Lesson 01 Demo 04

Dependency Injection with Setter and Constructor

Objective: To understand and implement dependency injection using setter and constructor methods in a Spring application

Tool required: Eclipse IDE

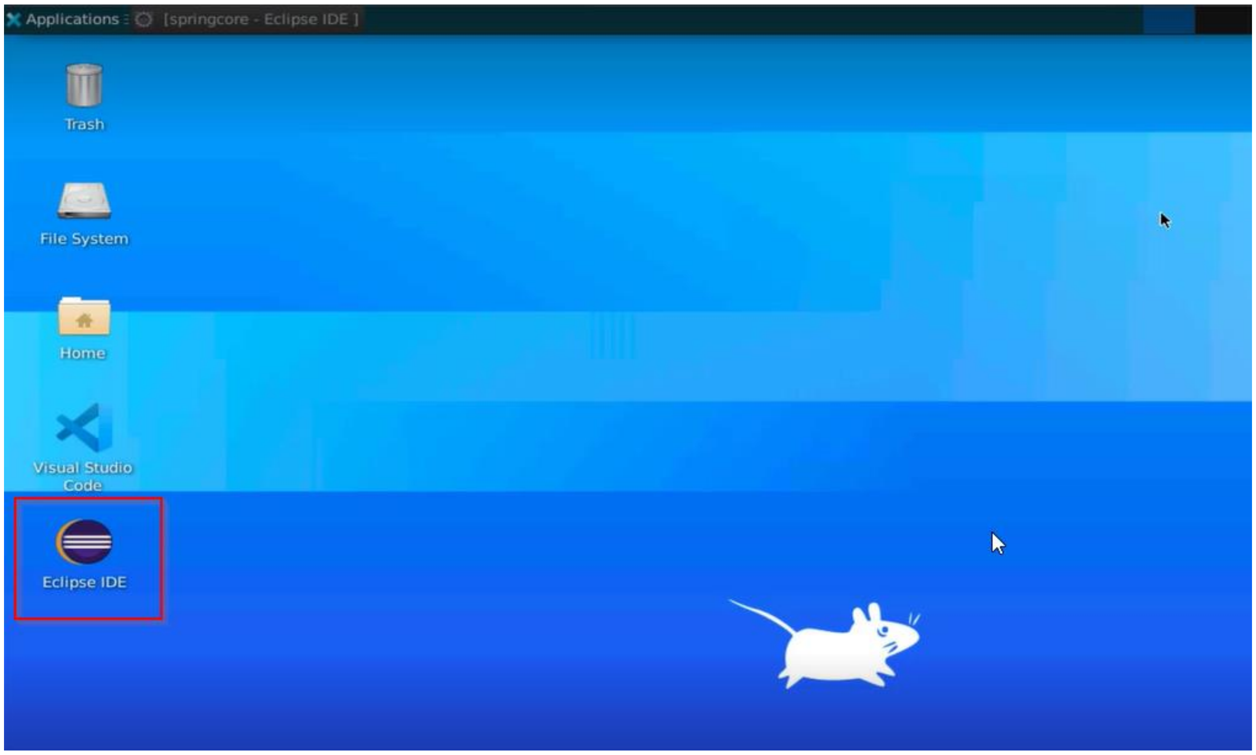
Prerequisites: None

Steps to be followed:

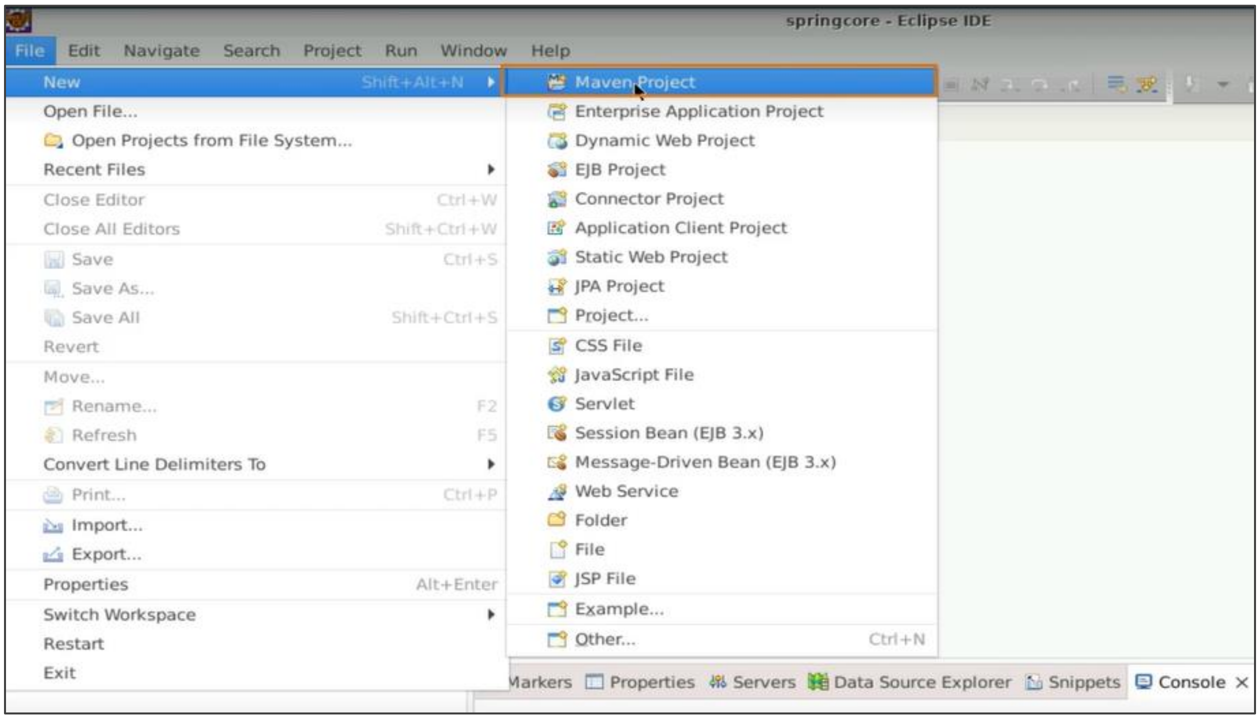
1. Creating a Maven project
2. Creating an Address bean
3. Creating the Restaurant bean
4. Configuring the **context.xml** for beans
5. Writing IOC code in **App.java**
6. Creating a parameterized constructor
7. Understanding one-to-many relationships

Step 1: Creating a Maven project

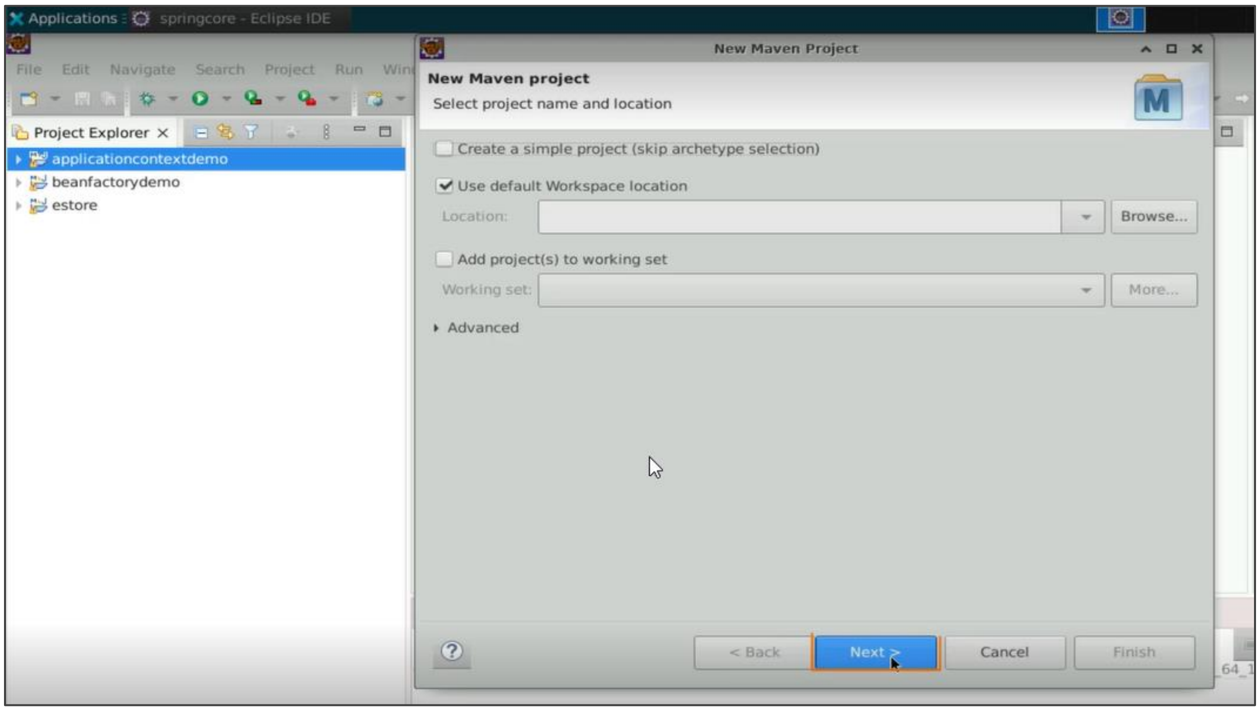
1.1 Open **Eclipse IDE**



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



1.3 Provide the desired workspace location for your project and click **Finish**



New Maven project

Select project name and location

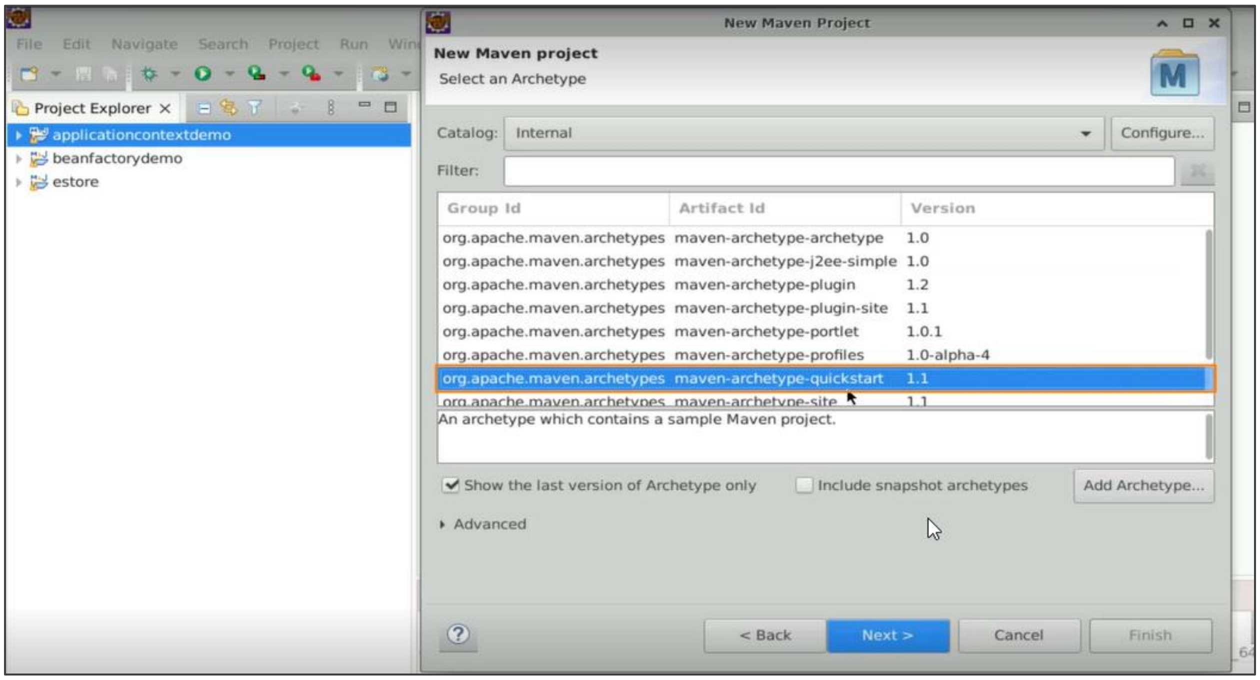
- [ ] Create a simple project (skip archetype selection)
- [x] Use default Workspace location

Location:

- [ ] Add project(s) to working set

Working set:

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



New Maven project
Select an Archetype

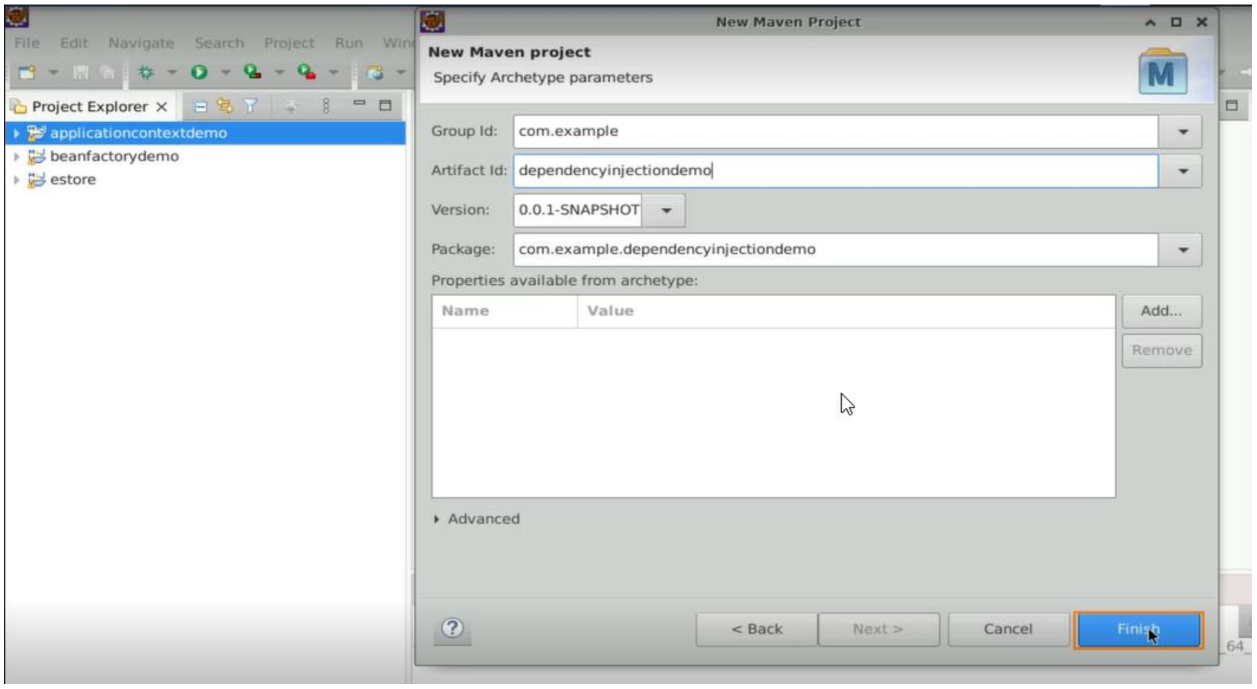
Catalog: Internal

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-archetype	1.0
org.apache.maven.archetypes	maven-archetype-j2ee-simple	1.0
org.apache.maven.archetypes	maven-archetype-plugin	1.2
org.apache.maven.archetypes	maven-archetype-plugin-site	1.1
org.apache.maven.archetypes	maven-archetype-portlet	1.0.1
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.1
org.apache.maven.archetypes	maven-archetype-site	1.1

An archetype which contains a sample Maven project.

Show the last version of Archetype only [✓]
Include snapshot archetypes [ ]

1.5 Enter the **Group Id**, which is typically the reverse order of the company's domain name, and the **Artifact Id** as **dependencyinjectiondemo**. Click **Finish** to create the Maven project.



