Lesson 01 Demo 02

IOC with BeanFactory

Objective: To understand how to implement inversion of control (IOC) using the BeanFactory in the Spring Core framework

Tool required: Eclipse IDE

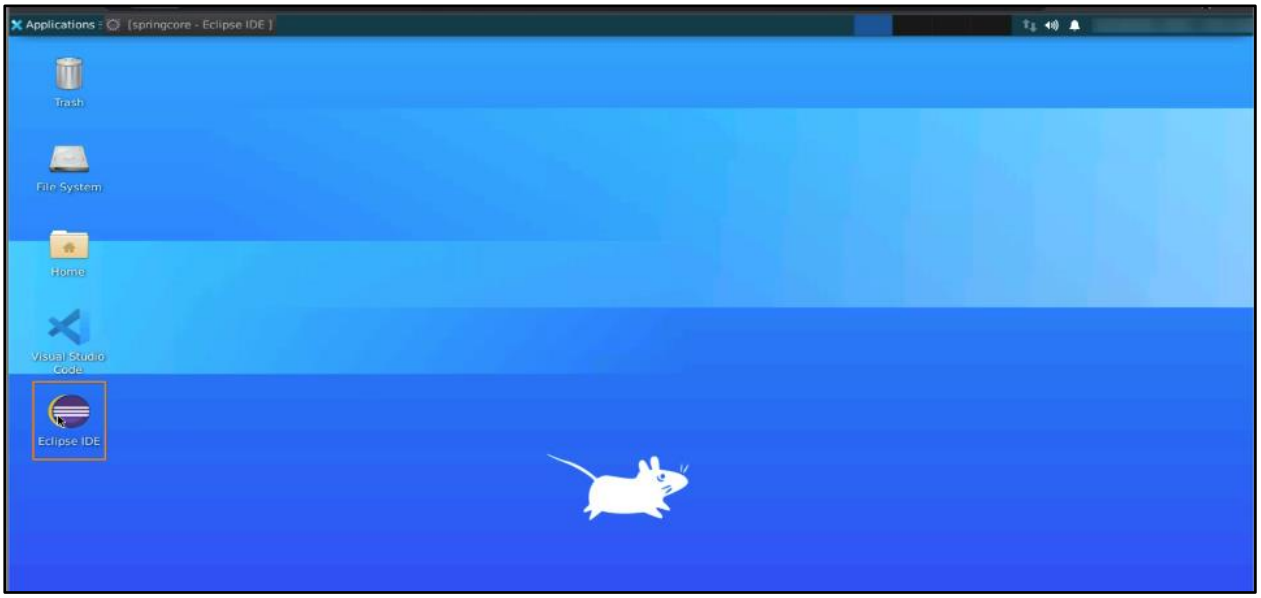
Prerequisites: None

Steps to be followed:

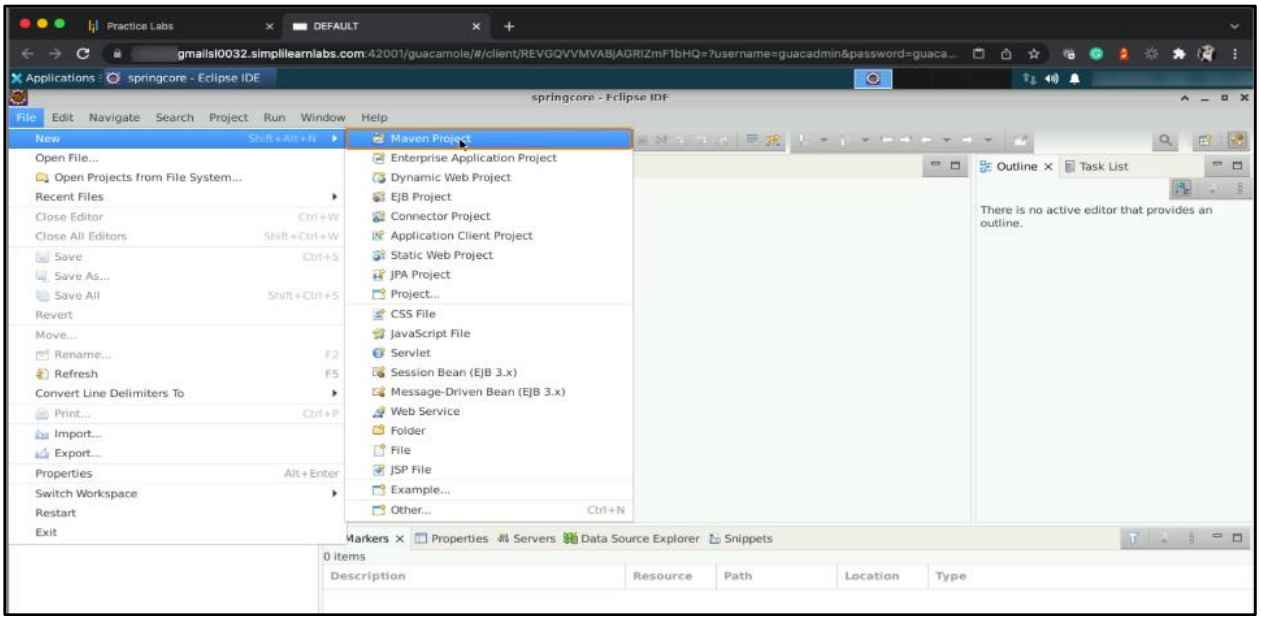
1. Setting up the development environment
2. Adding dependencies in the **pom.xml** file
3. Creating a bean class
4. Defining the attributes
5. Adding the Getters and Setters
6. Adding bean elements and properties
7. Configuring beans and properties
8. Accessing and running the beans

