**PRT 580 Software Engineering: Process and Tools**

ASSIGNMENT 1

Name: Nihal Akbarali Maredia

Student ID:s327992

Hangman Game TEsting

Table of Contents

[Hangman using TDD. 2](#_Toc49275729)

[Refactored code. 5](#_Toc49275730)

[Code Smells 5](#_Toc49275731)

[Updated Code: 5](#_Toc49275732)

[Creating a Github. 8](#_Toc49275733)

[How TDD has been implemented to create your program. 10](#_Toc49275734)

[How refactoring has been done, the code smell, issues and solutions. 14](#_Toc49275735)

[References 15](#_Toc49275736)

# GitHub Link: <https://github.com/nihalmaredia/Hangman>

# Hangman using TDD.

**Hangman.java**

|  |
| --- |
| import java.io.BufferedReader; import java.io.File; import java.io.FileReader; import java.io.IOException; import java.util.ArrayList; import java.util.Random; import java.util.stream.IntStream;  public class Hangman {  String secretWord;  StringBuilder Guess;  ArrayList<Character> lastGuess = new ArrayList<>();   int maximumTries = 6;  int yourTry = 0;   ArrayList<String> words = new ArrayList<>();   public Hangman() {  initializeStream();  secretWord = selectWord();  Guess = loadCurrentGuess();  }   public void initializeStream() {  try {  File inFile = new File("words.txt");  FileReader fileReader = new FileReader(inFile);  BufferedReader bufferedFileReader = new BufferedReader(fileReader);  String currentLine = bufferedFileReader.readLine();  while (currentLine != null) {  words.add(currentLine);  currentLine = bufferedFileReader.readLine();  }  bufferedFileReader.close();  fileReader.close();  }  catch (IOException e) {  System.*out*.println("Could not init streams");  }  }   public String selectWord() {  Random rand = new Random();  int wordIndex = Math.*abs*(rand.nextInt())%words.size();  return words.get(wordIndex);  }   public StringBuilder loadCurrentGuess() {  StringBuilder current = new StringBuilder();  IntStream.*range*(0, secretWord.length() \* 2).forEach(i -> {  if (i % 2 == 0) current.append("\_");  else current.append(" ");  });  return current;  }   public String getFormalCurrentGuess() {  return "Current guess: " + Guess.toString();  }   public boolean gameover() {  if (didWeWin()) {  System.*out*.println();  System.*out*.println("Congratulation. You won the game");  } else {  if (!didWeLose()) {  return didWeWin() || didWeLose();  }  System.*out*.println();  System.*out*.println("Sorry. You lost the game." +  " The secret word was: " + secretWord + ".");  }  return didWeWin() || didWeLose();  }   public boolean didWeWin() {  String guess;  guess = getCondensedCurrentGuess();  return guess.equals(secretWord);  }   public String getCondensedCurrentGuess() {  String guess = Guess.toString();  return guess.replace(" ", "");  }   public boolean didWeLose() {  return yourTry >= maximumTries;  }   public boolean isGuessedAlready(char guess) {  return lastGuess.contains(guess);  }   public boolean playGuess(char guess) {  boolean isItAGoodGuess = false;  lastGuess.add(guess);  for (int i=0;i<secretWord.length();i++)  if (secretWord.charAt(i) == guess) {  Guess.setCharAt(i \* 2, guess);  isItAGoodGuess = true;  }  if (!isItAGoodGuess) yourTry++;  return isItAGoodGuess;  }   } |

# HangmanApplication.java

|  |
| --- |
| import java.io.IOException; import java.util.Scanner;  public class HangmanApplication {  public static void main(String[] args) throws IOException {  Scanner sc = new Scanner(System.*in*);  System.*out*.println("Welcome to Hangman!");   boolean letsPlay = true;  while (true) {  if (!doYouWantToPlay) break;  System.*out*.println();  System.*out*.println(" Lets play Hangman");  Hangman game = new Hangman();  System.*out*.println();  System.*out*.println(game.printPicture());  System.*out*.println();  System.*out*.println(game.getFormalCurrentGuess());  System.*out*.println();    System.*out*.println("Enter a character");  char guess = (sc.next().toLowerCase()).charAt(0);  System.*out*.println();    if (!game.isGuessedAlready(guess)) {  } else {  System.*out*.println("Try again. You guessed already");  guess = (sc.next().toLowerCase()).charAt(0);  }   if (game.playGuess(guess)) {  System.*out*.println(" Guess correct");  }  else {  System.*out*.println("sorry that word is not in the guess");  }  while (!game.gameover()) {  System.*out*.println();  System.*out*.println();  System.*out*.println(game.getFormalCurrentGuess());  System.*out*.println();    System.*out*.println("Enter a character");  guess = (sc.next().toLowerCase()).charAt(0);  System.*out*.println();    if (game.isGuessedAlready(guess)) {  System.*out*.println("Try again. You guessed already");  guess = (sc.next().toLowerCase()).charAt(0);  }   if (game.playGuess(guess)) {  System.*out*.println(" Guess correct");  } else {  System.*out*.println("sorry that word is not in the guess");  }  }   System.*out*.println();  System.*out*.println("Do you want to play another game. Enter Y if you do or press any other word to exit.");  Character response = (sc.next().toUpperCase()).charAt(0);  letsPlay = (response == 'Y');  }   } } |

# Refactored code.

## Code Smells

The names of the variables should always complete. The variables are self-explained so that these do not become mislead or confusing. Different type of code smells exists in the current code. First is change preventers which means if we want to change the functionality a little bit at one place, it will have the ripple effect in other parts of the code too. The main change preventer of the code is a random generator for which I have created a separate method for generating the random numbers. Now the methods and variables are self-explanatory. There are also bloaters in the code which means the number of methods are excessive and causing the confusion in understanding the code. So I have simplified the code with the use of generic methods which can perform the desired tasks with having too much extra code.

