Cognizant Java FSE – (Deep Skilling)

(WEEK-2) ADDITIONAL EXCERCISES

**MODULE 1:** PL/SQL programming

**MODULE 2:** TDD using JUnit5 and Mockito

**MODULE 3:** SLF4J logging framework

**Submitted by**  
**Name:** SOLLETI VENKATA KUSUMA  
**Roll No:** 111522102145  
**Email:** [22102145@rmd.ac.in](mailto:22102145@rmd.ac.in)

**College:** RMD ENGINEERING COLLEGE  
**Batch:** Java FSE – 2026

**PL/SQL ADDITIONAL EXCERCISES**

**Exercise 4:** Functions

Scenario 1: Calculate the age of customers for eligibility checks.

Question: Write a function CalculateAge that takes a customer's date of birth as input and returns their age in years.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

OUTPUT:

A white background with black text

AI-generated content may be incorrect.

**Scenario 2:** The bank needs to compute the monthly installment for a loan.

**Question:** Write a function CalculateMonthlyInstallment that takes the loan amount, interest rate, and loan duration in years as input and returns the monthly installment amount.

A screenshot of a computer

AI-generated content may be incorrect.

OUTPUT:

A white background with black text

AI-generated content may be incorrect.

**Scenario 3:** Check if a customer has sufficient balance before making a transaction.

**Question:** Write a function **HasSufficientBalance** that takes an account ID and an amount as input and returns a boolean indicating whether the account has at least the specified amount.

A screenshot of a computer

AI-generated content may be incorrect.

OUTPUT:

