



ASSIGNMENT 1

Subject Code: PRT582 (Software Engineering: Process & Tools)



SUBMITTED BY: AJAYKUMAR PATEL

STUDENT ID: S328216

AUGUST 29, 2020
SUBMITTED TO: CHARLES YEO

TABLE OF CONTENTS

TITLE	PAGE NO.
QUESTION 1: Hangman Program (Before & After) with screenshots	2
QUESTION 2: Refactoring and code smells	7
QUESTION 3: Creating Git directory (Screenshots)	12
QUESTION 4: How TDD has been implemented to create the program	14
OUTPUT of the Hangman Game (Screenshots)	19
Github Link	21

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

- Program (Before Screenshots)

```
package hangman;

import java.util.Scanner;
import hangman.ConsoleColors;

public class HangmanGame {
    private String[] words = {
        "train", "chocolate", "algorithm", "canada", "awesome", "water",
        "mango", "wafer", "joker", "grandfather", "darwin", "america",
        "light", "laptop", "computer", "amazing", "adelaide", "river",
        "ocean", "car", "noodles", "bread", "pizza", "burger"};

    private String word = words[(int) (Math.random() * words.length)];
    private String lettersLine = new String(new char[word.length()]).replace("\\0", "_");
    private int count = 0;

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        System.out.println("Hello, welcome to the Hangman Game. Let's play...");
        System.out.println("\nThe number of letters in this word are "+word.length());
        System.out.println(ConsoleColors.ANSI_RED + "You have 6 player lives to attempt a wrong guess.");

        while (count < 6 && lettersLine.contains("_")) {
            System.out.println("\nPlease guess a character of the word");
            System.out.println(lettersLine);
            String guessWord = s.next();
            checkCharachter(guessWord);
        }
    }
}
```

```

    }
    s.close();
}
}
void checkCharachter(String guessWord) {
    String newLettersLine = "";
    for (int i = 0; i < word.length(); i++) {
        if (word.charAt(i) == guessWord.charAt(0)) {
            newLettersLine += guessWord.charAt(0);
        } else if (lettersLine.charAt(i) != '*' ) {
            newLettersLine += word.charAt(i);
        } else {
            newLettersLine += " ";
        }
    }
    if (lettersLine.equals(newLettersLine)) {
        count++;
        deductLife();
    } else {
        lettersLine = newLettersLine;
    }
    if (lettersLine.equals(word)) {
        System.out.println(ConsoleColors.ANSI_BLUE + "Woohoo...! You guessed the word correctly. You won! The word was \"
    }
}
}
void deductLife() {
    int count = 0;
    switch(count)

```

```

        case 1:
            System.out.println(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong guess, please try another letter");
            System.out.println(ConsoleColors.ANSI_RED + "Player Life Remaining: 5");
            break;
        case 2:
            System.out.println(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong guess, please try another letter");
            System.out.println(ConsoleColors.ANSI_RED + "Player Life Remaining: 4");
            break;
        case 3:
            System.out.println(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong guess, please try another letter");
            System.out.println(ConsoleColors.ANSI_RED + "Player Life Remaining: 3");
            break;
        case 4:
            System.out.println(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong guess, please try another letter");
            System.out.println(ConsoleColors.ANSI_RED + "Player Life Remaining: 2");
            break;
        case 5:
            System.out.println(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong guess, please try another letter");
            System.out.println(ConsoleColors.ANSI_RED + "Player Life Remaining: 1");
            break;
        case 6:
            System.out.println(ConsoleColors.ANSI_RED + "The game is over!");
            System.out.println(ConsoleColors.ANSI_RED + "Oh no...! Unfortunately you have lost the game. The word was \"" + word
            break;
    }
}
}
}

```

The actual “before” code in text format is as below:
package hangman;

```

import java.util.Scanner;
import hangman.ConsoleColors;

public class HangmanGame {
    private String[] words = {
        "train", "chocolate", "algorithm", "canada", "awesome", "water",
        "mango", "wafer", "joker", "grandfather", "darwin", "america",
        "light", "laptop", "computer", "amazing", "adelaide", "river",
        "ocean", "car", "noodles", "bread", "pizza", "burger"};
    private String word = words[(int) (Math.random() * words.length)];
    private String lettersLine = new String(new char[word.length()]).replace("\0", "_");
    private int count = 0;

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        System.out.println("Hello, welcome to the Hangman Game. Let's play...");
        System.out.println("\nThe number of letters in this word are "+word.length());
        System.out.println(ConsoleColors.ANSI_RED + "You have 6 player lives to attempt a
            wrong guess.");

        while (count < 6 && lettersLine.contains("_")) {
            System.out.println("\nPlease guess a character of the word");
            System.out.println(lettersLine);
            String guessWord = s.next();
            checkCharachter(guessWord);
        }
        s.close();
    }
}

```

```

    }

    void checkCharachter(String guessWord) {
        String newLettersLine = "";
        for (int i = 0; i < word.length(); i++) {
            if (word.charAt(i) == guessWord.charAt(0)) {
                newLettersLine += guessWord.charAt(0);
            } else if (lettersLine.charAt(i) != '*') {
                newLettersLine += word.charAt(i);
            } else {
                newLettersLine += "*";
            }
        }

        if (lettersLine.equals(newLettersLine)) {
            count++;
            deductLife();
        } else {
            lettersLine = newLettersLine;
        }

        if (lettersLine.equals(word)) {
            System.out.println(ConsoleColors.ANSI_BLUE + "Woohoo..! You
            guessed the word correctly. You won! The word was \"" + word + "\"");
        }
    }

    void deductLife() {
        int count = 0;
        switch(count)
        {
            case 1:
                System.out.println(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong

```

```

        guess, please try another letter");
    System.out.println(ConsoleColors.ANSI_RED + "Player Life Remaining: 5");
    break;
case 2:
    System.out.println(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong
        guess, please try another letter");
    System.out.println(ConsoleColors.ANSI_RED + "Player Life Remaining: 4");
    break;
case 3:
    System.out.println(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong
        guess, please try another letter");
    System.out.println(ConsoleColors.ANSI_RED + "Player Life Remaining: 3");
    break;
case 4:
    System.out.println(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong
        guess, please try another letter");
    System.out.println(ConsoleColors.ANSI_RED + "Player Life Remaining: 2");
    break;
case 5:
    System.out.println(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong
        guess, please try another letter");
    System.out.println(ConsoleColors.ANSI_RED + "Player Life Remaining: 1");
    break;
case 6:
    System.out.println(ConsoleColors.ANSI_RED + "The game is over!");
    System.out.println(ConsoleColors.ANSI_RED + "Oh no...! Unfortunately you have
        lost the game. The word was \"" + word + "\". Better luck next time");
    break;

