**Exercise 1: Mocking and Stubbing**

**Code:**

package week2.module4\_3;

import static org.mockito.Mockito.\*;

import static org.junit.jupiter.api.Assertions.\*;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension;

class ExternalApi {

String getData() {

return "Data";

}

}

class Service {

ExternalApi externalApi;

Service(ExternalApi externalApi) {

this.externalApi = externalApi;

}

String service() {

return "Hi " + externalApi.getData();

}

}

*@ExtendWith*(MockitoExtension.class)

public class Ex1ServiceTest {

*@Mock*

public ExternalApi mockExternalApi;

*@Test*

void testExternalApi() {

*when*(mockExternalApi.getData()).thenReturn("Mock Data");

Service service = new Service(mockExternalApi);

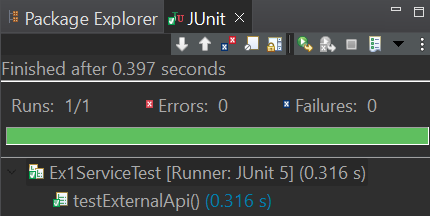
String result = service.service();

*assertEquals*("Hi Mock Data", result);

}

}

**Output:**



**Exercise 2: Verifying Interactions**

**Code:**

package week2.module4\_3;

import static org.junit.jupiter.api.Assertions.*assertEquals*;

import static org.mockito.Mockito.*verify*;

import static org.mockito.Mockito.*when*;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.junit.jupiter.MockitoExtension;

import org.mockito.Mock;

class ExternalApi1{

String getData(String parameter) {

return "Data";

}

}

class Service1{

ExternalApi1 externalApi;

Service1(ExternalApi1 externalApi){

this.externalApi=externalApi;

}

String service(String parameter) {

return "Hi "+externalApi.getData(parameter);

}

}

*@ExtendWith*(MockitoExtension.class)

public class Ex2VerifyingInteractions {

*@Mock*

ExternalApi1 externalApi;

*@Test*

void testService() {

*when*(externalApi.getData("Hello")).thenReturn("Mock Data");

Service1 service=new Service1(externalApi);

String result=service.service("Hello");

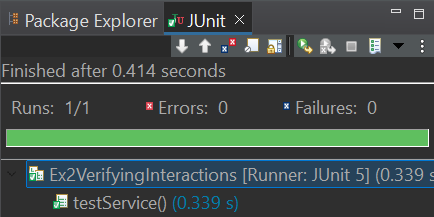
*verify*(externalApi).getData("Hello");

*assertEquals*("Hi Mock Data",result);

}

}

**Output:**



**Exercise 3: Argument Matching**

**Code:**

package week2.module4\_3;

import static org.junit.jupiter.api.Assertions.*assertEquals*;

import static org.mockito.Mockito.*verify*;

import static org.mockito.Mockito.*when*;

import static org.mockito.ArgumentMatchers.\*;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.junit.jupiter.MockitoExtension;

import org.mockito.Mock;

class ExternalApi2{

String getData(String parameter) {

return "Data";

}

}

class Service2{

ExternalApi2 externalApi;

Service2(ExternalApi2 externalApi){

this.externalApi=externalApi;

}

String service(String parameter) {

return "Hi "+externalApi.getData(parameter);

}

}

*@ExtendWith*(MockitoExtension.class)

public class Ex3ArgumentMatching {

*@Mock*

ExternalApi2 externalApi;

*@Test*

void testService() {

*when*(externalApi.getData(*anyString*())).thenReturn("Mock Data");

Service2 service=new Service2(externalApi);

String result=service.service("Hello");

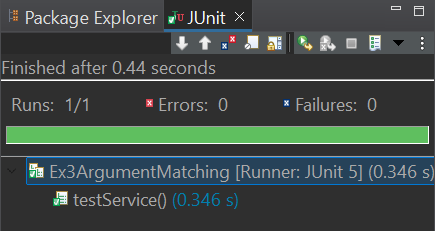
*verify*(externalApi).getData("Hello");

*assertEquals*("Hi Mock Data",result);

}

}

**Output:**



**Exercise 4: Handling Void Methods**

**Code:**

package week2.module4\_3;

import static org.mockito.ArgumentMatchers.*anyString*;

import static org.mockito.ArgumentMatchers.*eq*;

import static org.mockito.Mockito.*doNothing*;

import static org.mockito.Mockito.*verify*;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension;

class Notifier{

void notify(String to) {

System.***out***.println("Sending email to "+to);

}

}

class UserService{

private Notifier notifier;

public UserService(Notifier notifier) {

this.notifier=notifier;

}

public void sendEmail(String to) {

notifier.notify(to);

}

}

*@ExtendWith*(MockitoExtension.class)

public class Ex4HandlingVoidMethods {

*@Mock*

private Notifier notifier;

