# Q1: Your task is to write a program: Hangman using TDD.

import java.util.Random;

import java.util.Scanner;

public class HangmanGame {

String randomWord = generateRandomWord(10);

String lettersLine = new String(new char[randomWord.length()]).replace("\0", "-");

int count = 0;

public void main(String[] args) {

Scanner s = new Scanner(System.in);

while (count < 7 && lettersLine.contains("-")) {

message("Guess any letter in the word");

message(lettersLine);

String guessWord = s.next();

hangCharachter(guessWord);

}

s.close();

}

void hangCharachter(String guessWord) {

String newLettersLine = "";

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

if (randomWord.charAt(i) == guessWord.charAt(0)) {

newLettersLine += guessWord.charAt(0);

} else if (lettersLine.charAt(i) != '\*') {

newLettersLine += randomWord.charAt(i);

} else {

newLettersLine += "\*";

}

}

if (lettersLine.equals(newLettersLine)) {

count++;

hangmanLife();

} else {

lettersLine = newLettersLine;

}

if (lettersLine.equals(randomWord)) {

message("Congratulations! you guessed the right word. You win! The word was " + randomWord);

}

}

void hangmanLife() {

int count = 0;

switch(count)

{

case 1:

message("Wrong guess, try again");

message("Player Life Remaining: 10");

break;

case 2:

message("Wrong guess, try again");

message("Player Life Remaining: 8");

break;

case 3:

message("Wrong guess, try again");

message("Player Life Remaining: 6");

break;

case 4:

message("Wrong guess, try again");

message("Player Life Remaining: 4");

break;

case 5:

message("Wrong guess, try again");

message("Player Life Remaining: 2");

break;

case 6:

message("Wrong guess, try again");

message("Player Life Remaining: 1");

break;

case 7:

message("Game Over!");

message("Player Hanged");

break;

}

}

String generateRandomWord(int wordLength) {

Random r = new Random();

StringBuilder sb = new StringBuilder(wordLength);

for(int i = 0; i < wordLength; i++) {

char tmp = (char) ('a' + r.nextInt('z' - 'a'));

sb.append(tmp);

}

return sb.toString();

}

void message(String inputMessage)

{

System.out.println(inputMessage);

}

}

# Q2: Refactor your code – Code smell can give indications that there is some issue with the codes and can be solved by refactoring.