## Updated Code:

Hangman.java

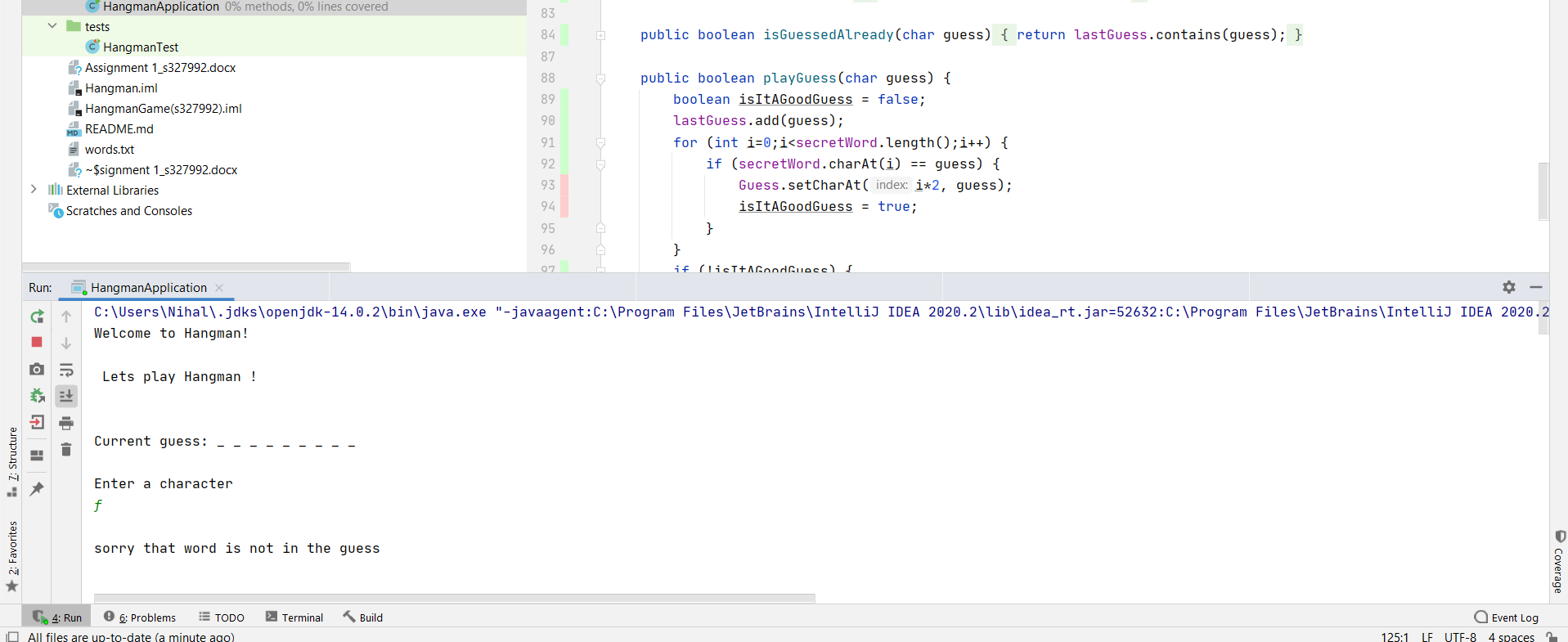
|  |
| --- |
| import java.io.\*; import java.util.ArrayList; import java.util.Random;  public class Hangman {  String secretWord;  StringBuilder Guess;  ArrayList<Character> lastGuess = new ArrayList<>();   int maximumTries = 6;  int yourTry = 0;   ArrayList<String> words = new ArrayList<>();   public Hangman() throws IOException {  initializeStream();  secretWord = selectWord();  Guess = loadCurrentGuess();  }   public void initializeStream() throws IOException {  File inFile = new File("words.txt");  FileReader fileReader = new FileReader(inFile);  BufferedReader bufferedFileReader = new BufferedReader(fileReader);  String currentLine = bufferedFileReader.readLine();  while (currentLine != null) {  words.add(currentLine);  currentLine = bufferedFileReader.readLine();  }  bufferedFileReader.close();  fileReader.close();  }   public String selectWord() {  Random rand = new Random();  int wordIndex = Math.*abs*(rand.nextInt())%words.size();  return words.get(wordIndex);  }   public StringBuilder loadCurrentGuess() {  StringBuilder current = new StringBuilder();  for (int i=0;i<secretWord.length()\*2;i++) {  if (i%2 == 0) {  current.append("\_");  }  else {  current.append(" ");  }  }  return current;  }   public String getFormalCurrentGuess() {  return "Current guess: " + Guess.toString();  }   public boolean gameover() {  if (didWeWin()) {  System.*out*.println();  System.*out*.println("Congratulation. You won the game");  }  else if (didWeLose()) {  System.*out*.println();  System.*out*.println("Sorry. You lost the game." +  " The secret word was: " + secretWord + ".");  }  return didWeWin() || didWeLose();  }   public boolean didWeWin() {  String guess = getCondensedCurrentGuess();  return guess.equals(secretWord);  }   public String getCondensedCurrentGuess() {  String guess = Guess.toString();  return guess.replace(" ", "");  }   public boolean didWeLose() {  return yourTry >= maximumTries;  }   public boolean isGuessedAlready(char guess) {  return lastGuess.contains(guess);  }   public boolean playGuess(char guess) {  boolean isItAGoodGuess = false;  lastGuess.add(guess);  for (int i=0;i<secretWord.length();i++) {  if (secretWord.charAt(i) == guess) {  Guess.setCharAt(i\*2, guess);  isItAGoodGuess = true;  }  }  if (!isItAGoodGuess) {  yourTry++;  }  return isItAGoodGuess;  }   } |