New Maven project
Specify Archetype parameters

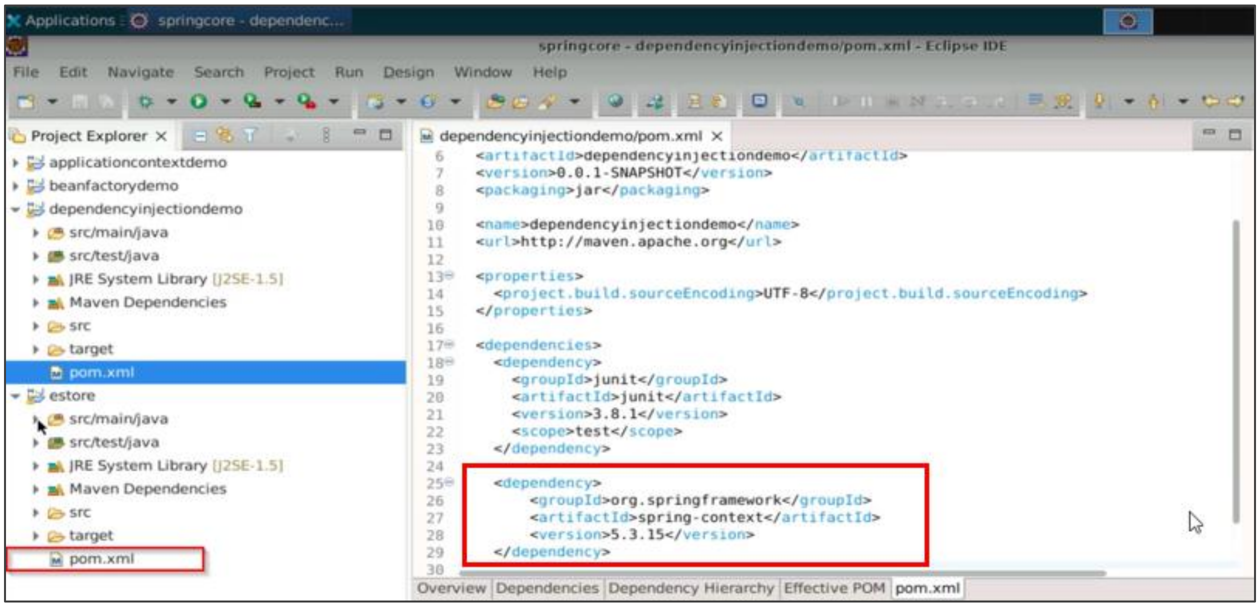
Group Id: com.example
Artifact Id: dependencyinjectiondemo
Version: 0.0.1-SNAPSHOT
Package: com.example.dependencyinjectiondemo

Properties available from archetype:

Name	Value

Step 2: Creating an Address bean

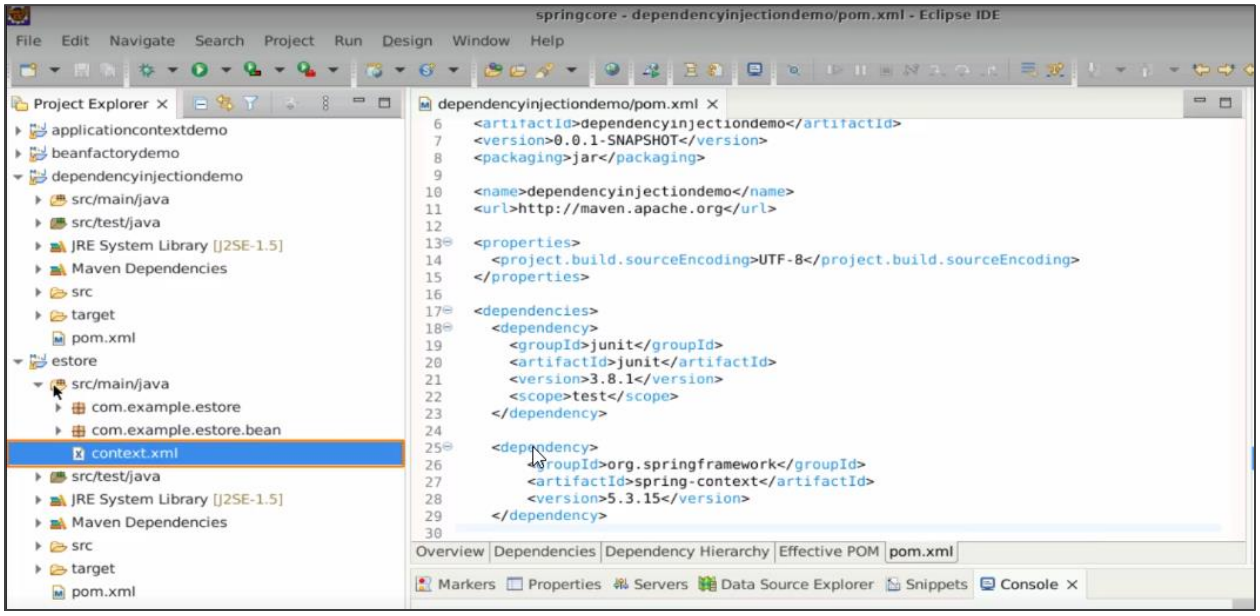
2.1 Copy the **spring-context** dependency from the pom.xml of the **estore** project and paste it into the **pom.xml** file of the current project.


springcore - dependencyinjectiondemo/pom.xml - Eclipse IDE
File Edit Navigate Search Project Run Design Window Help
Project Explorer
applicationcontextdemo
beanfactorydemo
dependencyinjectiondemo
src/main/java
src/test/java
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml
estore
src/main/java
src/test/java
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml
dependencyinjectiondemo/pom.xml
Overview Dependencies Dependency Hierarchy Effective POM pom.xml

```
<artifactId>dependencyinjectiondemo</artifactId>
<version>0.0.1-SNAPSHOT</version>
<packaging>jar</packaging>

<name>dependencyinjectiondemo</name>
<url>http://maven.apache.org</url>

<properties>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
</properties>

<dependencies>
  <dependency>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <version>3.8.1</version>
    <scope>test</scope>
  </dependency>

  <dependency>
      <groupId>org.springframework</groupId>
      <artifactId>spring-context</artifactId>
      <version>5.3.15</version>
  </dependency>
```

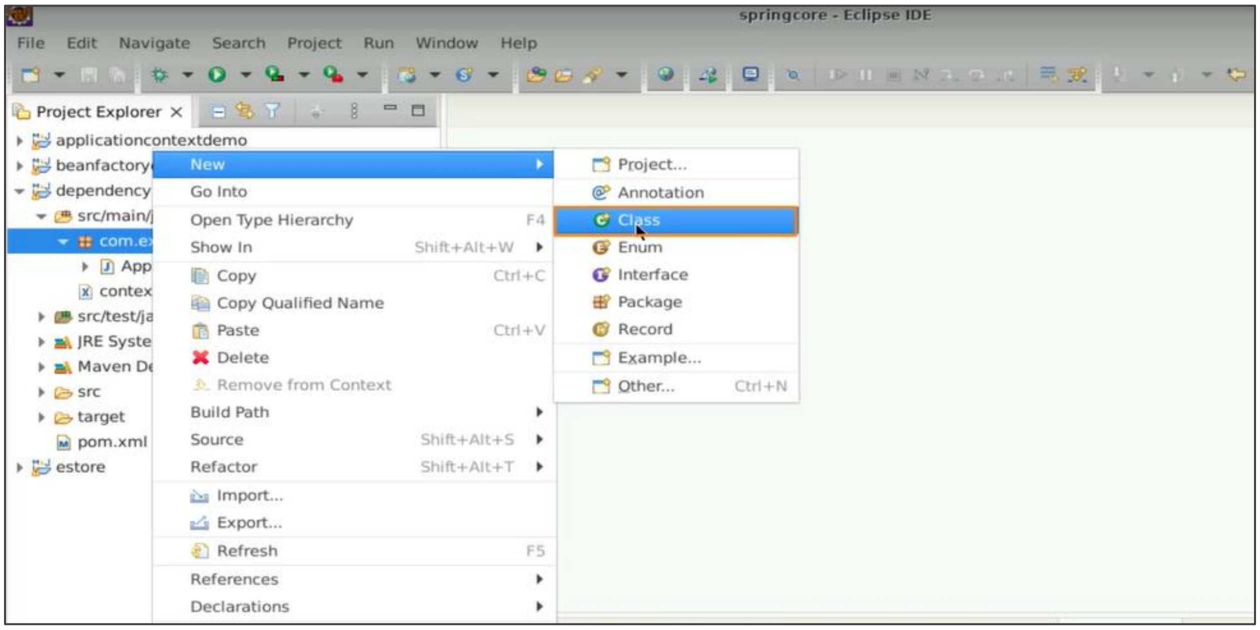
Note: Please refer to the previous demos for instructions on creating a Maven project with the Spring framework.

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

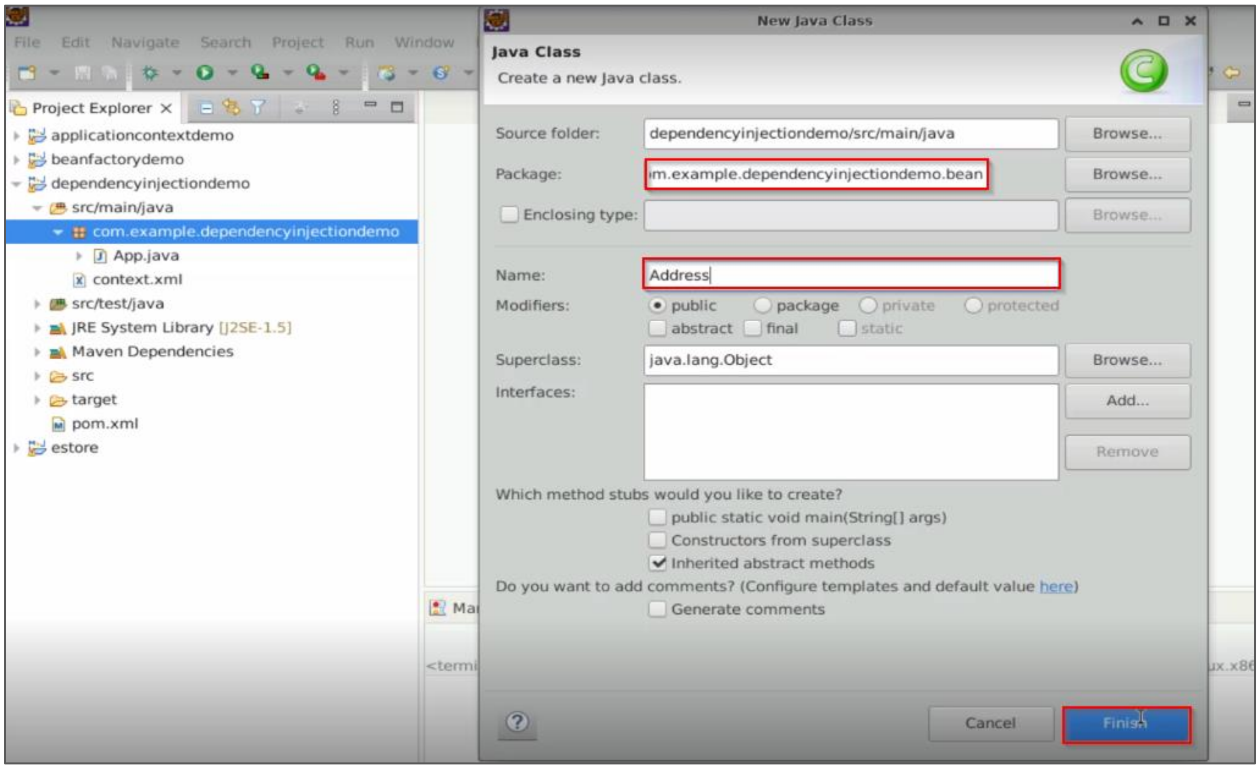

springcore - dependencyinjectiondemo/pom.xml - Eclipse IDE
File Edit Navigate Search Project Run Design Window Help
Project Explorer
applicationcontextdemo
beanfactorydemo
dependencyinjectiondemo
src/main/java
src/test/java
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml
estore
src/main/java
com.example.estore
com.example.estore.bean
context.xml
src/test/java
JRE System Library [J2SE-1.5]
Maven Dependencies
src
target
pom.xml
dependencyinjectiondemo/pom.xml
Overview Dependencies Dependency Hierarchy Effective POM pom.xml
Markers Properties Servers Data Source Explorer Snippets Console

```
  <artifactId>dependencyinjectiondemo</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>

  <name>dependencyinjectiondemo</name>
  <url>http://maven.apache.org</url>

  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  </properties>

  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>3.8.1</version>
      <scope>test</scope>
    </dependency>

    <dependency>
        <groupId>org.springframework</groupId>
        <artifactId>spring-context</artifactId>
        <version>5.3.15</version>
    </dependency>

```

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

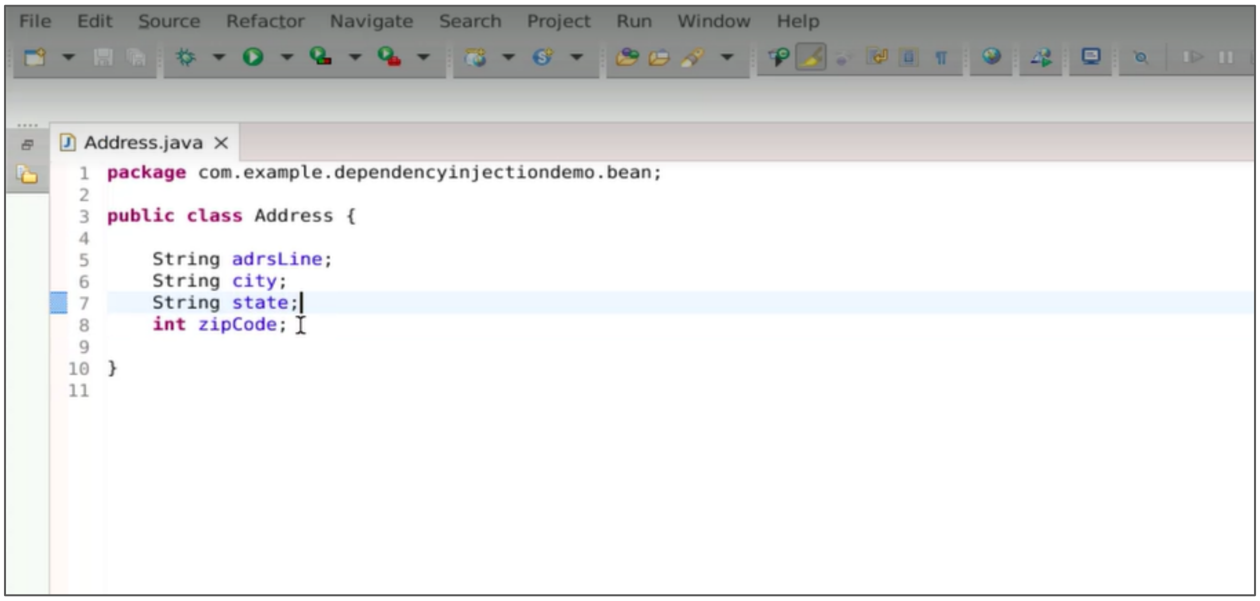


2.4 Provide a name for the class, such as **Address**, and append **.bean** to the package name. Click **Finish** to create the class.



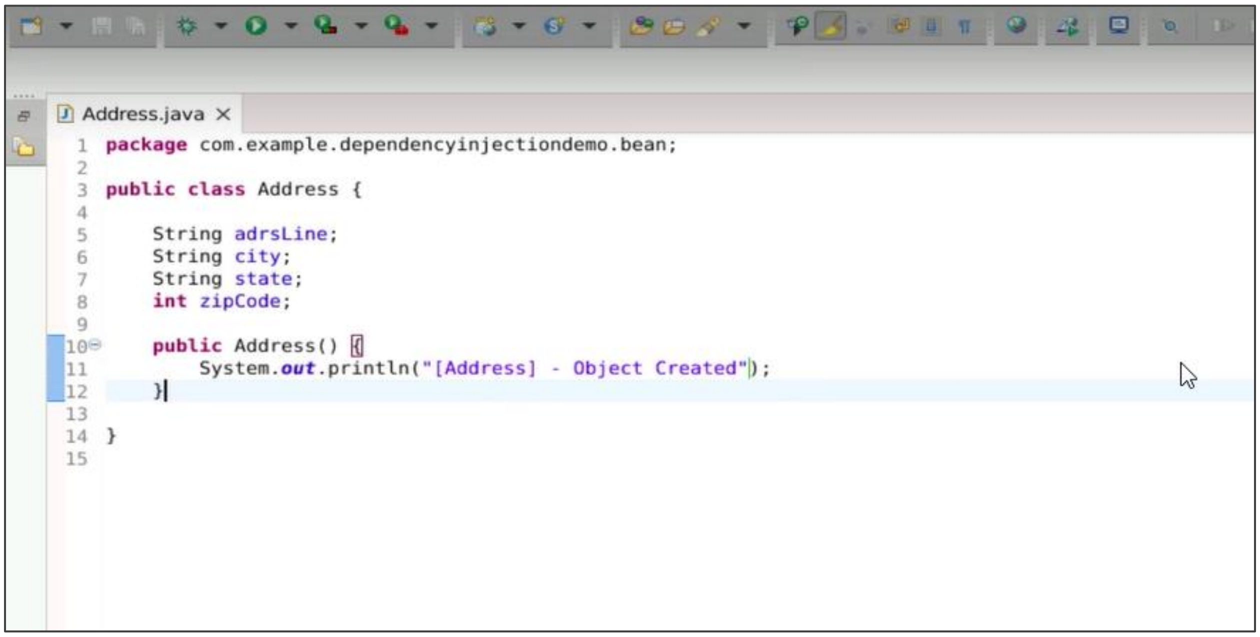
Source folder: dependencyinjectiondemo/src/main/java
Package: ım.example.dependencyinjectiondemo.bean
Enclosing type: [ ]
Name: Address
Modifiers: [✓ public] package private protected; abstract final static
Superclass: java.lang.Object
Interfaces:
Which method stubs would you like to create?
[ ] public static void main(String[] args)
[ ] Constructors from superclass
[✓] Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
[ ] Generate comments

