Guía paso a paso para probar una clase con Mockito en Java

Introducción

En esta guía, aprenderemos a utilizar **Mockito** para probar una clase en Java. Simularemos un servicio de precios de acciones (StockService) para probar una clase de **portafolio de inversiones** (Portfolio). En lugar de usar datos reales, crearemos un **mock** de StockService para devolver valores predefinidos y verificar si nuestra implementación funciona correctamente.

Paso 1: Crear la clase Stock

La clase Stock representará una acción con su identificador, nombre y cantidad.

Archivo: 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;
  }
}
```

Paso 2: Crear una interfaz StockService

Esta interfaz define un método para obtener el precio de una acción.

Archivo: StockService.java

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

Paso 3: Crear la clase Portfolio

La clase Portfolio representa el portafolio de un cliente. Utiliza StockService para calcular el valor total de las acciones.

Archivo: 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;
  }
}
```

Paso 4: Probar la clase Portfolio con Mockito

Usaremos **Mockito** para crear un **mock** de StockService, simulando precios ficticios y verificando si Portfolio calcula correctamente el valor total del mercado.

Archivo: PortfolioTester.java

```
package com.tutorialspoint.mock;

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(){
  // Crear un objeto Portfolio
  portfolio = new Portfolio();

  // Crear el mock del servicio de stock
  stockService = mock(StockService.class);

  // Asignar el mock a la cartera
  portfolio.setStockService(stockService);
}

public boolean testMarketValue(){
  // Crear lista de acciones
  List<Stock> stocks = new ArrayList<>();
  Stock googleStock = new Stock("1", "Google", 10);
  Stock microsoftStock = new Stock("2", "Microsoft", 100);

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

  // Agregar las acciones al portafolio
  portfolio.setStocks(stocks);
```

```
    // Simular los precios de las acciones
    when(stockService.getPrice(googleStock)).thenReturn(50.00);
    when(stockService.getPrice(microsoftStock)).thenReturn(1000.00);

    // Calcular el valor del mercado
    double marketValue = portfolio.getMarketValue();

    // Verificar si el valor calculado es correcto
    return marketValue == 100500.0;
  }
}
```

Paso 5: Compilar y ejecutar el código

Compilar los archivos

Ejecuta el siguiente comando en la terminal:

javac -cp .:mockito-all.jar Stock.java StockService.java Portfolio.java PortfolioTester.java

Asegúrate de incluir mockito-all.jar en el classpath.

Ejecutar la prueba

java -cp .:mockito-all.jar com.tutorialspoint.mock.PortfolioTester

Si todo funciona correctamente, verás en la salida:

pass

Guía paso a paso: Integración de JUnit y Mockito en Java

En este tutorial, aprenderemos a integrar **JUnit** y **Mockito** para probar una aplicación matemática. La aplicación utilizará un **servicio de calculadora** (CalculatorService) para realizar operaciones básicas como suma, resta, multiplicación y división.

Utilizaremos **Mockito** para simular (mock) el servicio de calculadora y probar la funcionalidad de la aplicación sin depender de una implementación real.

Paso 1: Crear la interfaz CalculatorService

Esta interfaz proporciona funciones matemáticas básicas.

Archivo: 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);
}
```

Paso 2: Crear la clase MathApplication

La clase MathApplication utilizará CalculatorService para realizar operaciones matemáticas.

Archivo: 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);
  }
```

}

Paso 3: Probar la clase MathApplication con Mockito

Utilizaremos **JUnit** para las pruebas unitarias y **Mockito** para simular el comportamiento de CalculatorService.

Archivo: 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 permite que JUnit use Mockito para las pruebas
@RunWith(MockitoJUnitRunner.class)
public class MathApplicationTester {

  // @InjectMocks crea e inyecta el mock en la instancia de MathApplication
  @InjectMocks
  MathApplication mathApplication = new MathApplication();

  // @Mock crea un mock de CalculatorService
  @Mock
  CalculatorService calcService;

  @Test
  public void testAdd(){
    // Simular el comportamiento de add
    when(calcService.add(10.0, 20.0)).thenReturn(30.00);
```

```
    // Verificar que el resultado es el esperado
    Assert.assertEquals(30.0, mathApplication.add(10.0, 20.0), 0);
  }
}
```

Explicación de las anotaciones utilizadas:

- @RunWith(MockitoJUnitRunner.class): Informa a JUnit que se usará Mockito para las pruebas.
- @Mock: Crea un objeto simulado de CalculatorService.
- @InjectMocks: Crea una instancia de MathApplication e inyecta el mock de CalculatorService.

Paso 4: Ejecutar las pruebas con TestRunner

Creamos una clase TestRunner que ejecutará las pruebas de MathApplicationTester.

Archivo: 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());
  }
}
```

Paso 5: Compilar y ejecutar las pruebas

Compilar los archivos

Ejecuta el siguiente comando en la terminal:

javac -cp .:junit-4.12.jar:mockito-all.jar CalculatorService.java MathApplication.java MathApplicationTester.java TestRunner.java

Asegúrate de incluir los archivos junit-4.12.jar y mockito-all.jar en el classpath.

Ejecutar las pruebas

java -cp .:junit-4.12.jar:mockito-all.jar TestRunner