Lesson 01 Demo 06

Spring Autowired Annotations

Objective: To understand and implement the **@Autowired** annotation in Spring for dependency injection

Tool required: Eclipse IDE

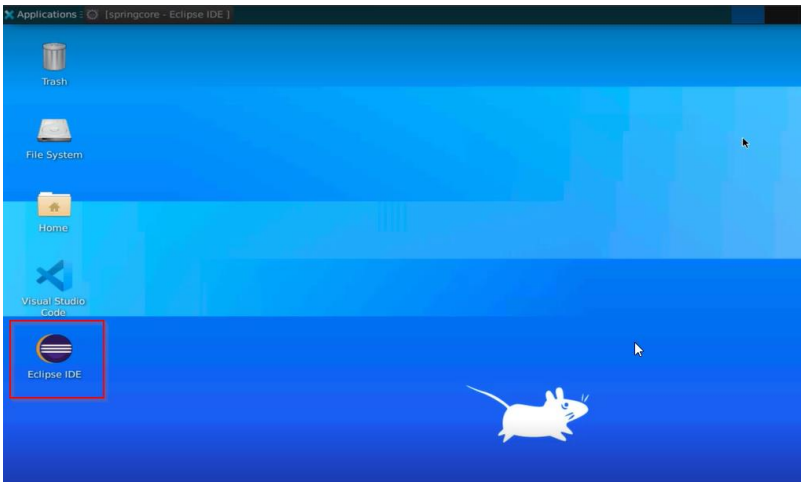
Prerequisites: None

Steps to be followed:

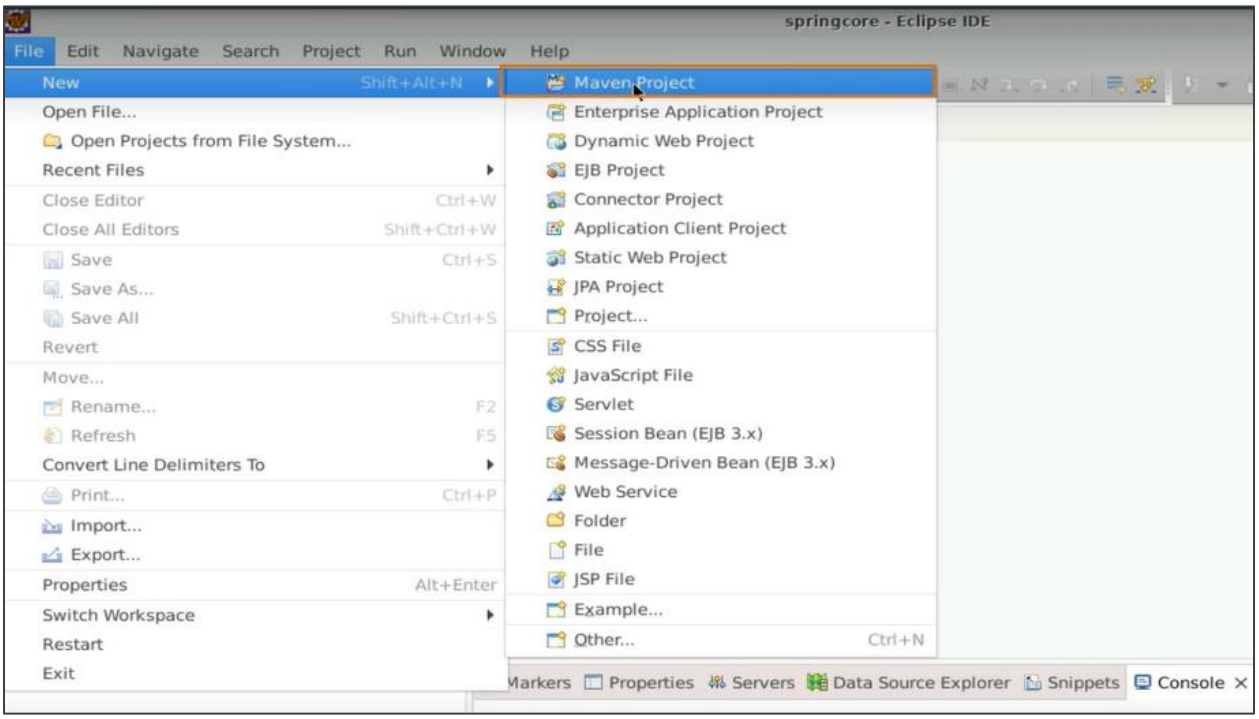
1. Creating a Maven project
2. Copying files and dependencies
3. Creating the order bean
4. Creating the user bean
5. Configuring the **context.xml** for beans
6. Writing IOC code in **App.java**