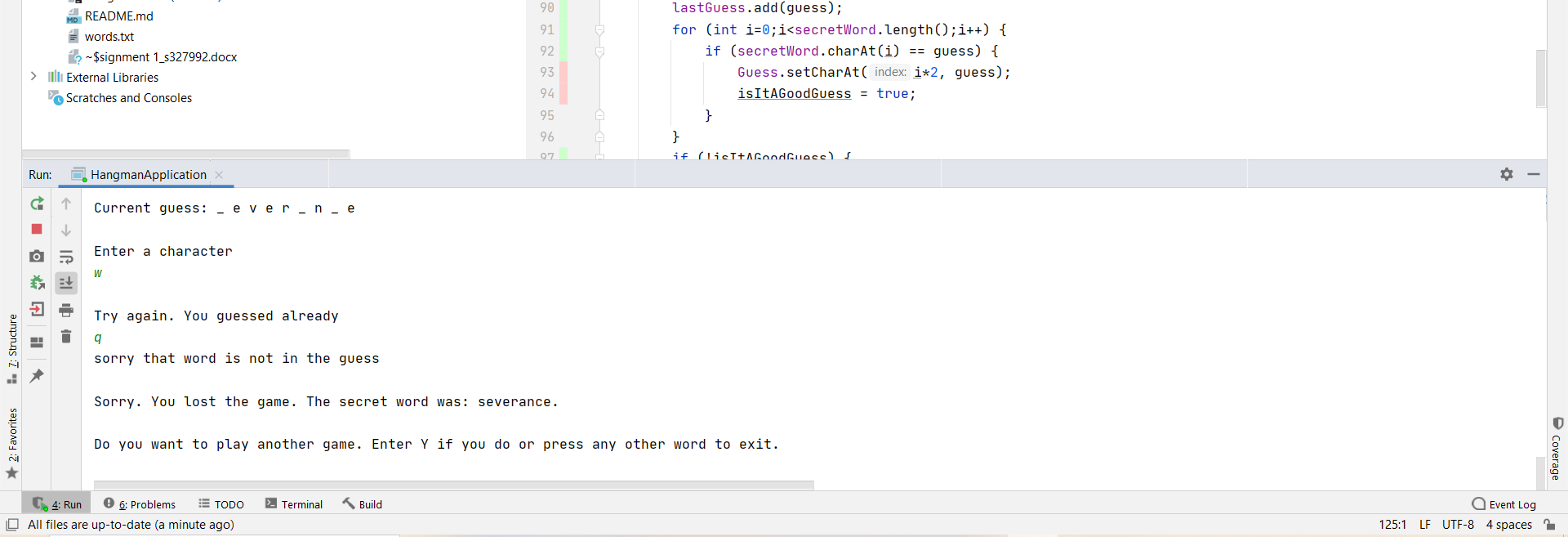
# HangmanApplication.java

|  |
| --- |
| import java.io.IOException; import java.util.Scanner;  public class HangmanApplication {  public static void main(String[] args) throws IOException {  Scanner sc = new Scanner(System.*in*);  System.*out*.println("Welcome to Hangman!");   boolean letsPlay = true;  while (letsPlay) {  System.*out*.println();  System.*out*.println(" Lets play Hangman !");  Hangman game = new Hangman();  do {  System.*out*.println();  System.*out*.println();  System.*out*.println(game.getFormalCurrentGuess());  System.*out*.println();    System.*out*.println("Enter a character");  char guess = (sc.next().toLowerCase()).charAt(0);  System.*out*.println();    if (game.isGuessedAlready(guess)) {  System.*out*.println("Try again. You guessed already");  guess = (sc.next().toLowerCase()).charAt(0);  }   if (game.playGuess(guess)) {  System.*out*.println(" Guess correct");  }  else {  System.*out*.println("sorry that word is not in the guess");  }  }  while (!game.gameover());   System.*out*.println();  System.*out*.println("Do you want to play another game. Enter Y if you do or press any other word to exit.");  char response = (sc.next().toUpperCase()).charAt(0);  letsPlay = (response == 'Y');  }  } } |

# **Code Output:**

The output of the game can be shown in the given screenshots. In the output, the user is given the random word to guess. The user started with the first attempt with guessing R for which program check that and shows that it is the wrong input and displays the output that Player health is decreasing. After that user continue guessing the wrongs character. After several failed attempt the system reaches the level of a maximum number of tries allowed for it after that program shows the output that player dies.





