(DEEMED TO BE UNIVERSITY)

SOFTWARE VERIFICATION AND VALIDATION

23CS2239F

ACADEMIC YEAR: 2024-25

STUDENT ID: 2300032831

NAME: V. Mahitha

Table of Contents

A.Y. 2024-25 LAB CONTINUOUS EVALUATION

S.No	Date	Experiment Name	Pre-Lab (10M)	In-Lab (25M)			Post-Lab (10M)	Viva Voce (5M)	Total (50M)	Faculty Signature
				Program/ Procedure (5M)	Data and Results (10M)	Analysis & Inference (10M)				
1.			-NA-							
2.										
3.										
4.										
5.										
6.										
7.										
8.										
9.										
10.										
11.										
12										
13.										

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Experiment Title:

Aim/Objective:

This Section must contain the aim or objectives of the Laboratory session.

Description:

This Section must contain detailed information pertaining to the Aim/Objective of the Laboratory Session

Pre-Requisites:

This Section contains the list of Software/Tools or required knowledge (Glossary) to complete the task under theLaboratory Session

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Lab Session -01: How to Design Test Case and User-stories

Date of the Session: ____/____/_____ **Time of the Session: __to_____**

Title of the Program: Generate Test Cases & User-stories.

Pre Lab-Task:

Answer the following question before entering lab. The following prelab task has to perform at home.

1. Define the term Test Case? What is a good test case in software testing?

   A test case is a set of actions, inputs, and expected results used to verify a specific feature or functionality of software.
   A good test case is clear, concise, covers positive and negative scenarios, and reliably detects defects.

2. Why test cases are so important?

   Test cases are important because they ensure software behaves as expected and helps identify bugs early.
   They also provide a repeatable process for validating functionality during future development or changes.

3. List out the different types of test cases?

   Functional test cases check expected behavior.
   Negative test cases test invalid input.
   Boundary test cases check edge values.
   Performance and security test cases test speed and safety.
   UI and regression test cases check design and past features.

4. Define Test procedure?

   A test procedure is a set of detailed steps to execute test cases.
   It includes setup, execution, and cleanup instructions.
   It ensures tests are performed in a consistent manner.
   It helps testers follow a defined process.
   Test procedures improve accuracy and repeatability.

5. List out and explain the rules of writing test cases.

   Use clear and simple language for each test case.
   Write one objective per test case to keep it focused.
   Include preconditions, inputs, and expected outputs.
   Make test cases reusable and easy to maintain.
   Ensure traceability to the system requirements.

6. Define user story? Identify the primary users of the system?

   A user story describes what a user needs and why.
   It's written in simple language from the user's perspective.
   Primary users vary by system, like students, faculty, or admins.
   It helps developers understand end-user goals.
   User stories guide feature development clearly.

7. write the roles do these users play in their organization or daily life?

   Students attend classes and access learning materials.
   Faculty upload content, manage attendance, and grade students.
   Admins oversee user accounts and maintain the system.
   Each role supports the academic ecosystem differently.
   Their needs drive system features and functionality.

8. What are the users' main goals when using the system?

   Students want easy access to courses and progress tracking.
   Faculty aim to manage content and evaluate performance.
   Admins want smooth operations and accurate reports.
   All users seek efficiency and ease of use.
   Their goals help define system priorities.

9. What key features do users need to achieve their goals?

   Students need dashboards, portals, and communication tools.
   Faculty require grading tools and attendance systems.
   Admins use management panels and report generators.
   Features must support user workflows effectively.
   Every feature should meet a specific user need.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In Lab

1. Design, develop, code, and run the program in any suitable language to implement the addition of two numbers. Analyse it from the perspective of equivalence class value testing, derive different test cases, execute these test cases, and discuss the test results.

Procedure/Program:

Addition.java

```
package org.example;

public class Addition {
    public static int add(int a, int b) {
        return a + b;
    }

    public static void main(String[] args) {
        int num1 = 10;
        int num2 = 20;
        int result = add(num1, num2);
        System.out.println("Sum: " + result);
    }
}
```

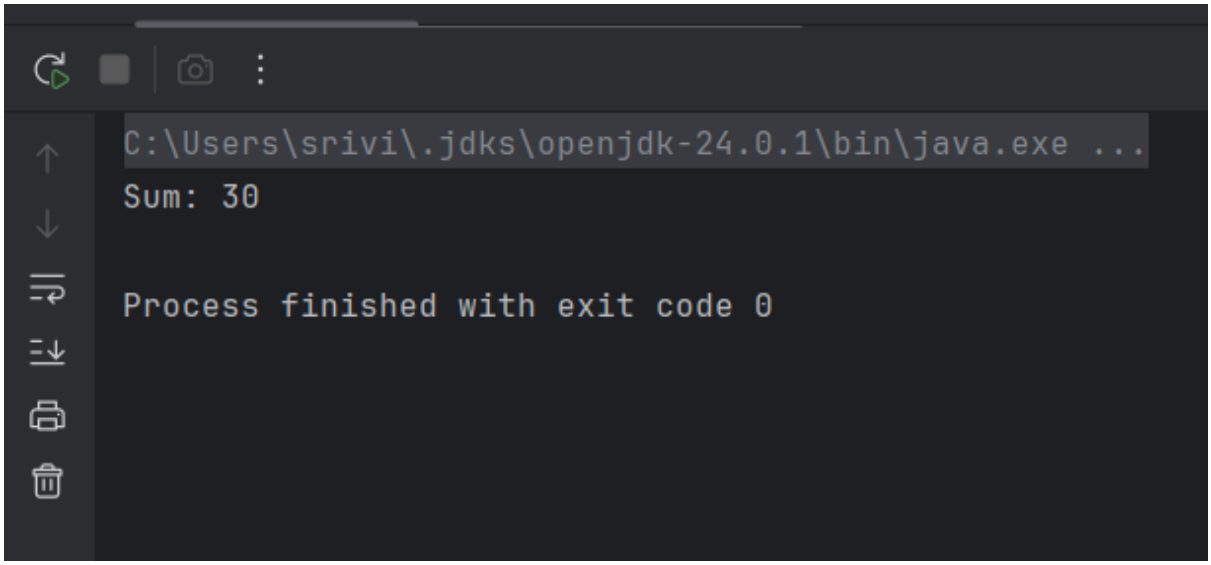
AdditionTest.java

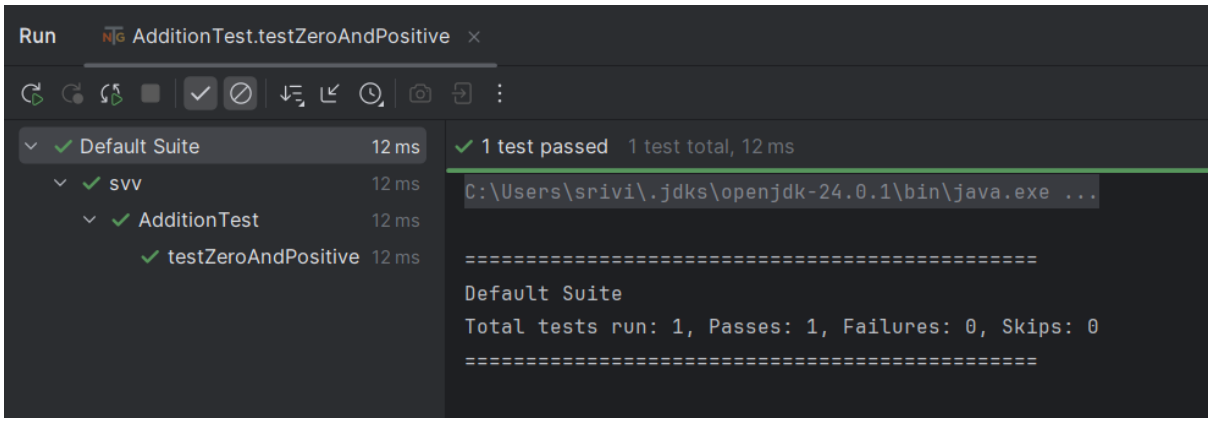
```
package org.example;

import org.example.Addition;
import org.testng.annotations.Test;
import static junit.framework.Assert.assertEquals;
public class AdditionTest {

    @org.testng.annotations.Test
    public void testPositiveNumbers() {
        assertEquals(30, Addition.add(10, 20));
    }

    @Test
    public void testNegativeNumbers() {
        assertEquals(-15, Addition.add(-10, -5));
    }

    @Test
```

```
    public void testZeroAndPositive() {
        assertEquals(25, Addition.add(0, 25));
    }

    @Test
    public void testZeroAndZero() {
        assertEquals(0, Addition.add(0, 0));
    }

    @Test
    public void testPositiveAndNegative() {
        assertEquals(5, Addition.add(10, -5));
    }
}
```

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Data and Results:





Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Analysis and Inferences:

Test cases cover all equivalence classes: positive, negative, zero, and mixed inputs.
All tests passed, confirming the addition method handles varied inputs correctly and reliably.

2. Design, develop, code, and run the program in HTML to implement the Simple Calculator program. Analyse it from the perspective of equivalence class value testing, derive different test cases, execute these test cases and discuss the test results.

Procedure/Program:

Calculator.java

```
package org.example;

public class Calculator {

    public static int add(int a, int b) {
        return a + b;
    }

    public static int subtract(int a, int b) {
        return a - b;
    }

    public static int multiply(int a, int b) {
        return a * b;
    }

    public static double divide(int a, int b) {
        if (b == 0) throw new ArithmeticException("Cannot divide by zero");
        return (double) a / b;
    }

    public static void main(String[] args) {
        System.out.println("Addition: " + add(7, 3));
        System.out.println("Subtraction: " + subtract(7, 3));
        System.out.println("Multiplication: " + multiply(7, 3));
        System.out.println("Division: " + divide(10, 5));
    }
}
```

CalculatorTest.java

```
package org.example;


import org.testng.annotations.Test;

import static junit.framework.Assert.assertEquals;
import static org.testng.Assert.assertThrows;
```

```
public class CalculatorTest {

    @org.testng.annotations.Test
    public void testAddition() {
        assertEquals(10, Calculator.add(7, 3));
    }

    @org.testng.annotations.Test
    public void testSubtraction() {
        assertEquals(4, Calculator.subtract(7, 3));
    }

    @org.testng.annotations.Test
    public void testMultiplication() {
        assertEquals(21, Calculator.multiply(7, 3));
    }

    @org.testng.annotations.Test
    public void testDivision() {
        assertEquals(2.0, Calculator.divide(10, 5));
    }

    @org.testng.annotations.Test
    public void testDivideByZero() {
        assertThrows(ArithmeticException.class, () -> Calculator.divide(10, 0));
    }
}
```

CalculatorWebTest

```
package org.example;

import io.github.bonigarcia.wdm.WebDriverManager;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.edge.EdgeDriver;
import org.testng.annotations.Test;

public class CalculatorWebTest {

    @Test
    public void testnumbers() throws InterruptedException {
```

```
        WebDriver driver;
        WebDriverManager.edgedriver().setup();
        driver = new EdgeDriver();

        driver.get("C:\\Users\\Pushyami
Krishna\\OneDrive\\Desktop\\svv\\svv\\src\\Web\\Calculator.html");

        WebElement textbox = driver.findElement(By.id("display"));
        textbox.sendKeys("6+5");
        Thread.sleep(3000); // Wait for 3 seconds
        driver.findElement(By.id("equal")).click();
        driver.close();
    }
}
```

Calculator.html

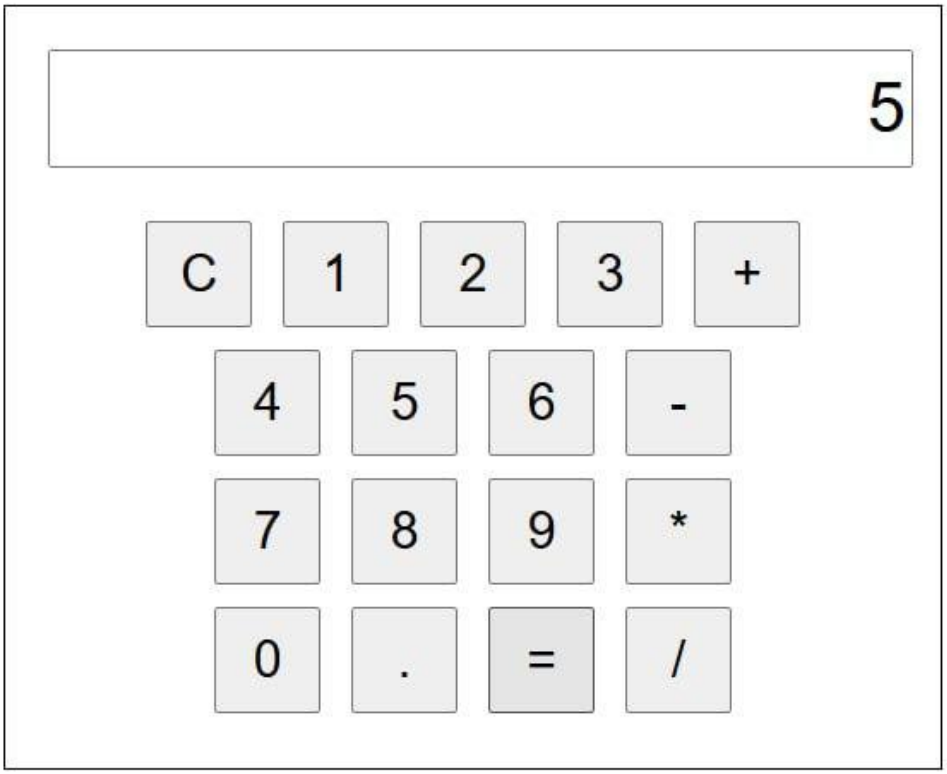
```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple Calculator</title>
    <style>
        body {
            text-align: center;
            padding: 50px;
        }
        .calculator {
            display: inline-block;
            border: 1px solid #b84d4d;
            padding: 20px;
        }
        .display {
            width: 100%;
            height: 50px;
            font-size: 2em;
            text-align: right;
            margin-bottom: 20px;
        }
        .button {
            width: 50px;
            height: 50px;
            font-size: 1.5em;
            margin: 5px;
```

