JUnit and Mockito Complete Solutions

JUnit Basic Testing

Exercise 1: Setting Up JUnit

xml

```xml
<dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>4.13.2</version>
    <scope>test</scope>
</dependency>
```

Exercise 2: Writing Basic JUnit Tests

java

```java
public class Calculator {
    public int add(int a, int b) {
        return a + b;
    }

    public int subtract(int a, int b) {
        return a - b;
    }

    public int multiply(int a, int b) {
        return a * b;
    }

    public int divide(int a, int b) {
        if (b == 0) throw new IllegalArgumentException("Division by zero");
        return a / b;
    }
}
```

java

```java
import org.junit.Test;
import static org.junit.Assert.*;

public class CalculatorTest {

    @Test
    public void testAdd() {
        Calculator calc = new Calculator();
        int result = calc.add(5, 3);
        System.out.println("Addition: 5 + 3 = " + result);
        assertEquals(8, result);
    }

    @Test
    public void testSubtract() {
        Calculator calc = new Calculator();
        int result = calc.subtract(10, 4);
        System.out.println("Subtraction: 10 - 4 = " + result);
        assertEquals(6, result);
    }

    @Test
    public void testMultiply() {
        Calculator calc = new Calculator();
        int result = calc.multiply(7, 6);
        System.out.println("Multiplication: 7 * 6 = " + result);
        assertEquals(42, result);
    }

    @Test
    public void testDivide() {
        Calculator calc = new Calculator();
        int result = calc.divide(20, 4);
        System.out.println("Division: 20 / 4 = " + result);
        assertEquals(5, result);
    }
}
```

Output

```
Addition: 5 + 3 = 8
Subtraction: 10 - 4 = 6
Multiplication: 7 * 6 = 42
Division: 20 / 4 = 5
```

Exercise 3: Assertions in JUnit

java

```java
import org.junit.Test;
import static org.junit.Assert.*;

public class AssertionsTest {
    @Test
    public void testAssertions() {
        System.out.println("Testing assertEquals: 2 + 3 = " + (2 + 3));
        assertEquals(5, 2 + 3);

        System.out.println("Testing assertTrue: 5 > 3 is " + (5 > 3));
        assertTrue(5 > 3);

        System.out.println("Testing assertFalse: 5 < 3 is " + (5 < 3));
        assertFalse(5 < 3);

        System.out.println("Testing assertNull: null is null");
        assertNull(null);

        System.out.println("Testing assertNotNull: new Object() is not null");
        assertNotNull(new Object());

        System.out.println("All assertions passed successfully");
    }
}
```

Output

```
Testing assertEquals: 2 + 3 = 5
Testing assertTrue: 5 > 3 is true
Testing assertFalse: 5 < 3 is false
Testing assertNull: null is null
Testing assertNotNull: new Object() is not null
All assertions passed successfully
```

Exercise 4: AAA Pattern with Setup and Teardown

java

```java
import org.junit.Before;
import org.junit.After;
import org.junit.Test;
import static org.junit.Assert.*;

public class AAA_PatternTest {
    private Calculator calculator;

    @Before
    public void setUp() {
        calculator = new Calculator();
        System.out.println("Setup: Calculator initialized");
    }

    @After
    public void tearDown() {
        calculator = null;
        System.out.println("Teardown: Calculator cleaned up");
    }

    @Test
    public void testAddition() {
        int a = 10;
        int b = 5;
        int result = calculator.add(a, b);
        System.out.println("Testing addition: " + a + " + " + b + " = " + result);
        assertEquals(15, result);
    }

    @Test
    public void testDivision() {
        int dividend = 20;
        int divisor = 4;
        int result = calculator.divide(dividend, divisor);
        System.out.println("Testing division: " + dividend + " / " + divisor + " = " + result);
        assertEquals(5, result);
    }
}
```

Output

```
Setup: Calculator initialized
Testing addition: 10 + 5 = 15
Teardown: Calculator cleaned up
Setup: Calculator initialized
Testing division: 20 / 4 = 5
Teardown: Calculator cleaned up
```

