**《软件测试》**

**实验报告九 ——综合测试4**

**姓 名： 黄勇斌 学 号：2019112321**

**院 系： 计算机与信息学院 专 业： 计算机科学与技术**

**实 验 室： 实验日期：2022/5/29**

**总评成绩： 审阅教师：**

目录

[**《软件测试》** 1](#_Toc104137972)

[**实验报告八 ——综合测试3** 1](#_Toc104137973)

[一、实验目的： 3](#_Toc104137974)

[二、实验环境： 3](#_Toc104137975)

[三、实验要求： 3](#_Toc104137976)

[四、实验步骤与内容 3](#_Toc104137977)

[五、结论分析与体会 12](#_Toc104137978)

[六、仓库地址 12](#_Toc104137979)

### 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
4. 熟悉静态代码分析工具

### 二、实验环境：

Eclipse2020,JUnit

### 三、实验要求：

1. 在实验六的代码基础上， 完成下面要求，提交到自己的代码仓库

分析需求并绘制决策表，设计测试用例并实现

编写测试报告，提交到雨课堂“软件测试实验九”



2.编写测试类。

基于碰撞的决策表，对level.PlayerCollisions设计测试类。不仅要考虑产生某种结果的碰撞，还要考虑“什么都没发生”的碰撞。

提示：该类中应尽可能采用mock方法测试。

3.静态代码检查测试代码

确保你的代码通过CheckStyle的检查，正确完成javadoc的注释。

### 四、实验步骤与内容

1. **在实验六的代码基础上， 完成下面要求，提交到自己的代码仓库**
2. **分析需求并绘制决策表，设计测试用例并实现**
3. **编写测试报告，提交到雨课堂“软件测试实验九”**

**测试用例：**

|  |  |  |  |
| --- | --- | --- | --- |
|  | 规则1 | 规则2 | 规则3 |
| Collider | Player | Player | Ghost |
| Collidee | Ghost | Pellet | Player |
| Consequence | Player 被Ghost 杀死 | Pellet从方格上消失 | Player 被Ghost 杀死 |

|  |  |  |  |
| --- | --- | --- | --- |
|  | 规则4 | 规则5 | 规则6 |
| Collider | Pellet | Ghost | Ghost |
| Collidee | Player | Ghost | Pellet |
| Consequence | Pellet从方格上消失 | 什么都没发生 | 什么都没发生 |

|  |  |
| --- | --- |
|  | 规则7 |
| Collider | Pellet |
| Collidee | Ghost |
| Consequence | 什么都没发生 |