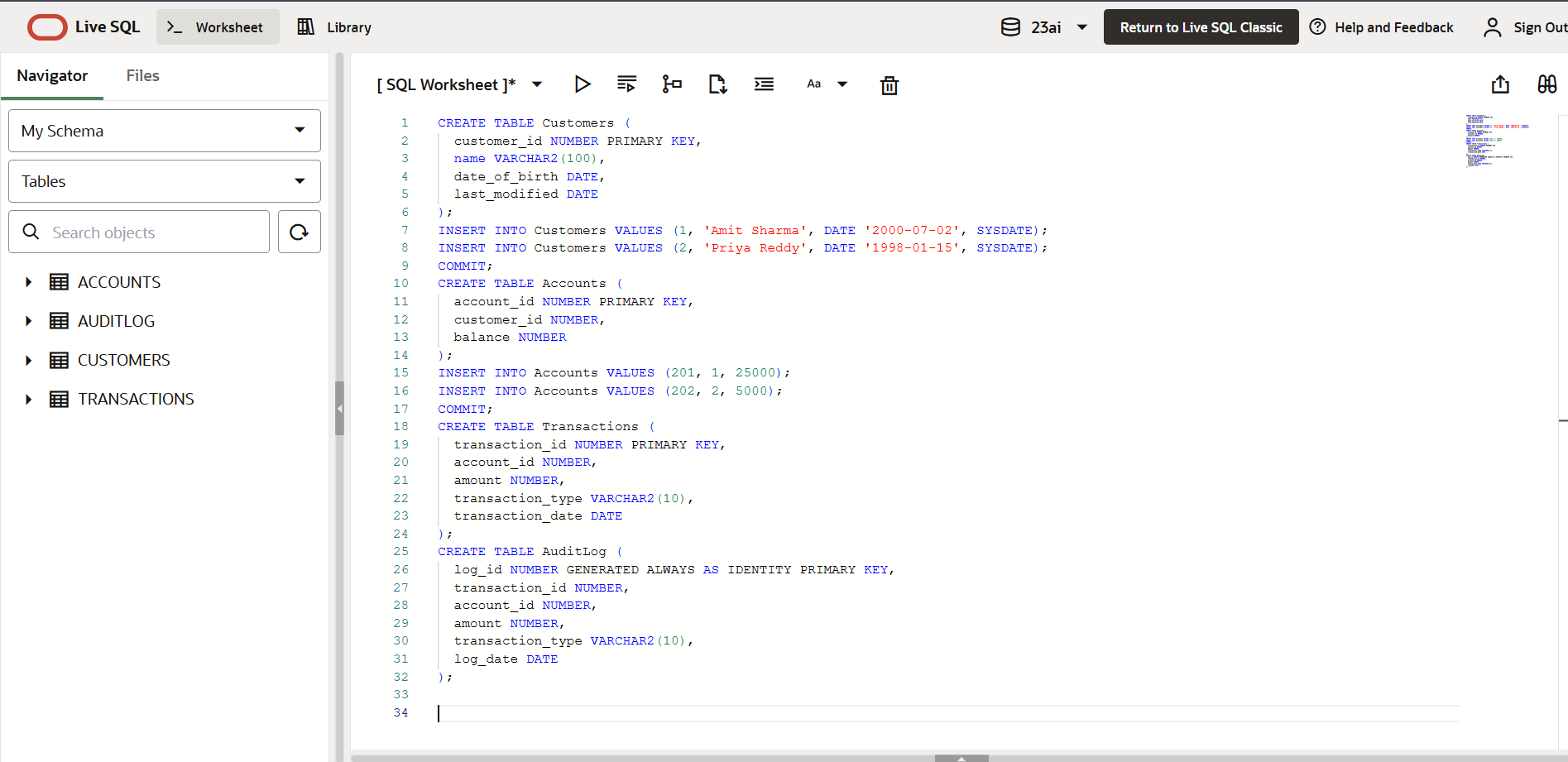
A white background with black text

AI-generated content may be incorrect.

**Exercise 5:** Triggers

**Scenario 1:** Automatically update the last modified date when a customer's record is updated.

**Question:** Write a trigger **UpdateCustomerLastModified** that updates the LastModified column of the Customers table to the current date whenever a customer's record is updated.



A screenshot of a computer

AI-generated content may be incorrect.

OUTPUT:

A close-up of a computer screen

AI-generated content may be incorrect.

**Scenario 2:** Maintain an audit log for all transactions.

**Question:** Write a trigger **LogTransaction** that inserts a record into an AuditLog table whenever a transaction is inserted into the Transactions table.

A screenshot of a computer

AI-generated content may be incorrect.

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

**Scenario 3:** Enforce business rules on deposits and withdrawals.

**Question:** Write a trigger **CheckTransactionRules** that ensures withdrawals do not exceed the balance and deposits are positive before inserting a record into the Transactions table.

A screenshot of a computer

AI-generated content may be incorrect.  
OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

**MOCKITO ADDITIONAL EXCERCISES**

**Exercise 3:** Argument Matching

**Scenario:**

You need to verify that a method is called with specific arguments.

Steps:

1. Create a mock object.

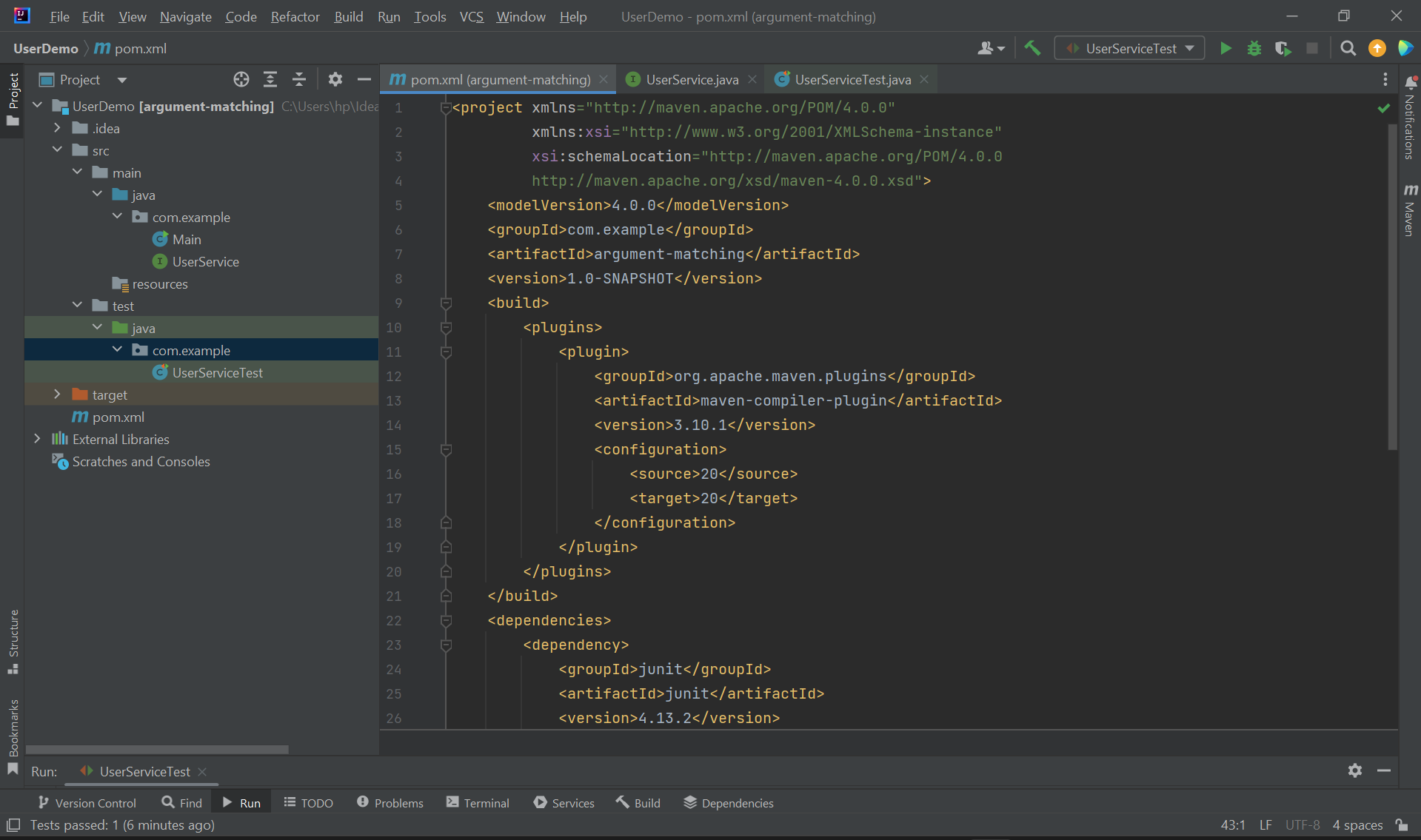
2. Call the method with specific arguments.