Step 1: Setting up the development environment

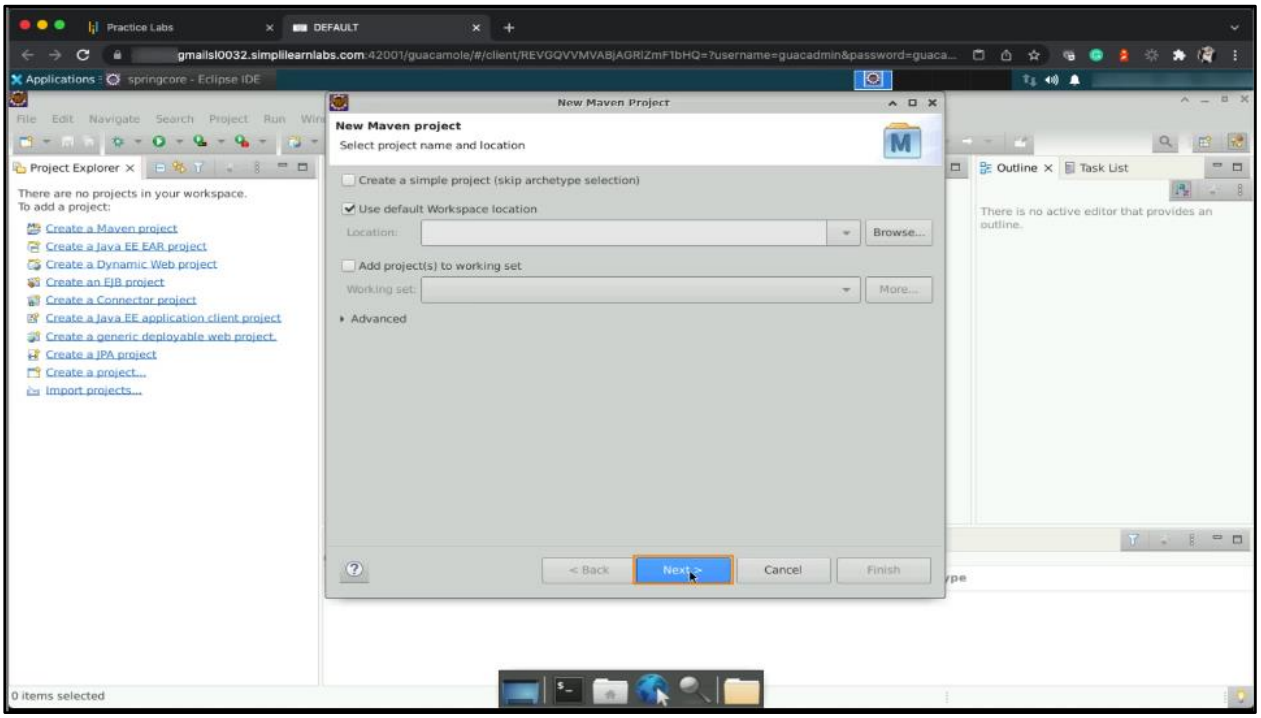
1.1 Open **Eclipse IDE**



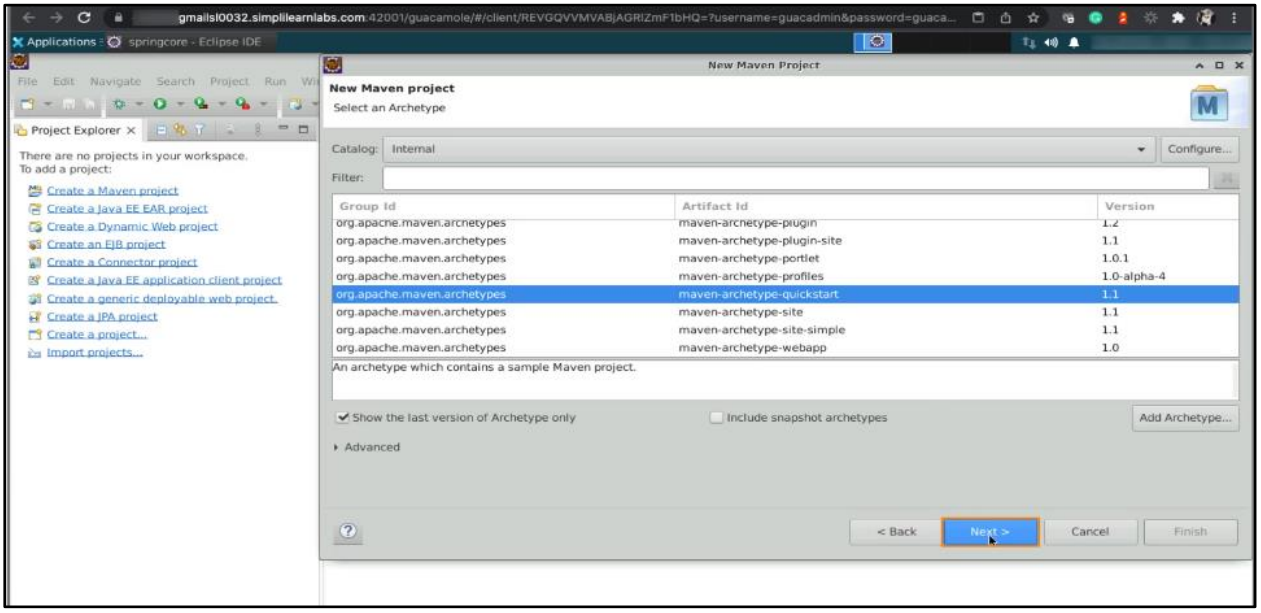
1.2 Click on **File** in the menu bar, select **New,** and choose **Maven Project**



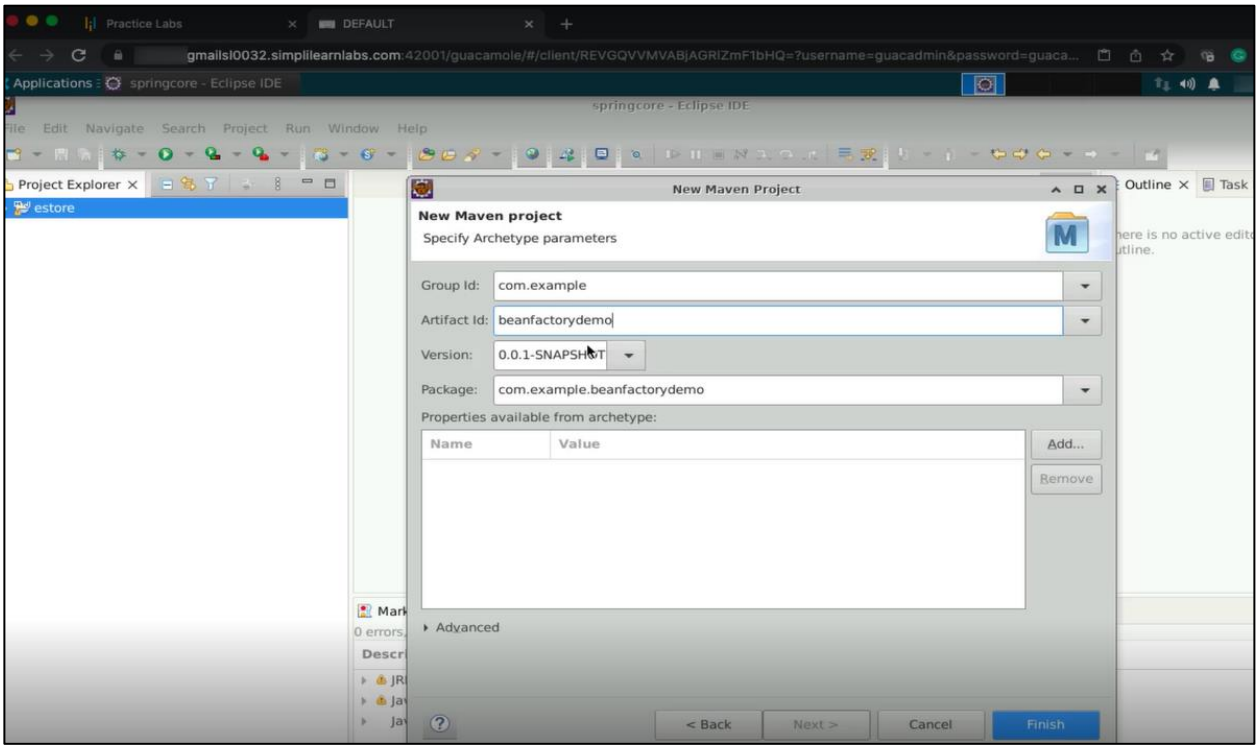
1.3 Choose the default workspace and click **Next**



