Assignment 4

Osama Zarraa

oz222am@student.lnu.se

https://github.com/osamaz26/1dv600

Linneuniversitet

Date
2019-04-18

Content

1. Test plan

1.1 Introduction

The Test Plan has been created to test hangman game project. It includes the objectives, what we are going to test and way. Also, it includes how to test and the time plan.

1.2 Motivation

The reason for selecting these tests is to check the functionality of the new features (Login, Register Player, List of Players) are working and give the expected results. It contains 6 manual test cases and testing for 2 classes as automated tests. The manual tests check that use case scenarios are valid. While the automated tests check the code for errors and coverage.

1.3 Objectives

The objective is to test part of the code that was implemented the last iteration.

1.4 Team Members

Resource Name	**Role**
Osama Zarraa	Tester

1.5 What to test and why

It is intended to test use cases (1,2), Login use case, Register use case and List of players use case by writing and running dynamic manual test-cases. The reason for testing these use cases is to test the fundamental functionalities of the project.

1.6 Scope

To test the project, manual and automated tests are made and saved.

1.7 Time plan

Task	Estimated	Actual
Manual Test	1h	1h
Unit tests	2h	1.5h
Running manual tests	15m	8m
Code inspection	30m	30m
Test plan	1h	2h

1.8 Risks

#	Risk	Impact	Effect	Mitigation Plan
1	Working alone	High	Delays in implementation during the time frame	Have and follow a strict plan
2	Hardware damage	Low	Loss the project	Upload the project to cloud service
3	Sickness	Medium	Product did not get delivered on schedule	None

2. Manual Test

2.1 TC1

This is the manual test for the use case 1 start game to ensure that game starts without an issue.

ID	**TC1**
Requirement / Use Case Coverage	UC1
Name	Start game successfully
Precondition	• Java 11 is installed • The words file is existed • The players file is existed
Test Steps	1. Pass path of the file, that contains a list of words which will be used in the game, as an argument to the program. 2. Pass path of the file, that contains a list of players and their scores which will be used in the game, as an argument to the program. 3. Start the application 4. System shows main menu 5. Enter 1 to start the game
Test data	Path for the existed words file Path for the existed players file
Expected	The system should show the text “# Game started (Press * if you would like to interrupt the game)”
Actual	As expected
Status	Pass
Date	20190308
Comment	

2.1 TC2

This is the manual test for the Login use case to ensure that player can login without an issue.

ID	**TC2**
Requirement / Use Case Coverage	Login Use Case
Name	Login successfully
Precondition	• Java 11 is installed • The words file is existed • The players file is exited
Test Steps	1. Pass path of the file, that contains a list of words which will be used in the game, as an argument to the program. 2. Pass path of the file, that contains a list of players and their scores which will be used in the game, as an argument to the program. 3. Run the application 4. System shows main menu 5. Enter 2 to login in 6. Write a correct player name that exists in the players data
Test data	Path for the existed words file Path for the existed players file
Expected	The system should show the text "# What would you like to do (PlayerName)" where PlayerName is the entered name of the player
Actual	As expected
Status	Pass
Date	20190418
Comment	

2.2 TC3

This is the manual test for the Login use case to check if system handles the player not found.

ID	**TC3**
Requirement / Use Case Coverage	Login Use Case
Name	Login failed
Precondition	• Java 11 is installed • The words file is existed • The players file is exited
Test Steps	1. Pass path of the file, that contains a list of words which will be used in the game, as an argument to the program. 2. Pass path of the file, that contains a list of players and their scores which will be used in the game, as an argument to the program. 3. Run the application 4. System shows main menu 5. Enter 2 to login in 6. Write an incorrect player name that is not existed in the players data
Test data	Path for the non-existed words file
Expected	The system should print an error message like “sorry, invalid name”
Actual	As expected
Status	Pass
Date	20190418
Comment	

2.3 TC4

This is the manual test for the use case 2 play game with failing due to the time limit to ensure that game has a time limit scenario.

ID	**TC4**
Requirement / Use Case Coverage	UC2
Name	Play game and lose due to time limit
Precondition	TC1
Test data	Characters
Test Steps	1. User enters one invalid character after 3 minutes from starting the game
Expected	The system should show the text “# You lost”
Actual	As expected
Status	Pass
Date	20190418
Comment	-

2.4 TC5

This is the manual test for the register player use case to ensure that register player scenario.

ID	**TC5**
Requirement / Use Case Coverage	Register Player Use Case
Name	Register player successfully
Precondition	• Java 11 is installed • The words file is existed • The players file is exited
Test data	Path for the existed players file
Test Steps	1. Pass path of the file, that contains a list of words which will be used in the game, as an argument to the program. 2. Pass path of the file, that contains a list of players and their scores which will be used in the game, as an argument to the program. 3. Run the application 4. System shows main menu 5. Enter 3 to enter to players menu 6. Enter 1 to start registering new player 7. Enter non-empty name for the player
Expected	The program shows the players menu again without any error messages
Actual	As expected.
Status	Pass
Date	20190418
Comment	-

2.5 TC6

This is the manual test for the display players use case to ensure that game has a display players with scores functionality.

