About the Tutorial

EasyMock is a mocking framework, JAVA-based library that is used for effective unit testing of JAVA applications. EasyMock is used to mock interfaces so that a dummy functionality can be added to a mock interface that can be used in unit testing.

This tutorial will help you learn how to create unit tests with EasyMock as well as how to use its APIs in a simple and intuitive way.

Audience

This tutorial is meant for Java developers, from novice to expert level, who would like to improve the quality of their software through unit testing and test-driven development.

After completing this tutorial, you will gain sufficient exposure to EasyMock from where you can take yourself to next levels of expertise.

Prerequisites

Readers must have a working knowledge of JAVA programming language in order to make the best of this tutorial. Knowledge of JUnit is an added advantage.

Copyright & Disclaimer

© Copyright 2014 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com
Table of Contents

About the Tutorial
Audience
Prerequisites
Copyright & Disclaimer
Table of Contents

1. OVERVIEW
   What is Mocking?
   EasyMock

2. ENVIRONMENT SETUP
   System Requirement
   Step 1 – Verify Java Installation on Your Machine
   Step 2: Set JAVA Environment
   Step 3: Download EasyMock Archive
   Step 4: Download EasyMock Dependencies
   Step 5: Set EasyMock Environment
   Step 5: Set CLASSPATH Variable
   Step 6: Download JUnit Archive
   Step 7: Set JUnit Environment
   Step 8: Set CLASSPATH Variable

3. FIRST APPLICATION

4. JUNIT INTEGRATION

5. ADDING BEHAVIOR
   Example without EasyMock.Replay()
   Example with EasyMock.Replay()
6. VERIFYING BEHAVIOR

   Example without EasyMock.Verify().................................................................28
   Example with EasyMock.Verify().....................................................................32

7. EXPECTING CALLS

   Example with calcService.serviceUsed() called once ........................................37
   Example with calcService.serviceUsed() Called Twice .................................41
   Example without Calling calcService.serviceUsed() .......................................45

8. VARYING CALLS

   Example with times (min,max) ..........................................................................50
   Example with atLeastOnce .................................................................................54
   Example with anyTimes ....................................................................................58

9. EXCEPTION HANDLING

   Example .............................................................................................................63

10. CREATEMOCK

    Example ...........................................................................................................68

11. CREATESTRICTMOCK

    Example ...........................................................................................................73

12. CREATENICEMOCK

    Example ...........................................................................................................78

13. EASYMOCKSUPPORT

    Example ...........................................................................................................83
What is Mocking?
Mocking is a way to test the functionality of a class in isolation. Mocking does not require a database connection or properties file read or file server read to test a functionality. Mock objects do the mocking of the real service. A mock object returns a dummy data corresponding to some dummy input passed to it.

EasyMock
EasyMock facilitates creating mock objects seamlessly. It uses Java Reflection in order to create mock objects for a given interface. Mock objects are nothing but proxy for actual implementations. Consider a case of Stock Service which returns the price details of a stock. During development, the actual stock service cannot be used to get real-time data. So we need a dummy implementation of the stock service. EasyMock can do the same very easily as its name suggests.

Benefits of EasyMock
- **No Handwriting** – No need to write mock objects on your own.
- **Refactoring Safe** – Renaming interface method names or reordering parameters will not break the test code as Mocks are created at runtime.
- **Return value support** – Supports return values.
- **Exception support** – Supports exceptions.
- **Order check support** – Supports check on order of method calls.
- **Annotation support** – Supports creating mocks using annotation.

Consider the following code snippet.

```java
package com.tutorialspoint.mock;
```
import java.util.ArrayList;
import java.util.List;

import org.easymock.EasyMock;

public class PortfolioTester {
    public static void main(String[] args) {

        // Create a portfolio object which is to be tested
        Portfolio portfolio = new Portfolio();

        // Creates a list of stocks to be added to the portfolio
        List<Stock> stocks = new ArrayList<Stock>();
        Stock googleStock = new Stock("1","Google", 10);
        Stock microsoftStock = new Stock("2","Microsoft",100);

        stocks.add(googleStock);
        stocks.add(microsoftStock);

        // Create the mock object of stock service
        StockService stockServiceMock = 
            EasyMock.createMock(StockService.class);
// mock the behavior of stock service to return
// the value of various stocks
EasyMock.expect(stockServiceMock.getPrice(googleStock))
  .andReturn(50.00);
EasyMock.expect(stockServiceMock.getPrice(microsoftStock))
  .andReturn(1000.00);

EasyMock.replay(stockServiceMock);

// add stocks to the portfolio
portfolio.setStocks(stocks);

// set the stockService to the portfolio
portfolio.setStockService(stockServiceMock);

double marketValue = portfolio.getMarketValue();

// verify the market value to be
// 10*50.00 + 100*1000.00 = 500.00 + 100000.00 = 100500
System.out.println("Market value of the portfolio: " + marketValue);
}
Let's understand the important concepts of the above program. The complete code is available in the chapter [First Application](#).