JUnit Advanced Testing

Exercise 1: Parameterized Tests

java

```java
public class EvenChecker {
    public boolean isEven(int number) {
        return number % 2 == 0;
    }
}
```

java

```java
import org.junit.jupiter.params.ParameterizedTest;
import org.junit.jupiter.params.provider.ValueSource;
import static org.junit.jupiter.api.Assertions.*;

public class EvenCheckerTest {

    @ParameterizedTest
    @ValueSource(ints = {2, 4, 6, 8, 10})
    public void testEvenNumbers(int number) {
        EvenChecker checker = new EvenChecker();
        boolean result = checker.isEven(number);
        System.out.println("Testing even number: " + number + " is even = " + result);
        assertTrue(result);
    }

    @ParameterizedTest
    @ValueSource(ints = {1, 3, 5, 7, 9})
    public void testOddNumbers(int number) {
        EvenChecker checker = new EvenChecker();
        boolean result = checker.isEven(number);
        System.out.println("Testing odd number: " + number + " is even = " + result);
        assertFalse(result);
    }
}
```

Output

```
Testing even number: 2 is even = true
Testing even number: 4 is even = true
Testing even number: 6 is even = true
Testing even number: 8 is even = true
Testing even number: 10 is even = true
Testing odd number: 1 is even = false
Testing odd number: 3 is even = false
Testing odd number: 5 is even = false
Testing odd number: 7 is even = false
Testing odd number: 9 is even = false
```

Exercise 2: Test Suites

java

```java
import org.junit.runner.RunWith;
import org.junit.runners.Suite;

@RunWith(Suite.class)
@Suite.SuiteClasses({
    CalculatorTest.class,
    AssertionsTest.class,
    AAA_PatternTest.class
})
public class AllTests {
}
```

Output

```
Running CalculatorTest...
Addition: 5 + 3 = 8
Subtraction: 10 - 4 = 6
Multiplication: 7 * 6 = 42
Division: 20 / 4 = 5
Running AssertionsTest...
All assertions passed successfully
Running AAA_PatternTest...
Setup: Calculator initialized
Testing addition: 10 + 5 = 15
Teardown: Calculator cleaned up
```

Exercise 3: Test Execution Order

java

```java
import org.junit.jupiter.api.Test;
import org.junit.jupiter.api.TestMethodOrder;
import org.junit.jupiter.api.MethodOrderer;
import org.junit.jupiter.api.Order;

@TestMethodOrder(MethodOrderer.OrderAnnotation.class)
public class OrderedTests {

    @Test
    @Order(1)
    public void testFirst() {
        System.out.println("First test executed");
    }

    @Test
    @Order(2)
    public void testSecond() {
        System.out.println("Second test executed");
    }

    @Test
    @Order(3)
    public void testThird() {
        System.out.println("Third test executed");
    }
}
```

Output

```
First test executed
Second test executed
Third test executed
```

Exercise 4: Exception Testing

java

```java
public class ExceptionThrower {
    public void throwException() {
        throw new IllegalArgumentException("This is a test exception");
    }

    public int divide(int a, int b) {
        if (b == 0) {
            throw new ArithmeticException("Cannot divide by zero");
        }
        return a / b;
    }
}
```

java

```java
import org.junit.Test;
import static org.junit.Assert.*;

public class ExceptionThrowerTest {

    @Test(expected = IllegalArgumentException.class)
    public void testThrowException() {
        System.out.println("Testing exception throwing...");
        ExceptionThrower thrower = new ExceptionThrower();
        thrower.throwException();
    }

    @Test(expected = ArithmeticException.class)
    public void testDivideByZero() {
        System.out.println("Testing divide by zero exception...");
        ExceptionThrower thrower = new ExceptionThrower();
        thrower.divide(10, 0);
    }
}
```

Output

```
Testing exception throwing...
Testing divide by zero exception...
```

