# TEST REPORT FOR 1DV600 SOFTWARE TECHNOLOGY

written by: *Ruslan Abdulin*
personal code: *ra222rt*
email: *ra222rt@student.lnu.se*

project name: *Hangman The Game*

GitHub link:
https://github.com/ruslanabdulin1985/ra222rt_1dv600

Assignment 3 release link:
https://github.com/ruslanabdulin1985/ra222rt_1dv600/releases/tag/Assigment3

# 1. Test plan

**Objective:**

The objective is to test the code that was implemented during the last iteration as a lot of changes have been done. Such a functionality to create\choose players as well as high score tables have been added which modified the entire structure of the program by adding several classes to its source code.  The program should be tested to make sure that it continue working according to the requirements.

**What to test and how:**

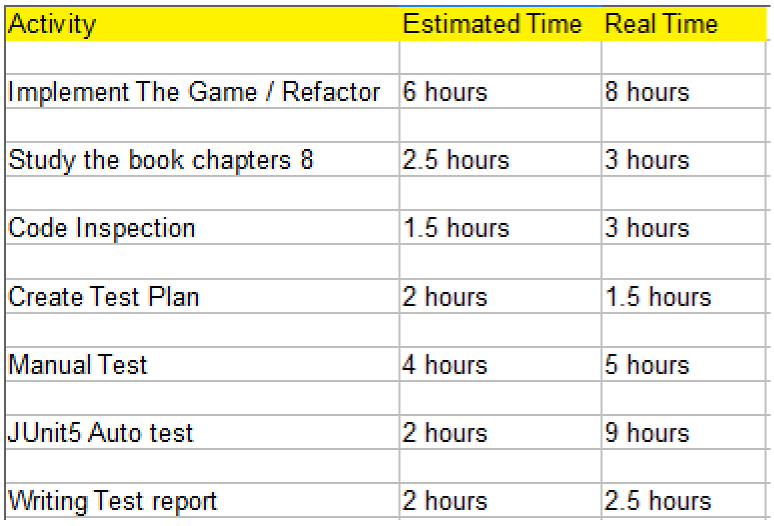
I intend to perform *functional – dynamic – manual - black box - use case* tests for two main use cases : **Use Case 1: Start Game** and **Use Case 2: Play the game** as these are key scenarios of the game. All the additional features like creating/choosing a player or save/load game will be tested later on because they are secondary and don't really affect the main functionality. The result of manual testing will be presented as four test cases and Requirements matrix at the end in order to show the overall result of the manual tests.

To fulfil basic testing requirements two methods belonged to **HangMan.java** class and one belongs to class **Player.java** will be *dynamically* tested using *automated JUnit white box testing*. These methods are **getStatus(), isThereALetterTest2()** and **GetSave()**. The methods are very important for the main functionality as Status of the game is a key parameter of the game process whereas existing or not a particular letter in a particular word is vital criteria of guessing process. Before the automatic tests source code would be *statically inspected* for mistakes. The results of automated testing will be attached to the report in corresponding section as screenshots from Eclipse IDE.

Also I am going to perform *dynamically automated JUnit* testing for all the public methods in all the 4 back end classes **DataBase.java**, **HangMan.java**, **Player.java Save.java** in order to increase confidence in the program. These tests will check the methods using assertions of possible data. The result of Junut testing will be presented as a screenshot of inbuilt Java Eclipse coverage tool after running tests.

While the testing process a careful time log will be made according to time-estimations plan:

**TIMELOG:**



| Activity | Estimated Time | Real Time |
|---|---|---|
| | | |
| Implement The Game / Refactor | 6 hours | 8 hours |
| | | |
| Study the book chapters 8 | 2.5 hours | 3 hours |
| | | |
| Code Inspection | 1.5 hours | 3 hours |
| | | |
| Create Test Plan | 2 hours | 1.5 hours |
| | | |
| Manual Test | 4 hours | 5 hours |
| | | |
| JUnit5 Auto test | 2 hours | 9 hours |
| | | |
| Writing Test report | 2 hours | 2.5 hours |

## 2. Manual Test cases

Use Case 1 – Start Game and Use Case 2. Play Game will be dynamically tested using command line interface. To make it easier to follow I include the main scenarios of these use cases from the previous iteration.