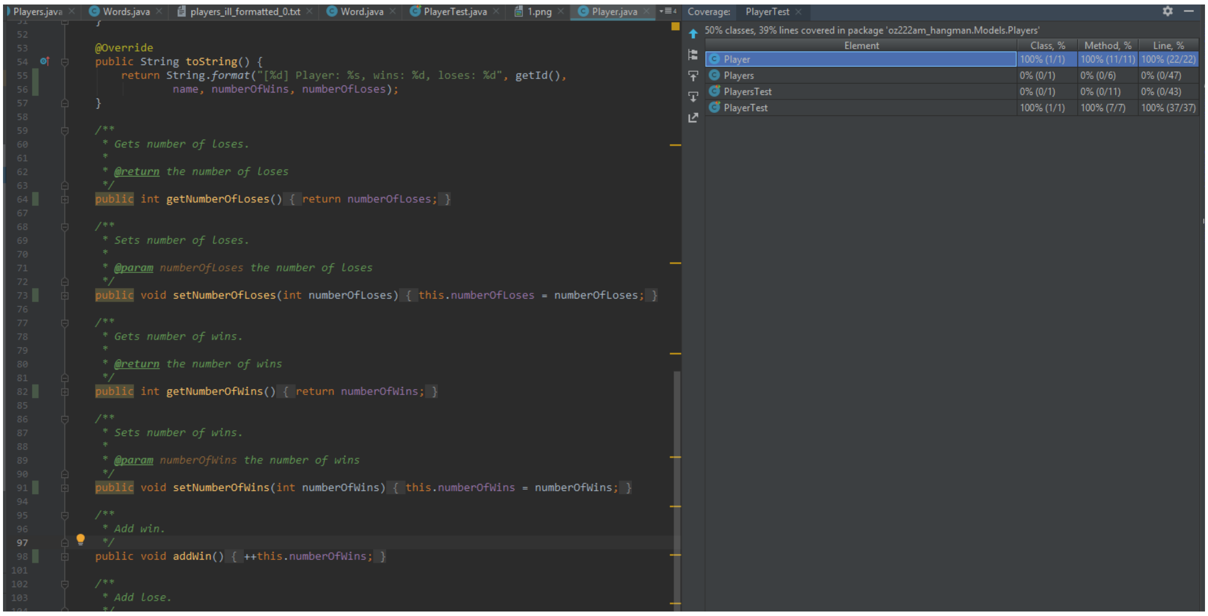
ID	**TC6**
Requirement / Use Case Coverage	List of Players Use Case
Name	Display players with scores
Precondition	• Java 11 is installed • The words file is existed • The players file is exited
Test Steps	1. Pass path of the file, that contains a list of words which will be used in the game, as an argument to the program. 2. Pass path of the file, that contains a list of players and their scores which will be used in the game, as an argument to the program. 3. Run the application 4. System shows main menu 5. Enter 3 to enter to players menu 6. Enter 3 to show players with scores
Expected	System should show players with their scores
Actual	As expected
Status	Pass
Date	20190418
Comment	-

3. Automated Test

3.1 Player class

The screenshot below shows “PlayerTest” class with six test methods.

As the screen shows, all methods passed the test in the word class. Furthermore, it shows the coverage on the left side. This test is made to verify that new implemented methods (setNumberOfWin, setNumberOfLose, addWin, AddLose) are working.



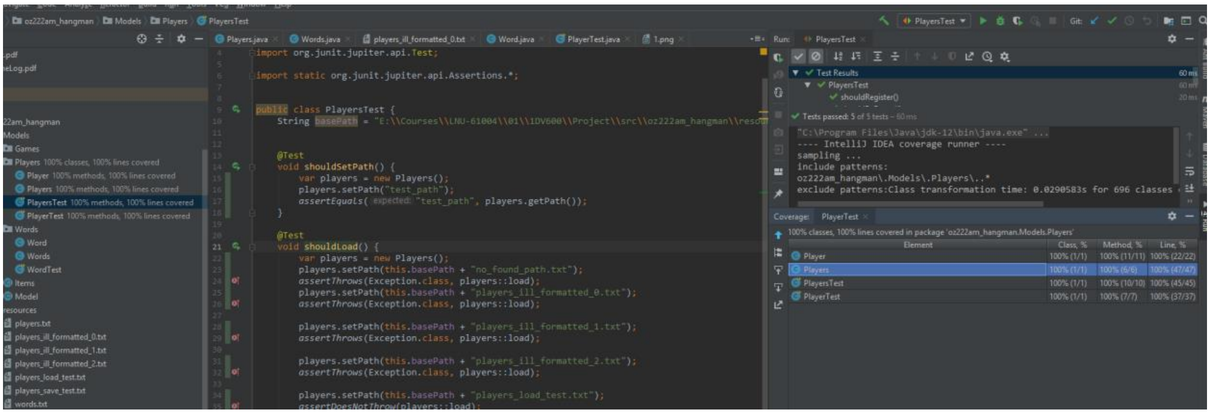
```java
    @Override
    public String toString() {
        return String.format("[%d] Player: %s, wins: %d, loses: %d", getId(),
                name, numberOfWins, numberOfLoses);
    }

    /**
     * Gets number of loses.
     *
     * @return the number of loses
     */
    public int getNumberOfLoses() { return numberOfLoses; }

    /**
     * Sets number of loses.
     *
     * @param numberOfLoses the number of loses
     */
    public void setNumberOfLoses(int numberOfLoses) { this.numberOfLoses = numberOfLoses; }

    /**
     * Gets number of wins.
     *
     * @return the number of wins
     */
    public int getNumberOfWins() { return numberOfWins; }

    /**
     * Sets number of wins.
     *
     * @param numberOfWins the number of wins
     */
    public void setNumberOfWins(int numberOfWins) { this.numberOfWins = numberOfWins; }

    /**
     * Add win.
     */
    public void addWin() { ++this.numberOfWins; }

    /**
     * Add lose.
```