```

```

    }
}
}

```

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

- Program (After Screenshots)

```

package hangman;

import java.util.Scanner; //Import scanner class to take input from user
import hangman.ConsoleColors; //Import ConsoleColors from hangman package to make output colourful

public class HangmanGame {

    //list of words for player to guess
    private static String[] words = {
        "train", "chocolate", "algorithm", "canada", "awesome", "water",
        "mango", "wafer", "joker", "grandfather", "darwin", "america",
        "light", "laptop", "computer", "amazing", "adelaide", "river",
        "ocean", "car", "noodles", "bread", "pizza", "burger"};

    private static String word = words[(int) (Math.random() * words.length)]; //get length of the word to be guessed
    private static String lettersLine = new String(new char[word.length()]).replace("\\0", "_"); //this variable hides the word with
    private static int count = 0; //declared a counter to count the wrong guesses of player

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        message("Hello, welcome to the Hangman Game. Let's play...");
        message("\nThe number of letters in this word are "+word.length());
        message(ConsoleColors.ANSI_RED + "You have 6 player lives to attempt a wrong guess.");

        while (count < 6 && lettersLine.contains("_")) {
            message("\nPlease guess a character of the word");

```



```

        message(ConsoleColors.ANSI_BLUE + lettersLine); //print the word either hidden or reveals gradually as the game
        String guessWord = sc.next(); //take guessed character from the user
        checkCharacter(guessWord); //pass the guessed character as an argument to validate it
    }
    sc.close(); //scanner object closed
}

//method to check whether the guessed character is right or wrong
public static void checkCharacter(String guessWord) {
    String newlettersLine = "";
    for (int i = 0; i < word.length(); i++) {
        if (word.charAt(i) == guessWord.charAt(0)) {
            newlettersLine += guessWord.charAt(0); //reveals characters if guessed character is true
        } else if (lettersLine.charAt(i) != '_') {
            newlettersLine += word.charAt(i);
        } else {
            newlettersLine += "_";
        }
    }

    if (lettersLine.equals(newlettersLine)) {
        count++; //increase counter if wrong guessed
        deductLife(); //call the method to deduct life when wrong guessed
    } else {
        lettersLine = newlettersLine; //updates the lettersLine
    }

    //when guessed the whole word correctly, print the following message

```

```

    if (lettersLine.equals(word)) {
        message(ConsoleColors.ANSI_BLUE + "Woohoo...! You guessed the word correctly. You won! The word was \" + word+"\"");
    }
}

//function containing if else conditions to print relevant messages according to the life remaining
public static void deductLife() {
    if (count == 1) {
        message(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong guess, please try another letter");
        message(ConsoleColors.ANSI_RED + "Player Life Remaining: 5");
    }
    if (count == 2) {
        message(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong guess, please try another letter");
        message(ConsoleColors.ANSI_RED + "Player Life Remaining: 4");
    }
    if (count == 3) {
        message(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong guess, please try another letter");
        message(ConsoleColors.ANSI_RED + "Player Life Remaining: 3");
    }
    if (count == 4) {
        message(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong guess, please try another letter");
        message(ConsoleColors.ANSI_RED + "Player Life Remaining: 2");
    }
    if (count == 5) {
        message(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong guess, please try another letter");
        message(ConsoleColors.ANSI_RED + "Player Life Remaining: 1");
    }
}

```

```

        if (count == 6) {
            message(ConsoleColors.ANSI_RED + "The game is over!");
            message(ConsoleColors.ANSI_RED + "Oh no...! Unfortunately you have lost the game. The word was \"" + word + "\".");
        }
    }

    //method to avoid repeated usage of System.out.println
    public static void message(String inputMessage)
    {
        System.out.println(inputMessage);
    }
}

```

The actual “after” code in the text format is as below:

```
package hangman;
```

```
import java.util.Scanner;
```

```
import hangman.ConsoleColors;
```

```
public class HangmanGame {
```

```
    private static String[] words = {
```

```
        "train", "chocolate", "algorithm", "canada", "awesome", "water",
        "mango", "wafer", "joker", "grandfather", "darwin", "america",
        "light", "laptop", "computer", "amazing", "adelaide", "river",
        "ocean", "car", "noodles", "bread", "pizza", "burger"};
```

```
    private static String word = words[(int) (Math.random() * words.length)];
```

```
    private static String lettersLine = new String(new char[word.length()]).replace("\0", "_");
```

```
    private static int count = 0;
```

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

```
        Scanner sc = new Scanner(System.in);
```

```
        message("Hello, welcome to the Hangman Game. Let's play...");
```

```
        message("\nThe number of letters in this word are "+word.length());
```

```
        message(ConsoleColors.ANSI_RED + "You have 6 player lives to attempt a wrong
guess.");
```

```
        while (count < 6 && lettersLine.contains("_")) {
```

```
            message("\nPlease guess a character of the word");
```

```
            message(ConsoleColors.ANSI_BLUE + lettersLine);
```

```
            String guessWord = sc.next();
```

```
            checkCharacter(guessWord);
```

```
        }
```

```

        sc.close();
    }

    public static void checkCharacter(String guessWord) {
        String newlettersLine = "";
        for (int i = 0; i < word.length(); i++) {
            if (word.charAt(i) == guessWord.charAt(0)) {
                newlettersLine += guessWord.charAt(0);
            } else if (lettersLine.charAt(i) != '_') {
                newlettersLine += word.charAt(i);
            } else {
                newlettersLine += "_";
            }
        }

        if (lettersLine.equals(newlettersLine)) {
            count++;
            deductLife();
        } else {
            lettersLine = newlettersLine;
        }
        if (lettersLine.equals(word)) {
            message(ConsoleColors.ANSI_BLUE + "Woohoo..! You guessed the
word correctly. You won! The word was \"" + word + "\"");
        }
    }

    public static void deductLife() {
        if (count == 1) {
            message(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong
guess, please try another letter");
            message(ConsoleColors.ANSI_RED + "Player Life Remaining: 5");
        }
        if (count == 2) {
            message(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong
guess, please try another letter");
            message(ConsoleColors.ANSI_RED + "Player Life Remaining: 4");
        }
    }