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

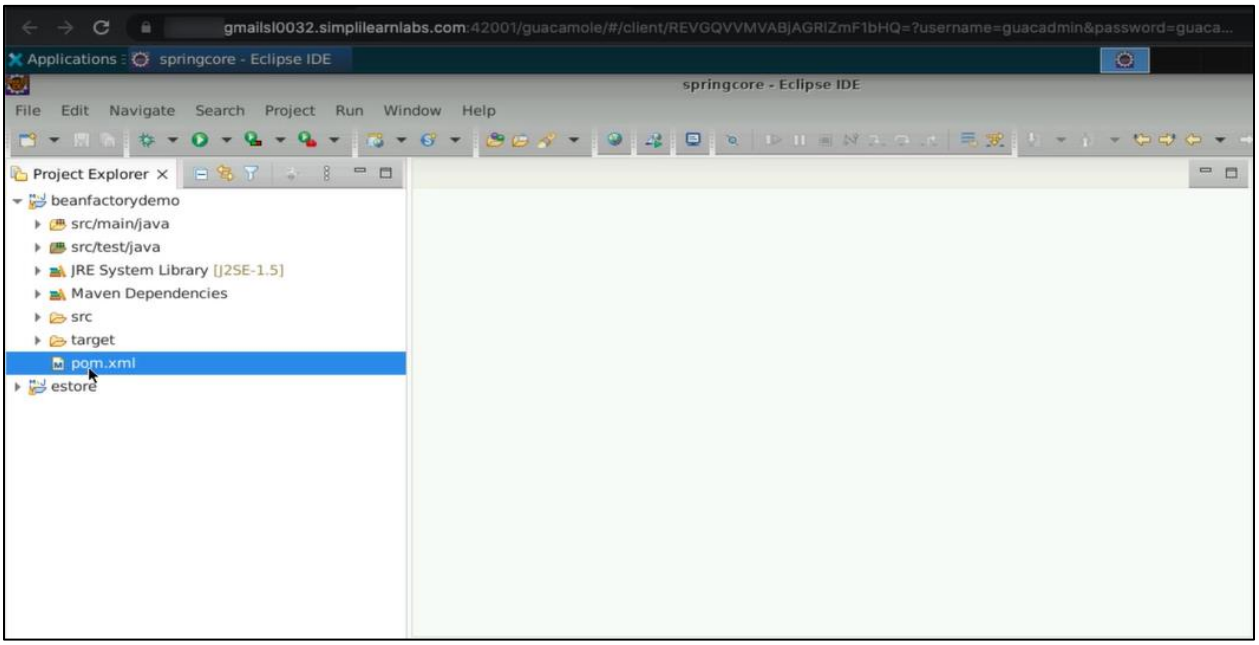


1.5 Provide the Group Id, which is typically the company's domain name in reverse order, and the Artifact Id as **beanfactorydemo**, then click **Finish**

