**STQA Mini Project No. 1**

**Group members -: 1) Divyank Shah (422048 div-B)**

1. **Ankit Sonje (422060 div-B)**
2. **Sneha Ukarande (422070 div-B)**
3. **Gibraan Jafar (421037 div-A)**

# Title

Mini-Project 1: Create a small application by selecting relevant system environment/ platform and programming languages. Narrate concise Test Plan consisting features to be tested and bug taxonomy. Prepare Test Cases inclusive of Test Procedures for identified Test Scenarios. Perform selective Black-box and White-box testing covering Unit and Integration test by using suitable Testing tools. Prepare Test Reports based on Test Pass/Fail Criteria and judge the acceptance of application developed.

## Problem Definition:

Perform Desktop Application testing using Automation Tool like JUnit generate Test Report by Using tool like Apache Maven.

## Prerequisite:

Knowledge of Core Java, Basic Concepts of Unit Testing, Test Cases Writing using Junit etc tool

## Software Requirements:

JDK 1.8, Eclipse java photon-R version, TestNG

## Hardware Requirement:

PIV, 2GB RAM, 500 GB HDD, Lenovo A13-4089Model.

## Learning Objectives:

We are going to learn how to Prepare Test Cases inclusive of Test Procedures for identified Test Scenarios. Perform selective Black-box and White-box testing covering Unit and Integration test by using suitable Testing tools. also Prepare Test Reports based on Test Pass/Fail Criteria

## Outcomes:

You are able to understand Unit and Integration testing with Tool with Test Report.

## Theory Concepts:

**What is Unit Testing?**

Unit Testing of software applications is done during the development (coding) of an application. The objective of Unit Testing is to isolate a section of code and verify its correctness. In procedural programming a unit may be an individual function or procedure

The goal of Unit Testing is to isolate each part of the program and show that the individual parts are correct. Unit Testing is usually performed by the developer.

## Unit Testing Tools

There are several automated tools available to assist with unit testing. We will provide a few examples below:

1. [Jtest](https://prsft.co/2n7GdAM): Parasoft Jtest is an IDE plugin that leverages open-source frameworks (Junit, Mockito, PowerMock, and Spring) with guided and easy one-click actions for creating, scaling, and maintaining unit tests. By automating these time-consuming aspects of unit testing, it frees the developer to focus on business logic and create more meaningful test suites.
2. [Junit](https://www.guru99.com/junit-tutorial.html): Junit is a free to use testing tool used for Java programming language. It provides assertions to identify test method. This tool test data first and then inserted in the piece of code.
3. [NUnit](http://nunit.org/): NUnit is widely used unit-testing framework use for all .net languages. It is open source tool which allows writing scripts manually. It supports data-driven tests which can run in parallel.
4. [JMockit](http://jmockit.github.io/index.html): JMockit is open source Unit testing tool. It is code coverage tool with line and path metrics. It allows mocking API with recording and verification syntax. This tool offers Line coverage, Path Coverage, and Data Coverage.
5. [EMMA](http://emma.sourceforge.net/): EMMA is an open-source toolkit for analyzing and reporting code written in Java language. Emma support coverage types like method, line, basic block. It is Java-based so it is without external library dependencies and can access to the source code.
6. [PHPUnit](https://phpunit.de/): PHPUnit is a unit testing tool for PHP programmer. It takes small portions of code which is called units and test each of them separately. The tool also allows developers to use pre- define assertion methods to assert that system behave in a certain manner.

Those are just a few of the available unit testing tools. There are lots more, especially for C languages and Java, but you are sure to find a unit testing tool for your programming needs regardless of the language you use.

## Extreme Programming & Unit Testing

Unit testing in Extreme Programming involves the extensive use of testing frameworks. A unit test framework is used in order to create automated unit tests. Unit testing frameworks are not unique to extreme programming, but they are essential to it. Below we look at some of what extreme programming brings to the world of unit testing:

* Tests are written before the code
* Rely heavily on testing frameworks
* All classes in the applications are tested
* Quick and easy integration is made possible

## Bug taxonomy

Bug taxonomies help in providing fast and effective feedback so that they can easily identify possible reasons for failure of the software. Using bug taxonomy, a large number of potential bugs can be grouped into few categories.

