Lesson 04 Demo 01

Spring JDBC

Objective: To demonstrate how to use Spring JDBC to perform CRUD operations on a MySQL database

Tool required: Eclipse IDE and MySQL

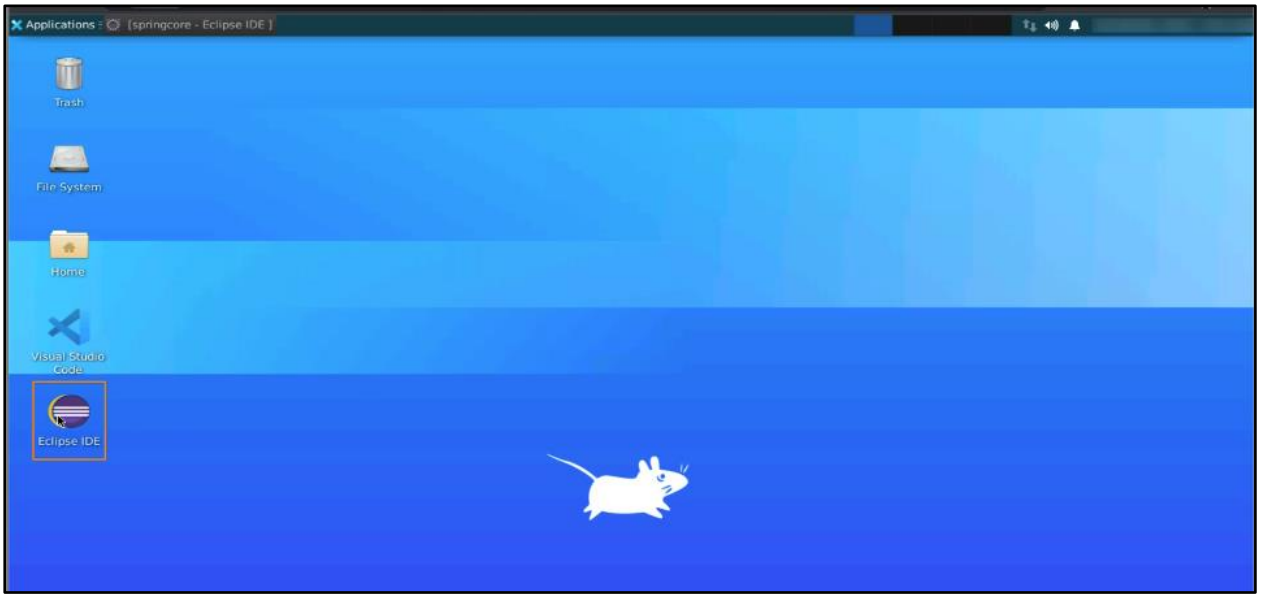
Prerequisites: None

Steps to be followed:

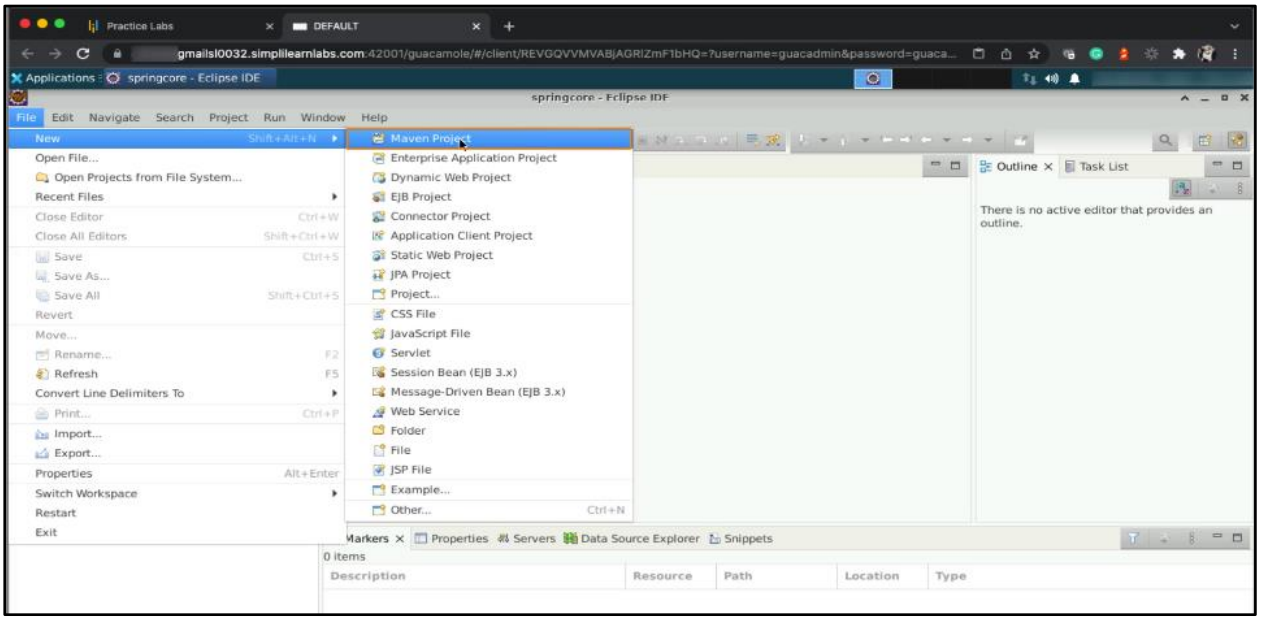
1. Setting up the Maven project and configuring the **pom.xml** file
2. Creating the model class for the database table
3. Configuring the MySQL database and creating the necessary table
4. Creating the DB class and configuring the JDBC template
5. Configuring the DB class in the XML file
6. Initializing the JDBC template using dependency injection
7. Performing the CRUD operations

