**MOCKITO HANDS-ON EXERCISES**

**Exercise 1: Mocking and Stubbing**

**ExternalApi.java**

package mockito.exercises;

public interface ExternalApi {

String getData();

}

**MyService.java**

package mockito.exercises;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

}

**MyServiceTest.java**

package mockito.exercises;

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

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

*@Test*

public void testExternalApi() {

ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);

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

MyService service = new MyService(mockApi);

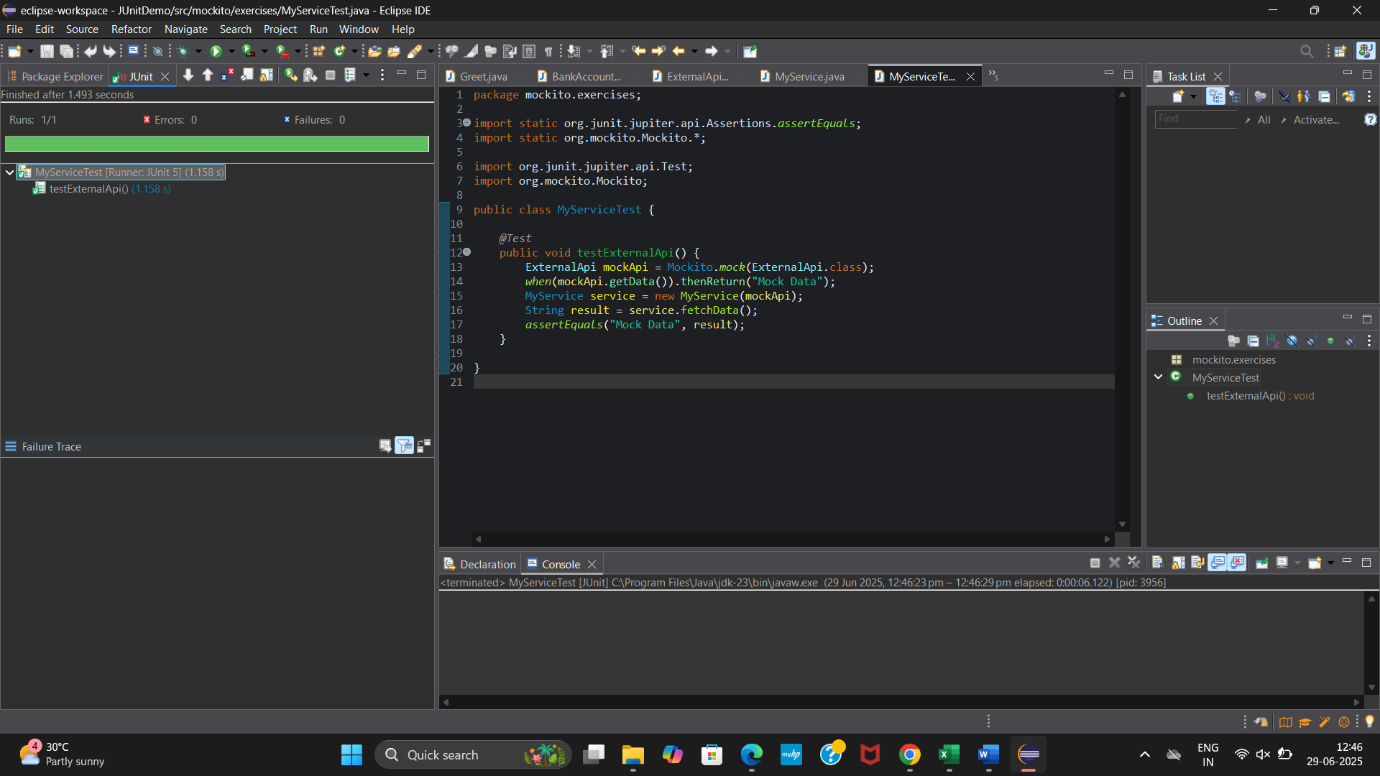
String result = service.fetchData();

