**《软件测试》**

**实验报告八 ——综合测试3**

**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：2022/5/22**

**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. 针对 level.MapParser 类， 首先从阳光行为开始考虑，棋盘文件包含期望的字符。使用 Mockito 模拟工厂方法，使用 Mockito 验证读取的地图与工厂具有正确的交互。
2. 扩展测试用例考虑异常错误场景，强制抛出正确的异常。很容易考虑到异常情景，但是可能有更重要的情况需要考虑，比如从文件中读取数据的时候可能发生的错误。
3. 实现测试用例，并执行覆盖率检查和代码规范检查。

### 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. **使用模拟框架，设计测试用例并实现**
3. **编写测试报告，提交到雨课堂“软件测试实验八”**

**测试用例：**

|  |  |
| --- | --- |
| 测试样例标号 | 读取文件 |
| 1 | 空文件 |
| 2 | 无法识别的charmap |
| 3 | 存在的文件 |
| 4 | 不存在的文件 |
| 5 | null文件名 |

1. **代码实现：**

|  |
| --- |
| **测试代码：**  **MapPaerse类：**  package nl.tudelft.jpacman.level;  import java.io.BufferedReader;  import java.io.IOException;  import java.io.InputStream;  import java.io.InputStreamReader;  import java.util.ArrayList;  import java.util.List;  import nl.tudelft.jpacman.PacmanConfigurationException;  import nl.tudelft.jpacman.board.Board;  import nl.tudelft.jpacman.board.BoardFactory;  import nl.tudelft.jpacman.board.Square;  import nl.tudelft.jpacman.npc.Ghost;  import edu.umd.cs.findbugs.annotations.SuppressFBWarnings;  /\*\*  \* Creates new {@link Level}s from text representations.  \*  \* @author Jeroen Roosen  \*/  public class MapParser {  /\*\*  \* The factory that creates the levels.  \*/  private final LevelFactory levelCreator;  /\*\*  \* The factory that creates the squares and board.  \*/  private final BoardFactory boardCreator;  /\*\*  \* Creates a new map parser.  \*  \* @param levelFactory  \* The factory providing the NPC objects and the level.  \* @param boardFactory  \* The factory providing the Square objects and the board.  \*/  public MapParser(LevelFactory levelFactory, BoardFactory boardFactory) {  this.levelCreator = levelFactory;  this.boardCreator = boardFactory;  }  /\*\*  \* Parses the text representation of the board into an actual level.  \*  \* <ul>  \* <li>Supported characters:  \* <li>' ' (space) an empty square.  \* <li>'#' (bracket) a wall.  \* <li>'.' (period) a square with a pellet.  \* <li>'P' (capital P) a starting square for players.  \* <li>'G' (capital G) a square with a ghost.  \* </ul>  \*  \* @param map  \* The text representation of the board, with map[x][y]  \* representing the square at position x,y.  \* @return The level as represented by this text.  \*/  public Level parseMap(char[][] map) {  int width = map.length;  int height = map[0].length;  Square[][] grid = new Square[width][height];  List<Ghost> ghosts = new ArrayList<>();  List<Square> startPositions = new ArrayList<>();  makeGrid(map, width, height, grid, ghosts, startPositions);  Board board = boardCreator.createBoard(grid);  return levelCreator.createLevel(board, ghosts, startPositions);  }  private void makeGrid(char[][] map, int width, int height,  Square[][] grid, List<Ghost> ghosts, List<Square> startPositions) {  for (int x = 0; x < width; x++) {  for (int y = 0; y < height; y++) {  char c = map[x][y];  addSquare(grid, ghosts, startPositions, x, y, c);  }  }  }  /\*\*  \* Adds a square to the grid based on a given character. These  \* character come from the map files and describe the type  \* of square.  \*  \* @param grid  \* The grid of squares with board[x][y] being the  \* square at column x, row y.  \* @param ghosts  \* List of all ghosts that were added to the map.  \* @param startPositions  \* List of all start positions that were added  \* to the map.  \* @param x  \* x coordinate of the square.  \* @param y  \* y coordinate of the square.  \* @param c  \* Character describing the square type.  \*/  protected void addSquare(Square[][] grid, List<Ghost> ghosts,  List<Square> startPositions, int x, int y, char c) {  switch (c) {  case ' ':  grid[x][y] = boardCreator.createGround();  break;  case '#':  grid[x][y] = boardCreator.createWall();  break;  case '.':  Square pelletSquare = boardCreator.createGround();  grid[x][y] = pelletSquare;  levelCreator.createPellet().occupy(pelletSquare);  break;  case 'G':  Square ghostSquare = makeGhostSquare(ghosts, levelCreator.createGhost());  grid[x][y] = ghostSquare;  break;  case 'P':  Square playerSquare = boardCreator.createGround();  grid[x][y] = playerSquare;  startPositions.add(playerSquare);  break;  default:  throw new PacmanConfigurationException("Invalid character at "  + x + "," + y + ": " + c);  }  }  /\*\*  \* creates a Square with the specified ghost on it  \* and appends the placed ghost into the ghost list.  \*  \* @param ghosts all the ghosts in the level so far, the new ghost will be appended  \* @param ghost the newly created ghost to be placed  \* @return a square with the ghost on it.  \*/  protected Square makeGhostSquare(List<Ghost> ghosts, Ghost ghost) {  Square ghostSquare = boardCreator.createGround();  ghosts.add(ghost);  ghost.occupy(ghostSquare);  return ghostSquare;  }  /\*\*  \* Parses the list of strings into a 2-dimensional character array and  \* passes it on to {@link #parseMap(char[][])}.  \*  \* @param text  \* The plain text, with every entry in the list being a equally  \* sized row of squares on the board and the first element being  \* the top row.  \* @return The level as represented by the text.  \* @throws PacmanConfigurationException If text lines are not properly formatted.  \*/  public Level parseMap(List<String> text) {  checkMapFormat(text);  int height = text.size();  int width = text.get(0).length();  char[][] map = new char[width][height];  for (int x = 0; x < width; x++) {  for (int y = 0; y < height; y++) {  map[x][y] = text.get(y).charAt(x);  }  }  return parseMap(map);  }  /\*\*  \* Check the correctness of the map lines in the text.  \* @param text Map to be checked  \* @throws PacmanConfigurationException if map is not OK.  \*/  private void checkMapFormat(List<String> text) {  if (text == null) {  throw new PacmanConfigurationException(  "Input text cannot be null.");  }  if (text.isEmpty()) {  throw new PacmanConfigurationException(  "Input text must consist of at least 1 row.");  }  int width = text.get(0).length();  if (width == 0) {  throw new PacmanConfigurationException(  "Input text lines cannot be empty.");  }  for (String line : text) {  if (line.length() != width) {  throw new PacmanConfigurationException(  "Input text lines are not of equal width.");  }  }  }  /\*\*  \* Parses the provided input stream as a character stream and passes it  \* result to {@link #parseMap(List)}.  \*  \* @param source  \* The input stream that will be read.  \* @return The parsed level as represented by the text on the input stream.  \* @throws IOException  \* when the source could not be read.  \*/  public Level parseMap(InputStream source) throws IOException {  try (BufferedReader reader = new BufferedReader(new InputStreamReader(  source, "UTF-8"))) {  List<String> lines = new ArrayList<>();  while (reader.ready()) {  lines.add(reader.readLine());  }  return parseMap(lines);  }  }  /\*\*  \* Parses the provided input stream as a character stream and passes it  \* result to {@link #parseMap(List)}.  \*  \* @param mapName  \* Name of a resource that will be read.  \* @return The parsed level as represented by the text on the input stream.  \* @throws IOException  \* when the resource could not be read.  \*/  @SuppressFBWarnings(  value = {"OBL\_UNSATISFIED\_OBLIGATION", "RCN\_REDUNDANT\_NULLCHECK\_OF\_NONNULL\_VALUE"},  justification = "try with resources always cleans up / false positive in java 11"  )  public Level parseMap(String mapName) throws IOException {  try (InputStream boardStream = MapParser.class.getResourceAsStream(mapName)) {  if (boardStream == null) {  throw new PacmanConfigurationException("Could not get resource for: " + mapName);  }  return parseMap(boardStream);  }  }  /\*\*  \* @return the BoardCreator  \*/  protected BoardFactory getBoardCreator() {  return boardCreator;  }  }**MapPaserTest类：**  package nl.tudelft.jpacman.level;  import nl.tudelft.jpacman.PacmanConfigurationException;  import nl.tudelft.jpacman.board.BoardFactory;  import nl.tudelft.jpacman.board.Square;  import nl.tudelft.jpacman.npc.Ghost;  import org.junit.jupiter.api.\*;  import org.junit.jupiter.api.io.TempDir;  import java.io.IOException;  import static org.junit.jupiter.api.Assertions.\*;  import static org.mockito.Mockito.mock;  import static org.mockito.Mockito.when;  @TestMethodOrder(MethodOrderer.OrderAnnotation.class)  class MapParserTest {  private MapParser mapParser;  private final LevelFactory levelCreator = mock(LevelFactory.class);  private final BoardFactory boardFactory = mock(BoardFactory.class);  @BeforeEach  void setup(){  mapParser = new MapParser(levelCreator,boardFactory);  when(boardFactory.createGround()).thenReturn(mock(Square.class));  when(boardFactory.createWall()).thenReturn(mock(Square.class));  when(levelCreator.createGhost()).thenReturn(mock(Ghost.class));  when(levelCreator.createPellet()).thenReturn(mock(Pellet.class));  }  @Test  @Order(1)  @DisplayName("null文件名")  void nullFile(){  assertThrows(NullPointerException.class,() ->{  mapParser.parseMap((String)null);  });  }  @Test  @Order(2)  @DisplayName("读取不存在的文件")  void nonExistFile(){  String file = "/error.txt";  String message = "Could not get resource for: " + file;  assertThrows(PacmanConfigurationException.class,() ->{  mapParser.parseMap(file);  },message);  }  @Test  @Order(3)  @DisplayName("读取存在的文件")  void existFile() throws IOException {  String file = "/simplemap.txt";  assertEquals(null,mapParser.parseMap(file));  }  @Test  @Order(4)  @DisplayName("读取无法识别的charmap")  void unrecongnizedCharMap() throws IOException {  String file = "/unrecognizedcharmap.txt";  assertThrows(PacmanConfigurationException.class,() ->{  mapParser.parseMap(file);  });  }  @Test  @Order(5)  @DisplayName("读取空文件")  void empty() throws IOException {  String file = "/empty.txt";  assertThrows(PacmanConfigurationException.class,() ->{  mapParser.parseMap(file);  },"Input text must consist of at least 1 row");  }  }测试结果：    Jacco |

### 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>