```

```

        if (count == 3) {
            message(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong
guess, please try another letter");
            message(ConsoleColors.ANSI_RED + "Player Life Remaining: 3");
        }
        if (count == 4) {
            message(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong
guess, please try another letter");
            message(ConsoleColors.ANSI_RED + "Player Life Remaining: 2");
        }
        if (count == 5) {
            message(ConsoleColors.ANSI_RED + "Oops...! You have made a wrong
guess, please try another letter");
            message(ConsoleColors.ANSI_RED + "Player Life Remaining: 1");
        }
        if (count == 6) {
            message(ConsoleColors.ANSI_RED + "The game is over!");
            message(ConsoleColors.ANSI_RED + "Oh no...! Unfortunately you have
lost the game. The word was \"" + word + "\". Better luck next time");
        }
    }

    public static void message(String inputMessage)
    {
        System.out.println(inputMessage);
    }
}

```

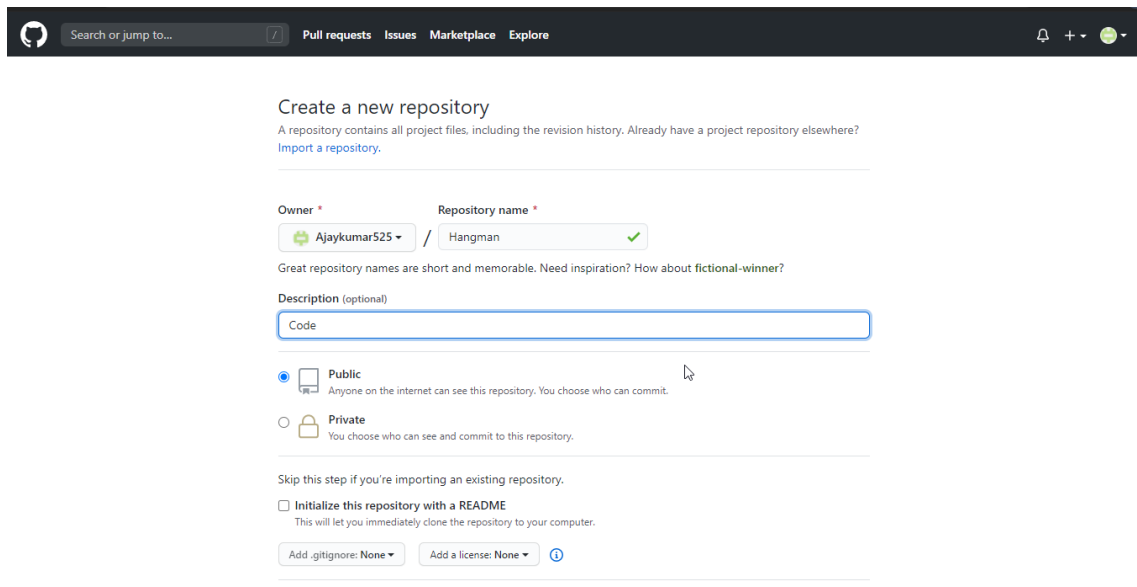
Code Smells:

- **Global Variable Class** found in the code. I have removed the global variables from the code.
- **Global Function Class** found in the code. I have removed the global functions from the code.
- **Variable Names** should always complete and meaningful. I have renamed the methods and variables names so that it do not become misleading or confusing.
- **Switch Statement** found in the code. I have removed the switch statement and replaced it with if-else condition.

- Method named **message** is created to avoid repeated usage of “System.out.println()” in order to print messages and output.
- Suitable and relevant comments are added to the program to increase readability of code

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



Creating a github repository to push the code on github. You can save a variety of projects in the GitHub Enterprise archives, including open source projects. With open source projects, you can share code to make better and more reliable software. You can create public repositories for an open source project. When creating your public domain, be sure to include a license file that determines how you want your project to be shared with others.



Create a new repository

A repository contains all project files, including the revision history. Already have a project repository elsewhere? [Import a repository.](#)


Owner ^{*} Repository name ^{*}


 Ajaykumar525 / Hangman 

Great repository names are short and memorable. Need inspiration? How about fictional-winner?

Description (optional)


Code

☒  **Public**
Anyone on the internet can see this repository. You choose who can commit.

☐  **Private**
You choose who can see and commit to this repository.

Skip this step if you're importing an existing repository.

☐ **Initialize this repository with a README**
This will let you immediately clone the repository to your computer.

Add .gitignore: **None** Add a license: **None** 

Github repository is created successfully and now we can upload the code on it. This is used for the version control.

Search or jump to... Pull requests Issues Marketplace Explore

Ajaykumar525 / Hangman Unwatch 1 Star 0 Fork 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

master 2 branches 0 tags Go to file Add file Code

Ajaykumar525 Add files via upload 5f022aa 1 minute ago 14 commits

Hangman	Add files via upload	1 minute ago
README.md	Create README.md	3 minutes ago

README.md

Hangman

About No description, website, or topics provided.

Readme

Releases No releases published. [Create a new release](#)

Packages No packages published. [Publish your first package](#)

You can upload and upload an existing file to the GitHub Enterprise repository. Drag and drop the file into any directory in the file tree, or upload files to the homepage. Files you add to the browser repository are limited to 25 MB per file. You can add large files, up to 100 MB each, per command line. If you would like to upload a new file to an empty repository, you must first start or set up your repository.