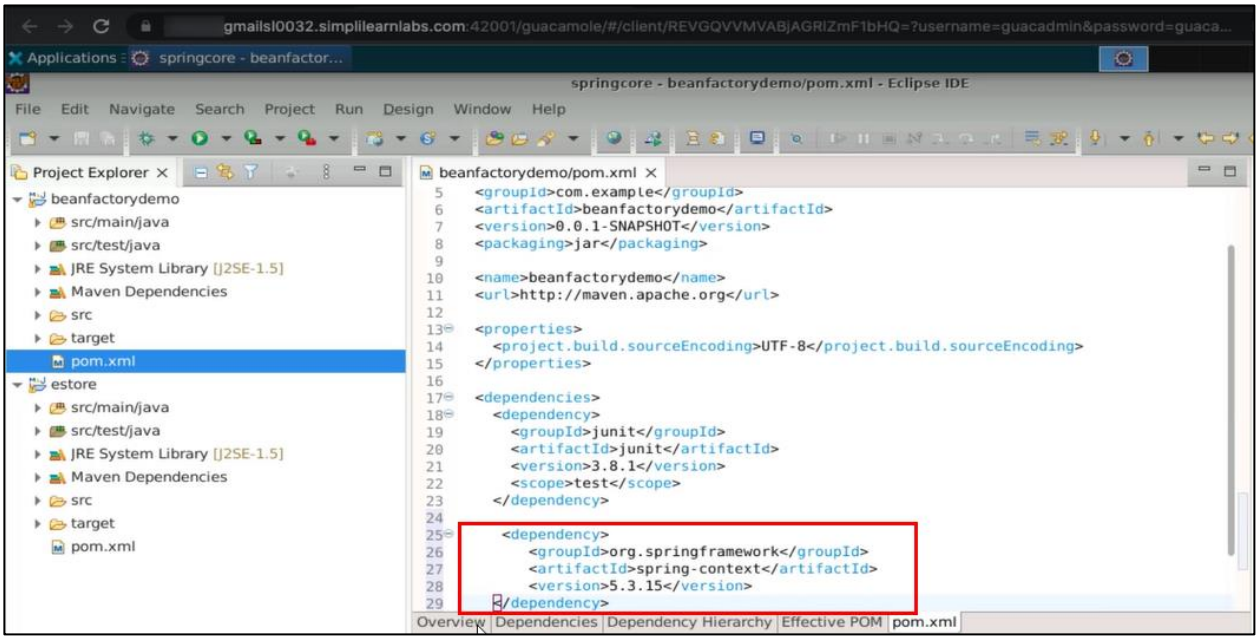


Step 2: Adding dependencies in the pom.xml file

2.1 Open the **pom.xml** file in the project's root directory

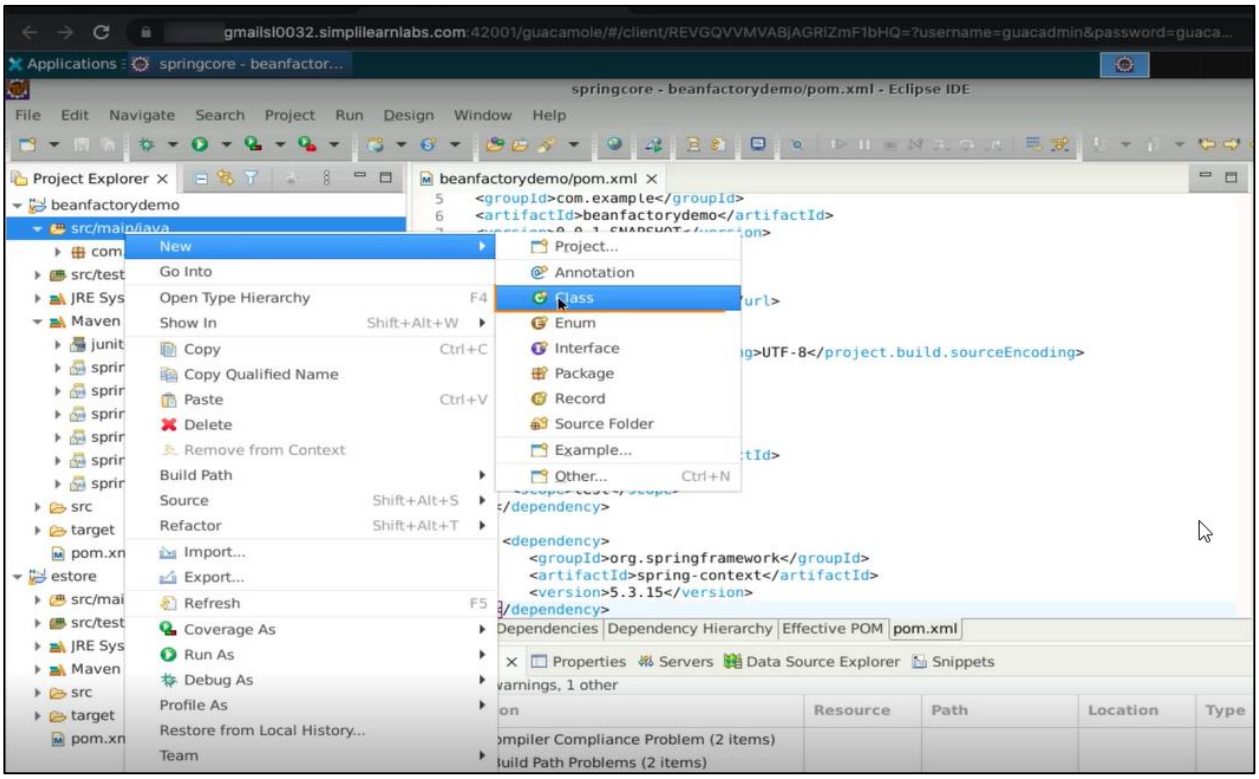


2.2 Add the necessary dependencies for Spring Core, such as **spring-context**

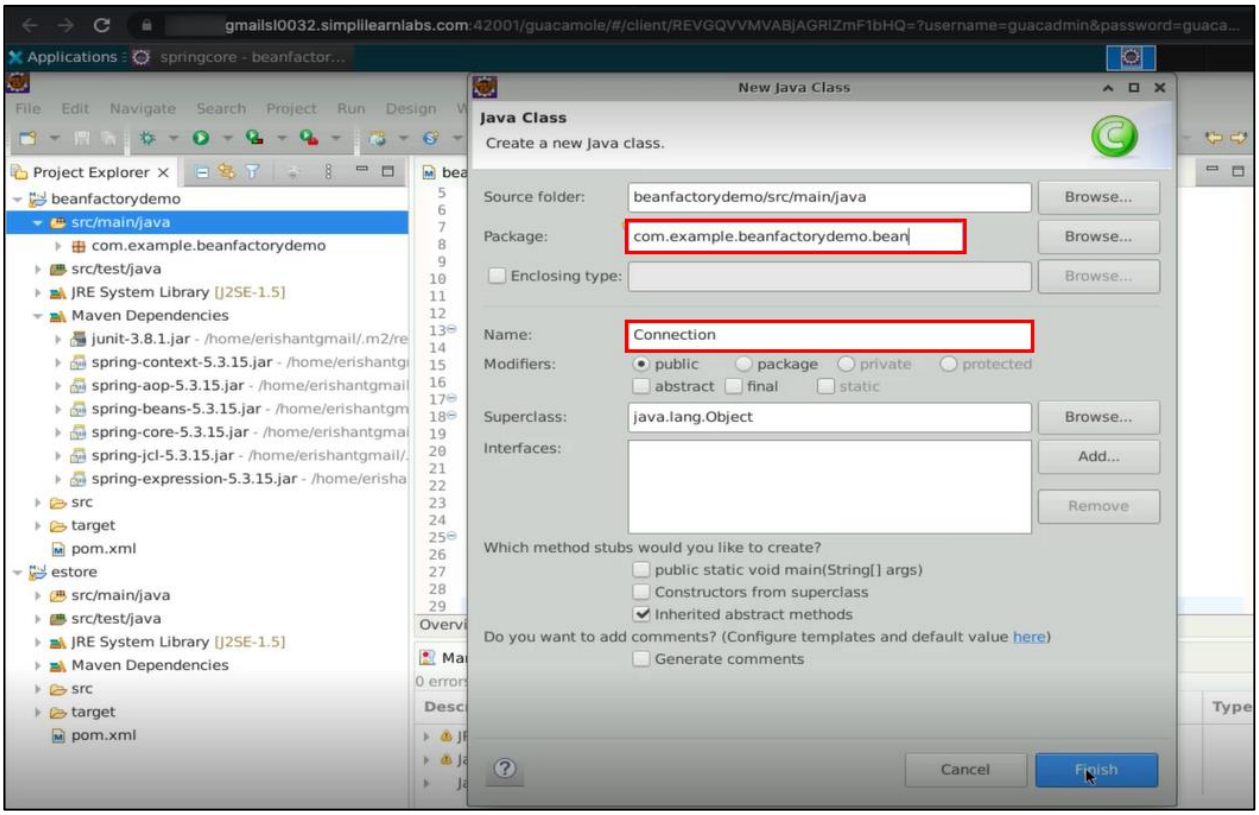


Step 3: Creating a bean class

3.1 In **Eclipse**, right-click on the **src/main/java** package, select **New ,** and click **Class**