- **Portfolio** – An object to carry a list of stocks and to get the market value computed using stock prices and stock quantity.
- **Stock** – An object to carry the details of a stock such as its id, name, quantity, etc.
- **StockService** – A stock service returns the current price of a stock.
- **EasyMock.createMock(...)** – EasyMock created a mock of stock service.
- **EasyMock.expect(...).andReturn(...)** – Mock implementation of getPrice method of stockService interface. For googleStock, return 50.00 as price.
- **EasyMock.replay(...)** – EasyMock prepares the Mock object to be ready so that it can be used for testing.
- **portfolio.setStocks(...)** – The portfolio now contains a list of two stocks.
- **portfolio.setStockService(...)** - Assigns the stockService Mock object to the portfolio.
- **portfolio.getMarketValue()()** – The portfolio returns the market value based on its stocks using the mock stock service.
EasyMock is a framework for Java, so the very first requirement is to have JDK installed in your machine.

**System Requirement**

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Requirement Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>JDK</td>
<td>1.5 or above.</td>
</tr>
<tr>
<td>Memory</td>
<td>no minimum requirement.</td>
</tr>
<tr>
<td>Disk Space</td>
<td>no minimum requirement.</td>
</tr>
<tr>
<td>Operating System</td>
<td>no minimum requirement.</td>
</tr>
</tbody>
</table>

**Step 1 – Verify Java Installation on Your Machine**

Open the console and execute the following `java` command.

<table>
<thead>
<tr>
<th>OS</th>
<th>Task</th>
<th>Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Open Command Console</td>
<td>c:&gt; java -version</td>
</tr>
<tr>
<td>Linux</td>
<td>Open Command Terminal</td>
<td>$ java -version</td>
</tr>
<tr>
<td>Mac</td>
<td>Open Terminal</td>
<td>machine:~ joseph$ java -version</td>
</tr>
</tbody>
</table>

Let's verify the output for all the operating systems:

<table>
<thead>
<tr>
<th>OS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>java version &quot;1.6.0_21&quot;</td>
</tr>
<tr>
<td></td>
<td>Java(TM) SE Runtime Environment (build 1.6.0_21-b07)</td>
</tr>
</tbody>
</table>
If you do not have Java installed, install the Java Software Development Kit (SDK) from http://www.oracle.com/technetwork/java/javase/downloads/index.html.

We assume you have Java 1.6.0_21 installed on your system for this tutorial.

**Step 2: Set JAVA Environment**

Set the `JAVA_HOME` environment variable to point to the base directory location where Java is installed on your machine. For example,

<table>
<thead>
<tr>
<th>OS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Set the environment variable JAVA_HOME to C:\Program Files\Java\jdk1.6.0_21</td>
</tr>
<tr>
<td>Linux</td>
<td>export JAVA_HOME=/usr/local/java-current</td>
</tr>
<tr>
<td>Mac</td>
<td>export JAVA_HOME=/Library/Java/Home</td>
</tr>
</tbody>
</table>

Append the location of the Java compiler to your System Path.

<table>
<thead>
<tr>
<th>OS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Verify Java Installation using the command `java -version` as explained above.

**Step 3: Download EasyMock Archive**

Download the latest version of EasyMock from [http://sourceforge.net/projects/easymock/files/EasyMock/3.2/easymock-3.2.zip/download](http://sourceforge.net/projects/easymock/files/EasyMock/3.2/easymock-3.2.zip/download). Save the zip folder on your C drive, let’s say, C:\EasyMock.

<table>
<thead>
<tr>
<th>OS</th>
<th>Archive name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>easymock-3.2.zip</td>
</tr>
<tr>
<td>Linux</td>
<td>easymock-3.2.zip</td>
</tr>
<tr>
<td>Mac</td>
<td>easymock-3.2.zip</td>
</tr>
</tbody>
</table>

**Step 4: Download EasyMock Dependencies**

Download the latest version of cglib jar file from [https://github.com/cglib/cglib/releases](https://github.com/cglib/cglib/releases) and copy it onto C:\EasyMock folder. At the time of writing this tutorial, the latest version was 3.1.

Download the latest version of objenesis zip file from [http://objenesis.org/download.html](http://objenesis.org/download.html) and copy it onto C:\EasyMock folder. At the time of writing this tutorial, the latest version was 2.1. Extract objenesis-2.1.jar to C:\EasyMock folder.
Step 5: Set EasyMock Environment

Set the EasyMock_HOME environment variable to point to the base directory location where EasyMock and dependency jars are stored on your machine. The following table shows how to set the environment variable on different operating systems, assuming we've extracted easymock-3.2.jar, cglib-3.1.jar, and objenesis-2.1.jar onto C:\EasyMock folder.

