**《软件测试》**

**实验报告六 ——综合测试1**

**姓 名： 胡国昌 学 号： 202210120518**

**院 系： 计算机与信息学院 专 业：计算机科学与技术**

**实 验 室： B119 实验日期： 2025.3.28**

**总评成绩： 审阅教师： 杨青**

### 一、实验目的：

1. 学习在一定规模实际项目中综合测试方法
2. 学习研读开源代码的技术
3. 熟悉项目构建工具gradle或Maven
4. 熟悉静态代码分析工具

### 二、实验要求：

1. 从 gradle 构建JPacman项目，[https://gitee.com/jmhui/jpacman-ctgu](https://gitee.com/jmhui/jpacman-ctgu" \o "https://gitee.com/jmhui/jpacman-ctgu) 仓库克隆代码到自己仓库，导入到自己使用的IDE中,并能够构建运行；也可以使用 Maven构建的JPacman 项目， [https://gitee.com/jmhui/jpackmanframework-ctgu](https://gitee.com/jmhui/jpackmanframework-ctgu.git" \o "https://gitee.com/jmhui/jpackmanframework-ctgu.git)
2. 运行冒烟测试smoketest，在src/default-test/java/nl.tudelft.jpacman/下；
3. 项目有两个测试文件目录，default-test (src/default-test)包含一些已经给出的测试用例样例，test (src/test)目录作为作业编写检查的目录
4. 修改标题：在PacManUI类（nl.tudelft.tudelft.jpacman.ui 包），在构造函数的第一行，即 super(“JPacman”);修改为super(“JPacman <当前年份>”);
5. 完成下面题目1和题目2，提交到自己的代码仓库
6. 编写测试报告，提交到雨课堂“软件测试实验六”

**实验题目**

**题目1**

在DirectionTest中创建测试方法，例如，南、东、西方向。运行测试，并确保测试通过。（注意在设置中配置测试运行器）

**题目2**

程序中留下了三个空的测试方法。你的目标是实现这些测试。

1. noStartSquare()，用来测试一个Unit在没有调用occupy来设置它的Square时，这个Unit不应该占据Square。程序中，一个Unit在最开始时，是没有Square的。
2. testOccupy()，用来验证Unit在占据一个Square后，它就拥有一个目标方格(target square)。换句话说，如果一个Unit占用了一个Square，那么这个Square应该容纳了(contains)一个Unit。
3. 最后，testReoccupy()是在Unit重复占用单元（连续调用两次occupy）后，验证2中的占用关系和容纳关系?
4. 确保所有的测试通过，提交并push你的测试代码。

**代码实现：**

#### Direction.java

package nl.tudelft.jpacman.board;

/\*\*

\* An enumeration of possible directions on a two-dimensional square grid.

\*

\* @author Jeroen Roosen

\*/

public enum Direction {

/\*\*

\* North, or up.

\*/

NORTH(0, -1),

/\*\*

\* South, or down.

\*/

SOUTH(0, 1),

/\*\*

\* West, or left.

\*/

WEST(-1, 0),

/\*\*

\* East, or right.

\*/

EAST(1, 0);

/\*\*

\* The delta x (width difference) to an element in the direction in a grid

\* with 0,0 (x,y) as its top-left element.

\*/

private final int deltaX;

/\*\*

\* The delta y (height difference) to an element in the direction in a grid

\* with 0,0 (x,y) as its top-left element.

\*/

private final int deltaY;

/\*\*

\* Creates a new Direction with the given parameters.

\*

\* @param deltaX

\* The delta x (width difference) to an element in the direction

\* in a matrix with 0,0 (x,y) as its top-left element.

\* @param deltaY

\* The delta y (height difference) to an element in the direction

\* in a matrix with 0,0 (x,y) as its top-left element.

\*/

Direction(int deltaX, int deltaY) {

this.deltaX = deltaX;

this.deltaY = deltaY;

}

/\*\*

\* @return The delta x (width difference) for a single step in this

\* direction, in a matrix with 0,0 (x,y) as its top-left element.

\*/

public int getDeltaX() {

return deltaX;

}

/\*\*

\* @return The delta y (height difference) for a single step in this

\* direction, in a matrix with 0,0 (x,y) as its top-left element.

\*/

public int getDeltaY() {

return deltaY;

}

}

#### DirectionTest.java

package nl.tudelft.jpacman.board;

import org.junit.jupiter.api.Test;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import static org.assertj.core.api.Assertions.assertThat;

/\*\*

\* A very simple (and not particularly useful)

\* test class to have a starting point where to put tests.

