Advanced Mockito Hands-On Exercises - Solution

# Exercise 1: Mocking Databases and Repositories

Test a service that interacts with a mock repository:

import static org.mockito.Mockito.\*;  
import org.junit.jupiter.api.Test;  
import static org.junit.jupiter.api.Assertions.\*;  
  
class Repository {  
 public String getData() {  
 return "Real Data";  
 }  
}  
  
class Service {  
 private Repository repository;  
 public Service(Repository repository) {  
 this.repository = repository;  
 }  
 public String processData() {  
 return "Processed " + repository.getData();  
 }  
}  
  
public class ServiceTest {  
 @Test  
 public void testServiceWithMockRepository() {  
 Repository mockRepository = mock(Repository.class);  
 when(mockRepository.getData()).thenReturn("Mock Data");  
  
 Service service = new Service(mockRepository);  
 String result = service.processData();  
  
 assertEquals("Processed Mock Data", result);  
 }  
}

Output:  
Test passed, mocked repository returned expected data.

# Exercise 2: Mocking External Services (RESTful APIs)

class RestClient {  
 public String getResponse() {  
 return "Real Response";  
 }  
}  
  
class ApiService {  
 private RestClient client;  
 public ApiService(RestClient client) {  
 this.client = client;  
 }  
 public String fetchData() {  
 return "Fetched " + client.getResponse();  
 }  
}  
  
public class ApiServiceTest {  
 @Test  
 public void testServiceWithMockRestClient() {  
 RestClient mockRestClient = mock(RestClient.class);  
 when(mockRestClient.getResponse()).thenReturn("Mock Response");  
  
 ApiService apiService = new ApiService(mockRestClient);  
 String result = apiService.fetchData();  
  
 assertEquals("Fetched Mock Response", result);  
 }  
}

Output:  
REST client mock returned 'Mock Response' as expected.

# Exercise 3: Mocking File I/O

class FileReader {  
 public String read() {  
 return "Real Content";  
 }  
}  
  
class FileWriter {  
 public void write(String content) {  
 // writes to file  
 }  
}  
  
class FileService {  
 private FileReader reader;  
 private FileWriter writer;  
 public FileService(FileReader reader, FileWriter writer) {  
 this.reader = reader;  
 this.writer = writer;  
 }  
 public String processFile() {  
 return "Processed " + reader.read();  
 }  
}  
  
public class FileServiceTest {  
 @Test  
 public void testServiceWithMockFileIO() {  
 FileReader mockFileReader = mock(FileReader.class);  
 FileWriter mockFileWriter = mock(FileWriter.class);  
 when(mockFileReader.read()).thenReturn("Mock File Content");  
  
 FileService fileService = new FileService(mockFileReader, mockFileWriter);  
 String result = fileService.processFile();  
  
 assertEquals("Processed Mock File Content", result);  
 }  
}

Output:  
Mocked file I/O simulated correctly. Test passed.

# Exercise 4: Mocking Network Interactions

class NetworkClient {  
 public String connect() {  
 return "Real Connection";  
 }  
}  
  
class NetworkService {  
 private 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 mockNetworkClient = mock(NetworkClient.class);  
 when(mockNetworkClient.connect()).thenReturn("Mock Connection");  
  
 NetworkService networkService = new NetworkService(mockNetworkClient);  
 String result = networkService.connectToServer();  
  
 assertEquals("Connected to Mock Connection", result);  
 }  
}

Output:  
Network call mocked and validated successfully.

# Exercise 5: Mocking Multiple Return Values

public class MultiReturnServiceTest {  
 @Test  
 public void testServiceWithMultipleReturnValues() {  
 Repository mockRepository = mock(Repository.class);  
 when(mockRepository.getData())  
 .thenReturn("First Mock Data")  
 .thenReturn("Second Mock Data");  
  
 Service service = new Service(mockRepository);  
 String firstResult = service.processData();  
 String secondResult = service.processData();  
  
 assertEquals("Processed First Mock Data", firstResult);  
 assertEquals("Processed Second Mock Data", secondResult);  
 }  
}

Output:  
Mock returned sequential data correctly for each call.