Step 1: Creating a Maven project

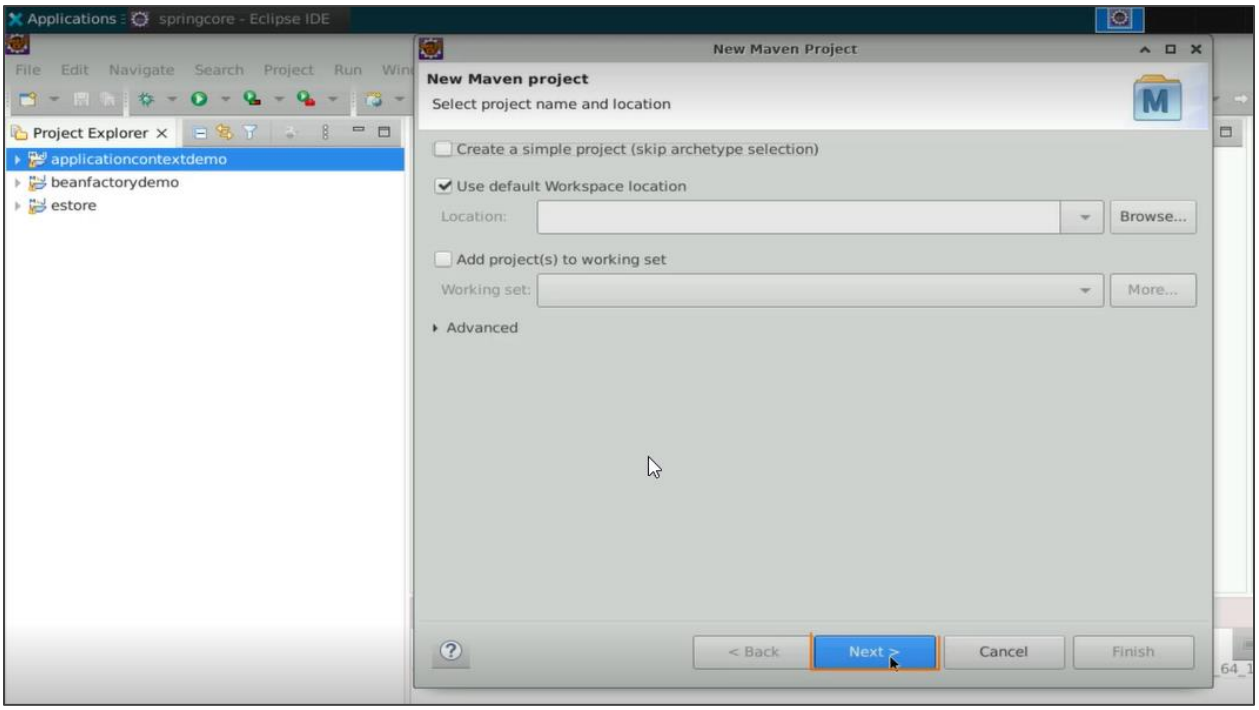
1.1 Open **Eclipse IDE**



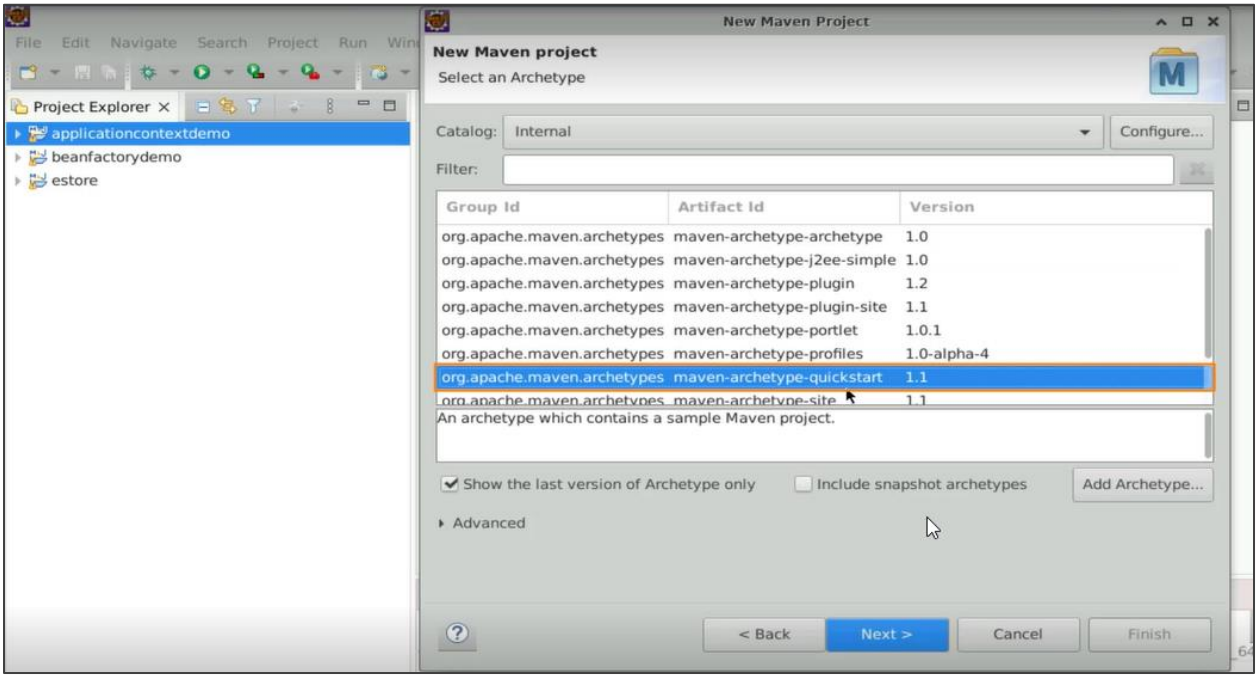
1.2 Click **File** > **New** > **Maven Project** to create a new Maven project



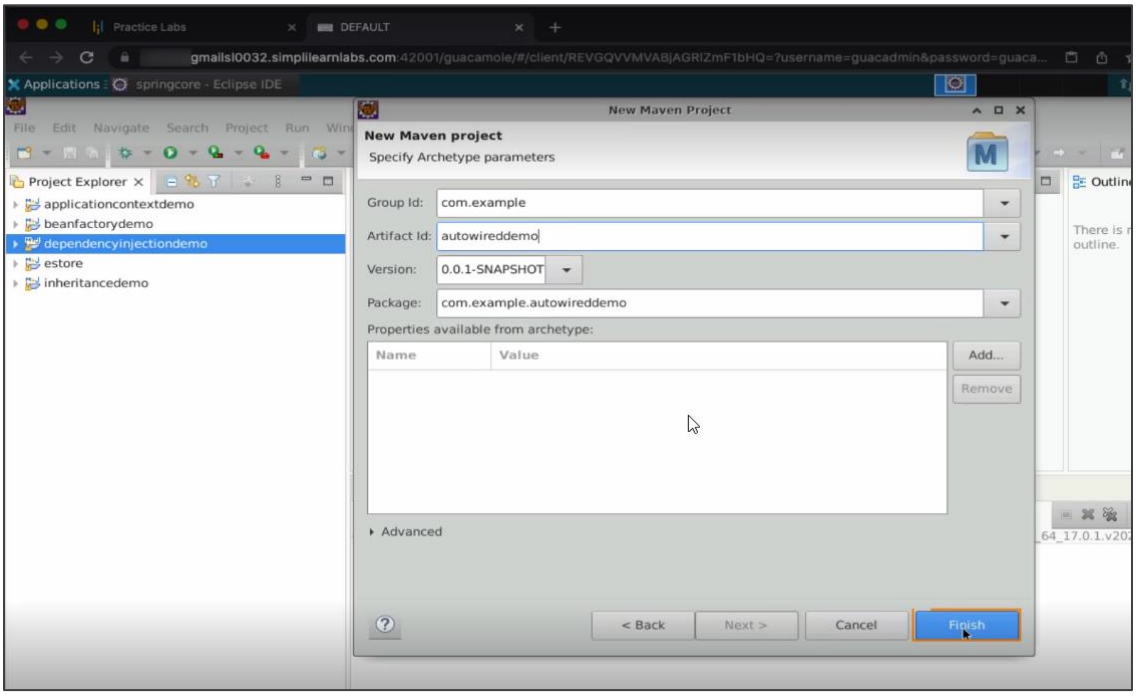
1.3 Select the default workspace location and click **Finish**



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

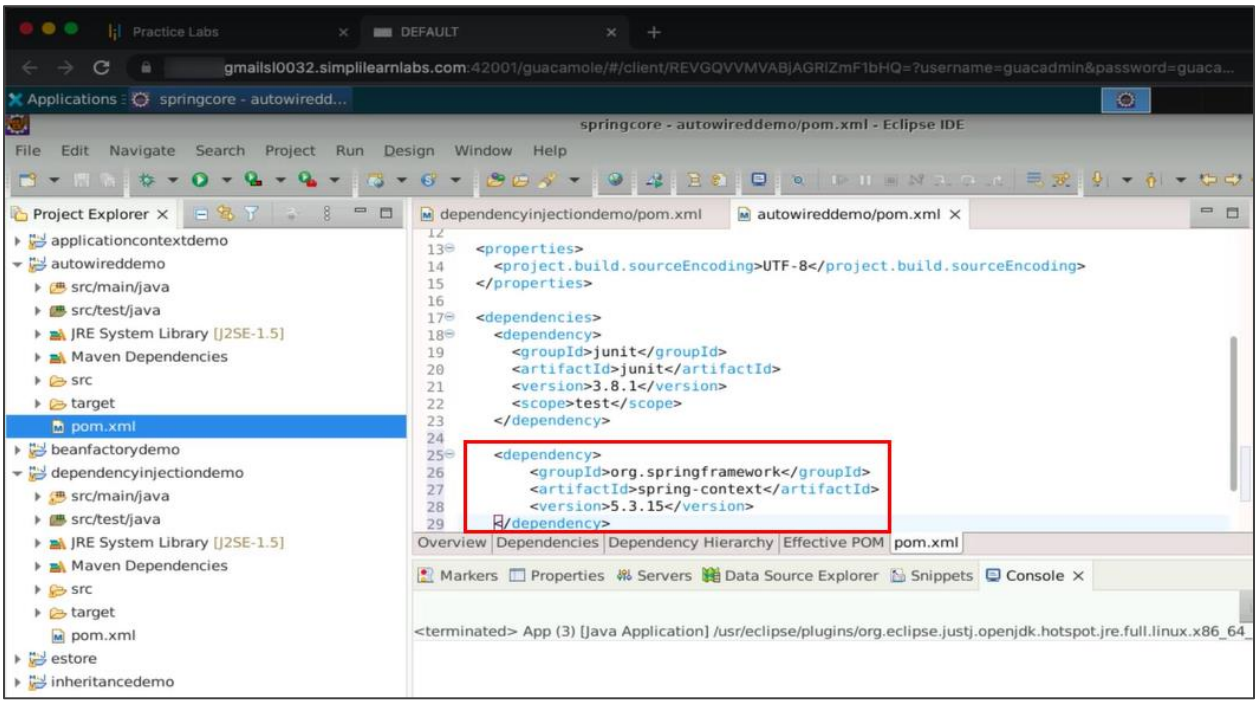


1.5 Provide the Group Id, typically the company's domain name in reverse order, and the Artifact Id as **autowireddemo**. Now, click **Finish**



