



**California State University**  
**DOMINGUEZ HILLS**

**EVENT MANAGEMENT SYSTEM: GROUP NO 6**  
**Part 5 Software Verification, Validation and Testing**

**Aishwarya Murkute**      **ID: 210567047**

**Aditya Lalchadani**      **ID:210052091**

**Hanish Tumnanapalli**      **ID:210570765**

**Nishant Joshi**      **ID:21055923**

**Susmita Patange**      **ID: 210551057**

**Rineel Reddy Lingala**      **ID:204950668**

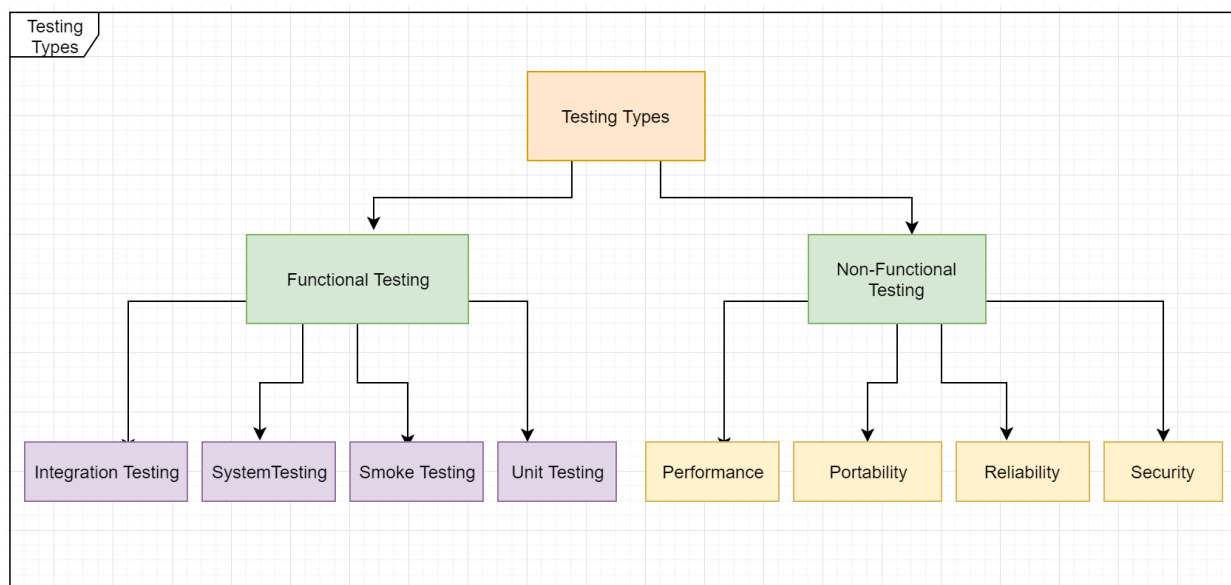
<b>INDEX</b>	<b>Page</b>
<b>Part 5: Software Verification, Validation and Testing</b>	
<b>1. A software test plan for your system, including various tests.</b>	4
<b>2. A list of unit test cases and their coverage.</b>	5
<b>3. The source code of unit test with documentation.</b>	7

# 1. A software test plan for your system, including various tests.

Software testing is the process which is used to evaluate the software and find the errors and bugs in the software. It is not a single step process it is used throughout the software development cycle. It helps to determine the software efficiency, the stability, the scalability and performance of the software. Software testing is one of the most significant parts of the development cycle, where there is an investigation conducted to provide detailed information and report about the quality of the software product. An overview of software testing involves finding the software defects fixing the errors and making sure that the developed software meets the requirements of the user.

Hence, in other words, it is the process of evaluating a system or the component with the intent to find whether or not it satisfies requirements specified by the user. Testing is involved in almost every stage of the software cycle. However, the objective of testing is different every time we perform a test. There are various approaches to testing and models for software development but there are very few process models which address the minimization of risk and prevention of defects.

