PRT-582 Software Engineering Process and Tools

Assignment 1

Name: Aafiyaben Ahmadbhai Polra

Student ID: s328620

Table of Contents

Hangman Program Using TDD and output:

Hangman.java

```
package se.lexicon;

import java.util.Arrays;

public class Hangman{

    String secretWord="Approx";

    char[] dashes=new char[secretWord.length()];

    int numberOfGuesses=8;

    int numOfExistedLetters=0;

    int numOfFaultGuesses=0;

    boolean win=false;

    public  Hangman()

    {

        Arrays.fill(dashes,'-');

    }

    public void showDashesToPlayer(){

        for (int i=0;i<secretWord.length();i++){

            System.out.print(dashes[i]);

        }

    }
```

```
//

//Send a message to player to guess a letter

public void askPlayerToGuess(int numberOfGuesses){

    System.out.println("\nGuess a letter"+ ", you have "+numberOfGuesses +" left");

}

public boolean checkGussLetter(String  letter) {

    boolean letterIsExist=false;

    for (int i = 0; i < secretWord.length(); i++) {

        if (secretWord.charAt(i)==letter.charAt(0))

        {

            letterIsExist=true;

            dashes[i]=letter.charAt(0);

            numOfExistedLetters +=1;

        }//End if

    }//End loop



    if (letterIsExist)

    {

        System.out.println("The letter is in the secret word");

    }
```

```
        else {
            System.out.println("The letter is not in the secret word");
            numOfFaultGuesses +=1;
        }
        return letterIsExist;
    }//End method

    public boolean isPlayerWin(int numOfExistedLetters) {
        if (numOfExistedLetters==secretWord.length())
        { return true; }
        else {return false;}
    }
}
```

HangmanApp.java

```
package se.lexicon;
import java.util.Arrays;
import java.util.Scanner;
public class App
{
    public  static Scanner scanner=new Scanner(System.in);
```

```
public static void main( String[] args )
{
   boolean keepPlaying = true;


   while (keepPlaying) {
      startGame();
      keepPlaying = DoYouPlayAgain();
   }
}//End main
private static boolean DoYouPlayAgain() {
   System.out.println("Select 1 to play again and 2 to quit\n");
   int playerSelect=scanner.nextInt();
   switch (playerSelect){
      case 1:
         return true;
      case 2:
         return false;
   }
   return false;
```

```
}
public static void startGame(){
    Hangman hangman=new Hangman();
    StringBuilder alreadyGuessedLetters=new StringBuilder();
    int guessNum=0;
    while (guessNum < hangman.numberOfGuesses){
        hangman.showDashesToPlayer();


        // Ask player to guess a letter.
        hangman.askPlayerToGuess(hangman.numberOfGuesses-guessNum);


        //Player input a guessed letter
        String inputUser = scanner.nextLine();
        if (inputUser.length() < 1)
        {
            //not a guess
            continue;
        }
        else if (inputUser.length() == 1)
        {
```

```
            //letter check
            // check if the guessed letter guessed by the player and ask him to chose another
one.
            if(alreadyGuessedLetters.toString().contains(inputUser))
            {
                System.out.println("Player already guessed this letter :"+
alreadyGuessedLetters.toString());
                continue;
            }
            // Save guessed letters in a string.
            alreadyGuessedLetters.append(inputUser).append(' ');
            // check if the guessed letter is exist in the secret word.
            hangman.checkGussLetter(inputUser);
            if (hangman.isPlayerWin(hangman.numOfExistedLetters)){
                System.out.println("You win!");
                break;
            }
        }
        else
        {
            // Word Check
```

```
            if(hangman.secretWord.length()< inputUser.length()){
                System.out.println("the word is shorter than the secret word");
            }
            else{
                if (hangman.secretWord.contentEquals(inputUser)){
                    System.out.println("You win!");
                    break;
                }
                else{
                    System.out.println("Incorrect guess");
                }
            }
        }
        guessNum++;
    }//End while loop
  }
}
```

- One Word generated.
- Player will be represented with 6 number of blank spaces and player need to find the missing word.
- If the player is chosen a word which existed in the answer then the output comes as a below picture.