Step 1: Setting up the Maven project and configuring the pom.xml file

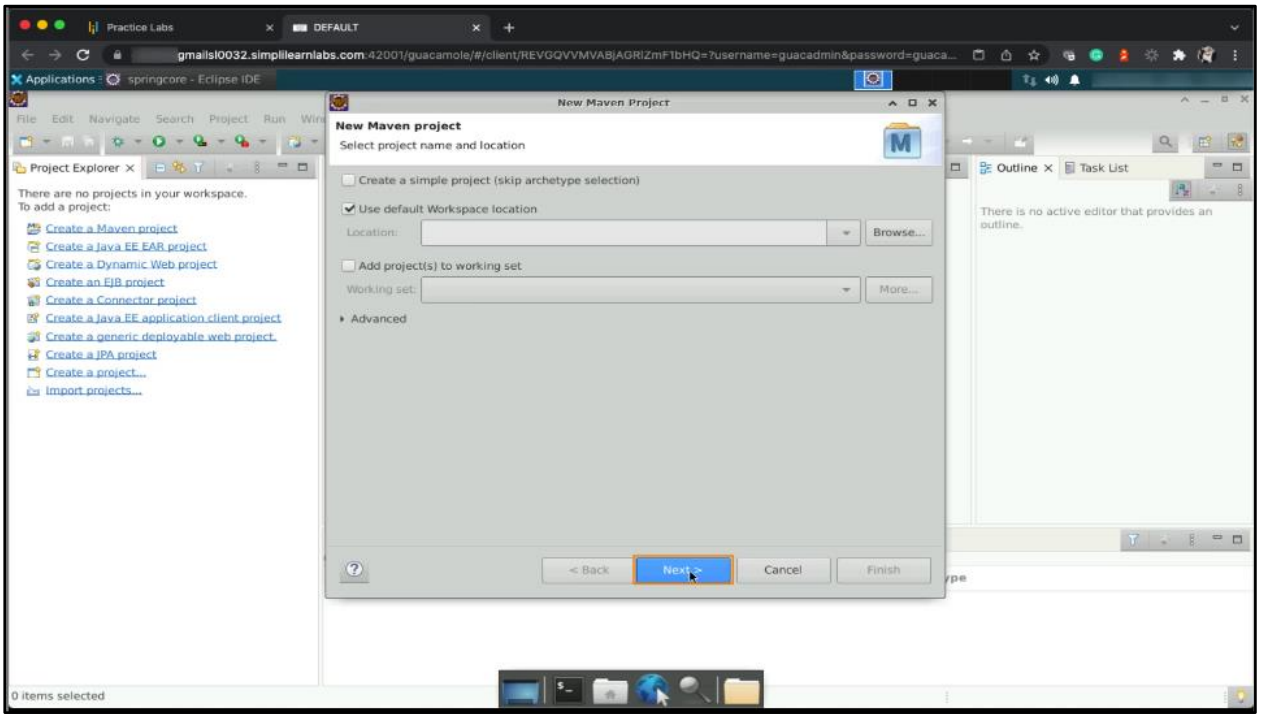
1.1 Open **Eclipse IDE**



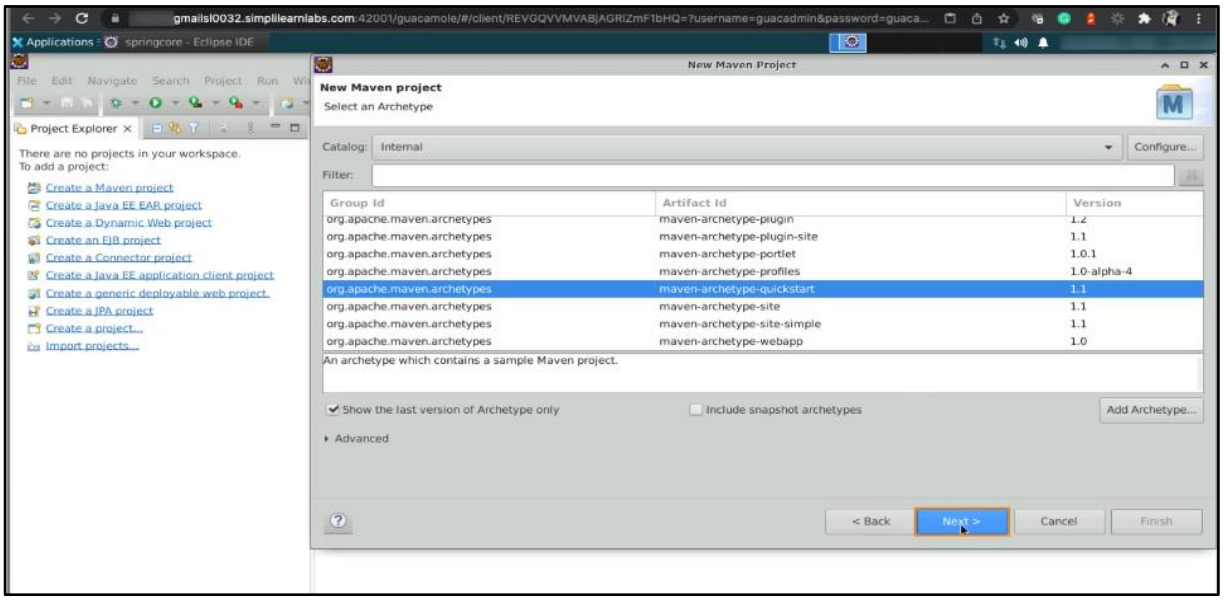
1.2 Click on **File** in the menu bar, select **New,** and choose **Maven Project**



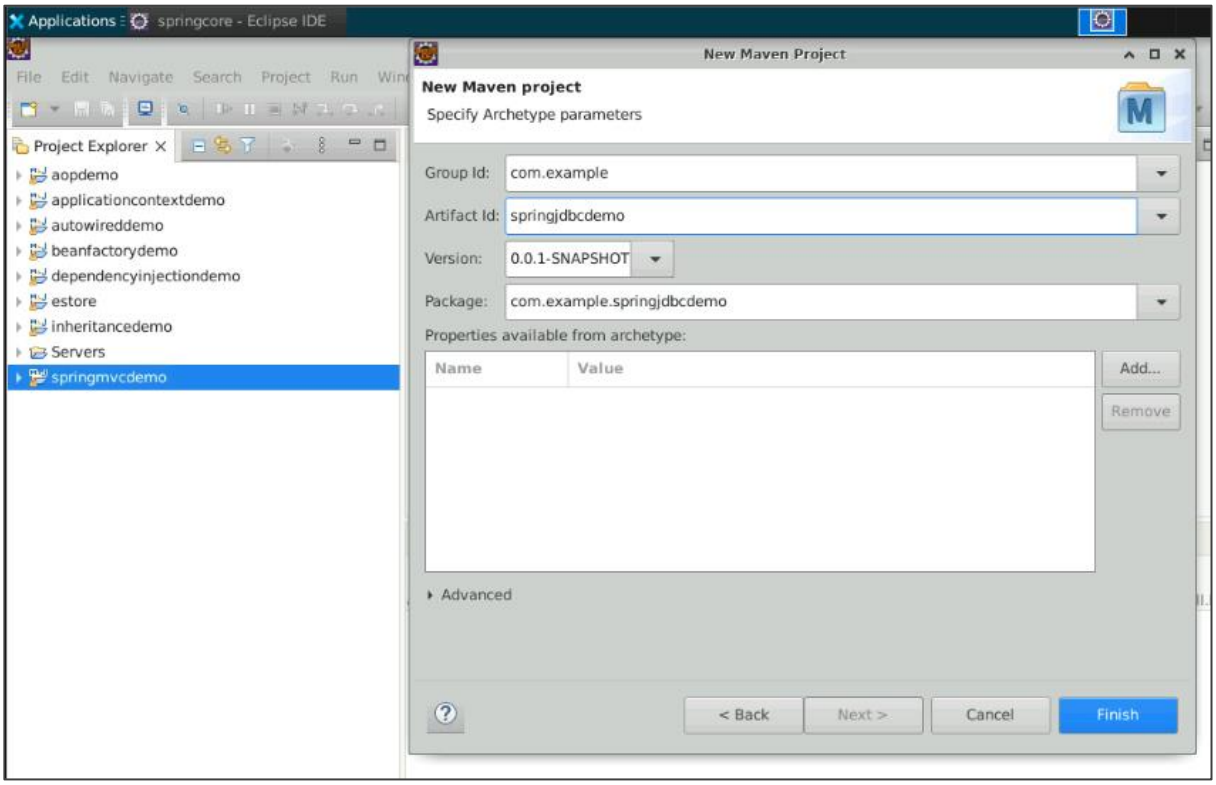
1.3 Select the default workspace location and click **Next**