### UC 1: Start Game

Gamer: wants to start playing Hangman The Game

**Main scenario**

1. Gamer runs the game program
2. Program asks to choose a player (see UC 3. choose a player) or to create a new one (see UC4 create
player)
3. Gamer choose or create a player
4. The Program presents the main menu with a title, the option to play or to load unfinished game or to
show the table of high scores or to quit the game or to show the rules.
5. Gamer makes choose to start the game.
6. System starts the playing process (see UC 2 play the game)

### UC 2: Play the game

*Precondition: The program is running and player is chosen.*

Gamer : wants to win the game

**Main scenario**

1. Gamer starts the game (see UC 1 Start the game)
2. The Program provides a basic part of a picture of hangman and a hidden word as number of dashes.
The number is equal to the number of letters

3. Gamer tries to guess a letter
4. The Program program shows all the letters instead of dashes if Gamer guessed. Otherwise The
programs adds parts to a Hangman Picture.
*Steps 3-4 repeats until the word is guessed or the picture is complete*
If the word is guessed player gets earned points and returns to step 2.
If the picture is complete player loses all the earned points and the Program asks if he wants play again

### TC 1.1 Start Game with default player is successful

| | |
|---|---|
| Use Case to test: | Use Case 1 – Start Game |
| Description: | HangmanCMD application is going to be tested dynamically. The aim of this test is to make sure that the application can start the game process with default player. |
| Preconditions : | The application is started and a player is chosen or created |
| Test steps: | 1. Start HangmanCMD<br><br>2. Program shows logo of hangman and greeting message “`Welcome to Hangman The Game!`”<br>Also program asks to `Enter 'n' for New Player or 'e' for Existing Player`<br>and provides a line to type an answer `Type here:`<br><br>3. Type ‘e’ in the line `Type here:` and press Enter<br><br>4. The logo is shown againg.<br>System ask to `Choose a player by typing a corresponding number`<br>Also system provides a list of existing players which are:<br><br>`1 - Default`<br>`2 - Ruslan`<br>`3 - Vasya Pupkin`<br><br>and provides a line to type an answer `Type here:`<br><br>5. Type 1 in the line `Type here:` and press Enter to choose Default Player<br><br>6. Program shows a logo again and changes the greting message adding the name in the end:<br><br>`Welcome to Hangman The Game, Default!`<br><br>Also Program shows main menu with options to choose :<br>`Enter 'p' to Play New Game`<br>`Enter 'l' to Load Your Previous Game`<br>`Enter 'h' to Look At Highscores`<br>`Enter 'q' to Exit Game` |

|  |  |
|---|---|
|  | and provides a line to type an answer `Type here:`<br><br>7. Type ‘p’ in the line `Type here:` and press Enter |
| **Expected** | A new game starts and the interface for the game is shown. The interface has line with alphabet looking excactly like this<br><br>`\|a\| \|b\| \|c\| \|d\| \|e\| \|f\| \|g\| \|h\| \|i\| \|j\| \|k\| \|l\| \|m\| \|n\| \|o\| \|p\| \|q\| \|r\| \|s\| \|t\| \|u\| \|v\| \|w\| \|x\| \|y\| \|z\|`<br><br>a word to guess is shown as dashes:<br><br>`_ _ _ _ _ _ _ _`<br><br>and a line to type a letter is shown as well<br><br>`Type here:` |
| Status | Passed |
| Comments | Proceeded as expected |

**TC 1.2 Start Game with a new player is successful**

|  |  |
|---|---|
| Use Case to test | Use Case 1 – Start Game |
| Description | HangmanCMD application is going to be tested dynamically. The aim of this test is to make sure that the application can start the game process with a new player. |
| Preconditions | The application is started and a player is chosen or created |
| Test steps: | 1. Start HangmanCMD<br><br>2. Program shows logo of hangman and greeting message “`Welcome to Hangman The Game!`”<br>Also program asks to `Enter 'n' for New Player or 'e' for Existing Player` and provides a line to type an answer `Type here:`<br><br>3. Type ‘n’ in the line `Type here:` and press Enter for a new player<br><br>4. The logo is shown again.<br>System ask to `Enter Your Name`<br><br>and provides a line to type an answer `Type here:`<br><br>5. Type `Daniel` in the line `Type here:` and press Enter to start the game<br><br>6. Program shows a logo again and changes the greting message adding the name in the end:<br><br>`Welcome to Hangman The Game, Daniel!`<br><br>Also Program shows main menu with options to choose :<br>`Enter 'p' to Play New Game`<br>`Enter 'l' to Load Your Previous Game` |

| | |
|---|---|
| | `Enter 'h' to Look At Highscores`<br>`Enter 'q' to Exit Game`<br>and provides a line to type an answer `Type here:`<br><br>7. Type ‘p’ in the line `Type here:` and press Enter |
| **Expected** | A new game starts and the interface for the game is shown. The interface has line with alphabet looking excactly like this<br><br>`\|a\| \|b\| \|c\| \|d\| \|e\| \|f\| \|g\| \|h\| \|i\| \|j\| \|k\| \|l\| \|m\| \|n\| \|o\| \|p\| \|q\| \|r\| \|s\| \|t\| \|u\| \|v\| \|w\| \|x\| \|y\| \|z\|`<br><br>a word to guess is shown as dashes:<br><br>`_ _ _ _ _ _ _ _`<br><br>and a line to type a letter is presented as well:<br><br>`Type here:` |
| Status | Passed |
| Comments | Proceeded as expected |