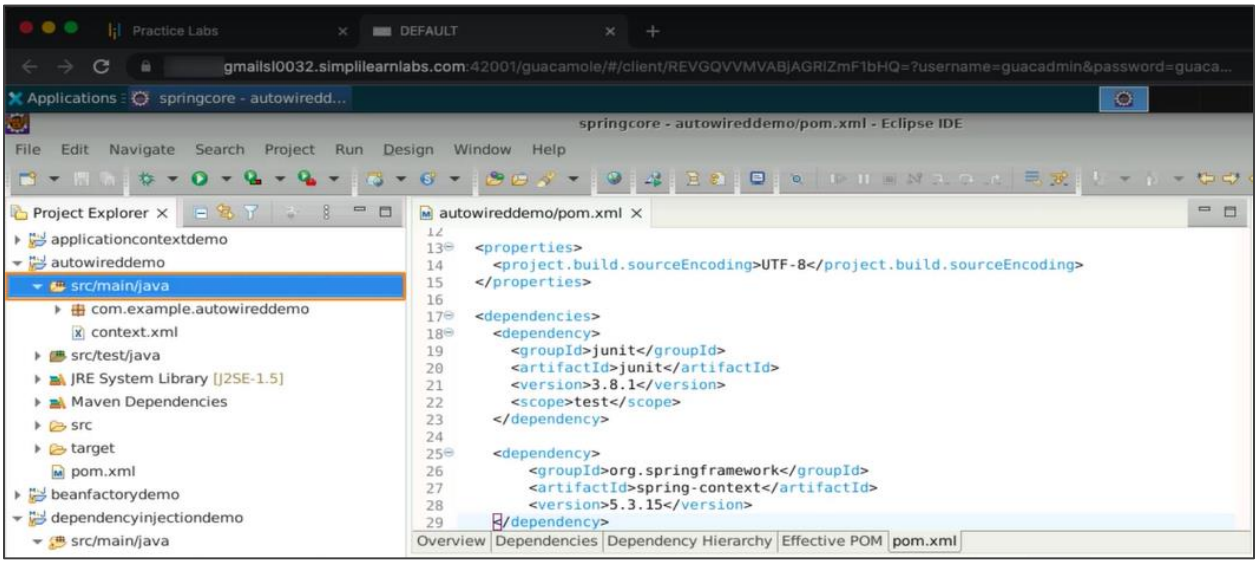
Step 2: Copying files and dependencies

2.1 Copy the **spring-context** dependency from the **pom.xml** of the **estore** project and paste it into the current project's **pom.xml**



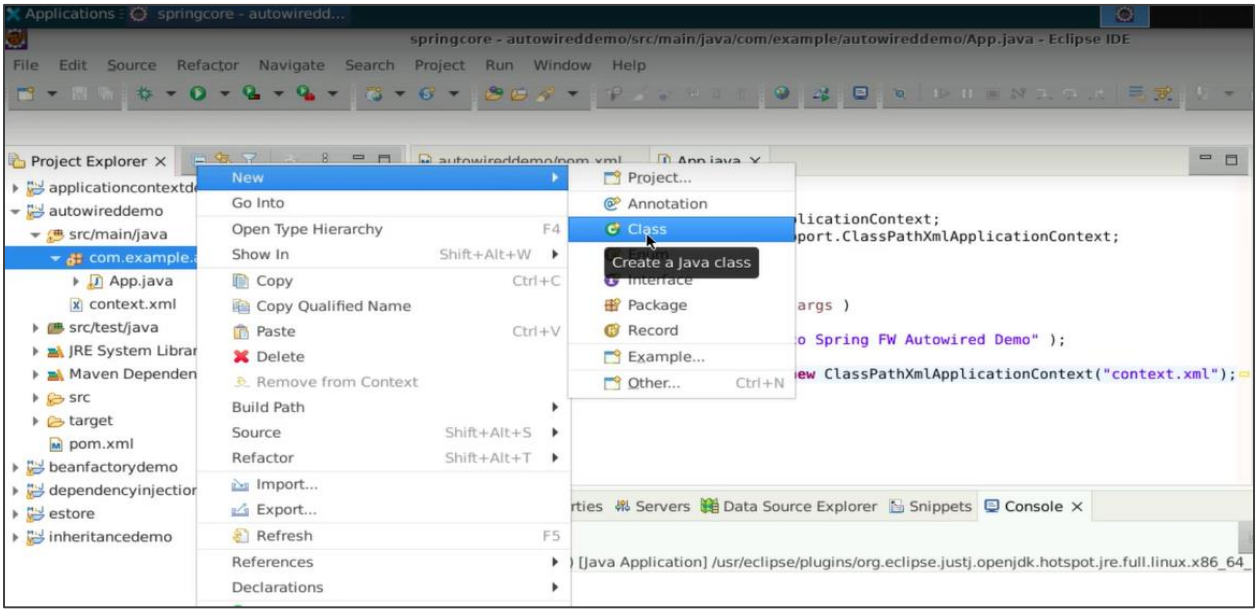
Note: Please refer to the previous demos on how to create the **estore** project

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

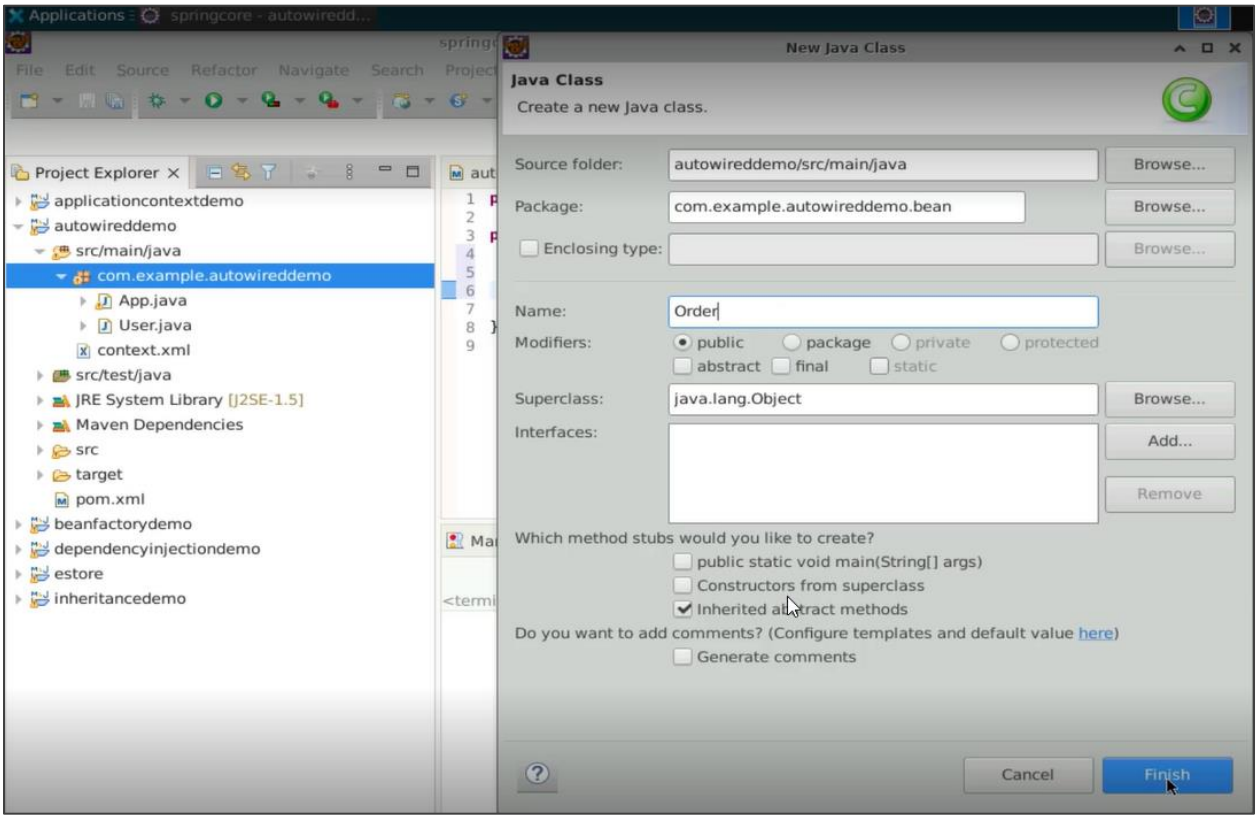


Step 3: Creating the order bean

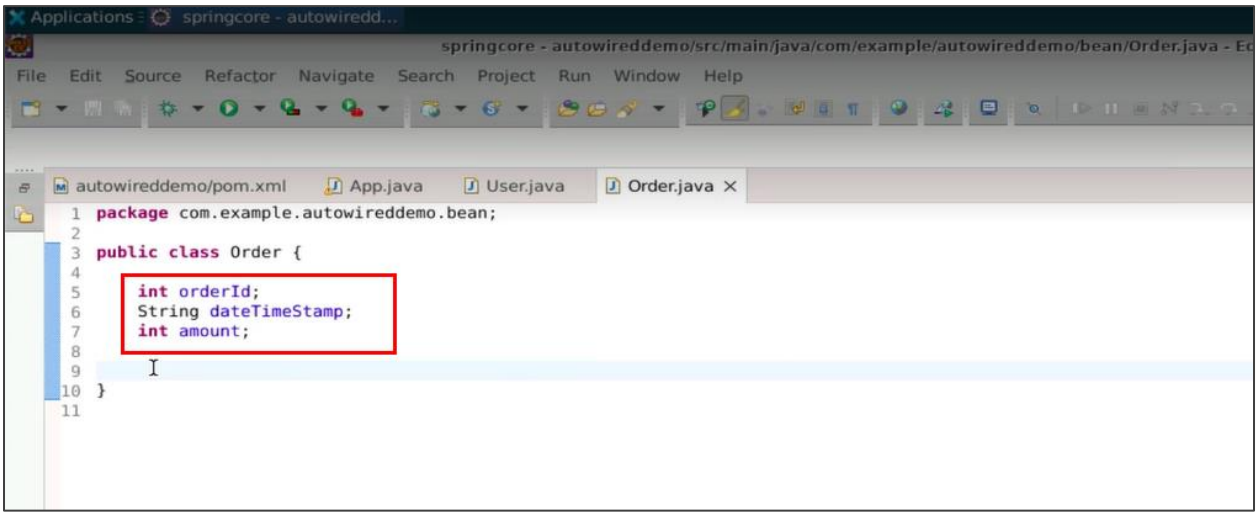
3.1 Right-click on the package and select **New** > **Class** to create a new class