Exercise 5: Timeout and Performance Testing

java

```java
public class PerformanceTester {
    public void performTask() {
        try {
            System.out.println("Starting task...");
            Thread.sleep(100);
            System.out.println("Task completed in 100ms");
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }

    public void performSlowTask() {
        try {
            System.out.println("Starting slow task...");
            Thread.sleep(2000);
            System.out.println("Slow task completed in 2000ms");
        } catch (InterruptedException e) {
            Thread.currentThread().interrupt();
        }
    }
}
```

java

```java
import org.junit.Test;

public class PerformanceTesterTest {

    @Test(timeout = 1000)
    public void testPerformTask() {
        System.out.println("Testing fast task with 1000ms timeout");
        PerformanceTester tester = new PerformanceTester();
        tester.performTask();
    }

    @Test(timeout = 500)
    public void testPerformSlowTaskTimeout() {
        System.out.println("Testing slow task with 500ms timeout");
        PerformanceTester tester = new PerformanceTester();
        tester.performSlowTask();
    }
}
```

Output

```
Testing fast task with 1000ms timeout
Starting task...
Task completed in 100ms
Testing slow task with 500ms timeout
Starting slow task...
Test timed out after 500 milliseconds
```

Mockito Exercises

Exercise 1: Mocking and Stubbing

java

```java
public interface ExternalApi {
    String getData();
    String fetchUserData(String userId);
}
```

java

```java
public class MyService {
    private ExternalApi externalApi;

    public MyService(ExternalApi externalApi) {
        this.externalApi = externalApi;
    }

    public String fetchData() {
        String data = externalApi.getData();
        System.out.println("Fetched data: " + data);
        return data;
    }

    public String getUserInfo(String userId) {
        String userData = externalApi.fetchUserData(userId);
        String result = "User: " + userData;
        System.out.println("User info: " + result);
        return result;
    }
}
```

java

```java
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class MyServiceTest {
  @Test
  public void testExternalApi() {
    System.out.println("Creating mock ExternalApi...");
    ExternalApi mockApi = mock(ExternalApi.class);
    when(mockApi.getData()).thenReturn("Mock Data");

    MyService service = new MyService(mockApi);
    String result = service.fetchData();

    assertEquals("Mock Data", result);
    System.out.println("Test passed: " + result);
  }

  @Test
  public void testGetUserInfo() {
    System.out.println("Testing user info with mock...");
    ExternalApi mockApi = mock(ExternalApi.class);
    when(mockApi.fetchUserData("123")).thenReturn("John Doe");

    MyService service = new MyService(mockApi);
    String result = service.getUserInfo("123");

    assertEquals("User: John Doe", result);
    System.out.println("Test passed: " + result);
  }
}
```

Output

```
Creating mock ExternalApi...
Fetched data: Mock Data
Test passed: Mock Data
Testing user info with mock...
User info: User: John Doe
Test passed: User: John Doe
```

Exercise 2: Verifying Interactions

```java
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;

public class MyServiceVerifyTest {
  @Test
  public void testVerifyInteraction() {
    System.out.println("Testing interaction verification...");
    ExternalApi mockApi = mock(ExternalApi.class);
    when(mockApi.getData()).thenReturn("Mock Data");

    MyService service = new MyService(mockApi);
    service.fetchData();

    verify(mockApi).getData();
    System.out.println("Verification passed: getData() was called");
  }

  @Test
  public void testVerifyInteractionWithArguments() {
    System.out.println("Testing interaction with arguments...");
    ExternalApi mockApi = mock(ExternalApi.class);
    when(mockApi.fetchUserData("123")).thenReturn("John Doe");

    MyService service = new MyService(mockApi);
    service.getUserInfo("123");

    verify(mockApi).fetchUserData("123");
    System.out.println("Verification passed: fetchUserData('123') was called");
  }
}
```

Output

```
Testing interaction verification...
Fetched data: Mock Data
Verification passed: getData() was called
Testing interaction with arguments...
User info: User: John Doe
Verification passed: fetchUserData('123') was called
```