Search or jump to... Pull requests Issues Marketplace Explore

Ajaykumar525 / Hangman Unwatch 1 Star 0 Fork 0

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Hangman /

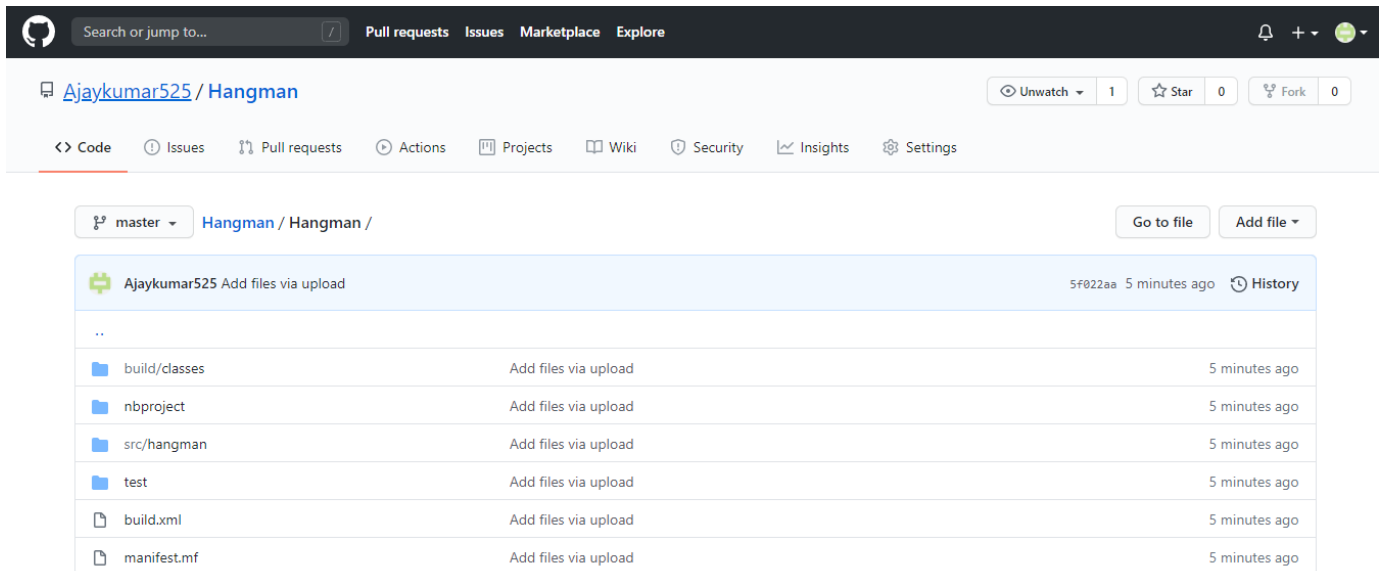
Drag files here to add them to your repository
Or [choose your files](#)

Commit changes

Add files via upload

Add an optional extended description...

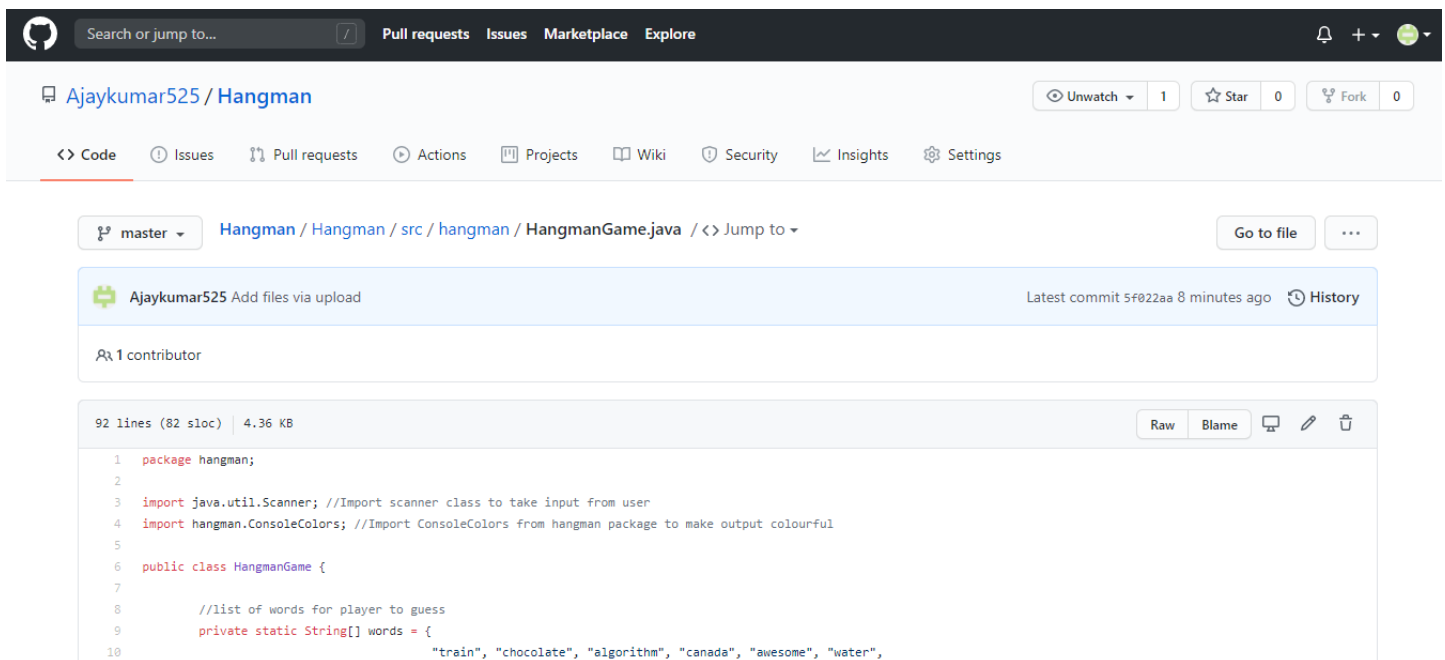
Finally, the code is uploaded on the github repository. You can use this for version control and using also for the reuse of the code.



The screenshot shows the GitHub repository page for 'Ajaykumar525 / Hangman'. The repository has 1 pull request, 0 stars, and 0 forks. The file list shows the following files and folders:

File/Folder	Commit	Time
..	5f022aa	5 minutes ago
build/classes	Add files via upload	5 minutes ago
nbproject	Add files via upload	5 minutes ago
src/hangman	Add files via upload	5 minutes ago
test	Add files via upload	5 minutes ago
build.xml	Add files via upload	5 minutes ago
manifest.mf	Add files via upload	5 minutes ago

You can see the code file. Code file is opened and you can make changes, update and delete this code whenever you want.