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

}

}

**OUTPUT**

****

**Exercise 2: Verifying Interactions**

**MyServiceTest.java**

package mockito.exercises;

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

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

*@Test*

public void testExternalApi() {

ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);

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

MyService service = new MyService(mockApi);

String result = service.fetchData();

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

}

*@Test*

public void testVerifyInteraction() {

ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);

MyService service = new MyService(mockApi);

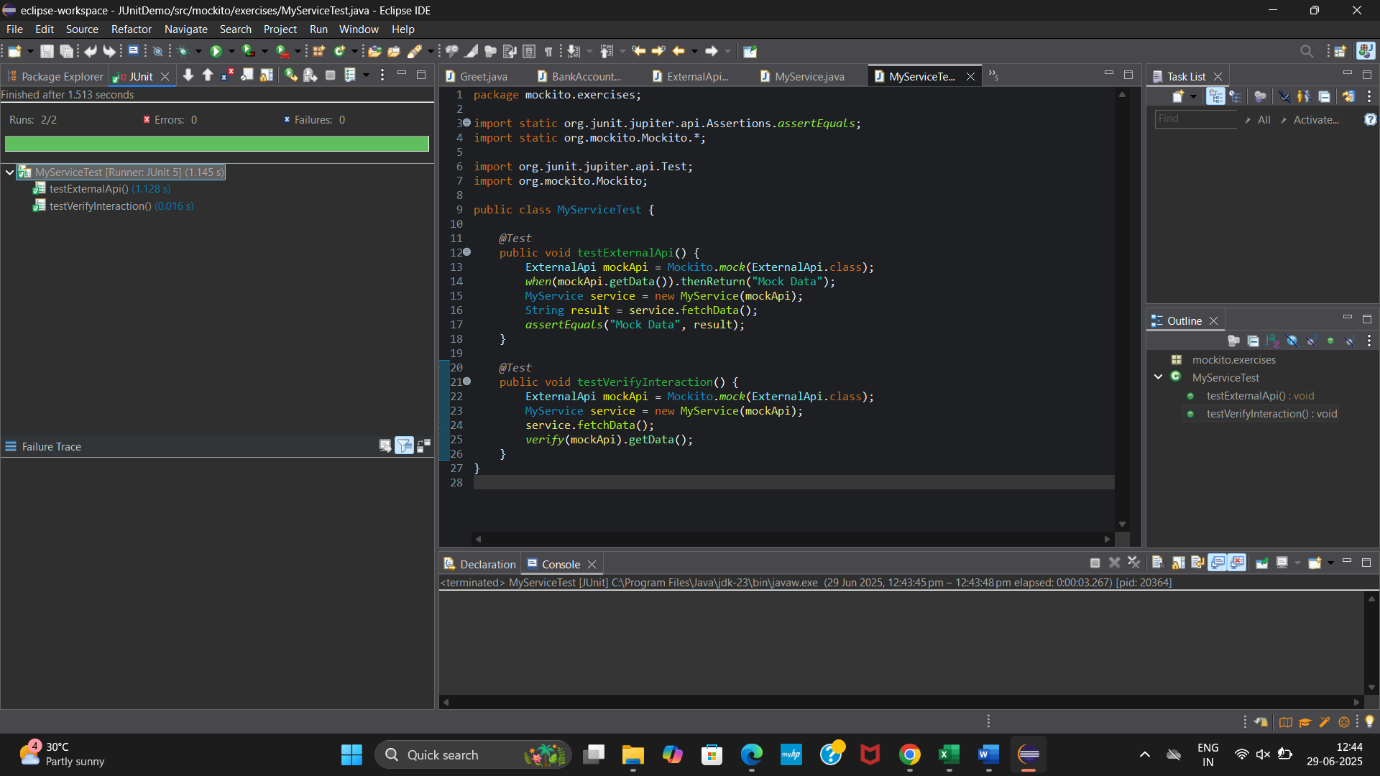
service.fetchData();

*verify*(mockApi).getData();