2.5 In the Address class, define attributes such as **adrsLine**, **city**, **state**, and **zipCode**


File Edit Source Refactor Navigate Search Project Run Window Help
Address.java

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

public class Address {

    String adrsLine;
    String city;
    String state;
    int zipCode;

}
```

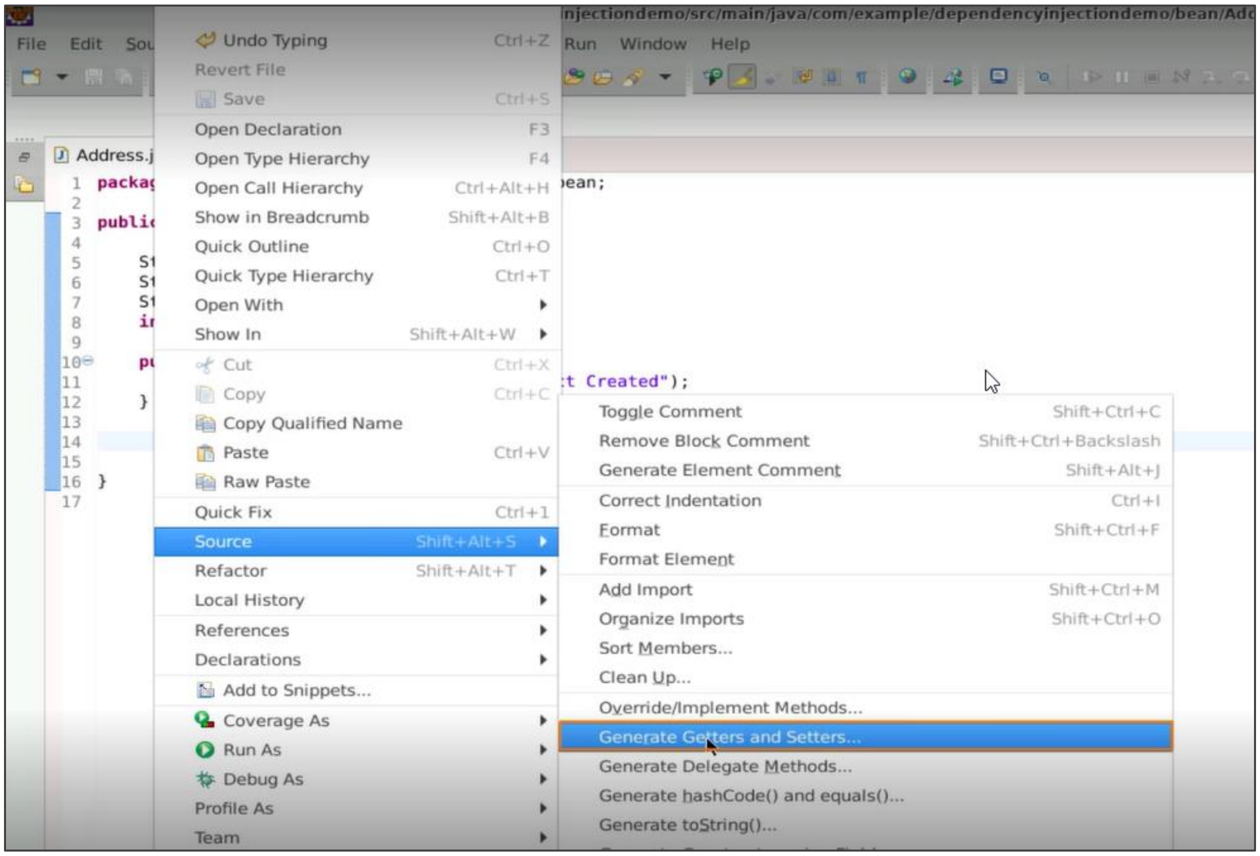
2.6 Create a default constructor for the class and include a print statement: **[Address] - Object Created**



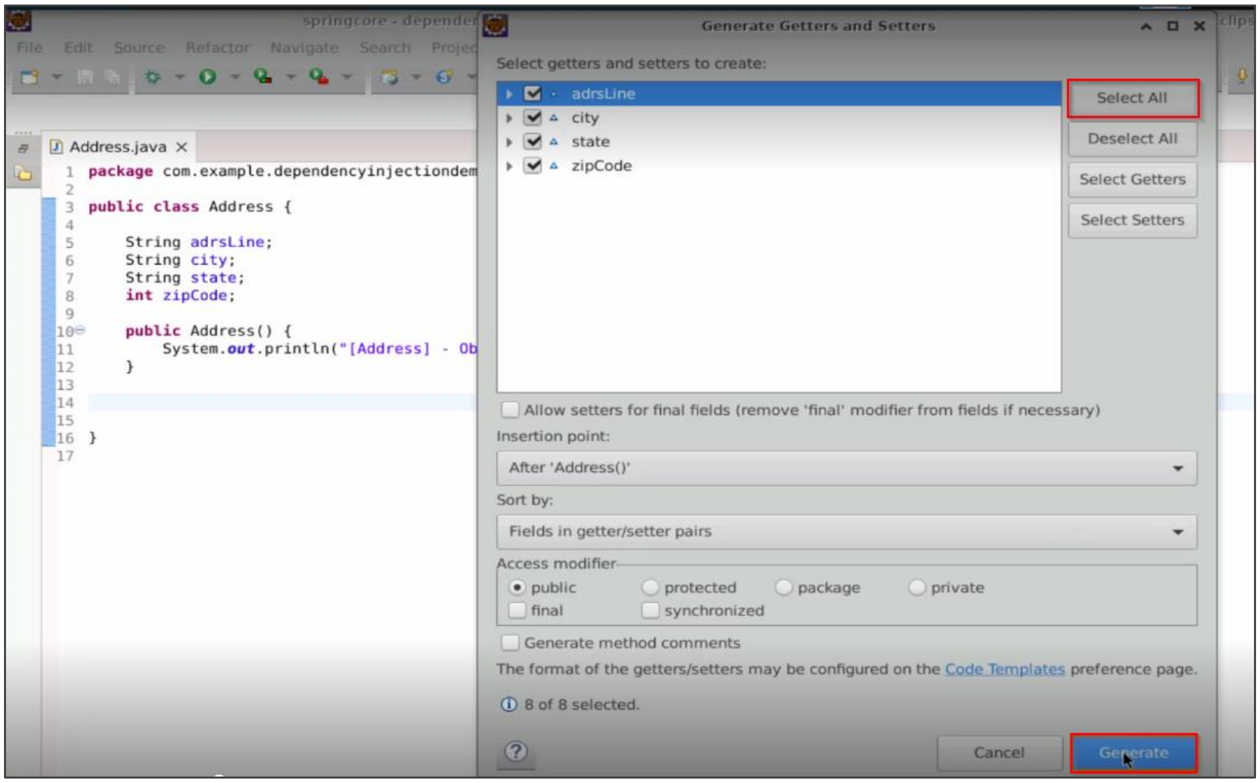
Address.java

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

public class Address {

    String adrsLine;
    String city;
    String state;
    int zipCode;

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

}
```

2.7 Right-click inside the **Address** class, select **Source,** and choose **Generate Getters and Setters**



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



```java
package com.example.dependencyinjectiondem

public class Address {

    String adrsLine;
    String city;
    String state;
    int zipCode;

    public Address() {
        System.out.println("[Address] - Ob
    }



}
```

Select getters and setters to create:

- [✓] adrsLine
- [✓] city
- [✓] state
- [✓] zipCode

Allow setters for final fields (remove 'final' modifier from fields if necessary)

Insertion point: After 'Address()'

Sort by: Fields in getter/setter pairs

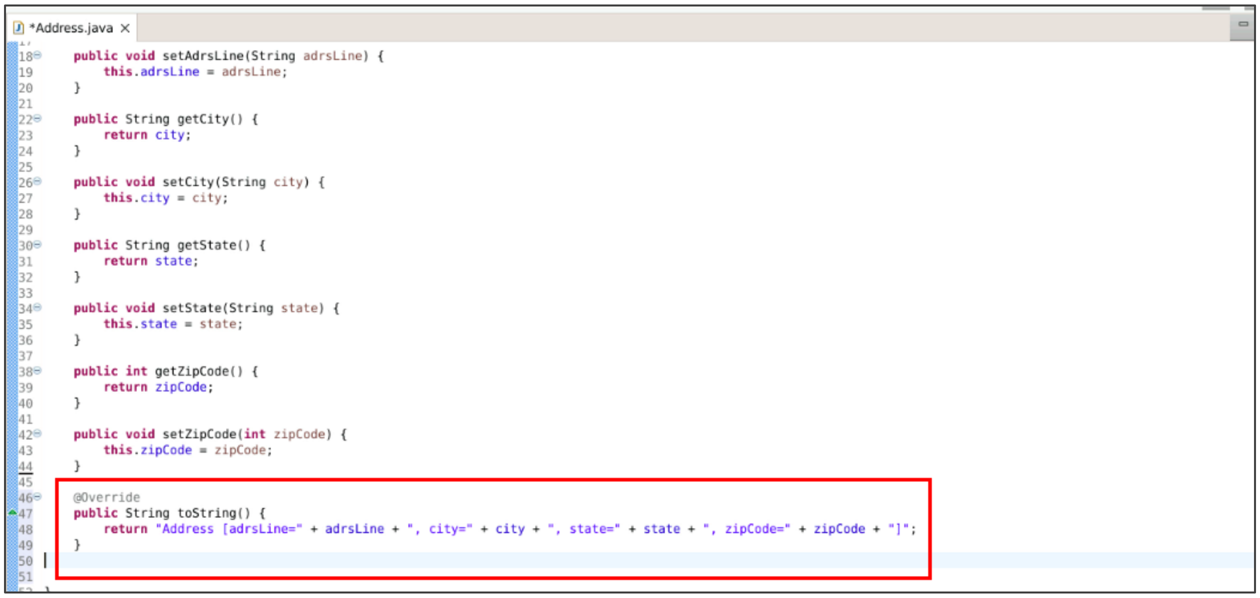
Access modifier: [✓ public] protected package private; final synchronized

Generate method comments

The format of the getters/setters may be configured on the Code Templates preference page.

8 of 8 selected.

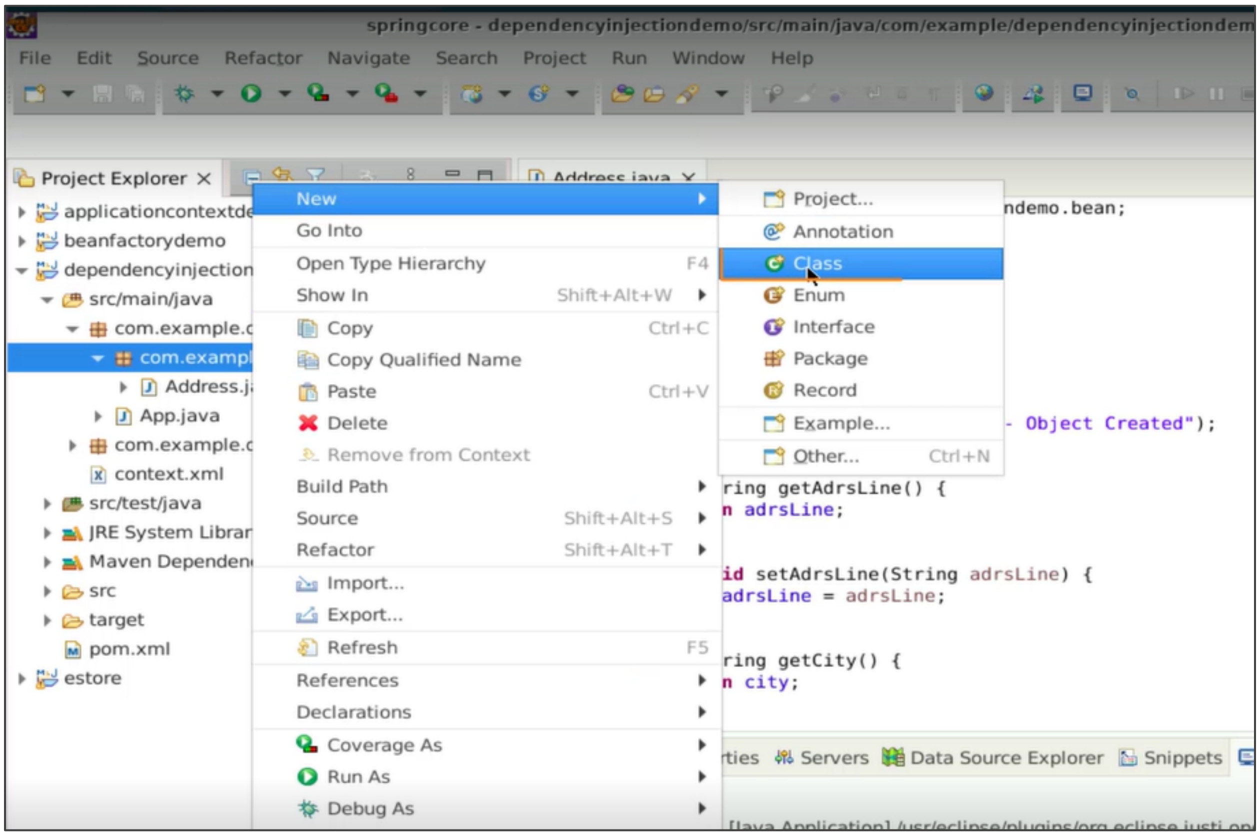
2.9 Repeat the previous step to generate a **toString()** method that returns all the attributes



```java
    public void setAdrsLine(String adrsLine) {
        this.adrsLine = adrsLine;
    }

    public String getCity() {
        return city;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public String getState() {
        return state;
    }

    public void setState(String state) {
        this.state = state;
    }

    public int getZipCode() {
        return zipCode;
    }

    public void setZipCode(int zipCode) {
        this.zipCode = zipCode;
    }

    @Override
    public String toString() {
        return "Address [adrsLine=" + adrsLine + ", city=" + city + ", state=" + state + ", zipCode=" + zipCode + "]";
    }
```

