**Objectives**

* Demonstrate creation of Spring Boot Application
  + Spring initializr, https://start.spring.io, @SpringBootApplication, SpringApplication.run()
    - Ref - https://start.spring.io
* Explain the need and benefits of Spring Boot
  + Makes Java development easy, avoids tedious development steps, reduces development time, avoids writing boilerplate code, provides embedded tomcat server, avoid XML configuration
    - Ref - https://www.journaldev.com/7969/spring-boot-tutorial
* Demonstrate loading bean from spring configuration file
  + Spring configuration xml, spring xml schema spring-beans.xsd, <bean>, id, class, <constructor-arg>, <property>, name, value, ClassPathXmlApplicationContext, ApplicationContext, context.getBean(), singleton scope, prototype scope
    - Ref - https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html
    - IoC Container - https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html#beans
    - Scopes - https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html#beans-factory-scopes
    - Constructor Injection - https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html#beans-constructor-injection
    - Setter method injection - https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html#beans-setter-injection
* Demonstrate inclusion of logging in Spring Boot Application
  + application.properties, logging.level, logging.pattern, server.port, LoggerFactory, Logger, log levels (trace, debug, info, warn, error)
    - Ref - https://docs.spring.io/spring-boot/docs/current/reference/html/boot-features-logging.html

**Hands on 1**

**Create a Spring Web Project using Maven**   
  
Follow steps below to create a project: 

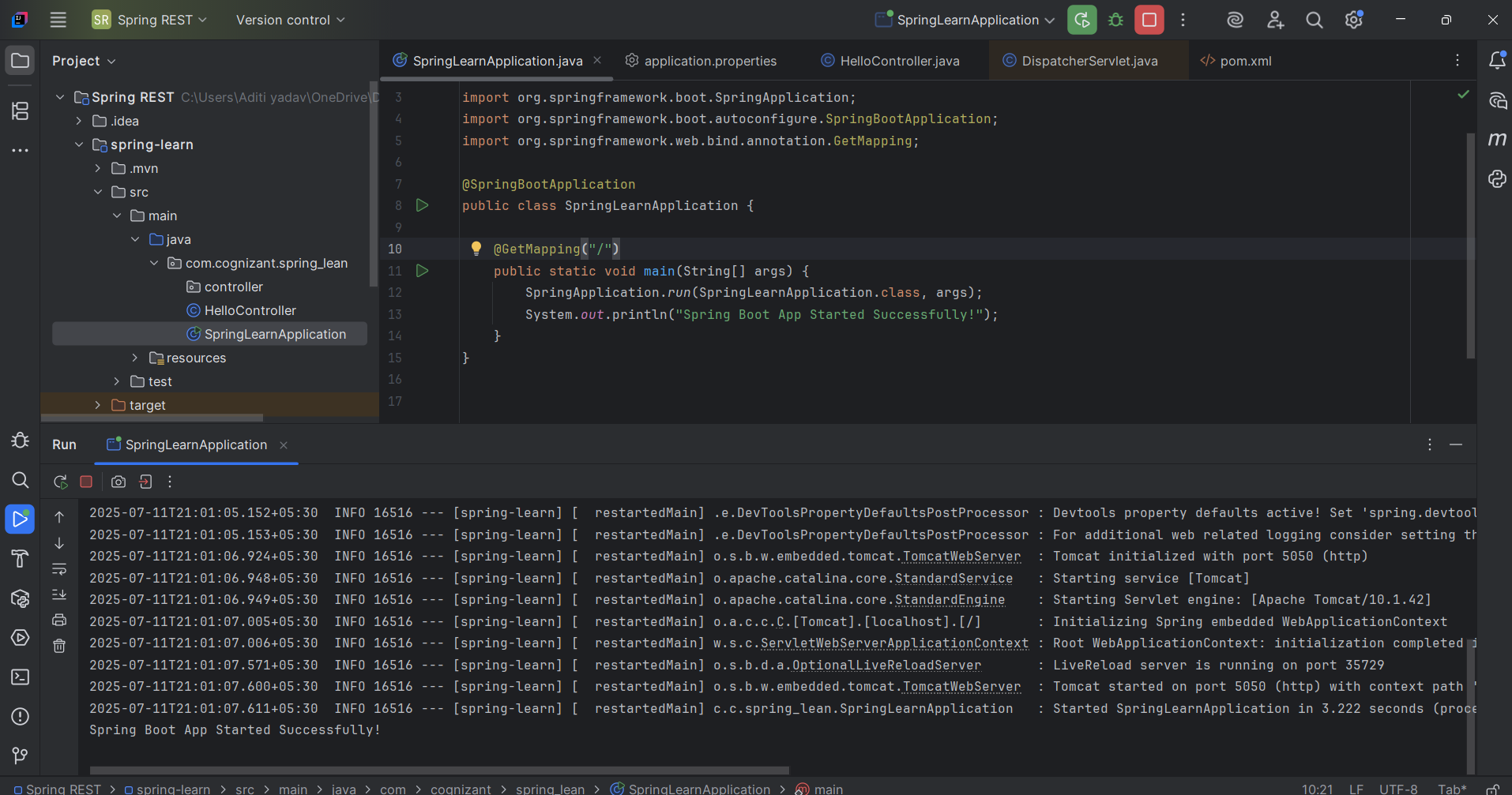
1. Go to <https://start.spring.io/>
2. Change Group as “com.cognizant”
3. Change Artifact Id as “spring-learn”
4. Select Spring Boot DevTools and Spring Web
5. Create and download the project as zip
6. Extract the zip in root folder to Eclipse Workspace
7. Build the project using ‘mvn clean package -Dhttp.proxyHost=proxy.cognizant.com -Dhttp.proxyPort=6050 -Dhttps.proxyHost=proxy.cognizant.com -Dhttps.proxyPort=6050 -Dhttp.proxyUser=123456’ command in command line
8. Import the project in Eclipse "File > Import > Maven > Existing Maven Projects > Click Browse and select extracted folder > Finish"
9. Include logs to verify if main() method of SpringLearnApplication.
10. Run the SpringLearnApplication class.