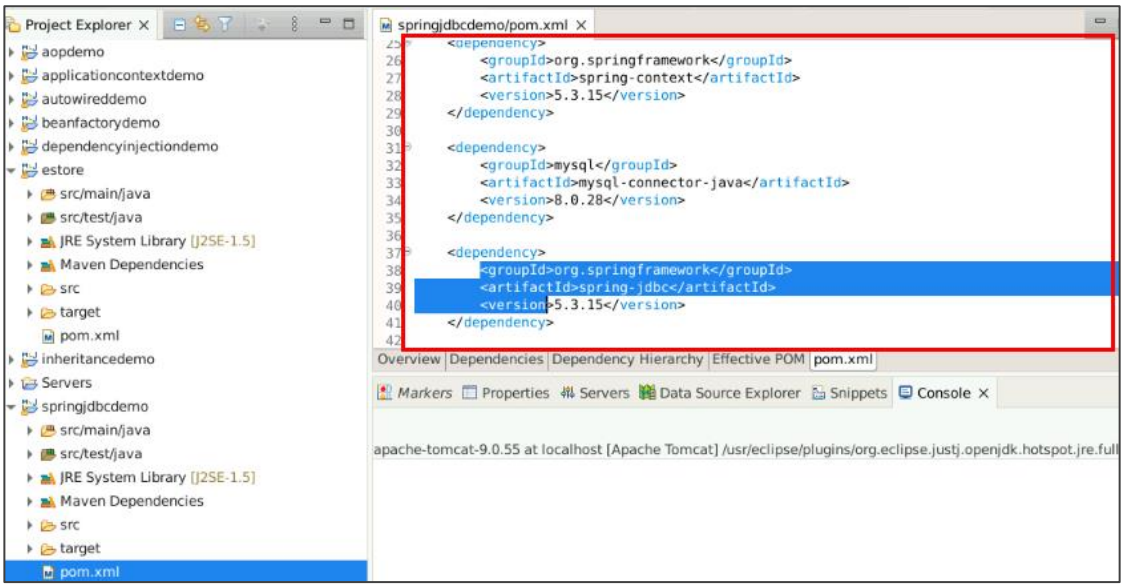
1.4 Select the **maven-archetype-quickstart** from the **Internal** catalog and click **Next**



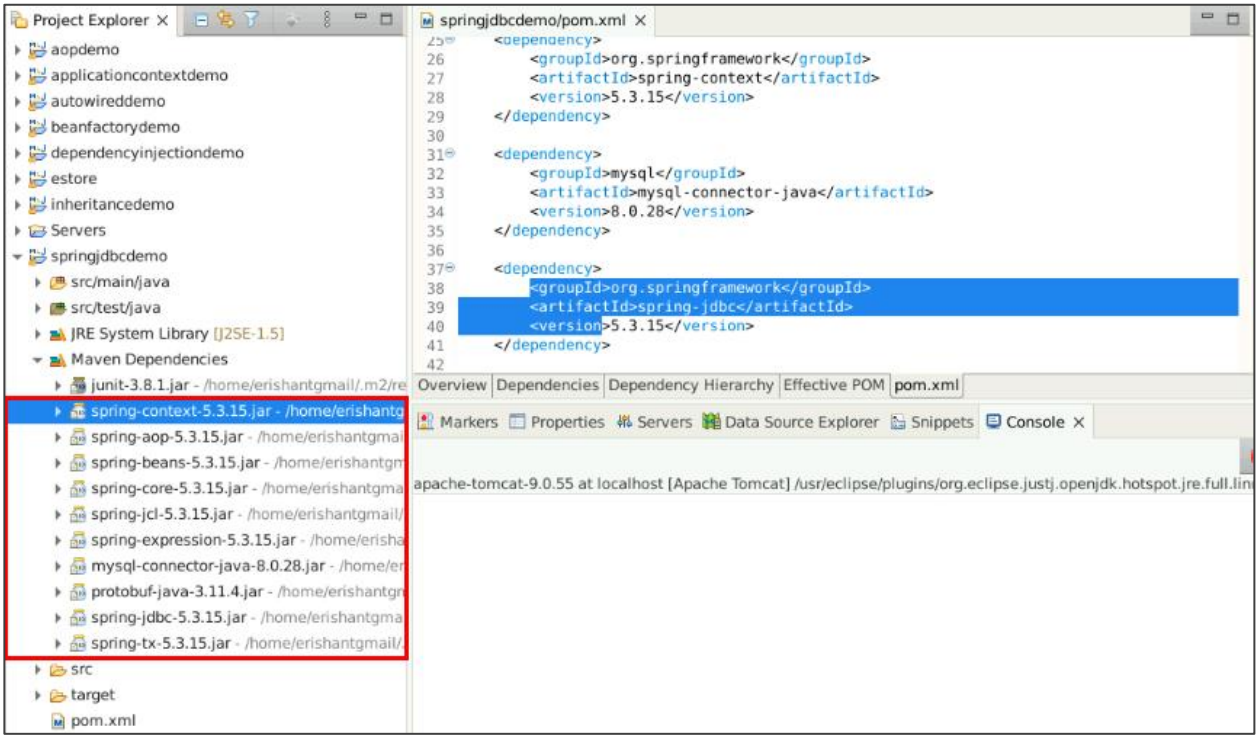
1.5 Provide the Group Id, which is typically the company's domain name in reverse order, and the Artifact Id as **springjdbcdemo**. Now, click **Finish**



1.6 Open the **pom.xml** file and add the following dependencies: **spring-context**, **mysql-connector-java**, and **spring-jdbc**