3. Use argument matchers to verify the interaction.

**Step 1: Create a Maven Project**

1. Open IntelliJ IDEA.
2. Click New Project(UserDemo) -Select Maven.
3. GroupId: com.example and ArtifactId:UserDemo
4. Click Finish.

**Step 2: Add Junit and Mockito to pom.xml**

****

**A screen shot of a computer

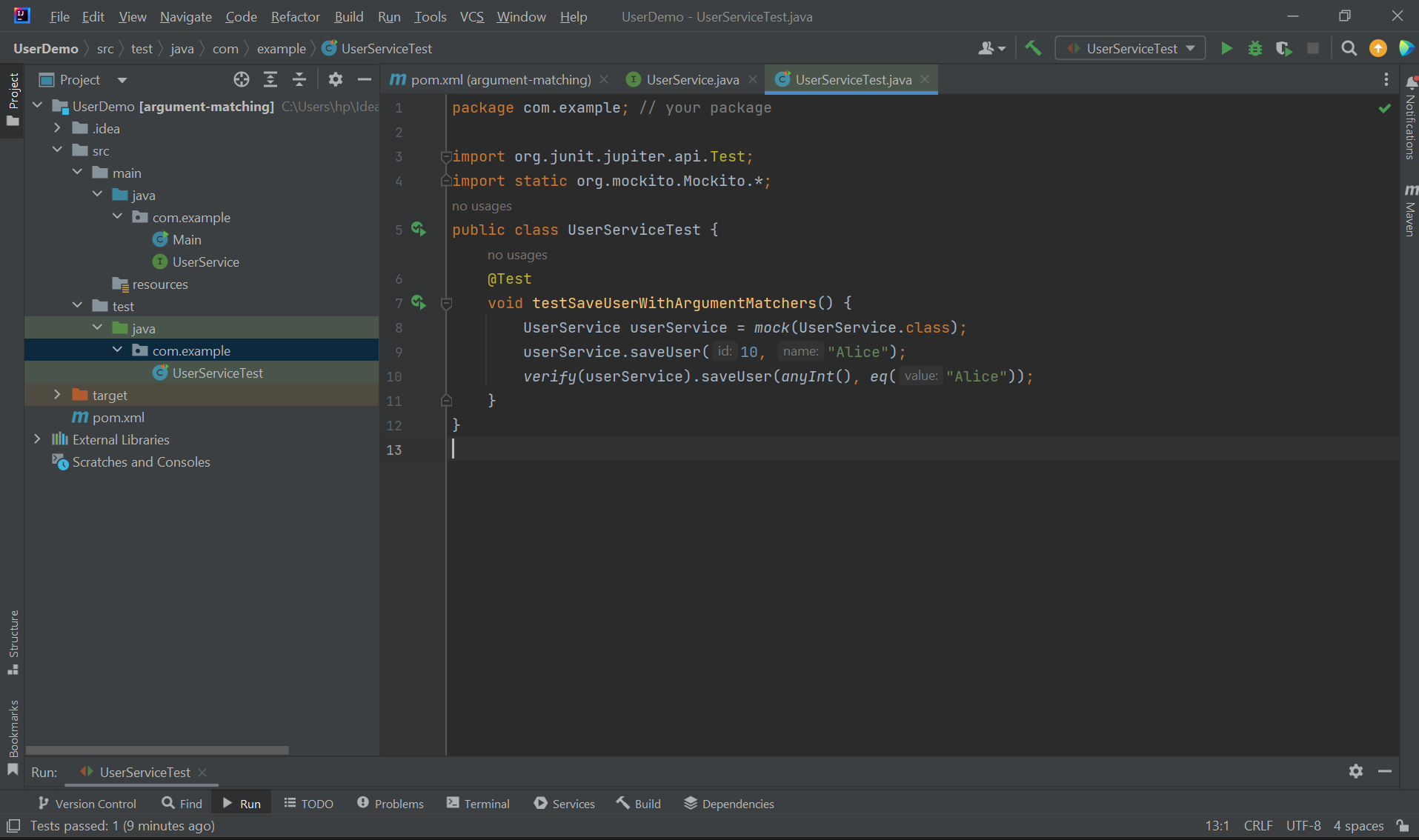
AI-generated content may be incorrect.**

