Ponto(4.0, 0.0),
      new Ponto(0.0, 0.0),
      new Ponto(0.0, 2.0)
    }, attackerTransform);
    attacker = new GameObject("TEST_ATTACKER", attackerTransform, attackerRect, new EnemyBehavior(),
    new Shape());
    attacker.onInit();

    Transform targetTransform = new Transform(new Ponto(0.0, 0.0), 0, 90, 1);
    Retangulo targetRect = new Retangulo(points, targetTransform);
    target = new GameObject("TEST_TARGET", targetTransform, targetRect, new PlayerBehavior(), new
    Shape());
    target.onInit();

    kamikazeAttack.execute(attacker, target);
    for (int i = 0; i < 3; i++)
      attacker.onUpdate();

    assertTrue(attacker.collider().colision(target.collider()));
    assertEquals(270, attacker.transform().angle());
  }
}

```



```


package test;
import core.*;
import core.objectsinterface.IGameObject;
import geometry.*;
import geometry.Polygon;
import geometry.Retangulo;
import gui.InputEvent;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;
import java.util.ArrayList;
import static org.junit.jupiter.api.Assertions.*;

public class TestPlayerBehavior
{
    private PlayerBehavior playerBehavior0;
    private GameObject gameObject0;

    private PlayerBehavior playerBehavior1;
    private GameObject gameObject1;

    private InputEvent inputEvent;

    @BeforeEach
    void setUp()
    {
        playerBehavior0 = new PlayerBehavior();
        Pontos pts = {new Ponto(2.0, 4), new Ponto(2.0, 0), new Ponto(0.0, 0.0), new Ponto(0.0, 4.0)};
        Transform transform = new Transform(new Ponto(1.0, 2), 0, 0, 1);
        Retangulo polygon = new Retangulo(pts, transform);
        gameObject0 = new GameObject("PLAYER 0", transform, polygon, playerBehavior0, new Shape());
        gameObject0.onInit();

        playerBehavior1 = new PlayerBehavior();
        gameObject1 = new GameObject("PLAYER 1", transform, polygon, playerBehavior1, new Shape());
        gameObject1.onInit();

        inputEvent = new InputEvent();
    }

    @Test
    void attackNotCreatesProjectileWhenNotAttacking()
    {
        IGameObject result = playerBehavior0.attack(inputEvent);
        assertNull(result);
    }

    @Test
    void attackReturnsNullWhenAlreadyAttacking()
    {
        inputEvent.setPlayerMove("MOUSE_LEFT", true);
        IGameObject firstAttack = playerBehavior0.attack(inputEvent);
        assertNotNull(firstAttack);

        IGameObject secondAttack = playerBehavior0.attack(inputEvent);
        assertNull(secondAttack);
    }

    @Test
    void onCollisionReducesLifeByOneWhenHit()
    {
        ArrayList<IGameObject> collisions = new ArrayList<>();
        collisions.add(gameObject1);

        playerBehavior0.onCollision(collisions);
        assertEquals(2, playerBehavior0.life());
    }

    @Test
    void onCollisionMakesPlayerInvincibleTemporarily() throws InterruptedException
    {
        ArrayList<IGameObject> collisions = new ArrayList<>();
        collisions.add(gameObject1);

        playerBehavior0.onCollision(collisions);
        assertTrue(playerBehavior0.isInvincible());
        Thread.sleep(1500);
        assertFalse(playerBehavior0.isInvincible());
    }

    @Test
    void playerRemainsAliveWhenHitWithInvincibility() throws InterruptedException
    {
        ArrayList<IGameObject> collisions = new ArrayList<>();
        collisions.add(gameObject1);

        playerBehavior0.onCollision(collisions); // hit
        playerBehavior0.onCollision(collisions);
        Thread.sleep(1300);
        assertEquals(2, playerBehavior0.life());
        playerBehavior0.onCollision(collisions); // hit
        assertEquals(1, playerBehavior0.life());
        assertEquals(true, playerBehavior0.isEnabled());
    }

    @Test
    void playerIsDestroyedWhenLifeReachesZero() throws InterruptedException
    {
        ArrayList<IGameObject> collisions = new ArrayList<>();
        collisions.add(gameObject1);

        assertEquals(2, playerBehavior0.life());
        playerBehavior0.onCollision(collisions); // hit
        Thread.sleep(1300);
        assertEquals(2, playerBehavior0.life());
        playerBehavior0.onCollision(collisions); // hit
        Thread.sleep(1300);
        assertEquals(1, playerBehavior0.life());
        playerBehavior0.onCollision(collisions); // hit
        Thread.sleep(1300);
        assertEquals(0, playerBehavior0.life());
        assertFalse(playerBehavior0.isEnabled());
    }

    @Test
    void evasiveManeuverMovesPlayerRight()
    {
        inputEvent.setPlayerMove("RIGHT", true);
        Ponto initialPosition = gameObject0.transform().position();
        playerBehavior0.moveAndEvasiveManeuver(inputEvent);

        Ponto newPosition = gameObject0.transform().position();
        assertTrue(newPosition.x() > initialPosition.x());
        assertEquals(initialPosition.y(), newPosition.y());
    }

    @Test
    void evasiveManeuverMovesPlayerLeft() throws InterruptedException
    {
        inputEvent.setPlayerMove("LEFT", true);
        Ponto initialPosition = gameObject0.transform().position();
        playerBehavior0.moveAndEvasiveManeuver(inputEvent);
        assertTrue(playerBehavior0.isInvincible());

        Ponto newPosition = gameObject0.transform().position();
        assertTrue(newPosition.x() < initialPosition.x());
        assertEquals(initialPosition.y(), newPosition.y());
        Thread.sleep(1200);
        assertFalse(playerBehavior0.isInvincible());
    }
}


```