

Berk Can BALATACI (Computer Science Erasmus Exchange Student)

bb222ji@lnu.se

GitHub: https://github.com/berkbltc/bb222ji_1dv600

Test Plan

Objective

The objective of the tests is check the program if it meets its requirements.

What to test and how?

The aim is to test use-cases by writing and running dynamic manual test-cases. I write test-cases for UC1(Start Game) and UC2(Play Game) as they are main functionalities of the program. Moreover, I write and run automated unit tests for all methods I implement so far.

Time plan

Task	Estimated	Actual
Test & Time Plan	30m	55m
Writing and Running Manual Test Cases	3h	3h45m
Writing and Running Unit Tests	4h	5h30m
Code inspection	40m	30m
Test Report	20m	15m
Write reflection	30m	25m

Manual Test-Cases

Test-Case 1.1: Start a round

Use case: UC1 Start Game

Scenario: Start a round successfully

Description: The first step of UC1 is tested where a gamer starts a round.

Precondition: “dictionary.csv” file must be in project folder.

Test Steps

- Run the application
- The system shows the main menu and expects an input.
- Type ‘1’ and press enter.

Expected

- The system picks a random word from “dictionary.csv” and starts the round.

Test Result: Test is passed successfully.

Test-Case 2.2: Win game

Use case: UC2 Play Game

Scenario: Win a round

Description: Main Scenario of UC2 is tested where a gamer plays the round and wins it.

Precondition: Starting a round choice must be selected. (TC1.1 was completed)

Note: *The word guessed is selected as “bass” for testing.*

Test Steps

- The program shows “ ***** The round is starting... *****
Guess this word : _ _ _ _
Guess a letter in the word :
 - Type ‘a’ and press enter.
 - System shows “ _a _ _
Guess a letter in the word : ”
 - Type ‘b’ and press enter.
 - System shows “ b a _ _
Guess a letter in the word : ”
 - Type ‘s’ and press enter.

Expected

- Score of gamer is displayed.
- Main menu is displayed.

Test Result: Test is passed successfully.

Test-Case 2.3 : Lose game

Use case: UC2 Play Game

Scenario: Lose a round

Description: Alternative Scenario of UC2 is tested where a gamer plays a round and loses it because of spending all 8 life chances.

Precondition: Starting a round choice must be selected. (TC1.1 was completed)

Note: The word guessed is selected as “bass” for testing.

Test Steps

- The program shows “ ***** The round is starting... *****
Guess this word : _ _ _ _
Guess a letter in the word :
 - Type ‘b’ and press enter.
 - System shows “ b _ _ _
Guess a letter in the word : ”
 - Type ‘e’ and press enter.
 - System shows “ b _ _ _
You have 7 life chances ! Guess a letter in the word : ”
 - Type ‘r’ and press enter.
 - System shows “ b _ _ _
You have 6 life chances ! Guess a letter in the word : ”
 - Type ‘r’ and press enter.
 - System shows “ b _ _ _
You have 5 life chances ! Guess a letter in the word : ”
 - Type ‘k’ and press enter.

- System shows “ b _ _ _
You have 4 life chances ! Guess a letter in the word : ”
- Type ‘c’ and press enter.
- System shows “ b _ _ _
You have 3 life chances ! Guess a letter in the word : ”
- Type ‘a’ and press enter.
- System shows “ b a _ _
Guess a letter in the word : ”
- Type ‘n’ and press enter.
- System shows “ b a _ _
You have 2 life chances ! Guess a letter in the word : ”
- Type ‘l’ and press enter.
- System shows “ b a _ _
You have 1 life chances ! Guess a letter in the word : ”
- Type ‘t’ and press enter.

Expected

- Round is lost as gamer is out of life chances.
- The word gamer tried to find is displayed.
- Main menu is displayed.

Test Result: Test is passed successfully.

Test-Case 2.4 : Unexpected or already used letters cause re-prompt

Use case: UC2 Play Game

Scenario: Win a round

Description: An alternate scenario of UC2 is tested where a gamer tries to enter unexpected letters and is forced to enter a new letter.