**Step 3:Create an interface UserService.java**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Step 4:Create a Test class named UserServiceTest.java**

****

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 4:** Handling Void Methods

**Scenario:**

You need to test a void method that performs some action.

Steps:

1. Create a mock object.

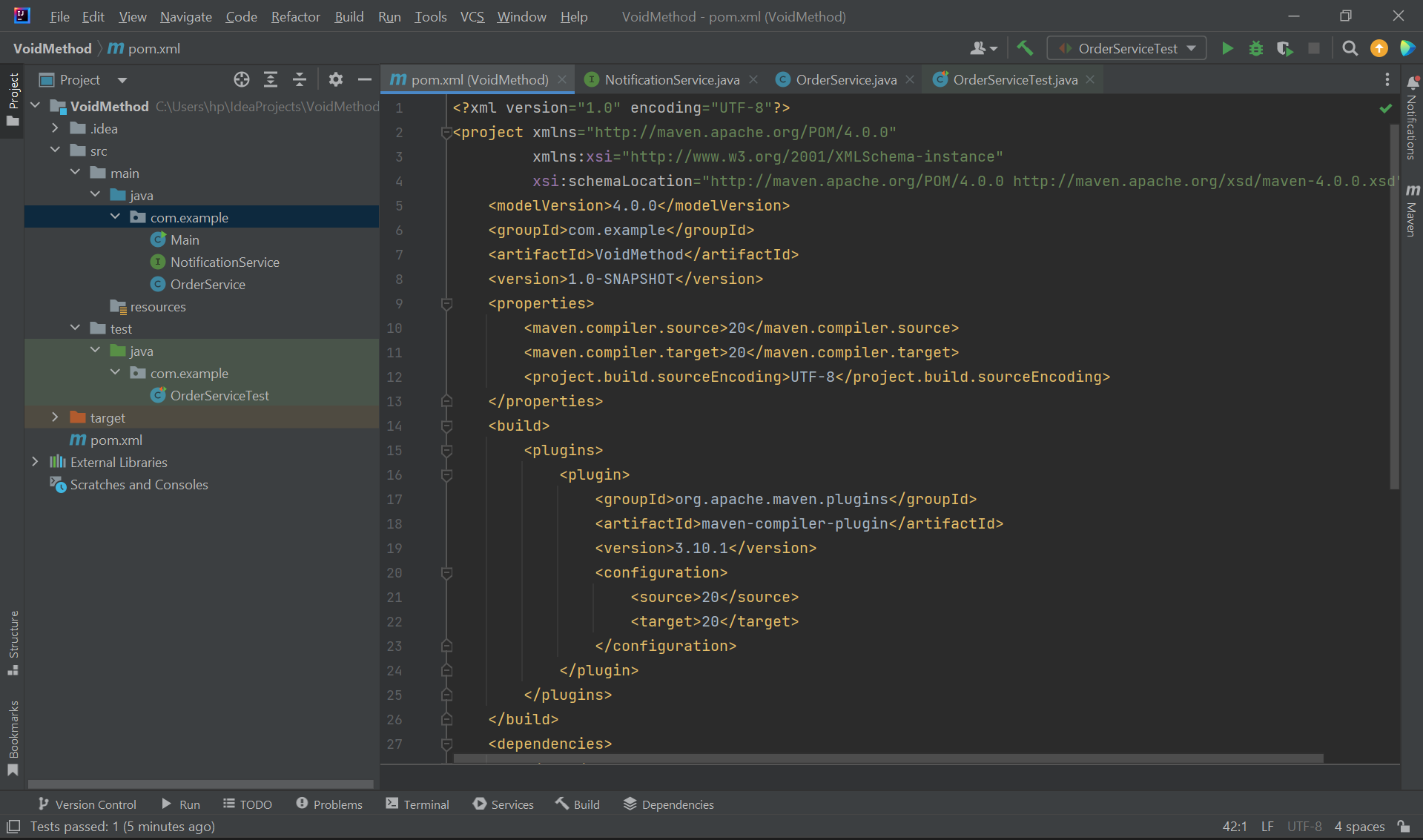
2. Stub the void method.