50% classes, 39% lines covered in package 'oz222am_hangman.Models.Players'

Element	Class, %	Method, %	Line, %
Player	100% (1/1)	100% (11/11)	100% (22/22)
Players	0% (0/1)	0% (0/6)	0% (0/47)
PlayersTest	0% (0/1)	0% (0/11)	0% (0/43)
PlayerTest	100% (1/1)	100% (7/7)	100% (37/37)

3.2 Players class

The screenshot below shows “PlayersTest” class, it includes five test methods and the code coverage of this class “100%” on the left side of the screen. All the methods have passed.



.pdf
eLog.pdf
22am_hangman
Models
Games
Players 100% classes, 100% lines covered
Player 100% methods, 100% lines covered
Players 100% methods, 100% lines covered
PlayersTest 100% methods, 100% lines covered
PlayerTest 100% methods, 100% lines covered
Words
Word
Words
WordTest
Items
Model
resources
players.txt
players_ill_formatted_0.txt
players_ill_formatted_1.txt
players_ill_formatted_2.txt
players_load_test.txt
players_save_test.txt
words.txt

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

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


public class PlayersTest {
    String basePath = "E:\\Courses\\LNU-61004\\01\\1DV600\\Project\\src\\oz222am_hangman\\resou


    @Test
    void shouldSetPath() {
        var players = new Players();
        players.setPath("test_path");
        assertEquals( expected: "test_path", players.getPath());
    }

    @Test
    void shouldLoad() {
        var players = new Players();
        players.setPath(this.basePath + "no_found_path.txt");
        assertThrows(Exception.class, players::load);
        players.setPath(this.basePath + "players_ill_formatted_0.txt");
        assertThrows(Exception.class, players::load);

        players.setPath(this.basePath + "players_ill_formatted_1.txt");
        assertThrows(Exception.class, players::load);

        players.setPath(this.basePath + "players_ill_formatted_2.txt");
        assertThrows(Exception.class, players::load);

        players.setPath(this.basePath + "players_load_test.txt");
        assertDoesNotThrow(players::load);
```

```
Test Results                60 ms
  PlayersTest               60 ms
    shouldRegister()        20 ms
Tests passed: 5 of 5 tests – 60 ms
```

```
"C:\Program Files\Java\jdk-12\bin\java.exe" ...
---- IntelliJ IDEA coverage runner ----
sampling ...
include patterns:
oz222am_hangman\.Models\.Players\..*
exclude patterns:Class transformation time: 0.0290583s for 696 classes
```

100% classes, 100% lines covered in package 'oz222am_hangman.Models.Players'

Element	Class, %	Method, %	Line, %
Player	100% (1/1)	100% (11/11)	100% (22/22)
Players	100% (1/1)	100% (6/6)	100% (47/47)
PlayersTest	100% (1/1)	100% (10/10)	100% (45/45)
PlayerTest	100% (1/1)	100% (7/7)	100% (37/37)

This test is made to check that the functionality of the Players class meets the requirements (save, load, register, remove).

4. Test Report

Test traceability matrix and success

Test	UC1	UC2	Login Use Case	Register Player Use Case	List of Players Use Case
TC1	1/OK				
TC2			1/OK		
TC3			1/OK		
TC4		1/OK			
TC5				1/OK	
TC6					1/OK
Coverage & Success	1/OK	1/OK	2/OK	1/OK	1/OK

Automated unit test coverage and success

Test	Player	Players
PlayerTest	100%/OK	
PlayersTest		100%/OK
Coverage & Success	100%/OK	100%/OK

5. Reflection

Writing a test plan was not a very hard task since I found many templates on the internet that I can follow. However, the strange thing that I found is that we must submit all the reports (test plan, test report, manual test cases, and personal reflection) in one pdf file. Thus, I had confusion with organizing my final pdf document. I found many templates for the manual and automated test cases. So, I found no difficulties writing them. The estimated time that I sat in my time log was reasonable, well estimated and close to the actual.