Software Engineering: Process and Tools

Software Unit Testing Report

Semester 2, 2020

Kieu Nguyen
Student ID: 101354326

Table of Contents

1. Executive Summary

This paper is a report for an assignment in PRT582- Software Engineering: Process and Tools to present how a small game Hangman is developed by applying Testing Driven Development (TDD). This paper will overview TDD process, the game hangman requirements and then how TDD is implemented to write the hangman program.

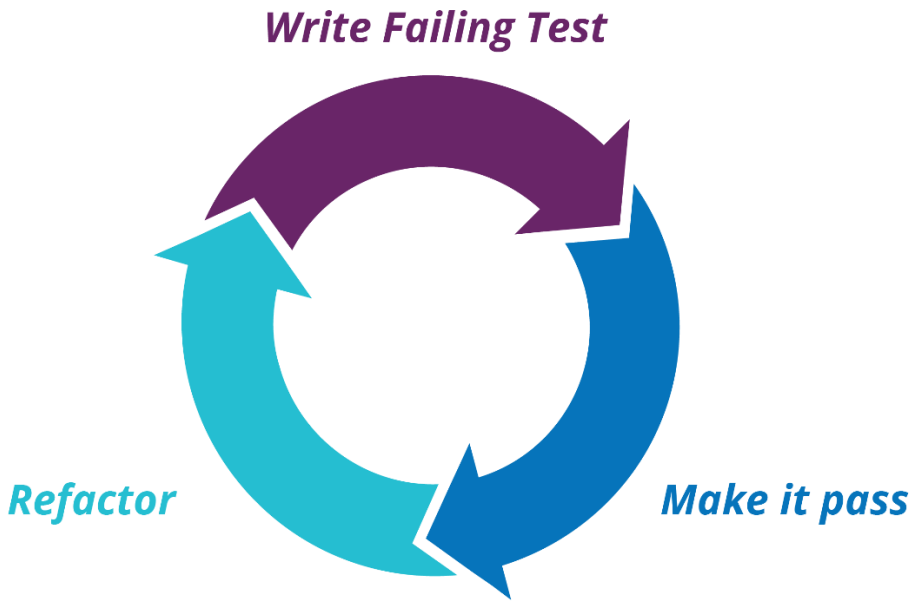
The link for code and test: https://github.com/melkong194/Hangman

2. Context and Background

2.1 Introduction of Testing Driven Development

Testing Driven Development (TDD) is a software development process that relies on the repetition of a very short development cycle:

- Firstly, writing a test case to test the requirement of the function.
- Secondly, produce the code to pass the test.
- Finally, refactoring the code.



This process of TDD is to test each logic of the program so TDD ensure the program meets requirements, preventing the cost for big bug and the possible solution is the minimum amount of code to pass the test. This help to avoid duplicating in writing code, clearer and better design of program. TDD becomes the good practice for developers because it helps developers get accomplishment feelings when each test pass or show them problems when testing results shows improper execution of functions.

Not similarity to traditional testing, TDD is 100% coverage test. It can be applied to test every logic in our program.

2.2 Hangman Game

How to play Hangman Game:

One word will be generated randomly. The word is hidden and only shown by a number of '*' to present of each letters of the words. Based on the length of the word, user will guess the word by entering a letter in turn. Initially, user has 5 lives and their lives will be decreased by one when making a wrong guess. Once user makes correct guess, the letter of that guess in all position will be revealed. If user makes guess correctly all letters in the words, they will win the game. However, when they got 5 times of incorrect guess, their lives will be 0 and lose the game.

Steps to create Hangman program: some steps are broken by the rules of the game.

- Pick a word randomly.
- Allow user to input
- Validate the input
- Compare the input to the word
- Check win or not.

2.3 Report Introduction

This report is to present the process of applying TDD to develop the hangman program. The Junit test cases will be specified the behaviors of functions in java code of hangman program.