3. Verify the interaction.

**Step 1: Create a Maven Project**

1. Open IntelliJ IDEA.
2. Click New Project(VoidMethod) -Select Maven.
3. GroupId: com.example and ArtifactId:VoidMethod
4. Click Finish.

**Step 2: Add Junit and Mockito to pom.xml**



A screenshot of a computer

AI-generated content may be incorrect.

**Step 3:Create an interface NotificationService.java**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Step 4:Create a class named OrderService.java**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Step 4:Create a Test class named OrderServiceTest.java**

**A screenshot of a computer

AI-generated content may be incorrect.**

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 5:** Mocking and Stubbing with Multiple Returns

**Scenario:**

You need to test a service that depends on an external API with multiple return values.

Steps:

1. Create a mock object for the external API.

2. Stub the methods to return different values on consecutive calls.

3. Write a test case that uses the mock object.

**Step 1: Create a Maven Project**

1. Open IntelliJ IDEA.
2. Click New Project(MultipleReturn) -Select Maven.
3. GroupId: com.example and ArtifactId:MultipleReturn
4. Click Finish.

**Step 2: Add Junit and Mockito to pom.xml**

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Step 3:Create the ExternalAPI interface**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Step 4:** **Create the DataService.java class**

**A screenshot of a computer program

AI-generated content may be incorrect.**

**Step 5: Create the DataServiceTest.java class**

**A screenshot of a computer program

AI-generated content may be incorrect.**

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 6:** Verifying Interaction Order

**Scenario:**

You need to ensure that methods are called in a specific order.

Steps:

1. Create a mock object.

2. Call the methods in a specific order.

3. Verify the interaction order.

**Step 1: Create a Maven Project**

1. Open IntelliJ IDEA.
2. Click New Project(DocumentService) -Select Maven.
3. GroupId: com.example and ArtifactId:DocumentService
4. Click Finish.

**Step 2: Add Junit and Mockito to pom.xml**

A screenshot of a computer program

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Step 3:Create the interface named Document.java**

**A screenshot of a computer

AI-generated content may be incorrect.**

**Step 4:** **Create the DocumentService.java class**

A screenshot of a computer

AI-generated content may be incorrect.

**Step 5: Create the DocumentServiceTest.java class**

A screenshot of a computer program

AI-generated content may be incorrect.

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

**SLF4J LOGGING ADDITIONAL EXCERCISES**

**Exercise 2:** Parameterized Logging

**Task:** Write a Java application that demonstrates parameterized logging using SLF4J.

Step-by-Step Solution:

1. Add SLF4J and Logback dependencies to your `pom.xml` file:

<dependency>

<groupId>org.slf4j</groupId>

<artifactId>slf4j-api</artifactId>

<version>1.7.30</version>

</dependency>

<dependency>

<groupId>ch.qos.logback</groupId>

<artifactId>logback-classic</artifactId>

<version>1.2.3</version>

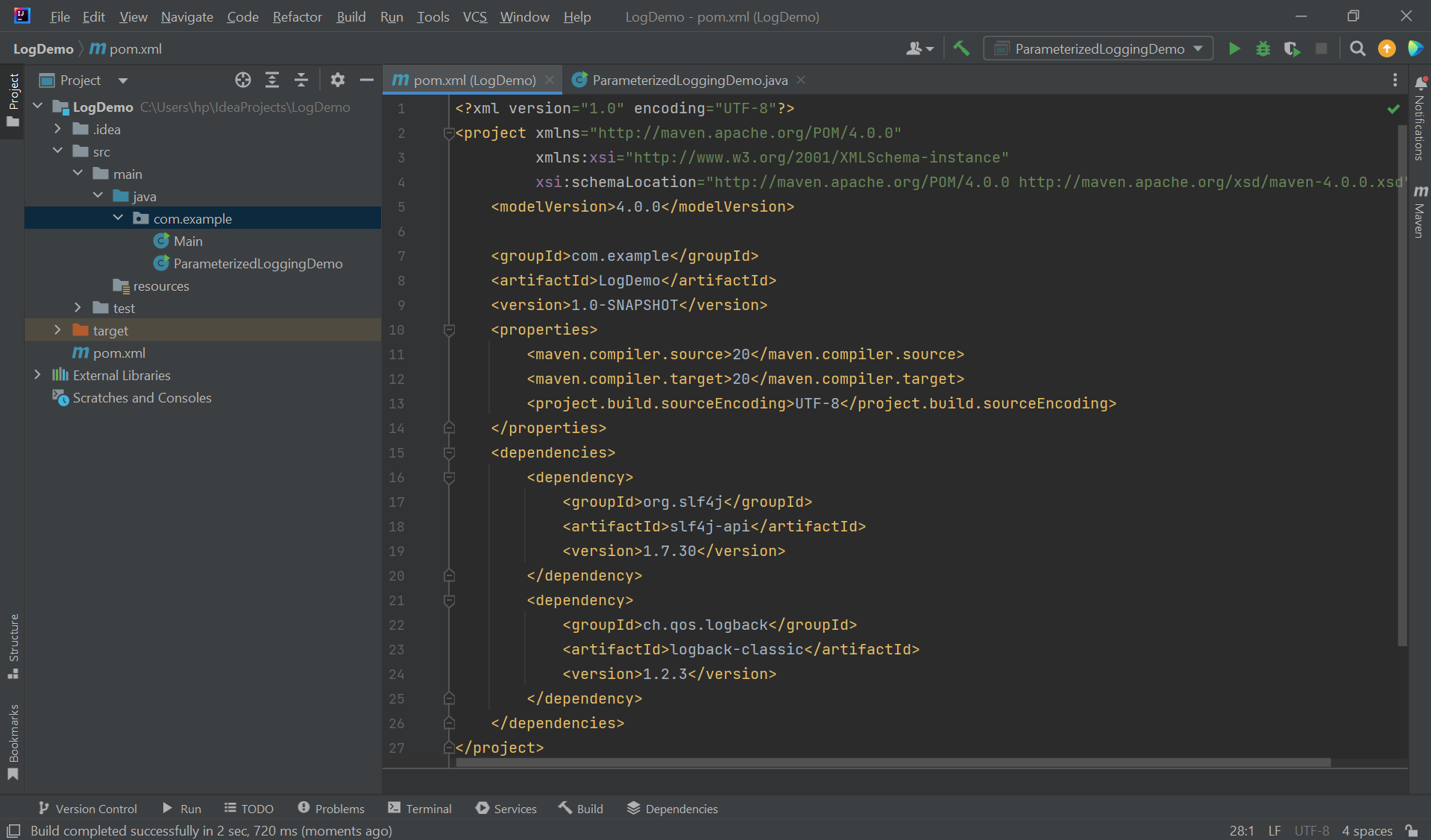
</dependency>

2. Create a Java class that uses SLF4J for parameterized logging.

**Step 1. Create a Maven Project in IntelliJ**

1. Open **IntelliJ IDEA**.
2. Click on **“New Project”**.
3. Select **“Maven”** from the left panel.
4. Uncheck **“Create from archetype”** if checked.
5. Click **Next**.
6. Enter:
   * **GroupId**: com.example
   * **ArtifactId**: LogDemo
7. Click **Finish**

**Step 2. Add dependencies to pom.xml**

****

**Step 3. Create Java class named ParameterizedLoggingDemo**

* Right-click on src/main/java.
* Go to New → Package, name it com.example.
* Right-click on the package com.example → New → Java Class.
* Name it ParameterizedLoggingDemo.

****

OUTPUT:

A screenshot of a computer

AI-generated content may be incorrect.