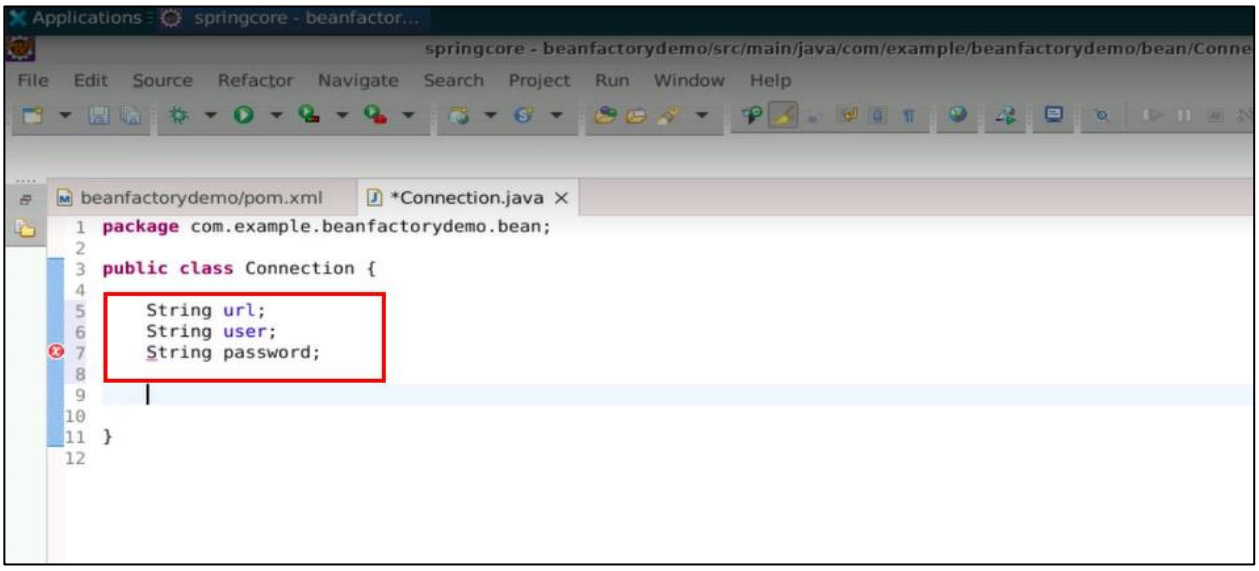


3.2 Name the class **Connection** and add **.bean** to the package name, then click **Finish**

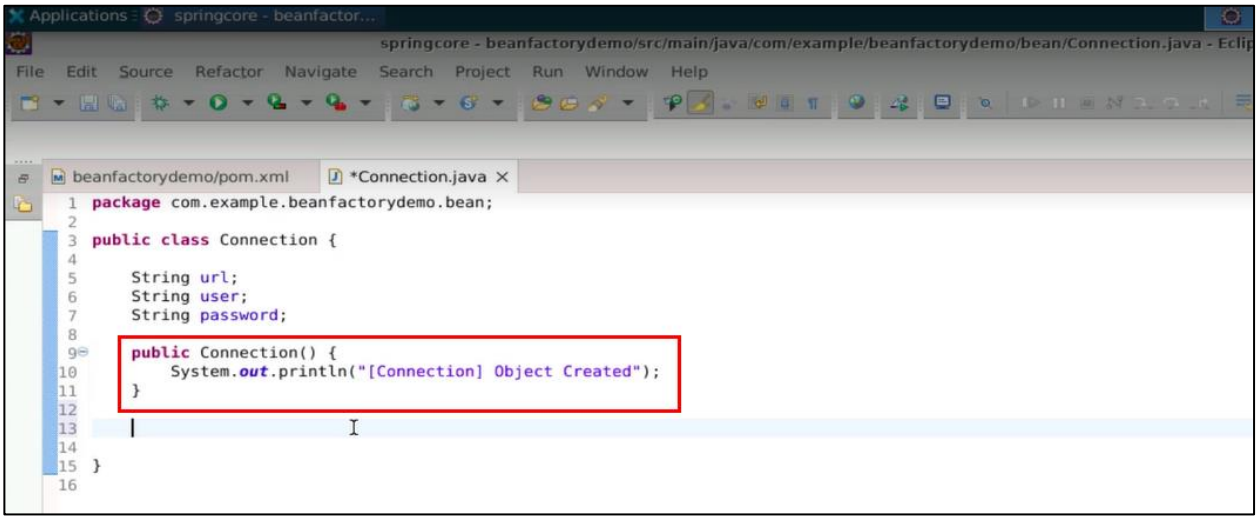


Step 4: Defining the attributes

4.1 Define attributes in the **Connection** class, such as **url**, **username**, and **password**

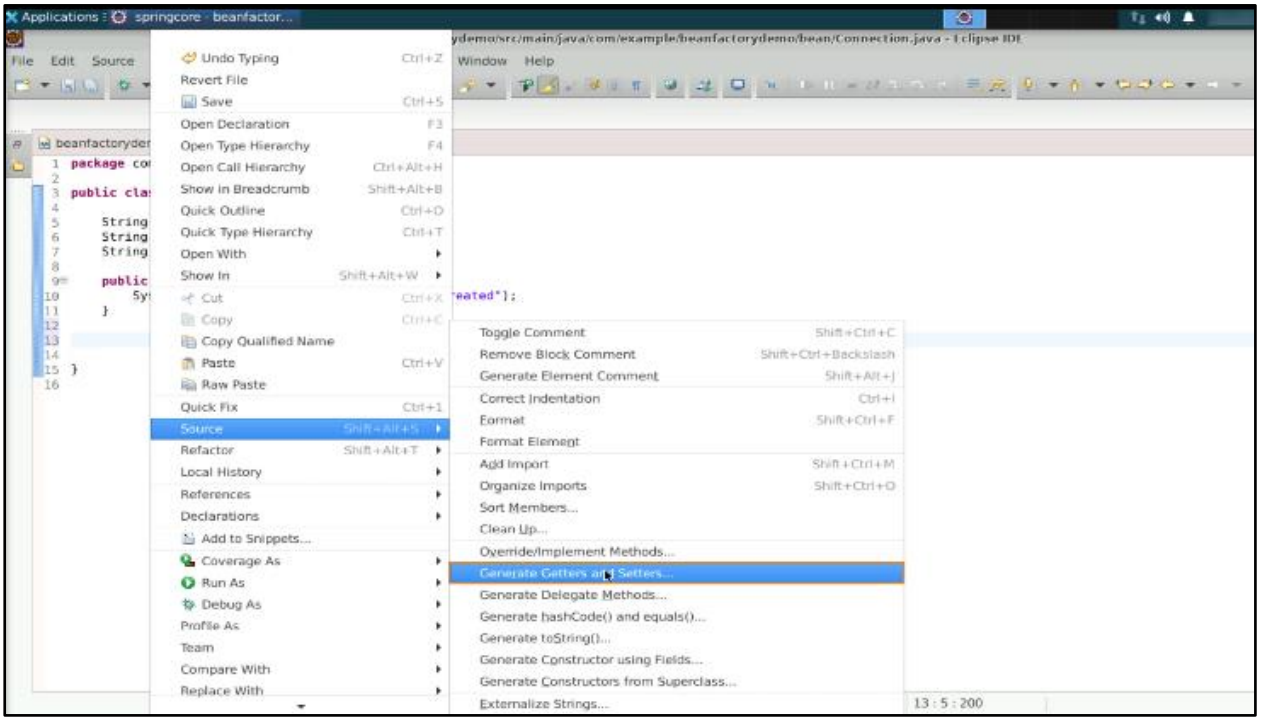


4.2 Create a default constructor for the class with a print statement

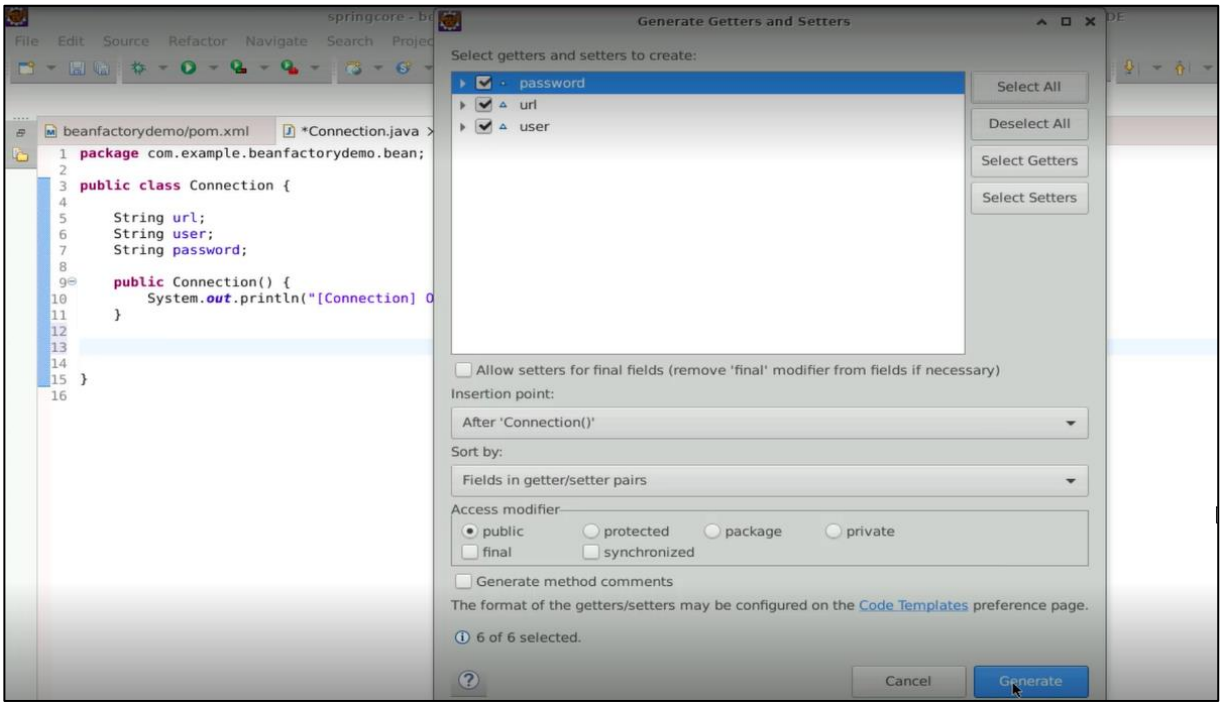


