Beyond the Syllabus

Experiment No: 9

Obj – Perform a case study on JUnit testing and write a program for understanding how testing is actually performed in Java.

J Unit Testing:

JUnit is a unit testing framework for Java programming language. It plays a crucial role test-driven development, and is a family of unit testing frameworks collectively known as xUnit.

JUnit promotes the idea of "first testing then coding", which emphasizes on setting up the test data for a piece of code that can be tested first and then implemented. This approach is like "test a little, code a little, test a little, code a little." It increases the productivity of the programmer and the stability of program code, which in turn reduces the stress on the programmer and the time spent on debugging.

Features of JUnit

- JUnit is an open source framework, which is used for writing and running tests.
- Provides annotations to identify test methods.
- Provides assertions for testing expected results.
- Provides test runners for running tests.
- JUnit tests allow you to write codes faster, which increases quality.
- JUnit is elegantly simple. It is less complex and takes less time.
- JUnit tests can be run automatically and they check their own results and provide immediate feedback. There's no need to manually comb through a report of test results.
- JUnit tests can be organized into test suites containing test cases and even other test suites.
- JUnit shows test progress in a bar that is green if the test is running smoothly, and it turns red when a test fails.

What is a Unit Test Case ?

A Unit Test Case is a part of code, which ensures that another part of code (method) works as expected. To achieve the desired results quickly, a test framework is required. JUnit is a perfect unit test framework for Java programming language.

A formal written unit test case is characterized by a known input and an expected output, which is worked out before the test is executed. The known input should test a precondition and the expected output should test a post-condition.

There must be at least two unit test cases for each requirement − one positive test and one negative test. If a requirement has sub-requirements, each sub-requirement must have at least two test cases as positive and negative.

Java program

```java
package JavaTpoint. JunitExamples;
import java.util.ArrayList;
import java.util.List;
public class JunitTestCaseExample {
    private List<String> students = new ArrayList<String>();

    public void remove(String name) {
        students.remove(name);
    }

    public void add(String name) {
        students.add(name);
    }

    public void removeAll(){
        students.clear();
    }

    public int sizeOfStudent() {
        return students.size();
    }

}
```

TestJunitTestCase.java

```java
package JavaTpoint.JunitExamples;
import static org.junit.Assert.assertEquals;
import org.junit.Test;
public class TestJunitTestCaseExample {

    JunitTestCaseExample obj = new JunitTestCaseExample();

    @Test
    public void testAdd() {
        obj.add("Emma");
        obj.add("Ronan");
        obj.add("Antonio");
        obj.add("Paul");
        assertEquals("Adding 4 student to list", 4, obj.sizeOfStudent());
    }

    @Test
    public void testSize() {
        obj.add("Emma");
        obj.add("Ronan");
        obj.add("Antonio");
        assertEquals("Checking size of List", 3, obj.sizeOfStudent());
    }

    @Test
    public void testRemove() {
        obj.add("Antonio");
        obj.add("Paul");
        obj.remove("Paul");
        assertEquals("Removing 1 student from list", 1, obj.sizeOfStudent());
    }

    @Test
    public void removeAll() {
        obj.removeAll();
    }
}
```

TestRunner.java

```java
package JavaTpoint.JunitExamples;

```

```java
import org.junit.runner.Result;
import org.junit.runner.JUnitCore;
import org.junit.runner.notification.Failure;

public class TestRunner {
  public static void main(String[] args) {
    Result result = JUnitCore.runClasses(TestJunitTestCaseExample.class);

    for (Failure fail : result.getFailures()) {
      System.out.println(fail.toString());
    }

    System.out.println(result.wasSuccessful());
  }
}
```

Output –

```
true
```