Testing: Hangman Game

Name: Donald Appiah
Student mail: da222pa@student.lnu.se
GitHub:

Task 1 | Test Plan

Aim:
The aim of this iteration is to test the code that has been implemented in the last iteration focusing on the different cases in the program, finding bugs, testing methods, buttons and any other type of hindrances the code might have. In order to have an easy approach and understanding towards the maintenance of the code in the last iteration.

What to test?

Use-case 2 will be tested through writing and running manual test-cases. It has already been implemented and ready to be tested. And we want to make sure that the game is doing what it is supposed to do. We are going to specifically test the methods getOnlyFirstLeter();,isOver (); and info();. These three methods are important methods that make the application works and therefore the methods have been selected to be tested.

How to test?

A manual testing and automated testing (using Junit) to be able to determine if the application is working according to its requirements.

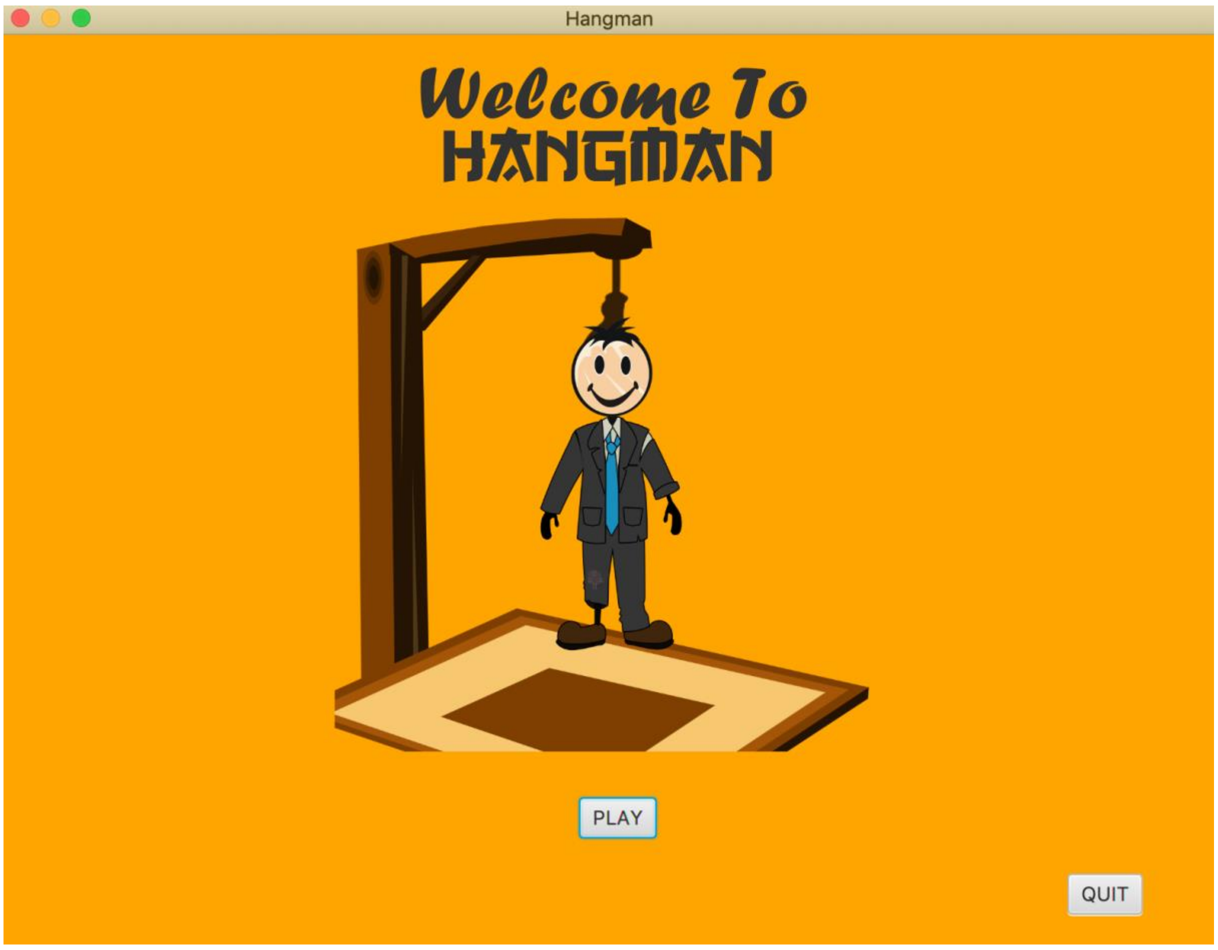
Time Plan:

Task	Estimated	Actual
Manual Taste Case	1h	30min
Unit Tests	1h 30min	2h
Running Manual Tests	30min	20min
Code Inspection	30min	30min
Test Report	2h30min	1h
Total	6h	4h20min

Task 2 | Manual Test Cases

TC1.1 Play

Requirement (use-case): UC1 Start Game Scenario: Click on the button “Play”



This is the main scenario of UC1, in which the player clicks on the button “Play” in order to continue to UC2 (game playing).

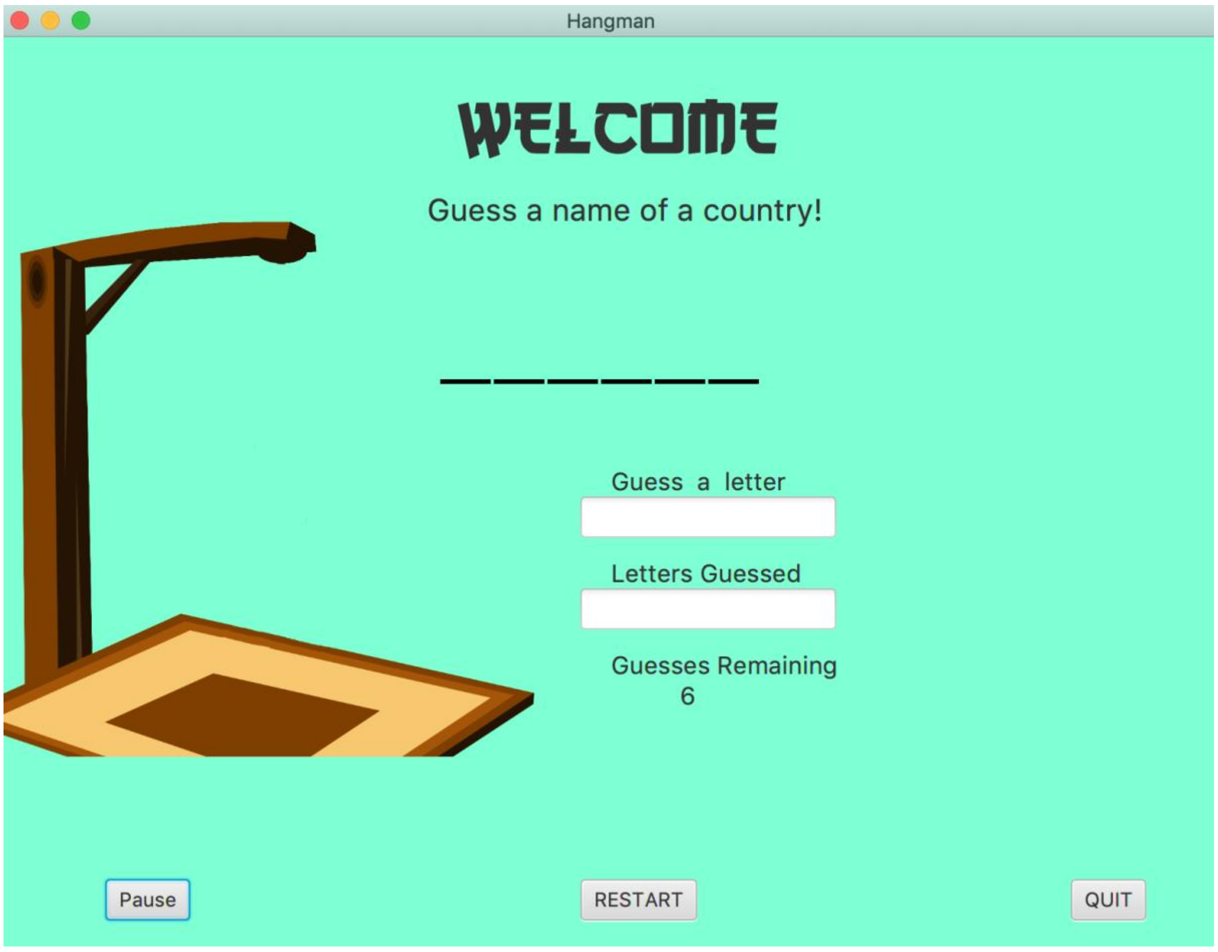
Precondition: Run a release from jar archive.