**Following are the different testing type:**



## 2. A list of unit test cases and their coverage.

We designed three test cases stated as follows, since our code is in java we are using junit for testing

Test Case ID	001
Test Case Summary	To verify whether the admin module is working or not
Prerequisite	The user should know how to access or login into the admin module
Test Procedure	Enter the details in the admin page. Validate the details. Update the organizers
Expected Output	All the details are visible to the users. The admin is able to update and access all the details
Actual Output	When the admin makes changes and updates the website then all the updates are visible to the users.
Status	Success
Test Environment	JUnit testing

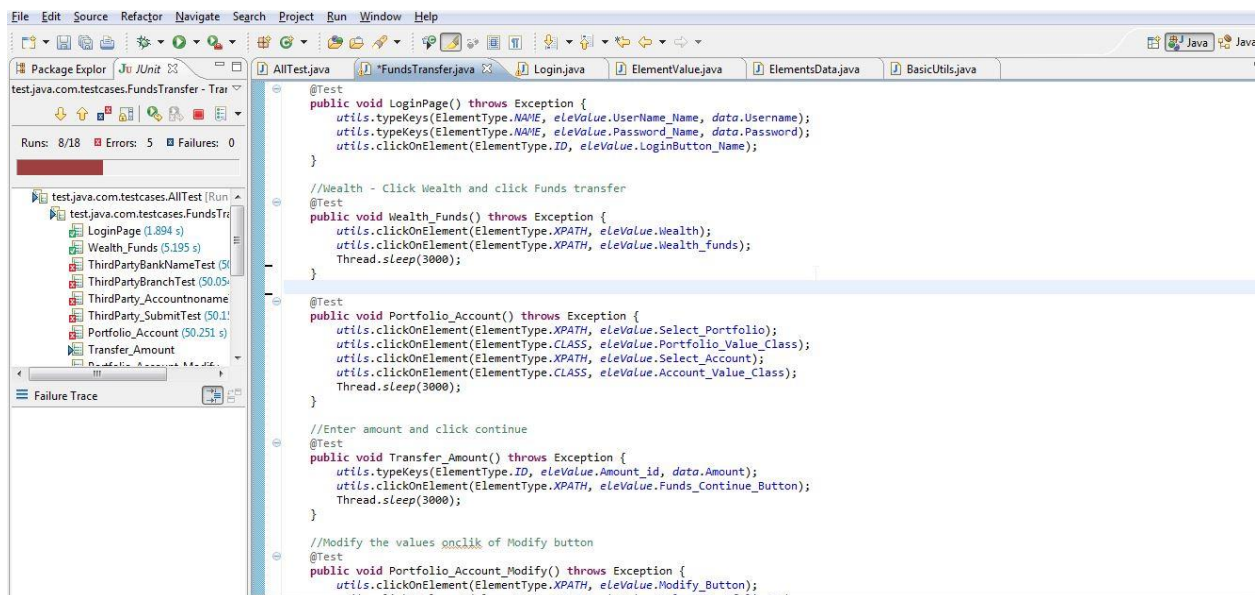
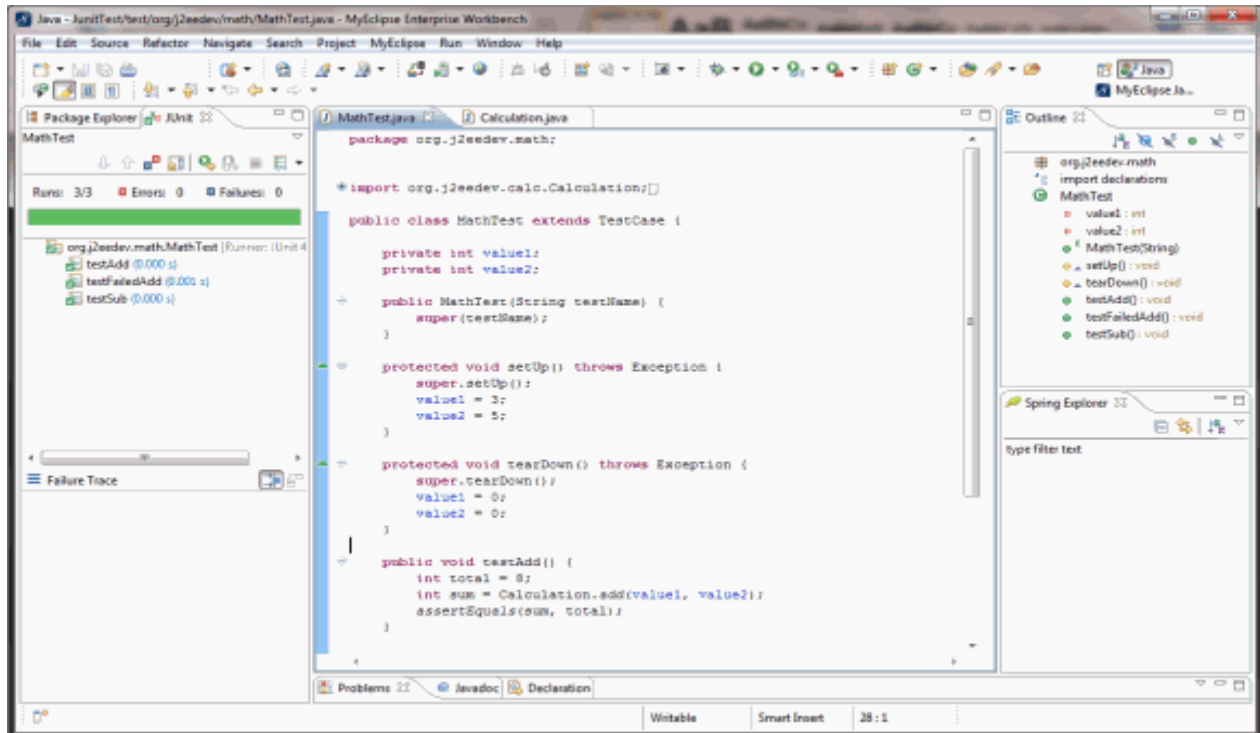
Test Case ID	002
Test Case Summary	To verify whether the Customer module is working or not
Prerequisite	The user should know how to access or login into the customer module and how to register as a new user
Test Procedure	Enter the details in the customer page. Validate the details. Make booking View events
Expected Output	All the details are visible to the users. The customer is able to register and attend the event
Actual Output	The customer is successfully able to register and <u>attent</u> and view booking for the event
Status	Success
Test Environment	JUnit testing