**TC 2.1 Play Game and Win**

| | |
|---|---|
| Use Case to test | Use Case 2. Play Game |
| Description | The test is supposed to check the game process which leading to a win situation. |
| Precondition | The game should be started and the game interface is shown according to TC1.1 with default player |
| Constant expectation | Line *`Type here:`* `is always available` |
| Test steps | 1. Type ‘n’ letter<br><br>2. System shows alphabet with ‘cross out’ letter n:<br><br>`\|a\| \|b\| \|c\| \|d\| \|e\| \|f\| \|g\| \|h\| \|i\| \|j\| \|k\| \|l\| \|m\| -n- \|o\| \|p\| \|q\| \|r\| \|s\| \|t\| \|u\| \|v\| \|w\| \|x\| \|y\| \|z\|`<br><br>and the word in dashes opens guessed letters<br><br>`_ n _ _ _ _ _ n`<br><br>3. Type ‘i’ letter and press Enter<br><br>4. System shows alphabet with ‘cross out’ letter i:<br><br>`\|a\| \|b\| \|c\| \|d\| \|e\| \|f\| \|g\| \|h\| -i- \|j\| \|k\| \|l\| \|m\| -n- \|o\| \|p\| \|q\| \|r\| \|s\| \|t\| \|u\| \|v\| \|w\| \|x\| \|y\| \|z\|`<br><br>and the word in dashes opens guessed letters |

| | |
|---|---|
| | `i  n  _  _  _  i  _  n`<br><br>5. Type ‘v’ letter and press Enter<br><br>6. System shows alphabet with ‘cross out’ letter v:<br><br>`\|a\| \|b\| \|c\| \|d\| \|e\| \|f\| \|g\| \|h\| -i- \|j\| \|k\| \|l\| \|m\| -n- \|o\| \|p\| \|q\| \|r\| \|s\| \|t\| \|u\| -v- \|w\| \|x\| \|y\| \|z\|`<br><br>and the word in dashes opens guessed letters :<br><br>`i  n  v  _  _  i  _  n`<br><br>7. Type ‘a’ letter and press Enter<br><br>8. System shows alphabet with ‘cross out’ letter a:<br><br>`-a- \|b\| \|c\| \|d\| \|e\| \|f\| \|g\| \|h\| -i- \|j\| \|k\| \|l\| \|m\| -n- \|o\| \|p\| \|q\| \|r\| \|s\| \|t\| \|u\| -v- \|w\| \|x\| \|y\| \|z\|`<br><br>and the word in dashes opens guessed letters :<br><br>`i  n  v  a  _  i  _  n`<br><br>9. Type ‘o’ letter and press Enter<br><br>10. System shows alphabet with ‘cross out’ letter s:<br><br>`-a- \|b\| \|c\| \|d\| \|e\| \|f\| \|g\| \|h\| -i- \|j\| \|k\| \|l\| \|m\| -n- \|o\| \|p\| \|q\| \|r\| -s- \|t\| \|u\| -v- \|w\| \|x\| \|y\| \|z\|`<br><br>and the word in dashes opens guessed letters :<br><br>`i  n  v  a  s  i  _  n`<br><br>11. Type ‘o’ letter and press Enter |
| **Expected** | System shows:<br><br>`Congratulations! You Win! The word was 'invasion'`<br>`Your earn 12 points`<br>`Do you want to play again ?`<br>`Enter 'p' to Play New Game`<br>`Enter 'h' to Look At Highscores`<br>`Enter 'q' to Exit Game` |
| Status | Passed |
| Comments | Proceeded as expected |

### TC 2.2 Play Game and Loose

| | |
|---|---|
| Use Case to test | Use Case 2. Play Game |
| Description | The test is supposed to check the game process which leading to a loose |

| | |
|---|---|
| | situation. |
| Precondition | The game should be started and the game interface is shown according to TC1.1 with default player |
| Constant expectation | Line `Type here: is always available` |
| Test steps | 1. Type ‘k’ letter<br><br>2. A part of Hangman is drawn:<br><br>`\|`<br>`\|`<br>`\|_`<br><br>System shows alphabet with ‘cross out’ letter k:<br><br>`\|a\| \|b\| \|c\| \|d\| \|e\| \|f\| \|g\| \|h\| \|i\| \|j\| -k- \|l\| \|m\| \|n\| \|o\| \|p\| \|q\| \|r\| \|s\| \|t\| \|u\| \|v\| \|w\| \|x\| \|y\| \|z\|`<br><br>and the word to guess still in dashes<br><br>`_ _ _ _ _ _ _ _`<br><br>3. Type ‘l’ letter and press Enter<br><br>4. A part of Hangman is drawn:<br><br>`\|`<br>`\|`<br>`\|`<br>`\|`<br>`\|_`<br><br>System shows alphabet with ‘cross out’ letter l:<br><br>`\|a\| \|b\| \|c\| \|d\| \|e\| \|f\| \|g\| \|h\| \|i\| \|j\| -k- -l- \|m\| \|n\| \|o\| \|p\| \|q\| \|r\| \|s\| \|t\| \|u\| \|v\| \|w\| \|x\| \|y\| \|z\|`<br><br>and the word to guess still in dashes<br><br>`_ _ _ _ _ _ _ _`<br><br>5. Type ‘m’ letter and press Enter<br><br>6.A part of Hangman is drawn:<br><br>`  __`<br>`\|   \|`<br>`    \|`<br>`    \|`<br>`    \|`<br>`    \|_`<br><br>System shows alphabet with ‘cross out’ letter m: |

```
||a| |b| |c| |d| |e| |f| |g| |h| |i| |j| -k- -l- -m- |n| |o| |p| |q| |r| |s| |t| |u| |v| |w| |x| |y| |z|
```