}

}

**OUTPUT**

****

**Exercise 3: Argument Matching**

**MessageSender.java**

package mockito.exercises;

public interface MessageSender {

void sendMessage(String user, String message);

}

**MessageService.java**

package mockito.exercises;

public class MessageService {

private MessageSender sender;

public MessageService(MessageSender sender) {

this.sender = sender;

}

public void notifyUser() {

sender.sendMessage("Admin", "Welcome!");

}

}

**MessageServiceTest.java**

package mockito.exercises;

import static org.mockito.Mockito.\*;

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

import org.junit.jupiter.api.Test;

public class MessageServiceTest {

*@Test*

public void testSendMessageWithArgumentMatchers() {

MessageSender mockSender = *mock*(MessageSender.class);

MessageService service = new MessageService(mockSender);

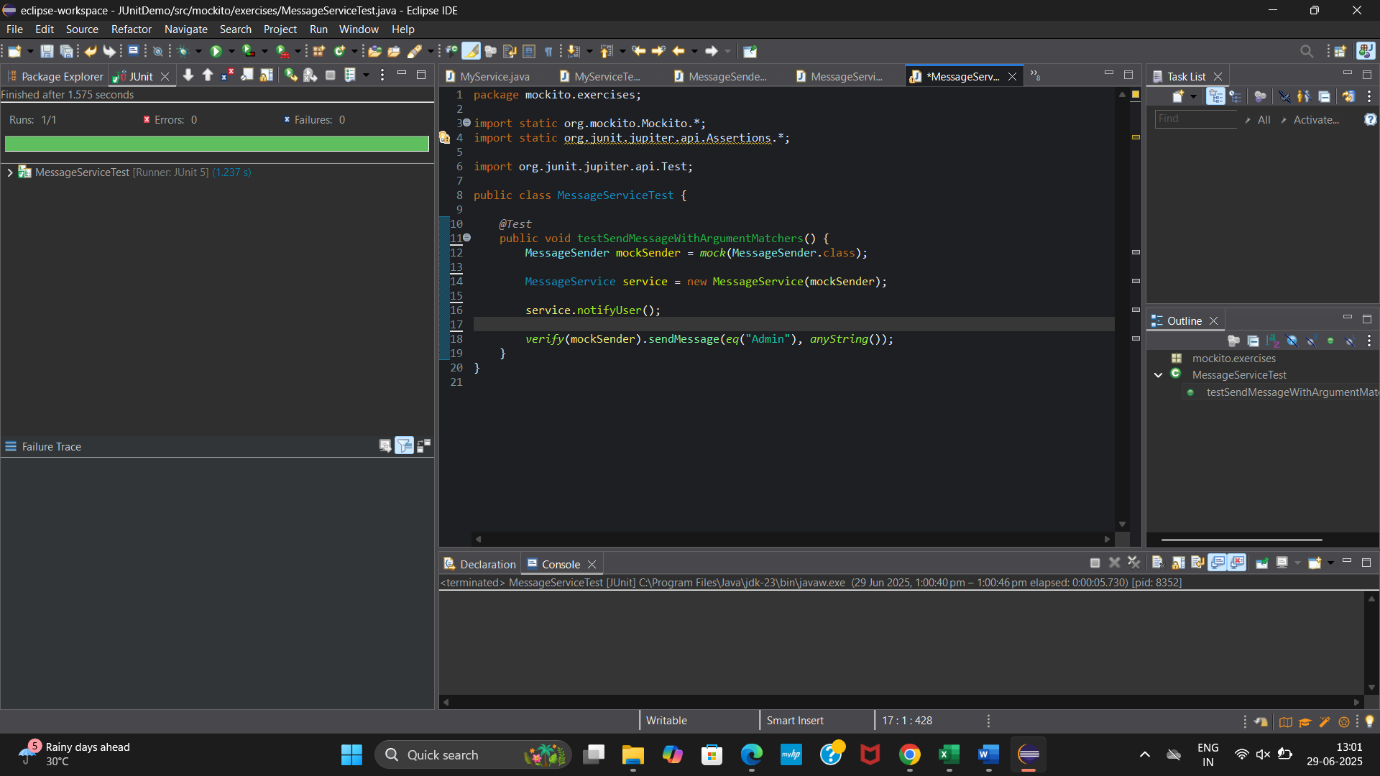
service.notifyUser();