Step 5: Adding the Getters and Setters

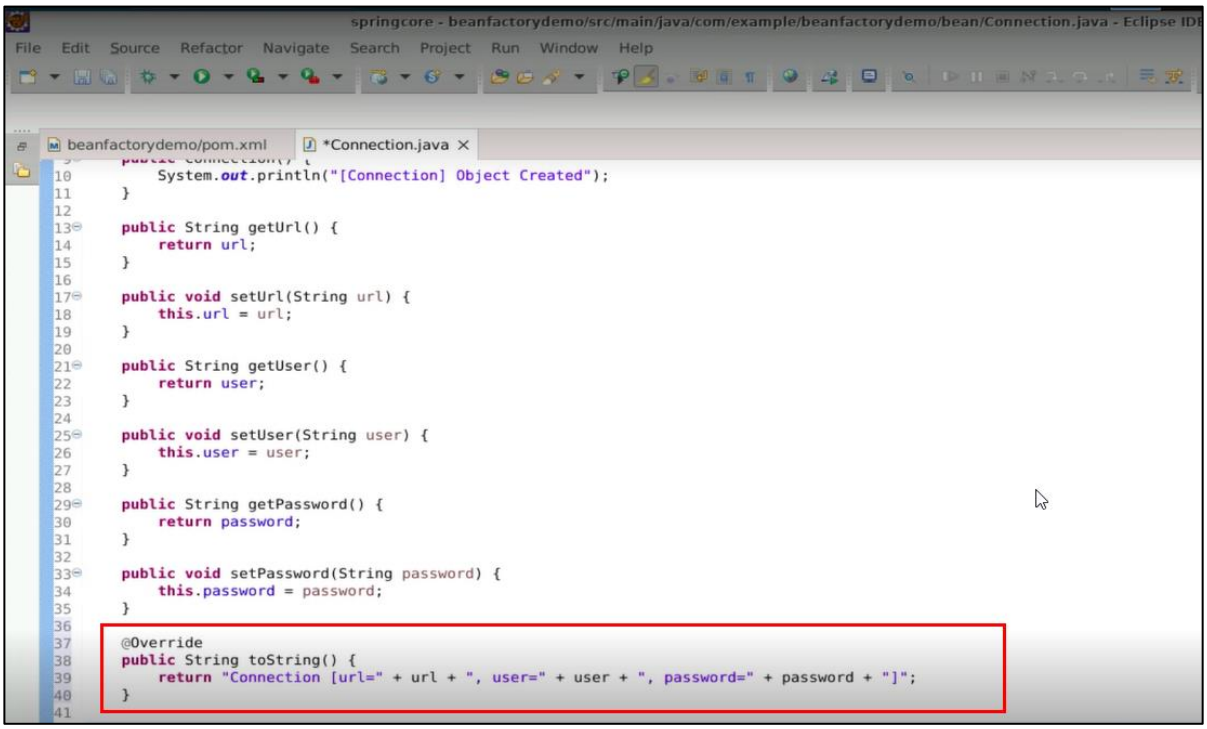
5.1 Right-click on the **Connection** class, select **Source**, and click **Generate Getters and Setters**



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

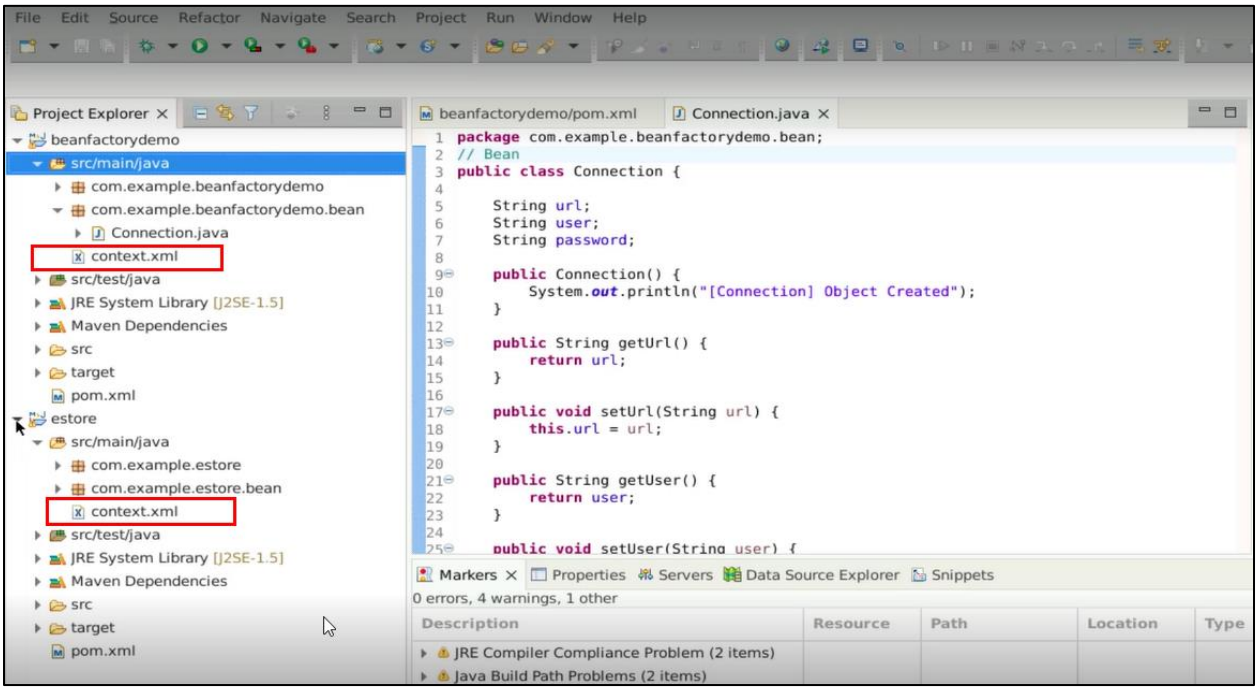


5.3 Repeat the same step to generate a **toString()** method that returns all the attributes



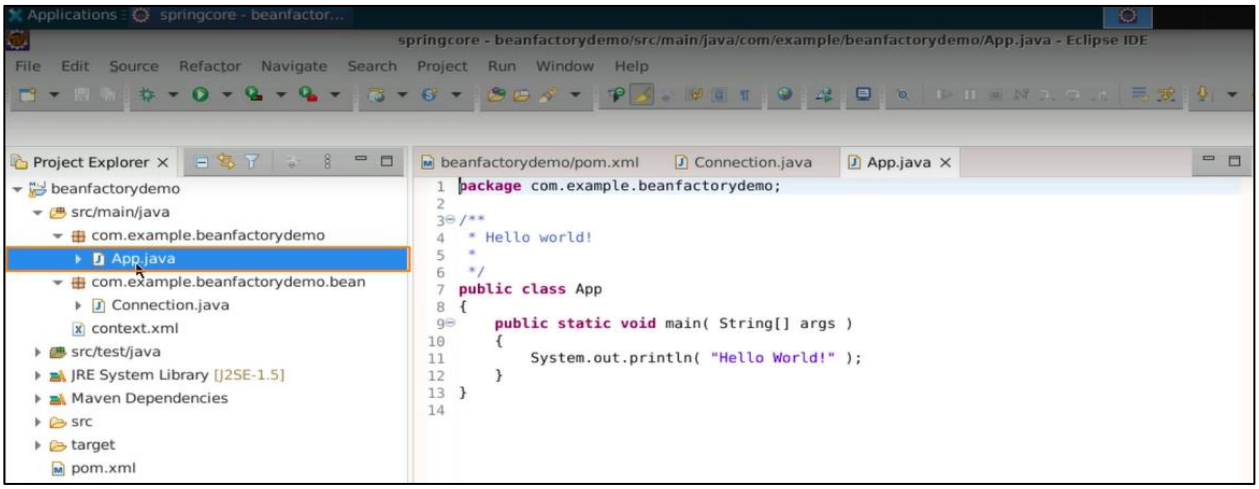
Step 6: Adding bean elements and properties