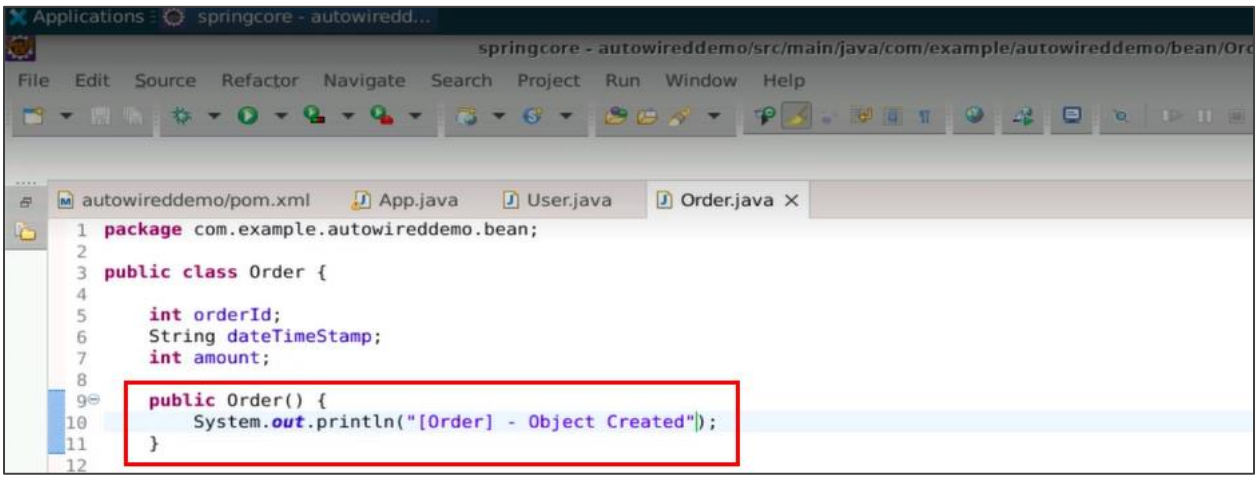
3.2 Provide a name for the class, such as **Order**, and add **.bean** to the package name. Now, click **Finish**



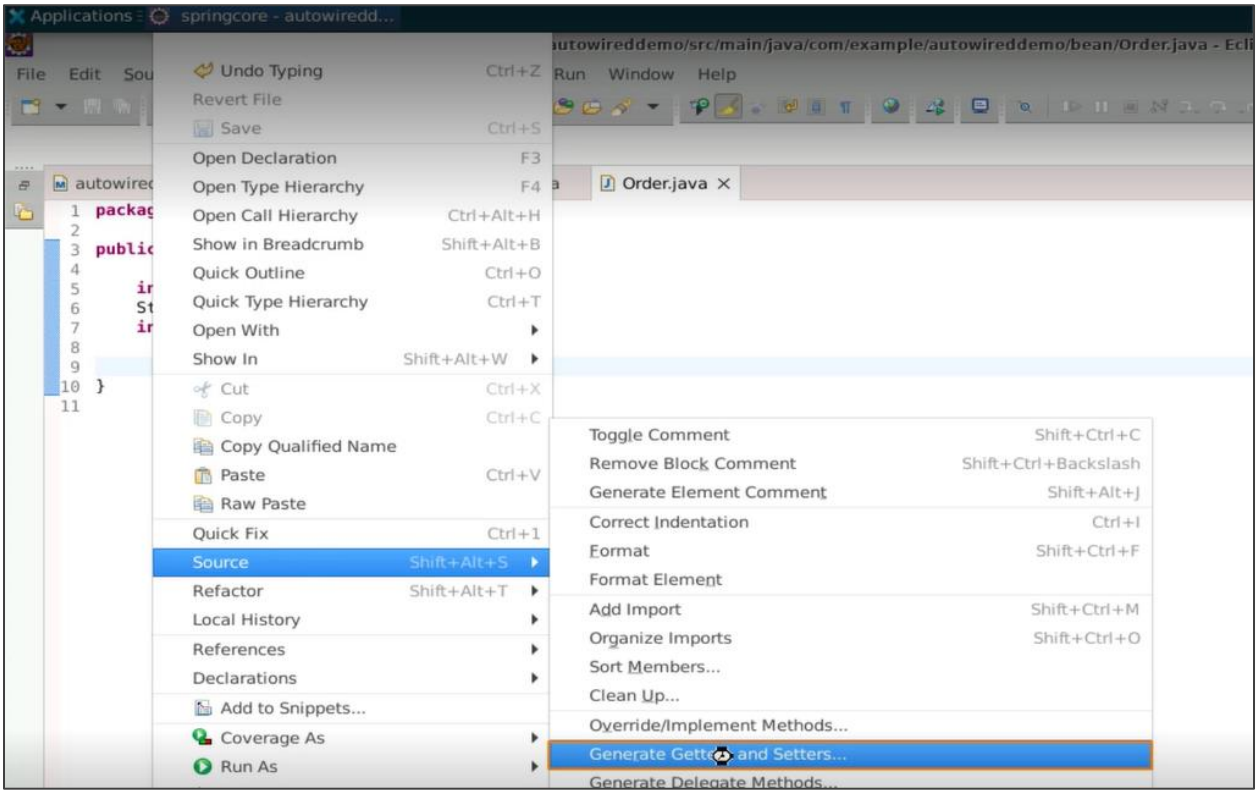
3.3 In the Order class, define attributes such as **orderId**, **dateTimeStamp**, and **amount**



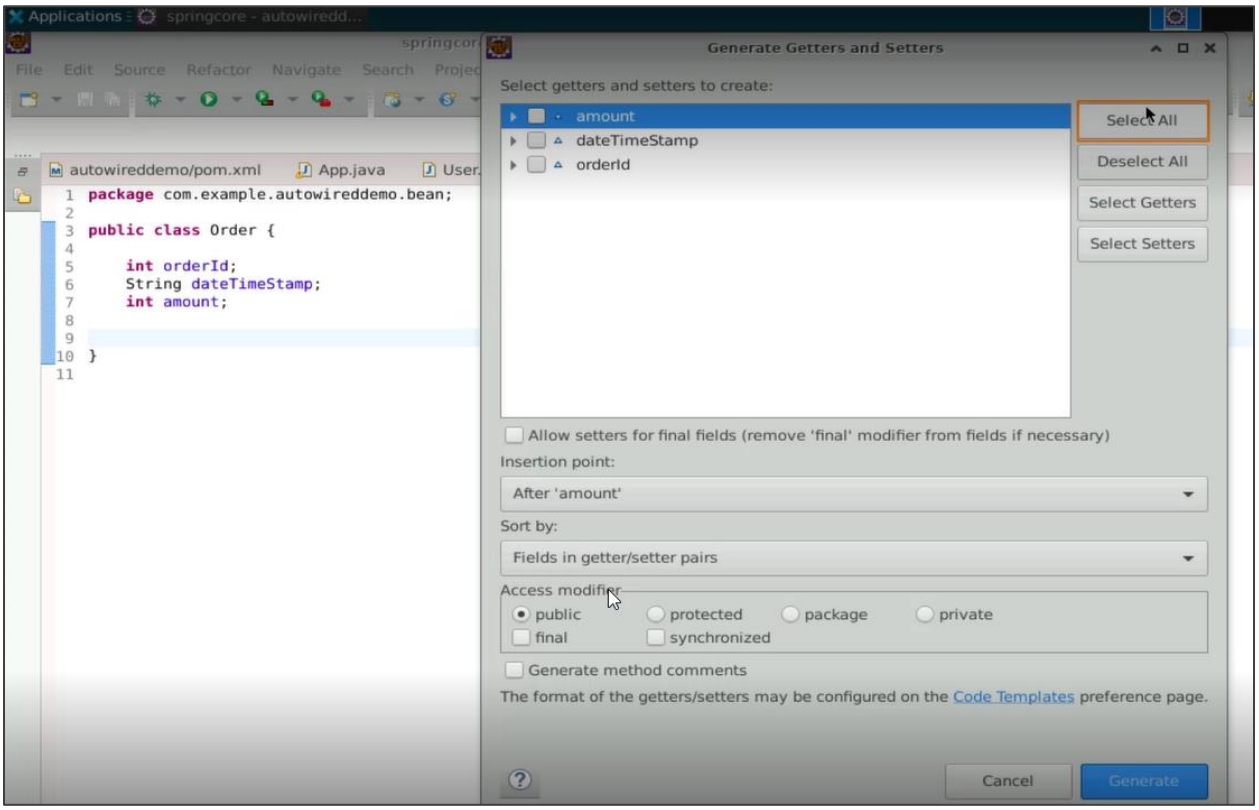
3.4 Create a default constructor for the class and add a print statement: **[Order] - Object Created**