**Exercise 3:** Using Different Appenders

Task: Write a Java application that demonstrates using different appenders with SLF4J.

Step-by-Step Solution:

1. Add SLF4J and Logback dependencies to your `pom.xml` file:

2. Create a `logback.xml` configuration file to define different appenders:

3. Create a Java class that uses SLF4J for logging

**Step 1. Create a Maven project**

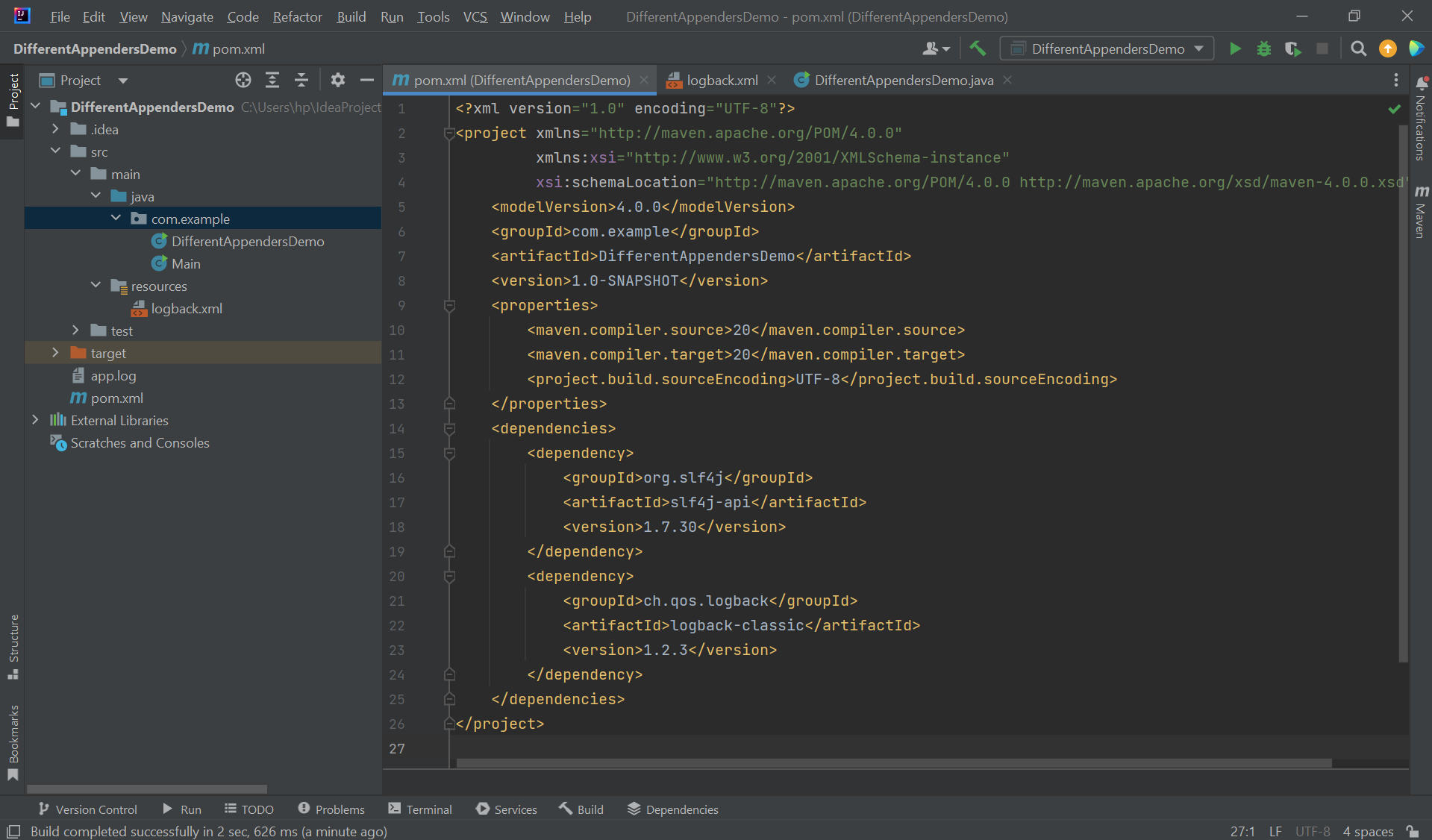
1. Open **IntelliJ IDEA**.
2. Click **“New Project”** → **Maven** → **Next**.