Whenever a new bug is reported, using bug taxonomy, a tester can easily analyse and put that bug into any of these categories.

At the end of testing, Testers can understand the type of categories of bugs that frequently occurred and thereby in successive rounds of testing he can focus on writing more test cases that would help to detect such bugs. In addition, test leaders can guide their testers to focus on such frequently occurring bugs.

The summary of the Bug Taxonomy is given below,

* Requirements, Features, and Functionality Bugs
* Structural Bugs
* Data Bugs
* Coding Bugs
* Interface, Integration, and System Bugs
* Test and Test Design Bugs
* Testing and Design Style

## What is Integration Testing?

In integration Testing, individual software modules are integrated logically and tested as a group. A typical software project consists of multiple software modules, coded by different programmers. integration Testing focuses on checking data communication amongst these modules. Hence it is also termed as 'I & T' (Integration and Testing), 'String Testing' and sometimes 'Thread Testing **Integration Test Case:**

Integration [Test Case](https://www.guru99.com/test-case.html) differs from other test cases in the sense it **focuses mainly on the interfaces & flow of data/information between the modules**. Here priority is to be given for the **integrating links** rather than the unit functions which are already tested.

Sample Integration Test Cases for the following scenario: Application has 3 modules say 'Login Page', 'Mail box' and 'Delete mails' and each of them are integrated logically.

Here do not concentrate much on the Login Page testing as it's already been done in [Unit Testing.](https://www.guru99.com/unit-testing-guide.html) But check how it's linked to the Mail Box Page.

Similarly Mail Box: Check its integration to the Delete Mails Module.

|  |  |  |  |
| --- | --- | --- | --- |
| **Test**  **Case ID** | **Test Case Objective** | **Test Case Description** | **Expected Result** |
| **1** | Check the interface link between the Login and Mailbox module | Enter login credentials and click on the Login button | To be directed to the Mail Box |
| **2** | Check the interface link between the Mailbox and Delete Mails Module | From Mail box select the an email and click delete button | Selected email should appear in the Deleted/Trash folder |

## Desktop Application Testing by Using Junit Tool What is Junit?

JUnit is a framework for implementing testing in Java.

It provides a simple way to explicitly test specific areas of a Java program, it is extensible and can be employed to test a hierarchy of program code either singularly or as multiple units.

Why use a testing framework? Using a testing framework is beneficial because it forces you to explicitly declare the expected results of specific program execution routes. When debugging it is possible to write a

test which expresses the result you are trying to achieve and then debug until the test comes out positive. By having a set of tests that test all the core components of the project it is possible to modify specific areas of the project and immediately see the effect the modifications have on the other areas by the

results of the test, hence, side-effects can be quickly realized.

JUnit promotes the idea of first testing then coding, in that it is possible to setup test data for a unit which defines what the expected output is and then code until the tests pass. It is believed by some that this practice of "test a little, code a little, test a little, code a little..." increases programmer productivity and stability of program code whilst reducing programmer stress and the time spent debugging.

JUnit is a simple open source Java testing framework used to write and run repeatable automated tests.

It is an instance of the xUnit architecture for unit testing framework. Eclipse supports creating test cases and running test suites, so it is easy to use for your Java applications.

JUnit features include:

* Assertions for testing expected results
* Test fixtures for sharing common test data
* Test suites for easily organizing and running tests
* Graphical and textual test runners

**Sample Code**

Input - import javax.swing.\*;

import java.awt.event.\*;

import java.awt.\*;

public class Hangman implements ActionListener{

public JFrame frame;

public JTextField userInput;

public JLabel textContents;

public JLabel letters;

public JLabel wordselected;

String array = "ant bat cat dog fish goat hen lion monkey ox penguin rat snake tiger whale yak zebra";

String[] split=array.split("\\ ");

int i=(int)(Math.random()\*17);

public String word=split[i];

public int correctGuesses;

public int incorrectGuesses;

public String guessesLeft;

public boolean lose = false;

public boolean flag;

public StringBuilder wordsGuessedCorrectly;

