SOFTWARE DEVELOPMENT PROJECT HANGMAN

Assignment 3

Date 2019-04-05

RASHED QAZIZADA

GitHub Link: https://github.com/rqkohistani/rq222ah-1dv600/tree/master/Assignment%203

rq222ah@student.lnu.se

Document version: 3.0

COURSE CODE: 1DV600

Contents

1. Vision

Hangman, a game where you must guess words. Most of us may know the principle of the Hangman game. The game is to challenge your field of knowledge on the selected category.

However, if the player enters 7 wrong letters the player loses, and a complete hangman figure will be shown.

The game starts if the user input a valid name and the user can continue guessing and revealing the secret words.

2. Use-cases

This is a simple client application with two use-cases.

UC1 checks user's input while giving a name

Pre-condition: the game has started

Post-condition: the user has entered a name

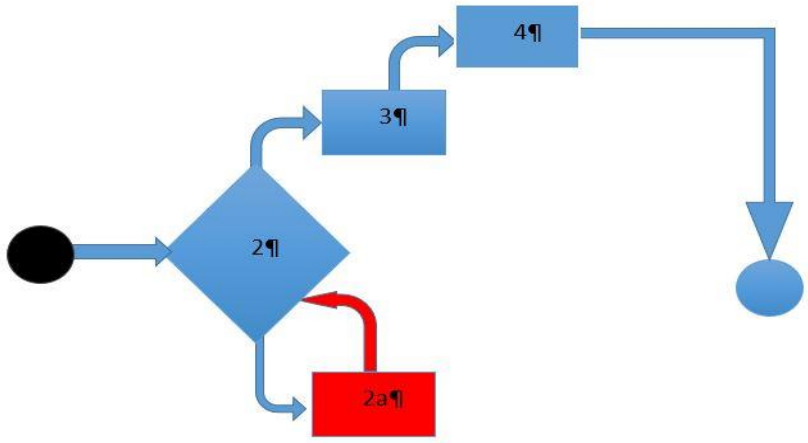
Main scenarios

1. Starts when the user wants to store their name in the game session.
2. The system asks for the name
3. The user gives a name
4. The system greets and prompts the user either to play or quit the game

Alternative scenarios

- 2a the user provides a negative integer, an empty name or characters other than English letters
- The system shows an error message

- Goes back to step 2. In main scenario



UC2: checks user's input to quit the game

Pre-condition: The user has successfully revealed the secret word.

Post-condition: The user has entered any positive integer other than "1" and the game terminates/quits

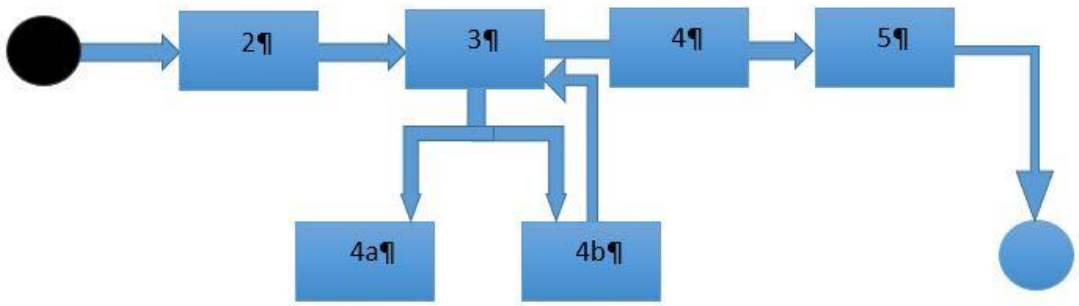
Main scenarios

1. The user has successfully revealed the secret word.
2. The system greets the user upon wining
3. The system asks the user "Do you want to play again?"
4. The user makes a choice to quit the game by choosing any positive integer other than "1"
5. A goodbye message will be displayed in the console.

Alternative scenarios

- 4a the user provides "1"
- The system restarts the game
- 4b the user provides a negative integer
- The system displays an error message "provide a positive integer"

Activity diagram



3. Test plan

Objectives

The main objective in this iteration is to test part of the code (both dynamic and static) that was implemented in iteration 2.0.

What to test and how

The manual test will be done in a static manner by following a set of scenarios and analyzing whether the output corresponds to the expect one or not. However, the Unit Test will also have a dynamic approach, and this is done by writing and running dynamic manual test cases. The reason for testing these use cases is to test the fundamental functionalities of the project.