and the word to guess still in dashes

```
 _ _ _ _ _ _ _ _
```

7. Type ‘p’ letter and press Enter

8. A part of Hangman is drawn:

```
    __
   |  |
   o  |
  /   |
      |
      |_
```

System shows alphabet with ‘cross out’ letter p:

```
|a| |b| |c| |d| |e| |f| |g| |h| |i| |j| -k- -l- -m- |n| |o| -p- |q| |r| |s| |t| |u| |v| |w| |x| |y| |z|
```

and the word to guess still in dashes

```
 _ _ _ _ _ _ _ _
```

9. Type ‘r’ letter and press Enter

10. A part of Hangman is drawn:

```
    __
   |  |
   o  |
  /|\ |
      |
      |_
```

System shows alphabet with ‘cross out’ letter r:

```
|a| |b| |c| |d| |e| |f| |g| |h| |i| |j| -k- -l- -m- |n| |o| -p- |q| -r- |s| |t| |u| |v| |w| |x| |y| |z|
```

and the word to guess still in dashes

```
 _ _ _ _ _ _ _ _
```

11. Type ‘t’ letter and press Enter

**Expected**

System shows:

12. The whole Hangman is drawn:

```
    __
   |  |
   o  |
  /|\ |
   (\ |
```

| | |
|---|---|
| | `    \|_`<br><br>System shows:<br><br>`Sorry.. You loose. The word was 'invasion'`<br>`Your final score is 0`<br>`Do you want to play again ?`<br>`Enter 'p' to Play New Game`<br>`Enter 'h' to Look At Highscores`<br>`Enter 'q' to Exit Game` |
| Status | Failed |
| Comments | Instead of expected YOU LOOSE message system continued playing the game but malfunctioning. |

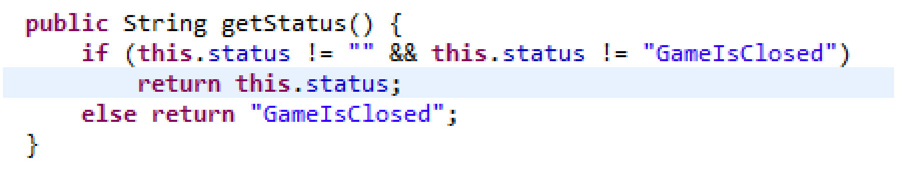
Test traceability matrix and success

| Use Case ID | Use Case Name | TestCase ID | Status |
|---|---|---|---|
| UC1 | Start Game | TC1.1 | Passed |
| UC1 | Start Game | TC1.2 | Passed |
| UC2 | Play Game | TC2.1 | Passed |
| UC2 | Play Game | TC2.2 | Failed |
| COVERAGE & SUCCESS | | | 3 of 4 Passed / 75%<br>1 of 4 Failed /25% |

## 3. Automated unit test

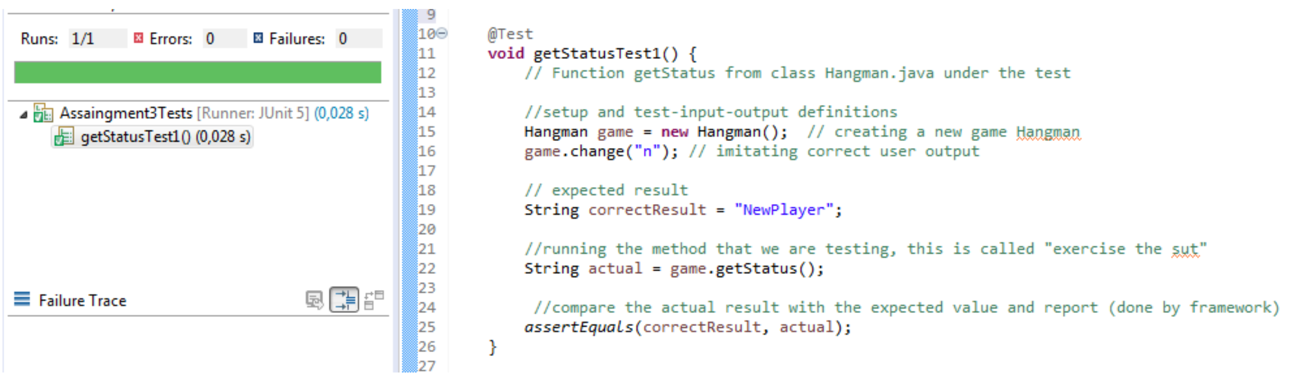
**Sut-method getStatus() from class Hangman under the test**