The screenshot shows the GitHub repository page for 'Ajaykumar525 / Hangman' with the file 'src/hangman/HangmanGame.java' open. The file is 92 lines (82 sloc) and 4.36 KB. The code is as follows:

```
1 package hangman;
2
3 import java.util.Scanner; //Import scanner class to take input from user
4 import hangman.ConsoleColors; //Import ConsoleColors from hangman package to make output colourful
5
6 public class HangmanGame {
7
8     //list of words for player to guess
9     private static String[] words = {
10         "train", "chocolate", "algorithm", "canada", "awesome", "water",
```

Q4: How TDD has been implemented to create your program

In this part, I have demonstrated the whole process of TDD and how actually the Hangman game was developed using Test Driven Development.

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 method() testing");
    }
}
```



```

    String[] args = null;
    Hangman.main(args);
    fail("Not yet implemented");
}

@Test
public void testCheckCharacter() {
    System.out.println("checkCharacter() method testing");
    String guess = "a";
    HangmanGame.checkCharacter(guess);
    fail("The test case is a prototype.");
}

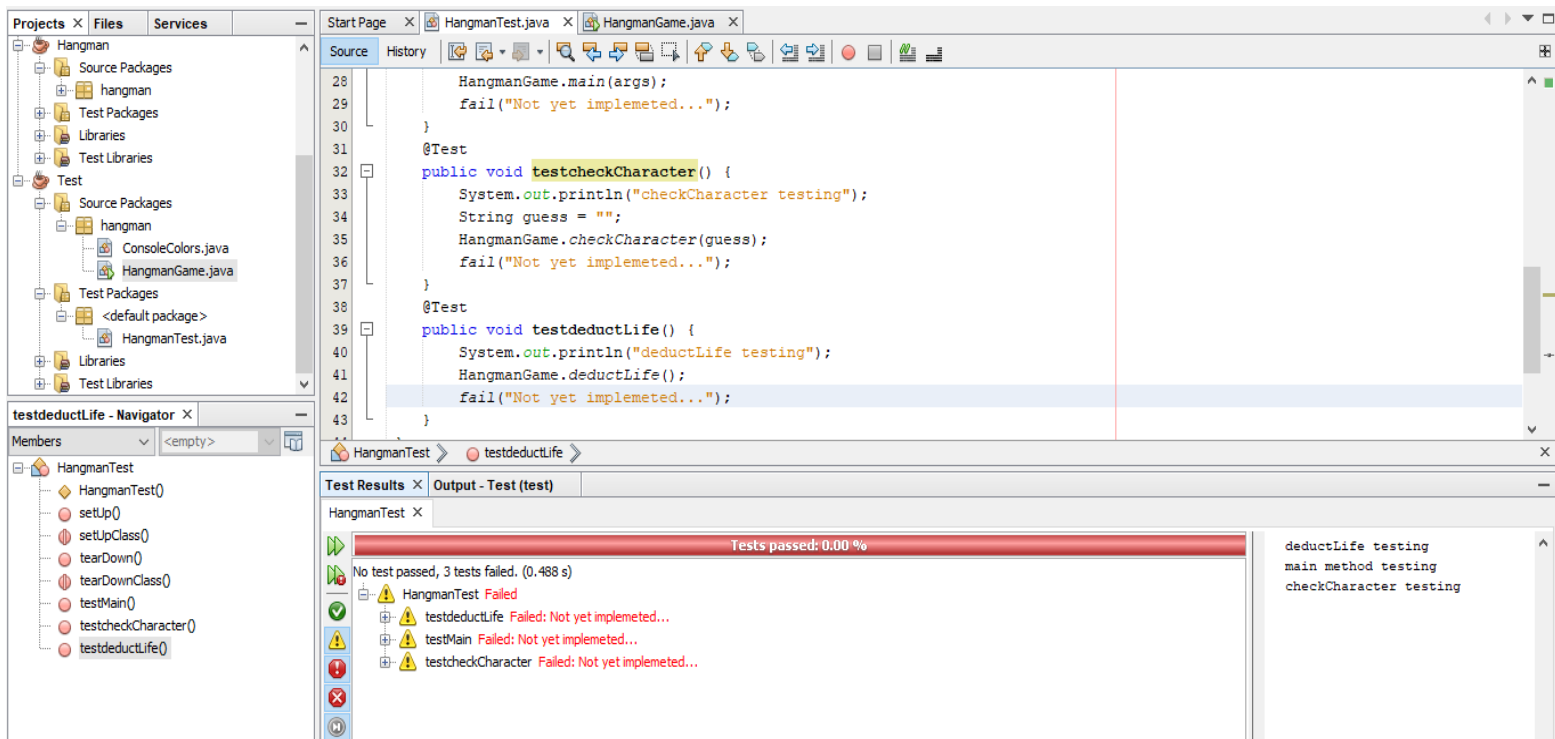
@Test
public void testDeductLife() {
    System.out.println("deductLife() method testing");
    HangmanGame.deductLife();
    fail("The test case is a prototype.");
}

@Test
public void testMessage() {
    System.out.println("message() method testing");
    String message = "Please input a character";
    HangmanGame.message(message);
    fail("Not yet implemented...");
}
}