3. Design

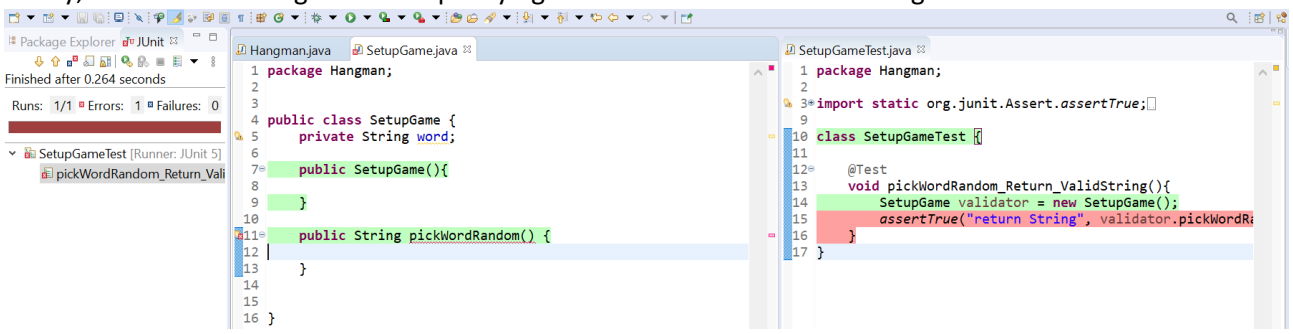
3.1 Requirements

Step	Functions	Requirements
1	Pick a word randomly	Pick a word in a list of words randomly and return a String.
2	Allow user to input	Allow user inputs a String
3	Validate the input	If the input is an alphabet then returns true, else returns false.
4	Compare the input to the word	If input is in words and not guessed before then returns true, else return false.
5	Check win or not	If all letters in the word are guessed then return true, else return false.

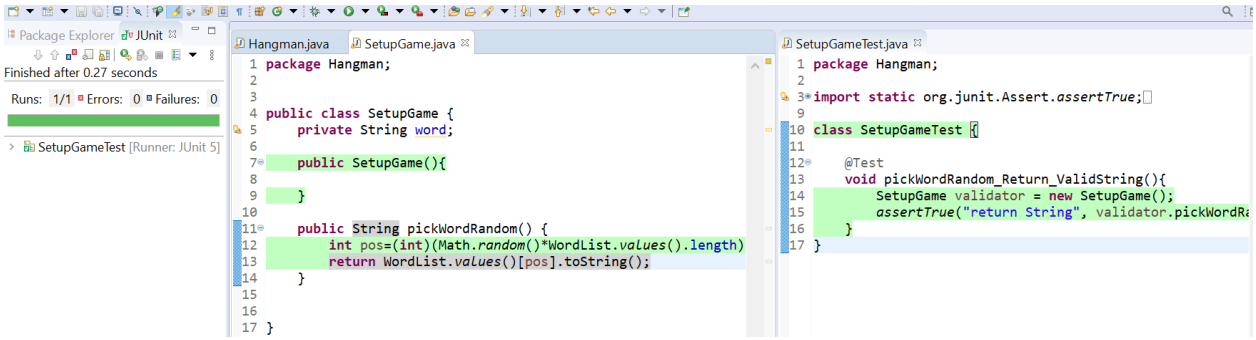
3.2 Design and approach

FUNCTION 1: Pick a word randomly

Firstly, we create a failing test with specifying the function will return a String.

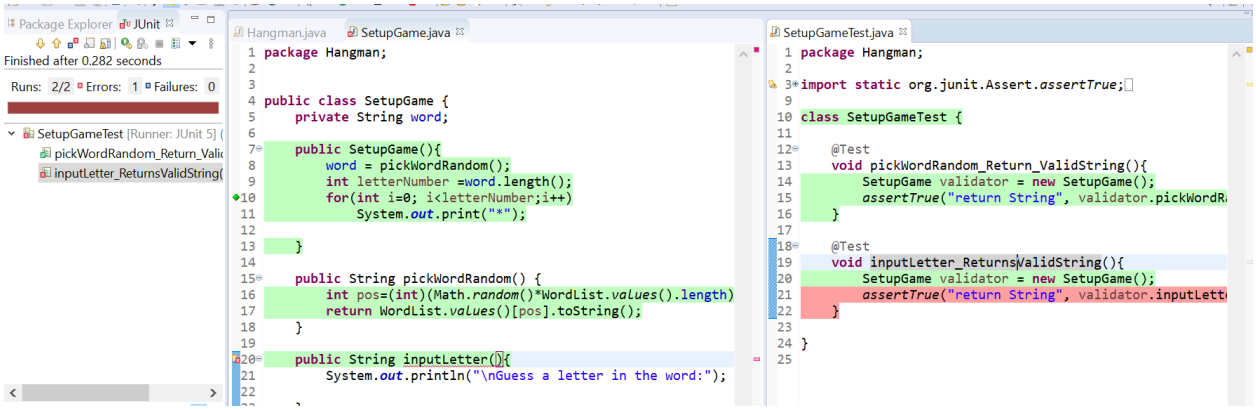


Then we produce the code to pass the test.