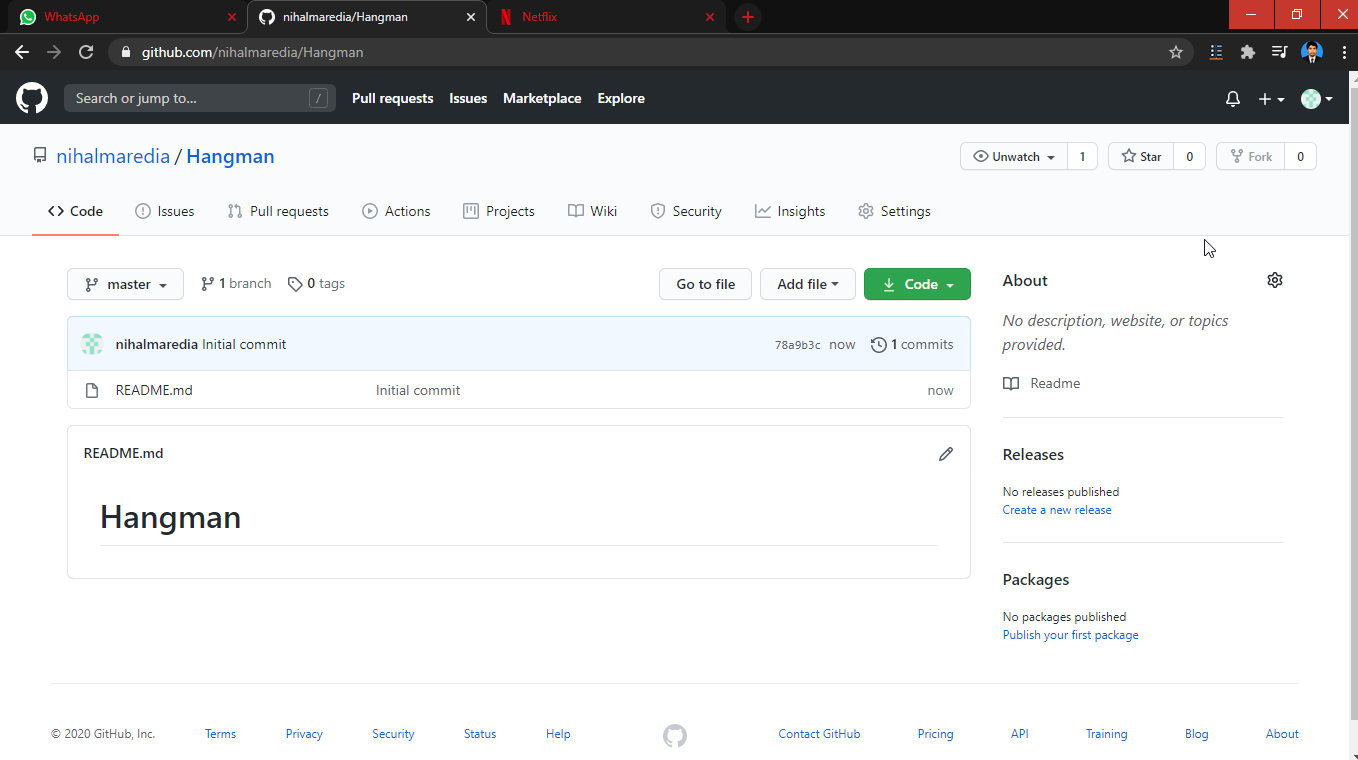
# 

# Creating a GitHub

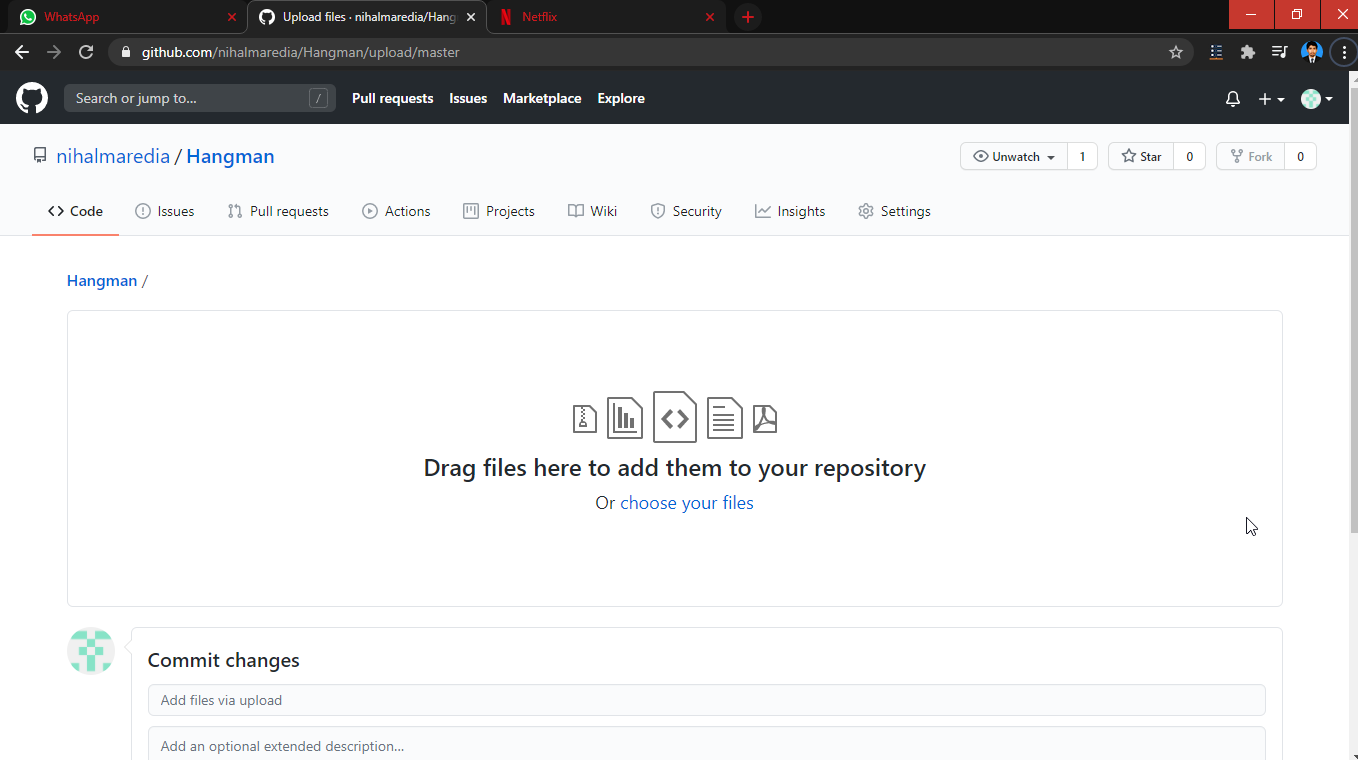
Creating a new repository in GitHub account for uploading the code on GitHub. Firstly we have to create a repository by adding a repository name and select the public and private for privacy purposes. Then we have to initialize the repository with the readme file if we want to.