Precondition: Starting a round choice must be selected. (TC1.1 was completed)

Test Steps

- The program shows “
Guess this word : b _ _ _
Guess a letter in the word : ”
- Type ‘b’ and press enter.
- System shows “ You are already used this letter or it is an unexpected character!
Guess a letter in the word : ”
- Type ‘15’ and press enter.
- System shows “ You are already used this letter or it is an unexpected character!
Guess a letter in the word : ”
- Type ‘C’ and press enter.
- System shows “ You are already used this letter or it is an unexpected character!
Guess a letter in the word : ”
- Type ‘*’ and press enter.
- System shows “ You are already used this letter or it is an unexpected character!
Guess a letter in the word :

Expected

- The system displays a warning for unexpected or used letters.

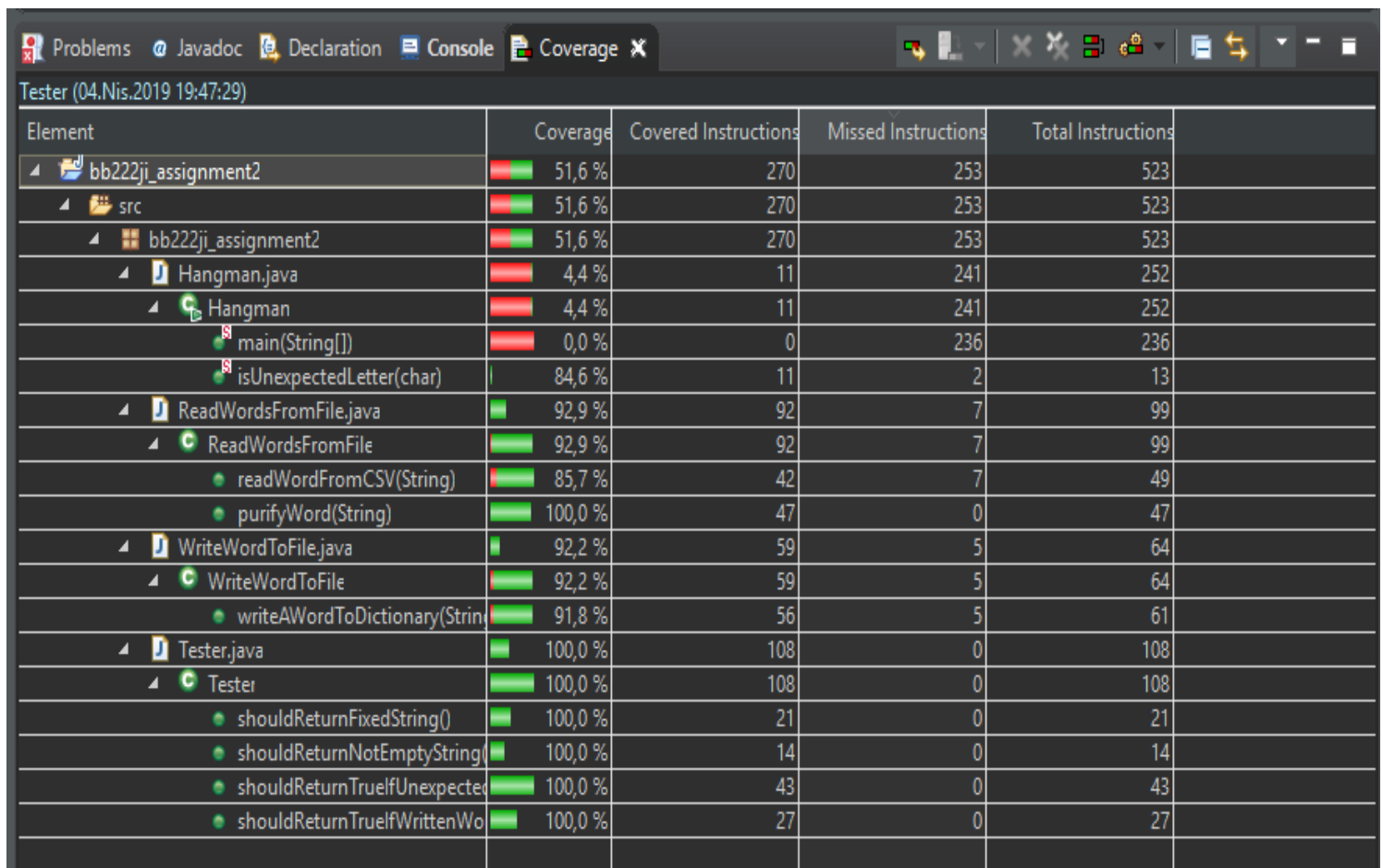
Test Result: Test is passed successfully.

