Name: INIGASHREE N S

College: Kongu Engineering College

**EXERCISE 1: MOCKING AND STUBBING**

This project demonstrates the use of **mocking and stubbing** in unit testing with **Mockito**, where an ExternalApi interface is mocked to simulate various behaviors such as data retrieval, connection status, login validation, and rate fetching. The goal is to isolate the logic of MyService and test its functionality without invoking real external dependencies.

**Objective:**

* **Mock External Dependencies**  
  Replace the real ExternalApi with a mock using Mockito to isolate the service logic.
* **Stub Return Values**  
  Simulate responses for methods like getData(), getUser(id), isConnected(), and login() using when(...).thenReturn(...).
* **Validate Service Logic**  
  Ensure MyService correctly uses the mocked API and returns expected results in different scenarios.
* **Verify Method Calls**  
  Confirm interactions like sendData(...) are properly invoked using verify(...).

**Code & Output:**

**ExternalApi.java**

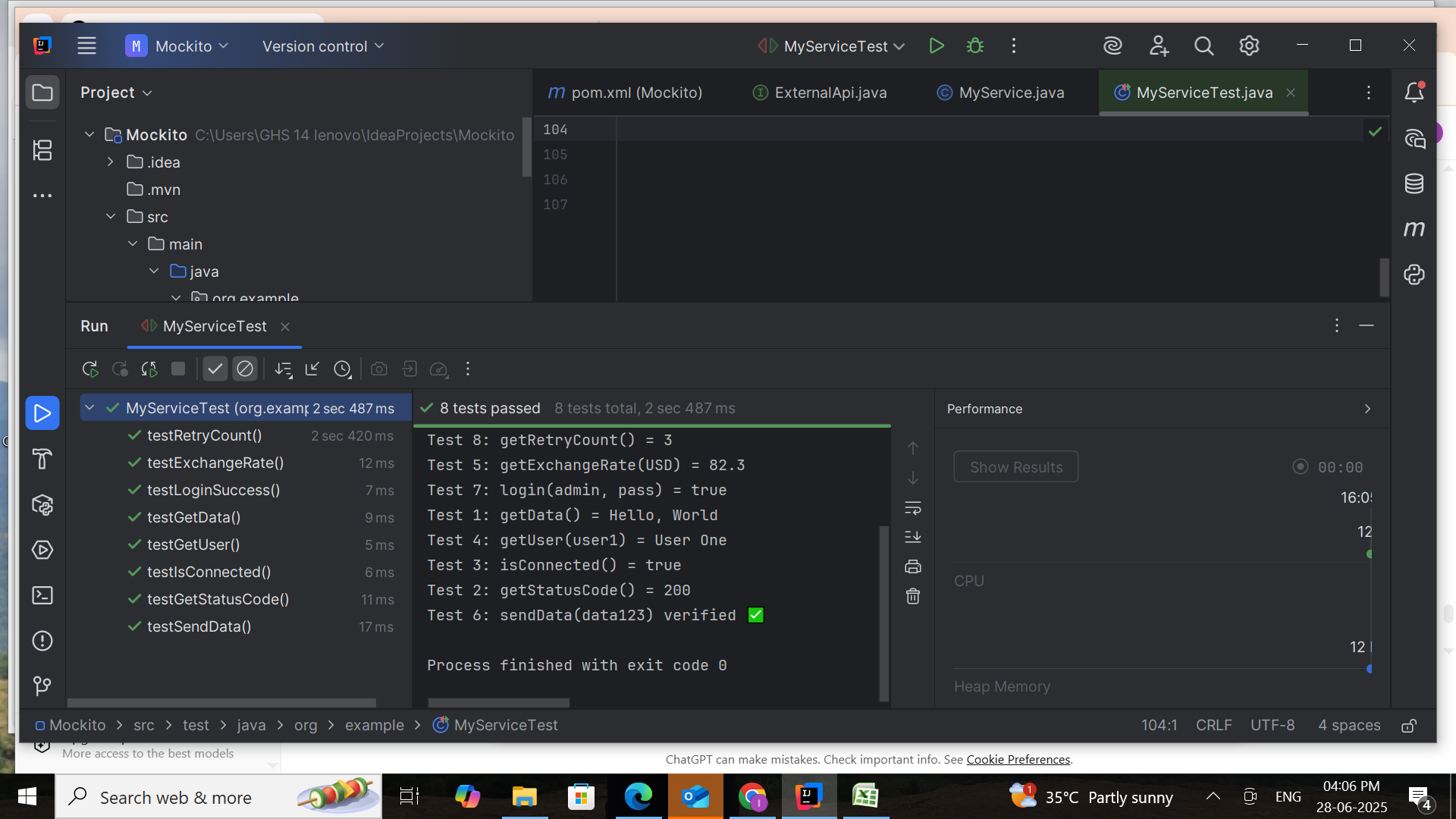
package org.example;  
  