1. **代码实现：**

|  |
| --- |
| **测试代码：**  **level.PlayerCollisions 类：**  package nl.tudelft.jpacman.level;  import nl.tudelft.jpacman.board.Unit;  import nl.tudelft.jpacman.npc.Ghost;  import nl.tudelft.jpacman.points.PointCalculator;  /\*\*  \* A simple implementation of a collision map for the JPacman player.  \* <p>  \* It uses a number of instanceof checks to implement the multiple dispatch for the  \* collisionmap. For more realistic collision maps, this approach will not scale,  \* and the recommended approach is to use a {@link CollisionInteractionMap}.  \*  \* @author Arie van Deursen, 2014  \*  \*/  public class PlayerCollisions implements CollisionMap {  private PointCalculator pointCalculator;  /\*\*  \* Create a simple player-based collision map, informing the  \* point calculator about points to be added.  \*  \* @param pointCalculator  \* Strategy for calculating points.  \*/  public PlayerCollisions(PointCalculator pointCalculator) {  this.pointCalculator = pointCalculator;  }  @Override  public void collide(Unit mover, Unit collidedOn) {  if (mover instanceof Player) {  playerColliding((Player) mover, collidedOn);  }  else if (mover instanceof Ghost) {  ghostColliding((Ghost) mover, collidedOn);  }  else if (mover instanceof Pellet) {  pelletColliding((Pellet) mover, collidedOn);  }  }  private void playerColliding(Player player, Unit collidedOn) {  if (collidedOn instanceof Ghost) {  playerVersusGhost(player, (Ghost) collidedOn);  }  if (collidedOn instanceof Pellet) {  playerVersusPellet(player, (Pellet) collidedOn);  }  }  private void ghostColliding(Ghost ghost, Unit collidedOn) {  if (collidedOn instanceof Player) {  playerVersusGhost((Player) collidedOn, ghost);  }  }  private void pelletColliding(Pellet pellet, Unit collidedOn) {  if (collidedOn instanceof Player) {  playerVersusPellet((Player) collidedOn, pellet);  }  }  /\*\*  \* Actual case of player bumping into ghost or vice versa.  \*  \* @param player  \* The player involved in the collision.  \* @param ghost  \* The ghost involved in the collision.  \*/  public void playerVersusGhost(Player player, Ghost ghost) {  pointCalculator.collidedWithAGhost(player, ghost);  player.setAlive(false);  player.setKiller(ghost);  }  /\*\*  \* Actual case of player consuming a pellet.  \*  \* @param player  \* The player involved in the collision.  \* @param pellet  \* The pellet involved in the collision.  \*/  public void playerVersusPellet(Player player, Pellet pellet) {  pointCalculator.consumedAPellet(player, pellet);  pellet.leaveSquare();  }  } }  **level.PlayerCollisionsTest类：**  package nl.tudelft.jpacman.level;  import nl.tudelft.jpacman.board.Direction;  import nl.tudelft.jpacman.board.Unit;  import nl.tudelft.jpacman.npc.Ghost;  import nl.tudelft.jpacman.points.PointCalculator;  import org.junit.jupiter.api.\*;  import org.mockito.Mock;  import static org.junit.jupiter.api.Assertions.\*;  import static org.mockito.Mockito.\*;  class PlayerCollisionsTest {  PlayerCollisions playerCollisions;  @BeforeEach  void setUp(){  playerCollisions =new PlayerCollisions(new PointCalculator() {  @Override  public void consumedAPellet(Player player, Pellet pellet) {  }  @Override  public void pacmanMoved(Player player, Direction direction) {  }  @Override  public void collidedWithAGhost(Player player, Ghost ghost) {  }  });  }  @Test  void nothingHappened(){  Player player = mock(Player.class);  Ghost ghost = mock(Ghost.class);  Pellet pellet = mock(Pellet.class);  // 规则5  playerCollisions.collide(ghost,ghost);  // 规则6  playerCollisions.collide(ghost,pellet);  // 规则7  playerCollisions.collide(pellet,ghost);  doCallRealMethod().when(player).setAlive(anyBoolean());  doCallRealMethod().when(player).setKiller(any(Unit.class));  doNothing().when(pellet).leaveSquare();  verify(player,times(0)).setAlive(false);  verify(player,times(0)).setKiller(ghost);  verify(pellet,times(0)).leaveSquare();  }  // 规则1  @Test  void playerVersusGhost(){  Player player = mock(Player.class);  Ghost ghost = mock(Ghost.class);  playerCollisions.collide(player,ghost);  doCallRealMethod().when(player).setAlive(anyBoolean());  doCallRealMethod().when(player).setKiller(any(Unit.class));  verify(player,times(1)).setAlive(false);  verify(player,times(1)).setKiller(ghost);  }  // 规则2  @Test  void playerVersusPellet(){  Player player = mock(Player.class);  Pellet pellet = mock(Pellet.class);  playerCollisions.collide(player,pellet);  doNothing().when(pellet).leaveSquare();  verify(pellet,times(1)).leaveSquare();  }  // 规则3  @Test  void ghostVersusPlayer(){  Player player = mock(Player.class);  Ghost ghost = mock(Ghost.class);  playerCollisions.collide(ghost,player);  doCallRealMethod().when(player).setAlive(anyBoolean());  doCallRealMethod().when(player).setKiller(any(Unit.class));  verify(player,times(1)).setAlive(false);  verify(player,times(1)).setKiller(ghost);  }  // 规则4  @Test  void pelletVersusPlayer(){  Player player = mock(Player.class);  Pellet pellet = mock(Pellet.class);  playerCollisions.collide(pellet,player);  doNothing().when(pellet).leaveSquare();  verify(pellet,times(1)).leaveSquare();  }  } 测试结果：    Jacco    Checkstyle |

### 五、结论分析与体会

Mock感觉比较实用简单一点，大作业就做这个方向好了。

### 六、仓库地址

<https://github.com/hyb1041739742/Software-Testing>