## import java.util.Random;

## import java.util.Scanner;

## public class HangmanGame {

## private static final String randomWord = generateRandomWord(10);

## private static String lettersLine = new String(new char[randomWord.length()]).replace("\0", "-");

## private static int count = 0;

## public static void main(String[] args) {

## Scanner s = new Scanner(System.in);

## while (count < 7 && lettersLine.contains("-")) {

## message("Guess any letter in the word");

## message(lettersLine);

## String guessWord = s.next();

## hangCharachter(guessWord);

## }

## s.close();

## }

## public static void hangCharachter(String guessWord) {

## String newLettersLine = "";

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

## if (randomWord.charAt(i) == guessWord.charAt(0)) {

## newLettersLine += guessWord.charAt(0);

## } else if (lettersLine.charAt(i) != '\*') {

## newLettersLine += randomWord.charAt(i);

## } else {

## newLettersLine += "\*";

## }

## }

## if (lettersLine.equals(newLettersLine)) {

## count++;

## hangmanLife();

## } else {

## lettersLine = newLettersLine;

## }

## if (lettersLine.equals(randomWord)) {

## message("Congratulations! you guessed the right word. You win! The word was " + randomWord);

## }

## }

## public static void hangmanLife() {

## if (count == 1) {

## message("Wrong guess, try again");

## message("Player Life Remaining: 10");

## }

## if (count == 2) {

## message("Wrong guess, try again");

## message("Player Life Remaining: 8");

## }

## if (count == 3) {

## message("Wrong guess, try again");

## message("Player Life Remaining: 6");

## }

## if (count == 4) {

## message("Wrong guess, try again");

## message("Player Life Remaining: 4");

## }

## if (count == 5) {

## message("Wrong guess, try again");

## message("Player Life Remaining: 2");

## }

## if (count == 6) {

## message("Wrong guess, try again");

## message("Player Life Remaining: 1");

## }

## if (count == 7) {

## message("Game Over!");

## message("Player Hanged");

## }

## }

## public static String generateRandomWord(int wordLength) {

## Random r = new Random();

## StringBuilder sb = new StringBuilder(wordLength);

## for(int i = 0; i < wordLength; i++) {

## char tmp = (char) ('a' + r.nextInt('z' - 'a'));

## sb.append(tmp);

## }

## return sb.toString();

## }

## public static void message(String inputMessage)

## {

## System.out.println(inputMessage);

## }

## } Code Smells:

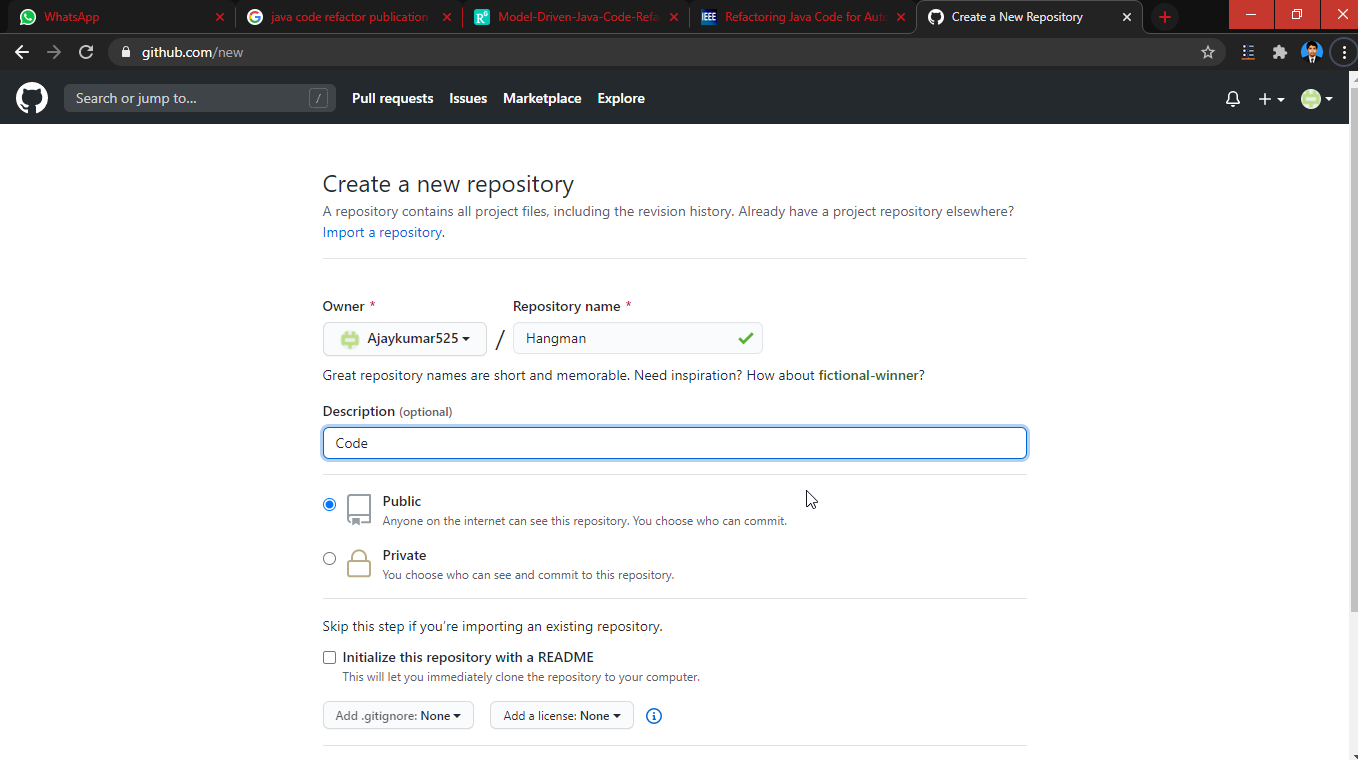
**Global Variable Class** founds in the code. I have removed the global variables from the code.