Test Steps:

- Run a release from jar archive.
- System shows the game menu
- Player clicks on the button “Play”

Expected:

• The system will then redirect the player to the UC2 game playing scenario where the player sees instructions on what to do, example is what categories of words to be guessed.



Results

Did the Test succeed*: YES*.
Comments:...

TC1.2 Quit

Requirements (use-case): UC1 Start Game

Description/ Scenario: Click on button "Quit". It is an alternate scenario where the user decides to terminate the program and quit the game.

Precondition: Run a release from jar archive.

Test Steps:

- Run a release from jar archive.
- System shows the game menu
- Player clicks on the button “Exit”

Expected:

• The system then terminates the program.

Results

Did the Test succeed*: YES*.
Comments:...

TC1.3.1 Pause

Requirements (use-case): UC1 Start Game

Description: Click on button “Play” the game is then paused and takes the player back to the main menu UC1 and the player can click on” Play” or decide to terminate the program with a click on the “Quit” button.

Precondition: User decided to click on “Play” button from main menu in UC1.

Test Steps:

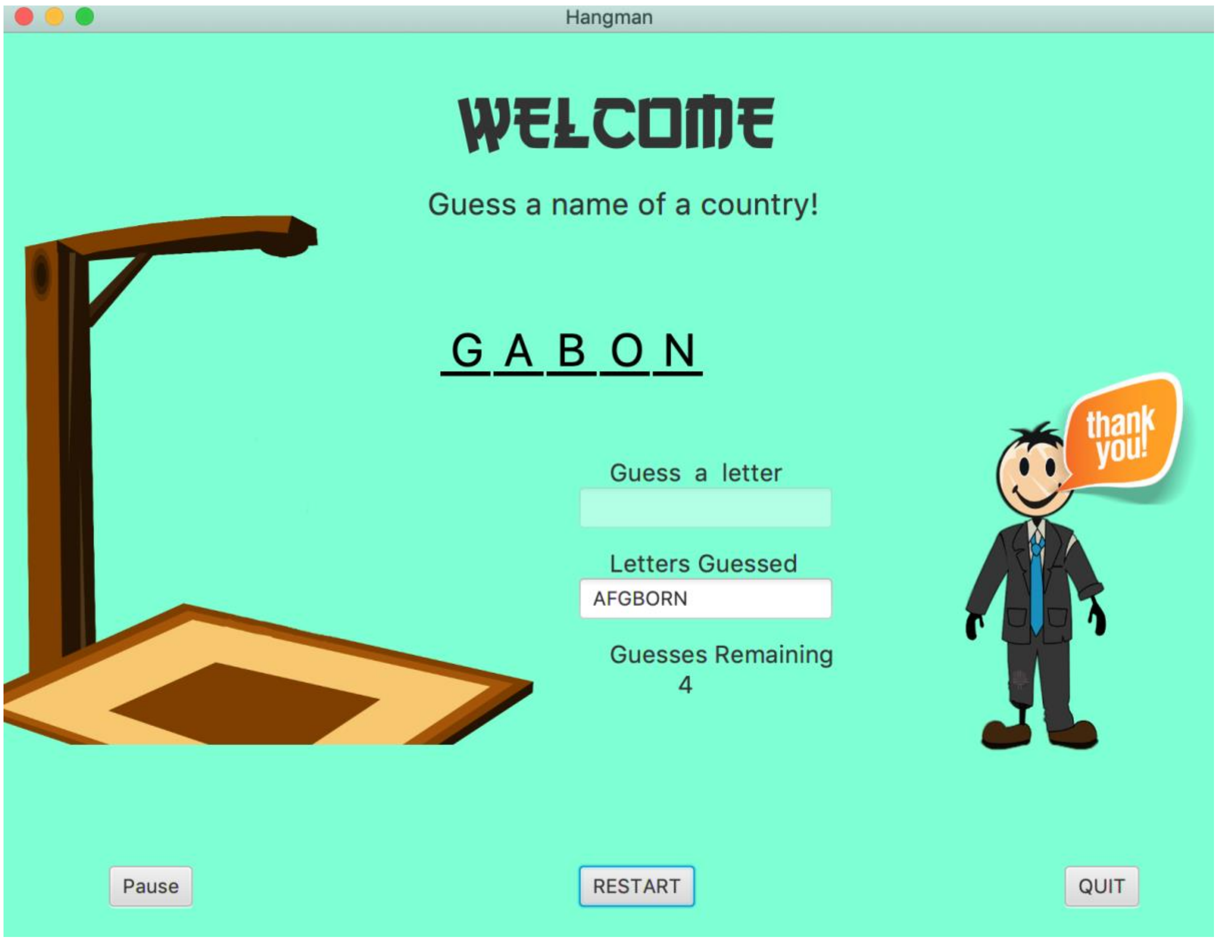
• User clicks on “Pause” button from the UC2.