Hangman() {

incorrectGuesses = 0;

correctGuesses = 0;

guessesLeft = "You have " + (6 - incorrectGuesses) +" chances left to guess a " + word.length() + " letter word";

wordsGuessedCorrectly = new StringBuilder();

frame = new JFrame("A Hangman Game");

frame.setSize(1380,720);

frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);

frame.setLayout(new GridLayout(4, 1));

userInput = new JTextField();

userInput.setHorizontalAlignment(JLabel.CENTER);

userInput.addActionListener(this);

frame.add(userInput);

textContents = new JLabel();

textContents.setText(guessesLeft);

textContents.setHorizontalAlignment(JLabel.CENTER);

textContents.setBorder(BorderFactory.createLineBorder(Color.BLUE));

frame.add(textContents);

letters = new JLabel("The letters you guess correctly go down here!");

letters.setHorizontalAlignment(JLabel.CENTER);

letters.setBorder(BorderFactory.createLineBorder(Color.RED));

frame.add(letters);

wordselected =new JLabel("Hint:It is an animal");

wordselected.setHorizontalAlignment(JLabel.CENTER);

frame.add(wordselected);

frame.setVisible(true);

}

String seperateLetter(String a, int x, int y) {

return a.substring(x, y);

}

void testLetter(String a) {

flag=false;

System.out.println(word);

if (word.contains(a) && wordsGuessedCorrectly.toString().contains(a) == false && correctGuesses != (word.length() - 1)) {

correctGuesses++;

wordsGuessedCorrectly.append(a);

letters.setText(wordsGuessedCorrectly.toString());

textContents.setText("Correct Guess!");

flag=true;

}

else if (word.contains(a) && wordsGuessedCorrectly.toString().contains(a)) {

textContents.setText("You've already guessed this letter!");

flag=true;

}

else if (word.contains(a) && wordsGuessedCorrectly.toString().contains(a) == false && correctGuesses == (word.length() - 1)) {

lose = true;

textContents.setText("You Win!");

wordsGuessedCorrectly.append(a);

letters.setText(wordsGuessedCorrectly.toString());

wordselected.setText("You guessed it right, the correct word was "+word);

flag=true;

}

else if (word.contains(a) == false && incorrectGuesses == 5){

textContents.setText("You lose!");

wordselected.setText("Ohh.. hard luck this time, the correct word was "+word);

lose = true;

}

else {

incorrectGuesses++;

textContents.setText("Incorrect Guess! You have " + (6 - incorrectGuesses) + " left.");

}

}

public void actionPerformed(ActionEvent ae) {

if (lose == false) {

String a=userInput.getText();

testLetter(seperateLetter(a, 0 , 1));

userInput.setText("");

}

}

public static void main(String[] args) {

SwingUtilities.invokeLater(new Runnable() {

public void run() {

new Hangman();

}

});

}

}

Test cases -

import static org.junit.Assert.\*;

import org.junit.Test;

public class HangmanTest {

Hangman h=new Hangman();

@Test() //To test whether correctGuess variable was initialized properly or not

public void testHangman1() {

if(h.correctGuesses==0) {

System.out.println("correctGuess was initialized successfully to 0");

assertTrue(true);

}

else

{

System.out.println("correctGuess was not intialized properly");

assertTrue(false);

}

}

@Test //To test whether incorrectGuess variable was initialized properly or not

public void testHangman2() {

if(h.incorrectGuesses==0) {

System.out.println("incorrectGuess was initialized successfully to 0");

assertTrue(true);

}

else

{

System.out.println("incorrectGuess was not intialized properly");

assertTrue(false);

}

}

@Test //To test whether userInput frame was initialized properly or not

public void testHangman3() {

if("".equals(h.userInput.getText())) {

System.out.println("userInput frame was initialized successfully");

assertTrue(true);

}

else

{

System.out.println("userInput frames was not intialized properly");

assertTrue(false);

}

}

@Test //To test whether letter frame was initialized properly or not

public void testHangman4() {

if("The letters you guess correctly go down here!".equals(h.letters.getText())) {

System.out.println("Letters frame was initialized successfully");

assertTrue(true);

}

else

{

System.out.println("Letters frame was not intialized properly");

assertTrue(false);

}

}

@Test //To test whether wordselected frame was initialized properly or not

public void testHangman5() {

if("Hint:It is an animal".equals(h.wordselected.getText())) {

System.out.println("wordselected frame was initialized successfully");

assertTrue(true);