Exercise 3: Argument Matching

java

```java
import static org.mockito.Mockito.*;
import static org.mockito.ArgumentMatchers.*;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class ArgumentMatchingTest {
  @Test
  public void testArgumentMatching() {
    System.out.println("Testing argument matching with anyString()...");
    ExternalApi mockApi = mock(ExternalApi.class);
    when(mockApi.fetchUserData(anyString())).thenReturn("Any User");

    MyService service = new MyService(mockApi);
    String result = service.getUserInfo("randomId");

    assertEquals("User: Any User", result);
    verify(mockApi).fetchUserData(anyString());
    System.out.println("Argument matching test passed");
  }

  @Test
  public void testSpecificArgumentMatching() {
    System.out.println("Testing specific argument matching...");
    ExternalApi mockApi = mock(ExternalApi.class);
    when(mockApi.fetchUserData(eq("123"))).thenReturn("Specific User");

    MyService service = new MyService(mockApi);
    service.getUserInfo("123");

    verify(mockApi).fetchUserData(eq("123"));
    System.out.println("Specific argument matching test passed");
  }
}
```

Output

```
Testing argument matching with anyString()...
User info: User: Any User
Argument matching test passed
Testing specific argument matching...
User info: User: Specific User
Specific argument matching test passed
```

Exercise 4: Handling Void Methods

java

```java
public interface NotificationService {
    void sendNotification(String message);
    void sendEmail(String to, String subject);
}
```

java

```java
public class AlertService {
    private NotificationService notificationService;

    public AlertService(NotificationService notificationService) {
        this.notificationService = notificationService;
    }

    public void sendAlert(String message) {
        String alertMessage = "Alert: " + message;
        System.out.println("Sending alert: " + alertMessage);
        notificationService.sendNotification(alertMessage);
    }

    public void sendEmailAlert(String email, String subject) {
        System.out.println("Sending email to: " + email + " with subject: " + subject);
        notificationService.sendEmail(email, subject);
    }
}
```

java

```java
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;

public class VoidMethodTest {
    @Test
    public void testVoidMethod() {
        System.out.println("Testing void method interaction...");
        NotificationService mockService = mock(NotificationService.class);

        AlertService alertService = new AlertService(mockService);
        alertService.sendAlert("Test Message");

        verify(mockService).sendNotification("Alert: Test Message");
        System.out.println("Void method test passed");
    }

    @Test
    public void testVoidMethodWithMultipleArgs() {
        System.out.println("Testing void method with multiple arguments...");
        NotificationService mockService = mock(NotificationService.class);

        AlertService alertService = new AlertService(mockService);
        alertService.sendEmailAlert("test@example.com", "Test Subject");

        verify(mockService).sendEmail("test@example.com", "Test Subject");
        System.out.println("Multiple args void method test passed");
    }
}
```

Output

```
Testing void method interaction...
Sending alert: Alert: Test Message
Void method test passed
Testing void method with multiple arguments...
Sending email to: test@example.com with subject: Test Subject
Multiple args void method test passed
```

Exercise 5: Multiple Return Values

```java
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;
import org.junit.jupiter.api.Test;

public class MultipleReturnsTest {
  @Test
  public void testMultipleReturnValues() {
    System.out.println("Testing multiple return values...");
    ExternalApi mockApi = mock(ExternalApi.class);
    when(mockApi.getData())
      .thenReturn("First Call")
      .thenReturn("Second Call")
      .thenReturn("Third Call");

    MyService service = new MyService(mockApi);

    String first = service.fetchData();
    String second = service.fetchData();
    String third = service.fetchData();

    assertEquals("First Call", first);
    assertEquals("Second Call", second);
    assertEquals("Third Call", third);

    System.out.println("Multiple return values test passed");
  }
}
```

Output

```
Testing multiple return values...
Fetched data: First Call
Fetched data: Second Call
Fetched data: Third Call
Multiple return values test passed
```