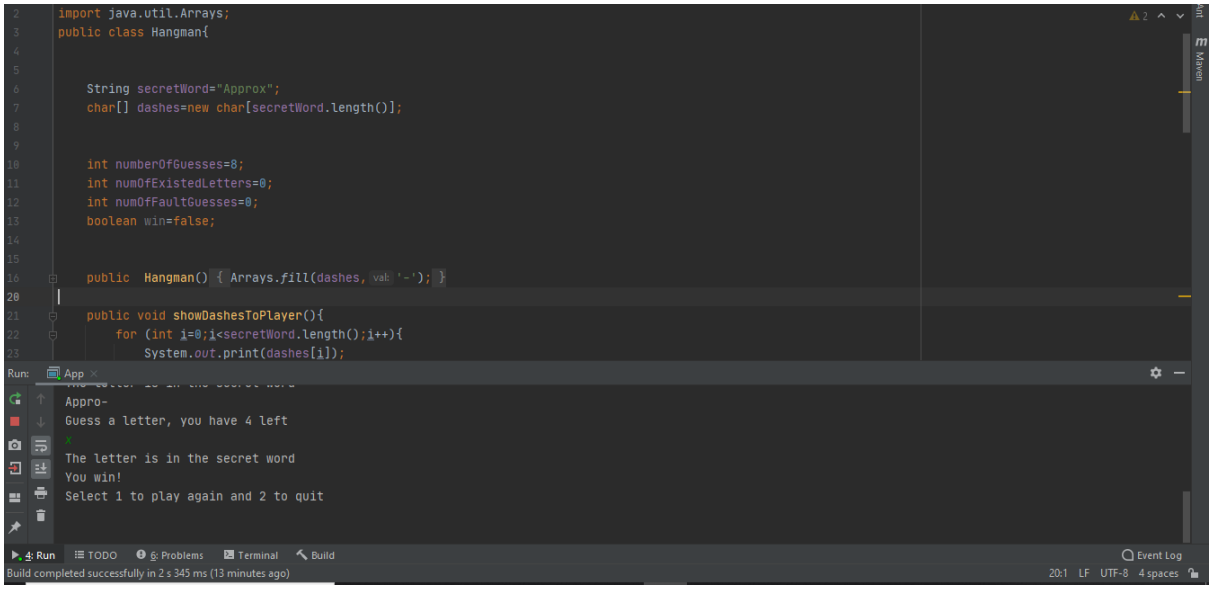

Figure 1

- If every time the player guesses a wrong word than the player's life will be deducted.
- Also, player need to find the missing word before the player's life becomes zero.

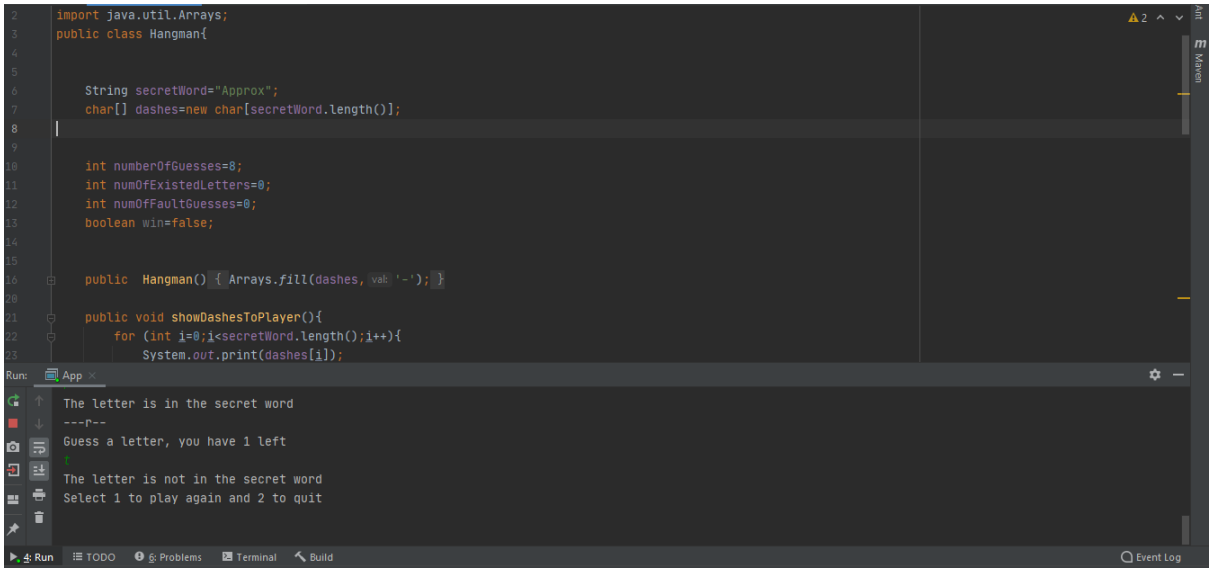

Figure 2

Refactor Code:

Code Smells

Firstly, the name of the variable must be complete always. Variables are self-explained so it will not become confusing. There is a different type of code smells exists in the current code.

- If we want to change the functionality a little bit at one place than it will give ripple effect in the other parts of the code.

So, I have created a separate method for generating the random word. Now, the variables are self-explanatory and also, there are bloaters in the code which means the number of methods is causing the confusion in understanding the code. So, I have simplified the code with the use of generic methods.

Refactor code: The best time to consider refactor is before adding any update or any new features. The below picture shows the automated refactoring. We converted expressions and in refactor option we selected introduce variable from the main menu or context menu.

If we found more than one occurrence expression than, **replace the occurrence** or **replace all occurrence** from the **multiple occurrence** list.