3.**GroupId**: com.example

**ArtifactId**: DifferentAppendersDemo

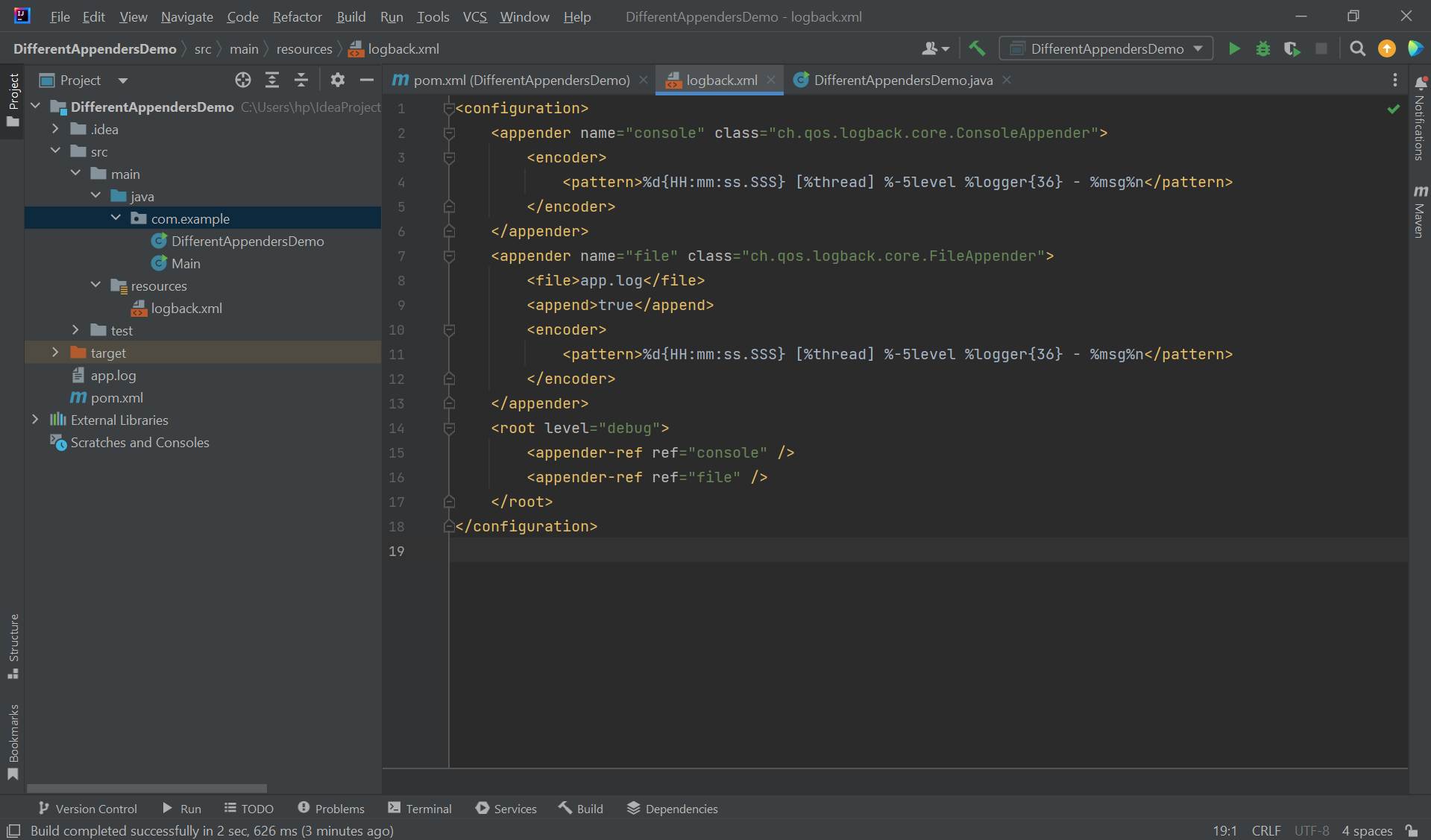
4.Click **Finish**.

**Step 2:Add dependencies to pom.xml**

****

**Step 3. Create logback.xml configuration**

* Right-click src/main/resources → New → File → name it logback.xml.



**Step 4. Create Java class for logging**

* In src/main/java/com/example create a new Java class named DifferentAppendersDemo.

A screen shot of a computer

AI-generated content may be incorrect.

OUTPUT:

A screen shot of a computer

AI-generated content may be incorrect.