Exercise 6: Verifying Interaction Order

java

```java
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
import org.mockito.InOrder;

public class InteractionOrderTest {
  @Test
  public void testInteractionOrder() {
    System.out.println("Testing interaction order...");
    ExternalApi mockApi = mock(ExternalApi.class);
    NotificationService mockNotification = mock(NotificationService.class);

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

    MyService service = new MyService(mockApi);
    AlertService alertService = new AlertService(mockNotification);

    service.fetchData();
    alertService.sendAlert("Processing complete");

    InOrder inOrder = inOrder(mockApi, mockNotification);
    inOrder.verify(mockApi).getData();
    inOrder.verify(mockNotification).sendNotification("Alert: Processing complete");

    System.out.println("Interaction order test passed");
  }
}
```

Output

```
Testing interaction order...
Fetched data: Data
Sending alert: Alert: Processing complete
Interaction order test passed
```

Exercise 7: Handling Void Methods with Exceptions

java

```java
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class VoidMethodExceptionTest {
  @Test
  public void testVoidMethodException() {
    System.out.println("Testing void method exception...");
    NotificationService mockService = mock(NotificationService.class);

    doThrow(new RuntimeException("Email service down")).when(mockService).sendEmail(anyString(), anyString());

    AlertService alertService = new AlertService(mockService);

    assertThrows(RuntimeException.class, () -> {
      alertService.sendEmailAlert("test@example.com", "Test");
    });

    System.out.println("Void method exception test passed");
  }
}
```

Output

```
Testing void method exception...
Sending email to: test@example.com with subject: Test
Void method exception test passed
```

Advanced Mockito Exercises

Exercise 1: Mocking Databases and Repositories

java

```java
public interface Repository {
  String getData();
  void saveData(String data);
}
```

java

```java
public class Service {
    private Repository repository;

    public Service(Repository repository) {
        this.repository = repository;
    }

    public String processData() {
        String data = repository.getData();
        String processed = "Processed " + data;
        System.out.println("Processing: " + data + " -> " + processed);
        return processed;
    }
}
```

java

```java
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class ServiceTest {
    @Test
    public void testServiceWithMockRepository() {
        System.out.println("Testing service with mock repository...");
        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);
        System.out.println("Repository mock test passed: " + result);
    }
}
```

Output

```
Testing service with mock repository...
Processing: Mock Data -> Processed Mock Data
Repository mock test passed: Processed Mock Data
```

Exercise 2: Mocking External Services (RESTful APIs)

java

```java
public interface RestClient {
    String getResponse();
    String postData(String data);
}
```

java

```java
public class ApiService {
    private RestClient restClient;

    public ApiService(RestClient restClient) {
        this.restClient = restClient;
    }

    public String fetchData() {
        String response = restClient.getResponse();
        String result = "Fetched " + response;
        System.out.println("API Response: " + response + " -> " + result);
        return result;
    }
}
```

java

```java
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class ApiServiceTest {
    @Test
    public void testServiceWithMockRestClient() {
        System.out.println("Testing API service with mock REST client...");
        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);
        System.out.println("REST client mock test passed: " + result);
    }
}
```

Output

```
Testing API service with mock REST client...
API Response: Mock Response -> Fetched Mock Response
REST client mock test passed: Fetched Mock Response
```

Exercise 3: Mocking File I/O

java

```java
public interface FileReader {
    String read();
}

public interface FileWriter {
    void write(String content);
}
```

java

```java
public class FileService {
    private FileReader fileReader;
    private FileWriter fileWriter;

    public FileService(FileReader fileReader, FileWriter fileWriter) {
        this.fileReader = fileReader;
        this.fileWriter = fileWriter;
    }

    public String processFile() {
        String content = fileReader.read();
        String processed = "Processed " + content;
        System.out.println("File processing: " + content + " -> " + processed);
        fileWriter.write(processed);
        return processed;
    }
}
```

java

```java
import static org.mockito.Mockito.*;
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class FileServiceTest {
    @Test
    public void testServiceWithMockFileIO() {
        System.out.println("Testing file service with mock file I/O...");
        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);
        verify(mockFileWriter).write("Processed Mock File Content");
        System.out.println("File I/O mock test passed: " + result);
    }
}
```

Output

```
Testing file service with mock file I/O...
File processing: Mock File Content -> Processed Mock File Content
File I/O mock test passed: Processed Mock File Content
```