\*

\* @author Arie van Deursen

\*/

public class DirectionTest {

/\*\*

\* Do we get the correct delta when moving north?

\*/

private static final Logger LOGGER = LoggerFactory.getLogger(DirectionTest.class);

@Test

void testNorth() {

Direction north = Direction.NORTH;

LOGGER.debug("north getDeltaX:{},north getDeltaY :{}", north.getDeltaX(), north.getDeltaY());

assertThat(north.getDeltaX()).isEqualTo(0);

assertThat(north.getDeltaY()).isEqualTo(-1);

}

@Test

void testSouth() {

Direction south = Direction.SOUTH;

assertThat(south.getDeltaX()).isEqualTo(0);

assertThat(south.getDeltaY()).isEqualTo(1);

}

@Test

void testWest() {

Direction west = Direction.WEST;

assertThat(west.getDeltaX()).isEqualTo(-1);

assertThat(west.getDeltaY()).isEqualTo(0);

}

@Test

void testEast() {

Direction east = Direction.EAST;

assertThat(east.getDeltaX()).isEqualTo(1);

assertThat(east.getDeltaY()).isEqualTo(0);

}

@Test

void testEnumValuesCount() {

Direction[] values = Direction.values();

assertThat(values).hasSize(4); // 确保只有四个方向

}

}

#### DirectionParameterizedTest.java

package nl.tudelft.jpacman.board;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.params.ParameterizedTest;

import org.junit.jupiter.params.provider.Arguments;

import org.junit.jupiter.params.provider.MethodSource;

import java.util.stream.Stream;

import static org.assertj.core.api.Assertions.assertThat;

/\*\*

\* 针对 Direction 枚举的参数化测试类。

\* 验证每个方向的 deltaX 和 deltaY 是否正确，并检查枚举值的完整性。

\*/

public class DirectionParameterizedTest {

/\*\*

\* 参数化测试方法：验证每个 Direction 的 deltaX 和 deltaY 是否符合预期。

\*

\* @param direction 被测试的 Direction 枚举实例

\* @param expectedDeltaX 期望的 x 坐标变化值

\* @param expectedDeltaY 期望的 y 坐标变化值

\*/

@ParameterizedTest

@MethodSource("directionProvider")

// 指定参数来源方法

void testDirection(Direction direction, int expectedDeltaX, int expectedDeltaY) {

// 断言 deltaX 是否符合预期

assertThat(direction.getDeltaX()).isEqualTo(expectedDeltaX);

// 断言 deltaY 是否符合预期

assertThat(direction.getDeltaY()).isEqualTo(expectedDeltaY);

}

/\*\*

\* 提供测试数据的静态方法。

\* 返回一个包含所有 Direction 实例及其预期坐标变化的流。

\*

\* @return Stream<Arguments> 参数流，每个参数对应一个测试用例

\*/

private static Stream < Arguments > directionProvider() {

return Stream.of(

// 每个 Arguments.of 对应一个测试用例：

// 参数1: Direction 实例

// 参数2: 预期 deltaX

// 参数3: 预期 deltaY

Arguments.of(Direction.NORTH, 0, -1), // 北方向：x不变，y减1

Arguments.of(Direction.SOUTH, 0, 1), // 南方向：x不变，y增1

Arguments.of(Direction.WEST, -1, 0), // 西方向：x减1，y不变

Arguments.of(Direction.EAST, 1, 0) // 东方向：x增1，y不变

);

}

/\*\*

\* 测试 Direction 枚举的 values() 方法。

\* 验证枚举值的数量和顺序是否正确。

\*/

@Test

void testEnumValues() {

// 获取所有 Direction 枚举值

Direction[] values = Direction.values();

// 断言枚举值数量为4，且顺序为 NORTH, SOUTH, WEST, EAST

assertThat(values)

.containsExactly(Direction.NORTH, Direction.SOUTH, Direction.WEST, Direction.EAST);

}

}

#### Unit.java

package nl.tudelft.jpacman.board;

import nl.tudelft.jpacman.sprite.Sprite;

/\*\*

\* 游戏板上可放置的单位（如Pacman或幽灵）的抽象基类。

\* 管理单位的位置、方向及移动逻辑。

\*

\* @author Jeroen Roosen

\*/

public abstract class Unit {

/\*\*

\* 当前单位所在的格子。

\*/

private Square square;