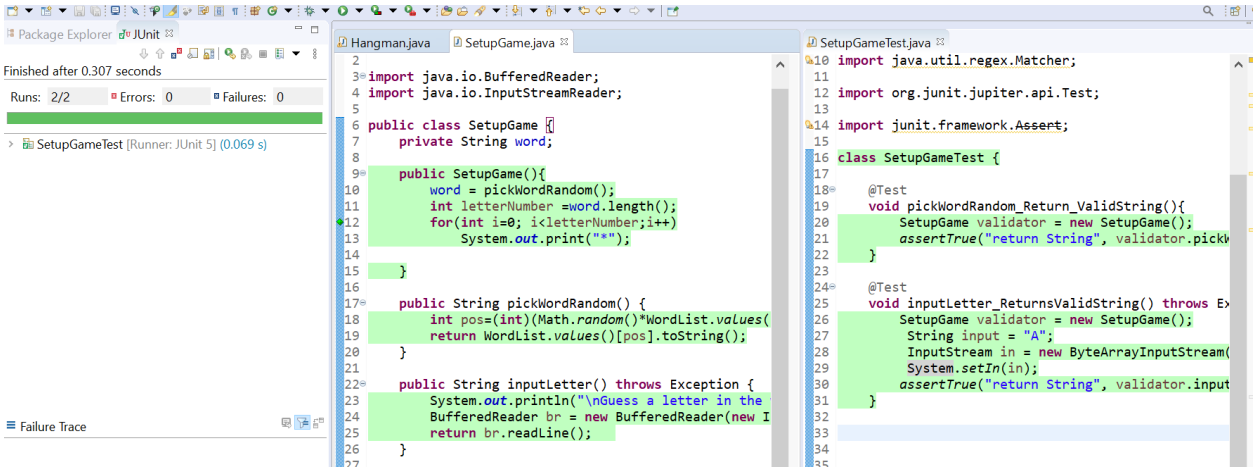


FUNCTION 2: Allow user to input

Function 2 is similar to the function 1, we create a failing test with specifying the function will return a String.



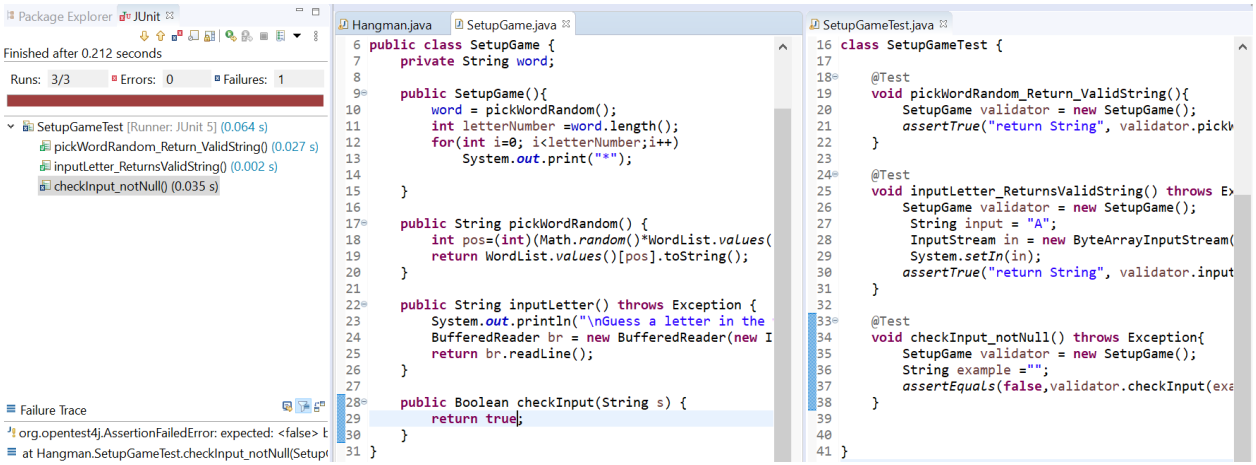
Then we produce the code to pass the test.