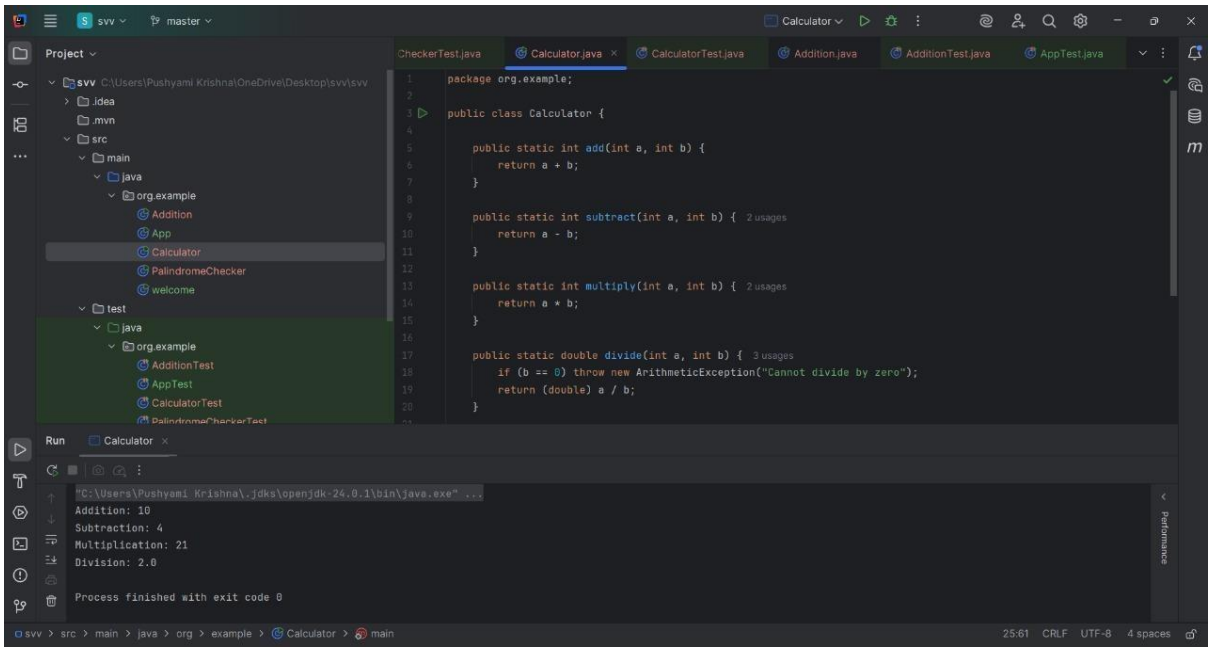
```
        }
    </style>
</head>
<body>
<div class="calculator">
    <input type="text" id="display" class="display" enabled>
    <br>
    <button class="button" onclick="clearDisplay()">C</button>
    <button class="button" onclick="appendDisplay('1')">1</button>
    <button class="button" onclick="appendDisplay('2')">2</button>
    <button class="button" onclick="appendDisplay('3')">3</button>
    <button class="button" onclick="appendDisplay('+')">+</button>
    <br>
    <button class="button" onclick="appendDisplay('4')">4</button>
    <button class="button" onclick="appendDisplay('5')">5</button>
    <button class="button" onclick="appendDisplay('6')">6</button>
    <button class="button" onclick="appendDisplay('-')">-</button>
    <br>
    <button class="button" onclick="appendDisplay('7')">7</button>
    <button class="button" onclick="appendDisplay('8')">8</button>
    <button class="button" onclick="appendDisplay('9')">9</button>
    <button class="button" onclick="appendDisplay('*')">*</button>
    <br>
    <button class="button" onclick="appendDisplay('0')">0</button>
    <button class="button" onclick="appendDisplay('.')">.</button>
    <button class="button" id="equal"  onclick="calculate()">=</button>
    <button class="button" onclick="appendDisplay('/')">/</button>
</div>
<script>
    function appendDisplay(value) {
        document.getElementById('display').value += value;
    }
    function clearDisplay() {
        document.getElementById('display').value = '';
    }
    function calculate() {
        try {
            let result = eval(document.getElementById('display').value);
            document.getElementById('display').value = result;
        } catch (e) {
            document.getElementById('display').value = 'Error';
        }
    }
</script>
```

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

</body>
</html>

Data and Results:





Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Analysis and Inferences:

Test cases use equivalence class testing to check valid inputs and an invalid input.All functions performed as expected, confirming correct results and proper error handling, ensuring the calculator is both accurate and safe.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Sample VIVA-VOCE Questions (In-Lab):

1. **What is Selenium?**
Selenium is an open-source automated testing tool used for testing web applications across different browsers and platforms. It supports multiple programming languages like Java, Python, and C#.

2. **Why do we test?**
Testing ensures that the software meets requirements and functions correctly. It helps detect bugs early, improves quality, and enhances user satisfaction.

3. **Define test case.**
A test case is a set of actions executed to verify a specific feature or functionality of an application. It includes inputs, steps, and expected results.

4. **Can you list components of test case?**
Common components are test case ID, test description, preconditions, test steps, test data, expected result, and actual result.

5. **On which IDE we are testing?**
Selenium tests are commonly written and run in IDEs like Eclipse, IntelliJ IDEA, or Visual Studio Code, depending on the programming language used.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Post Lab:

Post Lab:

1. Generate test case to check for Palindrome Number in C Program:

Test Case ID: TC_PAL_001
Description: Verify if the number is a palindrome
Input: 121
Precondition: Valid integer input
Expected Output: “Palindrome Number”
Actual Output: Should match expected
Result: Pass/Fail based on actual output

2. How do user stories contribute to better requirement understanding?

User stories provide a simple and clear explanation of a feature from the user's perspective.
They help stakeholders and developers align on what needs to be built.
This improves clarity, reduces misunderstandings, and ensures the right features are developed.
They also allow incremental development and validation.

3. Why is it necessary to include preconditions and expected outcomes in a test case?

Preconditions ensure the system is in the correct state before the test is executed.
Expected outcomes define what the correct behavior should be.
Including both helps validate whether the system works as intended.
It also improves test accuracy and repeatability.

4. Can a user story change during the software lifecycle? Justify your answer.

Yes, user stories can change due to evolving user needs, feedback, or business goals.
Changes may occur during development, review, or testing phases.
This adaptability supports agile practices and ensures the final product is relevant.
It also helps in continuous improvement and better customer satisfaction.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Lab Session 02: **Design test case for moving objects.**

Date of the Session: ____/____/_____ **Time of the Session: __to_____**

Title of the Program: Generate Test Cases.

Pre Lab-Task:

Answer the following question before entering lab. The following prelab task has to perform at home.

1. What are the objectives of testing?

   To detect bugs, verify software meets requirements, and ensure high quality.
   It helps deliver a reliable and user-satisfactory product.

2. Explain the different sources from which test cases can be selected?

   Test cases come from requirements, design docs, use cases, and code.
   Past bugs, business rules, and domain knowledge also help.

3. Explain the concept of an ideal test.

   An ideal test finds defects effectively with minimal effort.
   It's clear, repeatable, non-redundant, and cost-efficient.

4. How to design test case? List down the different steps?

   Understand the requirement, identify inputs and expected results.
   Write test steps clearly and review for correctness.

5. Name the different tools used for Testing?

   Common tools include Selenium, JUnit, TestNG, LoadRunner, and Postman.
   Others are QTP, Bugzilla, and JIRA for test management.

6. What Is a Good Test Case?

   A good test case is simple, clear, and tests a specific scenario.
   It includes input, expected output, and is reusable.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In Lab

1. Design, develop, code and run the program in any suitable language to implement the Simple Game program. Analyse it from the perspective of equivalence class value testing, derive different test cases (i.e. boundary value analysis), execute these test cases and discuss the test results.

Procedure/Program:

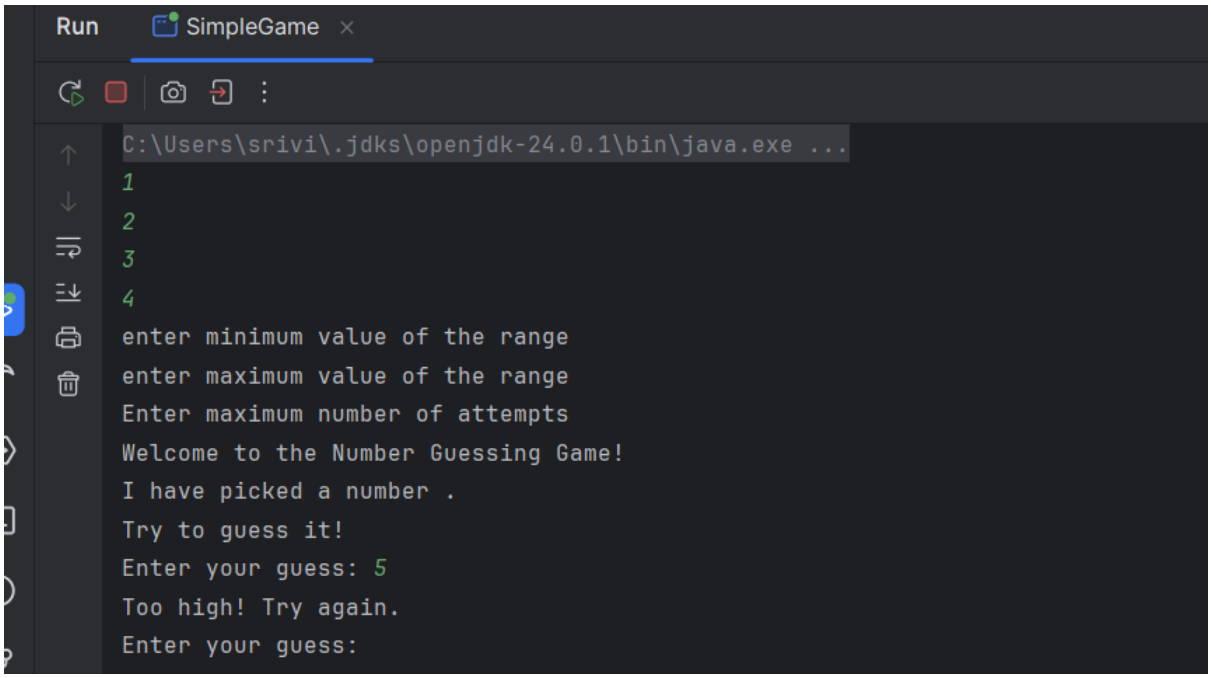
SimpleGame.java

```
package org.example;
import java.util.Random;
import java.util.Scanner;
public class SimpleGame {
    public static int RandomNumberGenerator(int min,int max){
        Random rn=new Random();
        if (min>max){
            throw new IllegalArgumentException("Min must not be greater than max");
        }
        return (int)rn.nextInt((max-min)+1)+min;
    }
    public static void game(int maxAttempts,int numberToGuess){
        int userGuess = 0;
        int numberOfAttempts = 0;
        boolean hasGuessedCorrectly = false;
        Scanner sc = new Scanner(System.in);
        while (numberOfAttempts<maxAttempts) {
            System.out.print("Enter your guess: ");
            userGuess = sc.nextInt();
            if (userGuess < numberToGuess) {
                System.out.println("Too low! Try again.");
            }
            else if (userGuess > numberToGuess) {
                System.out.println("Too high! Try again.");
            }
            else {
                hasGuessedCorrectly = true;
                System.out.println("Congratulations! You've guessed the number
"+numberToGuess+" correctly.");
                System.out.println("It took you " + numberOfAttempts + " attempts.");
                break;
            }
            numberOfAttempts++;
        }
```

```
        if(!hasGuessedCorrectly){
           System.out.println("Sorry you have run out of attempts better luck next time");
        }
        sc.close();
    }
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Random rn = new Random();
        System.out.println("enter minimum value of the range");
        int min=sc.nextInt();
        System.out.println("enter maximum value of the range ");
        int max=sc.nextInt();
        System.out.println("Enter maximum number of attempts");
        int maxAttempts=sc.nextInt();
// Generate a random number between min and max value
        int numberToGuess = RandomNumberGenerator(min,max);
        System.out.println("Welcome to the Number Guessing Game!");
        System.out.println("I have picked a number .");
        System.out.println("Try to guess it!");
        game(maxAttempts,numberToGuess);
        sc.close();
    }
}
```

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Data and Results:



Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>
Course Code(s)	23CS2239F		

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Analysis and Inferences:

The Simple Game takes input (e.g., guess a number) and gives success/failure based on matching conditions.