the original method:



```
public String getStatus() {
    if (this.status != "" && this.status != "GameIsClosed")
        return this.status;
    else return "GameIsClosed";
}
```

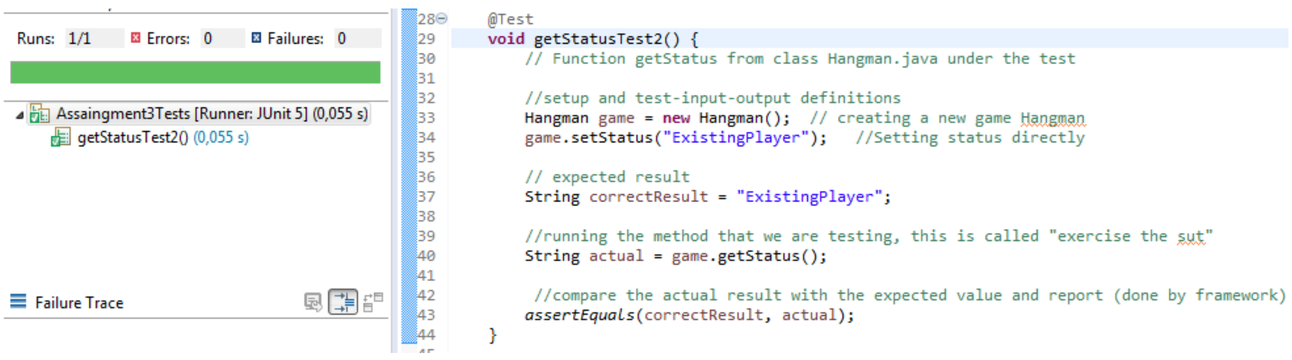
Testing status change emulating user input



```
Runs: 1/1    Errors: 0    Failures: 0

Assaingment3Tests [Runner: JUnit 5] (0,028 s)
    getStatusTest1() (0,028 s)

Failure Trace
```

```java
    @Test
    void getStatusTest1() {
        // Function getStatus from class Hangman.java under the test

        //setup and test-input-output definitions
        Hangman game = new Hangman();  // creating a new game Hangman
        game.change("n"); // imitating correct user output

        // expected result
        String correctResult = "NewPlayer";

        //running the method that we are testing, this is called "exercise the sut"
        String actual = game.getStatus();

         //compare the actual result with the expected value and report (done by framework)
        assertEquals(correctResult, actual);
    }
```

Testing status change setting status directly

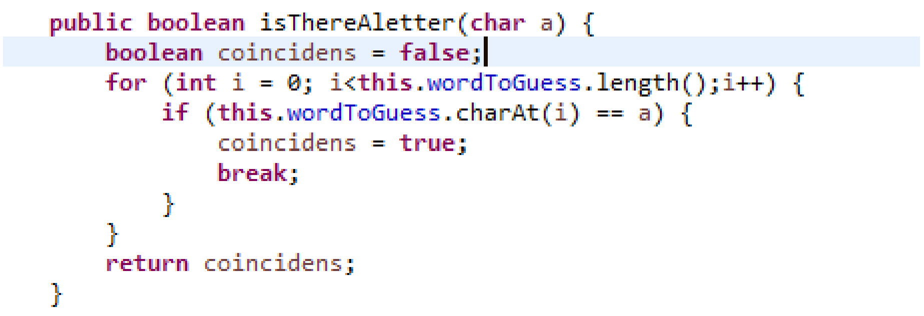


```
Runs: 1/1    Errors: 0    Failures: 0

Assaingment3Tests [Runner: JUnit 5] (0,055 s)
    getStatusTest2() (0,055 s)

Failure Trace
```

```java
    @Test
    void getStatusTest2() {
        // Function getStatus from class Hangman.java under the test

        //setup and test-input-output definitions
        Hangman game = new Hangman();  // creating a new game Hangman
        game.setStatus("ExistingPlayer");   //Setting status directly

        // expected result
        String correctResult = "ExistingPlayer";

        //running the method that we are testing, this is called "exercise the sut"
        String actual = game.getStatus();

         //compare the actual result with the expected value and report (done by framework)
        assertEquals(correctResult, actual);
    }
```