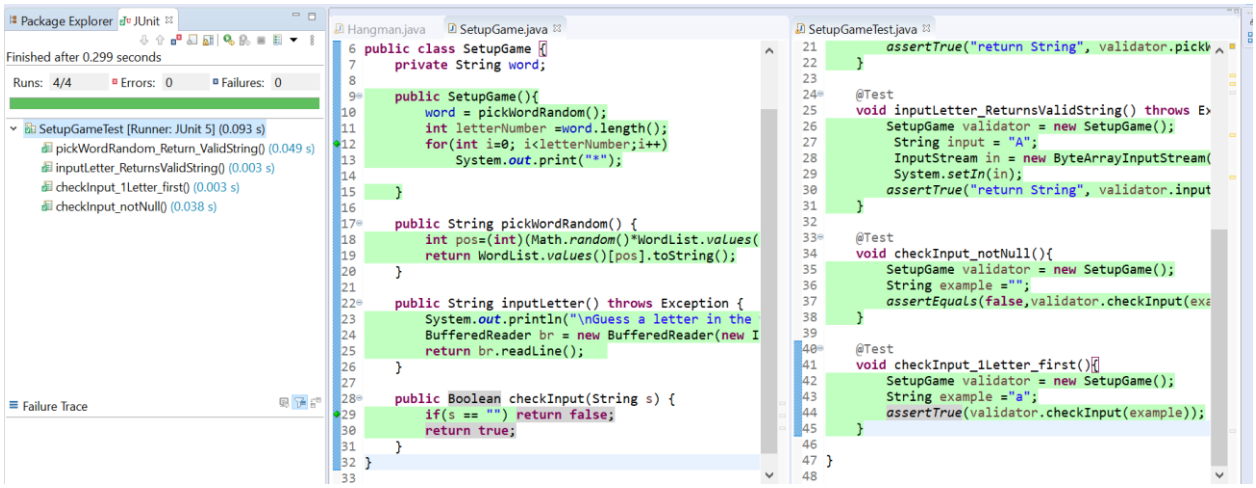
FUNCTION 3: Validate the input

The conditions in the function 3 requires more test cases to specify the requirements.

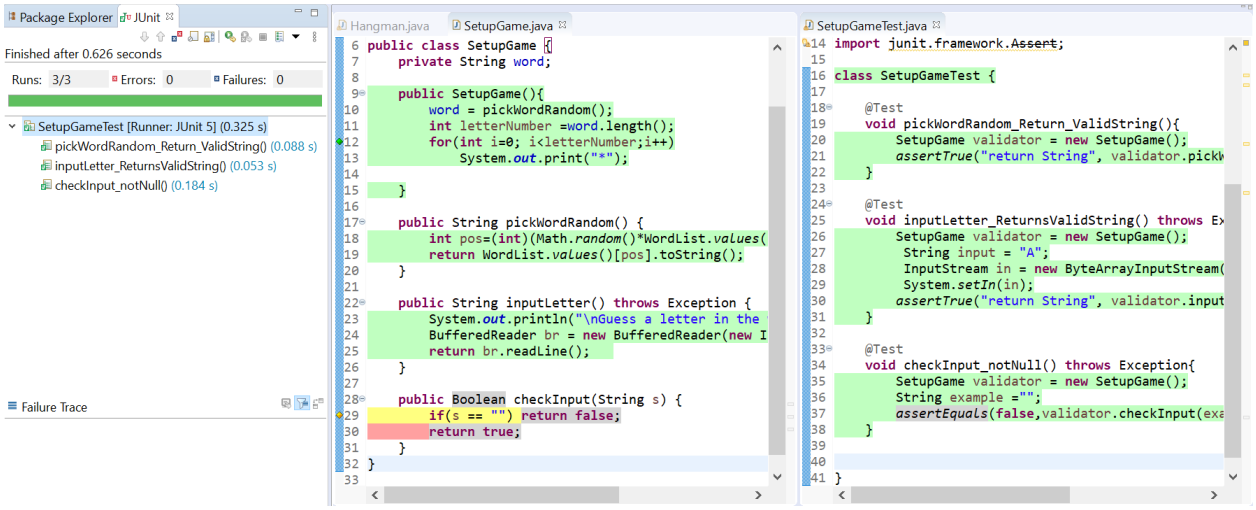
Apply the TDD, we also produce the failing test case firstly to ensure no empty input.



