Software Testing & Quality Assurance Lab

Subject Code: MCAL35

A Practical Journal Submitted in Fulfillment
of the Degree of

MASTER

In

COMPUTER APPLICATION

Year 2024-2025

By

Mr. Jadhav Harshal Sanjay

(Application Id-53942)

Seat No.: 1030199

Semester-III

Under the Guidance of

Asst. Prof. Dnyaneshwar Deore

Centre for Distance and Online Education
Vidya Nagari, Kalina, Santacruz East – 400098.
University of Mumbai

PCP Center [Satish Pradhan Dnyanasadhana College, Thane]

Institute of Distance and Open Learning
Vidya Nagari, Kalina, Santacruz East – 400098.

CERTIFICATE

This to certify that, **"Jadhav Harshal Sanjay"** appearing **Master's in computer application (Semester III) Application Id: 53942** has satisfactory completed the prescribed practical of **MCAL35- Software Testing & Quality Assurance Lab** as laid down by the University of Mumbai for the academic year 2024-25.

Teacher In Charge	External Examiner	Coordinator – M.C.A

Date:

Place: -

INDEX

Practical no. 1 Unit Testing using JUnit (Java)

Aim: Test a simple class and its methods using JUnit framework.

Description: In this experiment, you will create a class with some business logic and then write unit tests to validate its correctness.

Steps:

1. Create a Java class `Calculator` with methods like `add`, `subtract`, `multiply`, and `divide`.
2. Write JUnit test cases to test each method.

Code:

java

```
// Calculator.java
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }
    public int subtract(int a, int b) {
        return a - b;
    }
    public int multiply(int a, int b) {
        return a * b;
    }

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

java

```
// CalculatorTest.java
import org.junit.Test;
import static org.junit.Assert.*;

public class CalculatorTest {

    Calculator calc = new Calculator();

    @Test
    public void testAdd() {
        assertEquals(5, calc.add(2, 3));
    }

    @Test
    public void testSubtract() {
        assertEquals(1, calc.subtract(3, 2));
    }

    @Test
    public void testMultiply() {
        assertEquals(6, calc.multiply(2, 3));
    }

    @Test
    public void testDivide() {
        assertEquals(2.0, calc.divide(4, 2), 0.0);
    }


    @Test(expected = ArithmeticException.class)
    public void testDivideByZero() {
```

```
        calc.divide(1, 0);
    }
}
```

Output:

Output in IDE (JUnit Test Results):

```yaml
JUnit Version: 4.13.2
Time: 0.105
OK (5 tests)
```

Practical No. 2. Boundary Value Testing (BVT)

Aim: Test a method that checks if a given number is within a specific range using boundary value analysis.

Description: This experiment focuses on testing the boundaries of input values.

Steps:

1. Create a method to validate whether a number is within a given range.
2. Write test cases focusing on boundary values (values just inside and just outside the valid range).

Code:

java

```
// RangeValidator.java
public class RangeValidator {
    public boolean isInRange(int value) {
        return value >= 1 && value <= 100;
    }
}
```

java

```
// RangeValidatorTest.java
import org.junit.Test;
import static org.junit.Assert.*;

public class RangeValidatorTest {

    RangeValidator validator = new RangeValidator();
    @Test
    public void testValidRange() {
        assertTrue(validator.isInRange(50));  // Inside range
}
    @Test
```

```
    public void testBoundaryLower() {
        assertTrue(validator.isInRange(1));  // Lower boundary
    }

    @Test
    public void testBoundaryUpper() {
        assertTrue(validator.isInRange(100));  // Upper boundary
    }

    @Test
    public void testBelowLowerBoundary() {
        assertFalse(validator.isInRange(0));  // Below lower boundary
    }

    @Test
    public void testAboveUpperBoundary() {
        assertFalse(validator.isInRange(101));  // Above upper boundary
    }
}
```

Output:

Output in IDE (JUnit Test Results):

```yaml
JUnit Version: 4.13.2
Time: 0.054
OK (5 tests)
```

Practical no. 3. Regression Testing

Aim: Verify that changes to the software do not negatively impact the existing functionality.

Description: This experiment involves testing a set of methods after making changes to the codebase, ensuring that the existing code continues to work as expected.

Steps:

1. Create a basic class and a set of test cases.
2. Modify the class (e.g., add a new method or refactor existing logic).
3. Re-run the tests to ensure that no functionality is broken.

Code:

java

```
// Old Calculator (before modification)
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }
}
```

java

```
// CalculatorTest.java
import org.junit.Test;
import static org.junit.Assert.*;

public class CalculatorTest {

    Calculator calc = new Calculator();



    @Test
```

```
    public void testAdd() {
        assertEquals(5, calc.add(2, 3));
    }
}
```

Modification:

```java
// New Calculator (after modification)
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }
}
java
// New CalculatorTest.java
import org.junit.Test;
import static org.junit.Assert.*;

public class CalculatorTest {

    Calculator calc = new Calculator();