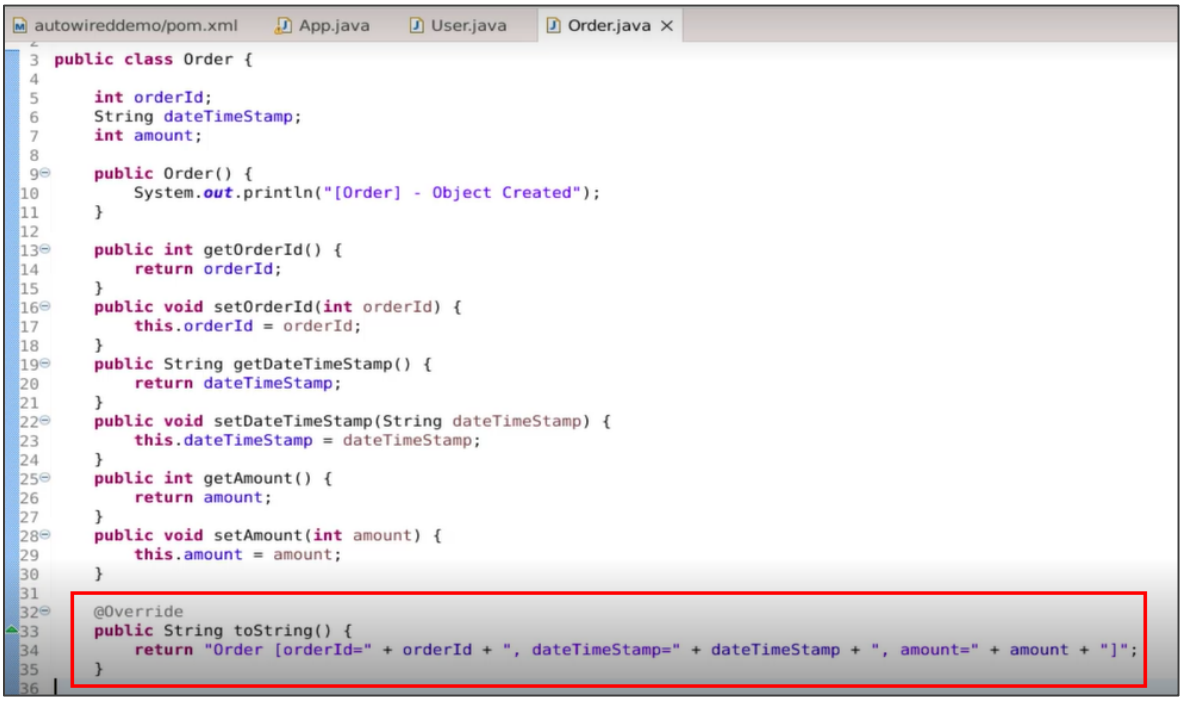
3.5 Right-click inside the **Order** class, select **Source**, and **Generate Getters and Setters**



3.6 Select all the attributes and click **Generate**

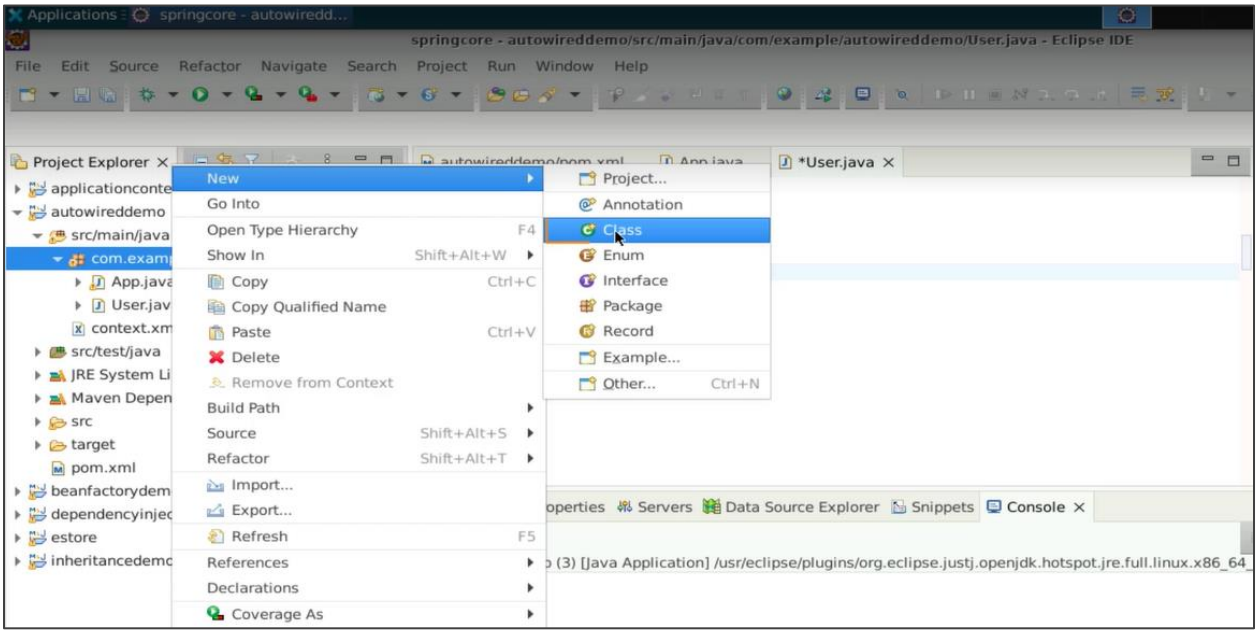


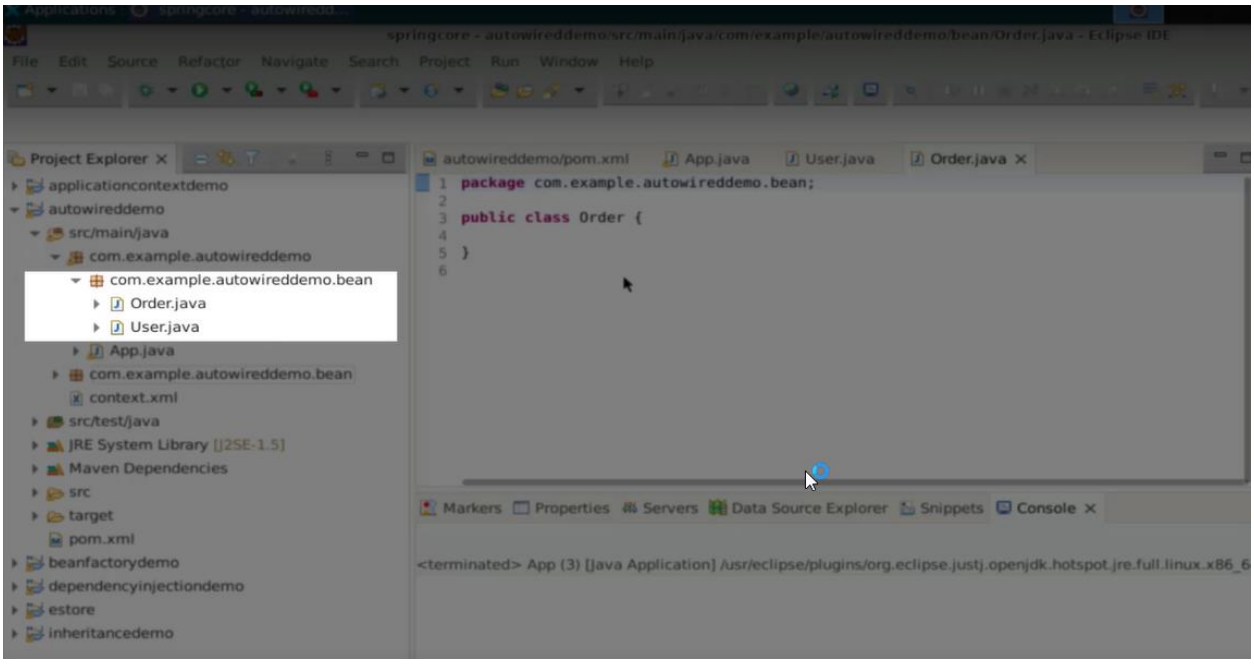
3.7 Repeat the same step to generate a **toString()** method that returns all the attributes