/\*\*

\* 单位当前面对的方向（默认向东）。

\*/

private Direction direction;

/\*\*

\* 构造方法，初始化单位方向为东。

\*/

protected Unit() {

this.direction = Direction.EAST;

}

/\*\*

\* 设置单位的新方向。

\* @param newDirection 新方向（不能为null）

\*/

public void setDirection(Direction newDirection) {

this.direction = newDirection;

}

/\*\*

\* 获取当前方向。

\* @return 当前方向实例

\*/

public Direction getDirection() {

return this.direction;

}

/\*\*

\* 获取当前所在格子（调用前需确保单位已放置在格子上）。

\* @return 当前格子对象

\* @throws AssertionError 若未放置在格子上（断言启用时）

\*/

public Square getSquare() {

assert invariant(); // 验证状态一致性

assert square != null; // 确保格子存在

return square;

}

/\*\*

\* 检查单位是否已放置在格子上。

\* @return true 如果格子存在

\*/

public boolean hasSquare() {

return square != null;

}

/\*\*

\* 占据目标格子（需确保目标允许进入）。

\* 先离开原格子（若存在），再占据新格子。

\* @param target 要占据的目标格子（非null）

\*/

public void occupy(Square target) {

assert target != null;

if (square != null) {

square.remove(this); // 从原格子移除

}

square = target; // 更新当前格子

target.put(this); // 添加到目标格子

assert invariant(); // 验证状态一致性

}

/\*\*

\* 离开当前格子，从板上移除。

\*/

public void leaveSquare() {

if (square != null) {

square.remove(this); // 从当前格子移除

square = null; // 清空引用

}

assert invariant();

}

/\*\*

\* 类不变式：验证格子与单位状态的一致性。

\* @return true 如果单位未占据格子，或其所在格子包含该单位

\*/

protected boolean invariant() {

return square == null || square.getOccupants().contains(this);

}

/\*\*

\* 抽象方法，获取单位的精灵（由子类实现）。

\* @return 代表单位外观的精灵对象

\*/

public abstract Sprite getSprite();

/\*\*

\* 计算单位前方指定步数的格子（用于幽灵AI）。

\* 沿当前方向逐步查找，若路径被阻挡可能返回不可达格子。

\* @param amountToLookAhead 向前查看的步数

\* @return 目标格子，可能为null（若路径不完整）

\*/

public Square squaresAheadOf(int amountToLookAhead) {

Direction targetDirection = this.getDirection();

Square destination = this.getSquare();

for (int i = 0; i < amountToLookAhead; i++) {

// 获取当前格子在目标方向的下一个格子

destination = destination.getSquareAt(targetDirection);

}

return destination;

}

}

#### OccupantTest.java

package nl.tudelft.jpacman.board;

import org.junit.jupiter.api.BeforeEach;

import org.junit.jupiter.api.Test;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import static org.assertj.core.api.Assertions.assertThat;

/\*\*

\* 测试 {@link Unit} 类在占据/离开 {@link Square} 时的正确性。

\* 验证单位与方块的交互逻辑是否符合预期。

\*

\* @author Jeroen Roosen

\*/

class OccupantTest {

private static final Logger LOGGER = LoggerFactory.getLogger(OccupantTest.class);

/\*\*

\* 被测试的 Unit 实例（可能为 Pacman 或幽灵的测试实现）。

\*/

private Unit unit;

/\*\*

\* 在每个测试方法执行前重置被测试的 Unit。

\* 使用 @BeforeEach 确保测试独立性。

\*/

@BeforeEach

void setUp() {

unit = new BasicUnit(); // BasicUnit 是 Unit 的简单测试实现

}

/\*\*

\* 测试初始状态：新创建的 Unit 不应关联任何 Square。

\* 验证 hasSquare() 返回 false。

\*/

@Test

void noStartSquare() {

LOGGER.debug("unit isNotNull :{}", unit != null);

assertThat(unit).isNotNull(); // 非空检查

LOGGER.debug("unit hasSquare :{}", unit.hasSquare());

assertThat(unit.hasSquare()).isFalse(); // 初始未占据任何方块

}

/\*\*

\* 测试正常占据逻辑：

\* 1. 调用 occupy() 后，Unit 应关联目标 Square

\* 2. 目标 Square 的占据者列表应包含此 Unit

\*/

@Test

void testOccupy() {

Square target = new BasicSquare(); // 基础 Square 实现

unit.occupy(target);