<b>Test Case Summary</b>	<b>003</b>
Prerequisite	To verify whether the Organizer module is working or not
Test Procedure	The organizer should be able to or login into the organizer module and how to register as a organizer user, and be able to upload the files, so that they can be viewed by the user.
Expected Output	The organizer should be able to upload the files and add details of the event to be organized
Actual Output	The organizer successfully upload the details of the event and it is visible to the users.
Status	Successful
Test Environment	Junit

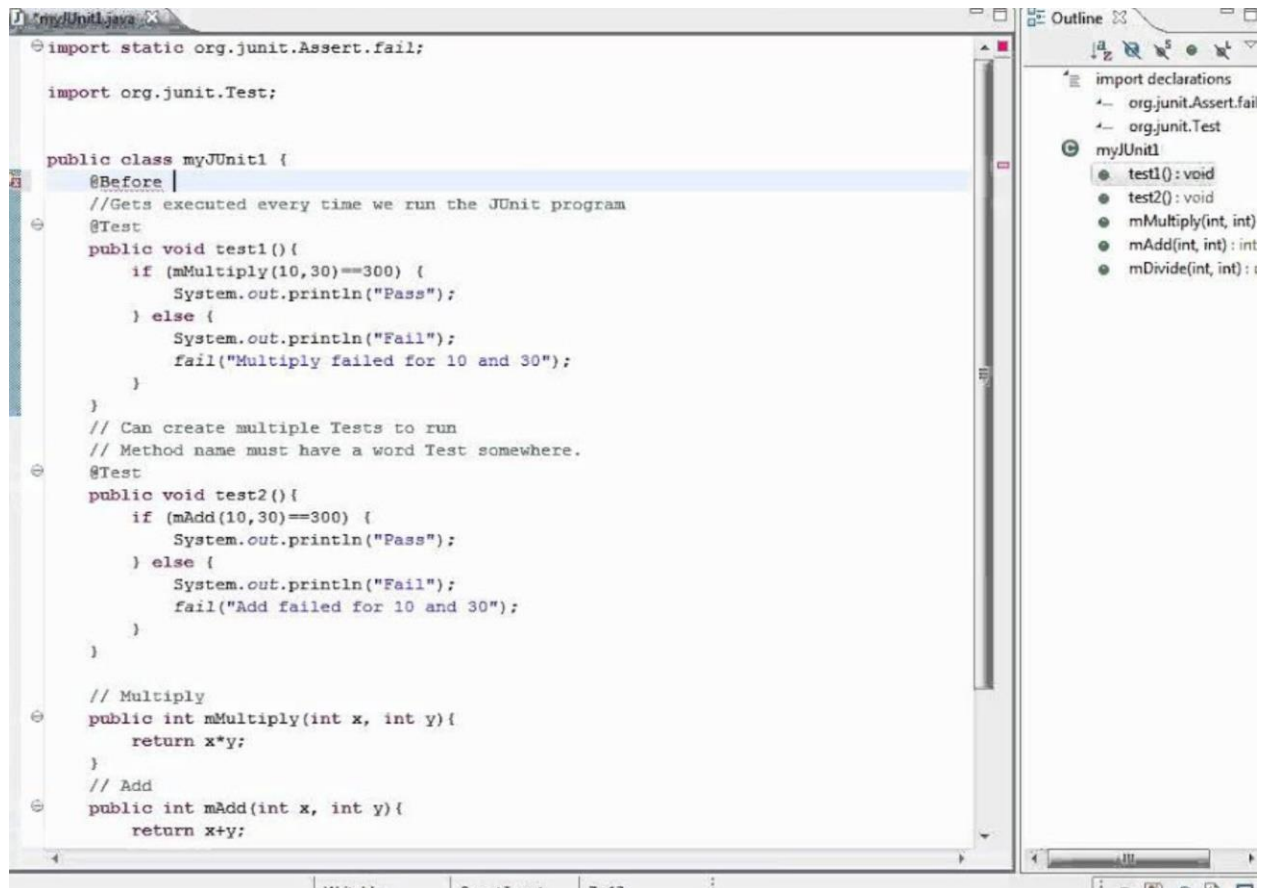
### 3. The source code of unit test with documentation.

Source for testing the three modules using junit

Results and Analysis



## EVENT MANAGEMENT SYSTEM GROUP NO. 6



The image shows a screenshot of an IDE with a Java file named `myJUnit1.java` open. The code is a JUnit test class that includes imports for `org.junit.Assert.fail` and `org.junit.Test`. It defines a `myJUnit1` class with two test methods, `test1()` and `test2()`, and two utility methods, `mMultiply()` and `mAdd()`. The `@Before` annotation is used to indicate that the code between `@Before` and `@Test` is executed before each test method. The `@Test` annotation is used to mark the test methods. The `fail()` method is used to indicate a test failure. The `mMultiply()` method returns the product of two integers, and the `mAdd()` method returns the sum of two integers. The IDE's Outline view on the right shows the class structure, including the imports, the `myJUnit1` class, and the methods `test1()`, `test2()`, `mMultiply()`, and `mAdd()`.

```
import static org.junit.Assert.fail;

import org.junit.Test;

public class myJUnit1 {
    @Before
    //Gets executed every time we run the JUnit program
    @Test
    public void test1(){
        if (mMultiply(10,30)==300) {
            System.out.println("Pass");
        } else {
            System.out.println("Fail");
            fail("Multiply failed for 10 and 30");
        }
    }
    // Can create multiple Tests to run
    // Method name must have a word Test somewhere.
    @Test
    public void test2(){
        if (mAdd(10,30)==300) {
            System.out.println("Pass");
        } else {
            System.out.println("Fail");
            fail("Add failed for 10 and 30");
        }
    }

    // Multiply
    public int mMultiply(int x, int y){
        return x*y;
    }
    // Add
    public int mAdd(int x, int y){
        return x+y;
    }
}
```

Outline

- import declarations
  - org.junit.Assert.fail
  - org.junit.Test
- myJUnit1
  - test1(): void
  - test2(): void
  - mMultiply(int, int)
  - mAdd(int, int): int
  - mDivide(int, int): int