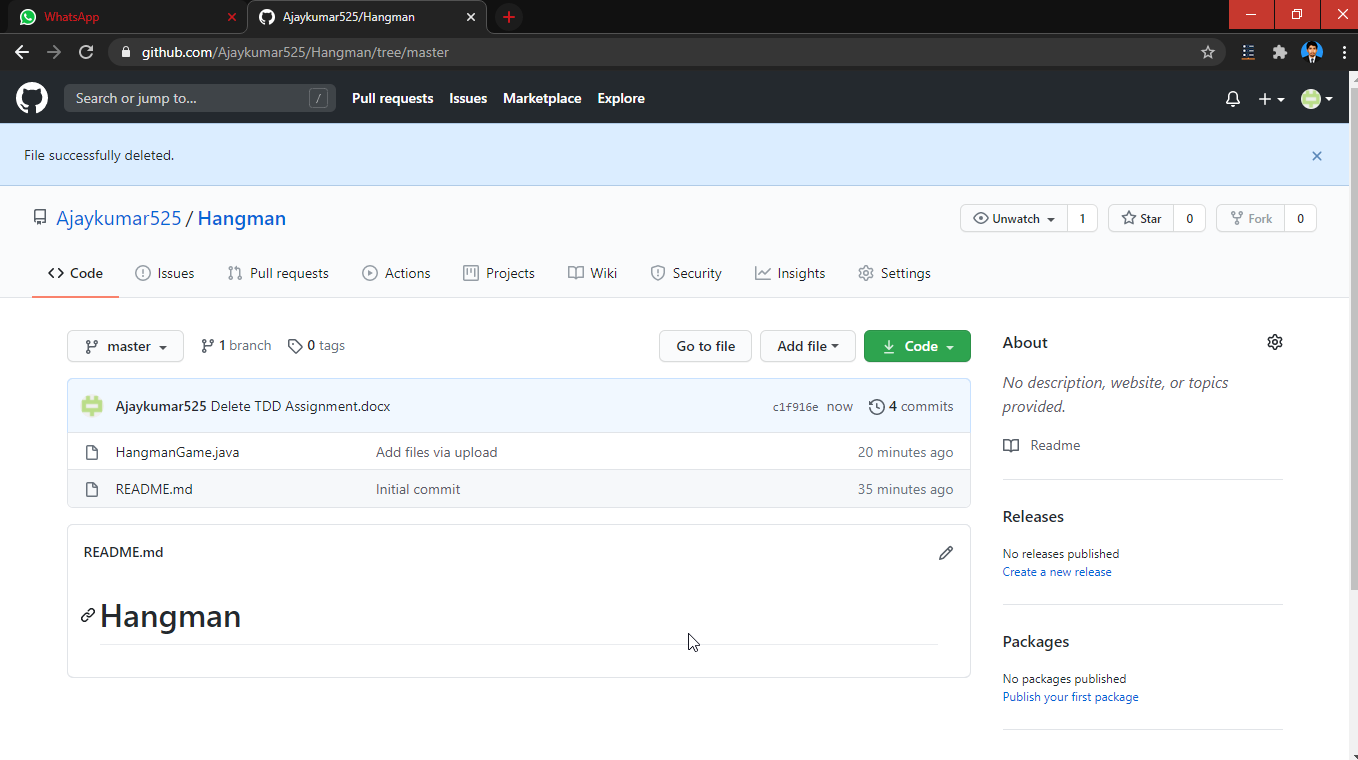
**Global Function Class** founds in the code. I have removed the global functions from the code.