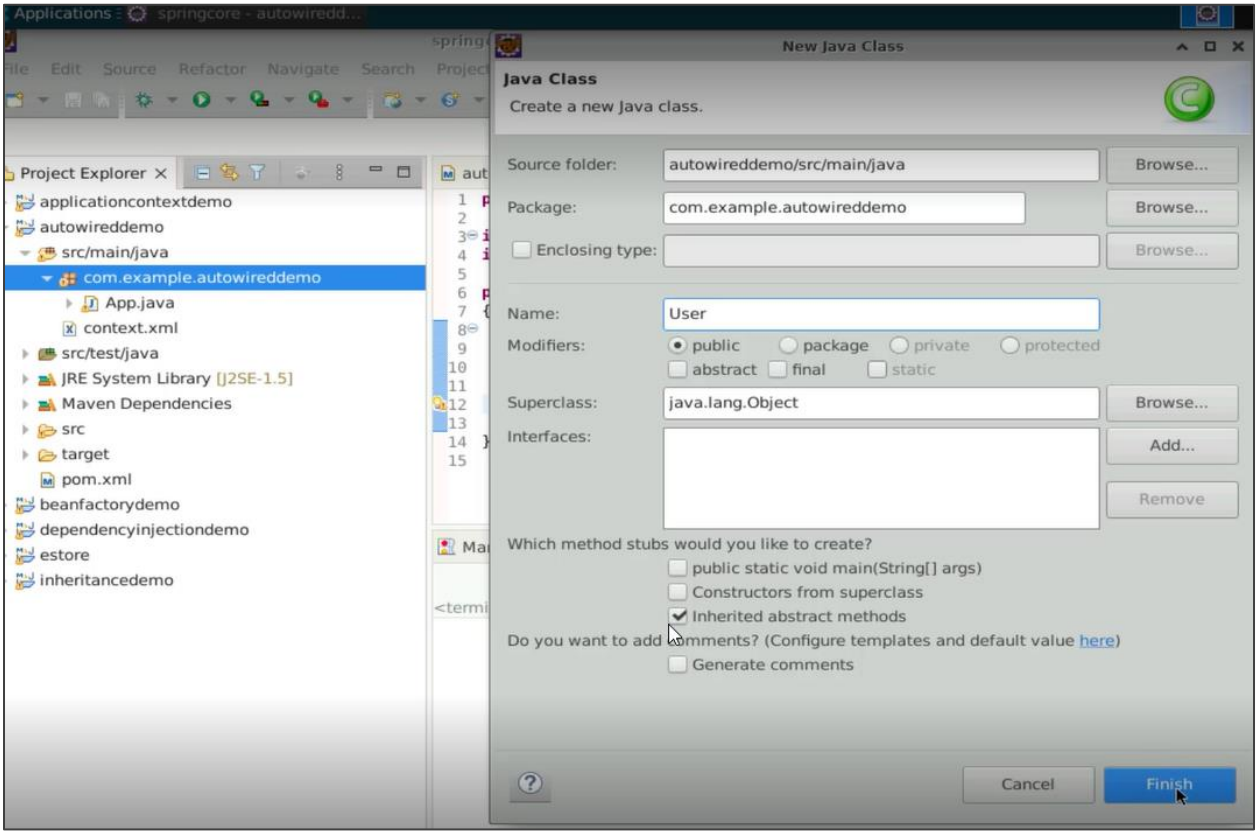
Step 4: Creating the user bean

4.1 Create a new class. Right-click on the bean package and select **New** > **Class**

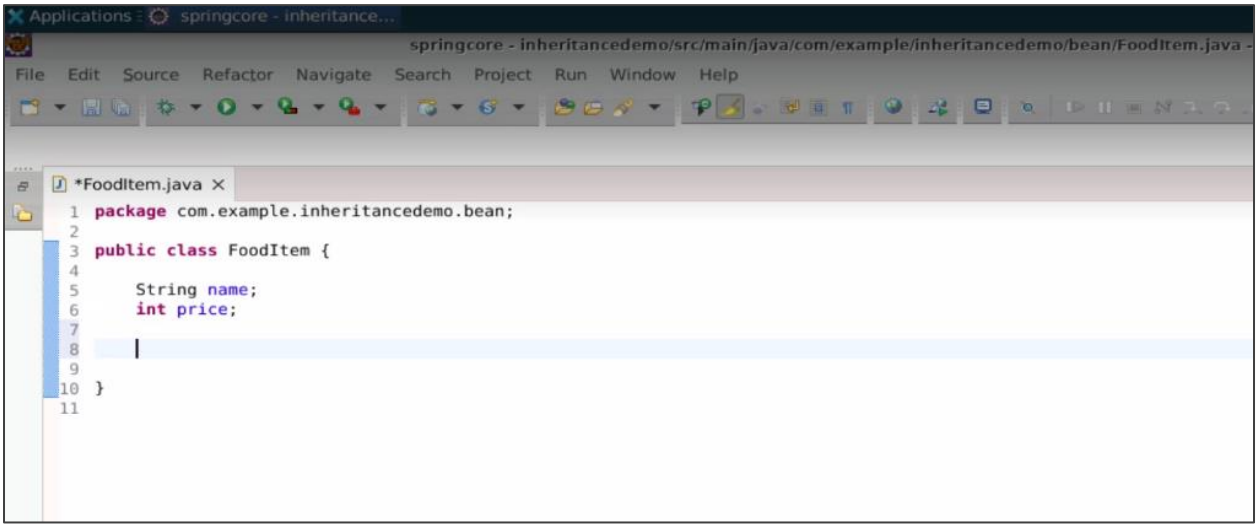




4.2 Name the class **User** and click **Finish**



4.3 Define attributes such as **name** and **price**



4.4 Instantiate the object **order** to establish a one-to-one relationship. Each user will have one order.

```
autowireddemo/pom.xml   App.java   User.java ×   Order.java
1  package com.example.autowireddemo.bean;
2
3  public class User {
4
5      String name;
6      String email;
7
8      Order order; // 1 to 1 relationship mapping
9
10 }
11
```

4.5 Create a default constructor for the class and add a print statement: **[User] - Object Created**

```
Applications : springcore - autowiredd...
springcore - autowireddemo/src/main/java/com/example/autowireddemo/bean/User.java - Ecli
File  Edit  Source  Refactor  Navigate  Search  Project  Run  Window  Help

autowireddemo/pom.xml   App.java   *User.java ×   Order.java
1  package com.example.autowireddemo.bean;
2
3  public class User {
4
5      String name;
6      String email;
7
8      Order order; // 1 to 1 relationship mapping
9
10     public User() {
11         System.out.println("[User] - Object Created");
12     }
13
14 }
15
```