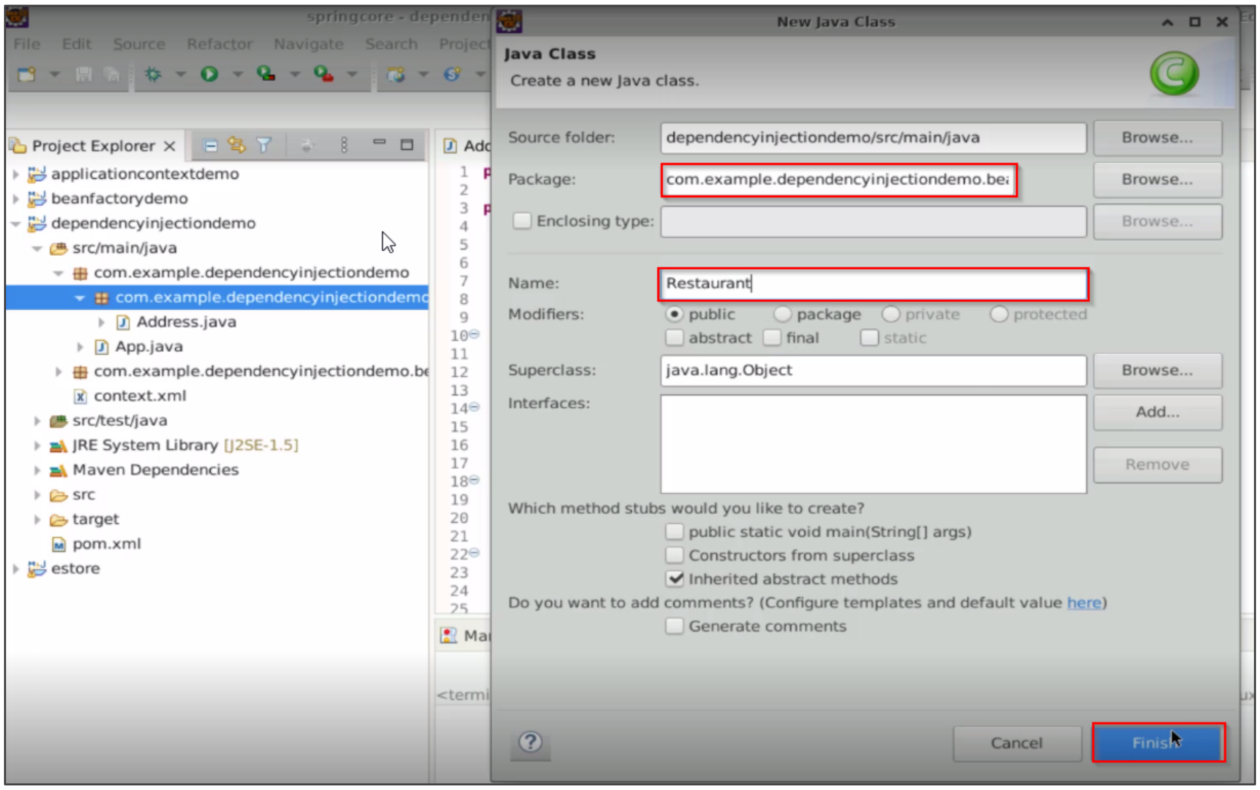
Step 3: Creating the Restaurant bean

3.1 Right-click on the bean package, select **New**, and click **Class**



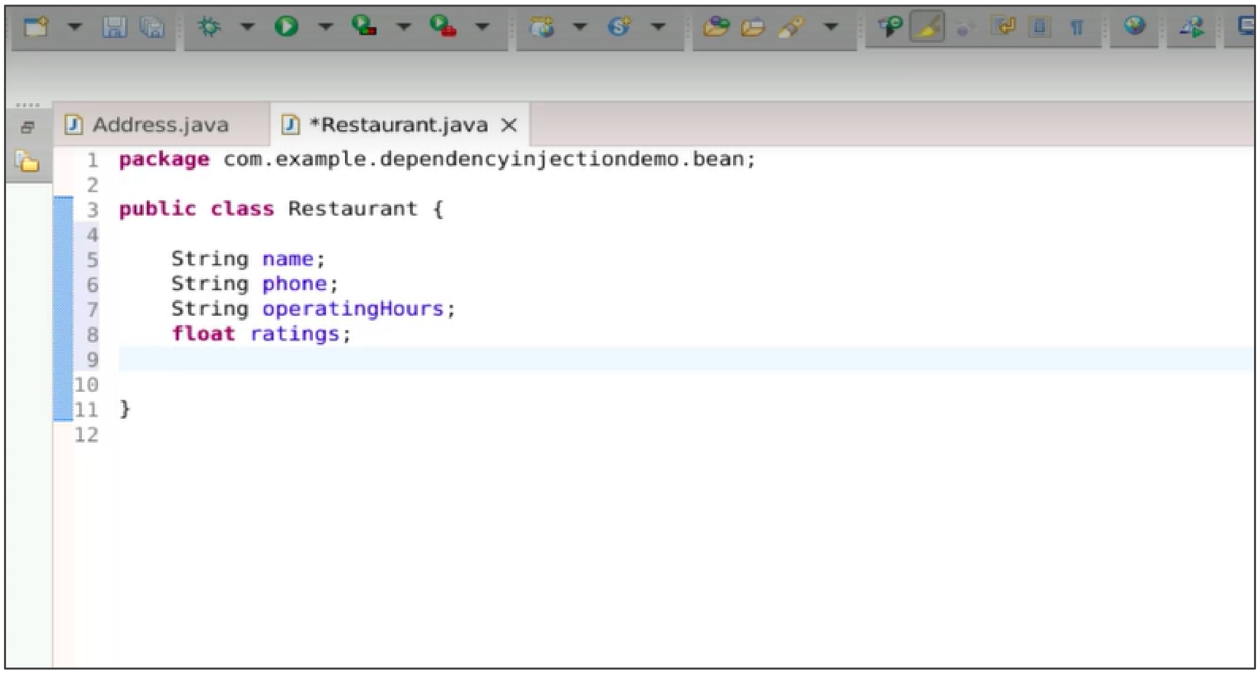
```
ndemo.bean;
- Object Created");
ring getAdrsLine() {
n adrsLine;
id setAdrsLine(String adrsLine) {
adrsLine = adrsLine;
ring getCity() {
n city;
```

3.2 Enter **Restaurant** in the Name field click on **Finish**



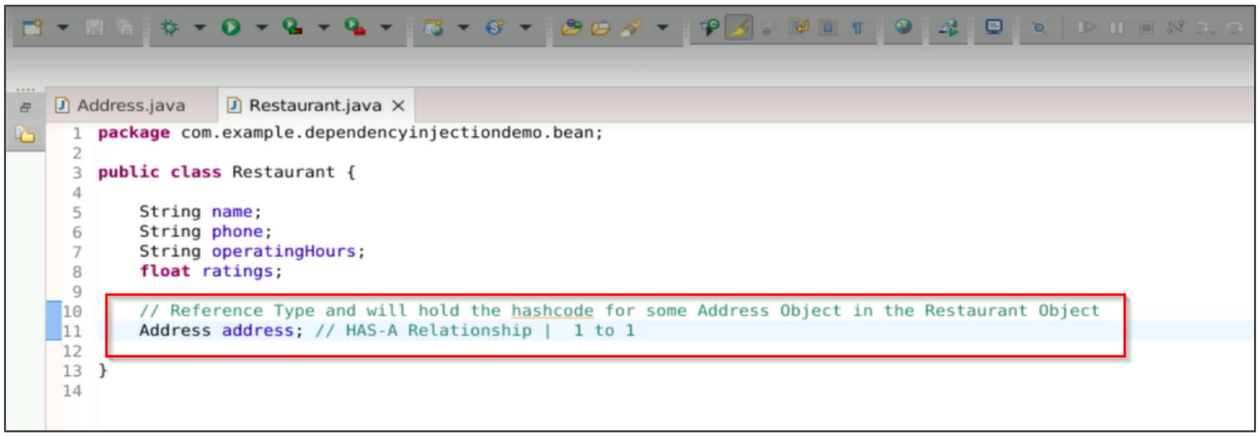
Source folder: dependencyinjectiondemo/src/main/java
Package: com.example.dependencyinjectiondemo.be
Enclosing type: [ ]
Name: Restaurant
Modifiers: [✓ public] package private protected
[ ] abstract [ ] final [ ] static
Superclass: java.lang.Object
Interfaces:
Which method stubs would you like to create?
[ ] public static void main(String[] args)
[ ] Constructors from superclass
[✓] Inherited abstract methods
Do you want to add comments? (Configure templates and default value here)
[ ] Generate comments

3.3 In the Restaurant class, define attributes such as **name**, **phone**, **operatingHours**, and **ratings**



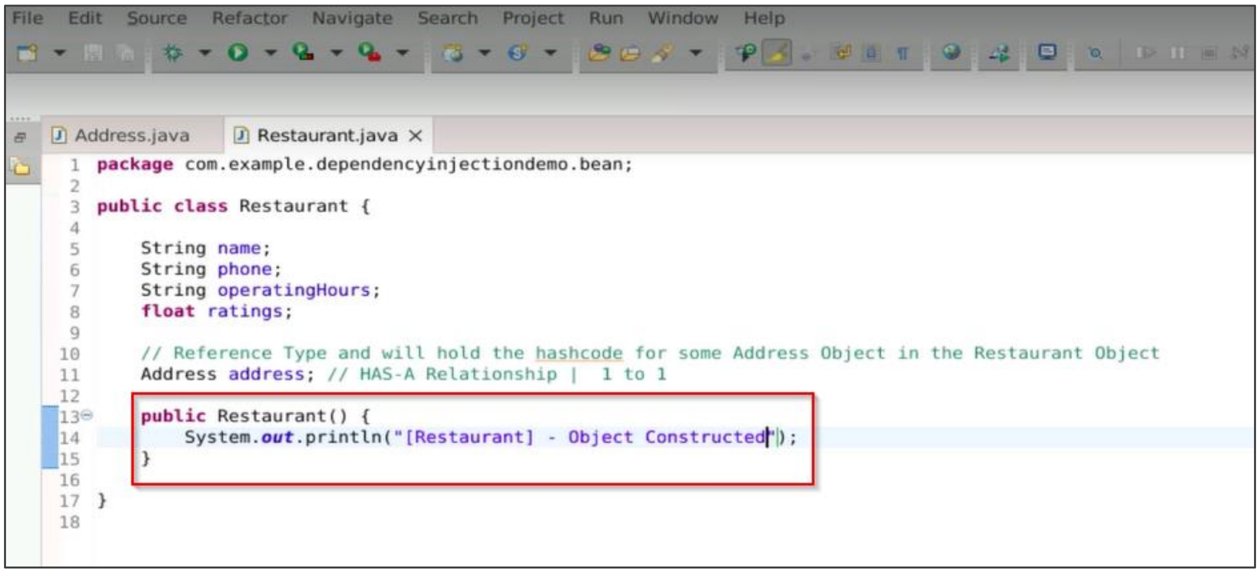
```java
package com.example.dependencyinjectiondemo.bean;

public class Restaurant {

    String name;
    String phone;
    String operatingHours;
    float ratings;


}
```

3.4 Create the **address** reference variable of type **Address** object to represent the restaurant's address



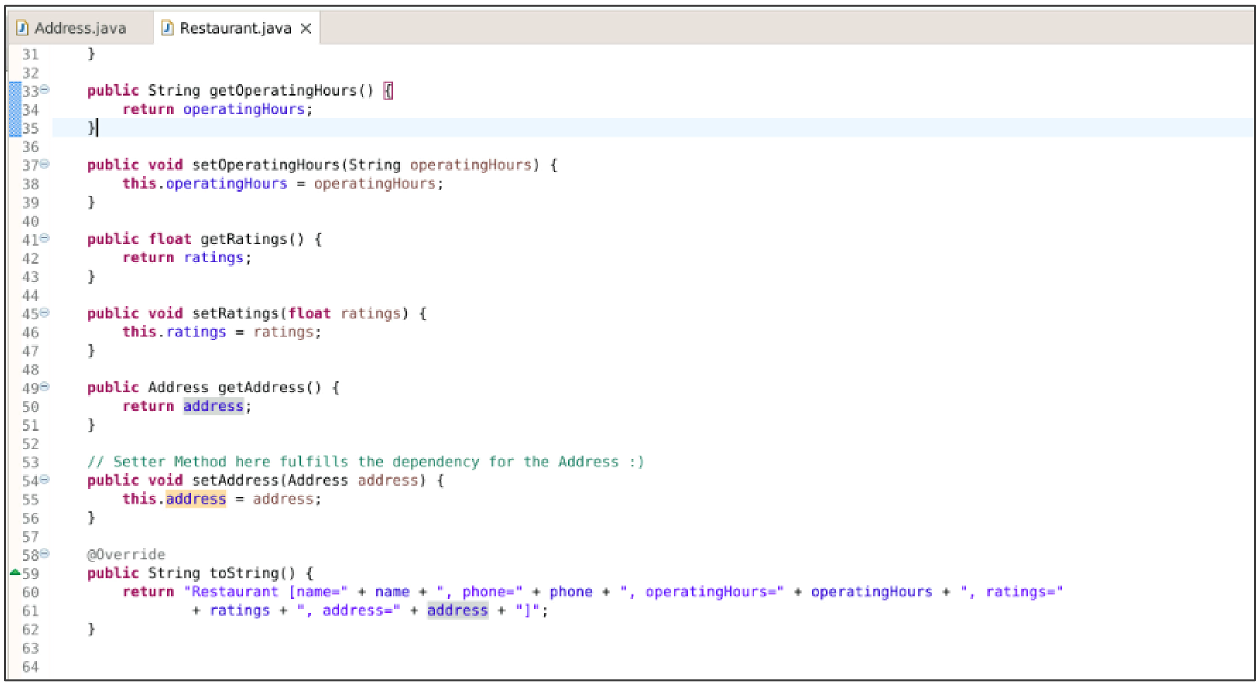
```java
package com.example.dependencyinjectiondemo.bean;

public class Restaurant {

    String name;
    String phone;
    String operatingHours;
    float ratings;

    // Reference Type and will hold the hashcode for some Address Object in the Restaurant Object
    Address address; // HAS-A Relationship |  1 to 1

}
```

3.5 Create a default constructor for the class and include a print statement: **[Restaurant] - Object Created**


File Edit Source Refactor Navigate Search Project Run Window Help
Address.java
Restaurant.java

```
package com.example.dependencyinjectiondemo.bean;

public class Restaurant {

    String name;
    String phone;
    String operatingHours;
    float ratings;

    // Reference Type and will hold the hashcode for some Address Object in the Restaurant Object
    Address address; // HAS-A Relationship |  1 to 1

    public Restaurant() {
        System.out.println("[Restaurant] - Object Constructed");
    }

}
```

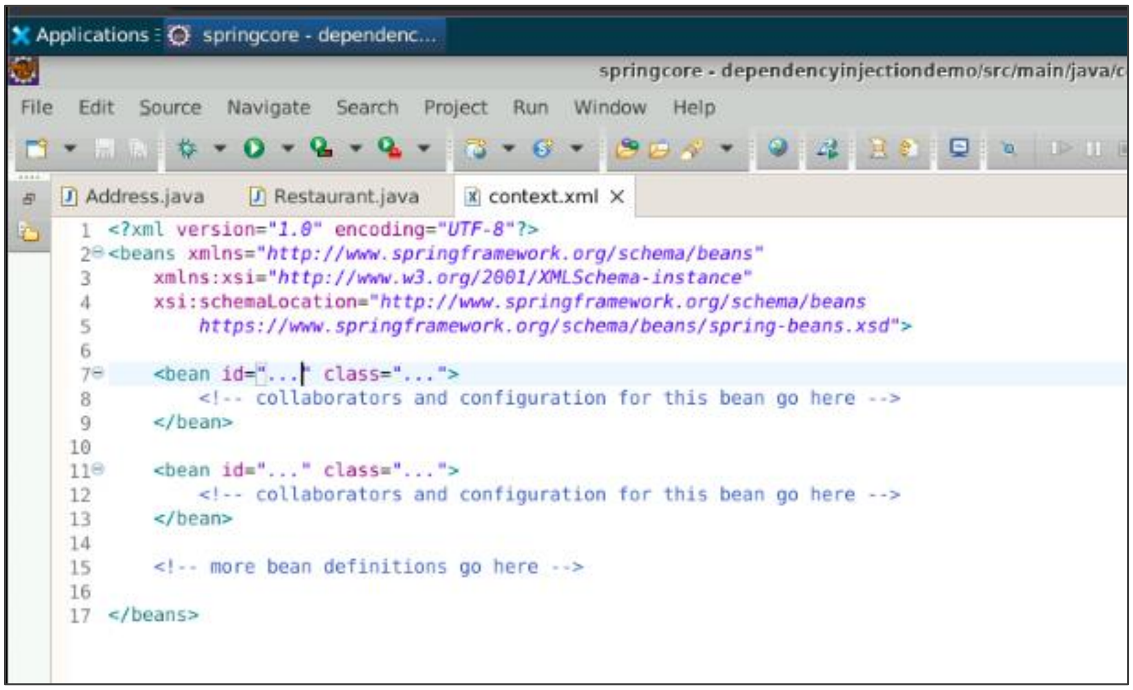
3.6 Generate getters and setters for the attributes and a **toString()** method for the class



```
    }

    public String getOperatingHours() {
        return operatingHours;
    }

    public void setOperatingHours(String operatingHours) {
        this.operatingHours = operatingHours;
    }

    public float getRatings() {
        return ratings;
    }

    public void setRatings(float ratings) {
        this.ratings = ratings;
    }

    public Address getAddress() {
        return address;
    }

    // Setter Method here fulfills the dependency for the Address :)
    public void setAddress(Address address) {
        this.address = address;
    }

    @Override
    public String toString() {
        return "Restaurant [name=" + name + ", phone=" + phone + ", operatingHours=" + operatingHours + ", ratings="
                + ratings + ", address=" + address + "]";
    }

```

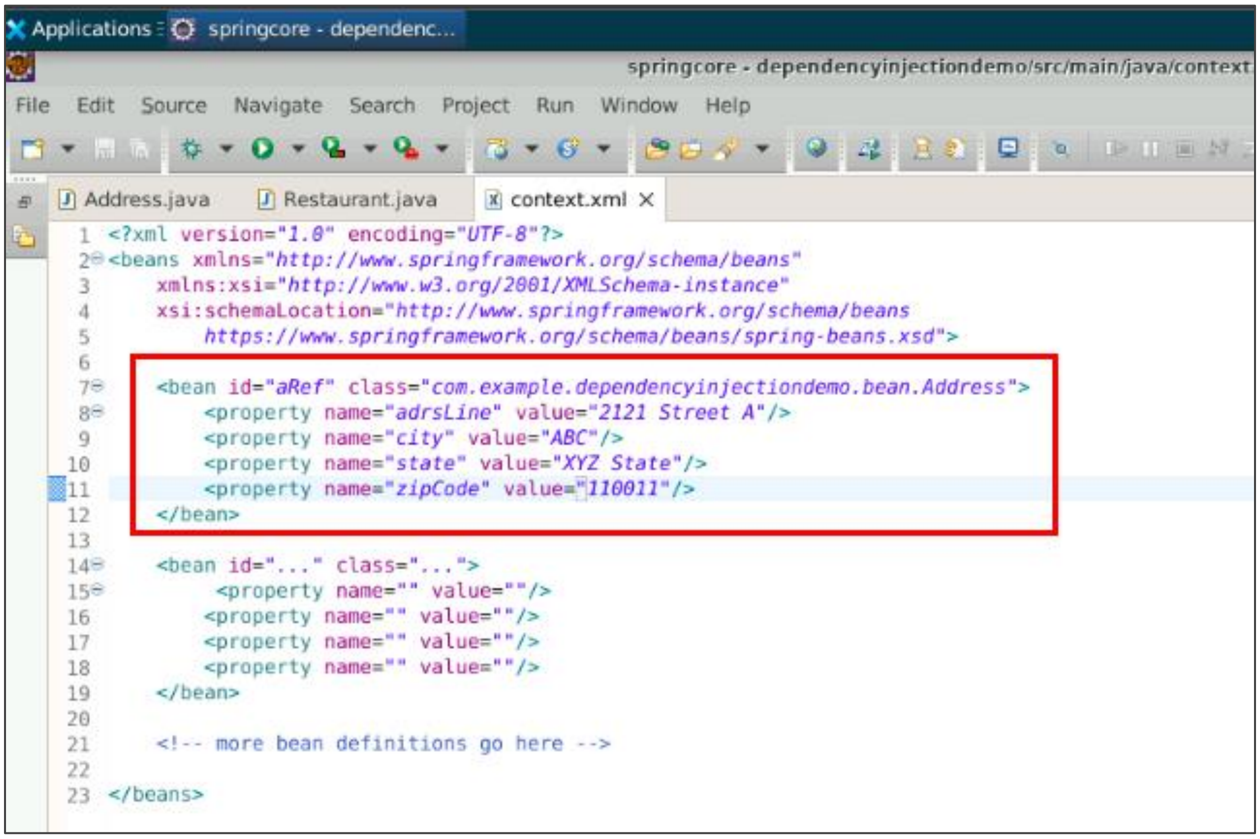
Step 4: Configuring the context.xml for beans