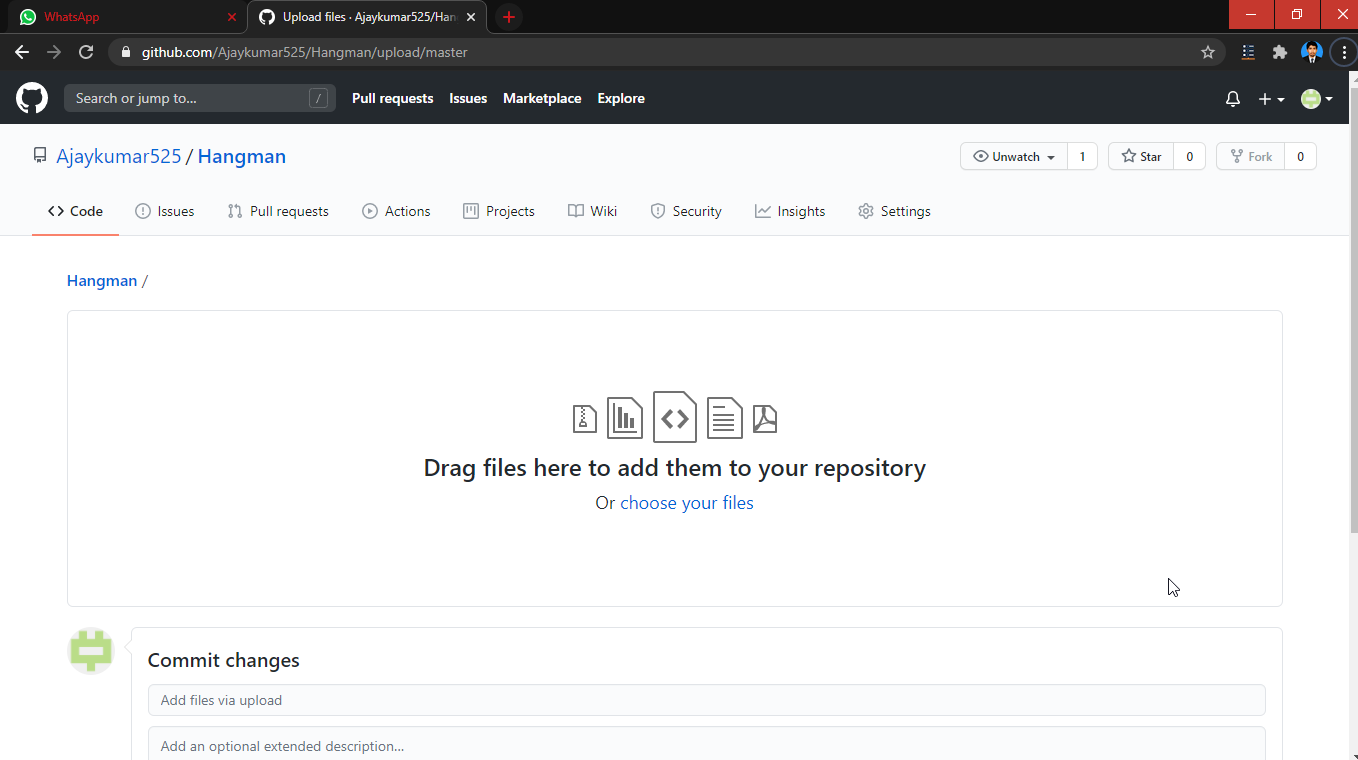
**Variable Names** should always complete. I have renamed the methods and variables names so that it cannot becomes mislead or confusing.