Test Report

Test traceability matrix and success

Test	UC1	UC2
Test-Case 1.1	1/OK	0
Test-Case 2.2	0	1/OK
Test-Case 2.3	0	1/OK
Test-Case 2.4	0	1/OK
COVERAGE & SUCCESS	1/OK	3/OK

Automated unit test coverage and success



The screenshot shows the 'Coverage' window of an IDE, displaying test results for the project 'bb222ji_assignment2'. The window has tabs for 'Problems', 'Javadoc', 'Declaration', 'Console', and 'Coverage'. The 'Coverage' tab is active, showing a table with columns: 'Element', 'Coverage', 'Covered Instructions', 'Missed Instructions', and 'Total Instructions'. The table lists various elements, including source files and methods, with their respective coverage percentages and instruction counts. The 'Tester' class and its methods show 100% coverage.

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
bb222ji_assignment2	51,6 %	270	253	523
src	51,6 %	270	253	523
bb222ji_assignment2	51,6 %	270	253	523
Hangman.java	4,4 %	11	241	252
Hangman	4,4 %	11	241	252
main(String[])	0,0 %	0	236	236
isUnexpectedLetter(char)	84,6 %	11	2	13
ReadWordsFromFile.java	92,9 %	92	7	99
ReadWordsFromFile	92,9 %	92	7	99
readWordFromCSV(String)	85,7 %	42	7	49
purifyWord(String)	100,0 %	47	0	47
WriteWordToFile.java	92,2 %	59	5	64
WriteWordToFile	92,2 %	59	5	64
writeAWordToDictionary(String)	91,8 %	56	5	61
Tester.java	100,0 %	108	0	108
Tester	100,0 %	108	0	108
shouldReturnFixedString()	100,0 %	21	0	21
shouldReturnNotEmptyString()	100,0 %	14	0	14
shouldReturnTrueIfUnexpectedLetter()	100,0 %	43	0	43
shouldReturnTrueIfWrittenWord()	100,0 %	27	0	27

JUnit Test Class

```
*Tester.java Hangman.java ReadWordsFromFile.java WriteWordToFile.java
1 package bb222ji_assignment2;
2
3 import static org.junit.Assert.*;
4 import org.junit.Test;
5
6 public class Tester {
7
8     @Test
9     public void shouldReturnTrueIfUnexpectedLetter() {
10
11         boolean expected = true;
12         boolean actual = Hangman.isUnexpectedLetter('4');
13         assertEquals(expected, actual);
14         actual = Hangman.isUnexpectedLetter('ç');
15         assertEquals(expected, actual);
16         actual = Hangman.isUnexpectedLetter('*');
17         assertEquals(expected, actual);
18         actual = Hangman.isUnexpectedLetter('T');
19         assertEquals(expected, actual);
20         actual = Hangman.isUnexpectedLetter('<');
21         assertEquals(expected, actual);
22     }
23
24     @Test
25     public void shouldReturnFixedString() {
26
27         ReadWordsFromFile sut = new ReadWordsFromFile();
28         String expected = "love";
29         String actual = sut.purifyWord("LOVE");
30         assertEquals(expected, actual);
31         actual = sut.purifyWord("lov*?^e");
32         assertEquals(expected, actual);
33     }
34 }
```

```
34
35     @Test
36     public void shouldReturnNotEmptyString() {
37
38         ReadWordsFromFile sut = new ReadWordsFromFile();
39
40         String actual = sut.readWordFromCSV("dictionary.csv");
41         assertNotNull(actual);
42         assertEquals("", actual);
43     }
44
45     @Test
46     public void shouldReturnTrueIfWrittenWordIsProper() {
47
48         WriteWordToFile sut = new WriteWordToFile();
49
50         boolean expected = true ;
51         boolean actual = sut.writeAWordToDictionary("berk") ;
52         assertEquals(expected, actual);
53
54         expected = false ;
55         actual = sut.writeAWordToDictionary("b3Rk?") ;
56         assertEquals(expected, actual);
57     }
58 }
```


Reflection: I was curious about testing part because I always wanted to know how I should test a software in appropriate way. I did not know anything about testing environments like JUnit. With this iteration, I think I learned a bit why and how I should do it. In addition, while I was working on writing unit-tests I realized that I should use more methods instead of coding same thing with less methods since it is better for testing. Due to the fact that I did not know much about unit-testing, it took more time than I thought.