*verify*(mockSender).sendMessage(*eq*("Admin"), *anyString*());

}

}

**OUTPUT**

****

**Exercise 4: Handling Void Methods**

**Notifier.java**

package mockito.exercises;

public interface Notifier {

void send(String message);

}

**NotifierTest.java**

package mockito.exercises;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class NotifierTest {

*@Test*

public void testVoidMethod() {

Notifier mockNotifier = *mock*(Notifier.class);

*doNothing*().when(mockNotifier).send(*anyString*());

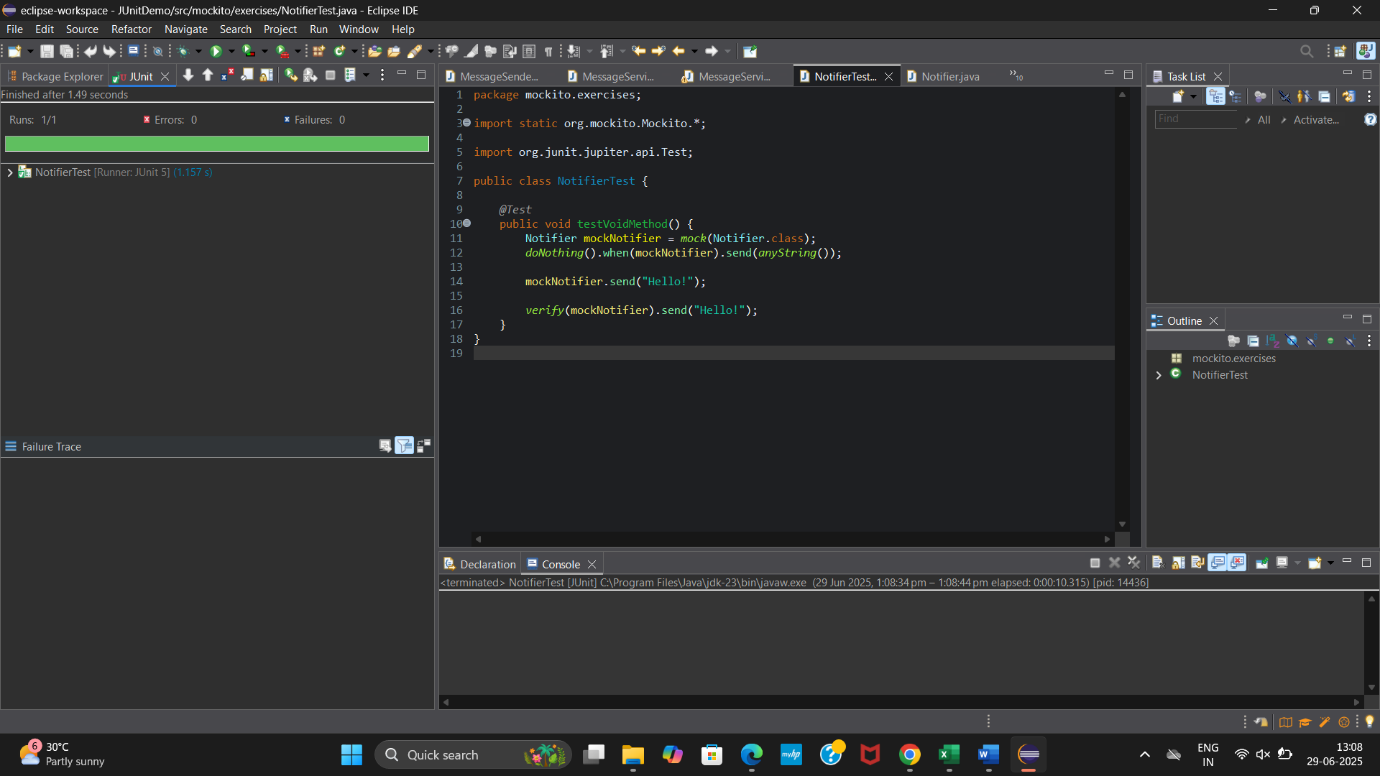
mockNotifier.send("Hello!");