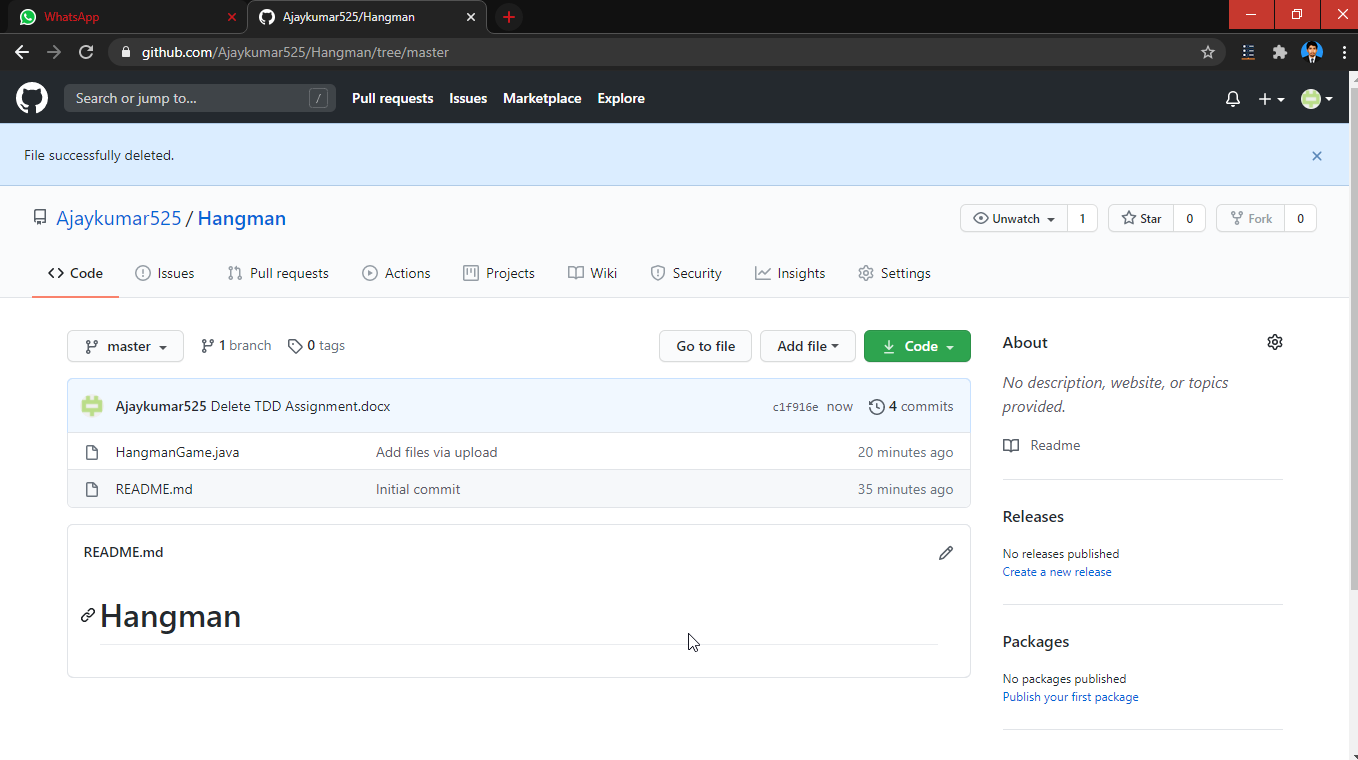
**Switch Statement** founds in the code. I have removed the switch statement and replaced it with if condition.