LOGGER.debug("unit.getSquare: {}", unit.getSquare().equals(target));

// 验证 Unit 的当前方块是同一个实例

assertThat(unit.getSquare()).isSameAs(target);

LOGGER.debug("target.getOccupants: {}", target.getOccupants().contains(unit));

// 验证方块的占用者列表包含该 Unit 实例

assertThat(target.getOccupants()).contains(unit);

}

/\*\*

\* 测试重复占据同一方块的场景：

\* 1. 第一次占据后状态正确

\* 2. 再次占据同一方块时，应保持一致性

\* 3. 方块的占据者列表不应重复添加 Unit

\*/

@Test

void testReoccupy() {

Square square = new BasicSquare();

// 第一次占据

unit.occupy(square);

assertThat(unit.getSquare()).isEqualTo(square);

assertThat(square.getOccupants()).contains(unit);

// 重复占据同一方块

unit.occupy(square);

// 验证方块引用未改变

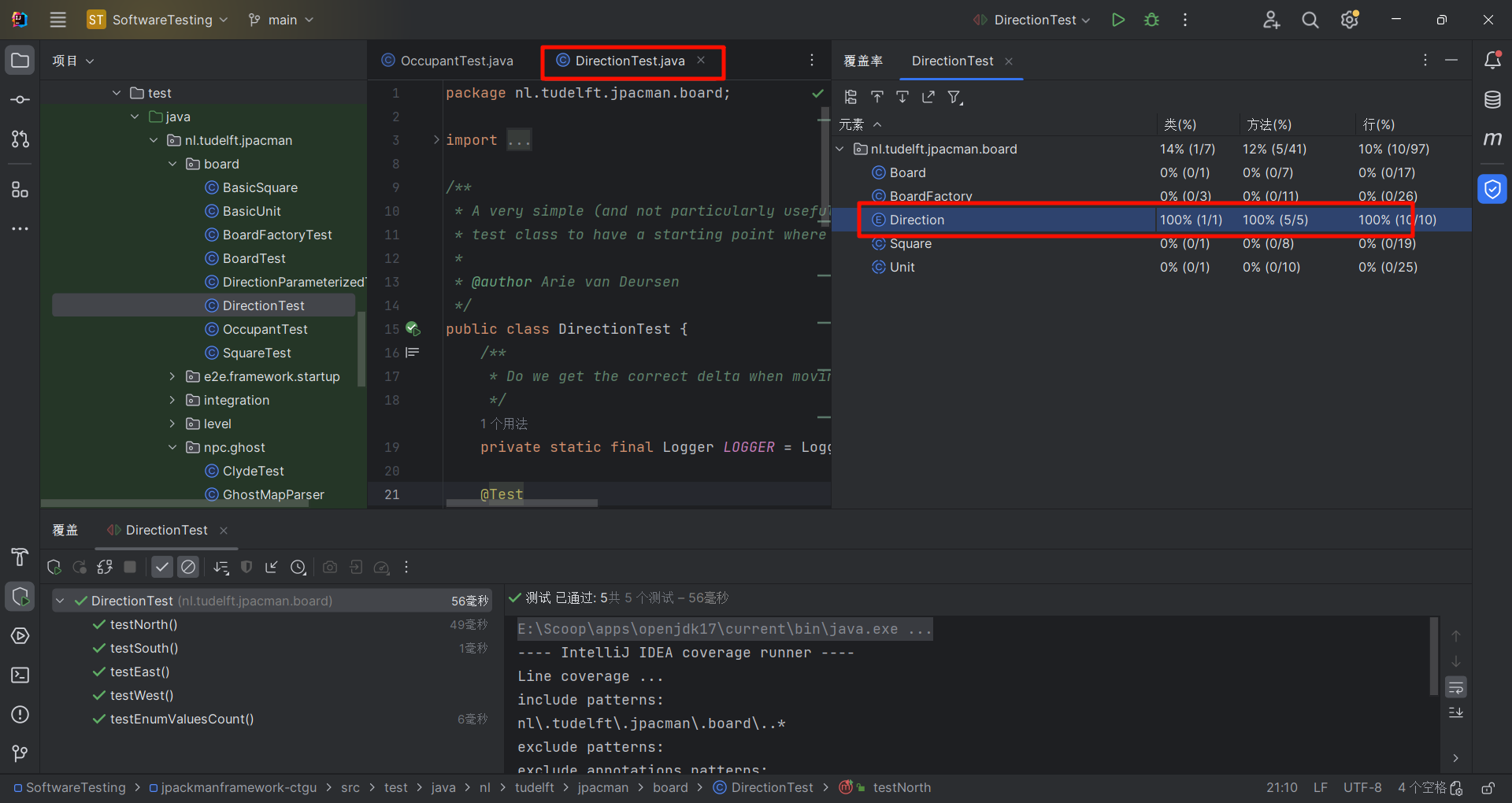
assertThat(unit.getSquare()).isEqualTo(square);

