MOCKITO BASIC

Superset ID : 6393540

Name : ANTONY PRAVEEN

E-mail : antonypraveen.2205009@srec.acin

Mandatory Questions:

1) Exercise 1: Mocking and Stubbing

Solution:

```
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
class ExternalApi {
    String getData() {
        return "Real Data";
    }
}
class MyService {
    private ExternalApi api;
    public MyService(ExternalApi api) {
        this.api = api;
    }
    public String fetchData() {
        return api.getData();
    }
}
public class MyServiceTest {
    @Test
    public void testExternalApi() {
        ExternalApi mockApi = mock(ExternalApi.class);
        when(mockApi.getData()).thenReturn("Mock Data");
        MyService service = new MyService(mockApi);
```

```
        String result = service.fetchData();
        assertEquals("Mock Data", result);
    }
}
```

2) Exercise 2: Verifying Interactions

Solution:

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

Other Questions:

3) Exercise 3: Argument Matching

Solution:

```
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
class Calculator {
    int add(int a, int b) {
        return a + b;
    }
}
```

```
public class CalculatorTest {
    @Test
    public void testArgumentMatching() {
        Calculator mockCalc = mock(Calculator.class);
        mockCalc.add(5, 3);
        verify(mockCalc).add(anyInt(), eq(3));
    }
}
```

4) Exercise 4: Handling Void Methods

Solution:

```
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
class Printer {
    void print(String msg) {
        System.out.println(msg);
    }
}
public class PrinterTest {
    @Test
    public void testVoidMethod() {
        Printer mockPrinter = mock(Printer.class);
        mockPrinter.print("Hello");
        verify(mockPrinter).print("Hello");
    }
}
```

5) Exercise 5: Mocking Multiple Returns

Solution:

```
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
class ExternalApi {
    String getData() {
        return "Real Data";
    }
}
public class MyServiceTest {
    @Test
    public void testMultipleReturns() {
        ExternalApi mockApi = mock(ExternalApi.class);
        when(mockApi.getData())
            .thenReturn("First Call")
            .thenReturn("Second Call");
        assertEquals("First Call", mockApi.getData());
        assertEquals("Second Call", mockApi.getData());
    }
}
```

6) Exercise 6: Verifying Interaction Order

Solution:

```
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
import org.mockito.InOrder;
class Logger {
    void logStart() {}
    void logEnd() {}
```

```
}
public class LoggerTest {
    @Test
    public void testInteractionOrder() {
        Logger mockLogger = mock(Logger.class);
        mockLogger.logStart();
        mockLogger.logEnd();
        InOrder inOrder = inOrder(mockLogger);
        inOrder.verify(mockLogger).logStart();
        inOrder.verify(mockLogger).logEnd();
    }
}
```

7) Exercise 7: Handling Void Methods with Exceptions

Solution:

```
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
class Printer {
    void print(String msg) {
        System.out.println(msg);
    }
}
public class PrinterTest {
    @Test
    public void testVoidMethodThrowsException() {
        Printer mockPrinter = mock(Printer.class);
        doThrow(new RuntimeException("Print error"))
            .when(mockPrinter).print("error");
        try {
            mockPrinter.print("error");
```

```
        } catch (RuntimeException e) {
            assert(e.getMessage().equals("Print error"));
        }
        verify(mockPrinter).print("error");
    }
}
```

MOCKITO ADVANCED

Other Questions:

1) Exercise 1: Mocking Databases and Repositories

Solution:

```
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
interface Repository {
    String getData();
}
class Service {
    Repository repository;
    public Service(Repository repository) {
        this.repository = repository;
    }
    public String processData() {
        return "Processed " + repository.getData();
    }
}
public class ServiceTest {
    @Test
    public void testServiceWithMockRepository() {
        Repository mockRepo = mock(Repository.class);
        when(mockRepo.getData()).thenReturn("Mock Data");
```

```
        Service service = new Service(mockRepo);
        String result = service.processData();
        assertEquals("Processed Mock Data", result);
    }
}
```

2) Exercise 2: Mocking External Services (RESTful APIs)

Solution:

```
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
interface RestClient {
    String getResponse();
}
class ApiService {
    RestClient restClient;
    public ApiService(RestClient restClient) {
        this.restClient = restClient;
    }
    public String fetchData() {
        return "Fetched " + restClient.getResponse();
    }
}
public class ApiServiceTest {
    @Test
    public void testServiceWithMockRestClient() {
        RestClient mockClient = mock(RestClient.class);
        when(mockClient.getResponse()).thenReturn("Mock Response");
        ApiService service = new ApiService(mockClient);
        String result = service.fetchData();
```

```
        assertEquals("Fetched Mock Response", result);
    }
}
```

3) Exercise 3: Mocking File I/O

Solution:

```
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
interface FileReader {
    String read();
}
interface FileWriter {
    void write(String data);
}
class FileService {
    FileReader reader;
    FileWriter writer;
    public FileService(FileReader reader, FileWriter writer) {
        this.reader = reader;
        this.writer = writer;
    }
    public String processFile() {
        String content = reader.read();
        writer.write("Processed " + content);
        return "Processed " + content;
    }
}
public class FileServiceTest {
    @Test
```

```
    public void testServiceWithMockFileIO() {
        FileReader mockReader = mock(FileReader.class);
        FileWriter mockWriter = mock(FileWriter.class);
        when(mockReader.read()).thenReturn("Mock File Content");
        FileService service = new FileService(mockReader, mockWriter);
        String result = service.processFile();
        assertEquals("Processed Mock File Content", result);
        verify(mockWriter).write("Processed Mock File Content");
    }
}
```

4) Exercise 4: Mocking Network Interactions

Solution:

```
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
interface NetworkClient {
    String connect();
}
class NetworkService {
    NetworkClient client;
    public NetworkService(NetworkClient client) {
        this.client = client;
    }
    public String connectToServer() {
        return "Connected to " + client.connect();
    }
}
public class NetworkServiceTest {
    @Test
```

```
    public void testServiceWithMockNetworkClient() {
        NetworkClient mockClient = mock(NetworkClient.class);
        when(mockClient.connect()).thenReturn("Mock Connection");
        NetworkService service = new NetworkService(mockClient);
        String result = service.connectToServer();
        assertEquals("Connected to Mock Connection", result);
    }
}
```

5) Exercise 5: Mocking Multiple Return Values

Solution:

```
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;
interface Repository {
    String getData();
}
class Service {
    Repository repository;
    public Service(Repository repository) {
        this.repository = repository;
    }
    public String processData() {
        return "Processed " + repository.getData();
    }
}
public class MultiReturnServiceTest {
    @Test
    public void testServiceWithMultipleReturnValues() {
        Repository mockRepo = mock(Repository.class);
```

```
        when(mockRepo.getData())
            .thenReturn("First Mock Data")
            .thenReturn("Second Mock Data");
        Service service = new Service(mockRepo);
        String result1 = service.processData();
        String result2 = service.processData();
        assertEquals("Processed First Mock Data", result1);
        assertEquals("Processed Second Mock Data", result2);
    }
}
```