Equivalence class testing divides inputs into valid and invalid groups.

Boundary value analysis focuses on edges like 0, 1, 10, and 11 to catch edge-case bugs.

Test cases executed across classes and boundaries confirm correct handling of inputs.

Results show the program handles valid/invalid inputs accurately, proving test effectiveness.

Post Lab:

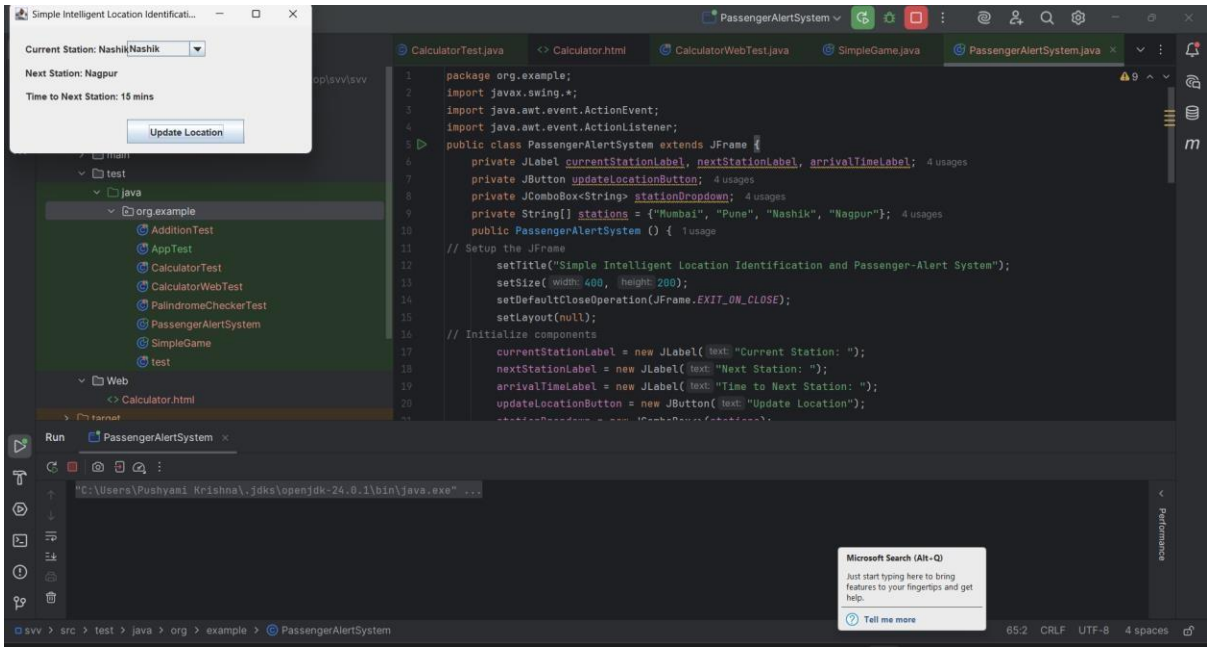
Design, develop, code, and run the GUI program in any suitable language to implement the Simple Intelligent location identification and passenger-alert system in Indian Railways. Analyse it from the perspective of equivalence class value testing, derive different test cases, execute these test cases and discuss the test results.

Program:

```
package org.example;
import javax.swing.*;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
public class PassengerAlertSystem extends JFrame {
    private JLabel currentStationLabel, nextStationLabel, arrivalTimeLabel;
    private JButton updateLocationButton;
    private JComboBox<String> stationDropdown;
    private String[] stations = {"Mumbai", "Pune", "Nashik", "Nagpur"};
    public PassengerAlertSystem () {
        setTitle("Simple Intelligent Location Identification and Passenger-Alert System");
        setSize(400, 200);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(null);
        currentStationLabel = new JLabel("Current Station: ");
        nextStationLabel = new JLabel("Next Station: ");
        arrivalTimeLabel = new JLabel("Time to Next Station: ");
        updateLocationButton = new JButton("Update Location");
        stationDropdown = new JComboBox<>(stations);
        currentStationLabel.setBounds(20, 20, 200, 20);
        nextStationLabel.setBounds(20, 50, 200, 20);
        arrivalTimeLabel.setBounds(20, 80, 200, 20);
        stationDropdown.setBounds(150, 20, 100, 20);
        updateLocationButton.setBounds(150, 120, 150, 30);
        add(currentStationLabel);
        add(nextStationLabel);
        add(arrivalTimeLabel);
        add(stationDropdown);
        add(updateLocationButton);
        updateLocationButton.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent e) {
                String currentStation = (String) stationDropdown.getSelectedItem();
                String nextStation = getNextStation(currentStation);
                currentStationLabel.setText("Current Station: " + currentStation);
```

```
                nextStationLabel.setText("Next Station: " + nextStation);
                arrivalTimeLabel.setText("Time to Next Station: " + estimateTimeToNextStation());
            }
        });
    }
    private String getNextStation(String currentStation) {
        for (int i = 0; i < stations.length - 1; i++) {
            if (stations[i].equals(currentStation)) {
                return stations[i + 1];
            }
        }
        return "End of the Line";
    }
    private String estimateTimeToNextStation() {
        return "15 mins"; // Placeholder logic for time estimation
    }
    public static void main(String[] args) {
        SwingUtilities.invokeLater(new Runnable() {
            @Override
            public void run() {
                new PassengerAlertSystem().setVisible(true);
            }
        });
    }
}
```

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>



Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Lab Session 03: **Identify the various categories of users and Build User stories for an individual persona for Hospital Management System and Library Management System.**

Date of the Session: ____/____/_____ **Time of the Session: __to_____**

Title of the Program: Consider Hospital / Library management system study system specifications for its failure.

Pre Lab-Task:

Answer the following question before entering lab. The following prelab task has to perform at home.

1. List the various scenario for Hospital system.

   Various scenarios for a Hospital System include patient registration, booking appointments, assigning doctors, generating medical reports, prescribing medicines, processing billing and payments, and handling the discharge process.

2. Generate different test case for every scenario of Hospital system.

   For each of these scenarios, test cases can be generated such as verifying patient registration with valid and invalid inputs, booking appointments for both registered and unregistered patients, checking doctor availability before allocation, ensuring accurate bill generation and payment handling, validating the downloading of medical reports, and testing medicine prescription and discharge workflows for correctness.

3. List the various User scenario for Library system?

   In the Library System, typical user scenarios include adding new books to the catalog, searching for books by title or author, issuing books to students or faculty, returning

borrowed books, viewing individual borrow history, and calculating or managing late return fees.

4. Generate different test case for every scenario of Library system.

The test cases for the Library System would include checking if books can be added with all required information, ensuring search functionality works correctly for various inputs, verifying book issuance reflects in the system, confirming that book return updates availability, validating that overdue returns trigger correct late fees, and testing access to borrowing history.

5. List the various categories of users in the Hospital System.

The various categories of users in the Hospital System are patients, doctors, nurses, receptionists, lab technicians, pharmacists, and system administrators, each having distinct roles and access rights within the application.

6. List the various categories of users in the Library System.

   The user categories in the Library System include students, librarians, faculty members, administrators, and guests or visitors, each interacting with the system differently based on their needs and privileges.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In Lab

1. Consider Hospital management system study system specifications and generate report for the various bugs. Introspect the causes for its failure and write down the possible reasons for its failure.

Objective

Bug Report and Failure Analysis –

Hospital Management System (HMS)

Objective

To study the specifications of a Hospital Management System (HMS), identify bugs, analyze causes of failure, and document reasons for potential or actual system failure.

System Context – Hospital Management System (HMS)

Key Functional Modules:

- Patient Registration and Profiles
- Appointment Scheduling
- Doctor Management
- Prescription and Medical Records
- Billing and Payment
- Lab Test Management
- Pharmacy Module
- Login and Role-Based Access (Admin, Doctor, Nurse, Receptionist)

Sample Bug Report

Bug ID	Module	Description	Severity	Type
H001	Login System	Allows login with blank username or password	High	Input Validation
H002	Appointment	System allows overlapping appointments for same doctor	High	Business Logic

Bug ID	Module	Description	Severity	Type
H003	Patient Module	Accepts invalid date of birth (e.g., future date)	Medium	Input Validation
H004	Billing	Incorrect calculation when multiple services are added	High	Calculation Bug
H005	Pharmacy	Allows negative stock values for medicines	High	Functional
H006	Lab Reports	Generates reports for unapproved tests	Medium	Logic Error
H007	Prescription	Doctor can edit prescriptions of other doctors	High	Role-Based Access
H008	Appointment	No notification for appointment cancellation	Low	Usability
H009	UI/UX	Confusing placement of 'Save' and 'Delete' buttons	Low	Usability
H010	Security	Password field allows copy-paste	Low	Security Flaw
H011	Medical Records	Ex-patient records can be modified without admin approval	High	Access Control

---

Introspection: Causes for Failure

- **Weak Validation Logic:** System accepts invalid data (e.g., blank fields, future dates).
- **Incomplete Business Logic:** No checks for appointment clashes or medication stock validation.
- **Lack of Role-Based Access Control:** Unauthorized access to critical features (e.g., prescription editing).

- **No Real-Time Notifications:** Delays in informing users about appointment updates or lab results.
- **Misinterpreted Requirements:** Logic for billing, appointments, or lab test approvals may not match real-world workflows.
- **Inadequate Testing:** Boundary and exception scenarios not tested, such as overlapping appointments or negative values.
- **Poor UI Design:** Confusing interfaces lead to user errors, especially in critical modules like prescriptions and billing.

Possible Reasons for System Failure

- **Ambiguous or Incomplete Requirements:** Key hospital workflows may not be well-defined or understood during development.
- **Lack of End-User Feedback:** No involvement of doctors, nurses, or administrative staff during the UAT phase.
- **Manual Testing Only:** Absence of automation results in bugs recurring in new versions.
- **No Boundary Testing:** Important edge cases like max appointments per day or minimum billing amounts are missed.
- **Weak Exception Handling:** System crashes or behaves unexpectedly with invalid inputs.
- **Outdated UI/UX:** Difficult navigation and lack of confirmations can cause data loss or entry errors.
- **Security Flaws:** Lack of secure input handling and weak session control can compromise patient data.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Data and Results:

Test Case ID	Module	Test Input	Expected Result	Actual Result	Status
TC_H001	**Login System**	**Username: "", Password: ""**	**Show error message**	**Login proceeds without validation**	**Fail**
TC_H002	**Appointment**	**Book same doctor for 2 patients at same time**	**Show slot already booked**	**Appointment allowed**	**Fail**
TC_H004	**Billing**	**Add charges: ₹2000 + ₹1500 + ₹500**	**Total: ₹4000**	**Total shown as ₹3500**	**Fail**
TC_H007	**Prescription**	**Doctor A edits Doctor B's patient prescription**	**Show "Unauthorized Action" error**	**Edit allowed**	**Fail**

Analysis and Inferences:

The system successfully registers patients with unique IDs and valid input.

Input validation prevents empty or null entries, ensuring data integrity.

Duplicate entries are blocked using a HashSet, avoiding redundancy.

Exception handling improves user experience by showing clear error messages.

The design is simple, scalable, and can be extended with file or DB storage.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Post Lab:

Consider Library management system study system specifications and generate report for the various bugs. Introspect the causes for its failure and write down the possible reasons for its failure.

Program:

BugReport

Bug ID	Module	Description	Severity	Type
B001	Login System	System allows login submission with empty username or password	High	Input Validation
B002	Book Catalog	Duplicate ISBN entries allowed without error	High	Functional
B003	Issue/Return	Issues book even when all copies are already issued	High	Business Logic
B004	Fine Calculation	Fine miscalculated (e.g., wrong number of late days)	Medium	Calculation Bug
B005	Search Functionality	Incomplete results for author or title search	Medium	Functional
B006	Member Module	Accepts invalid/future dates for Date of Birth	Low	Validation
B007	UI/UX	"Submit" and "Cancel" buttons look identical, causing confusion	Low	Usability
B008	Security	Allows copy-paste of password field, compromising security	Low	Security
B009	Reports	Overdue report shows books that were already returned	Medium	Data Integrity
B010	Book Return	No confirmation prompt before removing book from profile	High	Usability/Logic

Causes for Failure

Issue Area	Cause
Input Validation	Improper checks for required fields (e.g., empty login, future DOBs)
Business Logic	Fails to verify inventory before book issue
Error Handling	No safeguards for null/invalid/incorrect inputs

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Issue Area	Cause
Requirement Understanding	Fine logic and overdue logic not implemented as per rules
Testing Practices	No detailed boundary or edge-case testing performed
UI Design	UI elements are confusing or inconsistent
Role-Based Access	No clear access control for different user roles

Possible Reasons for System Failure

Category	Failure Reason
Ambiguous Requirements	Misunderstood rules (e.g., fine computation, return handling)
Incomplete User Testing	No input from librarians or students during UAT
Manual Testing Only	Lack of automation led to bugs reappearing in future builds
Missing Boundary Testing	Future dates, issue limits, empty inputs not handled
No Exception Handling	Crashes or incorrect behavior when encountering unexpected input
Poor UI/UX Design	User interface causes confusion, leading to incorrect actions
Weak Security Practices	No restrictions on password field interactions or session control mechanisms

Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Lab Session 04: **Identify the different scenarios based on user interaction and functionality with UI. Write specific test cases for each scenario, detailing input values, expected results and actions. (Gmail, ERP, and Outlook mail, etc.)**

Date of the Session: ____/____/_____ **Time of the Session: __to_____**

Title of the Program: Identify the test cases for user interface. Prepare formal documentation

Pre Lab-Task:

Answer the following question before entering lab. The following prelab task has to perform at home.