Spring Testing Exercises

Exercise 1: Basic Unit Test for a Service Method

java

```java
import org.springframework.stereotype.Service;

@Service
public class CalculatorService {
    public int add(int a, int b) {
        int result = a + b;
        System.out.println("CalculatorService: " + a + " + " + b + " = " + result);
        return result;
    }
}
```

java

```java
import org.junit.jupiter.api.Test;
import static org.junit.jupiter.api.Assertions.*;

public class CalculatorServiceTest {
    @Test
    public void testAdd() {
        System.out.println("Testing CalculatorService.add()...");
        CalculatorService calculatorService = new CalculatorService();
        int result = calculatorService.add(5, 3);
        assertEquals(8, result);
        System.out.println("CalculatorService test passed");
    }
}
```

Output

```
Testing CalculatorService.add()...
CalculatorService: 5 + 3 = 8
CalculatorService test passed
```

Exercise 2: Mocking a Repository in a Service Test

java

```java
import javax.persistence.Entity;
import javax.persistence.Id;

@Entity
public class User {
    @Id
    private Long id;
    private String name;

    public User() {}

    public User(Long id, String name) {
        this.id = id;
        this.name = name;
    }

    public Long getId() { return id; }
    public void setId(Long id) { this.id = id; }
    public String getName() { return name; }
    public void setName(String name) { this.name = name; }

    @Override
    public String toString() {
        return "User{id=" + id + ", name='" + name + '\'' + '}';
    }
}
```

java

```java
import org.springframework.data.jpa.repository.JpaRepository;

public interface UserRepository extends JpaRepository<User, Long> {
}
```

java

```java
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import java.util.Optional;

@Service
public class UserService {
    @Autowired
    private UserRepository userRepository;

    public UserService(UserRepository userRepository) {
        this.userRepository = userRepository;
    }

    public User getUserById(Long id) {
        Optional<User> user = userRepository.findById(id);
        System.out.println("UserService: Finding user with ID " + id);
        User result = user.orElse(null);
        System.out.println("UserService: Found user: " + result);
        return result;
    }
}
```

java

```java
import org.junit.jupiter.api.Test;
import org.mockito.Mock;
import org.mockito.InjectMocks;
import org.mockito.junit.jupiter.MockitoExtension;
import org.junit.jupiter.api.extension.ExtendWith;
import static org.mockito.Mockito.*;
import static org.junit.jupiter.api.Assertions.*;
import java.util.Optional;

@ExtendWith(MockitoExtension.class)
public class UserServiceTest {

  @Mock
  private UserRepository userRepository;

  @InjectMocks
  private UserService userService;

  @Test
  public void testGetUserById() {
    System.out.println("Testing UserService.getUserById() with mock repository...");
    User mockUser = new User(1L, "John Doe");
    when(userRepository.findById(1L)).thenReturn(Optional.of(mockUser));

    User result = userService.getUserById(1L);

    assertNotNull(result);
    assertEquals("John Doe", result.getName());
    assertEquals(1L, result.getId());
    System.out.println("UserService mock test passed: " + result);
  }
}
```

Output

```
Testing UserService.getUserById() with mock repository...
UserService: Finding user with ID 1
UserService: Found user: User{id=1, name='John Doe'}
UserService mock test passed: User{id=1, name='John Doe'}
```

SLF4J Logging Exercises

Exercise 1: Logging Error Messages and Warning Levels

java

```java
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class LoggingExample {
    private static final Logger logger = LoggerFactory.getLogger(LoggingExample.class);

    public static void main(String[] args) {
        System.out.println("Starting logging example...");
        logger.error("This is an error message");
        logger.warn("This is a warning message");
        logger.info("This is an info message");
        logger.debug("This is a debug message");
        System.out.println("Logging example completed");
    }
}
```

Output

```
Starting logging example...
ERROR LoggingExample - This is an error message
WARN LoggingExample - This is a warning message
INFO LoggingExample - This is an info message
Logging example completed
```

Exercise 2: Parameterized Logging

java

```java
import org
```