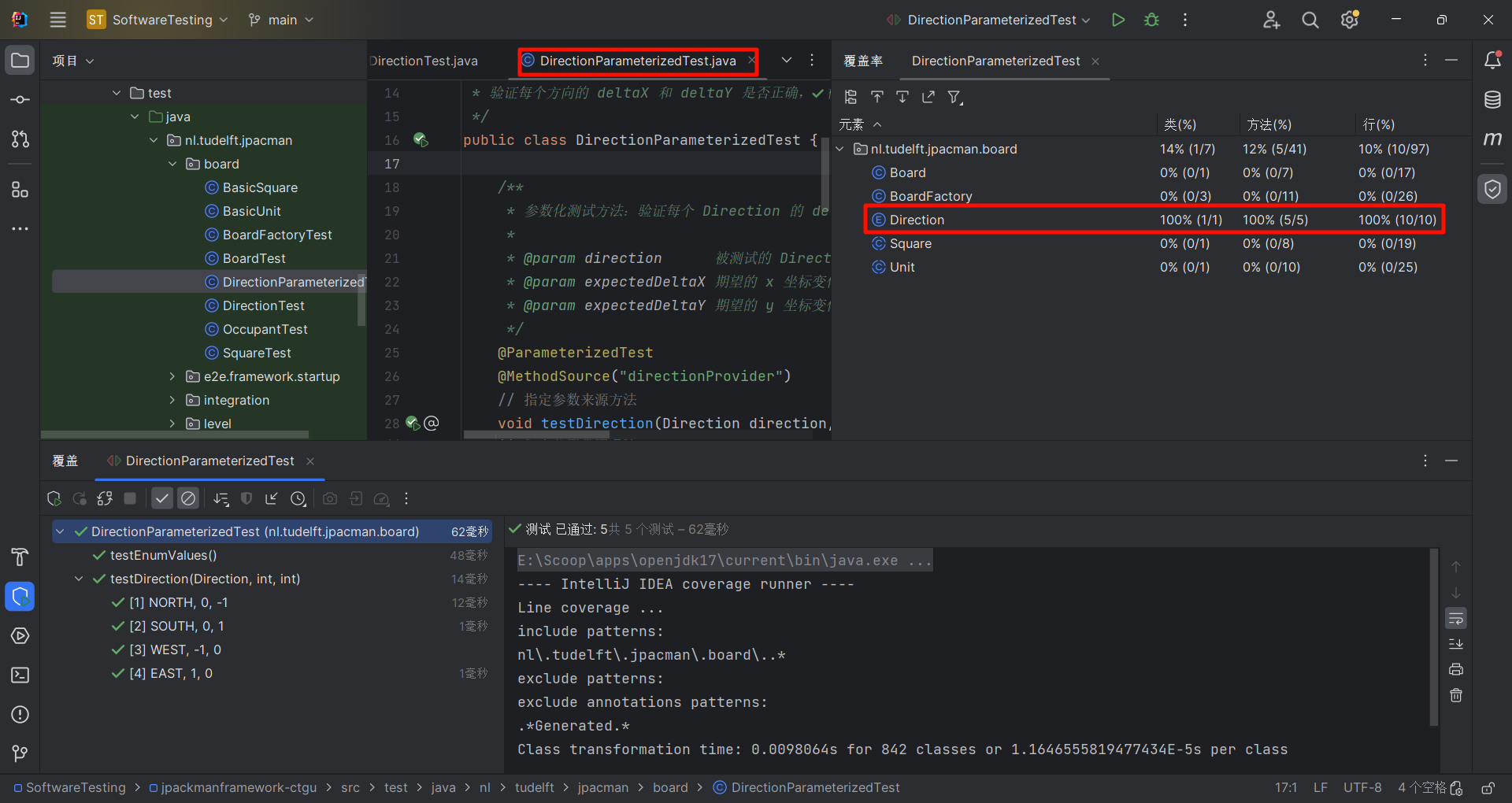
// 验证占据者列表仅包含一次（避免重复添加）

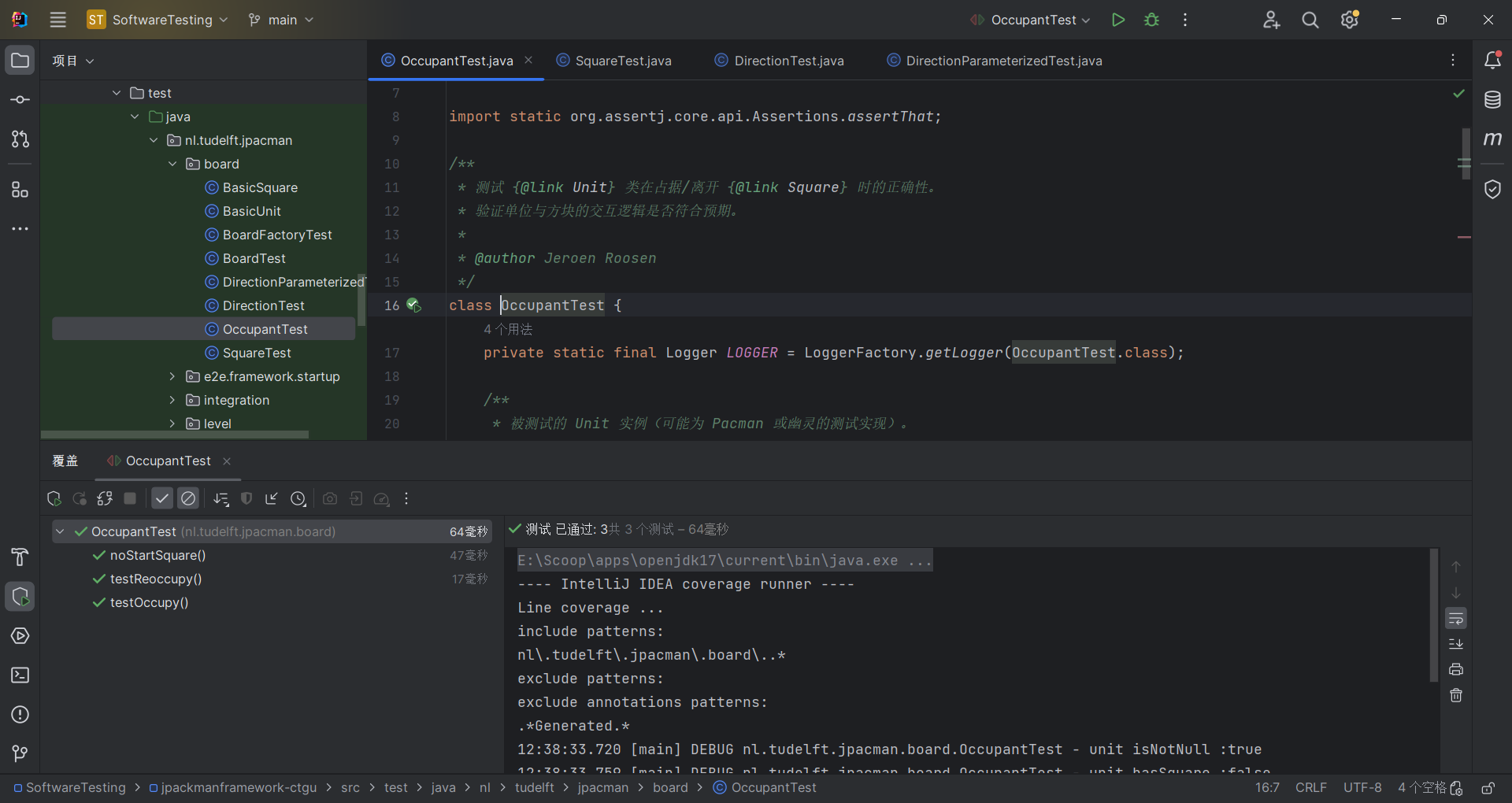
assertThat(square.getOccupants()).containsExactly(unit);

}

}







### 六、仓库地址

https://github.com/xieyangyuyue/SoftwareTesting

[xieyangyuyue/SoftwareTesting: 软件测试--java (github.com)](https://github.com/xieyangyuyue/SoftwareTesting)