}

else

{

System.out.println("wordselected frame was not intialized properly");

assertTrue(false);

}

}

@Test //To test whether SeperateLetter function is working properly for single alphabet or not

public void testSeperateLetter1() {

String a=h.seperateLetter("a", 0, 1);

if("a".equals(a))

{

System.out.println("Seperate Letter is working propely for single alphabet");

assertTrue(true);

}

else

{

System.out.println("Seperate letter is not working properly for single alphabet");

assertTrue(false);

}

}

@Test //To test whether SeperateLetter function is working properly for double alphabet or not

public void testSeperateLetter2() {

String a=h.seperateLetter("bc", 0, 1);

if("b".equals(a))

{

System.out.println("Seperate Letter is working propely for double alphabet");

assertTrue(true);

}

else

{

System.out.println("Seperate letter is not working properly for double alphabet");

assertTrue(false);

}

}

@Test //To test whether SeperateLetter function is working properly for triple alphabet or not

public void testSeperateLetter3() {

String a=h.seperateLetter("cde", 0, 1);

if("c".equals(a))

{

System.out.println("Seperate Letter is working propely for triple alphabet");

assertTrue(true);

}

else

{

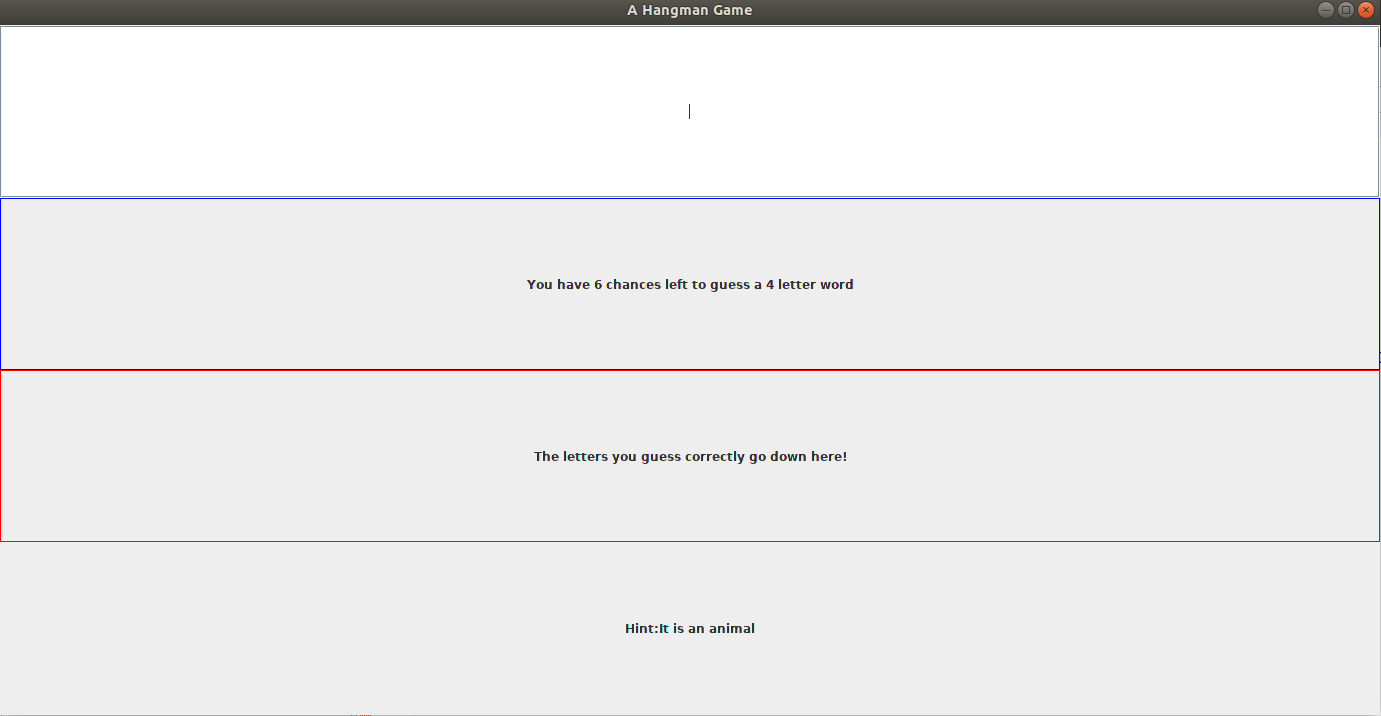
System.out.println("Seperate letter is not working properly for triple alphabet");