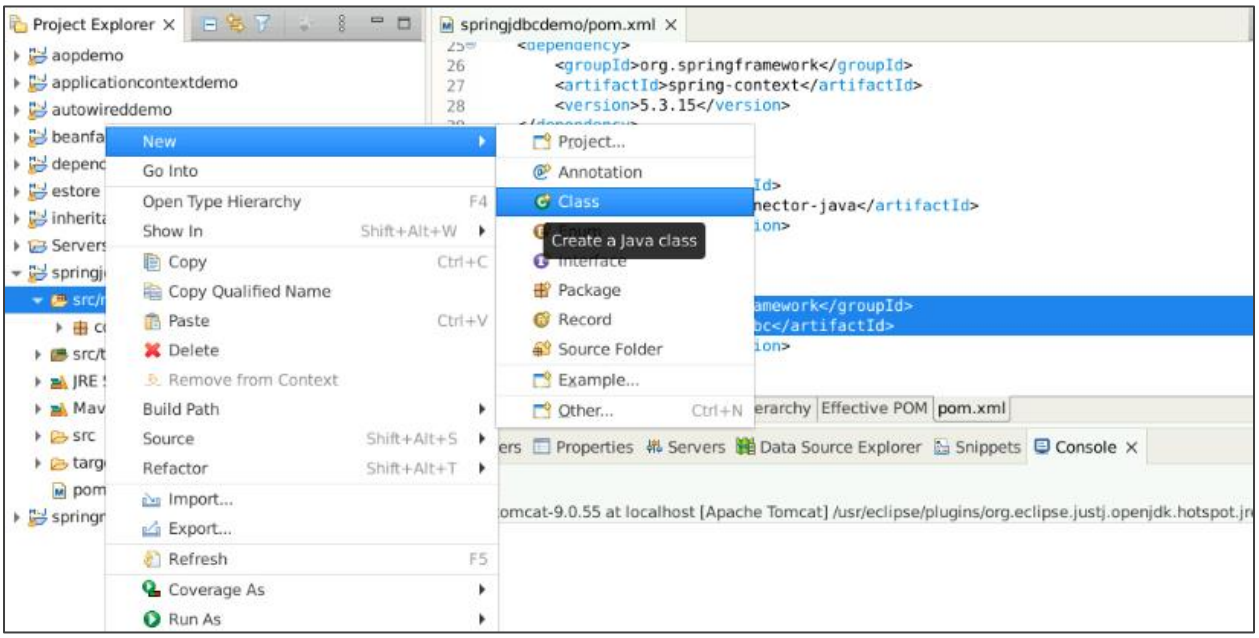


1.7 Save the **pom.xml** file and the highlighted dependencies will be added to the project:

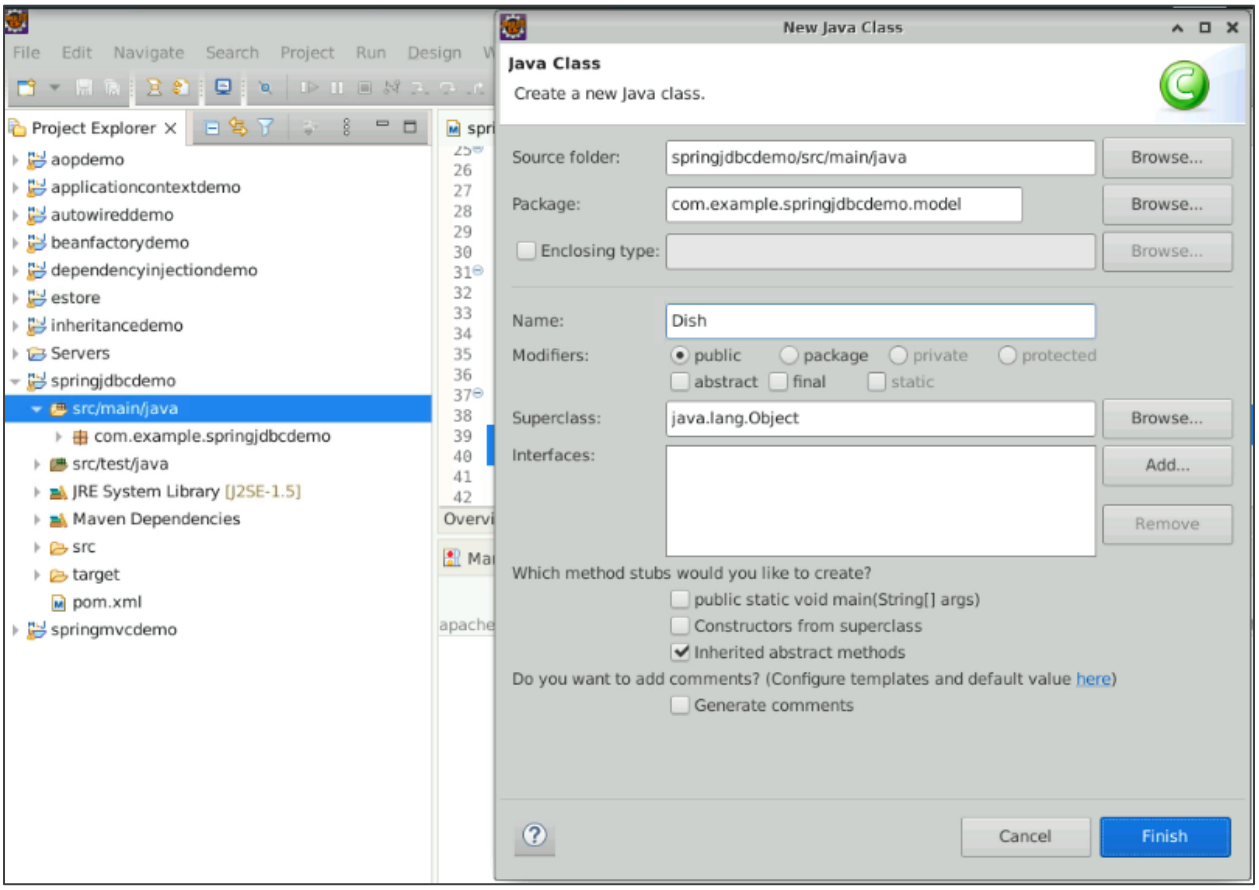


Step 2: Creating the model class for the database table

2.1 Create a new class by right-clicking **src/main/java** and selecting **New** > **Class**



2.2 Provide a name for the class, such as **Dish**. Add **.model** to the package name and click **Finish**



2.3 Define the fields for the Dish class, such as **dishId**, **name**, and **price**

springjdbcdemo/pom.xml | *Dish.java

```
1  package com.example.springjdbcdemo.model;
2
3  public class Dish {
4
5      int dishId;
6      String name;
7      int price;
8
9
10
11 }
12
```

2.4 Generate constructors, getters, setters, and a **toString()** method for the Dish class

springjdbcdemo/pom.xml | *Dish.java

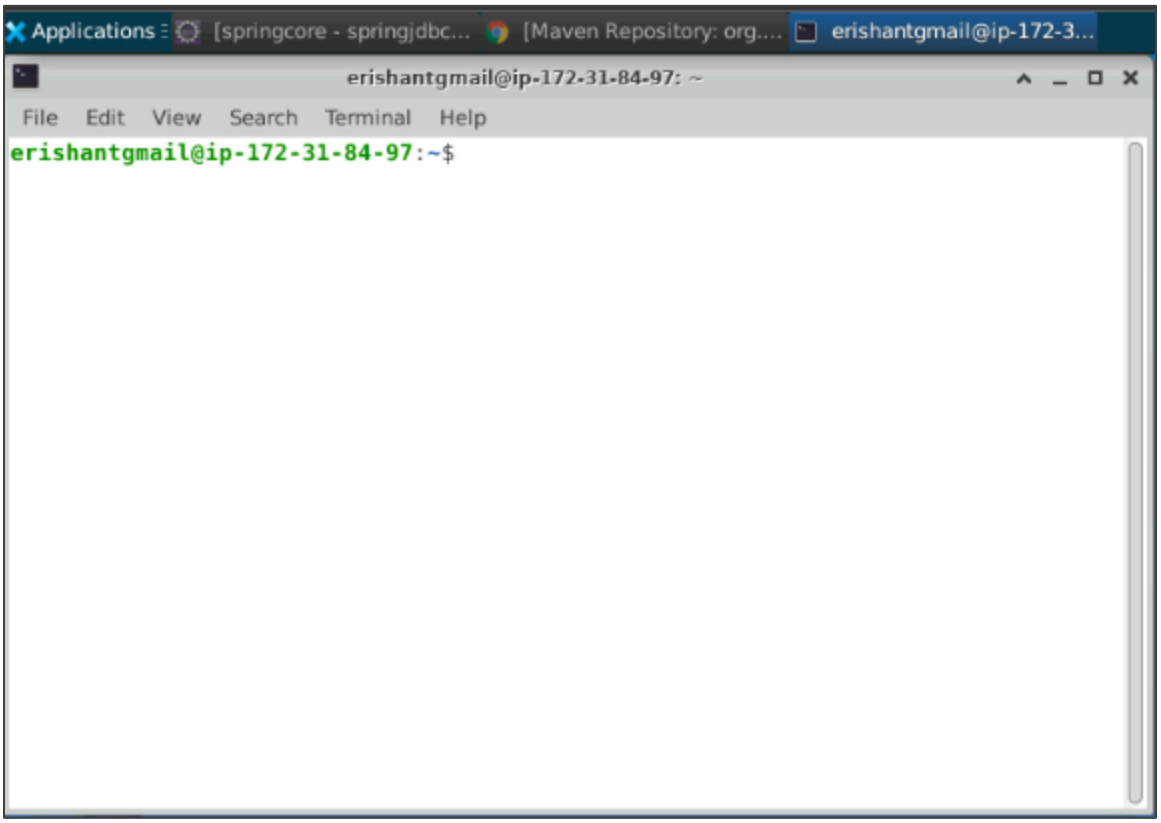
```java
package com.example.springjdbcdemo.model;

public class Dish {

    int dishId;
    String name;
    int price;

    public Dish() {
        // TODO Auto-generated constructor stub
    }
    public Dish(int dishId, String name, int price) {
        this.dishId = dishId;
        this.name = name;
        this.price = price;
    }


}
```