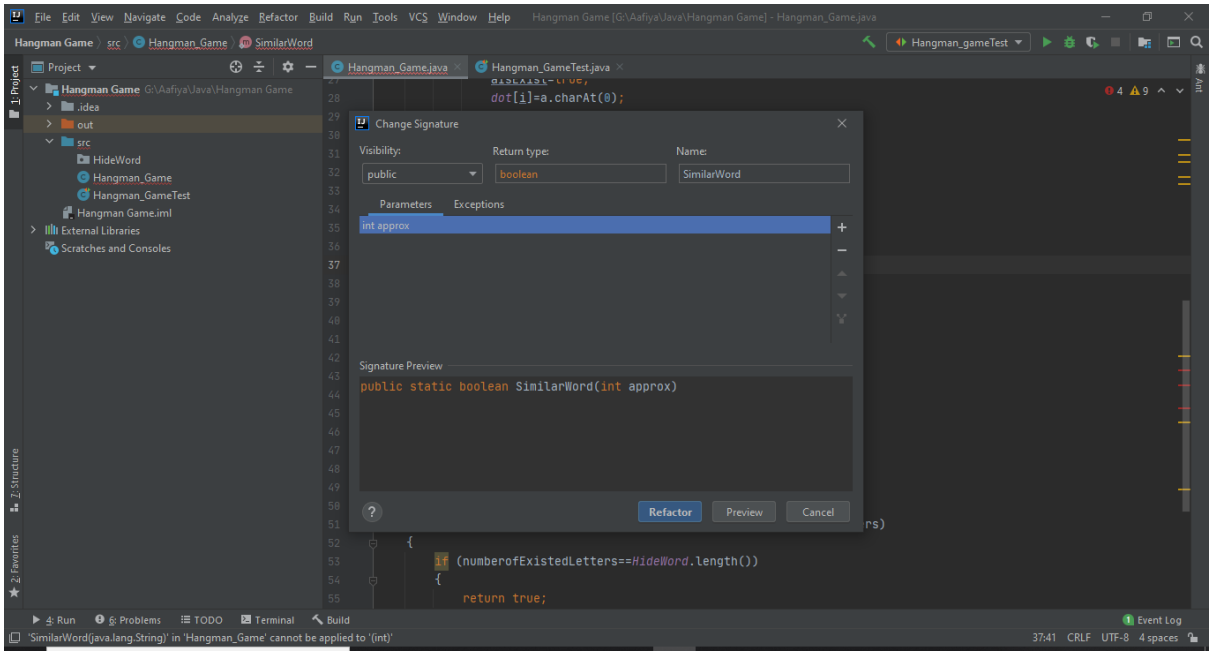

Figure 3

Refactor the test code which shows in the below image.

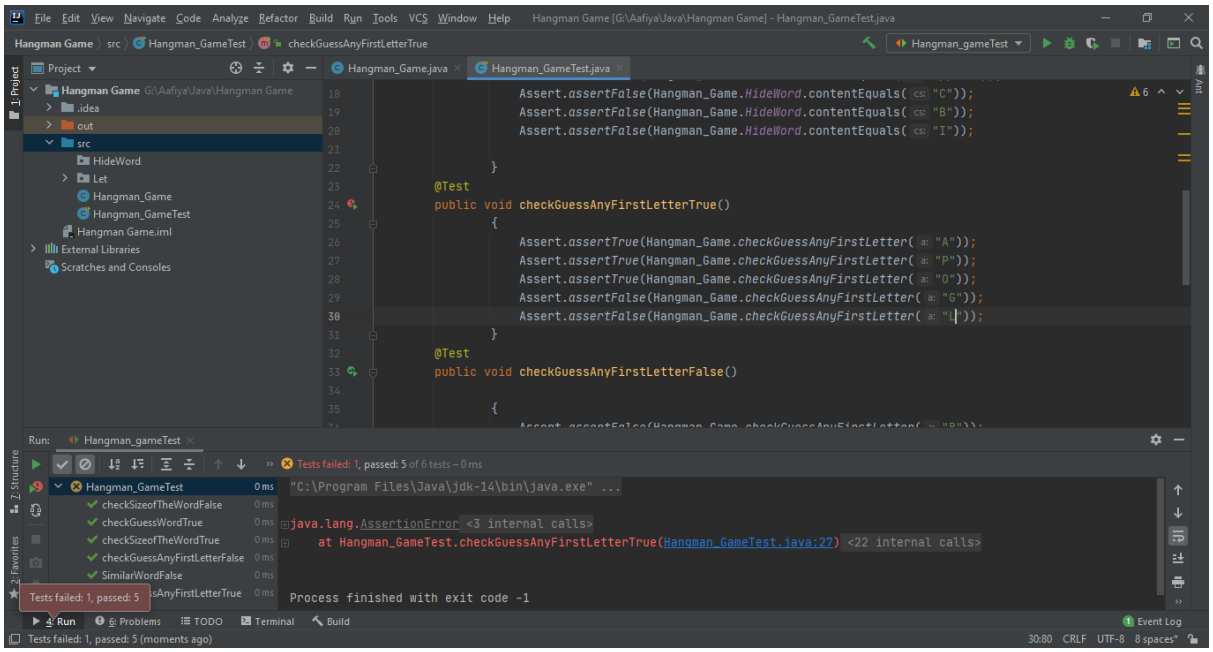

Figure 4

Refactor Test passed successfully.