public interface ExternalApi {  
 String getData();  
 int getStatusCode();  
 boolean isConnected();  
 String getUser(String userId);  
 double getExchangeRate(String currency);  
 void sendData(String payload);  
 boolean login(String username, String password);  
 int getRetryCount();  
}

**MyService.java**

package org.example;  
  
public class MyService {  
 private final ExternalApi api;  
  
 public MyService(ExternalApi api) {  
 this.api = api;  
 }  
  
 public String fetchData() {  
 return api.getData();  
 }  
  
 public int fetchStatusCode() {  
 return api.getStatusCode();  
 }  
  
 public boolean isServerConnected() {  
 return api.isConnected();  
 }  
  
 public String getUserDetails(String id) {  
 return api.getUser(id);  
 }  
  
 public double fetchRate(String currency) {  
 return api.getExchangeRate(currency);  
 }  
  
 public void submitPayload(String data) {  
 api.sendData(data);  
 }  
  
 public boolean validateLogin(String user, String pass) {  
 return api.login(user, pass);  
 }  
  
 public int getRetries() {  
 return api.getRetryCount();  
 }  
}

**MyServiceTest.java**

package org.example;  
  
import org.junit.jupiter.api.Test;  
import static org.mockito.Mockito.\*;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class MyServiceTest {  
  
 @Test  
 public void testGetData() {  
 ExternalApi api = *mock*(ExternalApi.class);  
 *when*(api.getData()).thenReturn("Hello, World");  
  
 MyService service = new MyService(api);  
 String result = service.fetchData();  
  
 System.*out*.println("Test 1: getData() = " + result);  
 *assertEquals*("Hello, World", result);  
 }  
  
 @Test  
 public void testGetStatusCode() {  
 ExternalApi api = *mock*(ExternalApi.class);  
 *when*(api.getStatusCode()).thenReturn(200);  
  
 MyService service = new MyService(api);  
 int code = service.fetchStatusCode();  
  
 System.*out*.println("Test 2: getStatusCode() = " + code);  
 *assertEquals*(200, code);  
 }  
  
 @Test  
 public void testIsConnected() {  
 ExternalApi api = *mock*(ExternalApi.class);  
 *when*(api.isConnected()).thenReturn(true);  
  
 MyService service = new MyService(api);  
 boolean status = service.isServerConnected();  
  
 System.*out*.println("Test 3: isConnected() = " + status);  
 *assertTrue*(status);  
 }  
  
 @Test  
 public void testGetUser() {  
 ExternalApi api = *mock*(ExternalApi.class);  
 *when*(api.getUser("user1")).thenReturn("User One");  
  
 MyService service = new MyService(api);  
 String user = service.getUserDetails("user1");  
  
 System.*out*.println("Test 4: getUser(user1) = " + user);  
 *assertEquals*("User One", user);  
 }  
  
 @Test  
 public void testExchangeRate() {  
 ExternalApi api = *mock*(ExternalApi.class);  
 *when*(api.getExchangeRate("USD")).thenReturn(82.3);  
  