Expected:

• The system redirects the user to the UC1 main menu.

TC2.1 Win Game
Requirements (use-case): UC2 Play Game

Description: In this test case if the player manages to guess all the right letters of the word without using all guesses remaining and win, Eventually the game will display an image with description thank you!



Precondition: UC1 must have been started and the button played must have been clicked in order to get to the UC2 scenario.

Test Steps:

- The player gets the instructions.
- The player started by inserting letters in such order:
- Inserts: A
- Inserts: F
- Inserts: G
- Inserts: B
- Inserts: O
- Inserts: R
- Inserts: N

Expected:

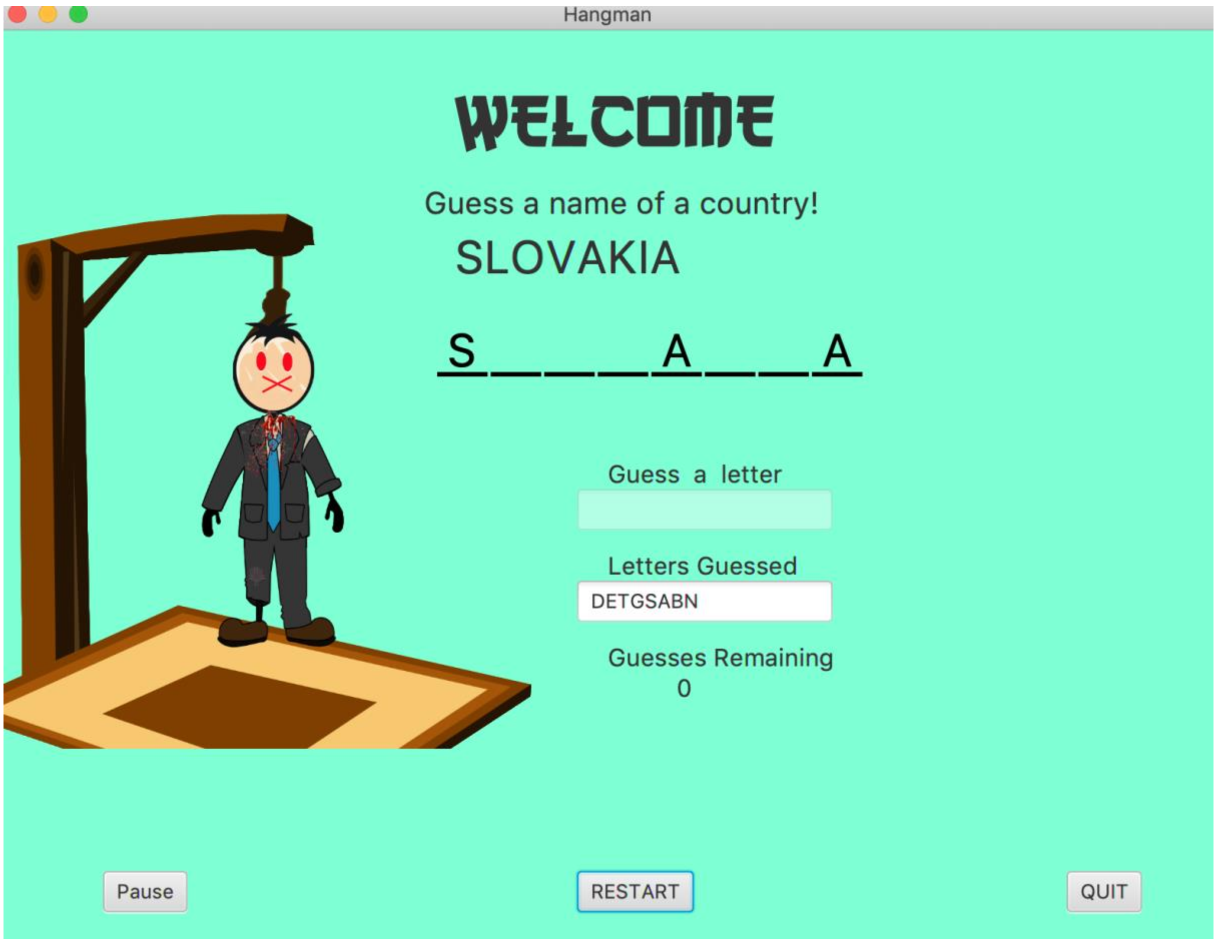
- The "hangman" gets saved and off the hanging.
- An image is displayed: "Thank you!"