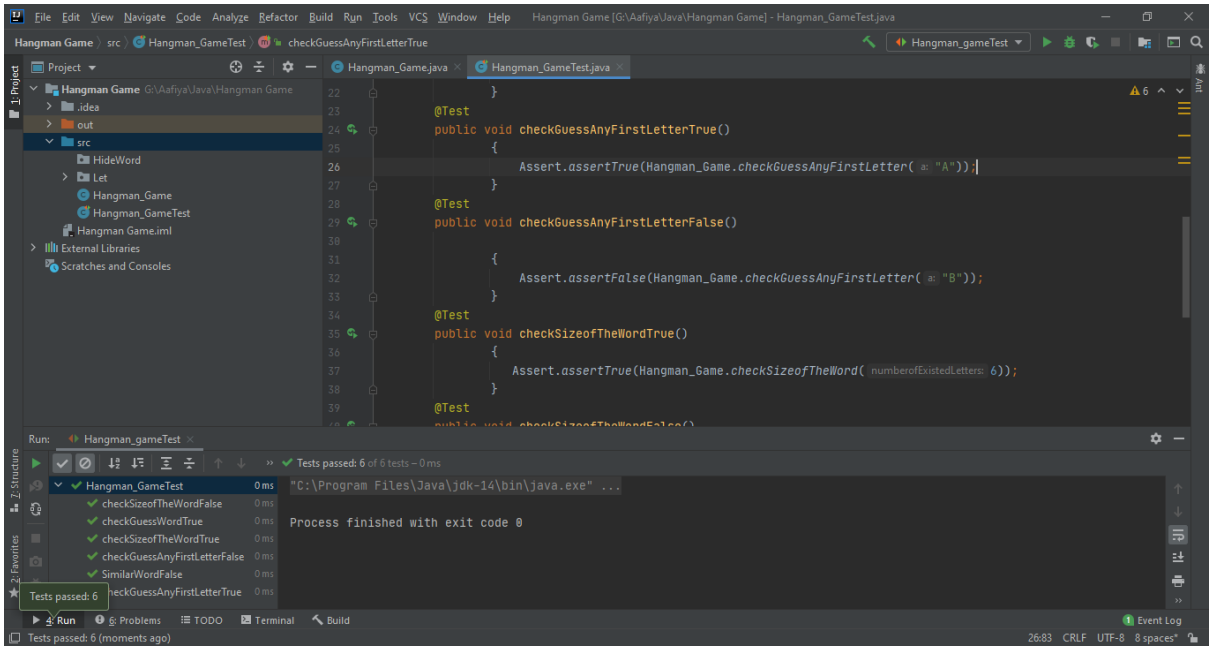

Figure 5

Before Refactor the Test Code:

```
package se.lexicon;
import org.junit.Assert;
import org.junit.Test;



public class AppTest
{
   Hangman hangman=new Hangman();
   @Test
   public void checkGuessLetterTrue()
   {
      Assert.assertTrue(hangman.checkGussLetter("s"));
   }

   @Test
   public void checkGuessLetterFalse()
   {
      Assert.assertFalse(hangman.checkGussLetter("m"));
   }




   @Test
   public void checkGuessWordTrue()
   {
      Assert.assertTrue(hangman.secretWord.contentEquals("Approx"));
   }

   @Test
   public void checkGuessWordFalse()
   {
      Assert.assertFalse(hangman.secretWord.contentEquals("Approximately"));
```

```
    }


    @Test
    public void isPlayerWinTrue()
    {
        Assert.assertTrue(hangman.isPlayerWin(6));
    }

    @Test
    public void isPlayerWinFalse()
    {
        Assert.assertFalse(hangman.isPlayerWin(5));
    }

}
```

After the refactoring, see the code:

```
import org.junit.Test;

import org.junit.Assert;



public class Hangman_GameTest {

    private static String HideWord;

    Hangman_Game hangman_game= new Hangman_Game();
```

```
@Test
public void checkGuessWordTrue()
    {
       Assert.assertTrue(Hangman_Game.HideWord.contentEquals("Approx"));
       Assert.assertFalse(Hangman_Game.HideWord.contentEquals("C"));
       Assert.assertFalse(Hangman_Game.HideWord.contentEquals("B"));
       Assert.assertFalse(Hangman_Game.HideWord.contentEquals("I"));

    }
@Test
public void checkGuessAnyLetterTrue()
    {
       Assert.assertTrue(Hangman_Game.checkGuessAnyLetter("A"));
       Assert.assertFalse(Hangman_Game.checkGuessAnyLetter("M"));
       Assert.assertFalse(Hangman_Game.checkGuessAnyLetter("G"));
       Assert.assertFalse(Hangman_Game.checkGuessAnyLetter("L"));
    }

@Test
public void checkSizeofTheWordTrue()
    {
      Assert.assertTrue(Hangman_Game.checkSizeofTheWord(6));
      Assert.assertFalse(Hangman_Game.checkSizeofTheWord(5));
```

```
            Assert.assertFalse(Hangman_Game.checkSizeofTheWord(4));
            Assert.assertFalse(Hangman_Game.checkSizeofTheWord(8));

        }
    @Test
    public void checkSizeofTheWordFalse()
        {
            Assert.assertFalse(Hangman_Game.checkSizeofTheWord(10));
            Assert.assertTrue(Hangman_Game.checkSizeofTheWord(6));
        }

    @Test
    public void SimilarWordFalse()
        {
            Assert.assertFalse(Hangman_Game.SimilarWord("Approximately"));


        }
}
```

