# Java FSE Deep Skilling - Advanced Testing & Logging - Exercise Solutions

## SLF4J Logging Exercises

Exercise 1: Logging Error and Warning

```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) {

logger.error("This is an error message");

logger.warn("This is a warning message");

}

}

```

Exercise 2: Parameterized Logging

```java

public class ParamLogging {

private static final Logger logger = LoggerFactory.getLogger(ParamLogging.class);

public static void main(String[] args) {

String username = "Kapilan";

int attempts = 3;

logger.info("User {} tried to login {} times", username, attempts);

}

}

```

Exercise 3: logback.xml configuration for different appenders

```xml

<configuration>

<appender name="console" class="ch.qos.logback.core.ConsoleAppender">

<encoder>

<pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<appender name="file" class="ch.qos.logback.core.FileAppender">

<file>app.log</file>

<encoder>

<pattern>%d{HH:mm:ss.SSS} [%thread] %-5level %logger{36} - %msg%n</pattern>

</encoder>

</appender>

<root level="debug">

<appender-ref ref="console" />

<appender-ref ref="file" />

</root>

</configuration>

```

## Mockito Hands-On

Exercise 1: Mocking and Stubbing

```java

@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);

}

```

Exercise 2: Verifying Interactions

```java

@Test

public void testVerifyInteraction() {

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

MyService service = new MyService(mockApi);

service.fetchData();

verify(mockApi).getData();

}

```

## Advanced Mockito

Exercise 1: Mock Repository

```java

@Test

public void testServiceWithMockRepository() {

Repository mockRepository = mock(Repository.class);

when(mockRepository.getData()).thenReturn("Mock Data");

Service service = new Service(mockRepository);

assertEquals("Processed Mock Data", service.processData());

}

```

Exercise 2: Mock External API

```java

@Test

public void testServiceWithMockRestClient() {

RestClient mockClient = mock(RestClient.class);

when(mockClient.getResponse()).thenReturn("Mock Response");

ApiService api = new ApiService(mockClient);

assertEquals("Fetched Mock Response", api.fetchData());

}

```

## JUnit Basic & Advanced

Basic Assertions

```java

@Test

public void testAssertions() {

assertEquals(5, 2 + 3);

assertTrue(5 > 3);

assertFalse(2 > 3);

assertNull(null);

assertNotNull(new Object());

}

```

Parameterized Test

```java

@ParameterizedTest

@ValueSource(ints = {2, 4, 6})

void isEvenTest(int number) {

assertTrue(number % 2 == 0);

}

```

Exception Test

```java

@Test

void testThrowException() {

ExceptionThrower et = new ExceptionThrower();

assertThrows(IllegalArgumentException.class, () -> et.throwException());

}

```

Timeout Test

```java

@Test

@Timeout(value = 500, unit = TimeUnit.MILLISECONDS)

void testPerformance() {

new PerformanceTester().performTask();

}

```

Test Suite

```java

@Suite

@SelectClasses({ TestClass1.class, TestClass2.class })

public class AllTests {}

```

Setup/Teardown

```java

@BeforeEach

void init() {

System.out.println("Init before test");

}

@AfterEach

void cleanup() {

System.out.println("Cleanup after test");

}

```