4.6 Create a parameterized constructor for the class with the field **order** as input

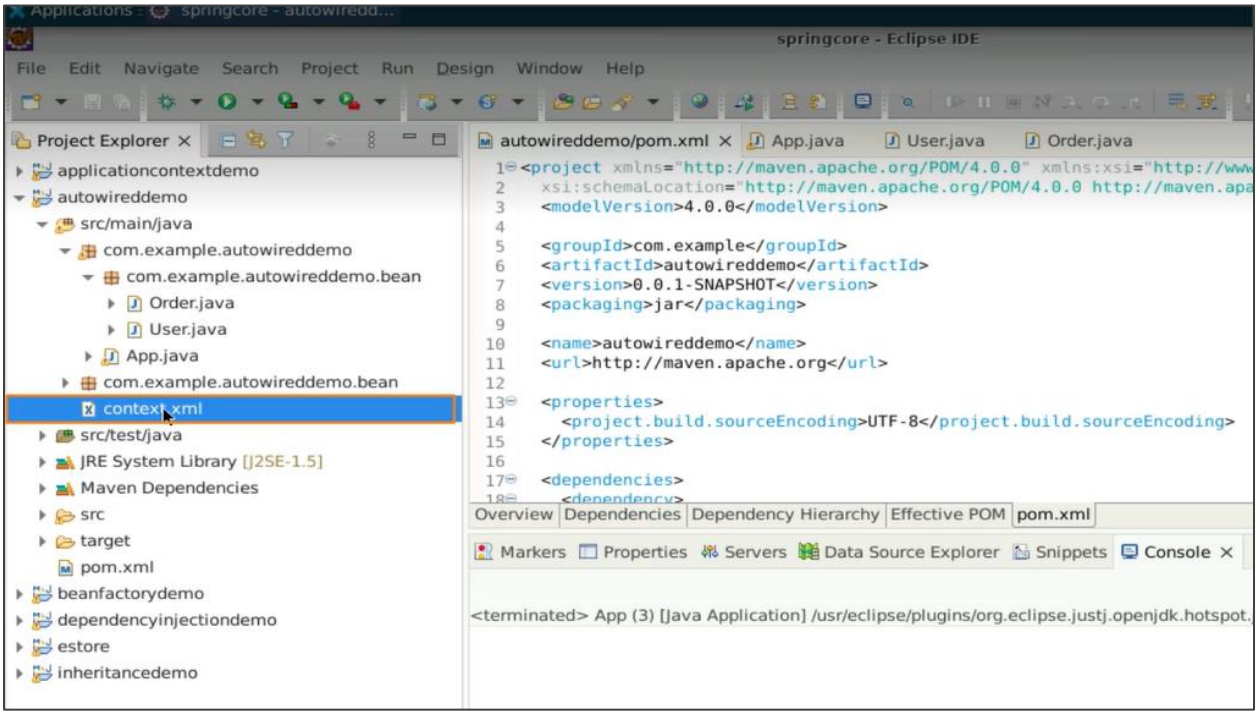
```java
package com.example.autowireddemo.bean;

public class User {

    String name;
    String email;

    Order order; // 1 to 1 relationship mapping

    public User() {
        System.out.println("[User] - Object Created");
    }

    public User(Order order) {
        System.out.println("[User] - CONSTRUCTOR INJECTION - Object Created with Parametrized Constructor having Order as Input")
        this.order = order;
    }


}
```

4.7 Generate getters and setters for the attributes and a **toString()** method for the class

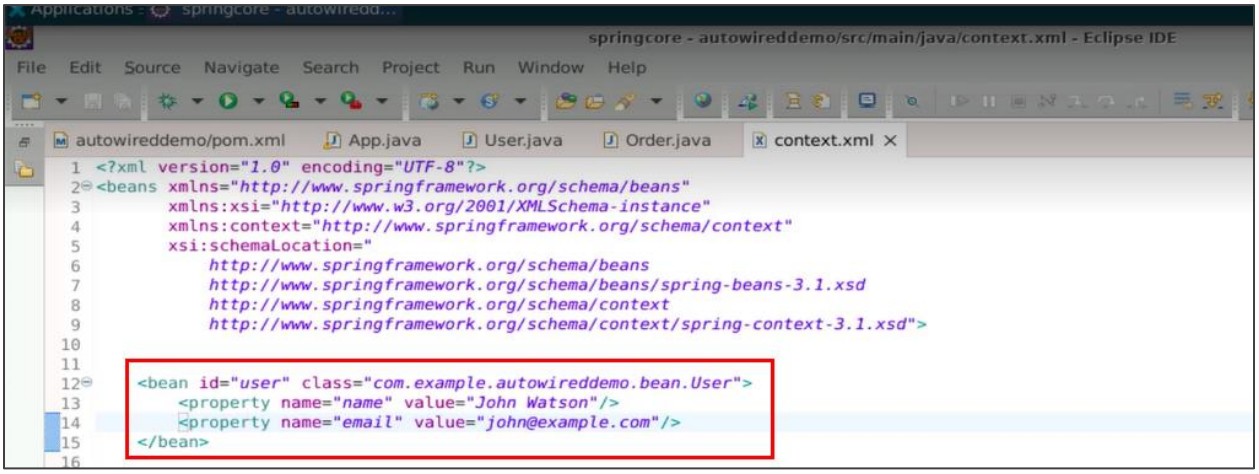
```java
    public String getName() {
        return name;
    }

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

    public String getEmail() {
        return email;
    }

    public void setEmail(String email) {
        this.email = email;
    }

    public Order getOrder() {
        return order;
    }

    // Setter Method will inject Order as Dependency which is pretty much the reference type
    public void setOrder(Order order) {
        System.out.println("[User] - SETTER INJECTION - setOrder executed having Order as Input");
        this.order = order;
    }

    @Override
    public String toString() {
        return "User [name=" + name + ", email=" + email + ", order=" + order + "]";
    }
```

Step 5: Configuring the context.xml for beans

5.1 Open the **context.xml** file



5.2 Define a bean for the user class with an id **user** and set the key-value pairs for its attributes



5.3 Define another bean for the Order class with an id **order** and set the key-value pairs for its attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.1.xsd">


    <bean id="user" class="com.example.autowireddemo.bean.User">
        <property name="name" value="John Watson"/>
        <property name="email" value="john@example.com"/>
    </bean>

    <bean id="order" class="com.example.autowireddemo.bean.Order">
        <property name="orderId" value="101"/>
        <property name="dateTimeStamp" value="20th Feb, 2022 20:00"/>
        <property name="amount" value="2000"/>
    </bean>


</beans>
```

5.4 Under the **user** bean, add a property tag and link it to the **order** bean using the **ref** attribute

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.1.xsd">


    <bean id="user" class="com.example.autowireddemo.bean.User">
        <property name="name" value="John Watson"/>
        <property name="email" value="john@example.com"/>
        <property name="order" ref="order"/>
    </bean>

    <bean id="order" class="com.example.autowireddemo.bean.Order">
        <property name="orderId" value="101"/>
        <property name="dateTimeStamp" value="20th Feb, 2022 20:00"/>
        <property name="amount" value="2000"/>
    </bean>


</beans>
```

5.5 In the **context.xml**, add the tag **context:annotation-config** to enable dependency injection using the **@Autowired** annotation

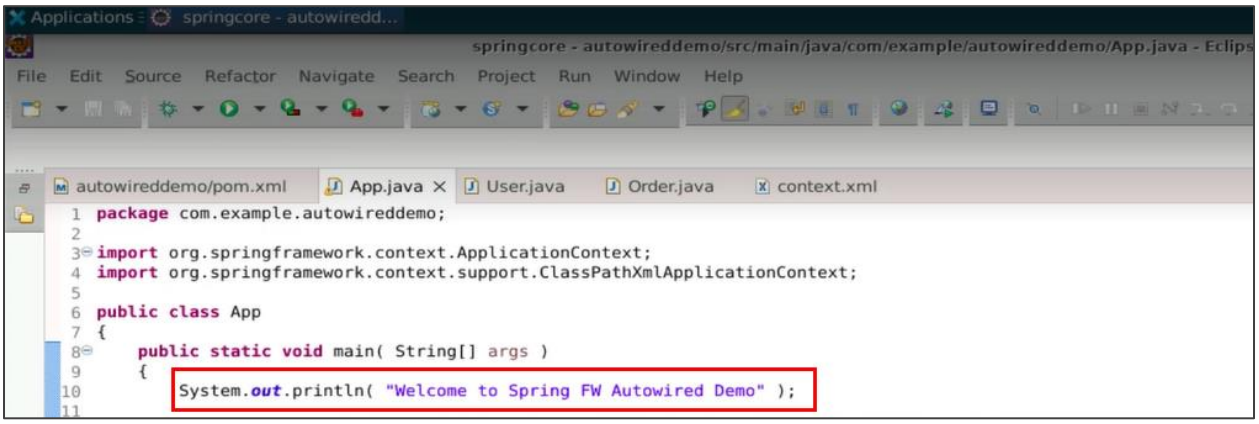
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:context="http://www.springframework.org/schema/context"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.1.xsd
           http://www.springframework.org/schema/context
           http://www.springframework.org/schema/context/spring-context-3.1.xsd">