<table>
<thead>
<tr>
<th>OS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Set the environment variable EasyMock_HOME to C:\EasyMock</td>
</tr>
<tr>
<td>Linux</td>
<td>export EasyMock_HOME=/usr/local/EasyMock</td>
</tr>
<tr>
<td>Mac</td>
<td>export EasyMock_HOME=/Library/EasyMock</td>
</tr>
</tbody>
</table>

Step 6: Set CLASSPATH Variable

Set the CLASSPATH environment variable to point to the location where EasyMock and dependency jars are stored. The following table shows how to set the CLASSPATH variable on different operating systems.

<table>
<thead>
<tr>
<th>OS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Set the environment variable CLASSPATH to %CLASSPATH%;%EasyMock_HOME%easymock-3.2.jar;%EasyMock_HOME%cglib-3.1.jar;%EasyMock_HOME%objenesis-2.1.jar;;</td>
</tr>
<tr>
<td>Linux</td>
<td>export CLASSPATH=$CLASSPATH:$EasyMock_HOME/easymock-3.2.jar:$EasyMock_HOME/cglib-3.1.jar:$EasyMock_HOME/objenesis-2.1.jar:;</td>
</tr>
<tr>
<td>Mac</td>
<td>export CLASSPATH=$CLASSPATH:$EasyMock_HOME/easymock-3.2.jar:$EasyMock_HOME/cglib-3.1.jar:$EasyMock_HOME/objenesis-2.1.jar:;</td>
</tr>
</tbody>
</table>
Step 7: Download JUnit Archive

Download the latest version of JUnit jar file from https://github.com/junit-team/junit/wiki/Download-and-Install. Save the folder at the location C:\JUnit.

<table>
<thead>
<tr>
<th>OS</th>
<th>Archive name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>junit4.11.jar, hamcrest-core-1.2.1.jar</td>
</tr>
<tr>
<td>Linux</td>
<td>junit4.11.jar, hamcrest-core-1.2.1.jar</td>
</tr>
<tr>
<td>Mac</td>
<td>junit4.11.jar, hamcrest-core-1.2.1.jar</td>
</tr>
</tbody>
</table>

Step 8: Set JUnit Environment

Set the JUNIT_HOME environment variable to point to the base directory location where JUnit jars are stored on your machine. The following table shows how to set this environment variable on different operating systems, assuming we've stored junit4.11.jar and hamcrest-core-1.2.1.jar at C:\JUnit.

<table>
<thead>
<tr>
<th>OS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Set the environment variable JUNIT_HOME to C:\JUNIT</td>
</tr>
<tr>
<td>Linux</td>
<td>export JUNIT_HOME=/usr/local/JUNIT</td>
</tr>
<tr>
<td>Mac</td>
<td>export JUNIT_HOME=/Library/JUNIT</td>
</tr>
</tbody>
</table>

Step 9: Set CLASSPATH Variable

Set the CLASSPATH environment variable to point to the JUNIT jar location. The following table shows how it is done on different operating systems.
<table>
<thead>
<tr>
<th>OS</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Windows</td>
<td>Set the environment variable CLASSPATH to %CLASSPATH%;%JUNIT_HOME\junit4.11.jar;%JUNIT_HOME\hamcrest-core-1.2.1.jar;;</td>
</tr>
<tr>
<td>Linux</td>
<td>export CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.11.jar:$JUNIT_HOME/hamcrest-core-1.2.1.jar:.</td>
</tr>
<tr>
<td>Mac</td>
<td>export CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.11.jar:$JUNIT_HOME/hamcrest-core-1.2.1.jar:.</td>
</tr>
</tbody>
</table>
Before going into the details of the EasyMock Framework, let’s see an application in action. In this example, we've created a mock of Stock Service to get the dummy price of some stocks and unit tested a java class named Portfolio.