# 

New repository in GitHub is created successfully. Now we can see the project name is written in the box after the readme file. At the top right of that box a green box is appearing which has the option for file upload. We have to select that button a file upload window will appear.

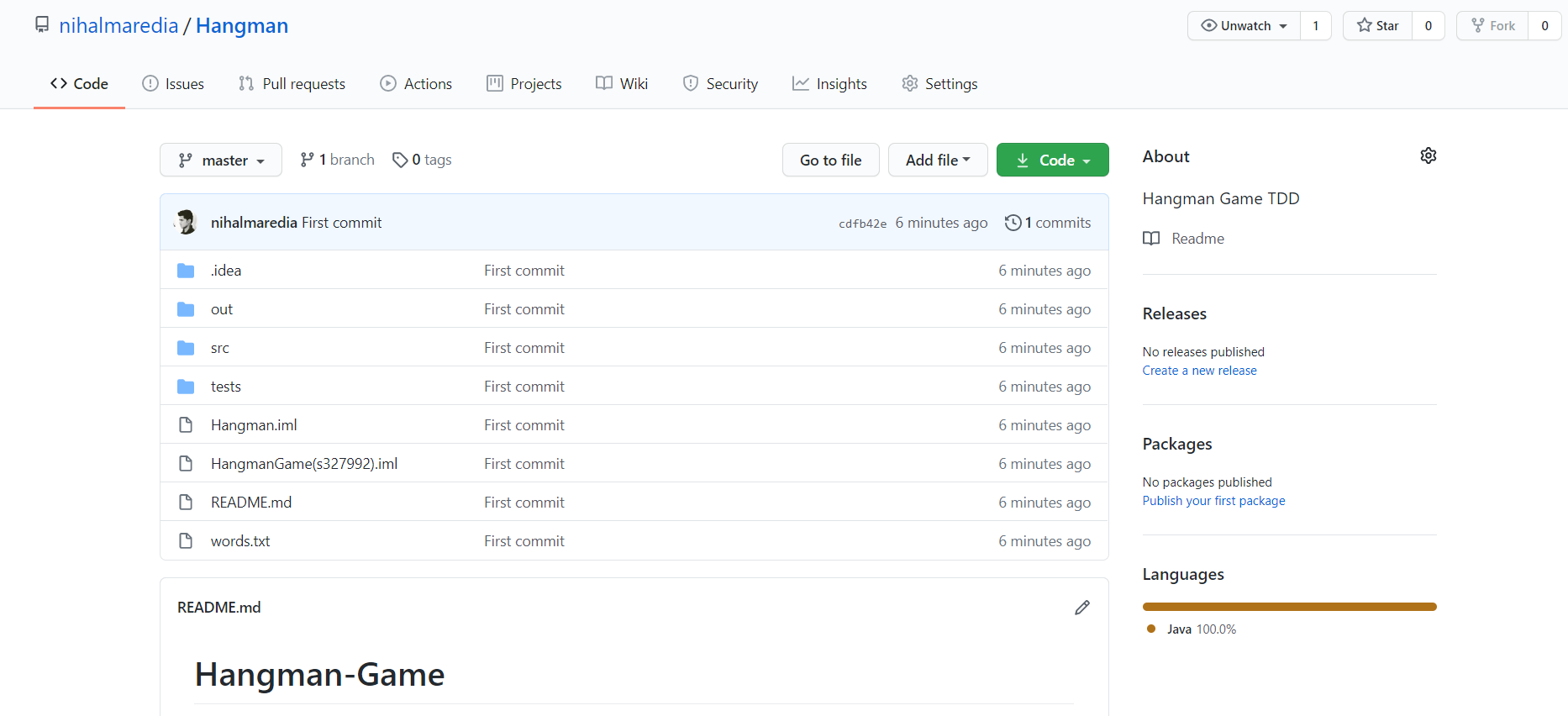


Now add the code files and report into the repository in the file upload window which has multiple options of uploading files of multiple file types including code files, zip files, documents in pdf or word format. We will select the item from our local system directory by clicking the “choose your files” button and upload the project folder into it. We can add title and description for the uploaded project for the future reference. We have to click the commit changes button to upload the documents to the created directory.



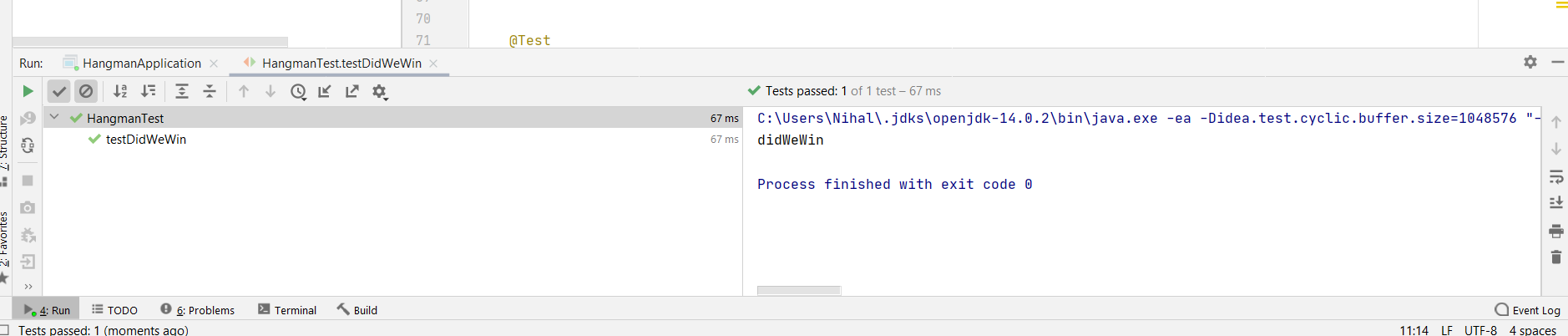
Now the code has been uploaded to the directory. Now we can share the link of that particular directory to the person we want to share the code with.

