EASYMOCK - OVERVIEW

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.

```
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 stocks = new ArrayList();
      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.