*verify*(mockNotifier).send("Hello!");

}

}

**OUTPUT**

****

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

**ExternalApi.java**

package mockito.exercises;

public interface ExternalApi {

String getData();

String getStatus();

}

**MyService.java**

package mockito.exercises;

public class MyService {

private ExternalApi api;

public MyService(ExternalApi api) {

this.api = api;

}

public String fetchData() {

return api.getData();

}

public String checkStatus() {

return api.getStatus();

}

}

**MyServiceTest.java**

package mockito.exercises;

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

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.Mockito;

public class MyServiceTest {

*@Test*

public void testExternalApi() {

ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);

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

MyService service = new MyService(mockApi);

String result = service.fetchData();

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

}

*@Test*

public void testVerifyInteraction() {

ExternalApi mockApi = Mockito.*mock*(ExternalApi.class);

MyService service = new MyService(mockApi);

service.fetchData();

*verify*(mockApi).getData();

}

*@Test*

public void testMultipleReturns() {

ExternalApi mockApi = *mock*(ExternalApi.class);

*when*(mockApi.getStatus()).thenReturn("Pending", "Processing", "Done");

MyService service = new MyService(mockApi);

*assertEquals*("Pending", service.checkStatus());

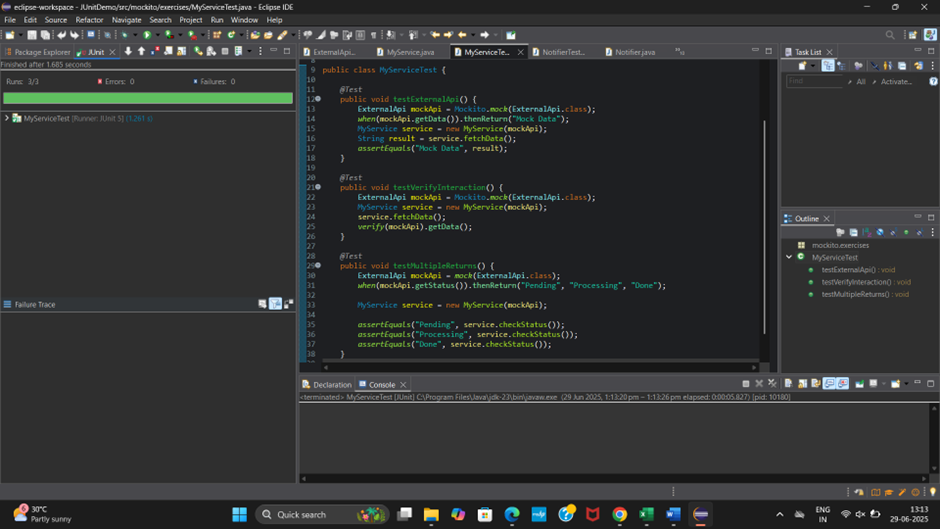
*assertEquals*("Processing", service.checkStatus());

*assertEquals*("Done", service.checkStatus());

}

}

**OUTPUT**

****

**Exercise 6: Verifying Interaction Order**

**UserManager.java**

package mockito.exercises;

public interface UserManager {

void login();

void fetchData();

void logout();

}

**UserManagerTest.java**

package mockito.exercises;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

import org.mockito.InOrder;

public class UserManagerTest {

*@Test*

public void testInteractionOrder() {

UserManager mockManager = *mock*(UserManager.class);

mockManager.login();

mockManager.fetchData();

mockManager.logout();