*@Test*

public void testSendEmail() {

UserService userService=new UserService(notifier);

*doNothing*().when(notifier).notify(*anyString*());

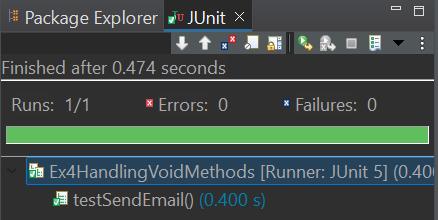
userService.sendEmail("User2@gmail.com");

*verify*(notifier).notify("User2@gmail.com");

}

}

**Output:**



**Exercise 5: Mocking and Stubbing with Multiple Returns**

**Code:**

package week2.module4\_3;

import static org.junit.jupiter.api.Assertions.*assertTrue*;

import static org.mockito.Mockito.*times*;

import static org.mockito.Mockito.*verify*;

import static org.mockito.Mockito.*when*;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.Mock;

import org.mockito.Mockito;

import org.mockito.junit.jupiter.MockitoExtension;

class ExternalApi3{

String getStatus() {

return "Some Status";

}

}

class StatusService{

private ExternalApi3 externalApi;

public StatusService(ExternalApi3 externalApi) {

this.externalApi=externalApi;

}

public boolean waitForCompletion() {

for(int i=0;i<3;i++) {

if("COMPLETED".equalsIgnoreCase(externalApi.getStatus())) {

return true;

}

}

return false;

}

}

*@ExtendWith*(MockitoExtension.class)

public class Ex5MultipleReturnTest {

*@Mock*

ExternalApi3 externalApi;

*@Test*

public void testMethod() {

*when*(externalApi.getStatus()).thenReturn("INITIAL").thenReturn("PROCESSING").thenReturn("COMPLETED");

StatusService statusService=new StatusService(externalApi);

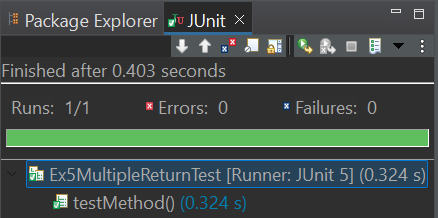
*assertTrue*(statusService.waitForCompletion());

*verify*(externalApi,*times*(3)).getStatus();

}

}

**Output:**



**Exercise 6: Verifying Interaction Order**

**Code:**

package week2.module4\_3;

import static org.mockito.Mockito.*inOrder*;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.InOrder;

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension;

class Printer{

void initiate() {

System.***out***.println("Initiated..");

}

void print() {

System.***out***.println("Printing..");

}

void complete() {

System.***out***.println("Completed");

}

}

class PrinterProcessor{

private Printer printer;

public PrinterProcessor(Printer printer) {

this.printer=printer;

}

void processPrint() {

printer.initiate();

printer.print();

printer.complete();

}

}

*@ExtendWith*(MockitoExtension.class)

public class Ex6VerifyingInteractionOrder {

*@Mock*

Printer printer;

*@Test*

void testProcessPrint() {

PrinterProcessor printerProcessor=new PrinterProcessor(printer);

printerProcessor.processPrint();

InOrder inOrder=*inOrder*(printer);

inOrder.verify(printer).initiate();

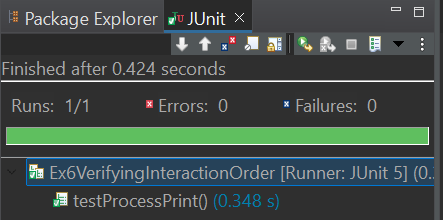
inOrder.verify(printer).print();

inOrder.verify(printer).complete();

}

}

**Output:**



**Exercise 7: Handling Void Methods with Exceptions**

**Code:**

package week2.module4\_3;

import static org.junit.jupiter.api.Assertions.*assertThrows*;

import static org.mockito.Mockito.*doThrow*;

import static org.mockito.Mockito.*verify*;

import org.junit.jupiter.api.Test;

import org.junit.jupiter.api.extension.ExtendWith;

import org.mockito.Mock;

import org.mockito.junit.jupiter.MockitoExtension;

class Printer1{

void print() {

System.***out***.println("Printing..");

}

}

class PrinterProcessor1{

private Printer1 printer;

public PrinterProcessor1(Printer1 printer) {

this.printer=printer;

}

void processPrint() {

printer.print();

}

}

*@ExtendWith*(MockitoExtension.class)

public class Ex7VoidMethodWithException {

*@Mock*

Printer1 printer;

*@Test*

void testProcessPrint() {

*doThrow*(new RuntimeException("Printer Not Working")).when(printer).print();

PrinterProcessor1 printerProcessor=new PrinterProcessor1(printer);

*assertThrows*(RuntimeException.class,()->{

printerProcessor.processPrint();

});

*verify*(printer).print();

}

}

**Output:**