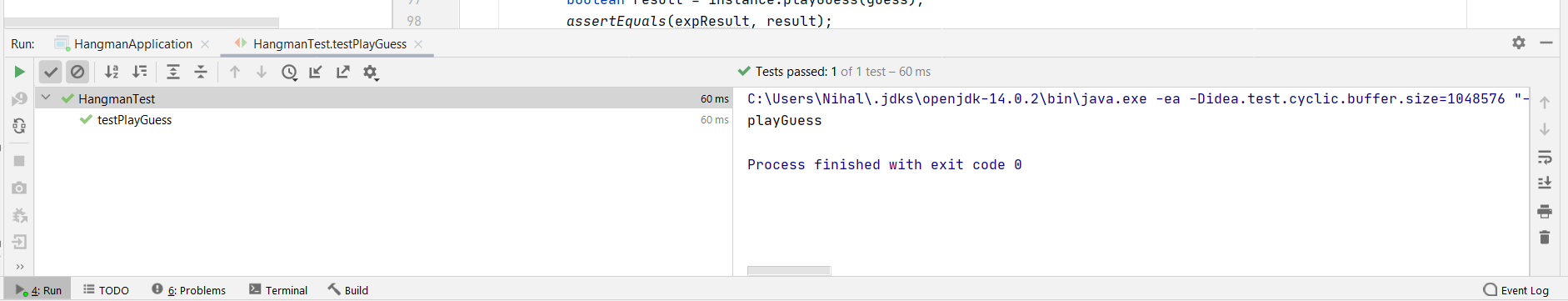
Github Link: <https://github.com/nihalmaredia/Hangman>

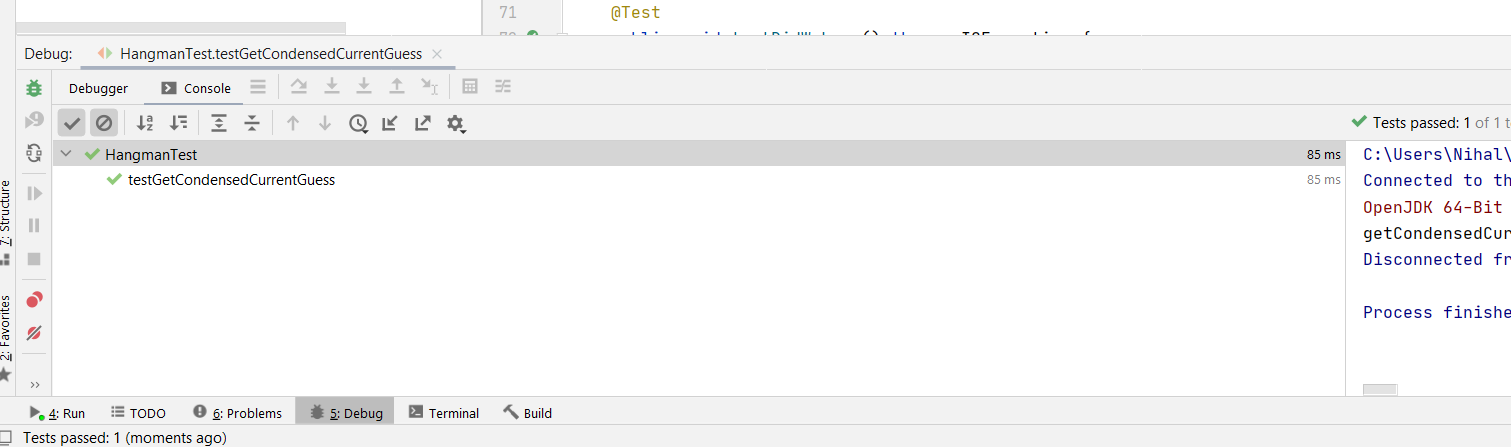


How TDD has been implemented to create your program.We have used the TDD approach for the development of this application. We could easily test using the method of Random Generator to check that a random no of specified is being generated or not. We have tested our code by running the first test on it which is failed initially, and the test status and errors are displayed on the console of the IDE. This test throws the **NoSuchElementException** of the **java.Util** class. The errors messages are concise and have much information to explain the error and reason of error in the expandable error links. The screenshot of the tests is being attached.