    @Test
    public void testAdd() {
        assertEquals(5, calc.add(2, 3));
    }
    @Test
    public void testSubtract() {
```

```
        assertEquals(1, calc.subtract(3, 2));
    }
}
```

Output:

Output in IDE (JUnit Test Results):

```yaml
JUnit Version: 4.13.2
Time: 0.084
OK (2 tests)
```

Practical no. 4. Test-Driven Development (TDD)

Aim: Write the test cases before implementing the functionality (following the TDD cycle: Red-Green-Refactor).

Description: Implement a simple functionality like checking whether a string is a palindrome.

Steps:

1. Write a test case for a palindrome function.
2. Implement the function to make the test pass.
3. Refactor the code as needed.

Code:

java

```
// PalindromeTest.java (initially fails)
import org.junit.Test;
import static org.junit.Assert.*;

public class PalindromeTest {

    @Test
    public void testIsPalindrome() {
        Palindrome p = new Palindrome();
        assertTrue(p.isPalindrome("madam"));
        assertFalse(p.isPalindrome("hello"));
    }
}
```

java

```
// Palindrome.java (empty)
public class Palindrome {
    public boolean isPalindrome(String str) {
        // Placeholder logic
        return false;
    }
}
```

Fix:

java

```
// Palindrome.java (corrected)
public class Palindrome {
    public boolean isPalindrome(String str) {
        String reversed = new StringBuilder(str).reverse().toString();
        return str.equals(reversed);
    }
}
```

Output:

Output without implementation (JUnit Test Results):

vbnet

```
JUnit Version: 4.13.2
Time: 0.016
FAILURES!!!
Testcase: testIsPalindrome
  Expected :true
  Actual   :false
```

Output after implementation (JUnit Test Results):

```yaml
JUnit Version: 4.13.2
Time: 0.052
OK (2 tests)
```

Practical no. 5. Exception Testing

Aim: Verify that the system handles exceptions correctly.

Description: Write test cases to check for expected exceptions when invalid inputs are provided.

Steps:

1. Create a method that throws an exception for invalid input.
2. Write a test case to ensure that the exception is correctly thrown.

Code:

java

```
// StringProcessor.java
public class StringProcessor {
    public String toUpperCase(String str) {
        if (str == null) throw new IllegalArgumentException("Input string cannot be null");
        return str.toUpperCase();
    }
}
```

java

```
// StringProcessorTest.java
import org.junit.Test;
import static org.junit.Assert.*;

public class StringProcessorTest {

    StringProcessor processor = new StringProcessor();




    @Test
```

```
    public void testToUpperCaseValid() {
        assertEquals("HELLO", processor.toUpperCase("hello"));
    }

    @Test(expected = IllegalArgumentException.class)
    public void testToUpperCaseNull() {
        processor.toUpperCase(null);
    }
}
```

Output:

Output (JUnit Test Results):

```yaml
JUnit Version: 4.13.2
Time: 0.032
OK (2 tests)
```

Practical no. 6. Load Testing (Using JMeter)

Aim: Simulate multiple users accessing the application and measure its response time.

Description: You can use Apache JMeter to simulate load on a web service and analyze its performance.

Steps:

1. Download and set up Apache JMeter.
2. Create a test plan with HTTP requests.
3. Run the test and observe the results.

Code:

- For this experiment, you'd be using a tool (JMeter) rather than writing Java code directly. Follow the official documentation to create a test plan:

  - Add a Thread Group (users).
  - Add HTTP Request samplers.
  - Add Listeners to observe the response.

Output:

Sample Output in JMeter:

```yaml
Summary Report:
  Number of Samples: 100
  Average Response Time: 250ms
  Throughput: 50 requests/sec
  Min Response Time: 200ms
  Max Response Time: 300ms
  Error %: 0%
```

Practical no. 7. UI Testing using Selenium WebDriver

Aim: Automate browser testing using Selenium WebDriver.

Description: Automate the testing of a simple web page (e.g., checking if a button click leads to the expected page).

Steps:

1. Install Selenium WebDriver.
2. Write a script that opens a browser, navigates to a page, and verifies a button click.

Code (Java with Selenium WebDriver):

java

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.By;
import org.junit.Test;
import static org.junit.Assert.*;

public class WebDriverTest {

    @Test
    public void testButtonClick() {
        System.setProperty("webdriver.chrome.driver", "path/to/chromedriver");
        WebDriver driver = new ChromeDriver();

        driver.get("https://example.com");
        driver.findElement(By.id("someButton")).click();

        // Check the result (e.g., URL change or page title)
        assertTrue(driver.getTitle().contains("New Page"));
```

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

Output:

Output (Console or IDE):

```arduino
Test passed: Button click leads to the correct page
```

```arduino
Page Title: "New Page"
Test passed successfully.
```