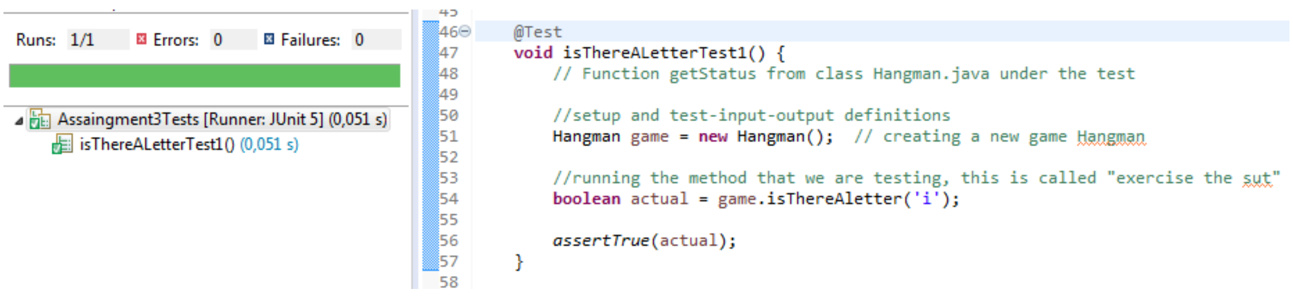
**Sut-method isThereALetterTest2() from class Hangman under the test**

The original function is



```java
public boolean isThereAletter(char a) {
    boolean coincidens = false;
    for (int i = 0; i<this.wordToGuess.length();i++) {
        if (this.wordToGuess.charAt(i) == a) {
            coincidens = true;
            break;
        }
    }
    return coincidens;
}
```

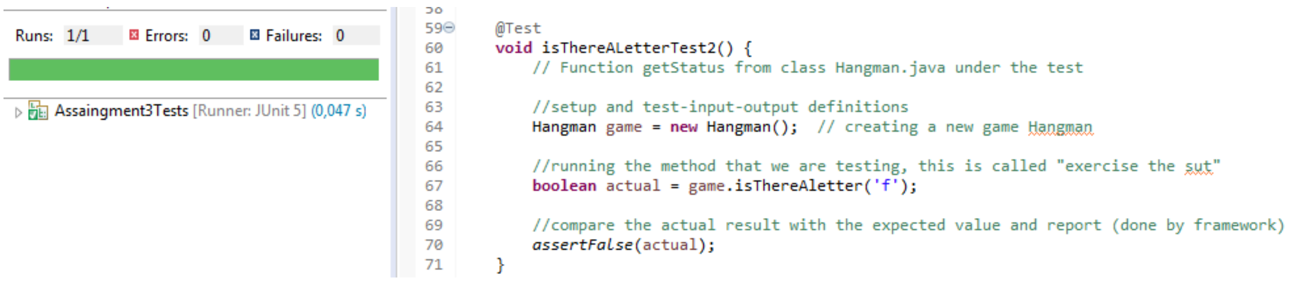
Testing for existing the letter ‘i’ in the given word



```
Runs: 1/1    Errors: 0    Failures: 0

Assaingment3Tests [Runner: JUnit 5] (0,051 s)
    isThereALetterTest1() (0,051 s)
```

```java
    @Test
    void isThereALetterTest1() {
        // Function getStatus from class Hangman.java under the test

        //setup and test-input-output definitions
        Hangman game = new Hangman();  // creating a new game Hangman

        //running the method that we are testing, this is called "exercise the sut"
        boolean actual = game.isThereAletter('i');

        assertTrue(actual);
    }
```

Testing for not-existing the letter ‘f’ in the given word

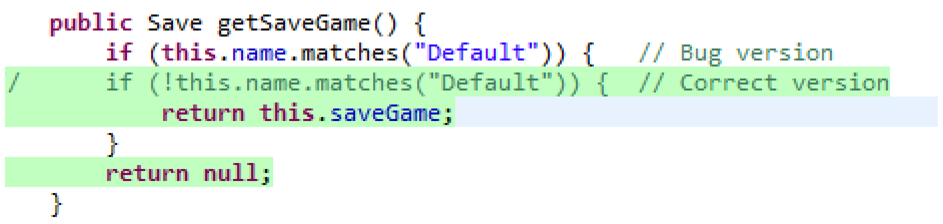


```
Runs: 1/1    Errors: 0    Failures: 0

Assaingment3Tests [Runner: JUnit 5] (0,047 s)
```

```
@Test
void isThereALetterTest2() {
    // Function getStatus from class Hangman.java under the test

    //setup and test-input-output definitions
    Hangman game = new Hangman();  // creating a new game Hangman

    //running the method that we are testing, this is called "exercise the sut"
    boolean actual = game.isThereAletter('f');

    //compare the actual result with the expected value and report (done by framework)
    assertFalse(actual);
}
```