Results

Did the Test succeed*: YES*. Comments: ....

- **TC2.1 Lose Game**
  Requirements (use-case): UC2 Play Game

Description: In this test case if the player does not guess all the right letters of the word and uses all guesses remaining, the hangman dies and the word which was to be guessed will be displayed.



Precondition: UC1 must have been started and the button played must have been clicked in order to get to the UC2 scenario.

Test Steps:

- The player gets the instructions.
- The player started by inserting letters in such order:
- Inserts: D

- Inserts: E
- Inserts: T
- Inserts: G
- Inserts: S
- Inserts: A
- Inserts: B
- Inserts: N

Expected:

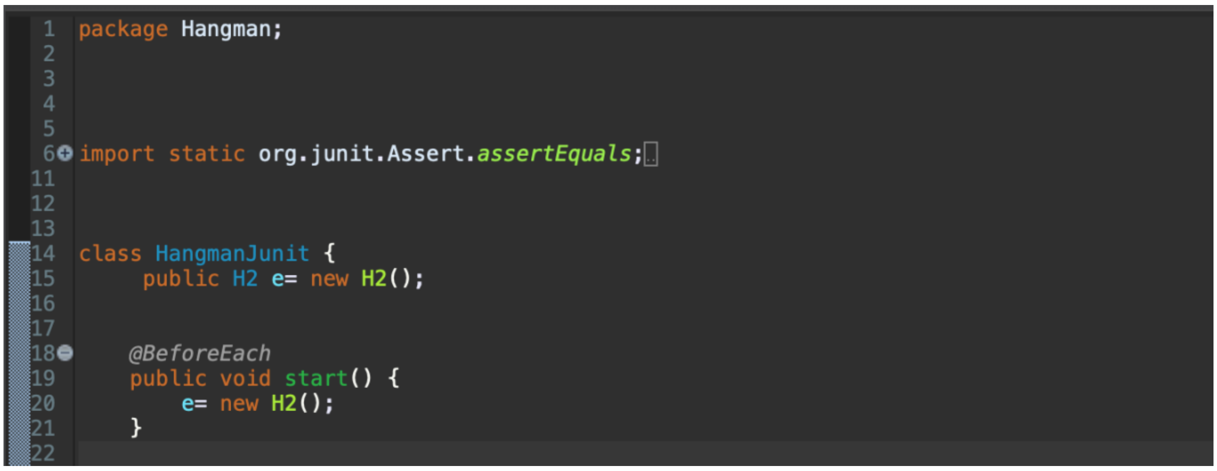
- The "hangman" gets killed.
- The unguessed word is shown to the player.

Results

Did the Test succeed*: YES*.
Comments:..

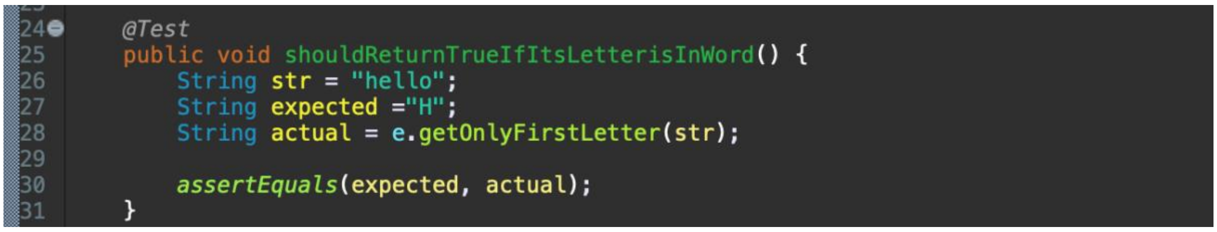
Task 3| Unit Tests

JUnit Test Class:

