EASYMOCK - JUNIT INTEGRATION

In this chapter, we'll learn how to integrate JUnit and EasyMock together. Here we will create a Math Application which uses CalculatorService to perform basic mathematical operations such as addition, subtraction, multiply, and division. We'll use EasyMock to mock the dummy implementation of CalculatorService. In addition, we've made extensive use of annotations to showcase their compatibility with both JUnit and EasyMock.

The process is discussed below in a step-by-step manner.

Step 1: Create an interface called CalculatorService to provide mathematical functions

File: CalculatorService.java

```
public interface CalculatorService {
   public double add(double input1, double input2);
   public double subtract(double input1, double input2);
   public double multiply(double input1, double input2);
   public double divide(double input1, double input2);
}
```

Step 2: Create a JAVA class to represent MathApplication

File: MathApplication.java

```
public class MathApplication {
   private CalculatorService calcService;

   public void setCalculatorService(CalculatorService calcService){
      this.calcService = calcService;
   }

   public double add(double input1, double input2){
      return calcService.add(input1, input2);
   }

   public double subtract(double input1, double input2){
      return calcService.subtract(input1, input2);
   }

   public double multiply(double input1, double input2){
      return calcService.multiply(input1, input2);
   }

   public double divide(double input1, double input2){
      return calcService.divide(input1, input2);
   }
}
```

Step 3: Test the MathApplication class

Let's test the MathApplication class, by injecting in it a mock of calculatorService. Mock will be created by EasyMock.

File: MathApplicationTester.java

```
import org.easymock.EasyMock;
import org.easymock.EasyMockRunner;
import org.easymock.Mock;
import org.easymock.TestSubject;

import org.junit.Assert;
import org.junit.Before;
```

```
import org.junit.Test;
import org.junit.runner.RunWith;

// @RunWith attaches a runner with the test class to initialize the test data
@RunWith(EasyMockRunner.class)
public class MathApplicationTester {

   // @TestSubject annotation is used to identify class which is going to use the mock
object
   @TestSubject
   MathApplication mathApplication = new MathApplication();

   //@Mock annotation is used to create the mock object to be injected
   @Mock
   CalculatorService calcService;

   @Test
   public void testAdd(){
      //add the behavior of calc service to add two numbers
      EasyMock.expect(calcService.add(10.0,20.0)).andReturn(30.00);

      //activate the mock
      EasyMock.replay(calcService);

      //test the add functionality
      Assert.assertEquals(mathApplication.add(10.0, 20.0),30.0,0);
   }
}
```

Step 4: Create a class to execute to test cases

Create a java class file named TestRunner in **C:\> EasyMock_WORKSPACE** to execute Test cases .

File: TestRunner.java

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

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

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

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

Step 5: Verify the Result

Compile the classes using **javac** compiler as follows:

```
C:\EasyMock_WORKSPACE>javac CalculatorService.java MathApplication.java
MathApplicationTester.java TestRunner.java
```

Now run the Test Runner to see the result:

```
C:\EasyMock_WORKSPACE>java TestRunner
```

Verify the output.

```
true
```

To learn more about JUnit, please refer to JUnit Tutorial at Tutorials Point.