4.1 Open the **context.xml** file


Applications
springcore - dependenc...
springcore - dependencyinjectiondemo/src/main/java/c
File Edit Source Navigate Search Project Run Window Help
Address.java
Restaurant.java
context.xml

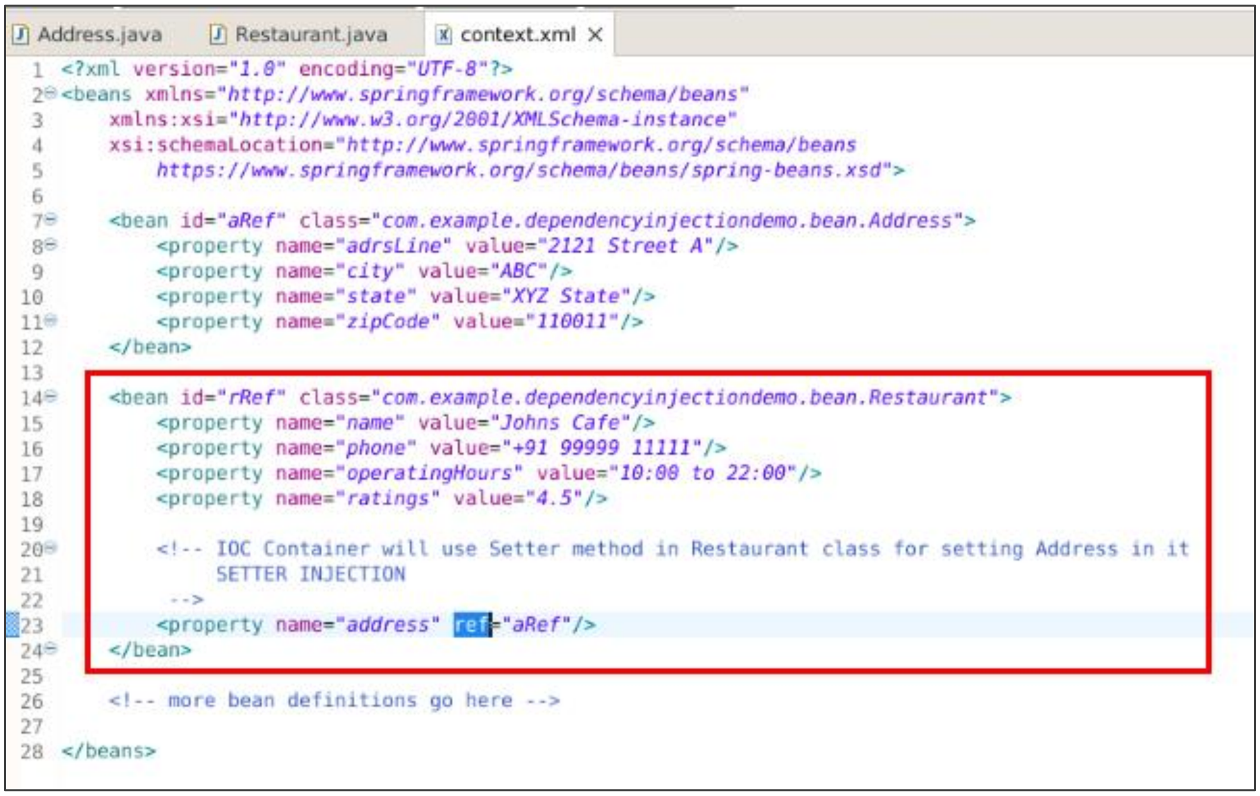
```xml
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="..." class="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>

    <bean id="..." class="...">
        <!-- collaborators and configuration for this bean go here -->
    </bean>

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

</beans>
```

4.2 Define a **bean** for the **Address** class with an id **aRef** and set the key-value pairs for its attributes


Applications
springcore - dependenc...
springcore - dependencyinjectiondemo/src/main/java/context
File Edit Source Navigate Search Project Run Window Help
Address.java
Restaurant.java
context.xml

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="aRef" class="com.example.dependencyinjectiondemo.bean.Address">
        <property name="adrsLine" value="2121 Street A"/>
        <property name="city" value="ABC"/>
        <property name="state" value="XYZ State"/>
        <property name="zipCode" value="110011"/>
    </bean>

    <bean id="..." class="...">
         <property name="" value=""/>
        <property name="" value=""/>
        <property name="" value=""/>
        <property name="" value=""/>
    </bean>

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

</beans>
```

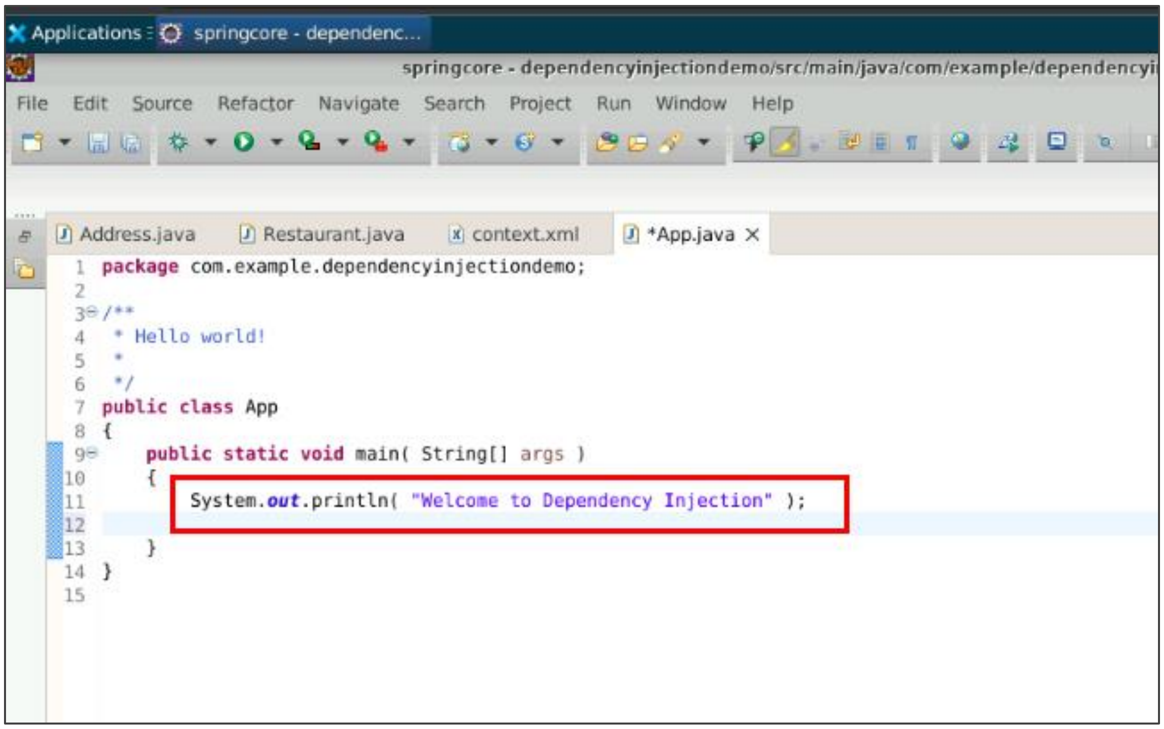
4.3 Define another bean for the **Restaurant** class with an id **rRef** and set the key-value pairs for its attributes. Use the **ref** attribute to refer to the Address bean.



```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="aRef" class="com.example.dependencyinjectiondemo.bean.Address">
        <property name="adrsLine" value="2121 Street A"/>
        <property name="city" value="ABC"/>
        <property name="state" value="XYZ State"/>
        <property name="zipCode" value="110011"/>
    </bean>

    <bean id="rRef" class="com.example.dependencyinjectiondemo.bean.Restaurant">
        <property name="name" value="Johns Cafe"/>
        <property name="phone" value="+91 99999 11111"/>
        <property name="operatingHours" value="10:00 to 22:00"/>
        <property name="ratings" value="4.5"/>

        <!-- IOC Container will use Setter method in Restaurant class for setting Address in it
             SETTER INJECTION
          -->
        <property name="address" ref="aRef"/>
    </bean>

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

</beans>
```

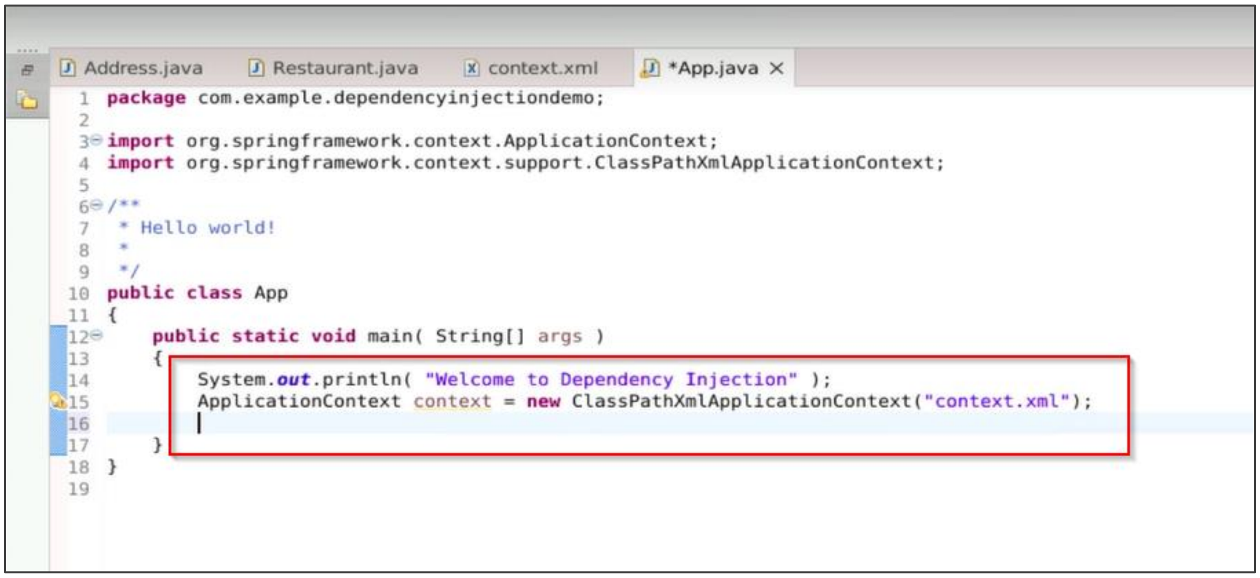
Step 5: Writing IOC code in App.java

5.1 Navigate to the **App.java** class and update the print statement to **Welcome to Dependency Injection**


Applications
springcore - dependenc...
springcore - dependencyinjectiondemo/src/main/java/com/example/dependencyi
File Edit Source Refactor Navigate Search Project Run Window Help
Address.java
Restaurant.java
context.xml
*App.java

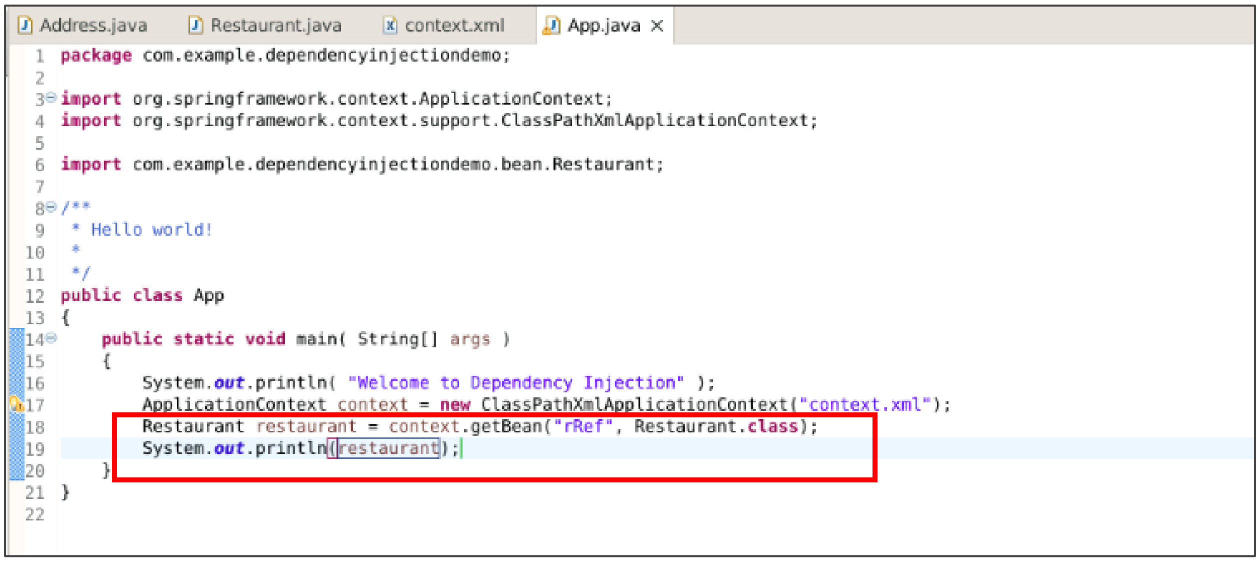
```
package com.example.dependencyinjectiondemo;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Welcome to Dependency Injection" );

    }
}

```

5.2 Create an instance of the **ApplicationContext** interface using the **ClassPathXmlApplicationContext**. Pass the **context.xml** file to the **ApplicationContext** constructor



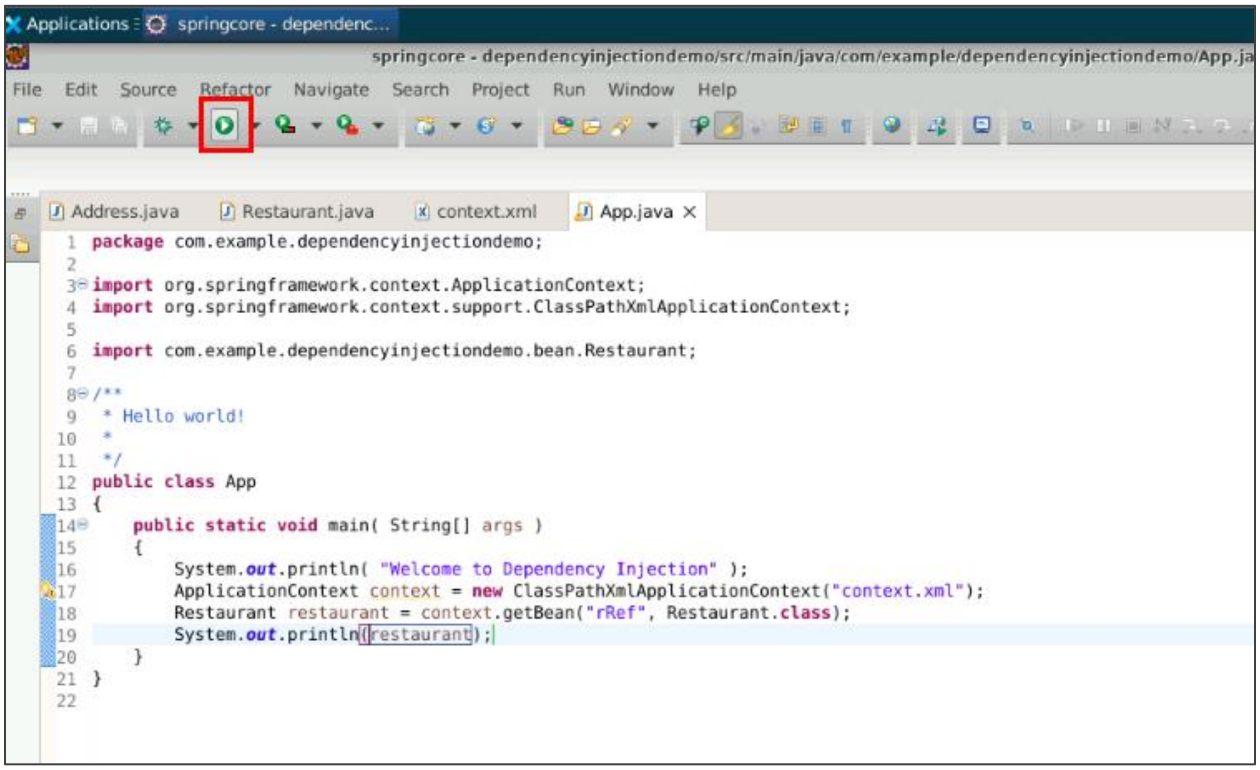
```
package com.example.dependencyinjectiondemo;