springjdbcdemo/pom.xml | *Dish.java

```java
        this.price = price;
    }

    public int getDishId() {
        return dishId;
    }

    public void setDishId(int dishId) {
        this.dishId = dishId;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getPrice() {
        return price;
    }

    public void setPrice(int price) {
        this.price = price;
    }

    @Override
    public String toString() {
        return "Dish [dishId=" + dishId + ", name=" + name + ", price=" + price + "]";
    }
```

Step 3: Configuring the MySQL database and creating the necessary table

3.1 Open **Terminal Emulator**



To store data from the Spring application to MySQL, the database and the tables should be created before.

3.2 Connect to the MySQL server using the following command: **mysql -u john -p**

```
erishantgmail@ip-172-31-84-97:~$ mysql -u john -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 8.0.28-0ubuntu0.20.04.3 (Ubuntu)

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql>
```

Note: Create the user **john** in the MySQL database if it doesn't exist

3.3 Run the following command to switch to the **estore** database (already created) or create a new one:

use estore;

```
erishantgmail@ip-172-31-84-97: ~
File  Edit  View  Search  Terminal  Help

Copyright (c) 2000, 2022, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| estore             |
| information_schema |
+--------------------+
2 rows in set (0.00 sec)

mysql> use estore;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql>
```

3.4 Create a new table called **Dish** with the required columns (dishId, name, price)

```
File  Edit  View  Search  Terminal  Help
+--------------------+
2 rows in set (0.00 sec)

mysql> use estore;
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Database changed
mysql> show tables;
+------------------+
| Tables_in_estore |
+------------------+
| Product          |
+------------------+
1 row in set (0.00 sec)

mysql> create table Dish(
    -> dishId int primary key auto_increment,
    -> name varchar(256),
    -> price int
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql>
```

3.5 Verify the table creation by running the following commands:

show tables;

describe Dish;

```
erishantgmail@ip-172-31-84-97: ~
File  Edit  View  Search  Terminal  Help
    -> price int
    -> );
Query OK, 0 rows affected (0.03 sec)

mysql> show tables;
+------------------+
| Tables_in_estore |
+------------------+
| Dish             |
| Product          |
+------------------+
2 rows in set (0.00 sec)

mysql> describe Dish;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| dishId | int          | NO   | PRI | NULL    | auto_increment |
| name   | varchar(256) | YES  |     | NULL    |                |
| price  | int          | YES  |     | NULL    |                |
+--------+--------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql>
```

3.6 Run the following query to check if any records are available in the **Dish** table:

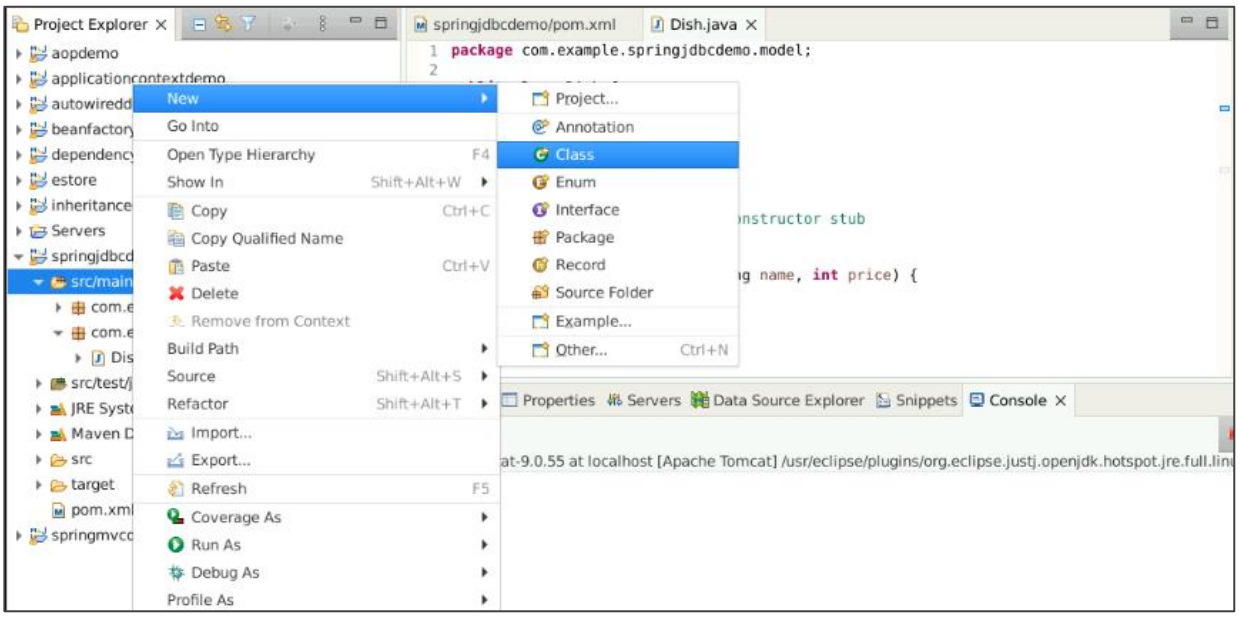
select * from Dish;

```
erishantgmail@ip-172-31-84-97: ~
File  Edit  View  Search  Terminal  Help

mysql> show tables;
+------------------+
| Tables_in_estore |
+------------------+
| Dish             |
| Product          |
+------------------+
2 rows in set (0.00 sec)

mysql> describe Dish;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| dishId | int          | NO   | PRI | NULL    | auto_increment |
| name   | varchar(256) | YES  |     | NULL    |                |
| price  | int          | YES  |     | NULL    |                |
+--------+--------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> select * from Dish;
Empty set (0.00 sec)

mysql>
```