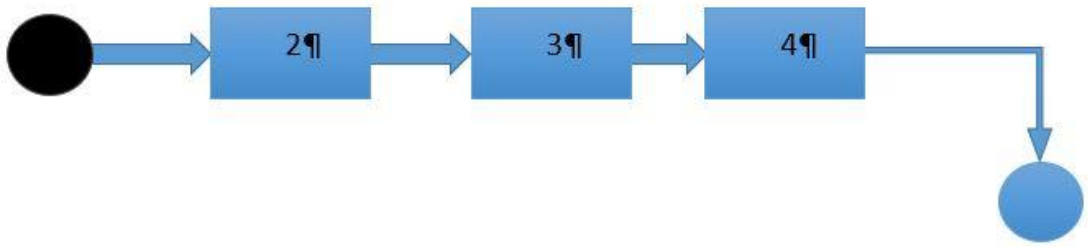
Time plan

Theme 3	**Actual**	**Estimated**
Vision	5m	10m
Test plan	30m	30m
Manual test cases	3h	5h
Unit tests	1h	1h
Reading the book	12h	10h
Coding	15h	10h
Test report	2h	2h
Reflection	1h	20m
Completed the documentation in	**34 h & 35m**	**29h & 0m**
Theme 2	**Duration**	**Estimated**

4. Manual Test-Cases

TC1.1 User made a valid input

Scenario: checks user's input successful



The main scenario of UC1 is tested where a user makes a valid input.

Pre-condition: Start the game.

Test steps

- Start the game
- The system shows a message "Enter a name: (only English letters or positive integers)"
- Enter the name "Rashed" and press enter

Expected

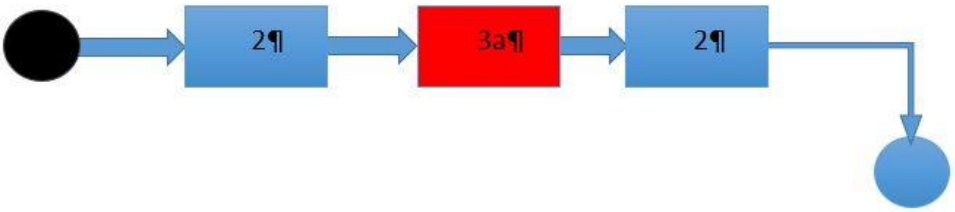
- The system should store the name and greets the user "Rashed"
- The game will ask the user to select 1 to play and 2 to quit.

TC1.2 user made an invalid input

Use-Cases: UC1 checks user's input

Scenarios: invalid input force to input again

The alternative scenario comes into forces where user inputs rather than English alphabets or negative integers.



Pre-condition: The game has already started

Test steps

- Start the game
- The system shows a message "Enter a name: (only English letters or positive integers)"
- Enter the name "Åsa" or press enter without entering a name

Expected

- The system should display the error message "English Letters Or positive integers!".
- The system prompts the user to enter a valid name

Test report -TC 1.1, TC1.2

Test traceability matrix and success

Test	UC1	UC2
TC1.1	1/OK	0
TC1.2	1/OK	0
Coverage and success	2/OK	

Automated unit test coverage and success

Test	Main	Quiting_Test	Hangman
UserName_Test	0	0	100%OK
Coverage and success	0/NA	0/NA	100%/OK

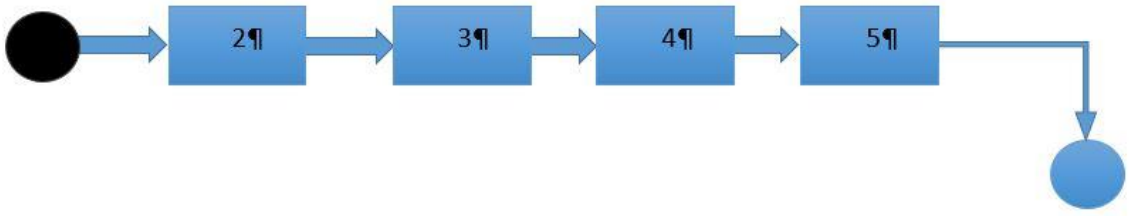
Comment

All the tests successfully passed, next iteration we need to focus on UC2 and the codes inspected and most classes require mocks to be testable.

TC2.1 User made a valid input to quit the game

Use-cases: UC2 checks user's input

Scenario: checks user's input successful.



The main scenario of UC2 is tested where a user made a valid input to quit the game.