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( "Welcome to Dependency Injection" );
        ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");

    }
}
```

5.3 Use the **getBean()** method to retrieve the **Restaurant** bean instance by its reference ID and print the restaurant

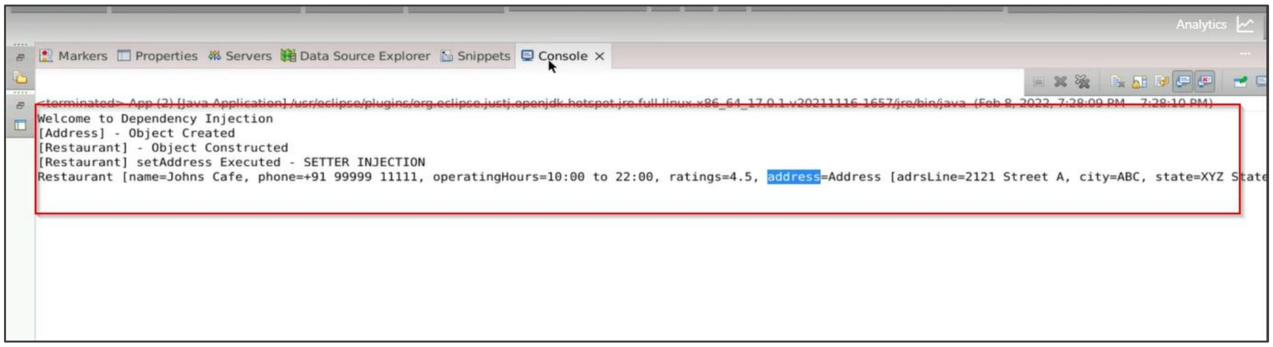


```
package com.example.dependencyinjectiondemo;

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

import com.example.dependencyinjectiondemo.bean.Restaurant;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Welcome to Dependency Injection" );
        ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");
        Restaurant restaurant = context.getBean("rRef", Restaurant.class);
        System.out.println(restaurant);
    }
}
```

5.4 Run the project by clicking on the green run button


Applications
springcore - dependenc...
springcore - dependencyinjectiondemo/src/main/java/com/example/dependencyinjectiondemo/App.ja
File Edit Source Refactor Navigate Search Project Run Window Help
Address.java
Restaurant.java
context.xml
App.java

```java
package com.example.dependencyinjectiondemo;

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

import com.example.dependencyinjectiondemo.bean.Restaurant;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Welcome to Dependency Injection" );
        ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");
        Restaurant restaurant = context.getBean("rRef", Restaurant.class);
        System.out.println(restaurant);
    }
}
```

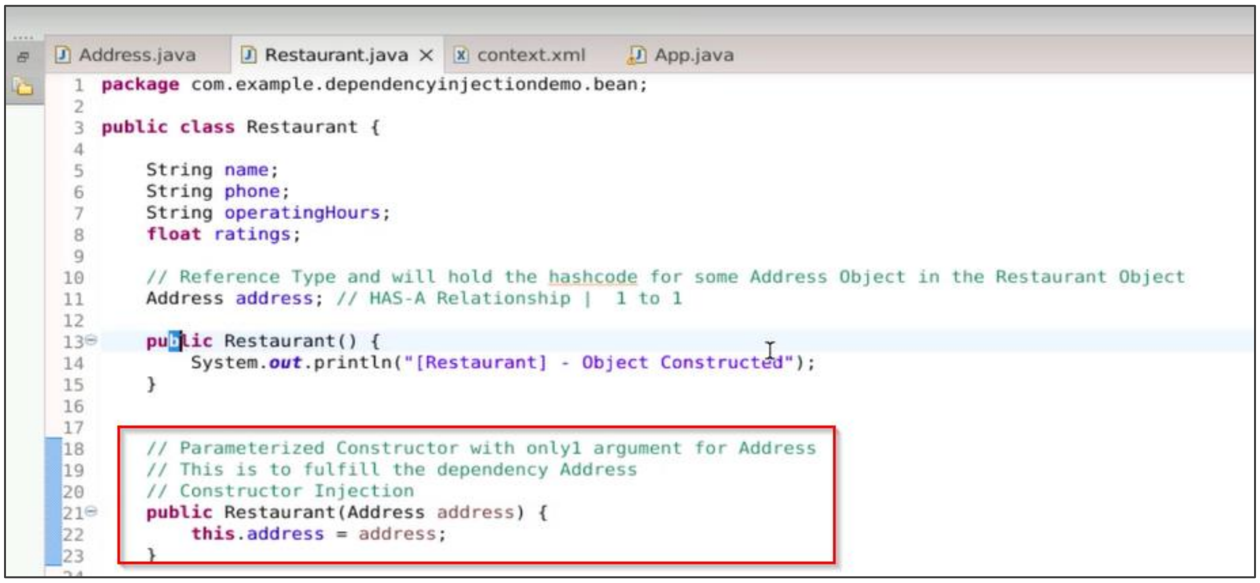


```
~~<terminated> App (2) [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java (Feb 8, 2022, 7:28:09 PM – 7:28:10 PM)~~
Welcome to Dependency Injection
[Address] - Object Created
[Restaurant] - Object Constructed
[Restaurant] setAddress Executed - SETTER INJECTION
Restaurant [name=Johns Cafe, phone=+91 99999 11111, operatingHours=10:00 to 22:00, ratings=4.5, address=Address [adrsLine=2121 Street A, city=ABC, state=XYZ State
```

You can see that both the Address and Restaurant objects are created, as Address is a dependency of the Restaurant bean. All the values are printed on the console.

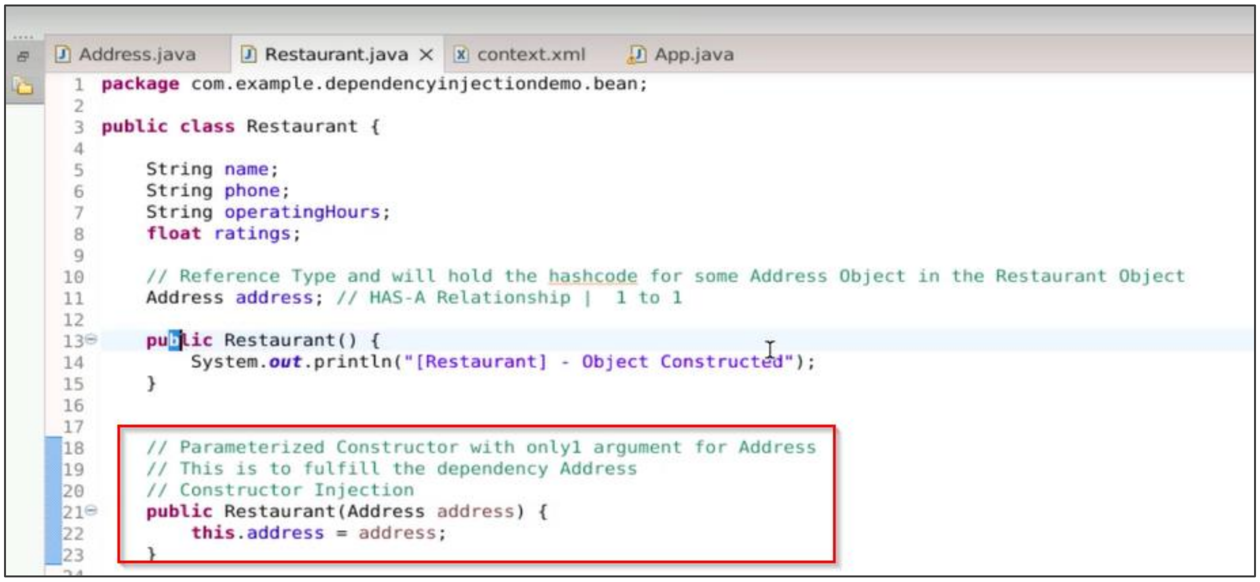
Step 6: Creating a parameterized constructor

6.1 Open the **Restaurant.java** class



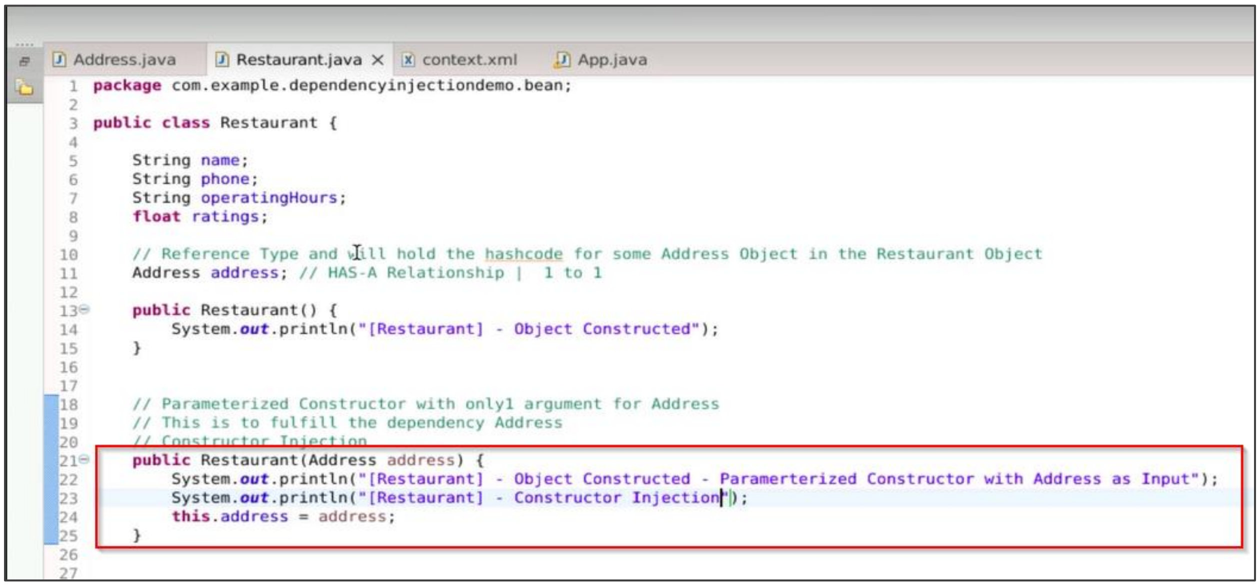
```java
package com.example.dependencyinjectiondemo.bean;

public class Restaurant {

    String name;
    String phone;
    String operatingHours;
    float ratings;

    // Reference Type and will hold the hashcode for some Address Object in the Restaurant Object
    Address address; // HAS-A Relationship |  1 to 1

    public Restaurant() {
        System.out.println("[Restaurant] - Object Constructed");
    }


    // Parameterized Constructor with only1 argument for Address
    // This is to fulfill the dependency Address
    // Constructor Injection
    public Restaurant(Address address) {
        this.address = address;
    }
```

6.2 Create a parameterized constructor for the **Restaurant** that takes an Address input



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

public class Restaurant {

    String name;
    String phone;
    String operatingHours;
    float ratings;

    // Reference Type and will hold the hashcode for some Address Object in the Restaurant Object
    Address address; // HAS-A Relationship |  1 to 1

    public Restaurant() {
        System.out.println("[Restaurant] - Object Constructed");
    }


    // Parameterized Constructor with only1 argument for Address
    // This is to fulfill the dependency Address
    // Constructor Injection
    public Restaurant(Address address) {
        this.address = address;
    }
```

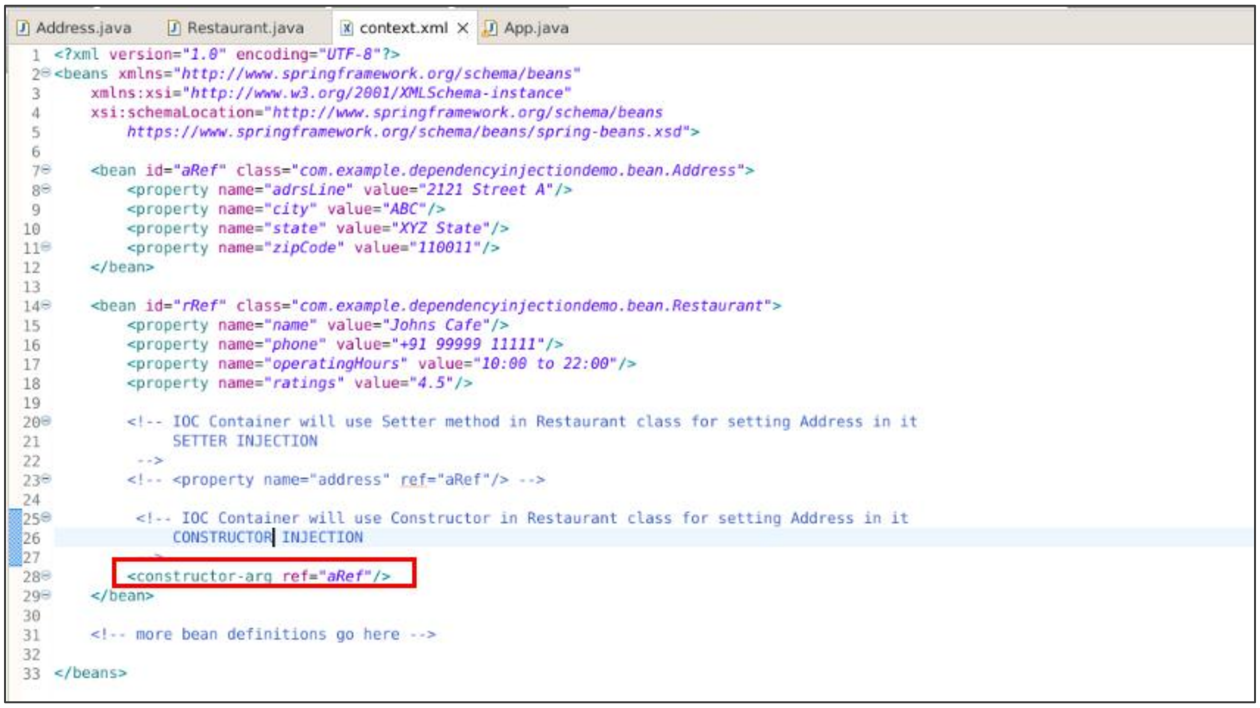
6.3 Add two print statements in the constructor: **[Restaurant] - Object Constructed** and **[Restaurant] - Constructor Injection**



Address.java | Restaurant.java | context.xml | App.java

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

public class Restaurant {

    String name;
    String phone;
    String operatingHours;
    float ratings;

    // Reference Type and will hold the hashcode for some Address Object in the Restaurant Object
    Address address; // HAS-A Relationship |  1 to 1

    public Restaurant() {
        System.out.println("[Restaurant] - Object Constructed");
    }


