**Basic Mockito Testing:**

1. System Requirements (JAR Files)
   1. mockito-all-2.0.2-beta.jar
   2. junit4.11.jar
   3. hamcrest-core-1.2.1.jar
2. First Application that uses Mockito Testing (see attached java files, this should return true when program is executed)

***Portfolio.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();

}

return marketValue;

}

}

***PortfolioTester.java***

import java.util.ArrayList;

import java.util.List;

import static org.mockito.Mockito.\*;

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 = mock(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

when(stockService.getPrice(googleStock)).thenReturn(60.00);

when(stockService.getPrice(microsoftStock)).thenReturn(1000.00);

double marketValue = portfolio.getMarketValue();

return marketValue == 100500.0;

}

}

***Stock.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;

}

public void setStockId(String stockId) {

this.stockId = stockId;

}

public int getQuantity() {

return quantity;

}

public String getTicker() {

return name;

}

}

***StockService.java***

public interface StockService {

public double getPrice(Stock stock);

}

1. Mockito with Junit Integration (Calculator Service)

***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);

}

***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);

}

}

***MathApplicationTester.java***

import static org.mockito.Mockito.when;

import org.junit.Assert;

import org.junit.Test;

import org.junit.runner.RunWith;

import org.mockito.InjectMocks;

import org.mockito.Mock;

import org.mockito.runners.MockitoJUnitRunner;

// @RunWith attaches a runner with the test class to initialize the test data

@RunWith(MockitoJUnitRunner.class)

public class MathApplicationTester {

//@InjectMocks annotation is used to create and inject the mock object

@InjectMocks

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

when(calcService.add(10.0,20.0)).thenReturn(30.00);

//test the add functionality

Assert.assertEquals(mathApplication.add(10.0, 20.0),30.0,0);

}

}

***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());

}

}

1. The following Mockito Behavior can be added to the code shown above to perform Mockito Testing:

//add the behavior of calc service to add two numbers

when(calcService.add(10.0,20.0)).thenReturn(30.00);

//test the add functionality

Assert.assertEquals(mathApplication.add(10.0, 20.0),30.0,0);

//verify call to calcService is made or not with same arguments.

verify(calcService).add(10.0, 20.0);

//add the behavior of calc service to add two numbers

when(calcService.add(10.0,20.0)).thenReturn(30.00);

//limit the method call to 1, no less and no more calls are allowed

verify(calcService, times(1)).add(10.0, 20.0);

//check a minimum 1 call count

verify(calcService, atLeastOnce()).subtract(20.0, 10.0);

//check if add function is called minimum 2 times

verify(calcService, atLeast(2)).add(10.0, 20.0);

//check if add function is called maximum 3 times

verify(calcService, atMost(3)).add(10.0,20.0);

//add the behavior to throw exception

doThrow(new Runtime Exception("divide operation not implemented"))

.when(calcService).add(10.0,20.0);

Mockito provides the capability to reset a mock so that it can be reused later

//reset mock

reset(calcService);

*source of this documentation is from: https://www.tutorialspoint.com/mockito/*