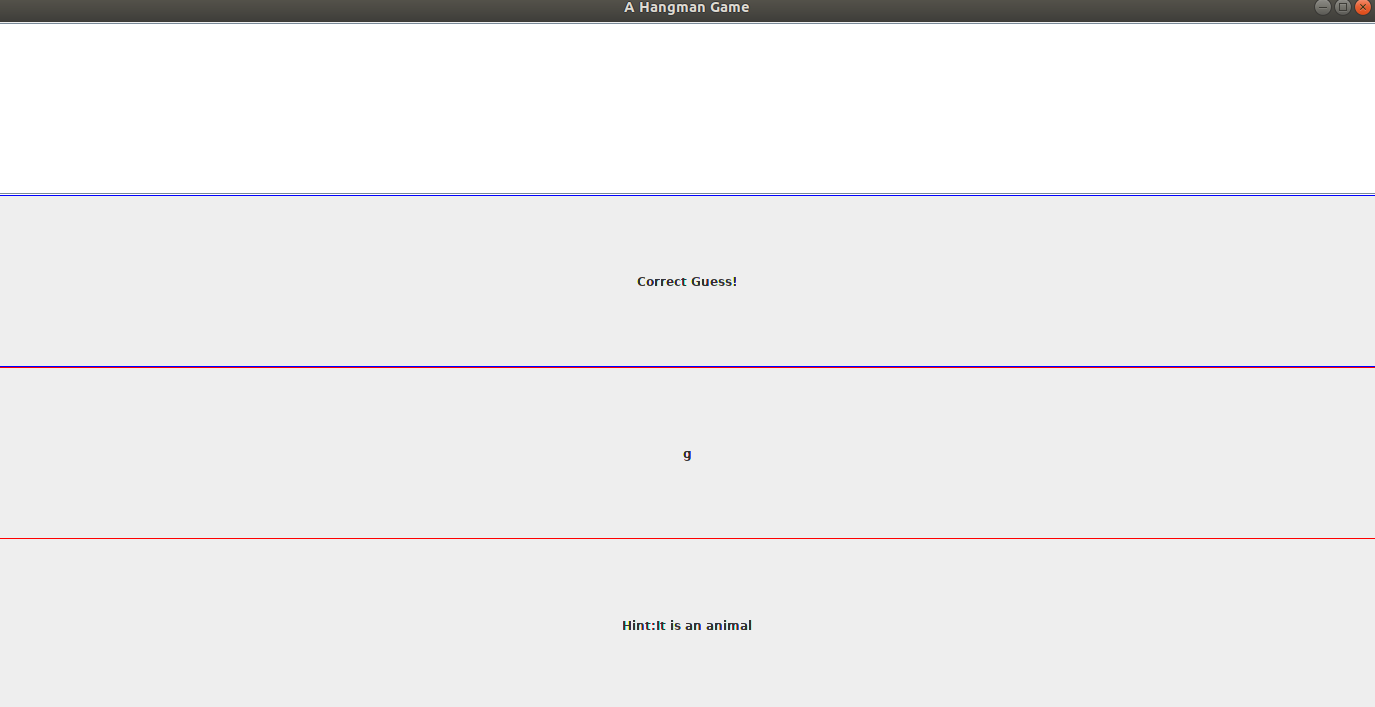
assertTrue(false);

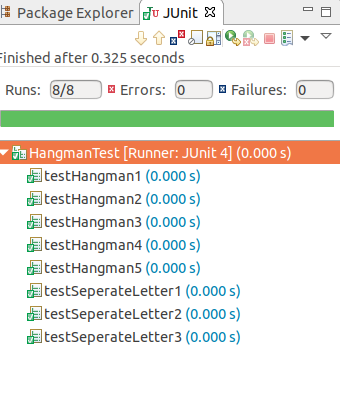
}

}

}

Output - : 





**Conclusion**

In this way using JUnit and Maven Automation tool we are Perform Unit Testing and Prepare Test Report of same.