Then, minimum code to pass the test.

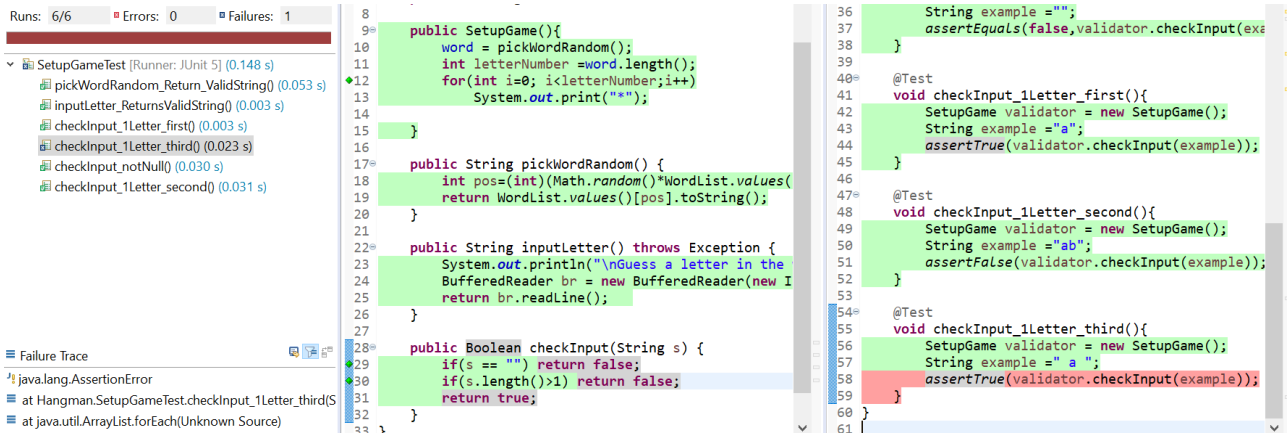


Testing for a single letter input, it works properly and pass the test.

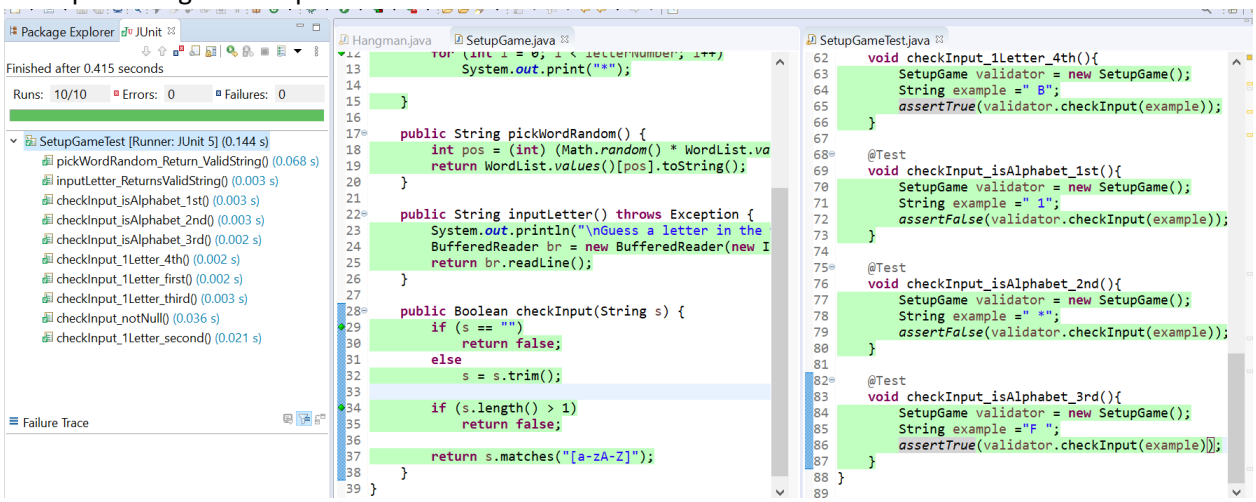


By repeating the process of TDD, test cases are written before the produce of code, the function is tested for other requirements, such as, testing for more than 1 letter inputs, special letters, digits, leading space before inputs. When we produce code to pass all test cases, we have a complete function.