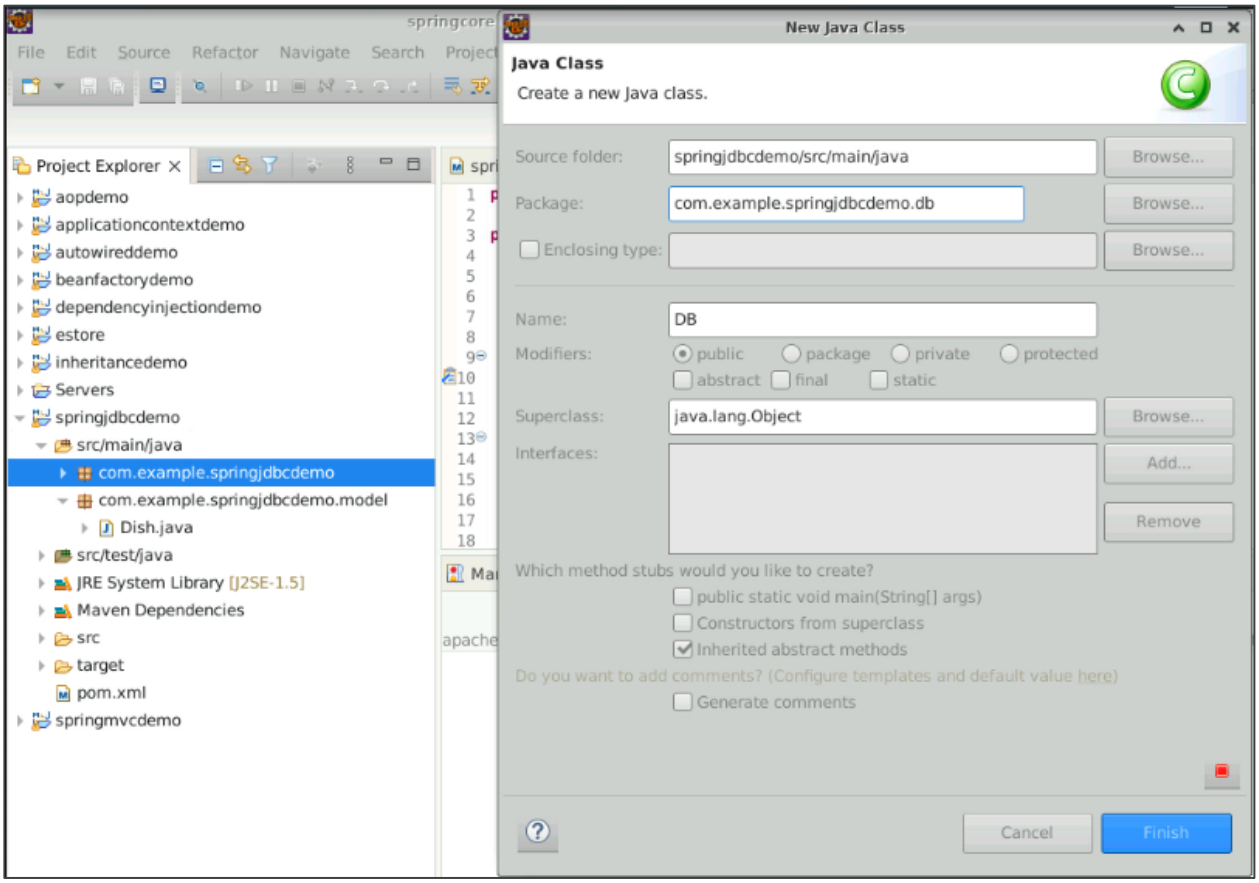
As you can see, there are no records currently available.

Step 4: Creating the DB class and configuring the JDBC template

4.1 Return to the Eclipse IDE and create a new class by right-clicking on **src/main/java** and selecting **New** > **Class**



4.2 Provide a name for the class, such as **DB**, and the package name as **com.example.springjdbcdemo.db**. Now, click **Finish**



4.3 Add two fields to the DB class: **dataSource** of type DataSource and **jdbcTemplate** of type JdbcTemplate

springjdbcdemo/pom.xml | Dish.java | DB.java

```java
package com.example.springjdbcdemo.db;

import javax.sql.DataSource;

import org.springframework.jdbc.core.JdbcTemplate;

public class DB {

    DataSource dataSource;
    JdbcTemplate jdbcTemplate;


}
```

4.4 Generate a default constructor for the class and getters, and setters for the dataSource field

springjdbcdemo/pom.xml | Dish.java | DB.java

```java
package com.example.springjdbcdemo.db;

import javax.sql.DataSource;

import org.springframework.jdbc.core.JdbcTemplate;

public class DB {

    DataSource dataSource;
    JdbcTemplate jdbcTemplate;

    public DB() {
        // TODO Auto-generated constructor stub
    }

    public DataSource getDataSource() {
        return dataSource;
    }

    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
    }

```

4.5 In the setter method for **dataSource**, initialize the **jdbcTemplate** object using the **dataSource**

springjdbcdemo/pom.xml | Dish.java | DB.java

```java
package com.example.springjdbcdemo.db;

import javax.sql.DataSource;

import org.springframework.jdbc.core.JdbcTemplate;

public class DB {

    DataSource dataSource;
    JdbcTemplate jdbcTemplate;

    public DB() {
        // TODO Auto-generated constructor stub
    }

    public DataSource getDataSource() {
        return dataSource;
    }

    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
        jdbcTemplate = new JdbcTemplate(dataSource);
    }

```

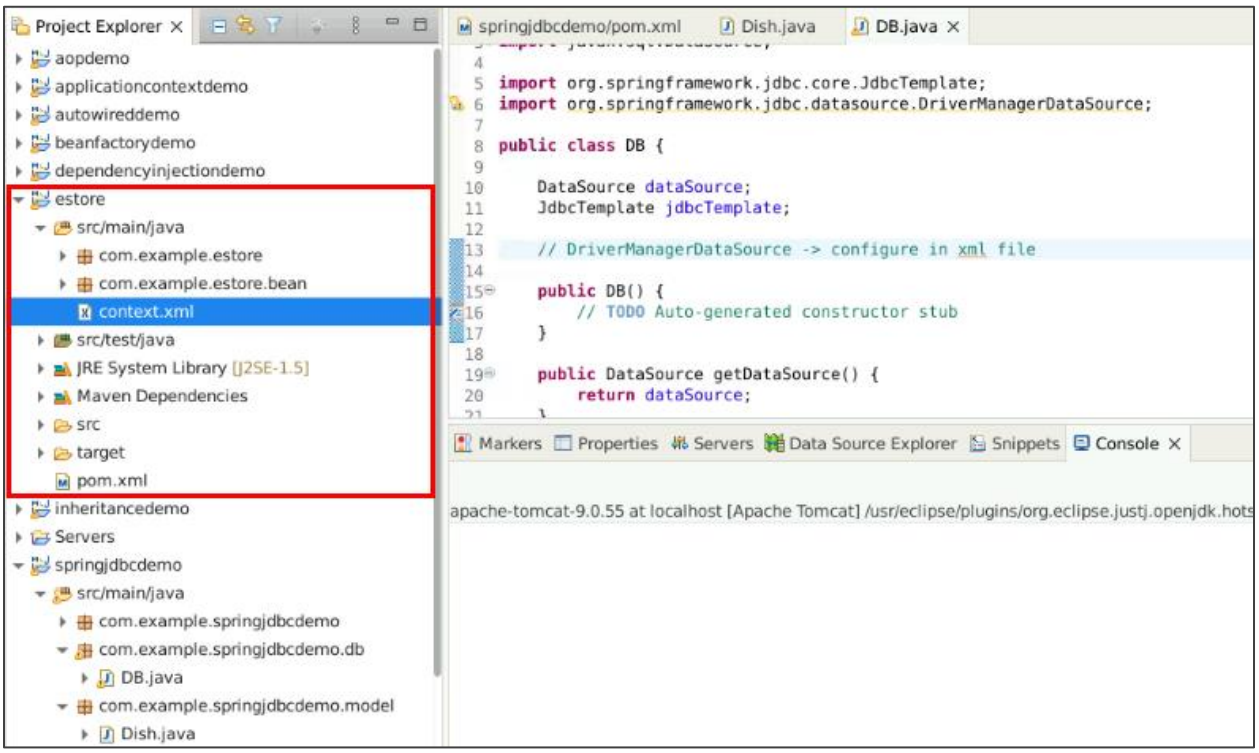
4.6 Print a message indicating that the dataSource and jdbcTemplate have been initialized