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

**Step 1: Create a JAVA class to represent the Stock**

**Stock.java**

```java
public class Stock {
    private String stockId;
    private String name;
    private int quantity;

    public Stock(String stockId, String name, int quantity){
        this.stockId = stockId;
        this.name = name;
        this.quantity = quantity;
    }

    public String getStockId() {
        return stockId;
    }
}
```

3. FIRST APPLICATION
public void setStockId(String stockId) {
    this.stockId = stockId;
}

public int getQuantity() {
    return quantity;
}

public String getTicker() {
    return name;
}

---

**Step 2: Create an interface StockService to get the price of a stock**

**StockService.java**

```java
public interface StockService {
    public double getPrice(Stock stock);
}
```

**Step 3: Create a class Portfolio to represent the portfolio of any client**

**Portfolio.java**

```java
import java.util.List;

public class Portfolio {
    private StockService stockService;
```
private List<Stock> stocks;

public StockService getStockService() {
    return stockService;
}

public void setStockService(StockService stockService) {
    this.stockService = stockService;
}

public List<Stock> getStocks() {
    return stocks;
}

public void setStocks(List<Stock> stocks) {
    this.stocks = stocks;
}

public double getMarketValue() {
    double marketValue = 0.0;
    for (Stock stock : stocks) {
        marketValue += stockService.getPrice(stock) * stock.getQuantity();
    }
}
Step 4: Test the Portfolio class

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

PortfolioTester.java

```java
import java.util.ArrayList;
import java.util.List;
import org.easymock.EasyMock;
public class PortfolioTester {
    Portfolio portfolio;
    StockService stockService;

    public static void main(String[] args){
        PortfolioTester tester = new PortfolioTester();
        tester.setUp();
        System.out.println(tester.testMarketValue() ? "pass" : "fail");
    }

    public void setUp(){
        //Create a portfolio object which is to be tested
```
portfolio = new Portfolio();

//Create the mock object of stock service
stockService = EasyMock.createMock(StockService.class);

//set the stockService to the portfolio
portfolio.setStockService(stockService);
}

public boolean testMarketValue(){

  //Creates a list of stocks to be added to the portfolio
  List<Stock> stocks = new ArrayList<Stock>();
  Stock googleStock = new Stock("1","Google", 10);
  Stock microsoftStock = new Stock("2","Microsoft",100);
  stocks.add(googleStock);
  stocks.add(microsoftStock);

  //add stocks to the portfolio
  portfolio.setStocks(stocks);

  // mock the behavior of stock service to return
  // the value of various stocks
  EasyMock.expect(stockService.getPrice(googleStock)).andReturn(50.00);
  EasyMock.expect(stockService.getPrice(microsoftStock))
.andReturn(1000.00);

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

double marketValue = portfolio.getMarketValue();
return marketValue == 100500.0;
}
}

Step 5: Verify the result
Compile the classes using javac compiler as follows:

```
C:\EasyMock_WORKSPACE>javac Stock.java StockService.java Portfolio.java PortfolioTester.java
```

Now run the PortfolioTester to see the result:

```
C:\EasyMock_WORKSPACE>java PortfolioTester
pass
```
In this chapter, we'll learn how to integrate JUnit and EasyMock together. For JUnit tutorial, please refer to JUnit. 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**

**CalculatorService.java**

```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**

**MathApplication.java**

```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.

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(EasyMockRunner.class)
public class MathApplicationTester {

    @TestSubject
    MathApplication mathApplication = new MathApplication();

    @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 case(s).

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());
        }
    }
}
Step 5: Verify the Result

Compile the classes using `javac` compiler as follows:

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

Now run the Test Runner to see the result:

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

Verify the output.

```bash
true
```
5. ADDING BEHAVIOR

EasyMock adds a functionality to a mock object using the methods `expect()` and `expectLastCall()`. Take a look at the following code snippet.

```java
//add the behavior of calc service to add two numbers
EasyMock.expect(calcService.add(10.0, 20.0)).andReturn(30.00);
```

Here we've instructed EasyMock to give a behavior of adding 10 and 20 to the add method of calcService and as a result, to return the value of 30.00.

At this point of time, Mock simply recorded the behavior but it is not working as a mock object. After calling replay, it works as expected.

```java
//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);
```

**Example without EasyMock.Replay()**

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

**CalculatorService.java**

```java
public interface CalculatorService {
    public double add(double input1, double input2);
    public double subtract(double input1, double input2);
}
```
Step 2: Create a JAVA class to represent MathApplication

MathApplication.java

```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.

MathApplicationTester.java

```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
//@RunWith(EasyMockRunner.class)
public class MathApplicationTester {

    // @TestSubject annotation is used to identify the 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: Execute test cases
Create a java class file named TestRunner in C:\EasyMock_WORKSPACE to execute the test case(s).
**TestRunner.java**

```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.

```
testAdd(MathApplicationTester): expected:<0.0> but was:<30.0>
```
Example with EasyMock.Replay()

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

CalculatorService.java

```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.

MathApplication.java

```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.

MathApplicationTester.java

```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
Step 4: Execute test cases
Create a java class file named TestRunner in C:\EasyMock_WORKSPACE to execute Test case(s).

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(failure.getMessage());
        }
        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.
End of ebook preview
If you liked what you saw...
Buy it from our store @ https://store.tutorialspoint.com