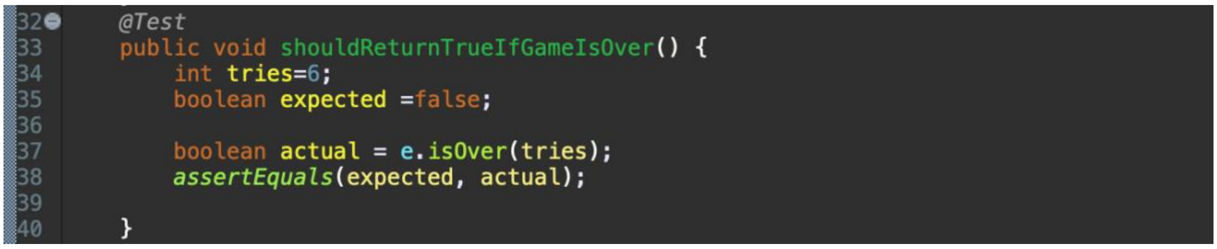


```
package Hangman;



import static org.junit.Assert.assertEquals;


class HangmanJunit {
     public H2 e= new H2();


    @BeforeEach
    public void start() {
        e= new H2();
    }

```

Automated Unit test 1: getOnlyFirstLetter();

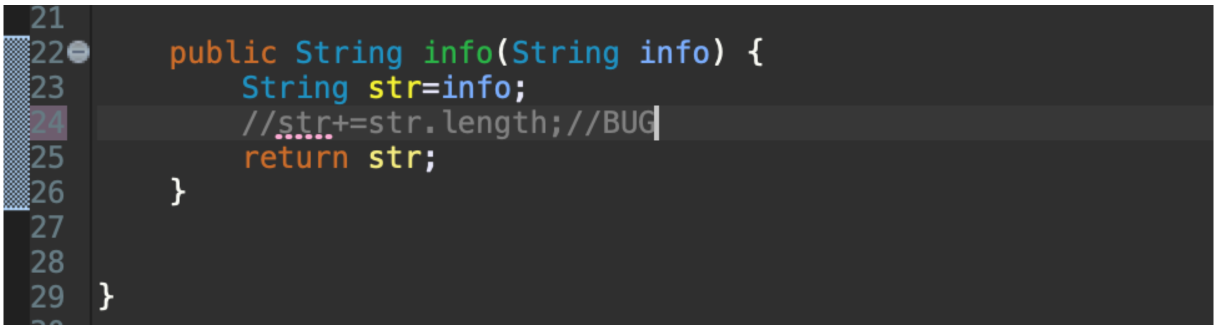

```java
    @Test
    public void shouldReturnTrueIfItsLetterisInWord() {
        String str = "hello";
        String expected ="H";
        String actual = e.getOnlyFirstLetter(str);

        assertEquals(expected, actual);
    }
```

Automated Unit test 2: isOver();

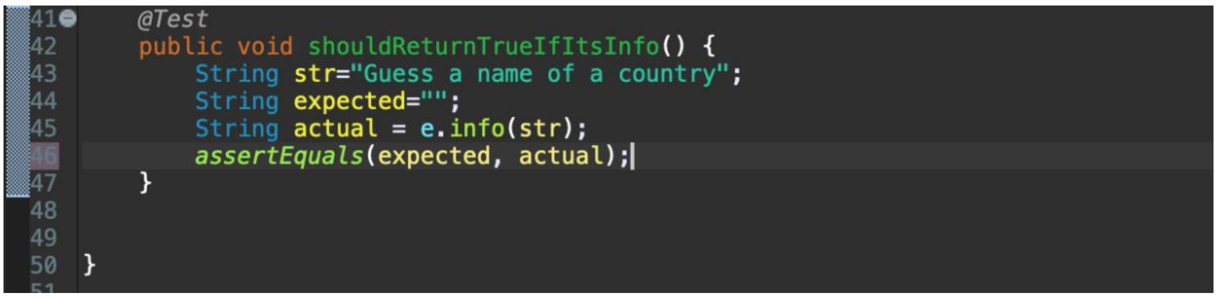

```java
    @Test
    public void shouldReturnTrueIfGameIsOver() {
        int tries=6;
        boolean expected =false;

        boolean actual = e.isOver(tries);
        assertEquals(expected, actual);

    }
```

Automated Unit test 2: info()//BUG;