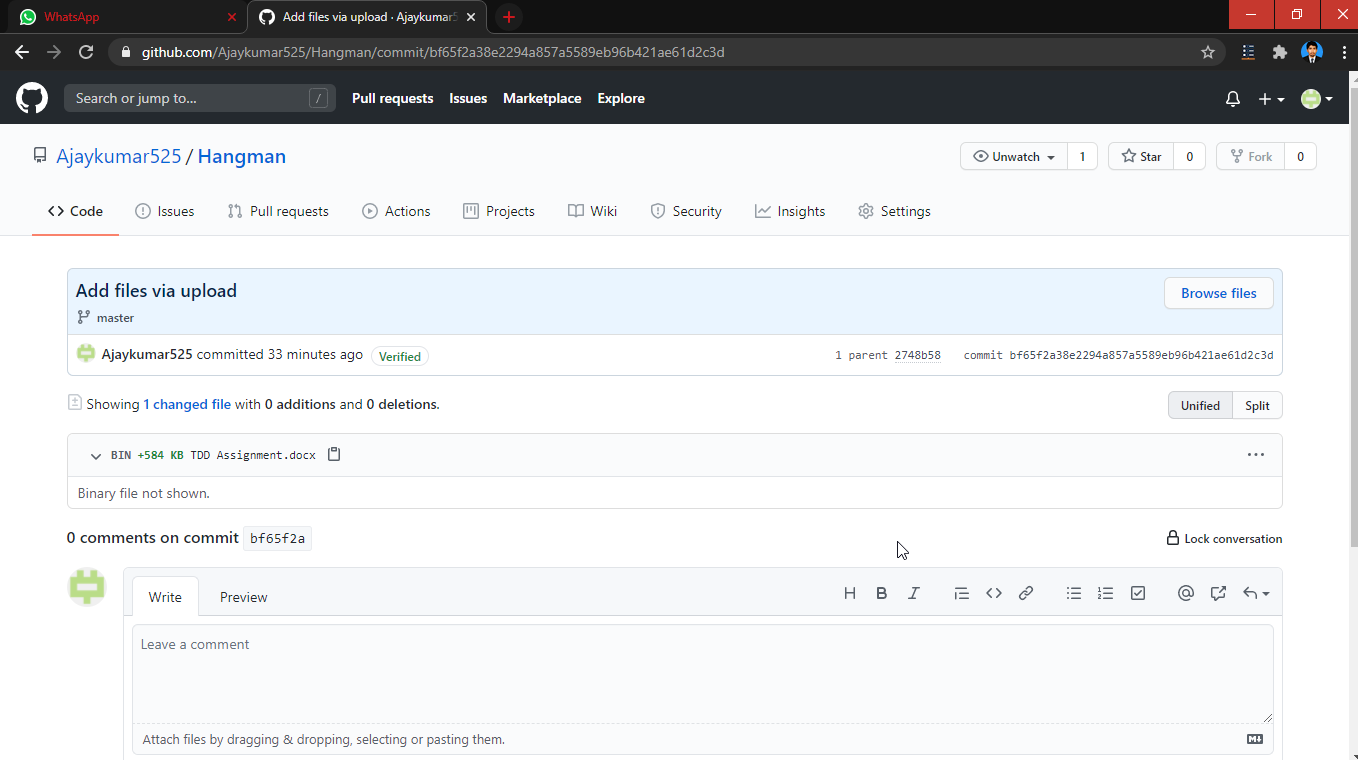
# Q3: Create a Git directory for your assignment (including word or pdf documents and programming code)