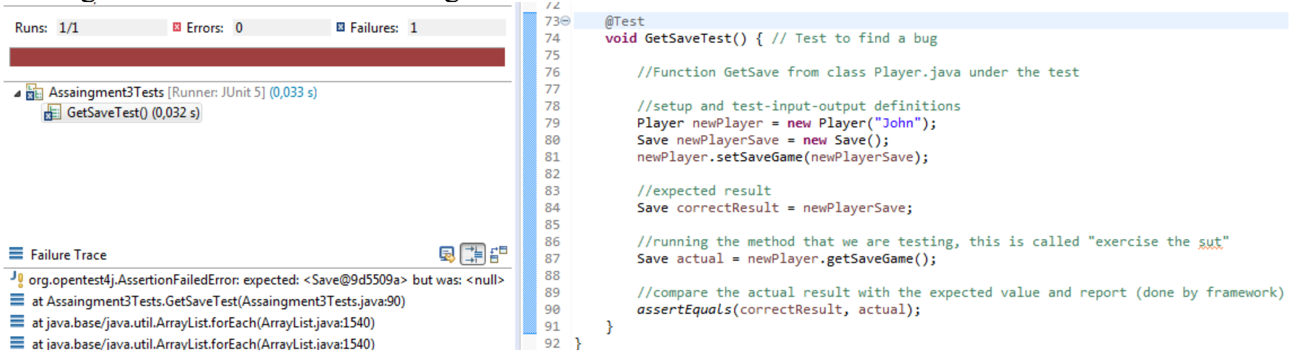
**Sut-method GetSave from class Player.java under the test**

The original mehtod



```
    public Save getSaveGame() {
        if (this.name.matches("Default")) {   // Bug version
/       if (!this.name.matches("Default")) {  // Correct version
            return this.saveGame;
        }
        return null;
    }
```

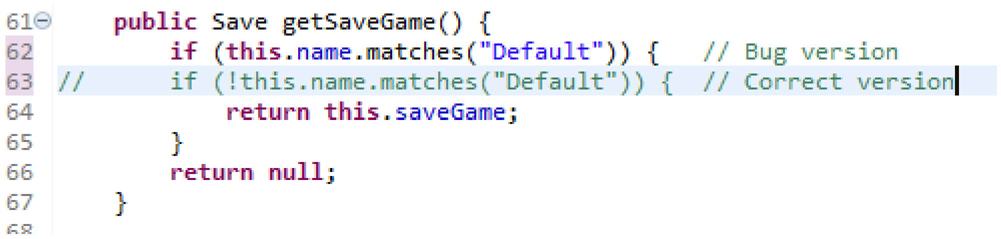
Testing for collection correct load game information.



```
Runs: 1/1    Errors: 0    Failures: 1

Assaingment3Tests [Runner: JUnit 5] (0,033 s)
    GetSaveTest() (0,032 s)

Failure Trace
org.opentest4j.AssertionFailedError: expected: <Save@9d5509a> but was: <null>
at Assaingment3Tests.GetSaveTest(Assaingment3Tests.java:90)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1540)
at java.base/java.util.ArrayList.forEach(ArrayList.java:1540)
```

```
@Test
void GetSaveTest() { // Test to find a bug

    //Function GetSave from class Player.java under the test

    //setup and test-input-output definitions
    Player newPlayer = new Player("John");
    Save newPlayerSave = new Save();
    newPlayer.setSaveGame(newPlayerSave);

    //expected result
    Save correctResult = newPlayerSave;

    //running the method that we are testing, this is called "exercise the sut"
    Save actual = newPlayer.getSaveGame();

    //compare the actual result with the expected value and report (done by framework)
    assertEquals(correctResult, actual);
}
}
```