Create a GitHub directory for our Assignment (including word or pdf document and programming code)

Link: https://github.com/polraaafiya/Assignment1_PRT582_2020

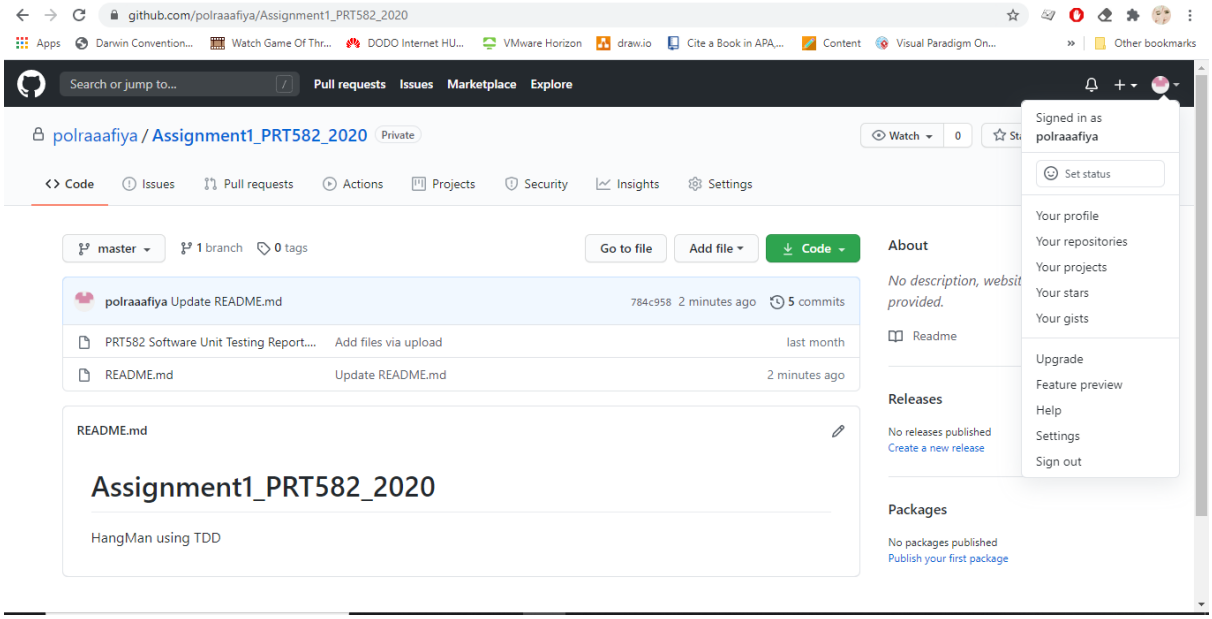

Figure 6

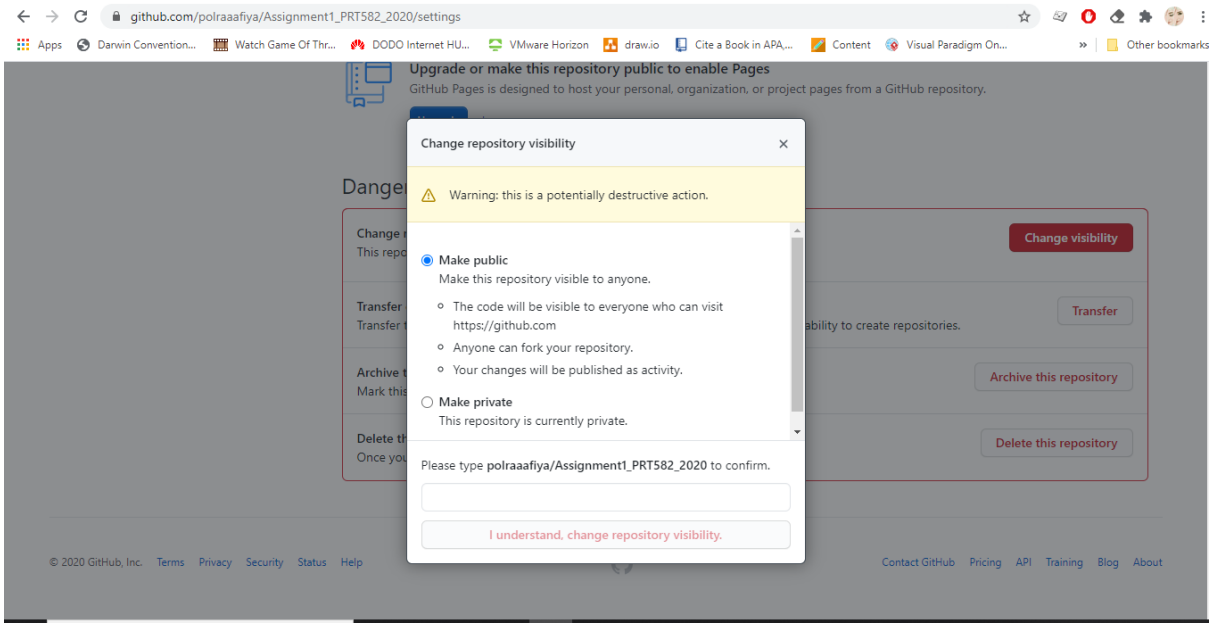

Figure 7

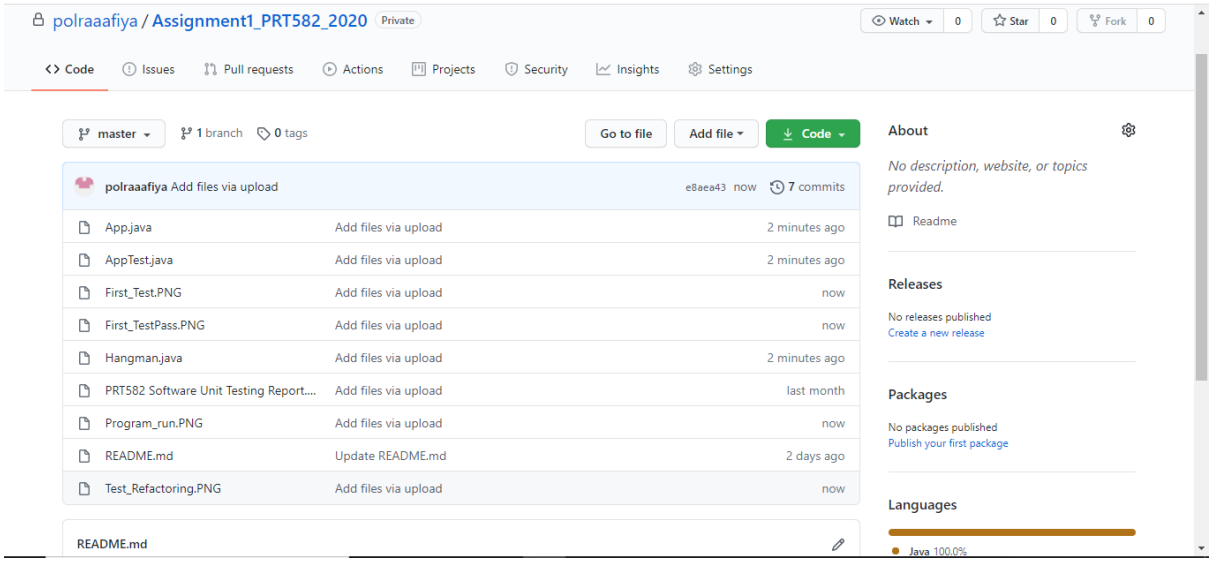

Figure 8

How TDD implemented to create our program:

We have used TDD approach for creating the hangman game. We could easily test the method of any word generator to check that a random word is being generated or not. I have test a code by running the first test case which is failed. The below first test shows the error like **Java.lang.NullPointerException** and the error message have much explanation and reason. I have attached the screenshots of test.