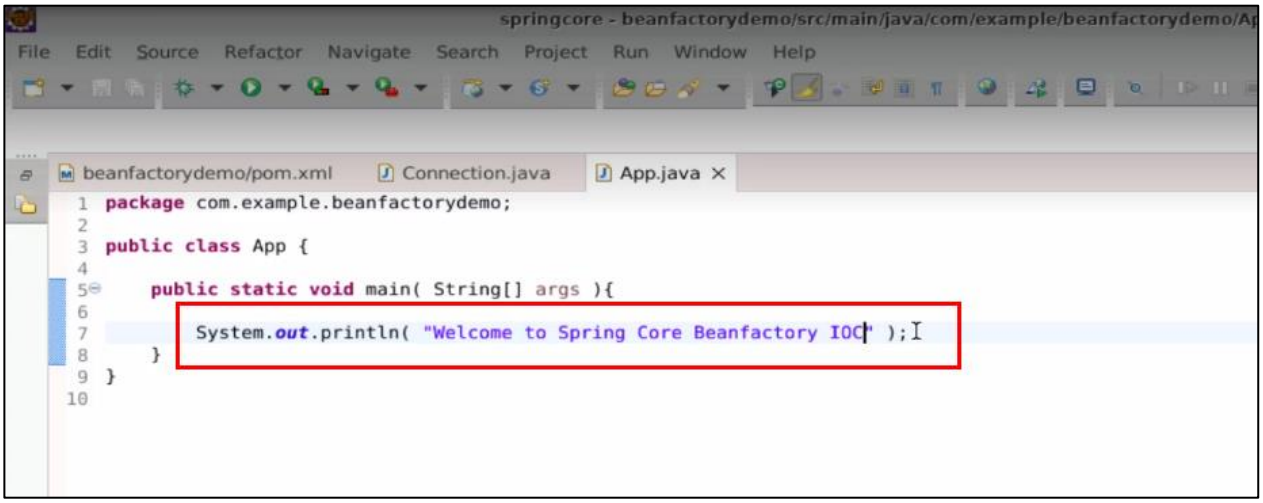
6.1 Copy the **context.xml** file from the estore project to the **beanfactorydemo** project



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

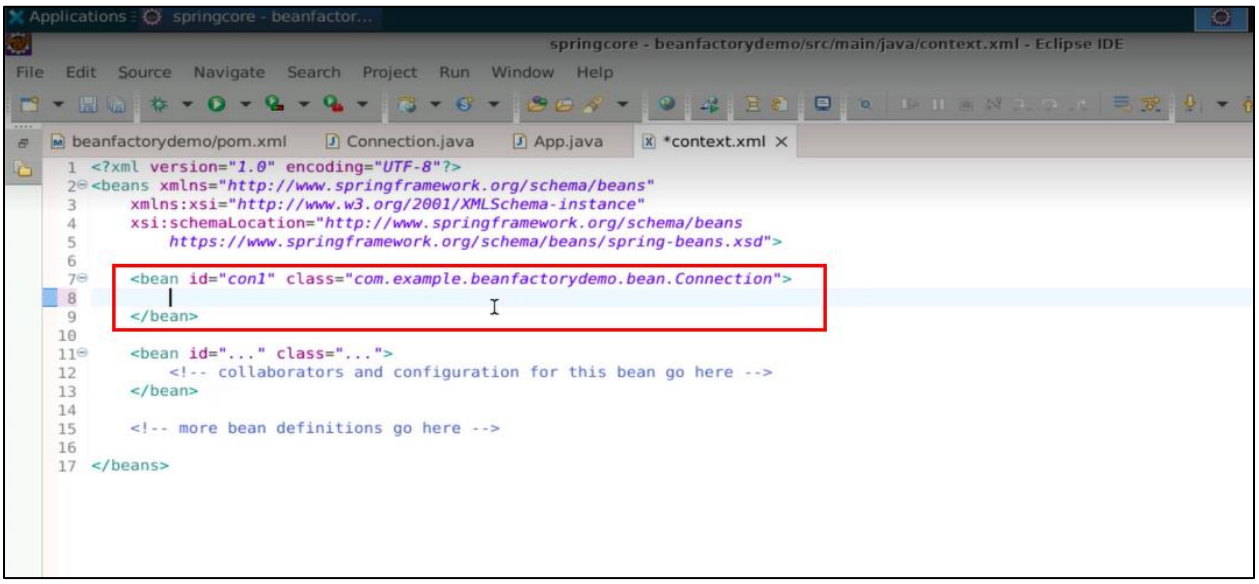
6.2 Navigate to the **App.java** file and update the print statement to **Welcome to Spring Core Beanfactory IOC**


springcore - beanfactorydemo/src/main/java/com/example/beanfactorydemo/App.java - Eclipse IDE
File Edit Source Refactor Navigate Search Project Run Window Help
Project Explorer
beanfactorydemo
src/main/java
com.example.beanfactorydemo
App.java
com.example.beanfactorydemo.bean
Connection.java
context.xml
src/test/java
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml
beanfactorydemo/pom.xml
Connection.java
App.java
package com.example.beanfactorydemo;
/**
* Hello world!
*
*/
public class App
{
public static void main(String[] args)
{
System.out.println("Hello World!");
}
}

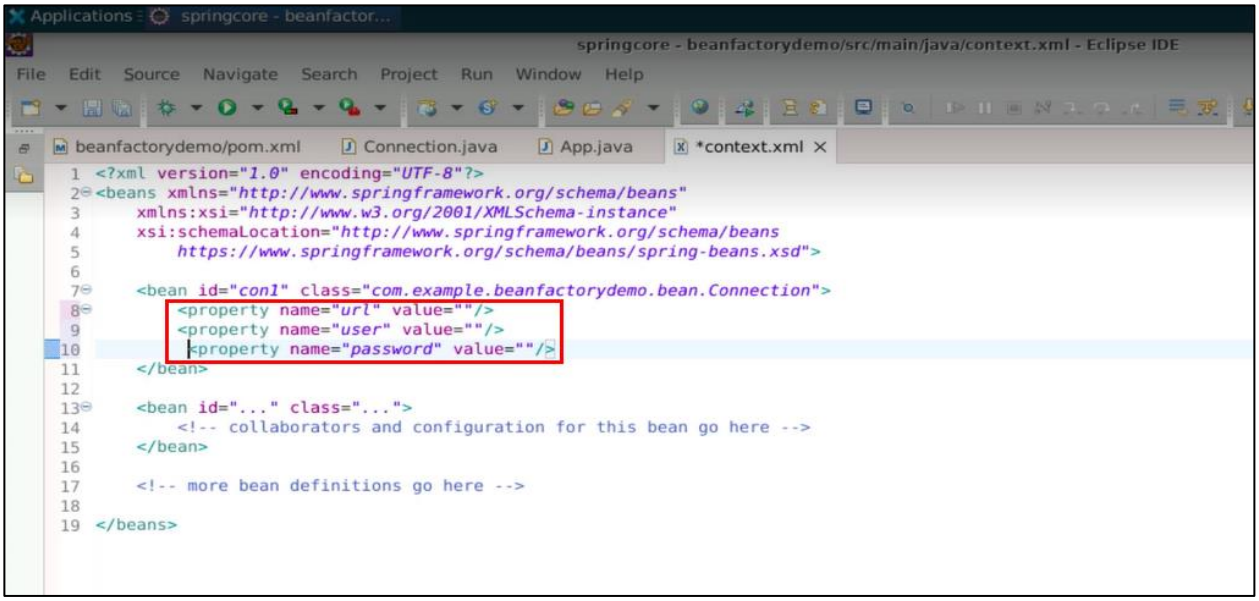

springcore - beanfactorydemo/src/main/java/com/example/beanfactorydemo/Ap
File Edit Source Refactor Navigate Search Project Run Window Help
beanfactorydemo/pom.xml
Connection.java
App.java
package com.example.beanfactorydemo;
public class App {
public static void main(String[] args){
System.out.println("Welcome to Spring Core Beanfactory IOC");
}
}