Before producing code:

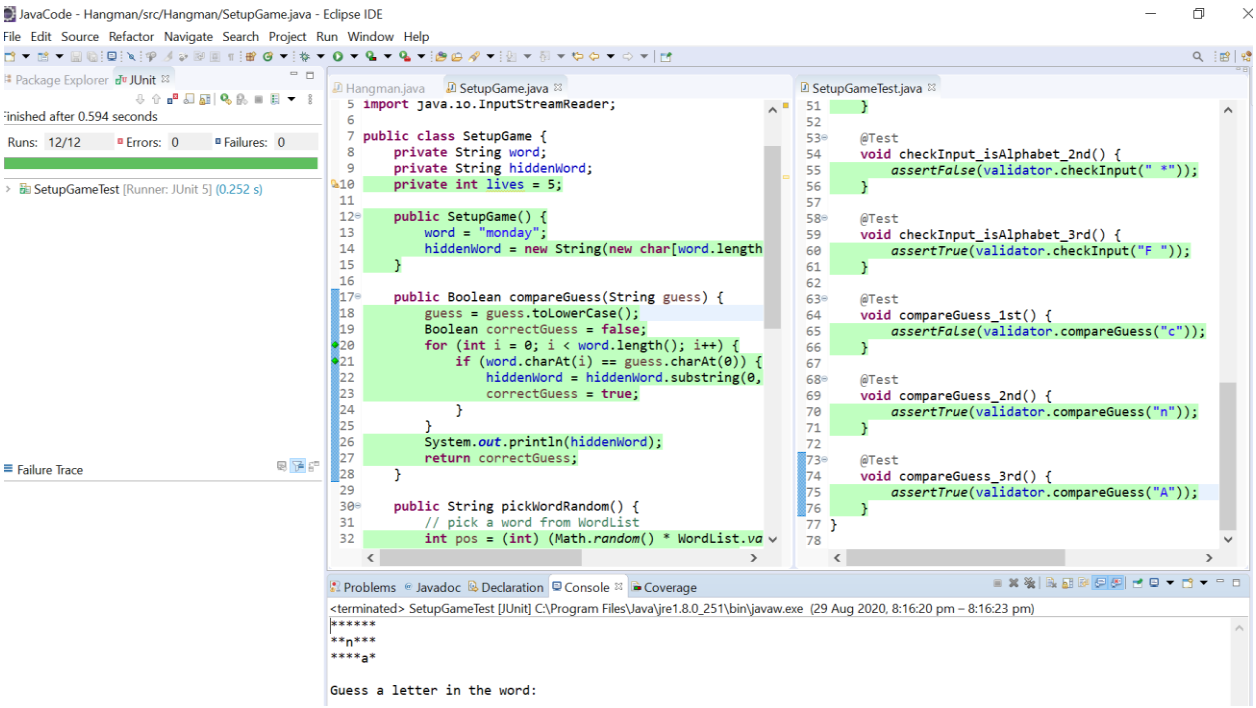


After producing code to pass all tests:



FUNCTION 4 and 5:

The process for function 4 and 5 is similar to 1,2,3.



REFACTORING THE CODE:

Refactoring the test cases:

@Before

SetupGameTest.java

```
40    @Test
41    void checkInput_1Letter_first(){
42        SetupGame validator = new SetupGame();
43        String example ="a";
44        assertTrue(validator.checkInput(example));
45    }
46
47    @Test
48    void checkInput_1Letter_second(){
49        SetupGame validator = new SetupGame();
50        String example ="ab";
51        assertFalse(validator.checkInput(example));
52    }
53
54    @Test
55    void checkInput_1Letter_third(){
56        SetupGame validator = new SetupGame();
57        String example =" B ";
58        assertTrue(validator.checkInput(example));
59    }
60
61    @Test
62    void checkInput_1Letter_4th(){
63        SetupGame validator = new SetupGame();
64        String example =" B";
65        assertTrue(validator.checkInput(example));
66    }
```