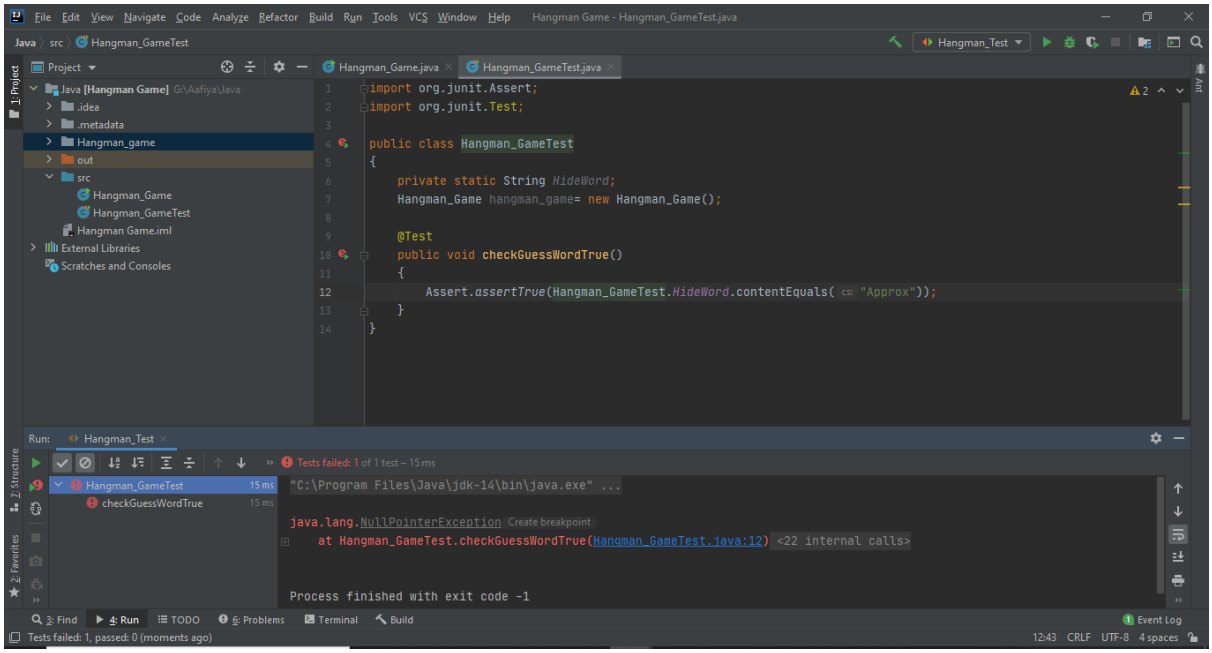

Figure 9

Running the Tests:

To follow a TDD approach, typically we go through a cycle of Red – Green - Refactor. We need to run a test and see it fail (show red), Created the simplest code to make the test pass (show green), and then refactor the code so our test will be stay green and our code is sufficiently clean always.

The first step is to run the test and see it fail.

In the below picture it's shows that I've use IntelliJ IDEA's features to create the simplest Hangman Game and implementation of the function we're testing.

Creating First Test Code:

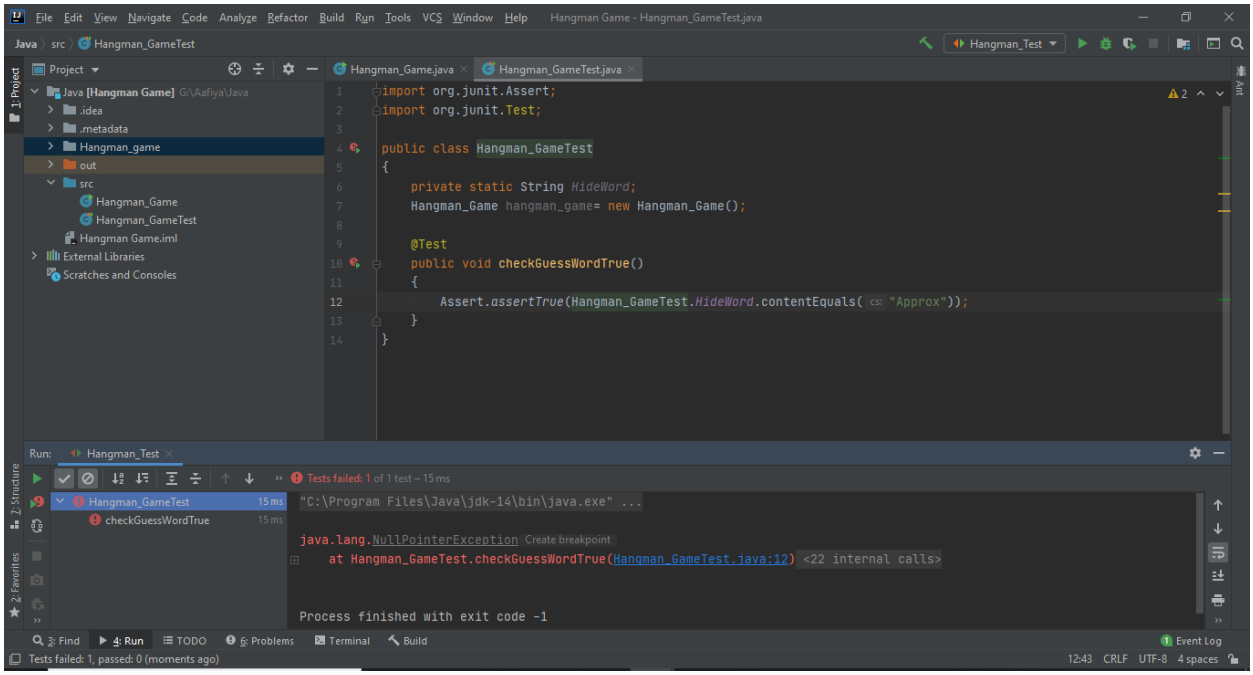

Figure 10

Implementing the Code:

It shows implemented the tested function which required a type and compile without any error.

```
public static java.util.Scanner Scanner = new Scanner(System.in);
public static String HideWord = "Approx";
private static char[] dot = new char[HideWord.length()];
private static int numberofExistedLetters = 0;

int getNumberofExistedLetters = 6;
boolean win=false;

public  Hangman_Game()
{
    Arrays.fill(dot, val: '.');

}

public static boolean checkGuessAnyLetter(String a)
{
    boolean aIsExist =false;
    for (int i = 0; i < HideWord.length(); i++)
    {
        if (HideWord.charAt(i) == a.charAt(0) )
        {
            aIsExist=true;
            dot[i]=a.charAt(0);
            numberofExistedLetters += 1;
        }
    }
    return aIsExist;
}
```

Figure 11

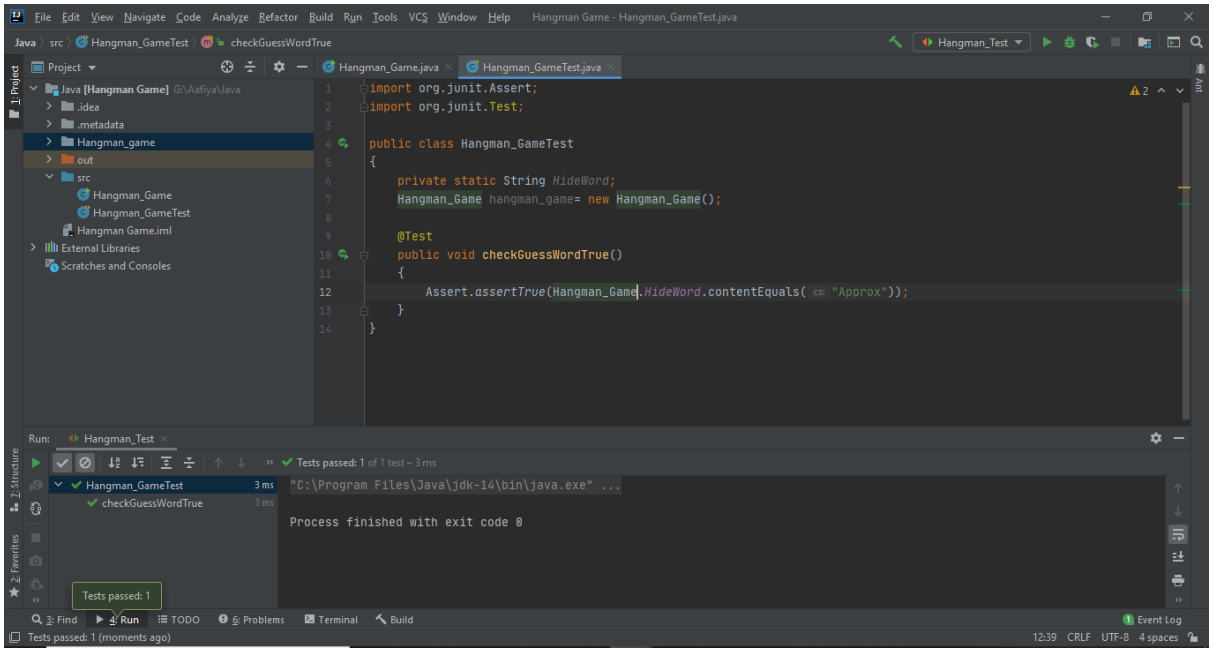

Figure 12

As we can see that our first test case passed successfully. And, if it is still showing red color than make the required changes until the test passes.

How refactoring has been done, the code smell, issues and solutions:

The refactoring code has been done. Also, the latest code posted on GitHub repository. Check the commit history. The main issue is that the variables were not self-explanatory. Whenever any small problem occurs then I need to debug and complete code to find out the issue. There is also a different method which are formed according to their functions. Recurrence is the most useful terms in software development and also, it played a major role in software maintenance. In this article, we will look at the definition of re-use literature, how we can use ensure our codebase which optimized for renewal. Moreover, we will go through the repetitions from beginning to end which shows the stretch and importance of this whole magnificent process. These all things help us to achieve simple and easy coding. There are some developers, convinced with the process of TDD code and it need to be rewritten which is easily understood and give the lowest number of comments.

"The smell of code" When the code needs to be repeated which is called smell. This statement is True but not real.

Reference:

Sohaib Hamioud, Fadila Atil. (2012). Model-Driven Java Code Refactoring. Model-Driven Java Code Refactoring, 28.

Singh, S. (2002). A systematic literature review: Refactoring for disclosing code smells in object oriented software. A systematic literature review: Refactoring for disclosing code smells in object oriented software, 13.

FOWLER, M., BECK, K., BRANT, J., OPDYKE, W. & ROBERTS, D. (1999) *Refactoring: Improving the Design of Existing Code*, Addison Wesley.

Genggeng Liu, Chuanshumin Hu. (2019). Refactoring Java Code for Automatic API Generation. Refactoring Java Code for Automatic API Generation, 29.

MENS, T. & TOURWE, T. (2004) A survey of software refactoring. *Software Engineering, IEEE Transactions on,* 30, 126-139.