    <context:annotation-config/>

    <bean id="user" class="com.example.autowireddemo.bean.User">
        <property name="name" value="John Watson"/>
        <property name="email" value="john@example.com"/>
        <!-- We are not configuring Setter or Constructor Injection in XML file
        <property name="order" ref="order"/>
        <constructor-arg ref="order"/>
         -->

    </bean>

    <bean id="order" class="com.example.autowireddemo.bean.Order">
        <property name="orderId" value="101"/>
        <property name="dateTimeStamp" value="20th Feb, 2022 20:00"/>
        <property name="amount" value="2000"/>
    </bean>


</beans>
```

5.6 In the **User.java** class, add the **@Autowired** annotation on top of the parameterized constructor (**User**) to use constructor injection

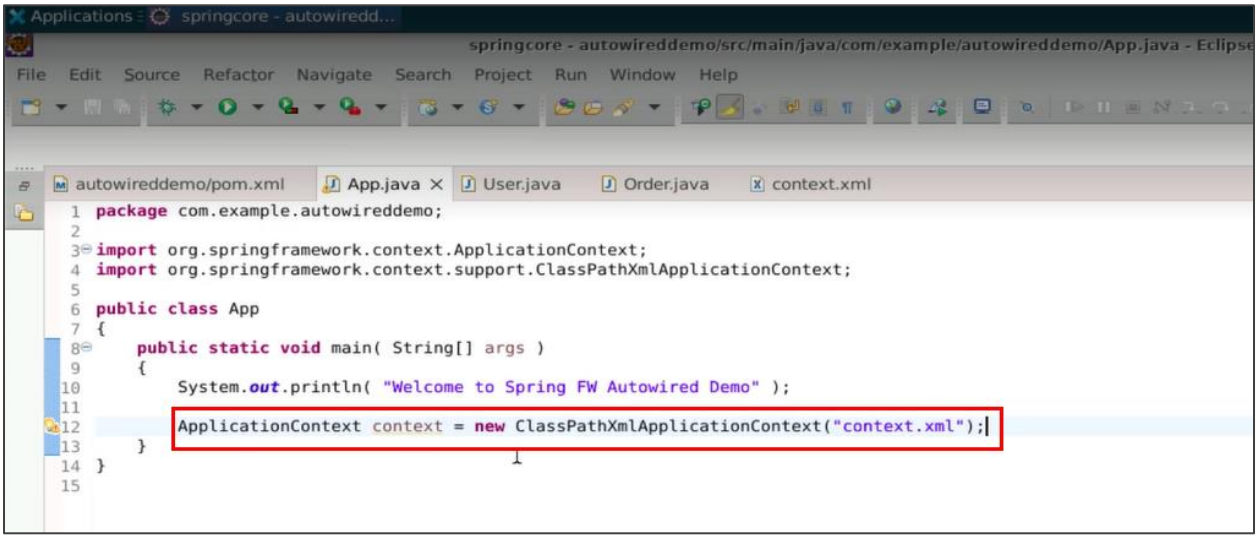
```java
package com.example.autowireddemo.bean;

import org.springframework.beans.factory.annotation.Autowired;

public class User {

    String name;
    String email;

    Order order; // 1 to 1 relationship mapping

    public User() {
        System.out.println("[User] - Object Created");
    }

    @Autowired
    public User(Order order) {
        System.out.println("[User] - CONSTRUCTOR INJECTION - Object Created with Parametrized Constructor having Order as Input");
        this.order = order;
    }

    public String getName() {
        return name;
    }

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

    public String getEmail() {
        return email;
    }
```

Step 6: Writing IOC code in App.java

6.1 Navigate to the **App.java** class and update the print statement to **Welcome to Spring FW Autowired Demo**



6.2 Create an instance of the **ApplicationContext** interface using the **ClassPathXmlApplicationContext** and pass the **context.xml** file to the **ApplicationContext** constructor



6.3 Use the **getBean()** method to retrieve the User bean instance by its reference ID and print the user

```
package com.example.autowireddemo;

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

import com.example.autowireddemo.bean.User;

public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Welcome to Spring FW Autowired Demo" );

        ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");
        User uRef = context.getBean("user", User.class);
        System.out.println(uRef);
    }
}
```

6.4 Run the project by clicking on the green play button

```
package com.example.autowireddemo;

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

import com.example.autowireddemo.bean.User;

public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Welcome to Spring FW Autowired Demo" );

        ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");
        User uRef = context.getBean("user", User.class);
        System.out.println(uRef);
    }
}
```

```
App.java ×   User.java   Order.java   context.xml
utowireddemo;

ework.context.ApplicationContext;
ework.context.support.ClassPathXmlApplicationContext;

towireddemo.bean.User;


d main( String[] args )

intln( "Welcome to Spring FW Autowired Demo" );

ntext context = new ClassPathXmlApplicationContext("context.xml");
ontext.getBean("user", User.class);
intln(uRef);
```

```
Markers   Properties   Servers   Data Source   Snippets   Console ×
<terminated> App (4) [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hots
Welcome to Spring FW Autowired Demo
[Order] - Object Created
[User] - CONSTRUCTOR INJECTION - Object Created with Parametrized Const
User [name=John Watson, email=john@example.com, order=Order [orderId=10 ,
```

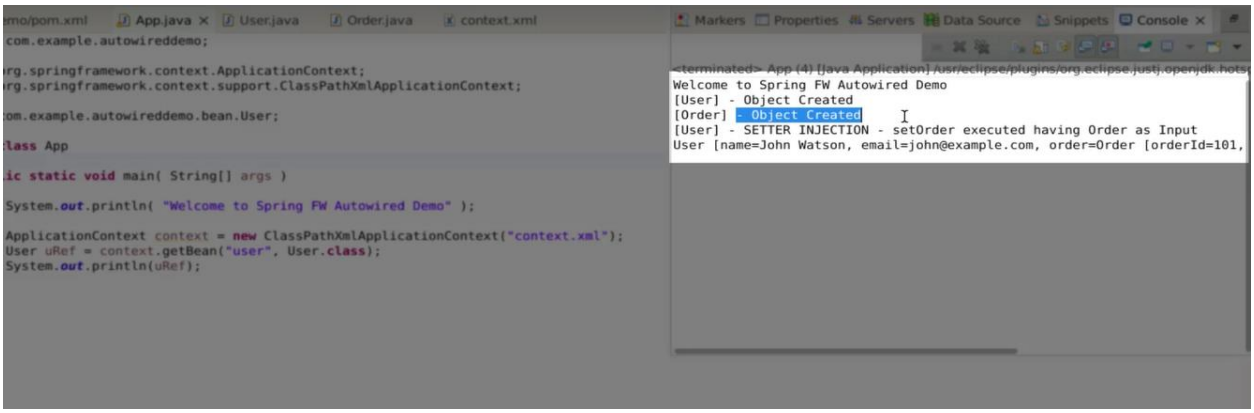
You may notice in the console logs that the user bean object is created through the parameterized constructor using the constructor injection technique.

6.5 Now, in the **User.java** class, change the **@Autowired** annotation from the parameterized constructor to the setter method **setOrder**

```
springcore - autowireddemo/src/main/java/com/example/autowireddemo/bean/User.java - Eclipse IDE
File   Edit   Source   Refactor   Navigate   Search   Project   Run   Window   Help
autowireddemo/pom.xml   App.java   User.java ×   Order.java   context.xml
22      public String getName() {
23          return name;
24      }
25
26      public void setName(String name) {
27          this.name = name;
28      }
29
30      public String getEmail() {
31          return email;
32      }
33
34      public void setEmail(String email) {
35          this.email = email;
36      }
37
38      public Order getOrder() {
39          return order;
40      }
41
42      // Setter Method will inject Order as Dependency which is pretty much the reference type
43      @Autowired
44      public void setOrder(Order order) {
45          System.out.println("[User] - SETTER INJECTION - setOrder executed having Order as Input");
46          this.order = order;
47      }
48
49      @Override
50      public String toString() {
51          return "User [name=" + name + ", email=" + email + ", order=" + order + "]";
52      }
53
```

With this update, the IOC container will know that dependency injection should be done using the setter method.

6.6 Rerun the project



In the console logs, you can observe that the order bean object has been created with a default constructor, and the setter injection method is executed, displaying all the user details.