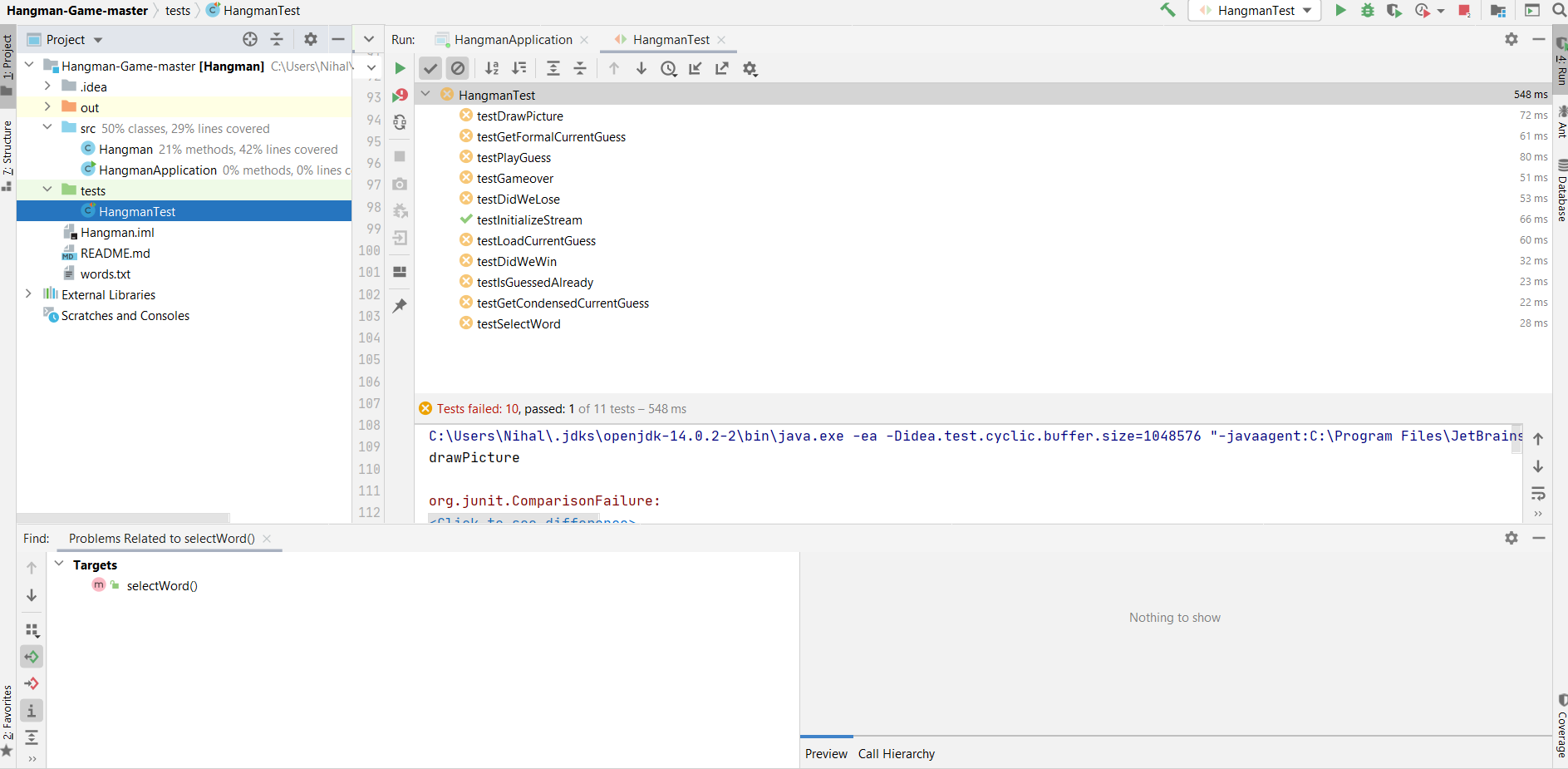
**JUNIT4 Result for this Test Class**





****

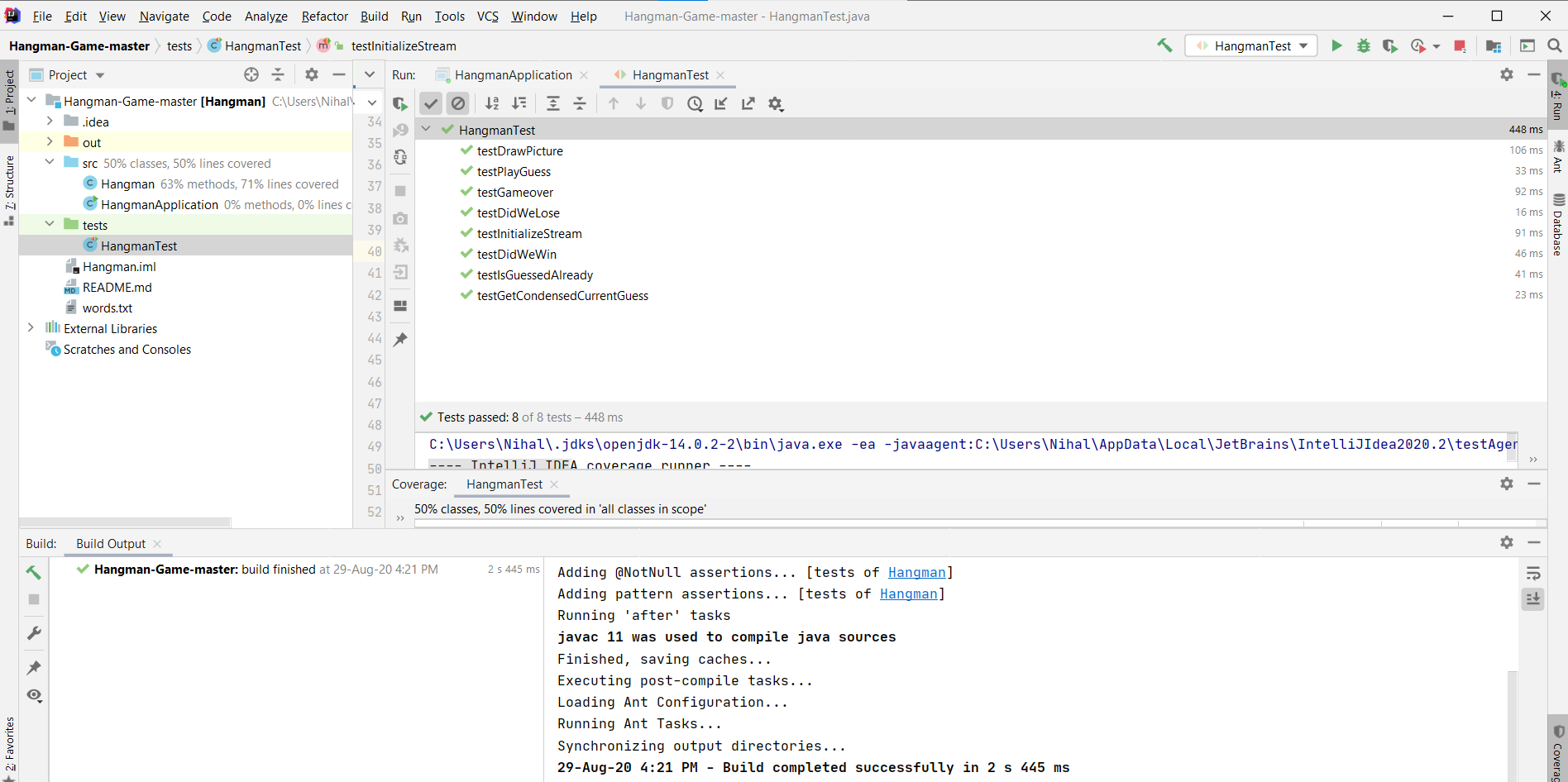
**Demonstrating Red -> Green -> Refactor using Junit4**