# Q4: how TDD has been implemented to create your program

## HangmanTest:

import org.junit.After;

import org.junit.AfterClass;

import org.junit.Before;

import org.junit.BeforeClass;

import org.junit.Test;

import static org.junit.Assert.\*;

public class HangmanTest {

public HangmanTest() {

}

@BeforeClass

public static void setUpClass() {

}

@AfterClass

public static void tearDownClass() {

}

@Before

public void setUp() {

}

@After

public void tearDown() {

}

@Test

public void testMain() {

System.out.println("main");

String[] args = null;

Hangman.main(args);

fail("The test case is a prototype.");

}

@Test

public void testHang() {

System.out.println("hang");

String guess = "";

Hangman.hang(guess);

fail("The test case is a prototype.");

}

@Test

public void testHangmanImage() {

System.out.println("hangmanImage");

Hangman.hangmanImage();

fail("The test case is a prototype.");

}

@Test

public void testGenerateRandomWord() {

System.out.println("generateRandomWord");

int wordLength = 0;

String expResult = "";

String result = Hangman.generateRandomWord(wordLength);

assertEquals(expResult, result);

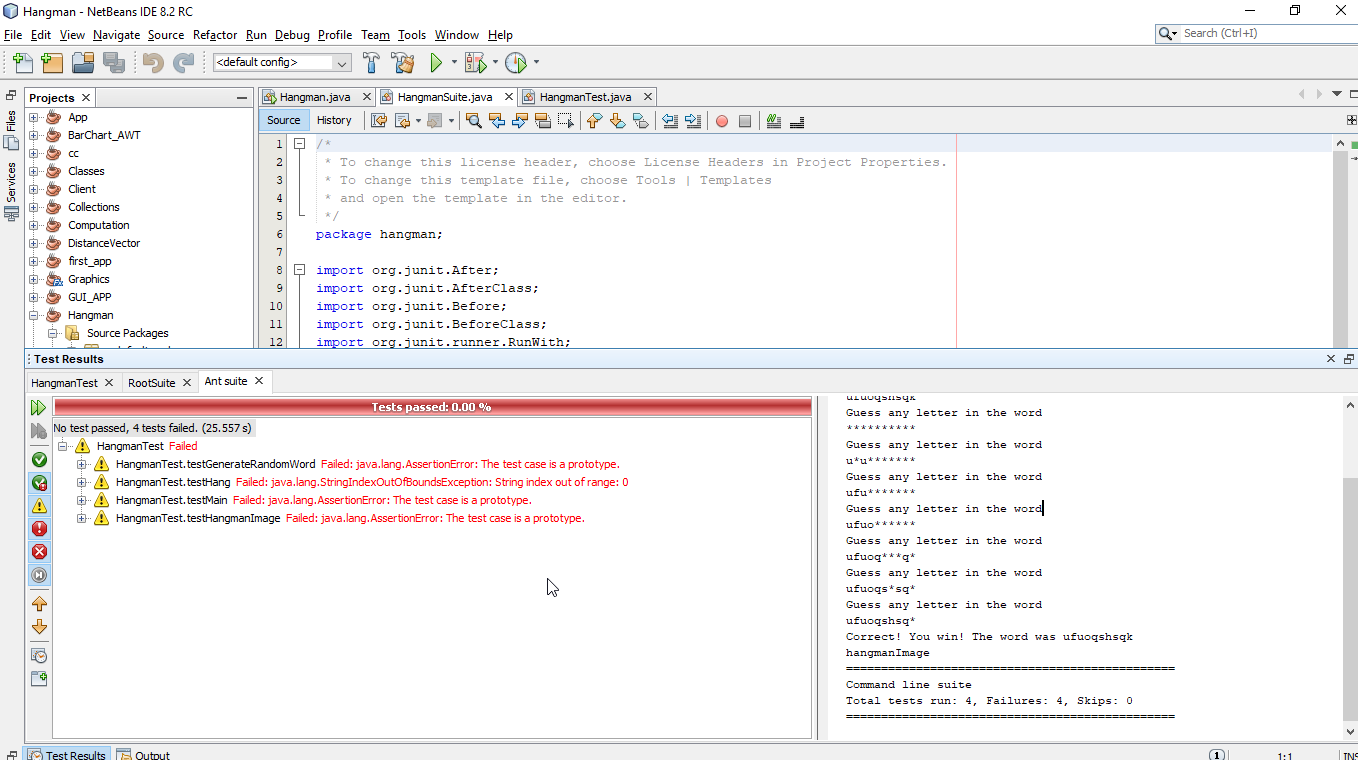
fail("The test case is a prototype.");

}

}

## Refactoring Using Junit:

We have write a Junit test for Hangman but it fails.



After refactoring the code completely, I have resolved the issue and test cases passed

A screenshot of a computer

Description automatically generated

## how refactoring has been done:

Data Clumps, Middle Man, Speculative Generality

## Choose some bad smells.

Zimmerman et al.’s (2007) fault identification approach:

* 1. Locate “bug”, “fix(ed)” and “update (d)” token in CVS comment messages.
  2. If a version entry in CVS contains one or more above tokens and those tokens are followed by numbers, this version entry is seen as a bug fixing update.

## Data Clumps:

What refactoring to apply when there is a Data Clump?

* Extract the fields and gathering them into a class. Along these lines numerous classes can allude to this regular class without having clusters of information things.
* For boundary list, adjust the strategy marks to utilize an occasion of the class. This cleans up the strategy marks and builds the lucidness of the code.

## Middle Man:

What refactoring to apply?

* When there is best way to accomplish a specific activity, and the center man is just appointing this activity to another class, we can basically expel the techniques from the interface and move those strategy calls straightforwardly into our usage. This reductions the measure of redirection and improves code lucidness.
* We can transform the center man into a section (subclass) of the genuine article. This permits us to broaden the conduct without pursuing the "indirection". This improves code intelligibility and diminishes the psychological burden on the engineer who needs to recall such appointment.

## Speculative Generality:

What Refactoring to apply?

* To refactor a path from theoretical sweeping statement, you can fall dynamic classes that aren't doing much by utilizing breakdown order. Pointless appointment can be evacuated utilizing inline class. Unused boundaries can be evacuated.

# References

Miguel P. Monteiro, João M. Fernandes. (2012). Aspect-oriented Refactoring of Java Programs. *AMADEUS (POCTI, PTDC/EIA/70271/2006)*, 19.

Moonen, L. (2002). Java Quality Assurance by Detecting Code Smells. *Java Quality Assurance by Detecting Code Smells*, 77.

Rohit Gheyi, Tiago Massoni, Dalton Serey, Gustavo Soares. (2010). Making Program Refactoring Safer. *Making Program Refactoring Safer*, 57.

Zimmermann, T. (2007). Predicting Defects for Eclipse. *Predicting Defects for Eclipse*, 24.