mockito

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Mockito is a mocking framework, JAVA-based library that is used for effective unit testing of JAVA applications. Mockito 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 should help you learn how to create unit tests with Mockito 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 should gain sufficient exposure to Mockito 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 most of this tutorial. Knowledge of JUnit is an added advantage.

Copyright & Disclaimer

Table of Contents

1. Mockito – 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.

Mockito

Mockito 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. Mockito can do the same very easily, as its name suggests.

Benefits of Mockito

- **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 static org.mockito.Mockito.*;


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 = mock(StockService.class);

      // mock the behavior of stock service to return the value of various stocks
      when(stockServiceMock.getPrice(googleStock)).thenReturn(50.00);
      when(stockServiceMock.getPrice(microsoftStock)).thenReturn(1000.00);

      //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.
- **mock(…)** – Mockito created a mock of stock service.
- **when(…).thenReturn(…)** – Mock implementation of getPrice method of stockService interface. For googleStock, return 50.00 as price.
- **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.

2. Mockito – Environment Setup

Mockito is a framework for Java, so the very first requirement is to have JDK installed in your machine.

System Requirement

JDK	1.5 or above.
Memory	no minimum requirement.
Disk Space	no minimum requirement.
Operating System	no minimum requirement.

Step 1: Verify Java Installation on Your Machine

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

OS	Task	Command
Windows	Open Command Console	c:\> java -version
Linux	Open Command Terminal	$ java -version
Mac	Open Terminal	machine:> joseph$ java -version

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

OS	Output
Windows	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
Linux	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07) Java HotSpot(TM) Client VM (build 17.0-b17, mixed mode, sharing)
Mac	java version "1.6.0_21" Java(TM) SE Runtime Environment (build 1.6.0_21-b07)

	Java HotSpot(TM)64-Bit Server VM (build 17.0-b17, mixed mode, sharing)

If you do not have Java installed, To install the Java Software Development Kit (SDK) click here.

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,

OS	Output
Windows	Set the environment variable JAVA_HOME to C:\Program Files\Java\jdk1.6.0_21
Linux	export JAVA_HOME=/usr/local/java-current
Mac	export JAVA_HOME=/Library/Java/Home

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

OS	Output
Windows	Append the string ;C:\Program Files\Java\jdk1.6.0_21\bin to the end of the system variable, Path.
Linux	export PATH=$PATH:$JAVA_HOME/bin/
Mac	not required

Verify Java Installation using the command **java -version** as explained above.

Step 3: Download Mockito-All Archive

To download the latest version of Mockito from Maven Repository click here.

Save the jar file on your C drive, let's say, C:\>Mockito.

OS	Archive name
Windows	mockito-all-2.0.2-beta.jar
Linux	mockito-all-2.0.2-beta.jar
Mac	mockito-all-2.0.2-beta.jar

Step 4: Set Mockito Environment

Set the **Mockito_HOME** environment variable to point to the base directory location where Mockito 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 mockito-all-2.0.2-beta.jar onto C:\>Mockito folder.

OS	Output
Windows	Set the environment variable Mockito_HOME to C:\Mockito
Linux	export Mockito_HOME=/usr/local/Mockito
Mac	export Mockito_HOME=/Library/Mockito

Step 5: Set CLASSPATH Variable

Set the **CLASSPATH** environment variable to point to the location where Mockito jar is stored. The following table shows how to set the CLASSPATH variable on different operating systems.

OS	Output
Windows	Set the environment variable CLASSPATH to %CLASSPATH%;%Mockito_HOME%\mockito-all-2.0.2-beta.jar;.;
Linux	export CLASSPATH=$CLASSPATH:$Mockito_HOME/mockito-all-2.0.2-beta.jar:.
Mac	export CLASSPATH=$CLASSPATH:$Mockito_HOME/mockito-all-2.0.2-beta.jar:.

Step 6: Download JUnit Archive

Download the latest version of JUnit jar file from Github. Save the folder at the location C:\>Junit.

OS	Archive name
Windows	junit4.11.jar, hamcrest-core-1.2.1.jar
Linux	junit4.11.jar, hamcrest-core-1.2.1.jar
Mac	junit4.11.jar, hamcrest-core-1.2.1.jar

Step 7: 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.

OS	Output
Windows	Set the environment variable JUNIT_HOME to C:\JUNIT
Linux	export JUNIT_HOME=/usr/local/JUNIT
Mac	export JUNIT_HOME=/Library/JUNIT

Step 8: 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.

OS	Output
Windows	Set the environment variable CLASSPATH to %CLASSPATH%;%JUNIT_HOME%\junit4.11.jar; %JUNIT_HOME%\hamcrest-core-1.2.1.jar;.;
Linux	export CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.11.jar:$JUNIT_HOME/hamcrest-core-1.2.1.jar:.
Mac	export CLASSPATH=$CLASSPATH:$JUNIT_HOME/junit4.11.jar:$JUNIT_HOME/hamcrest-core-1.2.1.jar:.