After refactoring the code and make changes in code, the test case is running perfectly. The test status is written in the console with the time that CPU takes to execute the test. The correct testing code is also written below, which is used for testing the program.

**After I handled the condition so, all the test are passed, it can be seen in the following screenshot.**

**This is a junit4 Test Class for methods in Hangman.java. It has various test methods.**



**Code :**

|  |
| --- |
| import static org.junit.Assert.\*; import org.junit.After; import org.junit.AfterClass; import org.junit.Before; import org.junit.BeforeClass; import org.junit.Test;  import java.io.IOException;   public class HangmanTest {   public HangmanTest() {  }   @BeforeClass  public static void setUpClass() {  }   @AfterClass  public static void tearDownClass() {  }   @Before  public void setUp() {  }   @After  public void tearDown() {  }   */\*\*  \* Test of initializeStream method, of class Hangman.  \*/* @Test  public void testInitializeStream() throws IOException {  System.*out*.println("initializeStream");  Hangman instance = new Hangman();  instance.initializeStream();  }     @Test  public void testGameover() throws IOException {  System.*out*.println("gameover");  Hangman instance = new Hangman();  boolean expResult =true;  boolean result = instance.gameover();  }  @Test  public void testDidWeWin() throws IOException {  System.*out*.println("didWeWin");  Hangman instance = new Hangman();  boolean expResult = false;  boolean result = instance.didWeWin();  *assertEquals*(expResult, false);  }    @Test  public void testGetCondensedCurrentGuess() throws IOException {  System.*out*.println("getCondensedCurrentGuess");  Hangman instance = new Hangman();  String expResult = "correct";  String result = instance.getCondensedCurrentGuess();   }    @Test  public void testDidWeLose() throws IOException {  System.*out*.println("didWeLose");  Hangman instance = new Hangman();  boolean expResult = instance.didWeWin();  boolean result = instance.didWeLose();  }    @Test  public void testIsGuessedAlready() throws IOException {  System.*out*.println("isGuessedAlready");  char guess = ' ';  Hangman instance = new Hangman();  boolean expResult = true;  boolean result = instance.isGuessedAlready(guess);  *assertEquals*(expResult, true);  }    @Test  public void testPlayGuess() throws IOException {  System.*out*.println("playGuess");  char guess = ' ';  Hangman instance = new Hangman();  boolean expResult = false;  boolean result = instance.playGuess(guess);  *assertEquals*(expResult, result);  }    } |

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

The refactoring of the code is done and also the latest code committed to GitHub repository. You could check the commit history. The issue was that the variables were not self-explanatory, and all the code is the main method. If any problem occurs, then I have to debug complete code to find out the issue. But now different methods are formed according to their functions. Now the methods and variables are well-defined. Recurrence is one of the most widely used terms in software development and has played a major role in software maintenance for decades. While many developers have an accurate understanding of the process of repetition, most of us do not have an accurate understanding of this important skill. In this article, we will look at the definition of re-use literature, how this definition adheres to the robustness of software development, and how we can ensure that our codebase is optimized for renewal. Along the way, we will go through a set of repetitions, from beginning to end, to show the stretch and importance of this magnificent process. There are several reasons. For example, achieving simple visibility and easy coding. Proponents of her case have been working to make the actual transcript of this statement available online. Some developers are convinced that the code needs to be rewritten to be understood with the lowest number of comments.

Each party accepted its position, but remember that repetition does not mean downgrading.

Its main purpose is to improve the structure of the code.

* Several functions can be included for this general purpose:
* Redesign improves the understanding of the code written by other engineers.
* It helps to find and fix bugs.
* It can speed up software development.
* Overall, it improves software design.
* If the repetition does not take place over a long period, progress may be encountered with difficulties, including a complete stagnation of work.

"The smell of code."

When the code needs to be repeated, it is said to smell. True, not real, but such code doesn't look very appealing. Below we will explore the basic techniques of repetition of the first phase.

# References

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

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.

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

Suri, B. (2017). A Systematic Literature Review: Code Bad Smells in Java Source Code. A Systematic Literature Review: Code Bad Smells in Java Source Code, 12.