InOrder inOrder = *inOrder*(mockManager);

inOrder.verify(mockManager).login();

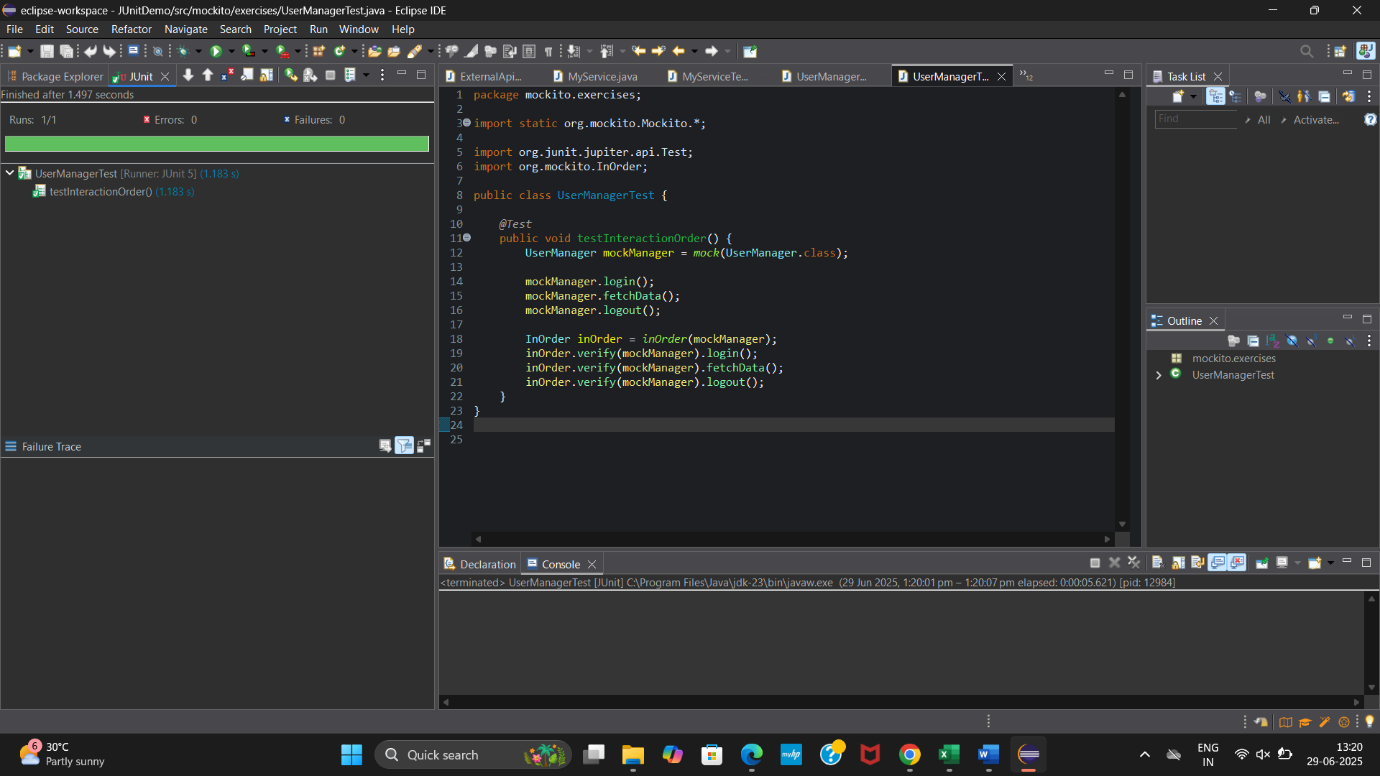
inOrder.verify(mockManager).fetchData();

inOrder.verify(mockManager).logout();

}

}

**OUTPUT**

****

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

**Logger.java**

package mockito.exercises;

public interface Logger {

void log(String message) throws Exception;

}

**LoggerTest.java**

package mockito.exercises;

import static org.mockito.Mockito.\*;

import org.junit.jupiter.api.Test;

public class LoggerTest {

*@Test*

public void testVoidMethodWithException() throws Exception {

Logger mockLogger = *mock*(Logger.class);