    // Parameterized Constructor with only1 argument for Address
    // This is to fulfill the dependency Address
    // Constructor Injection
    public Restaurant(Address address) {
        System.out.println("[Restaurant] - Object Constructed - Paramerterized Constructor with Address as Input");
        System.out.println("[Restaurant] - Constructor Injection");
        this.address = address;
    }

```

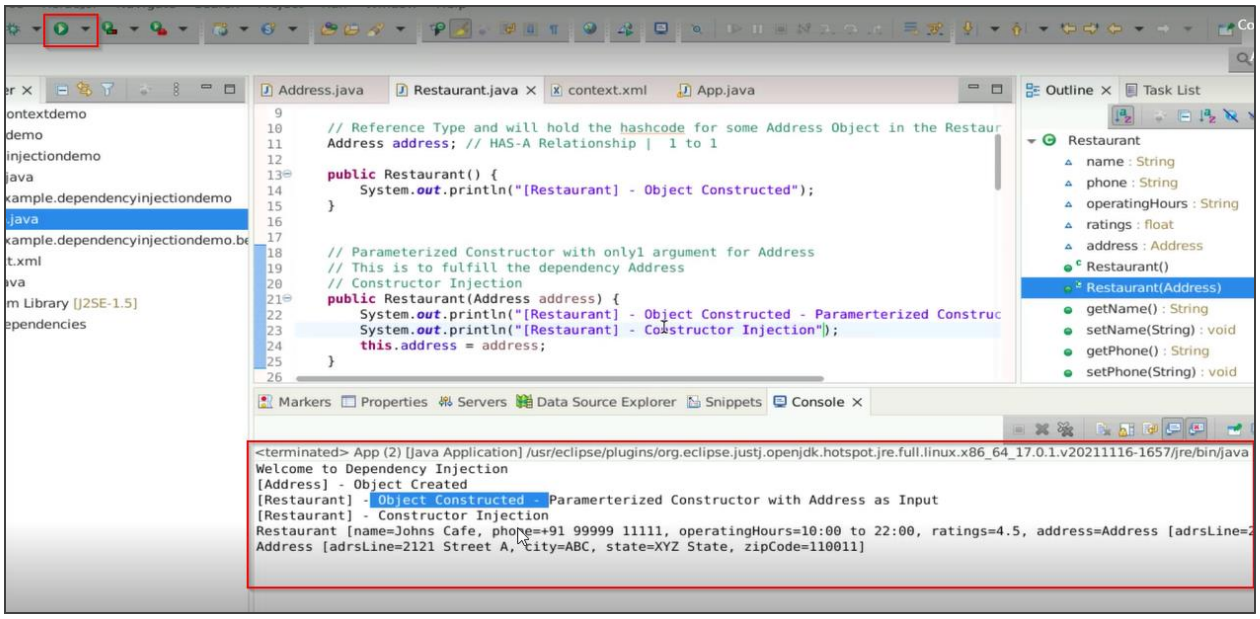
6.4 Update the **context.xml** file to use the **constructor-arg** tag instead of the **property** tag, providing the reference ID for the **Address** bean



Address.java | Restaurant.java | context.xml | App.java

```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"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
        https://www.springframework.org/schema/beans/spring-beans.xsd">

    <bean id="aRef" class="com.example.dependencyinjectiondemo.bean.Address">
        <property name="adrsLine" value="2121 Street A"/>
        <property name="city" value="ABC"/>
        <property name="state" value="XYZ State"/>
        <property name="zipCode" value="110011"/>
    </bean>

    <bean id="rRef" class="com.example.dependencyinjectiondemo.bean.Restaurant">
        <property name="name" value="Johns Cafe"/>
        <property name="phone" value="+91 99999 11111"/>
        <property name="operatingHours" value="10:00 to 22:00"/>
        <property name="ratings" value="4.5"/>

        <!-- IOC Container will use Setter method in Restaurant class for setting Address in it
            SETTER INJECTION
         -->
        <!-- <property name="address" ref="aRef"/> -->

         <!-- IOC Container will use Constructor in Restaurant class for setting Address in it
            CONSTRUCTOR INJECTION
            -->
        <constructor-arg ref="aRef"/>
    </bean>

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

</beans>
```

6.5 Run the project



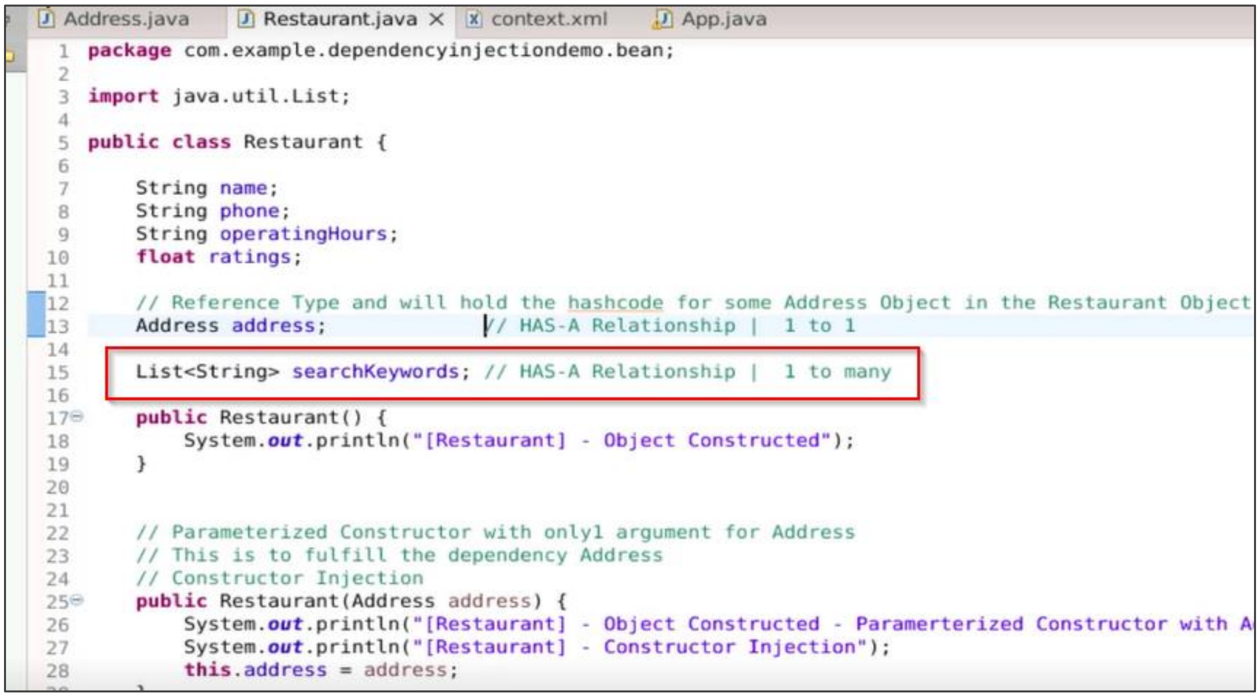
```java
    // Reference Type and will hold the hashcode for some Address Object in the Restaur
    Address address; // HAS-A Relationship |  1 to 1

    public Restaurant() {
        System.out.println("[Restaurant] - Object Constructed");
    }


    // Parameterized Constructor with only1 argument for Address
    // This is to fulfill the dependency Address
    // Constructor Injection
    public Restaurant(Address address) {
        System.out.println("[Restaurant] - Object Constructed - Paramerterized Construc
        System.out.println("[Restaurant] - Constructor Injection");
        this.address = address;
    }
```

```
<terminated> App (2) [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.linux.x86_64_17.0.1.v20211116-1657/jre/bin/java
Welcome to Dependency Injection
[Address] - Object Created
[Restaurant] - Object Constructed - Paramerterized Constructor with Address as Input
[Restaurant] - Constructor Injection
Restaurant [name=Johns Cafe, phone=+91 99999 11111, operatingHours=10:00 to 22:00, ratings=4.5, address=Address [adrsLine=2
Address [adrsLine=2121 Street A, city=ABC, state=XYZ State, zipCode=110011]
```

You will observe that the Address object is created separately, and the Restaurant object is constructed using the parameterized constructor instead of the default constructor. The overall structure of the application remains the same. The choice between using setter or constructor injection depends on your specific requirements.

Step 7: Understanding one-to-many relationships

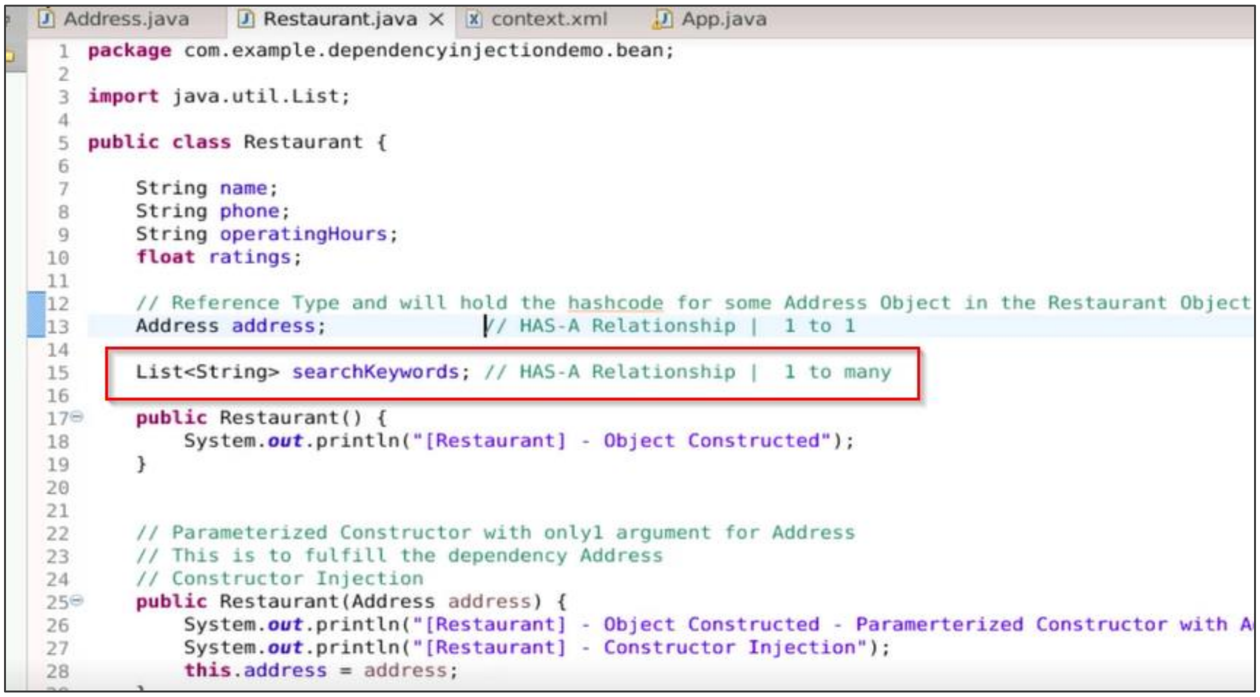
7.1 Open the **Restaurant.java** class



Address.java | Restaurant.java | context.xml | App.java

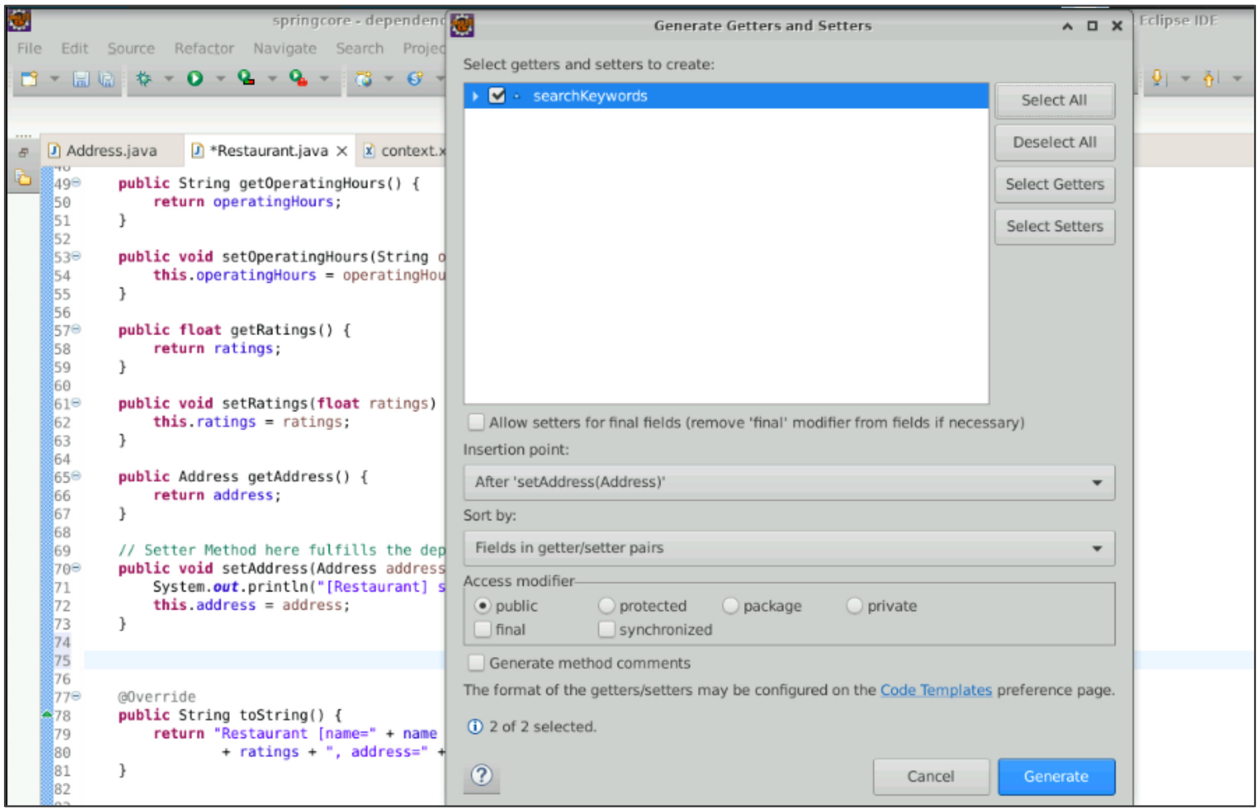
```java
package com.example.dependencyinjectiondemo.bean;

import java.util.List;

public class Restaurant {

    String name;
    String phone;
    String operatingHours;
    float ratings;

    // Reference Type and will hold the hashcode for some Address Object in the Restaurant Object
    Address address;              // HAS-A Relationship |  1 to 1

    List<String> searchKeywords; // HAS-A Relationship |  1 to many

    public Restaurant() {
        System.out.println("[Restaurant] - Object Constructed");
    }


    // Parameterized Constructor with only1 argument for Address
    // This is to fulfill the dependency Address
    // Constructor Injection
    public Restaurant(Address address) {
        System.out.println("[Restaurant] - Object Constructed - Paramerterized Constructor with A
        System.out.println("[Restaurant] - Constructor Injection");
        this.address = address;
```

7.2 Create a variable named **searchKeywords** of type **List<String>** to represent the search keywords



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

import java.util.List;

public class Restaurant {

    String name;
    String phone;
    String operatingHours;
    float ratings;

    // Reference Type and will hold the hashcode for some Address Object in the Restaurant Object
    Address address;                // HAS-A Relationship |  1 to 1

    List<String> searchKeywords; // HAS-A Relationship |  1 to many

    public Restaurant() {
        System.out.println("[Restaurant] - Object Constructed");
    }


    // Parameterized Constructor with only1 argument for Address
    // This is to fulfill the dependency Address
    // Constructor Injection
    public Restaurant(Address address) {
        System.out.println("[Restaurant] - Object Constructed - Paramerterized Constructor with A
        System.out.println("[Restaurant] - Constructor Injection");
        this.address = address;
```

7.3 Generate getters and setters for the **searchKeywords** variable



```
    public String getOperatingHours() {
        return operatingHours;
    }

    public void setOperatingHours(String o
        this.operatingHours = operatingHou
    }

    public float getRatings() {
        return ratings;
    }

    public void setRatings(float ratings)
        this.ratings = ratings;
    }

    public Address getAddress() {
        return address;
    }

    // Setter Method here fulfills the dep
    public void setAddress(Address address
        System.out.println("[Restaurant] s
        this.address = address;
    }



    @Override
    public String toString() {
        return "Restaurant [name=" + name
            + ratings + ", address=" +
    }
```

Select getters and setters to create:

- [✓] searchKeywords

[ ] Allow setters for final fields (remove 'final' modifier from fields if necessary)

Insertion point: After 'setAddress(Address)'

Sort by: Fields in getter/setter pairs

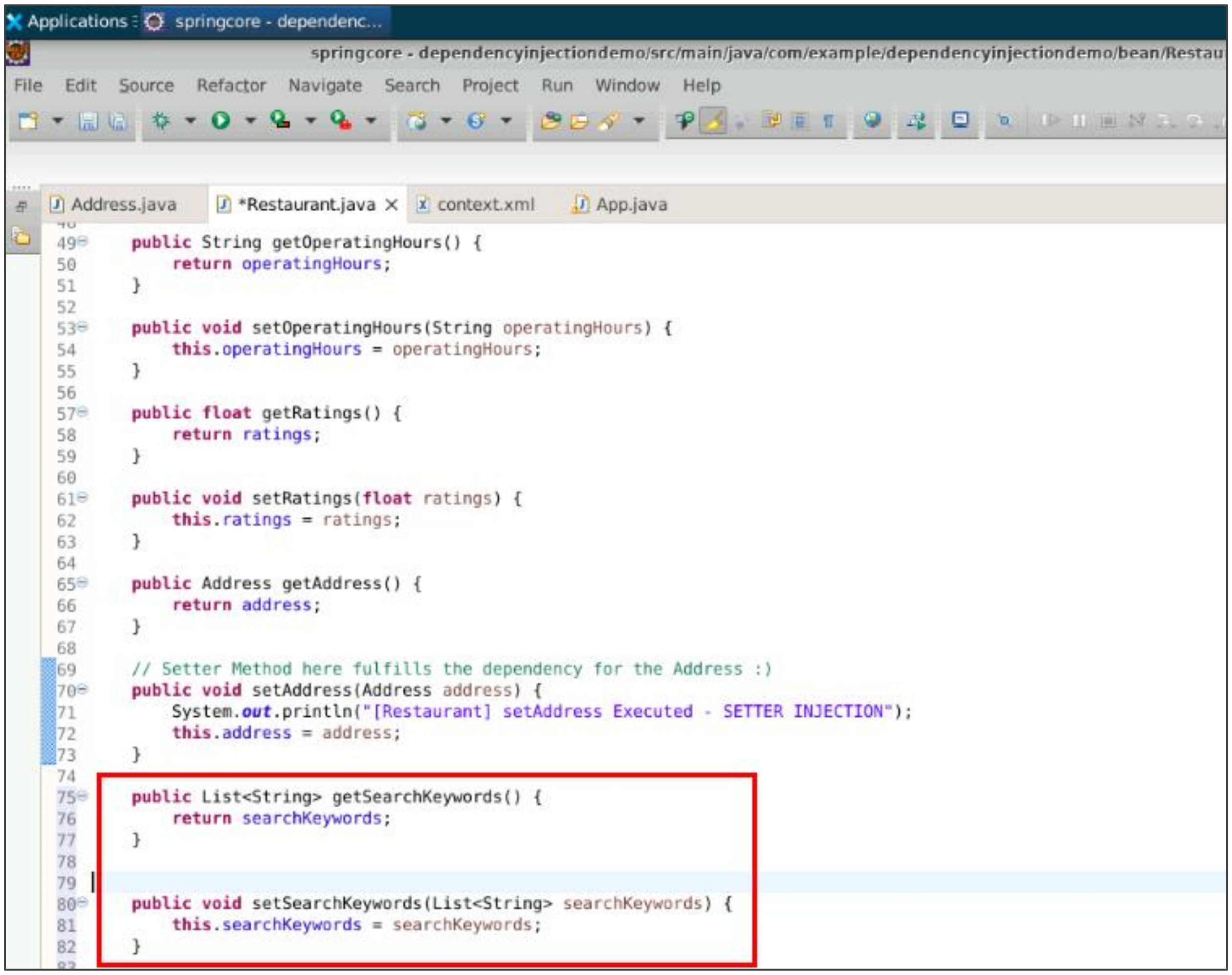
Access modifier: [✓ public] protected package private

[ ] final [ ] synchronized

[ ] Generate method comments

The format of the getters/setters may be configured on the Code Templates preference page.