springjdbcdemo/pom.xml | Dish.java | *DB.java | context.xml

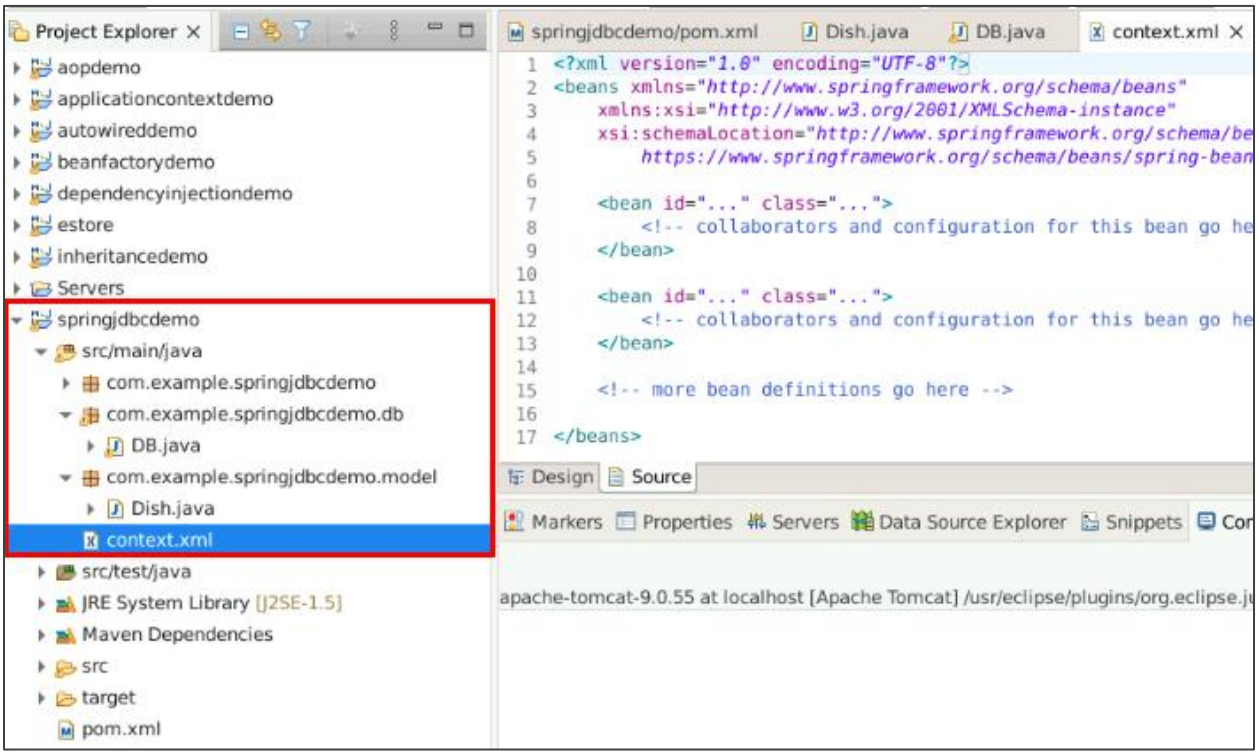
```java
package com.example.springjdbcdemo.db;

import javax.sql.DataSource;

import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

public class DB {

    DataSource dataSource;
    JdbcTemplate jdbcTemplate;

    // DriverManagerDataSource -> configure in xml file

    public DB() {
        // TODO Auto-generated constructor stub
    }

    public DataSource getDataSource() {
        return dataSource;
    }

    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
        jdbcTemplate = new JdbcTemplate(dataSource);
        System.out.println("[DB] Setter Injection on DataSource and JdbcTemplate Initializeds");
    }

```

Step 5: Configuring the DB class in the XML file

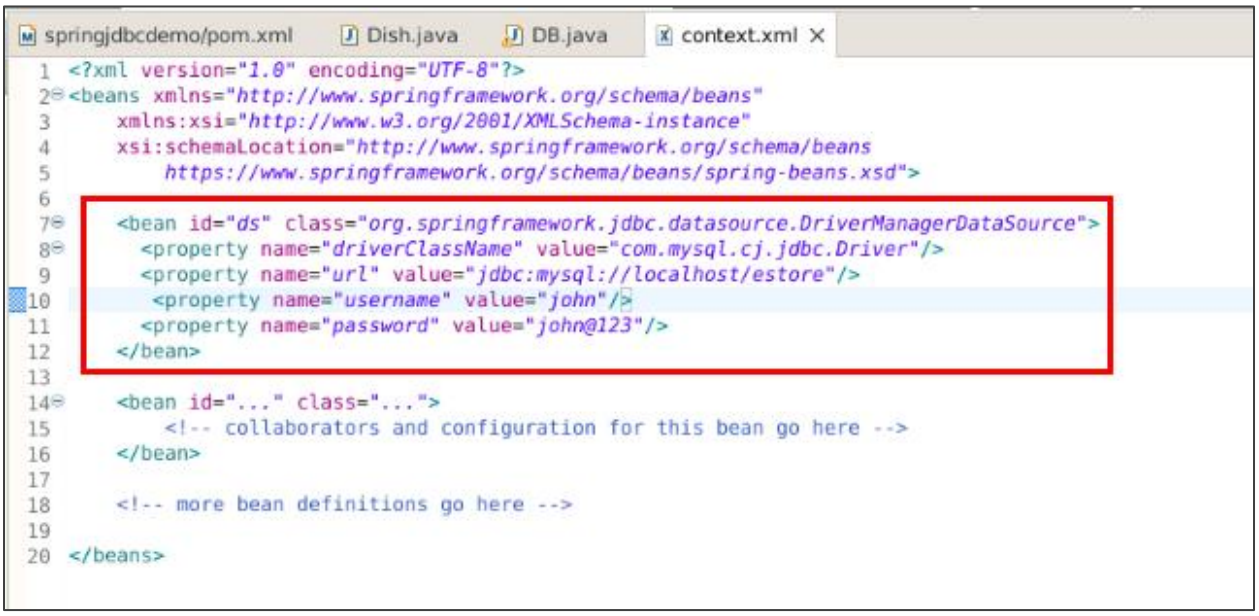
5.1 Copy the **context.xml** file from the **estore** project and paste it into the **src/main/java** package of the current project



Note: Please refer to the previous lesson demos on how to create the **estore** project



5.2 Open the **context.xml** file and configure the **dataSource** bean with the appropriate driver class, url, username, and password for your MySQL database