 MyService service = new MyService(api);  
 double rate = service.fetchRate("USD");  
  
 System.*out*.println("Test 5: getExchangeRate(USD) = " + rate);  
 *assertEquals*(82.3, rate);  
 }  
  
 @Test  
 public void testSendData() {  
 ExternalApi api = *mock*(ExternalApi.class);  
 MyService service = new MyService(api);  
  
 service.submitPayload("data123");  
 *verify*(api).sendData("data123");  
  
 System.*out*.println("Test 6: sendData(data123) verified ✅");  
 }  
  
 @Test  
 public void testLoginSuccess() {  
 ExternalApi api = *mock*(ExternalApi.class);  
 *when*(api.login("admin", "pass")).thenReturn(true);  
  
 MyService service = new MyService(api);  
 boolean result = service.validateLogin("admin", "pass");  
  
 System.*out*.println("Test 7: login(admin, pass) = " + result);  
 *assertTrue*(result);  
 }  
  
 @Test  
 public void testRetryCount() {  
 ExternalApi api = *mock*(ExternalApi.class);  
 *when*(api.getRetryCount()).thenReturn(3);  
  
 MyService service = new MyService(api);  
 int retries = service.getRetries();  
  
 System.*out*.println("Test 8: getRetryCount() = " + retries);  
 *assertEquals*(3, retries);  
 }  
}

****

The test successfully verifies MyService functionality using mocked ExternalApi data, ensuring correctness without relying on external services. This approach enhances test reliability, speed, and control, making it ideal for isolated unit testing.

**EXERCISE 2: VERIFYING INTERACTIONS**

This test illustrates the use of Mockito’s interaction verification to ensure that the VeggieService correctly interacts with its dependency, VeggieStoreApi, by calling the addToCart() method with the correct item and quantity. It also verifies that the checkout() method is invoked during order completion, confirming proper service behavior.

**Objective:**

* **Mock Dependencies**  
  Replace the real VeggieStoreApi with a mock to isolate the VeggieService from actual store operations.
* **Trigger Action**  
  Call the buyItem() and completeOrder() methods to simulate adding vegetables to the cart and placing an order.
* **Verify Interaction**  
  Use verify() to confirm that:
* addToCart("Tomato", 2) was called with the correct item and quantity.
* checkout() was invoked exactly once.

