**Task 5**

This project mainly deals with unit testing using JUnit framework. This assignment had three projects: Fibonacci, Rectangle and Vending Machine. The Fibonacci and Rectangle projects had errors and all the test cases were failed. We had to debug the code and find the errors. After correcting errors the test case were successful.

For Vending Machine project, we had to write test cases for VendingMachine.java and VendingMachineItem.java classes. It is necessary to understand how the code works while performing testing. While testing it is necessary to consider all aspects and check where the code fails. Testing should be done thoroughly. For example we have to test with good inputs as well as with bad inputs. Consider the pre-conditions and post conditions of the method to be tested. Such as addItem() method in VendingMachine.java class, item is added in Vending Machine only if the slot requested is empty and slot code is valid. While writing test cases, we have consider all these conditions and write three cases to test the addItem() method. First test case with good input and fulfills all the conditions. Second test case to add item in already occupied slot and third test with invalid slot code. It is required that one test case test only one feature of the method. In this way we know which test case failed or successful.

For writing tests using JUnit framework, we need to use methods of junit.framework.assert class. The test method checks the assertion and reports to Test Runner whether test failed or succeeded. It is necessary that all the methods must return void. For example assertEquals(double expected, double actual). Compares the actual and expected value and JUnit shows the results by indicating green bar if test succeed or red bar if test failed.

There are annotations provided by JUnit framework. These annotations provide information about test methods, which methods will run before or after test methods. Annotations

@Test: indicates that this method to be run as a test case.

@Before: indicates that this method will run before each test method

@After: indicates that this method will run after each test method

@BeforeClass: indicates that this method will run once before any of the test methods in the class.

@AfterClass: indicates that will run only one time after all test methods in the class. Usually to perform clean-up.

JUnit provides setUp() and tearDown() methods. The setUp() method usually have @Before or @BeforeClass annotations. In setUp() method all the fields are initialized or to open database connections . The tearDown() method performs cleanup functions such as setting the references to null.

JUnit framework is simple to use and understand. It helps developers to write test cases as the system is developed. JUnit makes it easy to add unit tests. If the developer is not using JUnit, then the developer will have to use the system.out.print() statement to view the output on the console. JUnit aggregates the results of the tests in a structured manner. With JUnit, we can run multiple tests simultaneously and we can correct the errors as they are detected.

Using JUnit assert methods we can compare Strings, double, Objects, arrays etc.

It was good learning experience in writing test cases using JUnit.