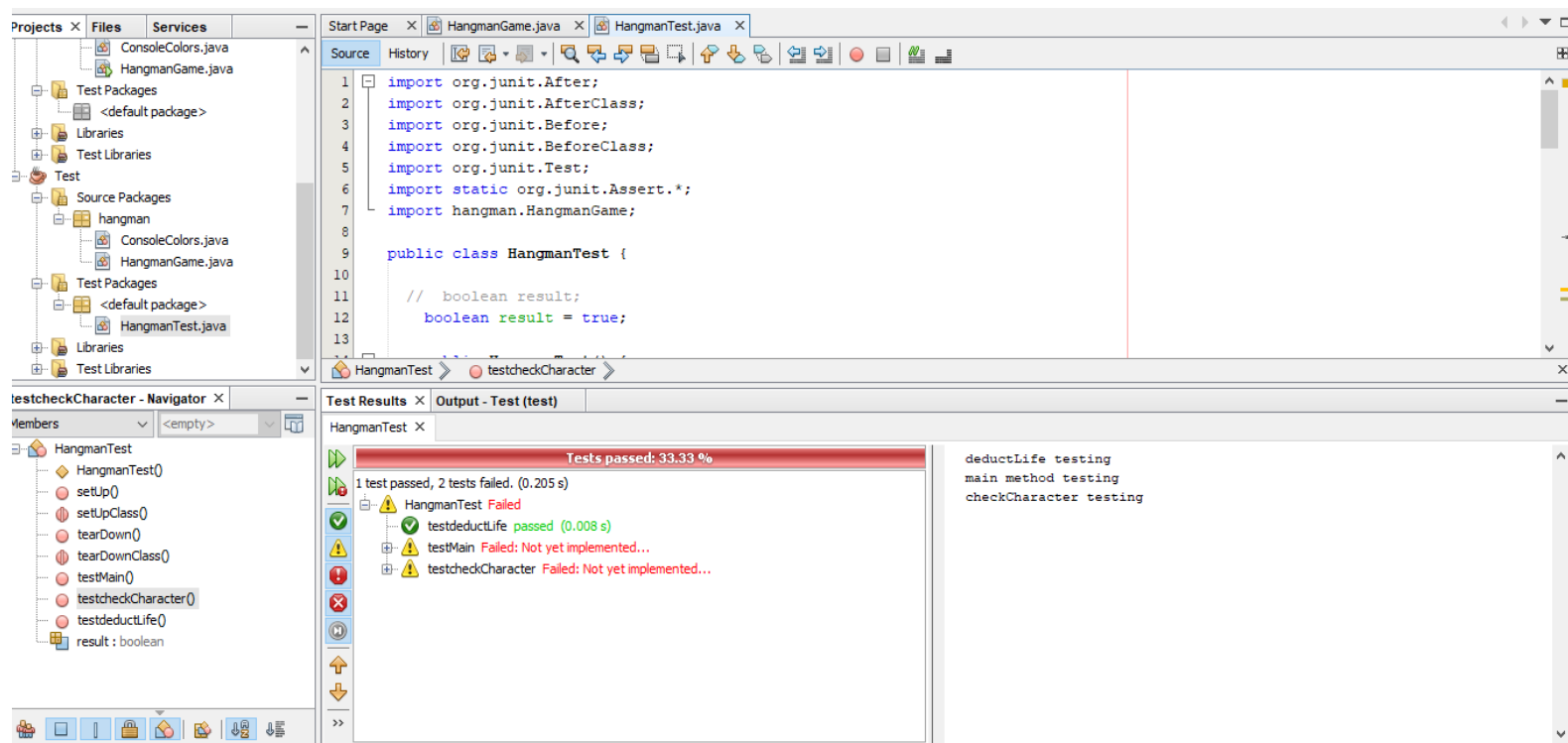
```

Testing, Refactoring and whole process of TDD using Junit:

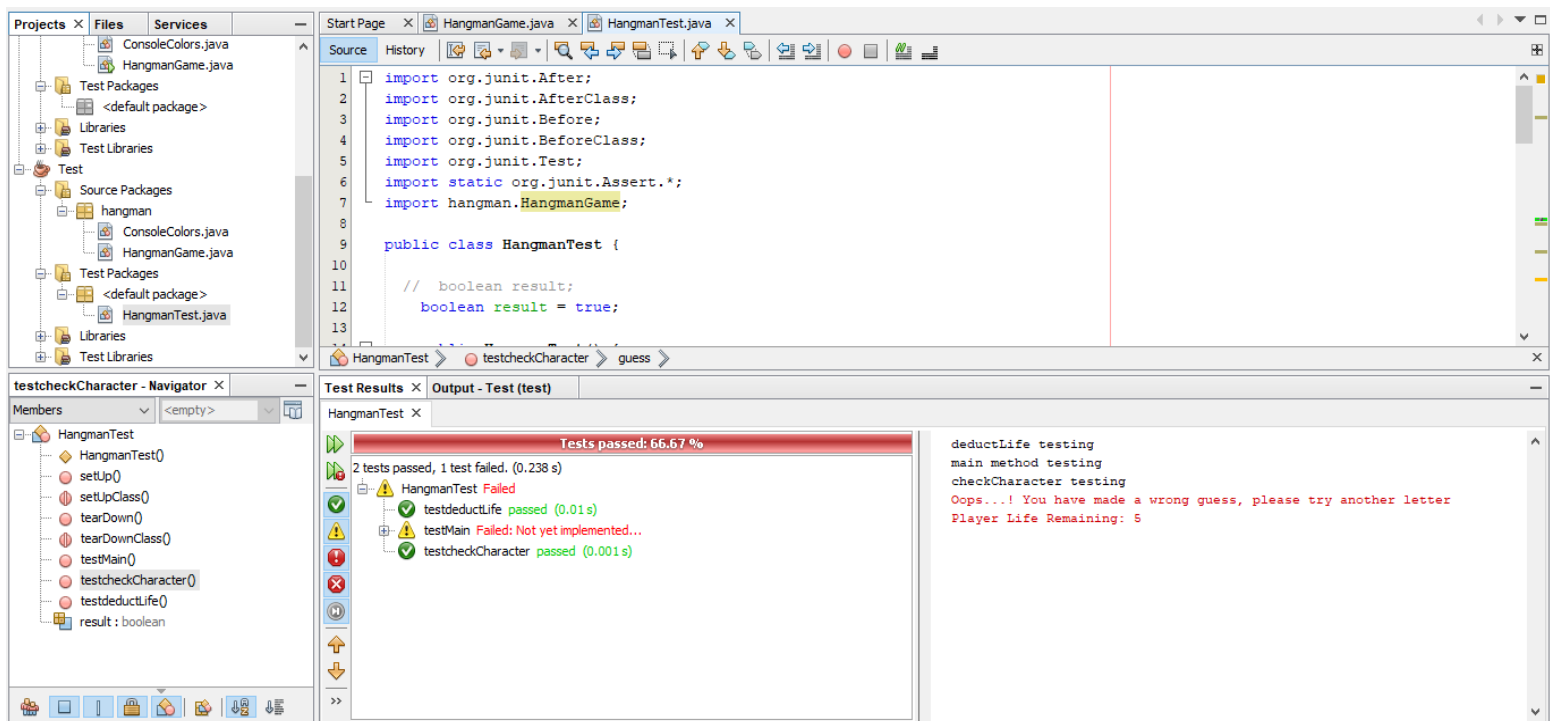
We have written a J-Unit tests for Hangman but they fails because they are not yet implemented. As we can see in the following screenshot it shows error that “Not yet implemented”. We will keep coding the Hangman game and keep testing parallelly to keep track of the issues.



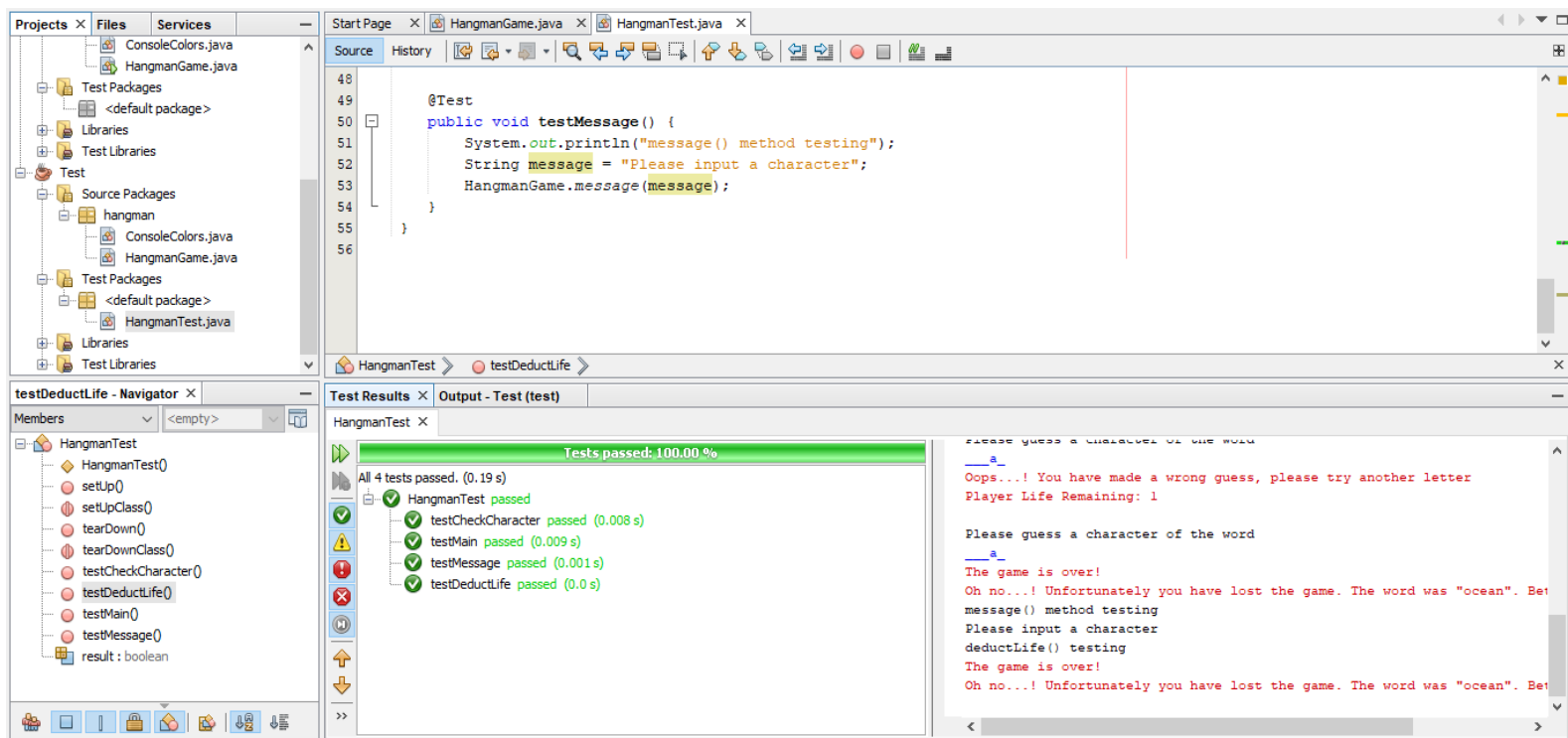
After implementing 1 method, I again tested and it passed by 33.33% as seen in the screenshot below:



After implementing 2 methods, I again tested the program and it passes by 66.66% as shown in following screenshot.



