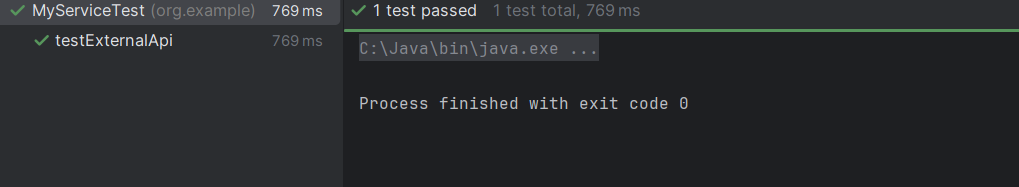
Exercise 1: Mocking and Stubbing  
@MyService  
package org.example;  
  
public class MyService {  
  
 private ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchData() {  
 return api.getData();  
 }  
  
 public void process(String input) {  
 api.performAction(input);  
 }  
  
 public void riskyProcess() throws Exception {  
 api.riskyAction();  
 }  
}

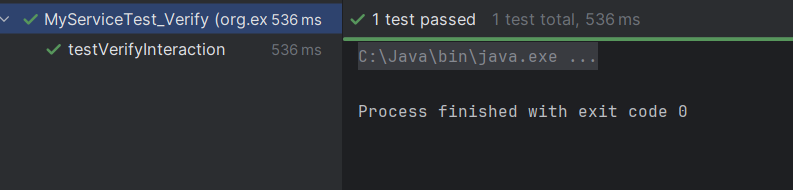
@ExternalApi  
package org.example;  
  
public interface ExternalApi {  
 String getData();  
 void performAction(String input);  
 void riskyAction() throws Exception;  
}



Exercise 2: Verifying Interactions

@class

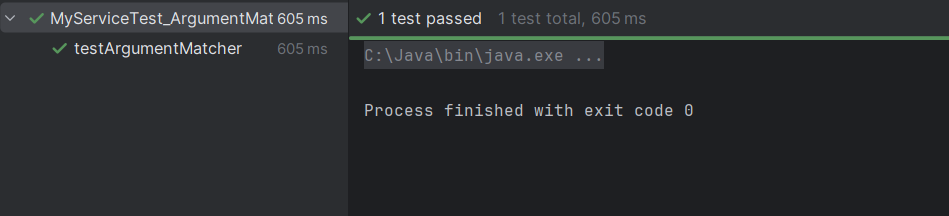
package org.example;  
  
import org.junit.Test;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceTest\_Verify {  
  
 @Test  
 public void testVerifyInteraction() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
 MyService service = new MyService(mockApi);  
 service.fetchData();  
  
 *verify*(mockApi).getData();  
 }  
}



Exercise 3: Argument Matching

@class

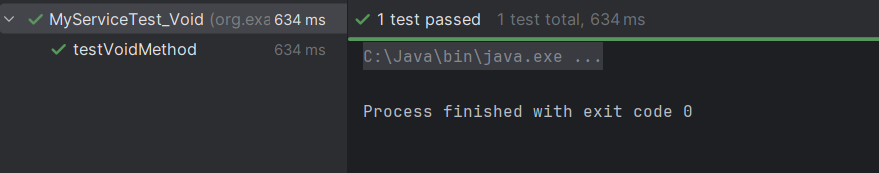
package org.example;  
  
import org.junit.Test;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceTest\_ArgumentMatcher {  
  
 @Test  
 public void testArgumentMatcher() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
  
 MyService service = new MyService(mockApi);  
 service.process("Hello");  
  
 *verify*(mockApi).performAction(*eq*("Hello"));  
 }  
}



Exercise 4: Handling Void Methods

@class

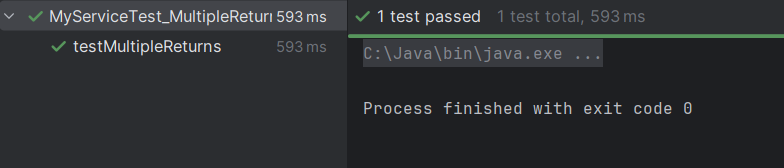
package org.example;  
  
import org.junit.Test;  
import static org.mockito.Mockito.\*;  
  
public class MyServiceTest\_Void {  
  
 @Test  
 public void testVoidMethod() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 *doNothing*().when(mockApi).performAction(*anyString*());  
  
 MyService service = new MyService(mockApi);  
 service.process("Test");  
  
 *verify*(mockApi).performAction("Test");  
 }  
}



Exercise 5: Mocking and Stubbing with Multiple Returns

@class

package org.example;  
  
import org.junit.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.Assert.\*;  
  
public class MyServiceTest\_MultipleReturns {  
  
 @Test  
 public void testMultipleReturns() {  
 ExternalApi mockApi = *mock*(ExternalApi.class);  
 *when*(mockApi.getData())  
 .thenReturn("First Call")  
 .thenReturn("Second Call");  
  
 MyService service = new MyService(mockApi);  
  
 *assertEquals*("First Call", service.fetchData());  
 *assertEquals*("Second Call", service.fetchData());  
 }  
}



Exercise 6: Verifying Interaction Order

@class

package org.example;

import org.junit.Test;

import static org.mockito.Mockito.\*;

import org.mockito.InOrder;

public class MyServiceTest\_InteractionOrder {

@Test

public void testInteractionOrder() {

ExternalApi mockApi = mock(ExternalApi.class);

MyService service = new MyService(mockApi);

service.process("First");

service.fetchData();

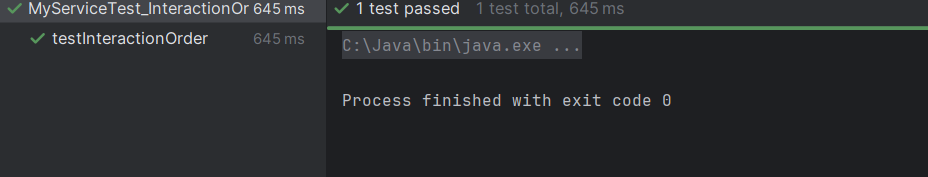
InOrder inOrder = inOrder(mockApi);

inOrder.verify(mockApi).performAction("First");

inOrder.verify(mockApi).getData();

}

}



Exercise 7: Handling Void Methods with Exceptions

@calss

package org.example;

import org.junit.Test;

import static org.mockito.Mockito.\*;

public class MyServiceTest\_VoidException {

@Test(expected = RuntimeException.class)

public void testVoidMethodWithException() throws Exception {

ExternalApi mockApi = mock(ExternalApi.class);

doThrow(new RuntimeException("Failure")).when(mockApi).riskyAction();

MyService service = new MyService(mockApi);

service.riskyProcess();

}

}