Step 7: Configuring beans and properties

7.1 In the **context.xml** file, under the **bean** tag, add the id as **con1**, and specify the class name of the **Connection** class



7.2 Add a properties element in the bean that contains the name and value



7.3 Configure another bean as **con2** with the same attributes as the **Connection** class

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

    <bean id="con1" class="com.example.beanfactorydemo.bean.Connection">
        <property name="url" value=""/>
        <property name="user" value=""/>
        <property name="password" value=""/>
    </bean>

    <bean id="con2" class="com.example.beanfactorydemo.bean.Connection">
        <property name="url" value=""/>
        <property name="user" value=""/>
        <property name="password" value=""/>
    </bean>

    <!-- more bean definitions go here -->

</beans>
```

7.4 Add values for both beans **con1** and **con2** with different database names, users, and passwords

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

    <bean id="con1" class="com.example.beanfactorydemo.bean.Connection">
        <property name="url" value="jdbc:mysql://localhost/estore"/>
        <property name="user" value="john.watson"/>
        <property name="password" value="pass123"/>
    </bean>

    <bean id="con2" class="com.example.beanfactorydemo.bean.Connection">
        <property name="url" value="jdbc:mysql://localhost/fooddelivery"/>
        <property name="user" value="fionna"/>
        <property name="password" value="fionna@123"/>
    </bean>

    <!-- more bean definitions go here -->

</beans>
```

Step 8: Accessing and running the beans

8.1 In the **App.java** file, within the main function, write the code to read the **context.xml** file using **ClassPathResource**. Then, utilize **BeanFactory** to obtain references to the beans, and print the values (**url**, **user**, and **password**) of the beans along with the hash code.

```
public static void main( String[] args ){

    System.out.println( "Welcome to Spring Core Beanfactory IOC" );
    /*
        IOC : Inversion Of Control
     */
    // Traditionally we create Objects using new operator and we write the data inside it
    /*Connection con1 = new Connection();
    con1.setUrl("jdbc:mysql://localhost/estore");
    con1.setUser("john.watson");
    con1.setPassword("password123");
    System.out.println("con1 is: "+con1);*/

    // As per IOC, let the objects be created by Spring FW's IOC Container
    // As a developer we will not create the objects rather we will configure the objects
    // We will use xml file in which we will configure the objects

    // BeanFactory - IOC Container
    Resource resource = new ClassPathResource("context.xml");
    BeanFactory factory = new XmlBeanFactory(resource);
    System.out.println("BeanFactory IOC Container Created");

    // Get the reference of the beans from the BeanFactory IOC Container
    Connection c1 = (Connection) factory.getBean("con1");
    Connection c2 = factory.getBean("con2", Connection.class);
    Connection c3 = factory.getBean("con1", Connection.class); // Get the Bean's Reference again for id con1

    System.out.println("c1 is: "+c1+" and hashcode is: "+c1.hashCode());
    System.out.println("c2 is: "+c2+" and hashcode is: "+c2.hashCode());
    System.out.println("c3 is: "+c3+" and hashcode is: "+c3.hashCode());

}
```

8.2 Run the project by clicking on the green run button

```
public static void main( String[] args ){

    System.out.println( "Welcome to Spring Core Beanfactory IOC" );
    /*
        IOC : Inversion Of Control
     */
    // Traditionally we create Objects using new operator and we write the data inside it
    /*Connection con1 = new Connection();
    con1.setUrl("jdbc:mysql://localhost/estore");
    con1.setUser("john.watson");
    con1.setPassword("password123");
    System.out.println("con1 is: "+con1);*/

    // As per IOC, let the objects be created by Spring FW's IOC Container
    // As a developer we will not create the objects rather we will configure the objects
    // We will use xml file in which we will configure the objects

    // BeanFactory - IOC Container
    Resource resource = new ClassPathResource("context.xml");
    BeanFactory factory = new XmlBeanFactory(resource);
    System.out.println("BeanFactory IOC Container Created");

    // Get the reference of the beans from the BeanFactory IOC Container
    Connection c1 = (Connection) factory.getBean("con1");
    Connection c2 = factory.getBean("con2", Connection.class);
    Connection c3 = factory.getBean("con1", Connection.class); // Get the Bean's Reference again for id con1

    System.out.println("c1 is: "+c1+" and hashcode is: "+c1.hashCode());
    System.out.println("c2 is: "+c2+" and hashcode is: "+c2.hashCode());
    System.out.println("c3 is: "+c3+" and hashcode is: "+c3.hashCode());

}
```

```
<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java (Feb 8, 2022
Welcome to Spring Core Beanfactory IOC
BeanFactory IOC Container Created
[Connection] Object Created
[Connection] Object Created
c1 is: Connection [url=jdbc:mysql://localhost/estore, user=john.watson, password=pass123] and hashcode is: 934275857
c2 is: Connection [url=jdbc:mysql://localhost/fooddelivery, user=fionna, password=fionna@123] and hashcode is: 1364913072
c3 is: Connection [url=jdbc:mysql://localhost/estore, user=john.watson, password=pass123] and hashcode is: 934275857
```

You can see the bean object values printed on the console with the hash code.

8.3 Now, change the value of the password attribute for both beans in the **context.xml** file

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

    <bean id="con1" class="com.example.beanfactorydemo.bean.Connection">
        <property name="url" value="jdbc:mysql://localhost/estore"/>
        <property name="user" value="john.watson"/>
        <property name="password" value="@vegers!@#"/>
    </bean>

    <bean id="con2" class="com.example.beanfactorydemo.bean.Connection">
        <property name="url" value="jdbc:mysql://localhost/fooddelivery"/>
        <property name="user" value="fionna"/>
        <property name="password" value="t@ngocha@rlie"/>
    </bean>

    <!-- more bean definitions go here -->

</beans>
```

8.4 Run the project again

```
<terminated> App [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java (Feb 8, 2022
Welcome to Spring Core Beanfactory IOC
BeanFactory IOC Container Created
[Connection] Object Created
[Connection] Object Created
c1 is: Connection [url=jdbc:mysql://localhost/estore, user=john.watson, password=@vegers!@#] and hashcode is: 934275857
c2 is: Connection [url=jdbc:mysql://localhost/fooddelivery, user=fionna, password=t@ngocha@rlie] and hashcode is: 1364913072
c3 is: Connection [url=jdbc:mysql://localhost/estore, user=john.watson, password=@vegers!@#] and hashcode is: 934275857
```

As you can see, the passwords are now updated on the console.

This demonstrates the beauty of Spring IOC, where you don't need to modify anything in the main code. The values are dynamically retrieved at runtime, allowing for flexible configuration without requiring code changes.