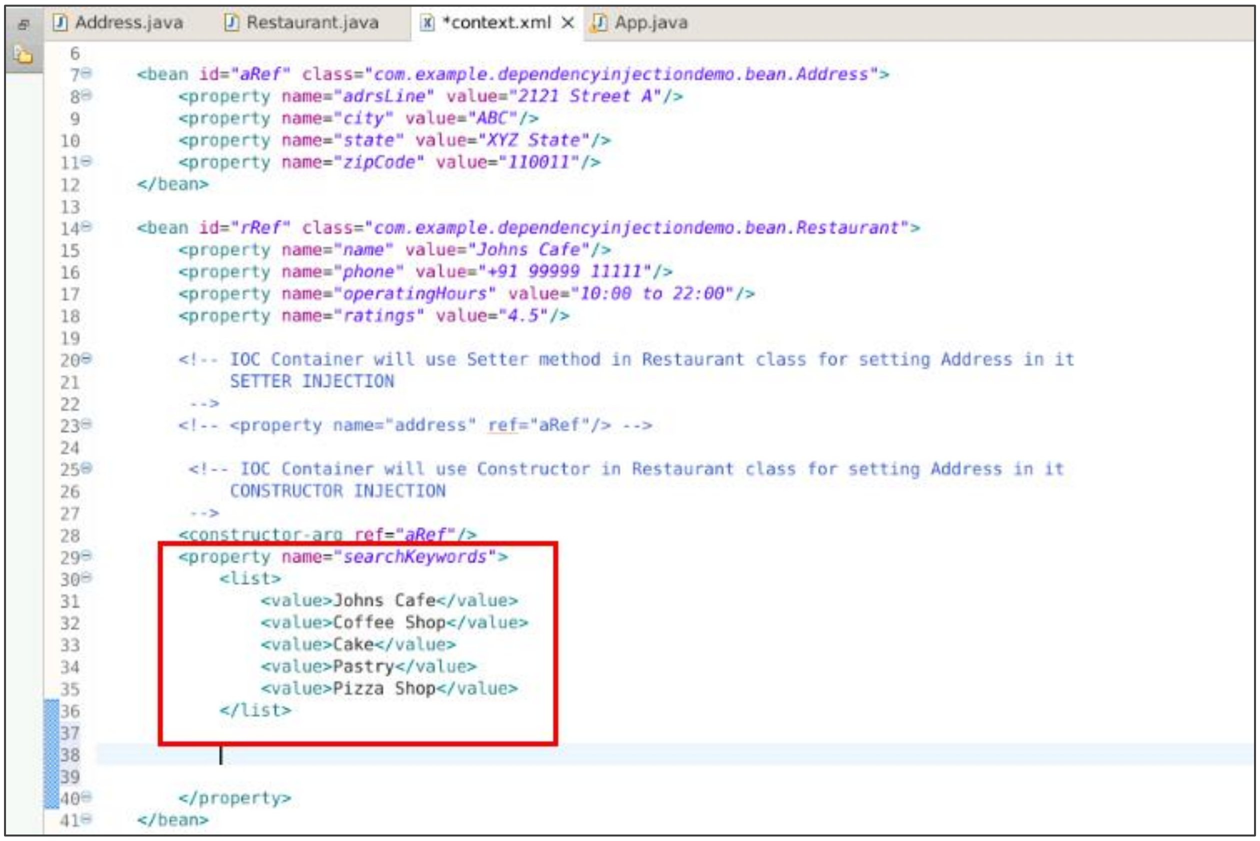
2 of 2 selected.



Applications
springcore - dependenc...
springcore - dependencyinjectiondemo/src/main/java/com/example/dependencyinjectiondemo/bean/Restau
File Edit Source Refactor Navigate Search Project Run Window Help
Address.java
*Restaurant.java
context.xml
App.java

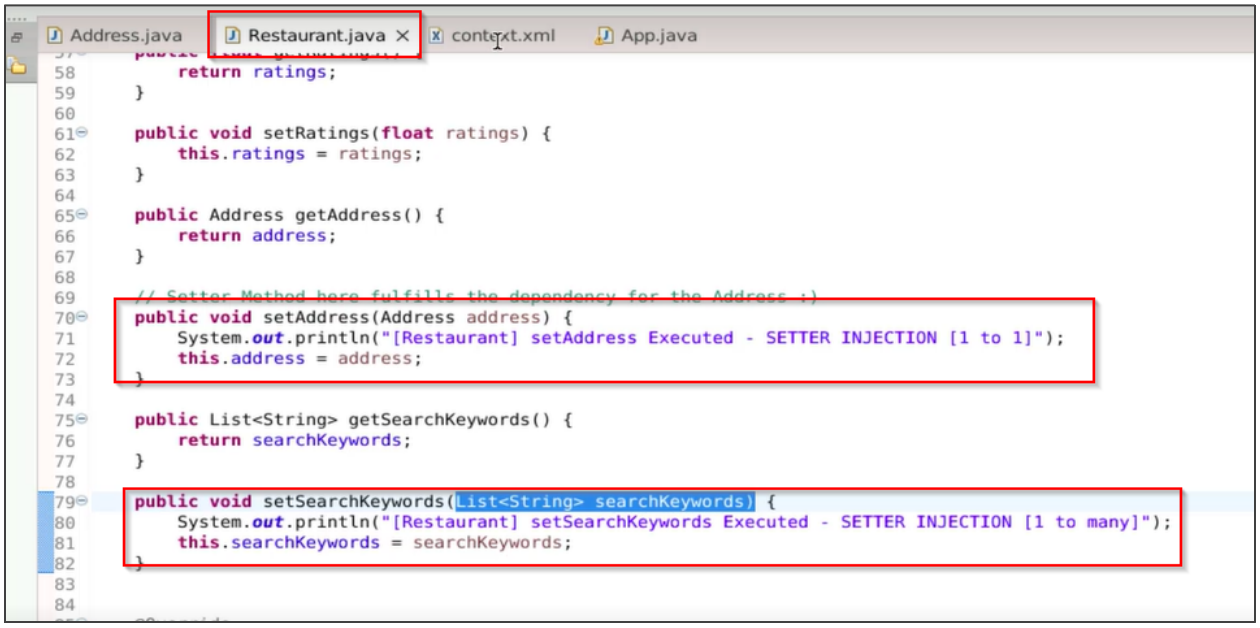
```java
    public String getOperatingHours() {
        return operatingHours;
    }

    public void setOperatingHours(String operatingHours) {
        this.operatingHours = operatingHours;
    }

    public float getRatings() {
        return ratings;
    }

    public void setRatings(float ratings) {
        this.ratings = ratings;
    }

    public Address getAddress() {
        return address;
    }

    // Setter Method here fulfills the dependency for the Address :)
    public void setAddress(Address address) {
        System.out.println("[Restaurant] setAddress Executed - SETTER INJECTION");
        this.address = address;
    }

    public List<String> getSearchKeywords() {
        return searchKeywords;
    }


    public void setSearchKeywords(List<String> searchKeywords) {
        this.searchKeywords = searchKeywords;
    }
```

7.4 Update the **context.xml** file to add a **property** tag for **searchKeywords** and set multiple values under the **list** tag



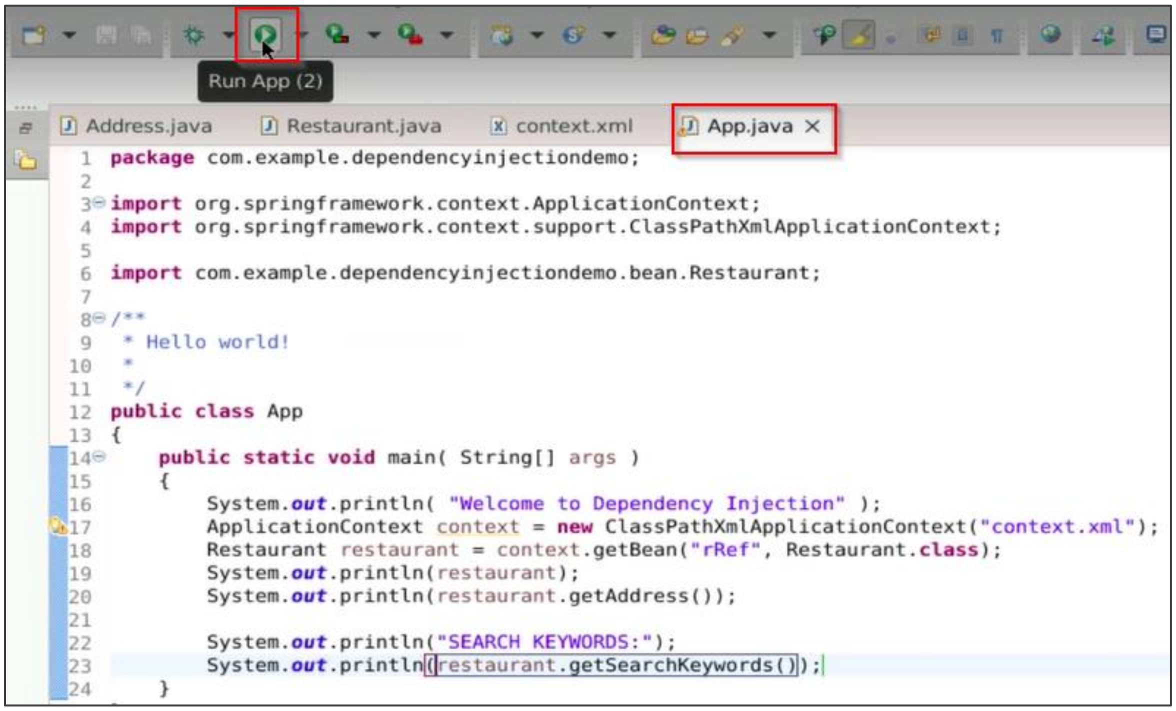
```xml

    <bean id="aRef" class="com.example.dependencyinjectiondemo.bean.Address">
        <property name="adrsLine" value="2121 Street A"/>
        <property name="city" value="ABC"/>
        <property name="state" value="XYZ State"/>
        <property name="zipCode" value="110011"/>
    </bean>

    <bean id="rRef" class="com.example.dependencyinjectiondemo.bean.Restaurant">
        <property name="name" value="Johns Cafe"/>
        <property name="phone" value="+91 99999 11111"/>
        <property name="operatingHours" value="10:00 to 22:00"/>
        <property name="ratings" value="4.5"/>

        <!-- IOC Container will use Setter method in Restaurant class for setting Address in it
             SETTER INJECTION
         -->
        <!-- <property name="address" ref="aRef"/> -->

         <!-- IOC Container will use Constructor in Restaurant class for setting Address in it
             CONSTRUCTOR INJECTION
         -->
        <constructor-arg ref="aRef"/>
        <property name="searchKeywords">
            <list>
                <value>Johns Cafe</value>
                <value>Coffee Shop</value>
                <value>Cake</value>
                <value>Pastry</value>
                <value>Pizza Shop</value>
            </list>



        </property>
    </bean>
```

7.5 Add print statements in the **setAddress** and **setSearchKeywords** methods of the Restaurant class, indicating **1 to 1** and **1 to many** relationships, respectively.



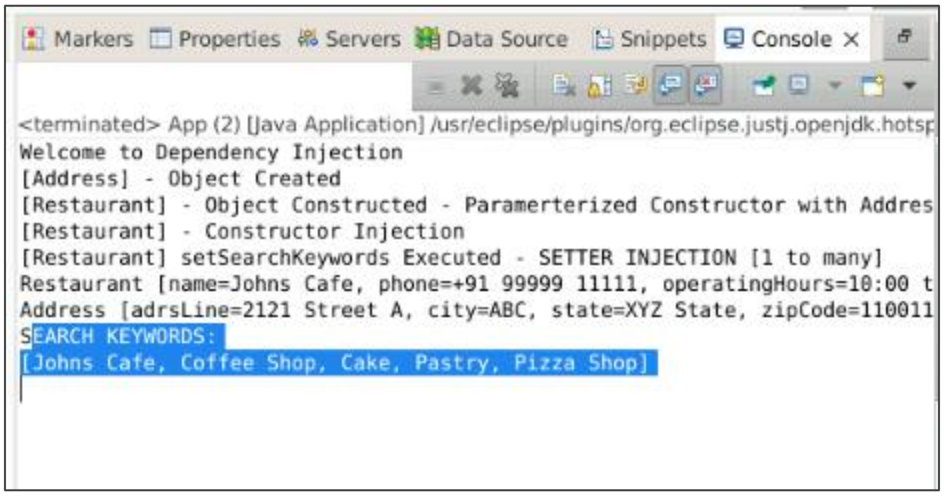
```java
        return ratings;
    }

    public void setRatings(float ratings) {
        this.ratings = ratings;
    }

    public Address getAddress() {
        return address;
    }

    // Setter Method here fulfills the dependency for the Address :)
    public void setAddress(Address address) {
        System.out.println("[Restaurant] setAddress Executed - SETTER INJECTION [1 to 1]");
        this.address = address;
    }

    public List<String> getSearchKeywords() {
        return searchKeywords;
    }

    public void setSearchKeywords(List<String> searchKeywords) {
        System.out.println("[Restaurant] setSearchKeywords Executed - SETTER INJECTION [1 to many]");
        this.searchKeywords = searchKeywords;
    }
```

7.6 Update the **App.java** class to display and list the search keywords on the console for each Restaurant



```java
package com.example.dependencyinjectiondemo;

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

import com.example.dependencyinjectiondemo.bean.Restaurant;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
        System.out.println( "Welcome to Dependency Injection" );
        ApplicationContext context = new ClassPathXmlApplicationContext("context.xml");
        Restaurant restaurant = context.getBean("rRef", Restaurant.class);
        System.out.println(restaurant);
        System.out.println(restaurant.getAddress());

        System.out.println("SEARCH KEYWORDS:");
        System.out.println(restaurant.getSearchKeywords());
    }
```

7.7 Run the project



```
<terminated> App (2) [Java Application] /usr/eclipse/plugins/org.eclipse.justj.openjdk.hotsp
Welcome to Dependency Injection
[Address] - Object Created
[Restaurant] - Object Constructed - Paramerterized Constructor with Addres
[Restaurant] - Constructor Injection
[Restaurant] setSearchKeywords Executed - SETTER INJECTION [1 to many]
Restaurant [name=Johns Cafe, phone=+91 99999 11111, operatingHours=10:00 t
Address [adrsLine=2121 Street A, city=ABC, state=XYZ State, zipCode=110011
SEARCH KEYWORDS:
[Johns Cafe, Coffee Shop, Cake, Pastry, Pizza Shop]
```

You will observe that the search keywords are listed on the console, showcasing a one-to-many relationship in the Restaurants bean.