1. What is the importance of UI testing

UI testing ensures that the user interface of an application works correctly and meets user expectations. It verifies layout, navigation, responsiveness, and that all visual elements function properly across devices and platforms.

2. What is the scope of UI Testing?

UI testing covers everything the user interacts with—buttons, forms, menus, inputs, links, page layout, responsiveness, and user workflows. It ensures usability, accessibility, consistency, and a seamless user experience.

3. What are the most common issues seen in UI testing?

Misaligned elements
Broken or unresponsive buttons
Inconsistent fonts or styles
Layout issues on different screen sizes
Overlapping content or missing components
Slow or failed page loads

4. List out a few scenarios for UI testing a web application?

Verify login button works and navigates correctly
Check if error messages display for invalid input
Ensure all links on the homepage are functional
Test layout consistency across different screen resolutions
Validate dropdowns, checkboxes, and form submissions

5. List out differences between Desktop application testing and Web application testing.

Desktop apps run on a specific operating system, while web apps run in browsers and are platform-independent. Desktop apps run on a specific operating system, while web apps run in browsers and are platform-independent. Desktop apps need manual updates; web apps can be updated centrally on the server.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In Lab

1. **Consider any software application identify UI test cases Generate the test report with respect of User Interface in given sample format.**

Application Name	Test Case ID	Test Scenario	Test Case	Expected Result	Actual Result	Status	Test Date
Gmail	TC_UI_001	Login Page Validation	Verify login with valid credentials	User should be logged in successfully	User logged in successfully	Pass	2024-09-14
Facebook	TC_UI_002	Login Page Validation	Verify login with invalid credentials	Error message should be	Error message displayed	Pass	2024-09-14
Amazon	TC_UI_003	Navigation Testing	Verify navigation to the Home	Home page should be displayed	Home page displayed	Pass	2024-09-14
Flipkart	TC_UI_004	Form Validation	Verify the “Add to Cart” button	Form should be submitted successfully	Form submitted successfully	Pass	2024-09-14

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Analysis and Inferences:

The tested UI components are functioning correctly with proper validation and navigation. This suggests the application ensures a good user experience and handles basic user interactions effectively.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Post Lab:

Identify different user interaction scenarios for Gmail, ERP, and Outlook Mail, and write specific test cases for each scenario, detailing input values, expected results, and actions.

Gmail:

Test Case ID	Scenario	Action/Input	Expected Result
GM_TC_001	Login with valid credentials	Email: test@gmail.com, Password: correct123	Login successful, inbox loads
GM_TC_002	Login with invalid password	Email: test@gmail.com, Password: wrong123	Error: "Wrong password. Try again."
GM_TC_003	Compose & send email	To: xyz@gmail.com, Subject: Test, Body: Hi	Email sent confirmation shown
GM_TC_004	Search in inbox	Input: invoice January	Email list filtered with relevant results
GM_TC_005	Logout	Click on profile > Sign out	Redirected to login page

ERP:

Test Case ID	Scenario	Action/Input	Expected Result
ERP_TC_001	Student login	ID: stu2025, Password: pass123	Dashboard loads with student info
ERP_TC_002	View attendance	Click on Attendance Module	Monthly attendance percentage displayed
ERP_TC_003	Fee payment	Click on Fees > Pay Now > Enter UPI	Payment success message and receipt generated
ERP_TC_004	Download	Click on	PDF

Test Case ID	Scenario	Action/Input	Expected Result
	semester marksheet	Results > Semester 3	marksheet downloads
ERP_TC_005	Submit course registration form	Fill all required fields > Click Submit	Confirmation message: "Registration successful"

Outlook Mail:

Test Case ID	Scenario	Action/Input	Expected Result
OL_TC_001	Sign in with valid credentials	Email: user@outlook.com, Password: abcd@123	Inbox opens
OL_TC_002	Add email signature	Settings > Mail > Compose and Reply > Signature text	Signature auto-attaches in new emails
OL_TC_003	Schedule a meeting	Calendar > New Event > Set time & participants	Event added to calendar & invite sent
OL_TC_004	Mark email as important	Open mail > Click on 'Important' star	Email marked and moved to Focused tab (if configured)
OL_TC_005	Use mail search	Input: meeting notes April	Results matching query displayed

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Lab Session05: **Write a java classes and methods to test simple Calculator using JUNIT.**

Date of the Session: ____/____/_____ **Time of the Session: __to_____**

Title of the Program: Implement a java classes and methods to test simple Calculator using JUNIT.

Pre Lab-Task:

Answer the following question before entering lab. The following prelab task has to perform at home.

1. What is JUnit, and why is it used in Java?
JUnit is a Java-based testing framework used for writing and running unit tests. It helps developers verify that individual parts of their code work correctly, promoting early bug detection and maintaining code quality.

2. Can you explain the difference between assertEquals and assertThrows methods used in the test cases?
assertEquals checks if two values are equal.
assertThrows verifies that a specific exception is thrown during execution.

3. Why is it important to handle edge cases such as division by zero in your code?
Handling edge cases like division by zero prevents unexpected crashes and ensures the application behaves safely under all input conditions, improving reliability and user trust.

4. How would you modify the test cases if you wanted to add a method for calculating the power of a number in the Calculator class?

You would first implement a power method, then write a new test case using assertEquals to verify correct outputs for various inputs, including 0 and negative values.

5. What is the importance of using assertions in JUnit test cases?

Assertions confirm that the code behaves as expected. They help automatically detect errors, enforce correctness, and serve as documentation for the intended behavior of methods.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In Lab

Design, develop, code, and run the program in HTML to implement the Simple Calculator program. Analyse it from the perspective of equivalence class value testing, derive different test cases, execute these test cases and discuss the test results.

Procedure/Program:

Calculator.java

```
package org.example;

public class Calculator {

    public static int add(int a, int b) {
        return a + b;
    }

    public static int subtract(int a, int b) {
        return a - b;
    }

    public static int multiply(int a, int b) {
        return a * b;
    }

    public static double divide(int a, int b) {
        if (b == 0) throw new ArithmeticException("Cannot divide by zero");
        return (double) a / b;
    }

    public static void main(String[] args) {
        System.out.println("Addition: " + add(7, 3));
        System.out.println("Subtraction: " + subtract(7, 3));
        System.out.println("Multiplication: " + multiply(7, 3));
        System.out.println("Division: " + divide(10, 5));
    }
}
```

CalculatorTest.java

```
package org.example;


import org.testng.annotations.Test;
```

```
import static junit.framework.Assert.assertEquals;
import static org.testng.Assert.assertThrows;


public class CalculatorTest {

    @org.testng.annotations.Test
    public void testAddition() {
        assertEquals(10, Calculator.add(7, 3));
    }

    @org.testng.annotations.Test
    public void testSubtraction() {
        assertEquals(4, Calculator.subtract(7, 3));
    }

    @org.testng.annotations.Test
    public void testMultiplication() {
        assertEquals(21, Calculator.multiply(7, 3));
    }

    @org.testng.annotations.Test
    public void testDivision() {
        assertEquals(2.0, Calculator.divide(10, 5));
    }

    @org.testng.annotations.Test
    public void testDivideByZero() {
        assertThrows(ArithmeticException.class, () -> Calculator.divide(10, 0));
    }
}
```

CalculatorWebTest

```
package org.example;

import io.github.bonigarcia.wdm.WebDriverManager;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.edge.EdgeDriver;
import org.testng.annotations.Test;

public class CalculatorWebTest {
```

```
    @Test
    public void testnumbers() throws InterruptedException {
        WebDriver driver;
        WebDriverManager.edgedriver().setup();
        driver = new EdgeDriver();

        driver.get("C:\\Users\\Pushyami
Krishna\\OneDrive\\Desktop\\svv\\svv\\src\\Web\\Calculator.html");

        WebElement textbox = driver.findElement(By.id("display"));
        textbox.sendKeys("6+5");
        Thread.sleep(3000); // Wait for 3 seconds
        driver.findElement(By.id("equal")).click();
        driver.close();
    }
}
```