**Code & Output:**

**VeggieStoreApi.java**

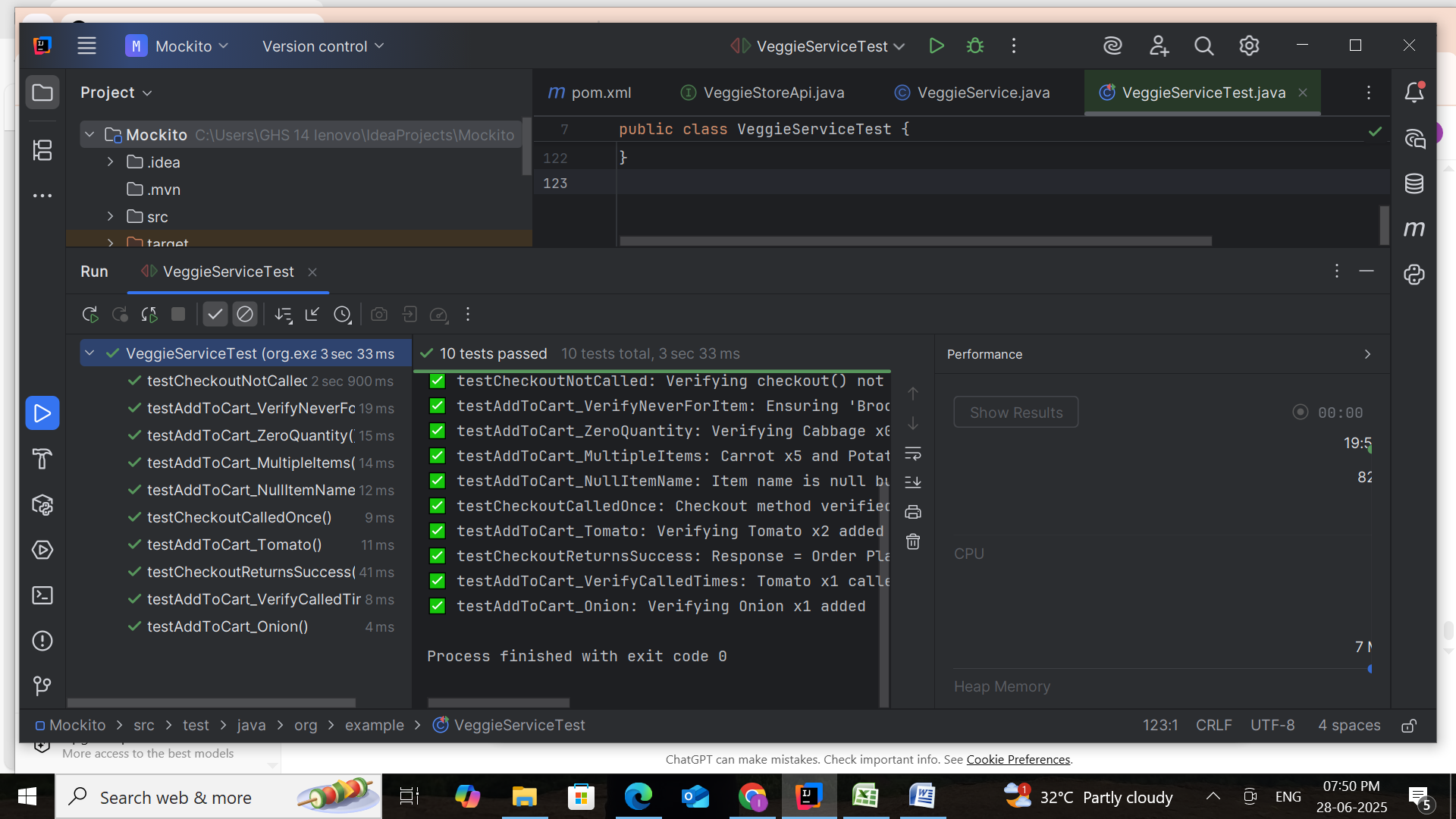
package org.example;  
  
public interface VeggieStoreApi {  
 void addToCart(String itemName, int quantity);  
 String checkout();  
}

**VeggieService.java**

package org.example;  
  
public class VeggieService {  
 private final VeggieStoreApi api;  
  
 public VeggieService(VeggieStoreApi api) {  
 this.api = api;  
 }  
  
 public void buyItem(String item, int qty) {  
 api.addToCart(item, qty);  
 }  
  
 public String completeOrder() {  
 return api.checkout();  
 }  
}

**VeggieServiceTest.java**

package org.example;  
  
import static org.mockito.Mockito.\*;  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
public class VeggieServiceTest {  
  
 @Test  
 public void testAddToCart\_Tomato() {  
 VeggieStoreApi mockApi = *mock*(VeggieStoreApi.class);  
 VeggieService service = new VeggieService(mockApi);  
  
 service.buyItem("Tomato", 2);  
  
 System.*out*.println("✅ testAddToCart\_Tomato: Verifying Tomato x2 added");  
 *verify*(mockApi).addToCart("Tomato", 2);  
 }  
  
 @Test  
 public void testAddToCart\_Onion() {  
 VeggieStoreApi mockApi = *mock*(VeggieStoreApi.class);  
 VeggieService service = new VeggieService(mockApi);  
  
 service.buyItem("Onion", 1);  
  
 System.*out*.println("✅ testAddToCart\_Onion: Verifying Onion x1 added");  
 *verify*(mockApi).addToCart("Onion", 1);  
 }  
  