5.3 Next, configure another bean for the **DB** class and add a property for the **dataSource** field, linking it with the **DriverManager** bean using the **ref** attribute

```
springjdbcdemo/pom.xml   Dish.java   DB.java   context.xml ×
1  <?xml version="1.0" encoding="UTF-8"?>
2  <beans xmlns="http://www.springframework.org/schema/beans"
3      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
4      xsi:schemaLocation="http://www.springframework.org/schema/beans
5          https://www.springframework.org/schema/beans/spring-beans.xsd">
6
7      <bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
8        <property name="driverClassName" value="com.mysql.cj.jdbc.Driver"/>
9        <property name="url" value="jdbc:mysql://localhost/estore"/>
10       <property name="username" value="john"/>
11       <property name="password" value="john@123"/>
12     </bean>
13
14     <bean id="db" class="com.example.springjdbcdemo.db.DB">
15         <property name="dataSource" ref="ds"/>
16     </bean>
17
18     <!-- more bean definitions go here -->
19
20 </beans>
```

Step 6: Initializing the JDBC template using dependency injection

6.1 Open the **App.java** file

```
Project Explorer ×
aopdemo
applicationcontextdemo
autowireddemo
beanfactorydemo
dependencyinjectiondemo
estore
inheritancedemo
Servers
springjdbcdemo
  src/main/java
    com.example.springjdbcdemo
      App.java
    com.example.springjdbcdemo.db
      DB.java
    com.example.springjdbcdemo.model
      Dish.java
      context.xml
  src/test/java
  JRE System Library [J2SE-1.5]
  Maven Dependencies
  src
  target
  pom.xml

springjdbcdemo/pom.xml   Dish.java   DB.java   context.xml   App.java ×
1  package com.example.springjdbcdemo;
2
3  /**
4   * Hello world!
5   *
6   */
7  public class App
8  {
9      public static void main( String[] args )
10     {
11         System.out.println( "Hello World!" );
12     }
13 }
14

Markers   Properties   Servers   Data Source Explorer   Snippets   Console ×
apache-tomcat-9.0.55 at localhost [Apache Tomcat] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hots
```

6.2 Create a new **ApplicationContext** object using the **ClassPathXmlApplicationContext** class and provide the path to the **context.xml** file

springjdbcdemo/pom.xml | Dish.java | DB.java | context.xml | App.java

```java
package com.example.springjdbcdemo;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Spring JDBC" );
        ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");
    }
}
```

6.3 Retrieve the DB bean from the **ApplicationContext** using the **getBean**() method

springjdbcdemo/pom.xml | Dish.java | DB.java | context.xml | App.java

```java
package com.example.springjdbcdemo;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.example.springjdbcdemo.db.DB;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Spring JDBC" );
        ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");
        DB dbRef = context.getBean("db", DB.class);
    }
}
```

6.4 Run the project by clicking on the green play button

```
File  Edit  Source  Refactor  Navigate  Search  Project  Run  Window  Help

springjdbcdemo/pom.xml   Dish.java   DB.java   context.xml   App.java ×

1  package com.example.springjdbcdemo;
2
3  import org.springframework.context.ApplicationContext;
4  import org.springframework.context.support.ClassPathXmlApplicationContext;
5
6  import com.example.springjdbcdemo.db.DB;
7
8  /**
9   * Hello world!
10  *
11  */
12 public class App
13 {
14     public static void main( String[] args )
15     {
16         System.out.println( "Spring JDBC" );
17         ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");
18         DB dbRef = context.getBean("db", DB.class);
19     }
20 }
21
```

```
App.java ×                Markers  Properties  Servers  Data Source  Snippets  Console ×

onContext;                <terminated> App (6) [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hots
                          Spring JDBC
                          [DB] Setter Injection on DataSource and JdbcTemplate Initialized

Context("context.xml");
```

You will see the console log indicating that the dataSource and jdbcTemplate have been initialized, and you can further perform CRUD operations on the database.

Step 7: Performing the CRUD operations

7.1 In the **DB.java** class, create a method **insertDish()** with an SQL query to add dish records to the MySQL table. Include a print statement for the dish name that is added to the console

springjdbcdemo/pom.xml | Dish.java | DB.java × | context.xml | App.java

```java
import org.springframework.jdbc.core.JdbcTemplate;
import org.springframework.jdbc.datasource.DriverManagerDataSource;

import com.example.springjdbcdemo.model.Dish;

public class DB {

    DataSource dataSource;
    JdbcTemplate jdbcTemplate;

    // DriverManagerDataSource -> configure in xml file

    public DB() {
        // TODO Auto-generated constructor stub
    }

    public DataSource getDataSource() {
        return dataSource;
    }

    public void setDataSource(DataSource dataSource) {
        this.dataSource = dataSource;
        jdbcTemplate = new JdbcTemplate(dataSource);
        System.out.println("[DB] Setter Injection on DataSource and JdbcTemplate Initialized");
    }

    public void insertDish(Dish dish) {
        String sql = "insert into Dish values(null, ?, ?)";
        int result = jdbcTemplate.update(sql, dish.getName(), dish.getPrice());
        System.out.println("Dish "+dish.getName()+" Inserted in Table");
    }
```

7.2 In the **App.java** file, create three Dish objects with different dish names and prices, and perform the insert operation using the **insertDish()** method

springjdbcdemo/pom.xml | Dish.java | DB.java | context.xml | *App.java

```java
package com.example.springjdbcdemo;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.example.springjdbcdemo.db.DB;
import com.example.springjdbcdemo.model.Dish;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Spring JDBC" );
        ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");
        DB dbRef = context.getBean("db", DB.class);


        Dish dish1 = new Dish(0, "Noodles", 200);
        Dish dish2 = new Dish(0, "Burger", 100);
        Dish dish3 = new Dish(0, "Pizza", 500);

        dbRef.insertDish(dish1);
        dbRef.insertDish(dish2);
        dbRef.insertDish(dish3);

    }
}
```

7.3 Rerun the project

```
App.java ×                    Markers  Properties  Servers  Data Source  Snippets  Console ×
                              <terminated> App (6) [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotsp
.onContext;                   Spring JDBC
                              [DB] Setter Injection on DataSource and JdbcTemplate Initialized
                              Dish Noodles Inserted in Table
                              Dish Burger Inserted in Table
                              Dish Pizza Inserted in Table

Context("context.xml");
```

7.4 Open the terminal and execute the following query to check the records in the Dish table:

select * from Dish;

```
erishantgmail@ip-172-31-84-97: ~
File  Edit  View  Search  Terminal  Help
mysql> describe Dish;
+--------+--------------+------+-----+---------+----------------+
| Field  | Type         | Null | Key | Default | Extra          |
+--------+--------------+------+-----+---------+----------------+
| dishId | int          | NO   | PRI | NULL    | auto_increment |
| name   | varchar(256) | YES  |     | NULL    |                |
| price  | int          | YES  |     | NULL    |                |
+--------+--------------+------+-----+---------+----------------+
3 rows in set (0.00 sec)

mysql> select * from Dish;
Empty set (0.00 sec)

mysql> select * from Dish;
+--------+---------+-------+
| dishId | name    | price |
+--------+---------+-------+
|      1 | Noodles |   200 |
|      2 | Burger  |   100 |
|      3 | Pizza   |   500 |
+--------+---------+-------+
3 rows in set (0.00 sec)

mysql>
```