Pre-condition: reveal the secret word

Test steps

- Finished the game by revealing the secret word
- The system greets the user upon wining
- System shows menu
    1. Press “1” to restart and play again
    2. Press “any positive integer” to quit the game
- Enter “3” and press enter

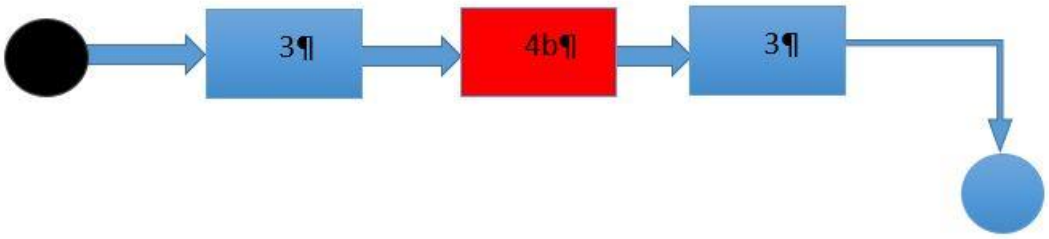
Expected

- The system terminates the game with a message “Thank you and Goodbye”

TC2.2 user made an invalid input to quit the game
Use-cases: UC2 check user input to quit the game

Scenario: The user enters a negative integer. Refer 4b under alternative scenario.

The alternative scenario comes into force while user inputs a negative integer and displays an error message.



Pre-condition: The secret word is revealed

Test steps

- Finished the game by revealing the secret word
- The system greets the user upon wining
- System shows menu
    1. Press “1” to restart and play again
    2. Press “any positive integer” to quit the game
- Enter “-3” and press enter

Expected

- The system should show an error message
- System displays “provide a positive integer”
- The system waits for the input

Test report TC 2.1, TC2.2

Test traceability matrix and success

Test	UC1	UC2
TC2.1	0	1/OK
TC2.2	0	1/OK
Coverage and success	0	2 /OK

Automated unit test coverage and success

Test	**UserName_Test**	**Main**	**Hangman**
Quiting_Test	0	0	100%/OK
Coverage and success	0/NA	0/NA	100%/OK

Comment

All the tests successfully passed. The codes inspected and most classes require mocks to be testable.

5. Unit Test

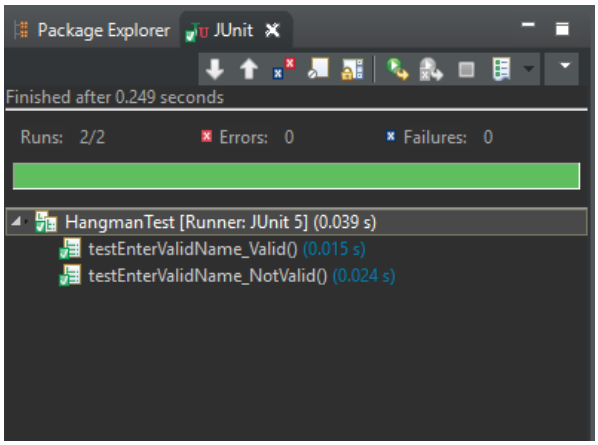
Source code 1:

```
public boolean enterValidName(String name) {
    if (!name.matches("[a-zA-Z0-9]+")){
        System.err.println("English Letters Or positive integers!");
        return false;
    }
    else {
        System.out.println(name+"! Welcome to the Hangman game");
        return true;
    }

}
```

Junit for Source code 1:

```
Hangman hangman = new Hangman();

@Test
void testEnterValidName_Valid() {
    assertEquals(true, hangman.enterValidName("Rashid"));
    assertEquals(true, hangman.enterValidName("Topias"));
    assertEquals(true, hangman.enterValidName("John"));
    assertEquals(true, hangman.enterValidName("Bond007"));


}

@Test
void testEnterValidName_NotValid() {
    assertEquals(false, hangman.enterValidName("+/*!%/&"));
    assertEquals(false, hangman.enterValidName(" "));
    assertEquals(false, hangman.enterValidName("-Ahmad"));

}
```

Result of source code 1:



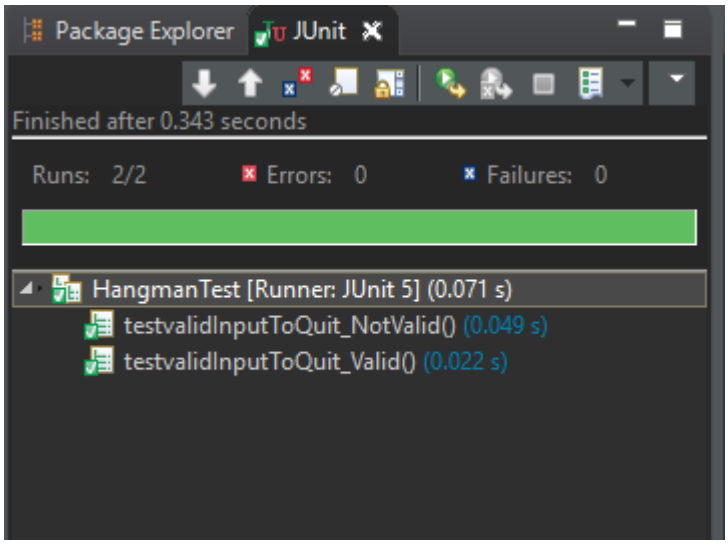
Source code 2:

```
public boolean validInputToQuit(int number) {
    while(number<0) {
        System.err.println("provide positive integer");
        return false;
    }
    if (number == 1) {
        secretWord = "";
        hiddenWord = "";
        return true;
    }
    return true;
}
```

Junit for Source code 2:

```
@Test
void testvalidInputToQuit_Valid() {
    // Quit the game while you input a positive number other than number 1
    assertEquals(true, hangman.validInputToQuit(2));
    assertEquals(true, hangman.validInputToQuit(10));
    assertEquals(true, hangman.validInputToQuit(400));

}

@Test
void testvalidInputToQuit_NotValid() {
    assertEquals(false, hangman.validInputToQuit(-2));
    assertEquals(false, hangman.validInputToQuit(-5));
    assertEquals(false, hangman.validInputToQuit(-20));

}
```

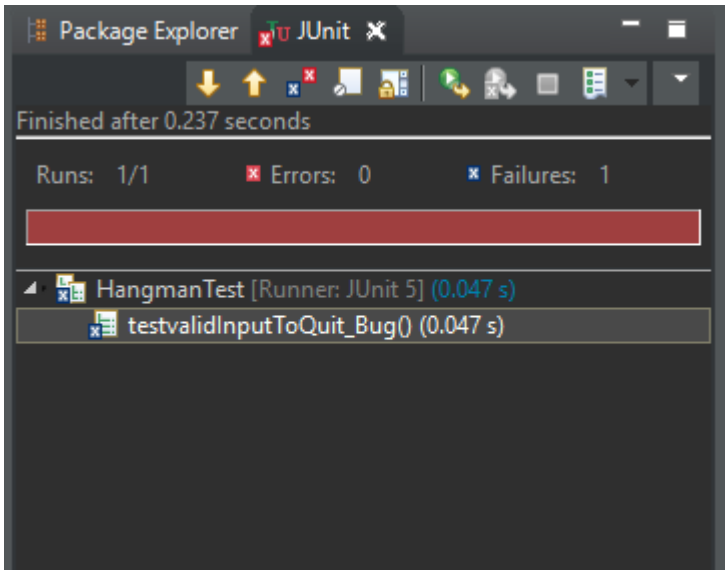
Result of source code 2



Junit for Source code 2 “Bug”:

```
@Test
void testvalidInputToQuit_Bug() {       // this method should return false, but it return true;
    assertEquals(false, hangman.validInputToQuit(0));
}
```

Result of source code 2 “Bug”:



6. Reflection:

This is my first software project where I wrote my own codes. In this iteration I improved a lot of coding skills and also manual testing by following a set of scenarios and analyzing whether the output of my codes corresponds to the expect one or not. Since I have never tested any software, I was not sure how it works.

However, at the beginning of this iteration I was a little confused due to the other assignments and exams but gradually I manage to figure it out. I believe Junit testing was very interesting, as it was about checking your own code to test if it works or not as well as fix any bugs in the code. At this step I have learnt new ways to code. I understood that it not only important to code but also test it and after doing this iteration I feel more confident.

I am getting better in planning and documentation. Performing these tests is not only crucial for coding defects but also helps in testing the code at an early stage. I had to re-write a lot my codes to be able to do the Junit testing since I called the objects through the constructor. Since I have not completed the assignment 2 yet, reading the book, watching lectures videos and developing my code took longer time until I got a clear picture which of course led me to spend my time going through Theme2 and Theme3.