Calculator.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Simple Calculator</title>
    <style>
        body {
            text-align: center;
            padding: 50px;
        }
        .calculator {
            display: inline-block;
            border: 1px solid #b84d4d;
            padding: 20px;
        }
        .display {
            width: 100%;
            height: 50px;
            font-size: 2em;
            text-align: right;
            margin-bottom: 20px;
        }
        .button {
            width: 50px;
            height: 50px;
```

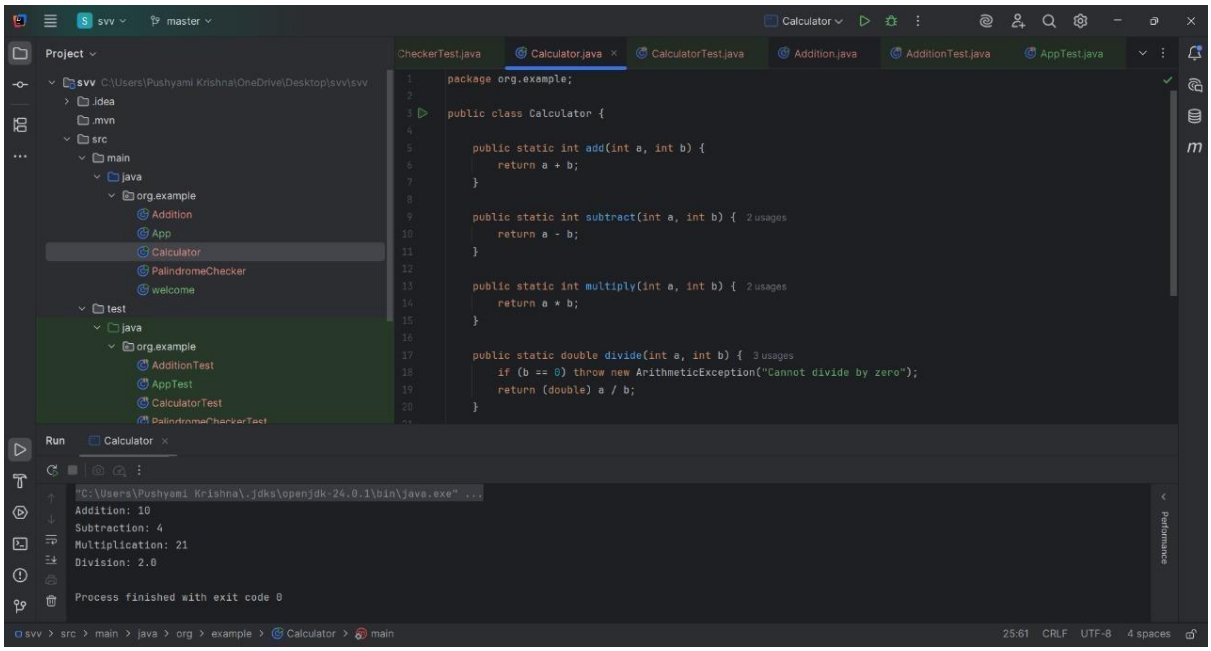
```
            font-size: 1.5em;
            margin: 5px;
        }
    </style>
</head>
<body>
<div class="calculator">
    <input type="text" id="display" class="display" enabled>
    <br>
    <button class="button" onclick="clearDisplay()">C</button>
    <button class="button" onclick="appendDisplay('1')">1</button>
    <button class="button" onclick="appendDisplay('2')">2</button>
    <button class="button" onclick="appendDisplay('3')">3</button>
    <button class="button" onclick="appendDisplay('+')">+</button>
    <br>
    <button class="button" onclick="appendDisplay('4')">4</button>
    <button class="button" onclick="appendDisplay('5')">5</button>
    <button class="button" onclick="appendDisplay('6')">6</button>
    <button class="button" onclick="appendDisplay('-')">-</button>
    <br>
    <button class="button" onclick="appendDisplay('7')">7</button>
    <button class="button" onclick="appendDisplay('8')">8</button>
    <button class="button" onclick="appendDisplay('9')">9</button>
    <button class="button" onclick="appendDisplay('*')">*</button>
    <br>
    <button class="button" onclick="appendDisplay('0')">0</button>
    <button class="button" onclick="appendDisplay('.')">.</button>
    <button class="button" id="equal"  onclick="calculate()">=</button>
    <button class="button" onclick="appendDisplay('/')">/</button>
</div>
<script>
    function appendDisplay(value) {
        document.getElementById('display').value += value;
    }
    function clearDisplay() {
        document.getElementById('display').value = '';
    }
    function calculate() {
        try {
            let result = eval(document.getElementById('display').value);
            document.getElementById('display').value = result;
        } catch (e) {
            document.getElementById('display').value = 'Error';
        }
```

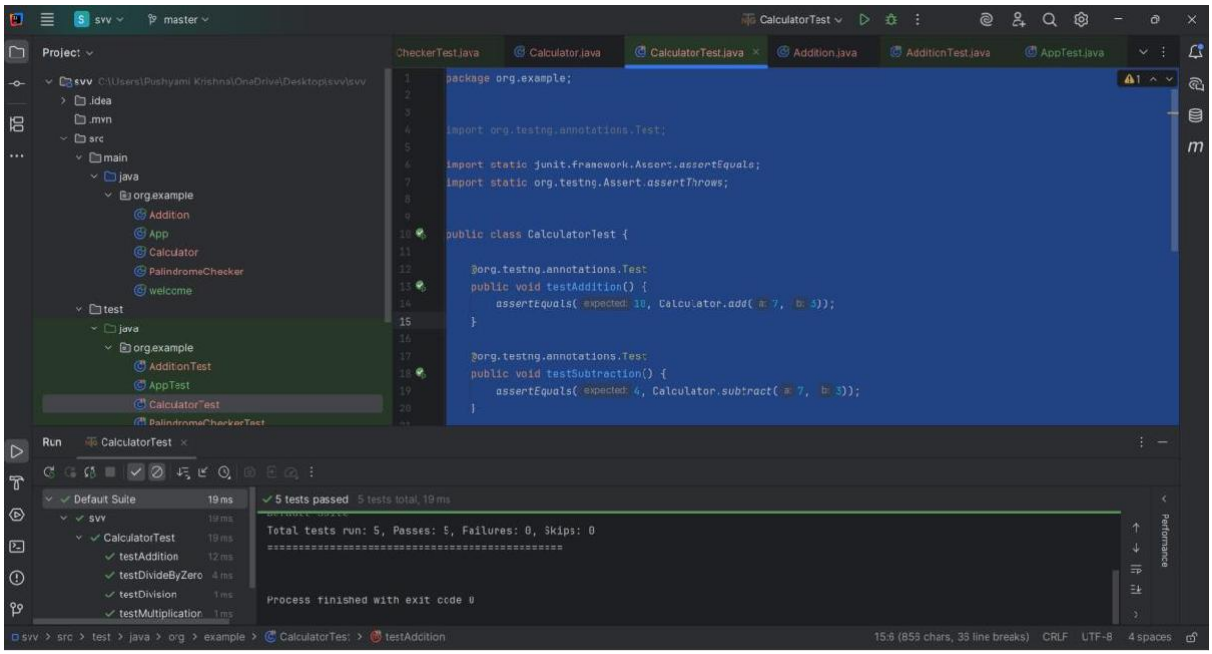
```
    }
</script>
</body>
</html>
```

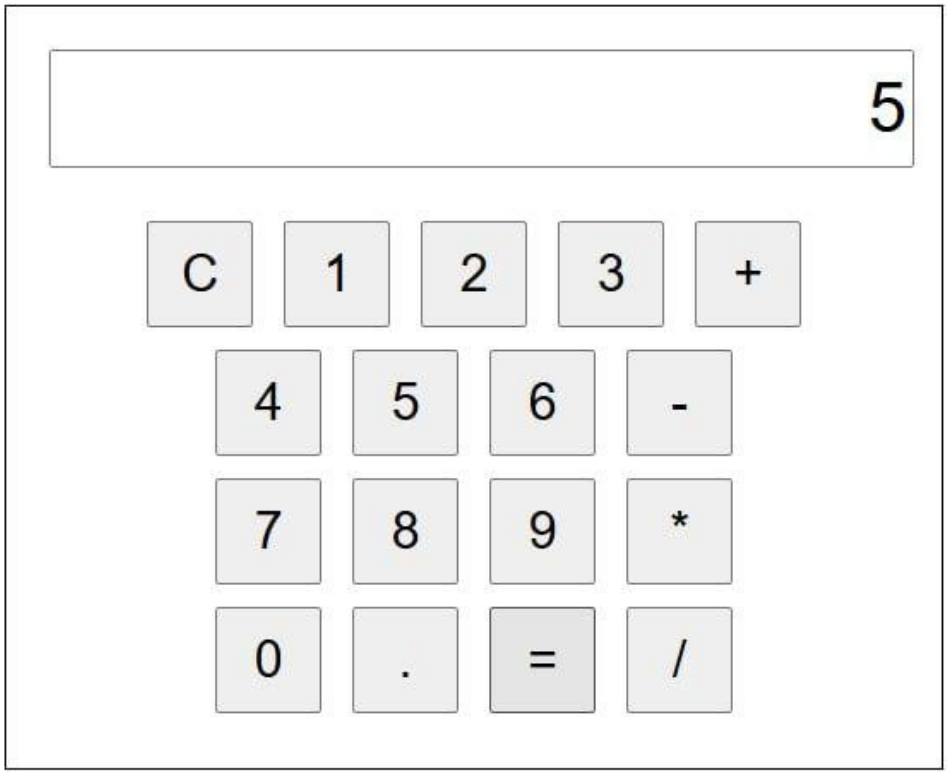
Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Data and Results:

Calculator.java






5
C
1
2
3
+
4
5
6
-
7
8
9
*
0
.
=
/

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Analysis and Inferences:

Test cases use equivalence class testing to check valid inputs and an invalid input.All functions performed as expected, confirming correct results and proper error handling, ensuring the calculator is both accurate and safe.

Post Lab:

How does the use of JUnit for unit testing contribute to software development, particularly in ensuring code quality and facilitating maintenance? Discuss its impact with examples, and mention any limitations that may arise.

Answer:
JUnit is a widely used testing framework in Java that plays a significant role in enhancing the reliability and quality of software. Its primary purpose is to help developers test individual units of code (like methods or classes) to ensure they work as expected. By doing so, JUnit contributes significantly to early bug detection, easier maintenance, and long-term code quality.

Contribution to Software Development

One of the biggest advantages of using JUnit is automated testing. Developers can write test cases that automatically verify the behavior of code whenever changes are made. This reduces the chances of introducing bugs and ensures that any issues are caught early in the development cycle.

Limitations of JUnit

Despite its strengths, JUnit has some limitations. It focuses on **unit-level testing** and doesn't cover other aspects like integration testing, performance testing, or UI testing. Real-world scenarios such as user interactions, database connections, or concurrent access often require additional testing tools beyond JUnit.

Moreover, for small-scale or one-off projects, setting up and maintaining test cases may feel like unnecessary overhead.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Lab Session06: **Introduction of Selenium IDE: Java with selenium Installation, process of recording a test case in IDE environment.**

Date of the Session: ____/____/_____ **Time of the Session: __to_____**

Title of the Program: Selenium IDE Commands

Pre Lab-Task:

Answer the following question before entering lab. The following prelab task has to perform at home.

1. What is Selenium? Who developed Selenium?

Selenium is an open-source automated testing tool used to test web applications across different browsers and platforms.
It was originally developed by Jason Huggins in 2004 while working at ThoughtWorks.
Selenium supports multiple programming languages like Java, Python, C#, and more for writing test scripts.

2. Mention the Pros and Cons of Selenium IDE

Pros of Selenium IDE:

1. Easy to use with a user-friendly interface – ideal for beginners.
2. No programming skills required – supports record and playback.

Cons of Selenium IDE:

1. Limited to testing only web applications in Firefox and Chrome.
2. Not suitable for complex test scenarios or cross-browser testing automation.

3. List out the different Selenium IDE Commands?

Selenium IDE commands (Selenese) include **actions** like click, type, and select, **assertions** like assertText and assertTitle, and **accessors** like storeText and storeValue.
They are used to perform operations, verify results, and store values during test automation.

4. How to Choose the Right Selenium Tool for Your Need

To choose the right Selenium tool, consider your project needs:

- Use **Selenium IDE** for simple, quick tests with record-and-playback.
- Choose **Selenium WebDriver** for complex, cross-browser, and language-specific automation.
- Opt for **Selenium Grid** if you need to run tests in parallel across multiple machines and browsers.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In Lab

Describe the process of recording, saving, and executing a test case in an IDE environment.

Procedure/Program:

1. Recording a Test Case

Open Selenium IDE: Launch the Selenium IDE as a browser extension (available in Chrome and Firefox).
Create a New Project: Click on "Create a new project", give it a name (e.g., "Login Test").
Start Recording: Enter the base URL of your web application and click "Start Recording".
Perform Actions: Selenium IDE will now record every interaction you do on the web page (clicks, inputs, navigation, etc.).

Example: Typing into a form, clicking a button, etc.

Stop Recording: After completing the steps, click "Stop Recording".

2. Saving a Test Case

Name the Test Case: Give your test case a clear, descriptive name Save the Project:
Click on the Save icon or use File > Save Project.
Choose the file location and save the project with a .side extension
This file stores your test cases and suite configurations.

3. Executing a Test Case

Open the Project: Reopen your saved .side project file in Selenium IDE.
Select the Test Case: Choose the test case you want to run from the list.
Click “Play”: Use the “Run current test” button to execute only the selected test.
Alternatively, click “Run all tests” to execute the entire test suite.
View Results:
Selenium IDE displays logs and test results in the bottom panel.
You can see passed/failed status for each command with time details.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Analysis and Inferences:

Selenium IDE is a powerful entry-level automation tool, perfect for quickly recording and testing basic user workflows. However, for more scalable, robust, and customizable testing, it is often used as a stepping stone before moving to WebDriver-based frameworks.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Post Lab:

Write the process of running the test case script by using Start Point in Selenium IDE

1. **Open Selenium IDE:**
   Launch the Selenium IDE extension in your browser (Chrome or Firefox).
   Load your .side project file if it's not already open.
2. **Open the Test Case**
   Click on the test case you want to execute.
   The list of recorded commands (like click, type, assert, etc.) will appear in the editor.
3. **Set a Start Point**
   Scroll to the command from where you want the test execution to begin.
   Right-click on that command.
   Choose "Set Start Point" from the context menu.
   The selected command will be marked as the Start Point.
4. **Run the Test**
   Click the "Run current test" button
   The test will now start execution from the marked Start Point, not from the top.
5. **View Results**
   The execution log will display the result of each command after the Start Point.
   You can observe which steps passed/failed and adjust accordingly.

Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Lab Session07: **Open any URL by using selenium.**

Date of the Session: ____/____/_____ **Time of the Session: __to_____**

Title of the Program: Open any URL by using selenium.

Pre Lab-Task:

Answer the following question before entering lab. The following prelab task has to perform at home.

1. How can you handle exceptions in Selenium scripts, such as when a web element is not found or a page fails to load?
   Exceptions in Selenium can be handled using try-catch blocks to prevent crashes when elements aren't found (NoSuchElementException) or pages fail to load (TimeoutException). Using WebDriverWait helps wait for elements and avoid such errors. This ensures smoother and more reliable test execution.

Exceptions in Selenium can be handled using try-except blocks, catching errors like NoSuchElementException when elements aren't found. Use WebDriverWait with expected_conditions to wait for elements, reducing TimeoutException. This ensures more stable and reliable script execution.

2. How do you set up Selenium WebDriver for a specific browser, such as Chrome, and what are the key steps involved?
Install the ChromeDriver executable matching your Chrome version and add it to your system path. Import Selenium's webdriver and initialize with webdriver.Chrome(). Use the driver to control the browser and close it with driver.quit() when done.

3. Explain the purpose of the driver.get() method in Selenium. What does it do, and how is it used in the context of opening a URL?

The driver.get() method in Selenium opens a specified URL in the browser controlled by the WebDriver. It loads the web page and waits until the page is fully loaded before proceeding. This is used to navigate to any web page during automation scripts.

3. What is the significance of using driver.quit() in a Selenium script, and how does it differ from driver.close()?

   driver.quit() closes all browser windows and ends the WebDriver session, freeing up resources. In contrast, driver.close() only closes the current browser window but keeps the session running if others are open. Using quit() ensures a complete cleanup after the test.

In Lab

Write a Selenium script in Java to automate the following tasks:

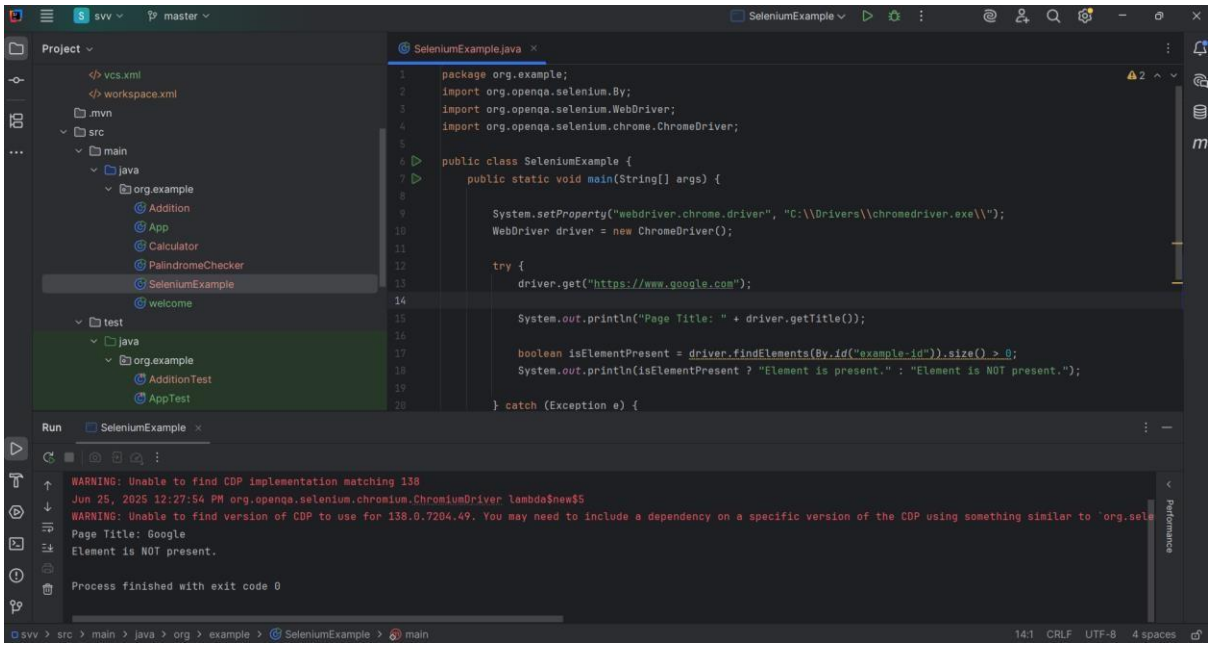
1. **Open the Chrome browser.**
2. **Navigate to https://www.example.com.**
3. **Retrieve and print the page title.**
4. **Check if a specific element (e.g., an element with the ID "example-id") is present on the page.**
5. **Close the browser.**

Procedure/Program:

```
package org.example;
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class SeleniumExample {
    public static void main(String[] args) {

        System.setProperty("webdriver.chrome.driver", "C:\\Drivers\\chromedriver.exe\\");
        WebDriver driver = new ChromeDriver();

        try {
            driver.get("https://www.google.com");

            System.out.println("Page Title: " + driver.getTitle());

            boolean isElementPresent = driver.findElements(By.id("example-id")).size() > 0;
            System.out.println(isElementPresent ? "Element is present." : "Element is NOT
present.");

        } catch (Exception e) {
            e.printStackTrace();
        } finally {

            driver.quit();
        }
    }
}
```

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Data and Results:



Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Analysis and Inferences:

This Selenium script launches Chrome, navigates to a specified URL, and prints the page title. It safely checks if an element with a given ID exists using findElements(). The use of try-catch-finally ensures proper error handling and browser closure. The driver path is hardcoded, which can be improved. Overall, it's a good basic structure for simple automation tasks.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Post Lab:

In the context of Selenium testing, what are some best practices for writing maintainable and scalable test scripts? Discuss strategies for organizing test code, handling test data, and managing browser sessions to ensure your Selenium tests remain effective and easy to maintain.

1. Organizing Test Code

Use the Page Object Model (POM):
Keep the structure and behavior of each web page in separate classes. This improves readability and makes it easier to update locators or actions if the UI changes.

Maintain a Clear Project Structure:
Organize files into folders such as tests, pages, utilities, and data so that your project is easy to navigate and scale.

Create Reusable Utility Methods:
Common actions like login, logout, waiting for elements, etc., should be placed in utility classes instead of repeating them in every test.

Give Meaningful Names to Tests:
Test names should clearly describe what the test is verifying. This helps during debugging and reporting.

2. Handling Test Data

Externalize Test Data:
Store data in files like Excel, CSV, JSON, or databases rather than hardcoding values in test scripts. This separates data from logic and makes updates easier.

Use Data-Driven Testing:
Run the same test with multiple data sets to cover more scenarios without duplicating test logic.

Keep Sensitive Data Secure:
Avoid storing passwords or tokens in plain text. Use environment variables or encrypted storage when necessary.

3. Managing Browser Sessions

Use Setup and Teardown Methods:
Always open and close the browser cleanly for each test to avoid session conflicts or memory issues.

Centralize WebDriver Management:

Manage browser setup in one place so you can easily switch between Chrome, Firefox, etc., without changing every test.

Use Headless Browsers for CI/CD:
Run tests in headless mode (without opening a browser window) for faster execution in pipelines.
Avoid Hardcoding Paths and URLs:
Keep such configurations in a separate file or properties file, making tests more flexible and environment-independent.

Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Lab Session08: **Locate any web element in any web page by using Locator.**

Date of the Session: ____/____/_____ **Time of the Session: __to_____**

Title of the Program: Locate any web element in any web page by using Locator (Selenium).
Pre Lab-Task:

Answer the following question before entering lab. The following prelab task has to perform at home.

1. Compare and contrast IntelliJ and Eclipse IDE.

IntelliJ IDEA offers a modern UI, smart code completion, and better out-of-the-box features, especially for Java and web development.
Eclipse is open-source, highly customizable, and widely used in enterprise environments with strong plugin support.
While IntelliJ is more intuitive and user-friendly, Eclipse is preferred for heavy modular projects and plugin-driven workflows.

2. How to inspect various elements in any commercial site

Right-click on any element in a commercial website and select **"Inspect"** to open browser DevTools.
The **Elements panel** will highlight the HTML structure, allowing you to view tags, IDs, classes, and attributes.
You can use the **inspect tool** to click on elements directly and copy their XPath or CSS selector for automation.

3. Write different set properties for various drivers.

Use System.setProperty to specify the path for browser drivers in Selenium.
For example: Chrome → webdriver.chrome.driver, Firefox → webdriver.gecko.driver, Edge → webdriver.edge.driver.
Each property points to the location of the corresponding .exe driver file on your system.

4. Mention different ways of writing set Properties in WebDriver

You can set properties using System.setProperty() or configure them via environment variables or build tools like Maven/Gradle.
Example: System.setProperty("webdriver.chrome.driver", "path") or use WebDriverManager to auto-manage drivers without manual paths.

In Lab

Google Company wants to test its account URL working perfectly or not. It instructs the test automation engineer to inspect a site by using the locator: link and partial link text by writing suitable Automated Test cases.

Hint: https://google.com/login

Procedure/Program:

```
package org.example;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class GoogleLoginTestLinkText {

    public static void main(String[] args) {

        // Set path to your chromedriver
        System.setProperty("webdriver.chrome.driver", "C:\\Drivers\\chromedriver.exe\\"); WebDriver
        driver = new ChromeDriver();

        try {
            // Navigate to Google's homepage
            driver.get("https://www.google.com");

            // Maximize browser
            driver.manage().window().maximize();

            // Click on the "Sign in" link using linkText
            driver.findElement(By.linkText("Sign in")).click();

            // Wait for a few seconds to observe (or use WebDriverWait)
            Thread.sleep(3000); // not recommended in real tests

            // Verify we navigated to the login page (optional check)
            if (driver.getCurrentUrl().contains("accounts.google.com")) { System.out.println("Test
                Passed: Navigated to Login page.");
            } else {
                System.out.println("Test Failed: Login page not loaded.");
            }

        } catch (Exception e) {
            e.printStackTrace();
        } finally {
```

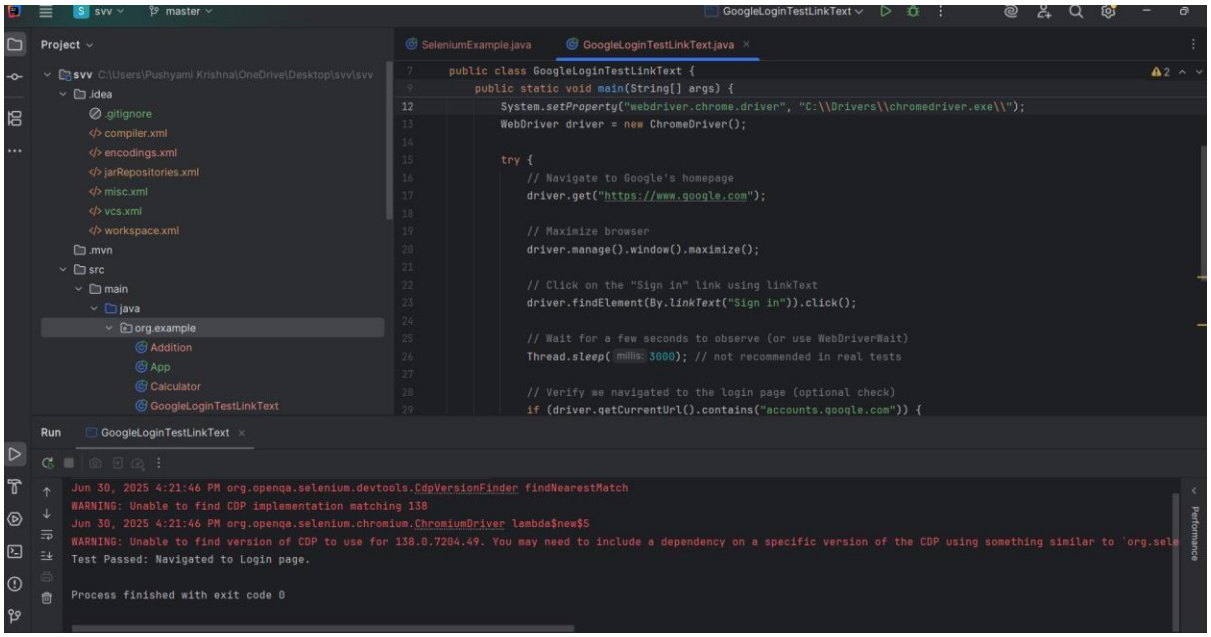
```
            driver.quit();
        }
    }
}
```

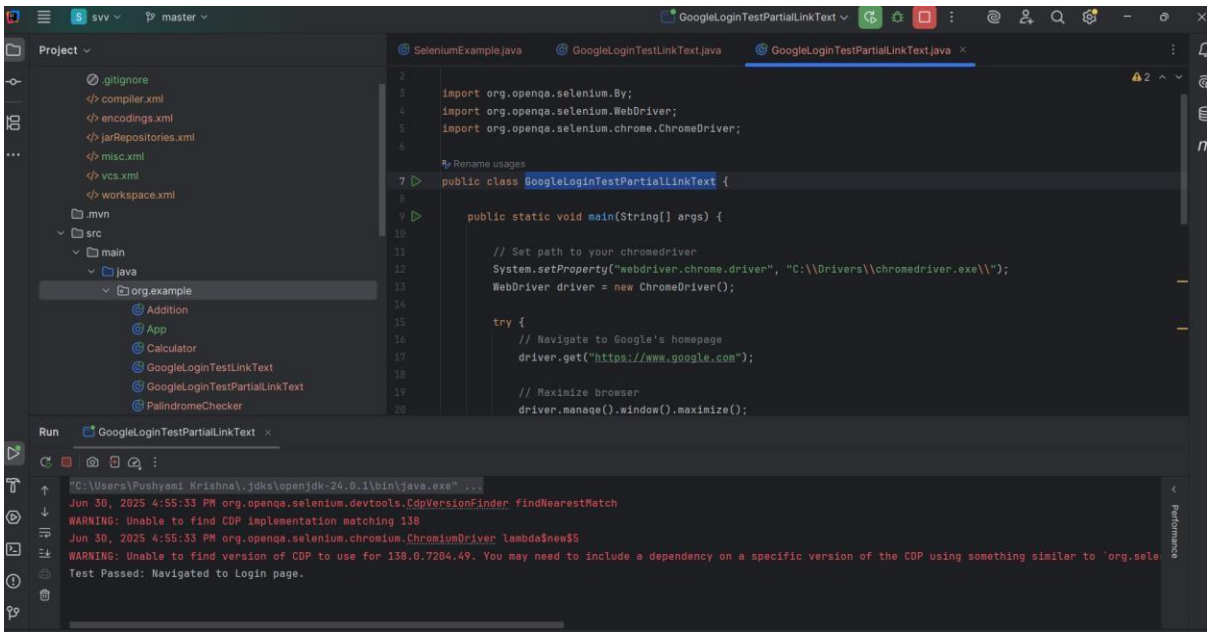
Partial

```
package org.example;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver; public

class GoogleLoginTestPartialLinkText {

    public static void main(String[] args) {

        // Set path to your chromedriver
        System.setProperty("webdriver.chrome.driver", "C:\\Drivers\\chromedriver.exe\\"); WebDriver
        driver = new ChromeDriver();

        try {
            // Navigate to Google's homepage
            driver.get("https://www.google.com");

            // Maximize browser
            driver.manage().window().maximize();

            // Click on the "Sign in" link using linkText
            driver.findElement(By.partialLinkText("Sign in")).click();

            // Wait for a few seconds to observe (or use WebDriverWait)
            Thread.sleep(3000); // not recommended in real tests

            // Verify we navigated to the login page (optional check)
            if (driver.getCurrentUrl().contains("accounts.google.com")) { System.out.println("Test
                Passed: Navigated to Login page.");
            } else {
                System.out.println("Test Failed: Login page not loaded.");
            }