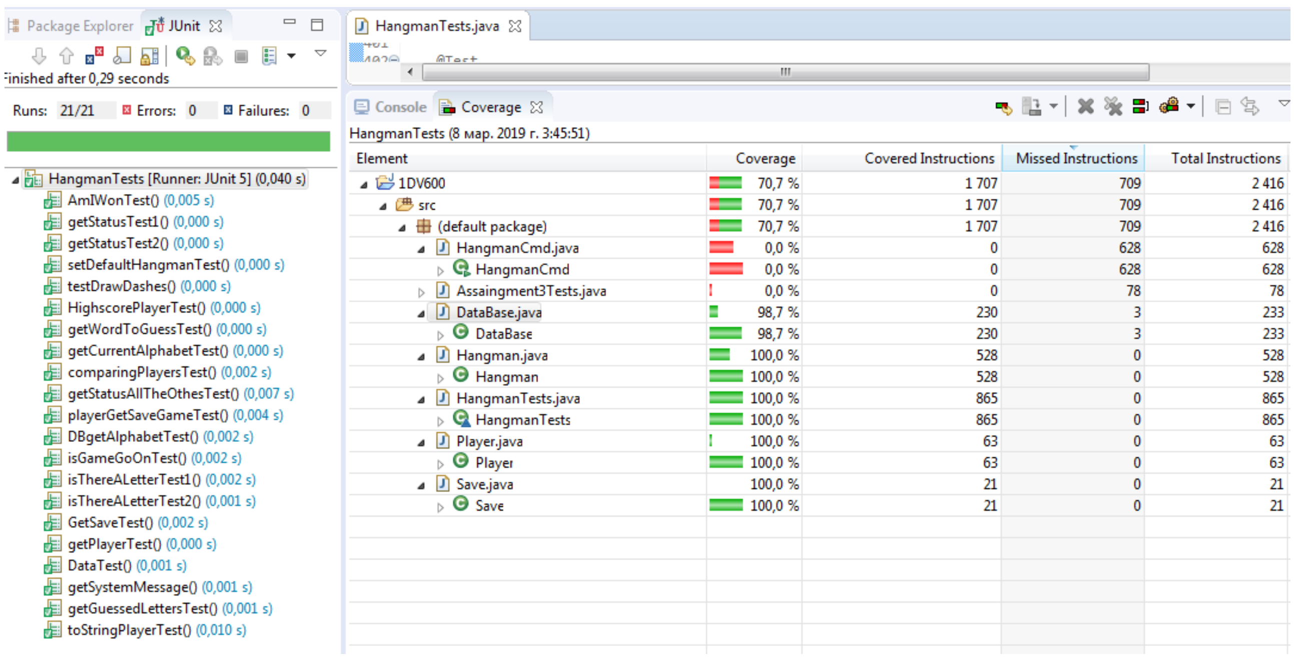
The reason of failure– bug within the method



```
    public Save getSaveGame() {
        if (this.name.matches("Default")) {   // Bug version
//      if (!this.name.matches("Default")) {  // Correct version
            return this.saveGame;
        }
        return null;
    }
```

**All the other tests**

All the other tests consist of 423 lines of code and I assume that trying to put it here is pointless. All the tests located in the file HangmanTests.java in the GitHub repository. However, you can find a


Finished after 0,29 seconds

Runs: 21/21 Errors: 0 Failures: 0

HangmanTests [Runner: JUnit 5] (0,040 s)
- AmIWonTest() (0,005 s)
- getStatusTest1() (0,000 s)
- getStatusTest2() (0,000 s)
- setDefaultHangmanTest() (0,000 s)
- testDrawDashes() (0,000 s)
- HighscorePlayerTest() (0,000 s)
- getWordToGuessTest() (0,000 s)
- getCurrentAlphabetTest() (0,000 s)
- comparingPlayersTest() (0,002 s)
- getStatusAllTheOthesTest() (0,007 s)
- playerGetSaveGameTest() (0,004 s)
- DBgetAlphabetTest() (0,002 s)
- isGameGoOnTest() (0,002 s)
- isThereALetterTest1() (0,002 s)
- isThereALetterTest2() (0,001 s)
- GetSaveTest() (0,002 s)
- getPlayerTest() (0,000 s)
- DataTest() (0,001 s)
- getSystemMessage() (0,001 s)
- getGuessedLettersTest() (0,001 s)
- toStringPlayerTest() (0,010 s)

HangmanTests (8 мар. 2019 г. 3:45:51)

| Element | Coverage | Covered Instructions | Missed Instructions | Total Instructions |
|---|---|---|---|---|
| 1DV600 | 70,7 % | 1 707 | 709 | 2 416 |
| src | 70,7 % | 1 707 | 709 | 2 416 |
| (default package) | 70,7 % | 1 707 | 709 | 2 416 |
| HangmanCmd.java | 0,0 % | 0 | 628 | 628 |
| HangmanCmd | 0,0 % | 0 | 628 | 628 |
| Assaingment3Tests.java | 0,0 % | 0 | 78 | 78 |
| DataBase.java | 98,7 % | 230 | 3 | 233 |
| DataBase | 98,7 % | 230 | 3 | 233 |
| Hangman.java | 100,0 % | 528 | 0 | 528 |
| Hangman | 100,0 % | 528 | 0 | 528 |
| HangmanTests.java | 100,0 % | 865 | 0 | 865 |
| HangmanTests | 100,0 % | 865 | 0 | 865 |
| Player.java | 100,0 % | 63 | 0 | 63 |
| Player | 100,0 % | 63 | 0 | 63 |
| Save.java | 100,0 % | 21 | 0 | 21 |
| Save | 100,0 % | 21 | 0 | 21 |

screenshot of coverage report bellow. Coverage report is mage using unbuilt coverage tool in Java Eclipse Editor

## 4. Reflections

The program has been tested accordingly to the plan. Manual testing helped to find mistakes in the interface behavior whereas static code inspection allowed to fix some within the source code. Despite the fact that the back end source code was fully covered (100%) doing automated tests there may still be some mistakes as the possibilities for input are unlimited. Therefore, the testing process helped a lot increasing the confidence in the program.