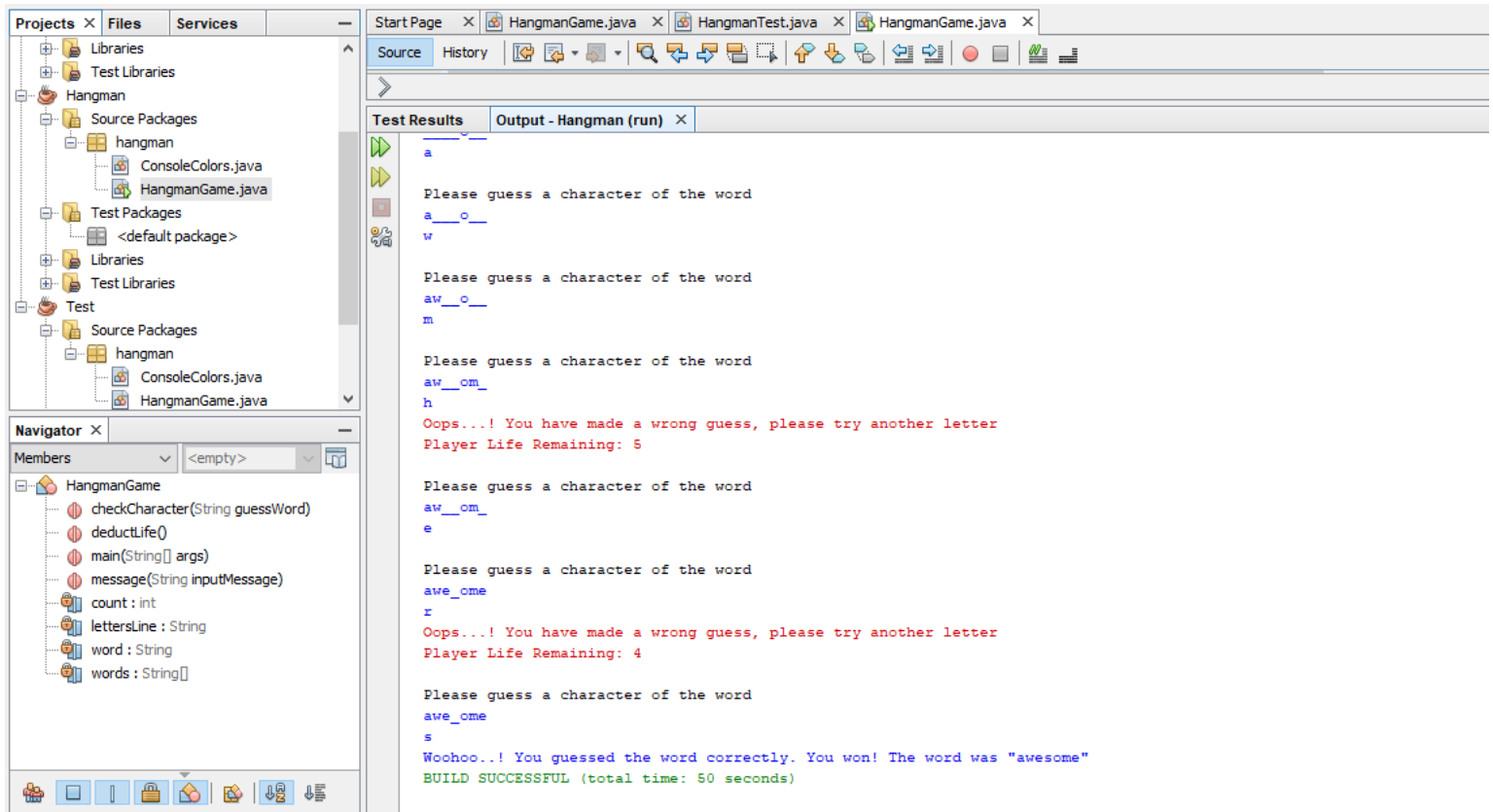
After that I implemented all the methods and also added one new method to the class. Moreover, after refactoring the code completely, and having resolved the issues I tested again and all the test cases passed as shown in screenshot below:



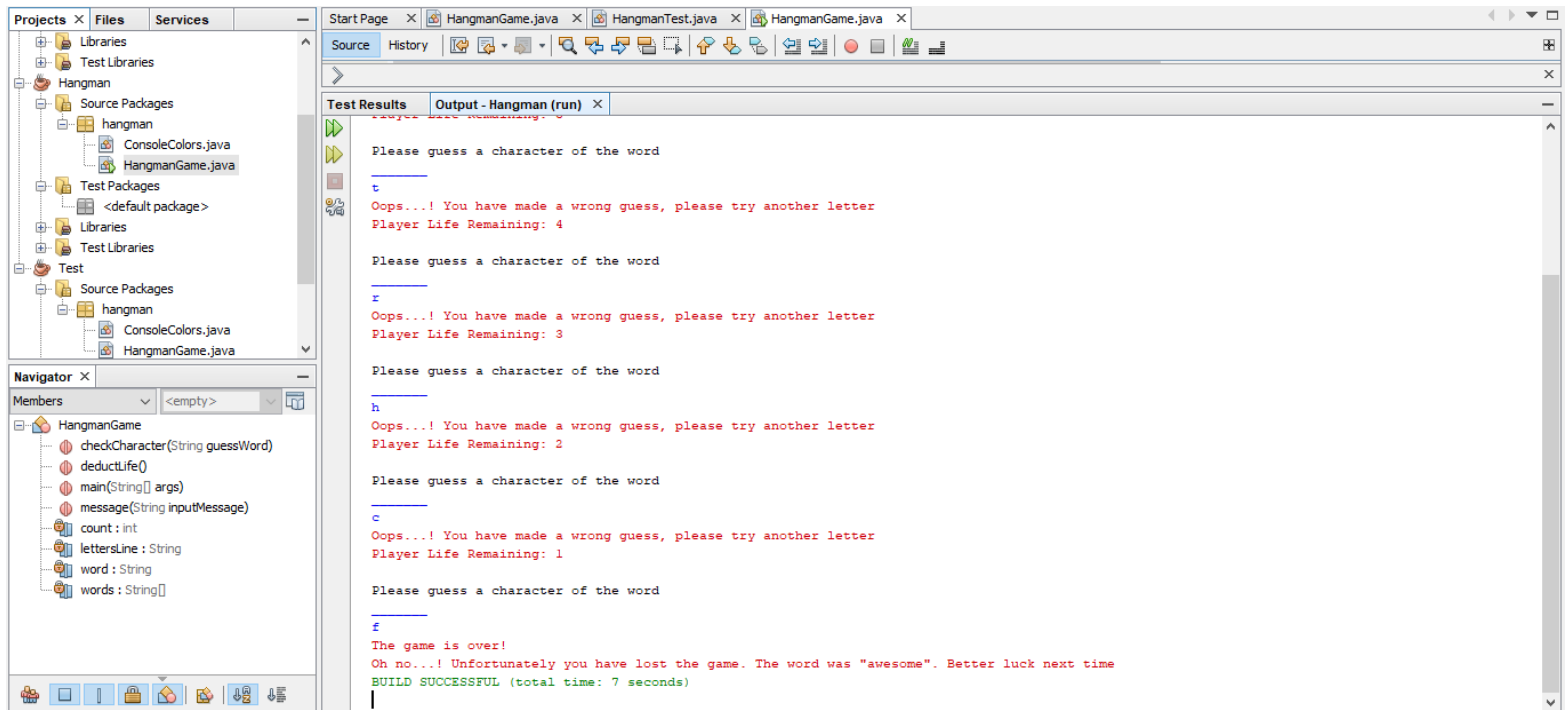
Output:

Output of the game can be shown in the following screenshot. User tries to guess the word “awesome”. Some of the guesses were correct and few were wrong but ultimately he managed to guess the whole word correctly. The hidden word is represented by a dashed line, representing each letter of the word. If user guesses a word that is present multiple times in the hidden word then all those places will be revealed. If the guessing player suggests a letter from the name, the program writes in all the relevant terms. If the suggested letter is not in the name, the program deducts the life of the player. In total 6 wrong guesses are allowed in this game.

I have used a class `ConsoleColors` to make the output look colourful which also can be seen in the following screenshot.



If the user makes 6 wrong guesses he will lose the game and message as in following screenshot will be displayed.



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.

Github Link:

- <https://github.com/Ajaykumar525/Hangman>

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.