        } catch (Exception e) {
            e.printStackTrace();
        } finally { driver.quit();
        }
    }
}
```

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Data and Results:





Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Analysis and Inferences:

This Selenium script automates the process of opening Google's homepage and clicking the "Sign in" link using linkText.
It checks if the navigation redirects to Google's login page by verifying the URL.
The use of Thread.sleep() is not ideal for real tests; WebDriverWait is preferred for reliability.
Overall, the test confirms basic navigation functionality and demonstrates link-based element interaction.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Post Lab:

Describe the concept and syntax of all the following locators: ID, class, XPATH, link text, partial link text, tag and CSS.

1. ID – Locates an element using its unique id attribute.
   driver.findElement(By.id("username"));
2. Class Name – Finds element(s) by the class attribute.
   driver.findElement(By.className("login-btn"));
3. XPath – Navigates the DOM tree to find elements using path expressions.
   driver.findElement(By.xpath("//input[@name='email']"));
4. Link Text – Finds anchor (<a>) tags by their exact visible text.
   driver.findElement(By.linkText("Forgot Password"));
5. Partial Link Text – Finds anchor tags using a part of the link text.
   driver.findElement(By.partialLinkText("Forgot"));
6. Tag Name – Selects elements by their HTML tag name (e.g., input, div).
   driver.findElement(By.tagName("button"));
7. CSS Selector – Uses CSS syntax to target elements based on ID, class, attributes, etc.
   driver.findElement(By.cssSelector("input[type='password']"));

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Lab Session 09: **Implement Selenium web driver Script: Test ERP/mail/Facebook login functionality with incorrect username and incorrect password**

Date of the Session: ____/____/_____ **Time of the Session: __to_____**

Title of the Program: Test ERP/Facebook/Gmail login functionality with incorrect username and incorrect

Pre Lab-Task:

Answer the following question before entering lab. The following prelab task has to perform at home.

1. Write the Processor for the installation of selenium web driver in your system.

1. Install Java JDK and configure JAVA_HOME environment variable.
2. Download and set up Eclipse IDE, then add Selenium WebDriver JAR files to your project's build path.
3. Download ChromeDriver (or any browser driver), place it in a folder, and set its path using System.setProperty in your Java code.

2. What is Selenium WebDriver and how is it different from Selenium IDE?

Selenium WebDriver is a powerful automation tool that allows you to write test scripts in programming languages like Java, Python, or C# to control web browsers.
Unlike Selenium IDE, which is a simple record-and-playback tool for beginners, WebDriver provides more flexibility, scalability, and real-time browser interaction.
WebDriver supports complex test logic and cross-browser testing, making it ideal for professional test automation.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In Lab

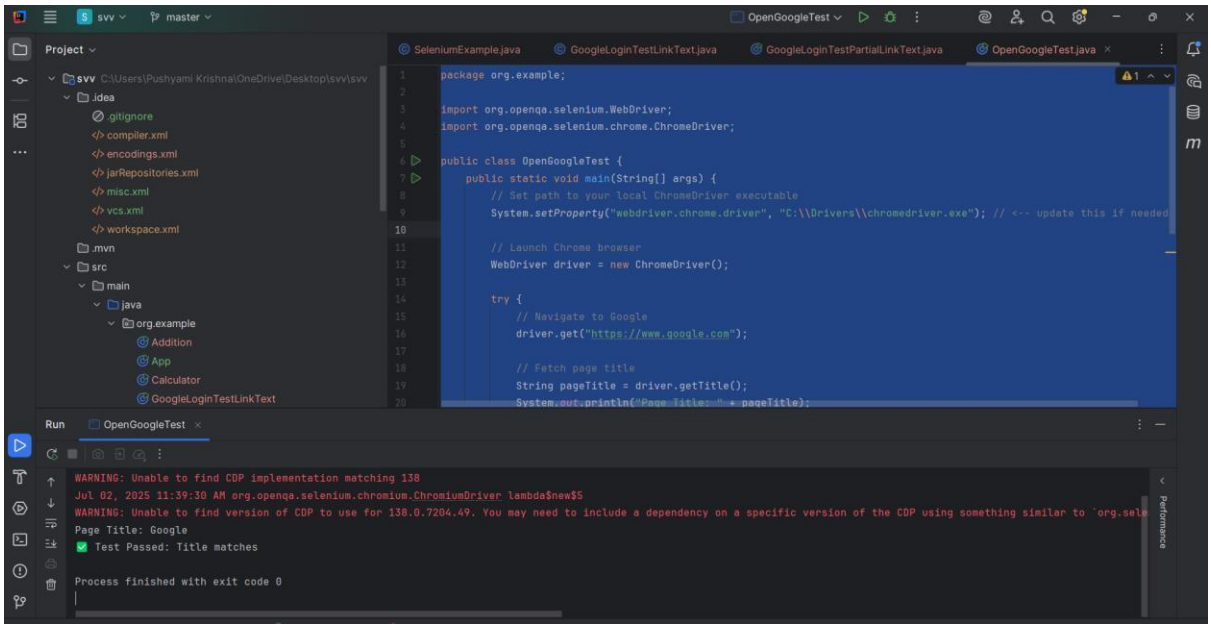
1. **Implement Selenium web driver Script: Java program open web application in browser and write test case.**

Procedure/Program:

```
package org.example;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class OpenGoogleTest {
    public static void main(String[] args) {
        // Set path to your local ChromeDriver executable
        System.setProperty("webdriver.chrome.driver", "C:\\Drivers\\chromedriver.exe"); // <--
update this if needed

        // Launch Chrome browser
        WebDriver driver = new ChromeDriver();

        try {
            // Navigate to Google
            driver.get("https://www.google.com");

            // Fetch page title
            String pageTitle = driver.getTitle();
            System.out.println("Page Title: " + pageTitle);

            // Check if title matches expected
            if (pageTitle.equals("Google")) {
                System.out.println("Test Passed: Title matches");
            } else {
                System.out.println(" Test Failed: Title does not match");
            }

        } catch (Exception e) {
            System.out.println(" Error: " + e.getMessage());
        } finally {
            // Close the browser
            driver.quit();
        }
    }
}
```

Data and Results:

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>



Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Analysis and Inferences:

This script tests whether the browser launches correctly, navigates to a specific URL, and verifies the page title.
It helps confirm that the correct web page is loaded and functioning as expected.
Such basic test cases serve as foundational checks in automated UI testing workflows.

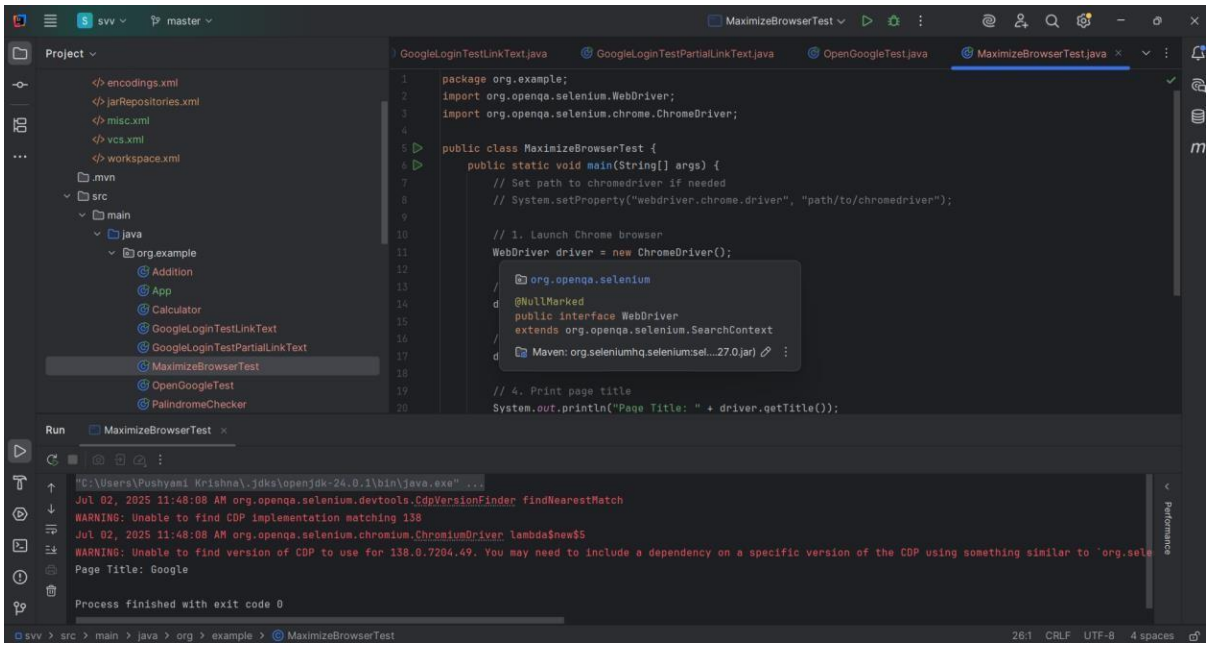
2. **Implement Selenium web driver Script: Maximize browser window**

Procedure/Program:

```
package org.example;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;

public class MaximizeBrowserTest { public
    static void main(String[] args) {
        // Set path to chromedriver if needed
        // System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");

        // 1. Launch Chrome browser
        WebDriver driver = new ChromeDriver();

        // 2. Maximize browser window
        driver.manage().window().maximize();

        // 3. Navigate to a website
        driver.get("https://www.google.com");

        // 4. Print page title
        System.out.println("Page Title: " + driver.getTitle());

        // 5. Close browser
        driver.quit();
    }
}
```

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Data and Results:



Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Analysis and Inferences:

Selenium WebDriver commands provide a structured way to automate browser interactions such as navigation, element handling, waiting, and switching contexts.
By mastering these commands, testers can simulate real user actions to validate web application functionality effectively.
Using the right combination of commands ensures robust, maintainable, and reliable automation scripts across browsers and platforms.

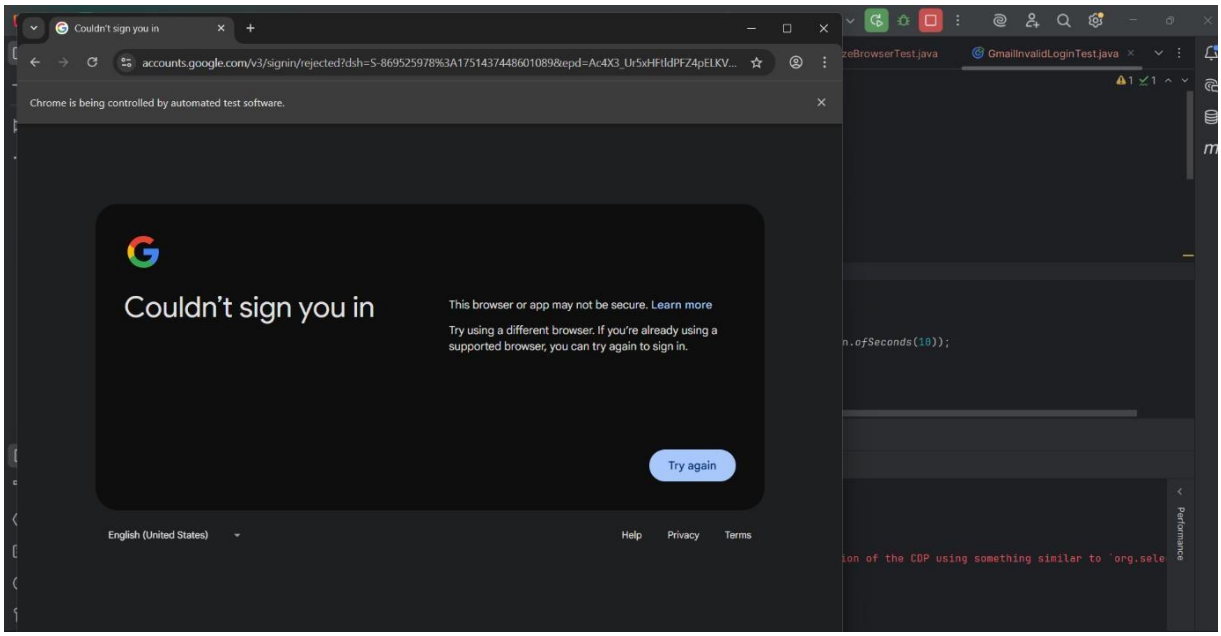
Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Post Lab:

1. **Implement Selenium web driver Script: Test Gmail login functionality with incorrect username and incorrect password.**

```
package org.example;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

import java.time.Duration;

public class GmailInvalidLoginTest {
   public static void main(String[] args) {
      // 1. Set up Chrome browser
      WebDriver driver = new ChromeDriver();
      WebDriverWait wait = new WebDriverWait(driver, Duration.ofSeconds(10));

      try {
         // 2. Go to Gmail login page
         driver.get("https://accounts.google.com/signin");

         // 3. Enter invalid email/username
         WebElement emailField = wait.until(
               ExpectedConditions.visibilityOfElementLocated(By.id("identifierId"))
         );
         emailField.sendKeys("invaliduser123456@gmail.com");

         driver.findElement(By.id("identifierNext")).click();

         // 4. Wait for error or password field
         wait.until(ExpectedConditions.or(

ExpectedConditions.visibilityOfElementLocated(By.xpath("//div[@class='o6cuMc']")),
               ExpectedConditions.visibilityOfElementLocated(By.name("Passwd"))
         ));

         // Check if error is displayed
         boolean errorAfterEmail =
driver.findElements(By.xpath("//div[@class='o6cuMc']")).size() > 0;
```

```
        if (errorAfterEmail) {
            WebElement errorElement =
driver.findElement(By.xpath("//div[@class='o6cuMc']"));
            System.out.println("✓ Test Passed: Invalid username error → " +
errorElement.getText());
        } else {
            // 5. Enter invalid password if email is accepted
            WebElement pwdField = wait.until(
                  ExpectedConditions.visibilityOfElementLocated(By.name("Passwd"))
            );
            pwdField.sendKeys("wrongpassword123");
            driver.findElement(By.id("passwordNext")).click();