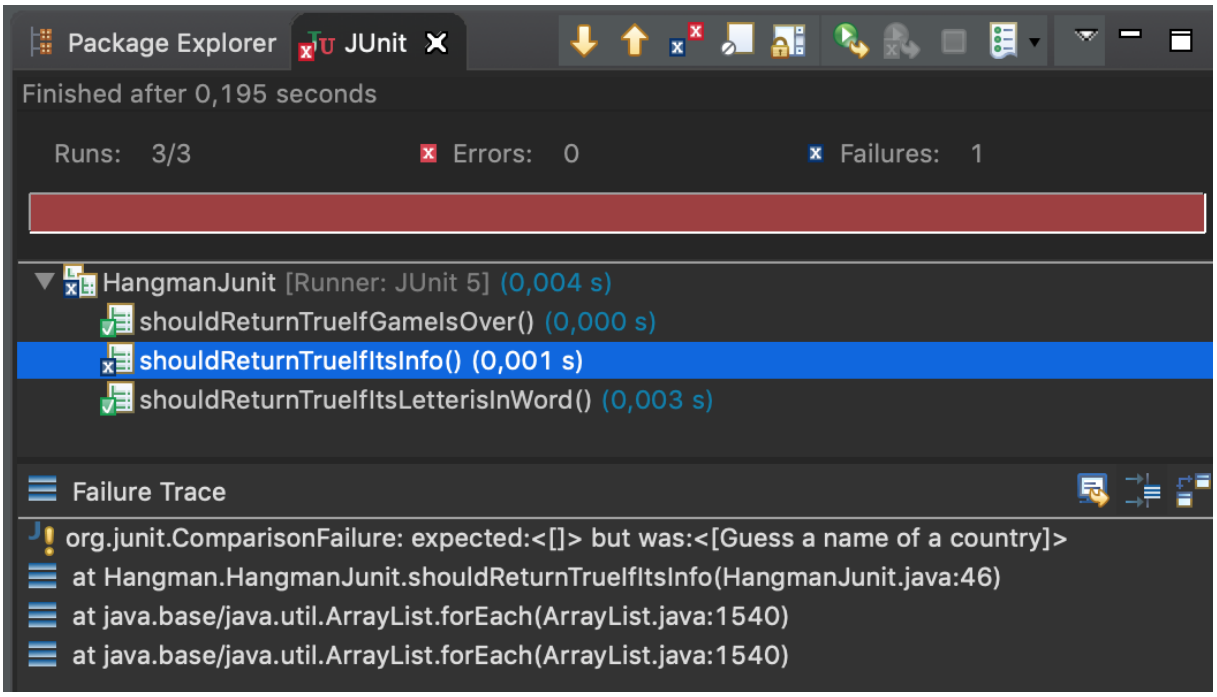
Code:


```java

    public String info(String info) {
        String str=info;
        //str+=str.length;//BUG
        return str;
    }


}
```

Test:


```java
    @Test
    public void shouldReturnTrueIfItsInfo() {
        String str="Guess a name of a country";
        String expected="";
        String actual = e.info(str);
        assertEquals(expected, actual);
    }



}
```

Test Result:



```
Finished after 0,195 seconds

Runs:   3/3          Errors:   0          Failures:   1

HangmanJunit [Runner: JUnit 5] (0,004 s)
    shouldReturnTrueIfGameIsOver() (0,000 s)
    shouldReturnTrueIfItsInfo() (0,001 s)
    shouldReturnTrueIfItsLetterisInWord() (0,003 s)

Failure Trace
org.junit.ComparisonFailure: expected:<[]> but was:<[Guess a name of a country]>
at Hangman.HangmanJunit.shouldReturnTrueIfItsInfo(HangmanJunit.java:46)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1540)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1540)
```

The first two methods were successful. But the test method for "shouldReturnTrueIfItsInfo()" did not work as expected.

Reflection:

Through this Testing iteration I have acquired a new understanding on why this should be done. It won't only provide the player useful information on how the program should look like but also provides me (the programmer) different ways on the development of software and product. Different testing approach provide different results to be analyzed and certain information. The manual testing helps the programmer and the player to know what is actually supposed to happen in the program, if anything goes not according to the test case then the program must be analyzed once more and looked for the specific problem causing the hindrances. Unit testing on the other hands provides actual information of the skeleton code of the program and helps the developer identify specific hindrances/bug location in the program. Which makes it much easier to approach uncertainties within the software. Being able to fix and correct problems. I will take this into account for the final iteration of this project and for my future projects.