 @Test  
 public void testAddToCart\_MultipleItems() {  
 VeggieStoreApi mockApi = *mock*(VeggieStoreApi.class);  
 VeggieService service = new VeggieService(mockApi);  
  
 service.buyItem("Carrot", 5);  
 service.buyItem("Potato", 3);  
  
 System.*out*.println("✅ testAddToCart\_MultipleItems: Carrot x5 and Potato x3 added");  
 *verify*(mockApi).addToCart("Carrot", 5);  
 *verify*(mockApi).addToCart("Potato", 3);  
 }  
  
 @Test  
 public void testAddToCart\_ZeroQuantity() {  
 VeggieStoreApi mockApi = *mock*(VeggieStoreApi.class);  
 VeggieService service = new VeggieService(mockApi);  
  
 service.buyItem("Cabbage", 0);  
  
 System.*out*.println("✅ testAddToCart\_ZeroQuantity: Verifying Cabbage x0 added");  
 *verify*(mockApi).addToCart("Cabbage", 0);  
 }  
  
 @Test  
 public void testCheckoutCalledOnce() {  
 VeggieStoreApi mockApi = *mock*(VeggieStoreApi.class);  
 VeggieService service = new VeggieService(mockApi);  
  
 service.completeOrder();  
  
 System.*out*.println("✅ testCheckoutCalledOnce: Checkout method verified");  
 *verify*(mockApi, *times*(1)).checkout();  
 }  
  
 @Test  
 public void testCheckoutReturnsSuccess() {  
 VeggieStoreApi mockApi = *mock*(VeggieStoreApi.class);  
 *when*(mockApi.checkout()).thenReturn("Order Placed Successfully");  
  
 VeggieService service = new VeggieService(mockApi);  
 String response = service.completeOrder();  
  
 System.*out*.println("✅ testCheckoutReturnsSuccess: Response = " + response);  
 *assertEquals*("Order Placed Successfully", response);  
 }  
  
 @Test  
 public void testAddToCart\_VerifyNeverForItem() {  
 VeggieStoreApi mockApi = *mock*(VeggieStoreApi.class);  
 VeggieService service = new VeggieService(mockApi);  
  
 service.buyItem("Spinach", 3);  
  
 System.*out*.println("✅ testAddToCart\_VerifyNeverForItem: Ensuring 'Broccoli' was never added");  
 *verify*(mockApi, *never*()).addToCart("Broccoli", 1);  
 }  
  
 @Test  
 public void testAddToCart\_VerifyCalledTimes() {  
 VeggieStoreApi mockApi = *mock*(VeggieStoreApi.class);  
 VeggieService service = new VeggieService(mockApi);  
  
 service.buyItem("Tomato", 1);  
 service.buyItem("Tomato", 1);  
  
 System.*out*.println("✅ testAddToCart\_VerifyCalledTimes: Tomato x1 called twice");  
 *verify*(mockApi, *times*(2)).addToCart("Tomato", 1);  
 }  
  
 @Test  
 public void testAddToCart\_NullItemName() {  
 VeggieStoreApi mockApi = *mock*(VeggieStoreApi.class);  
 VeggieService service = new VeggieService(mockApi);  
  
 service.buyItem(null, 2);  
  
 System.*out*.println("✅ testAddToCart\_NullItemName: Item name is null but called");  
 *verify*(mockApi).addToCart(null, 2);  
 }  
  
 @Test  
 public void testCheckoutNotCalled() {  
 VeggieStoreApi mockApi = *mock*(VeggieStoreApi.class);  
 VeggieService service = new VeggieService(mockApi);  
  
 service.buyItem("Capsicum", 2);  
  
 System.*out*.println("✅ testCheckoutNotCalled: Verifying checkout() not triggered yet");  
 *verify*(mockApi, *never*()).checkout();  
 }  
}

****

The test successfully validates that the VeggieService interacts correctly with the VeggieStoreApi, ensuring that the right items and quantities are added to the cart and that checkout is triggered as expected. This interaction-based testing reinforces confidence in the correctness of service-to-dependency communication within the shopping workflow.