            // 6. Wait for password error
            WebElement pwdError = wait.until(

ExpectedConditions.visibilityOfElementLocated(By.xpath("//div[@class='o6cuMc']"))
            );

            System.out.println("✓ Test Passed: Invalid password error → " +
pwdError.getText());
        }

      } catch (Exception e) {
         System.out.println("✗ Test Failed: " + e.getMessage());
      } finally {
         // 7. Close the browser
         driver.quit();
      }
   }
}
```



2. **List out the various selenium web driver commands**

1. Browser Commands
2. Navigation Commands
3. WebElement Commands
4. Wait Commands
5. Window & Frame Commands
6. Alert Commands

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Lab Session10: **Introduction to TestNG tool.**

Date of the Session: ____/____/_____ **Time of the Session: __to_____**

Title of the Program: Implement and test a program to login a specific webpage.

Pre Lab-Task:

Answer the following question before entering lab. The following prelab task has to perform at home.

1. Write Step by step procedure how to add TestNG plug in to Eclipse IDE.

1. Open Eclipse IDE

Launch your Eclipse (any version that supports plugins – ideally Eclipse IDE for Java Developers).

2. Go to Eclipse Marketplace

Click on Help in the top menu → Select Eclipse Marketplace…

3. Search for TestNG

In the search bar, type TestNG and press Enter.

4. Install TestNG

- From the list, find "TestNG for Eclipse".
- Click Install next to it.
- If prompted, select the required packages and click Confirm.

5. Accept License

- Review the license agreement.
- Accept the terms and click Finish.

6. Restart Eclipse

After installation, Eclipse will prompt you to restart. Click Yes to apply changes.

7. Verify TestNG Installation

- Right-click on any project → Build Path → Add Libraries…
- If TestNG is listed, the installation was successful.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

In Lab

1. **Implement TestNG Script: Write Test cases for Web application Title and URL.**

Procedure/Program:

UnderTest: WikipediaHomeTests.java

```
package org.example;

import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.*;

public class WikipediaHomeTests {
    private WebDriver driver;

    @BeforeClass
    public void setUp() {
        // If chromedriver isn't on your PATH, uncomment and adjust the path below:
        // System.setProperty("webdriver.chrome.driver", "C:/path/to/chromedriver.exe");

        // Launch Chrome
        driver = new ChromeDriver();
        driver.manage().window().maximize();
    }

    @BeforeMethod
    public void navigateToHome() {
        // Navigate to Wikipedia homepage
        driver.get("https://www.wikipedia.org/");
    }

    @Test(priority = 1, description = "Verify Wikipedia homepage title")
    public void testHomePageTitle() {
        String expectedTitle = "Wikipedia";
        String actualTitle  = driver.getTitle();

        Assert.assertTrue(
                actualTitle.contains(expectedTitle),
                "Page title should contain '" + expectedTitle + "'. Actual: " + actualTitle
        );
    }

    @Test(priority = 2, description = "Verify Wikipedia homepage URL")
    public void testHomePageURL() {
        String expectedURL = "https://www.wikipedia.org/";
        String actualURL  = driver.getCurrentUrl();
```

```
        Assert.assertEquals(
                actualURL, expectedURL,
                "Current URL should be exactly '" + expectedURL + "'."
        );
    }

    @AfterMethod
    public void afterMethod() {
        // Optional: clear cookies or reset state if needed
        driver.manage().deleteAllCookies();
    }

    @AfterClass
    public void tearDown() {
        if (driver != null) {
            driver.quit();
        }
    }
}
```

UnderSrc:testing.xml

```
<suite name="WikipediaHomeTestSuite">
    <test name="HomepageTests">
        <classes>
            <class name="org.example.WikipediaHomeTests"/>
        </classes>
    </test>
    <!-- <test name="WebFormTest">
            <classes>
                <class name="org.example.FormTest"/>
            </classes>
        </test> -->
</suite>
```

UnderWeb:loginstep1.html

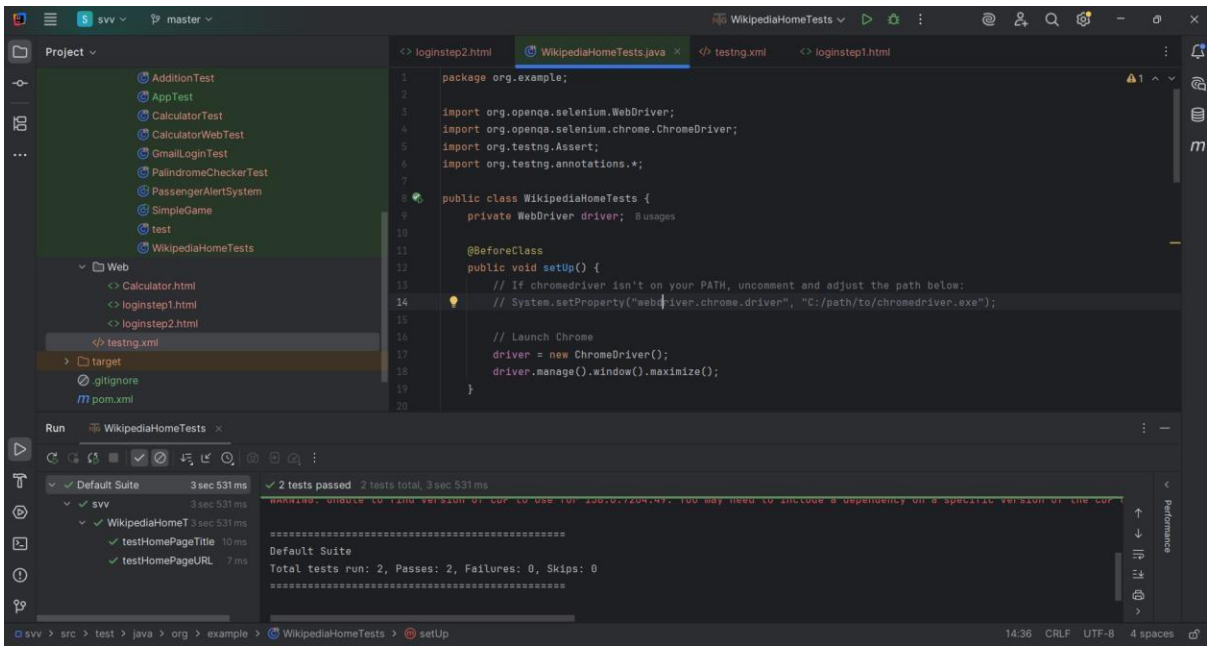
```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Mock Gmail – Step 1</title></head>
<body>
<h3>Gmail Login – Step 1</h3>
<input id="identifierId" placeholder="Email"><button id="identifierNext">Next</button>
<script>
    document.getElementById('identifierNext').onclick = function() {
        window.location = '"C:\\Users\\Pushyami Krishna\\Downloads\\Telegram
Desktop\\loginstep1 (2).html"';
    };
</script>
</body>
</html>
```

UnderWeb:loginstep2.html

```
<!doctype html>
<html>
<head><meta charset="utf-8"><title>Mock Gmail – Step 2</title></head>
<body>
<h3>Gmail Login – Step 2</h3>
<input id="password" type="password" placeholder="Password">
<button id="passwordNext">Sign In</button>
<div id="error" style="display:none;color:red;margin-top:1em">
  Wrong password. Try again.
</div>
<script>
  document.getElementById('passwordNext').onclick = function() {
    document.getElementById('error').style.display = 'block';
  };
</script>
</body>
</html>
```

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Data and Results:



Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Analysis and Inferences:

This script validates the correctness of a web page's title and URL using assertions.
It ensures the user is on the correct page and helps detect redirections or incorrect deployments early.
Such tests are basic yet essential for verifying application integrity during regression.

2. **Implement TestNG script: Test web application form using TestNG annotations.**

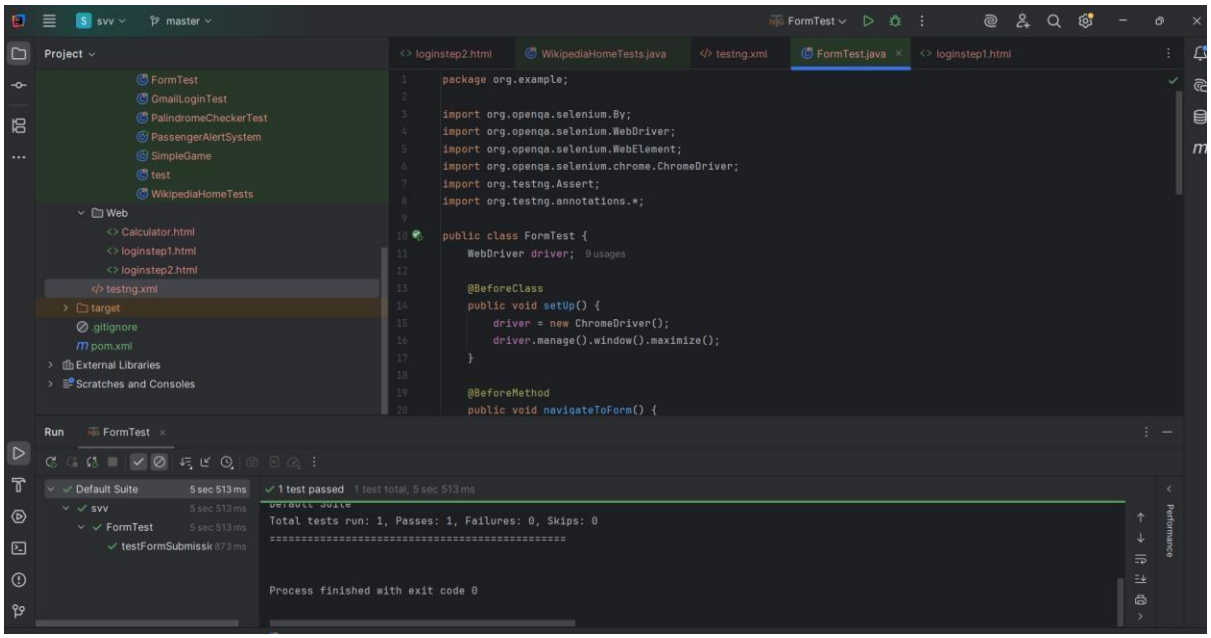
Procedure/Program:

```
package org.example;

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;
import org.testng.Assert;
import org.testng.annotations.*;

public class FormTest {
    WebDriver driver;

    @BeforeClass
    public void setUp() {
        driver = new ChromeDriver();
        driver.manage().window().maximize();
    }

    @BeforeMethod
    public void navigateToForm() {
        driver.get("https://www.selenium.dev/selenium/web/web-form.html");
    }

    @Test
    public void testFormSubmission() {
        // Fill out the text field
        WebElement textBox = driver.findElement(By.name("my-text"));
        textBox.clear();
        textBox.sendKeys("Hello TestNG!");

        // Fill out the password field
        WebElement password = driver.findElement(By.name("my-password"));
        password.clear();
        password.sendKeys("Password123");

        // Submit the form
        WebElement submitButton = driver.findElement(By.cssSelector("button"));
        submitButton.click();

        // Assert confirmation message
        WebElement message = driver.findElement(By.id("message"));
        String confirmation = message.getText();
        Assert.assertTrue(confirmation.contains("Received!"), "Form submission failed!");
    }
```

```
    @AfterMethod
    public void clearCookies() {
        driver.manage().deleteAllCookies();
    }

    @AfterClass
    public void tearDown() {
        driver.quit();
    }
}
```

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Data and Results:



Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Analysis and Inferences:

This TestNG script uses annotations like @BeforeClass, @Test, and @AfterClass to structure the form test.
It automates inputting values and submitting a web form to verify functional behavior.
Such scripts help validate form workflows efficiently and ensure consistent regression testing.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Post Lab:

1. List out Various TestNG annotations.

@BeforeSuite – Runs once before all tests in the suite.

@AfterSuite – Runs once after all tests in the suite.

@BeforeTest – Runs before any test methods in the `<test>` tag of `testng.xml`.

@AfterTest – Runs after all test methods in the `<test>` tag.

@BeforeClass – Runs before the first method in the current class.

@AfterClass – Runs after all methods in the current class.

@BeforeMethod – Runs before each `@Test` method.

@AfterMethod – Runs after each `@Test` method.

@Test – Marks a method as a test case.

@DataProvider – Supplies data to test methods for parameterized tests.

2. List the advantages of TestNG over JUnit

- Parallel test execution is built-in, improving test speed and efficiency.
- Dependency testing using dependsOnMethods or dependsOnGroups.
- Powerful reporting with detailed HTML and XML test reports.
- Supports parameterized tests without external tools via `@DataProvider`.
- Easier integration with tools like Maven, Jenkins, and Selenium.

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>

Evaluator Remark (if Any):	**Marks Secured _____ out of 50**
	Signature of the Evaluator with Date

Evaluator MUST ask Viva-voce prior to signing and posting marks for each experiment

Experiment #	<TO BE FILLED BY STUDENT>	Student ID	<TO BE FILLED BY STUDENT>
Date	<TO BE FILLED BY STUDENT>	Student Name	<TO BE FILLED BY STUDENT>