@After

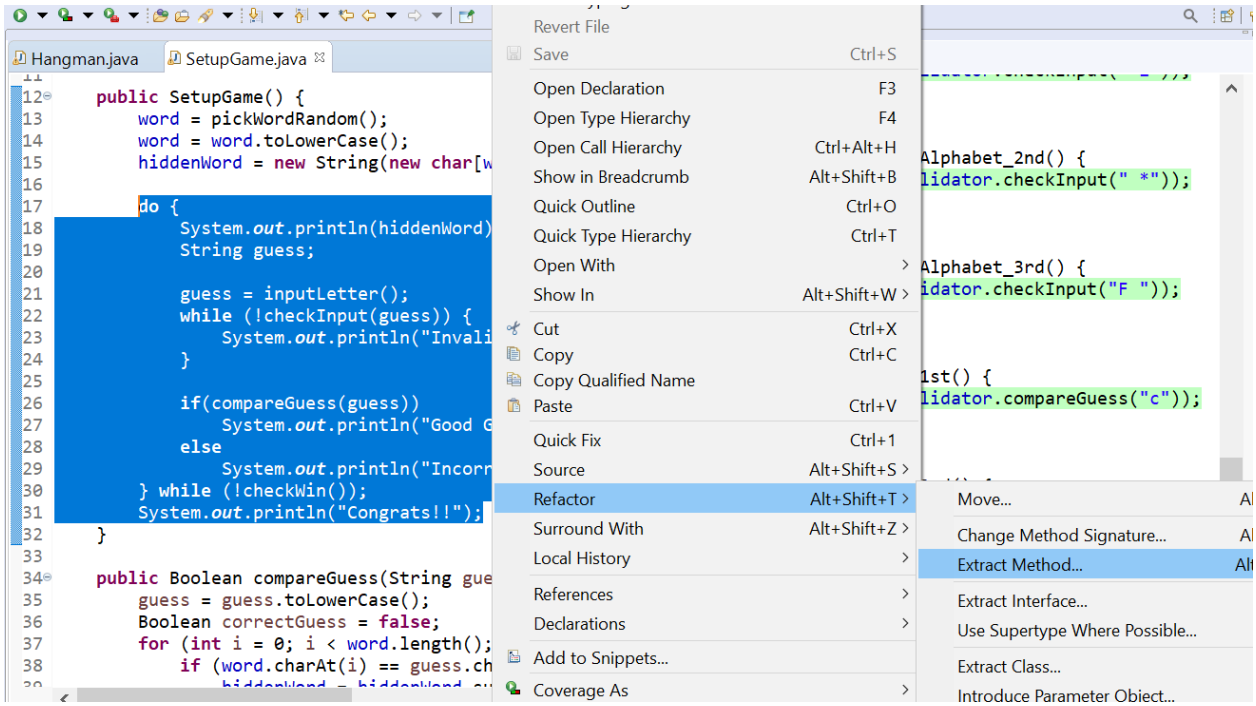
SetupGameTest.java

```
15
16 class SetupGameTest {
17     SetupGame validator = new SetupGame();
18
19     @Test
20     void pickWordRandom_Return_ValidString(){
21         assertTrue("return String", validator.pickW
22     }
23
24     @Test
25     void inputLetter_ReturnsValidString() throws E
26         String input = "A";
27         InputStream in = new ByteArrayInputStream(
28         System.setIn(in);
29         assertTrue("return String", validator.input
30     }
31
32     @Test
33     void checkInput_notNull(){
34         String example ="";
35         assertEquals(false,validator.checkInput(exa
36     }
37
38     @Test
39     void checkInput_1Letter_first(){
40         String example ="a";
41         assertTrue(validator.checkInput(example));
42     }
```

Refactoring the Programming Code:

Because after user got win or lose, the game will be repeated. Hence, we refactor the code and extract it into new method to reuse it and get better design for program.

@Before refactoring



@After Refactoring.

Hangman.java SetupGame.java

```
6
7  public class SetupGame {
8      private String word;
9      private String hiddenWord;
10     private int lives;
11
12     public SetupGame() {
13         while (true) {
14             System.out.println("++++NEW GAME++++++");
15             word = pickWordRandom().toLowerCase();
16             lives = 5;
17             hiddenWord = new String(new char[word.length()]).replace("\0", "*");
18             play();
19         }
20     }
21
22     public void play() {
23         do {
24             System.out.println("==== LIVES:" + lives + " ====");
25             System.out.println("The word: "+hiddenWord);
26
27             // Validate input
28             String guess = inputLetter();
29             while (!checkInput(guess)) {
30                 System.out.println("Invalid input! Please make guess again!!");
31                 guess = inputLetter();
32             }
33
34             // Result for each letter input.
35             if (compareGuess(guess)) {
36                 System.out.println("Good Guess!!\n");
37             } else {
```

4. Reference

https://en.wikipedia.org/wiki/Test-driven_development