*doThrow*(new RuntimeException("Log failed")).when(mockLogger).log("ERROR");

try {

mockLogger.log("ERROR");

} catch (RuntimeException ex) {

assert(ex.getMessage().equals("Log failed"));

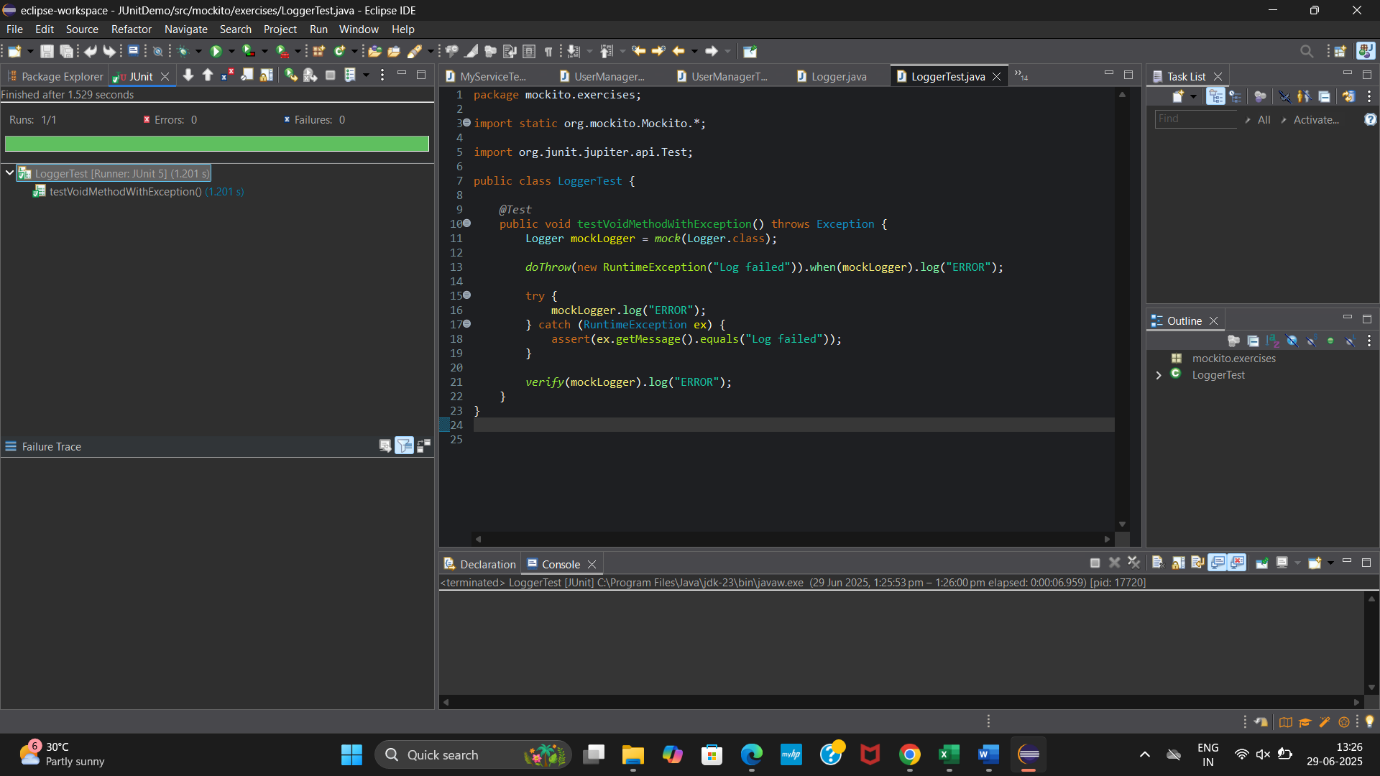
}

*verify*(mockLogger).log("ERROR");

}

}

**OUTPUT**

****