SME to walk through the following aspects related to the project created:

1. src/main/java - Folder with application code
2. src/main/resources - Folder for application configuration
3. src/test/java - Folder with code for testing the application
4. SpringLearnApplication.java - Walkthrough the main() method.
5. Purpose of @SpringBootApplication annotation
6. pom.xml
   1. Walkthrough all the configuration defined in XML file
   2. Open 'Dependency Hierarchy' and show the dependency tree.

Code:

package com.cognizant.spring\_learn;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
  
@SpringBootApplication  
@RestController  
public class SpringLearnApplication {  
  
 public static void main(String[] args) {  
 SpringApplication.*run*(SpringLearnApplication.class, args);  
 System.*out*.println("Spring Boot App Started Successfully!");  
 }  
  
 @GetMapping("/") // ✅ Correct: now it maps a GET request to "/"  
 public String home() {  
 return "Spring Boot App Started Successfully!";  
 }  
}



**Hands on 2**

**Spring Core – Load SimpleDateFormat from Spring Configuration XML**   
  
SimpleDateFormat with the pattern ‘dd/MM/yyyy’ is created in multiple places of an application. To avoid creation of SimpleDateFormat in multiple places, define a bean in Spring XML Configuration file and retrieve the date.  
  
Follow steps below to implement:

* Create spring configuration file date-format.xml in src/main/resources folder of 'spring-learn' project
* Open https://docs.spring.io/spring-framework/docs/current/spring-framework-reference/core.html#beans-factory-metadata
* Copy the XML defined in the section of previous step URL and paste it into date-format.xml
* Define bean tag in the XML with for date format. Refer code below.

<?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="dateFormat" class="java.text.SimpleDateFormat">

<constructor-arg value="dd/MM/yyyy" />

</bean>

</beans>

* Create new method displayDate() in SpringLearnApplication.java
* In displayDate() method create the ApplicationContext. Refer code below:

ApplicationContext context = new ClassPathXmlApplicationContext("date-format.xml");

* Get the dateFormat using getBean() method. Refer code below.

SimpleDateFormat format = context.getBean("dateFormat", SimpleDateFormat.class);

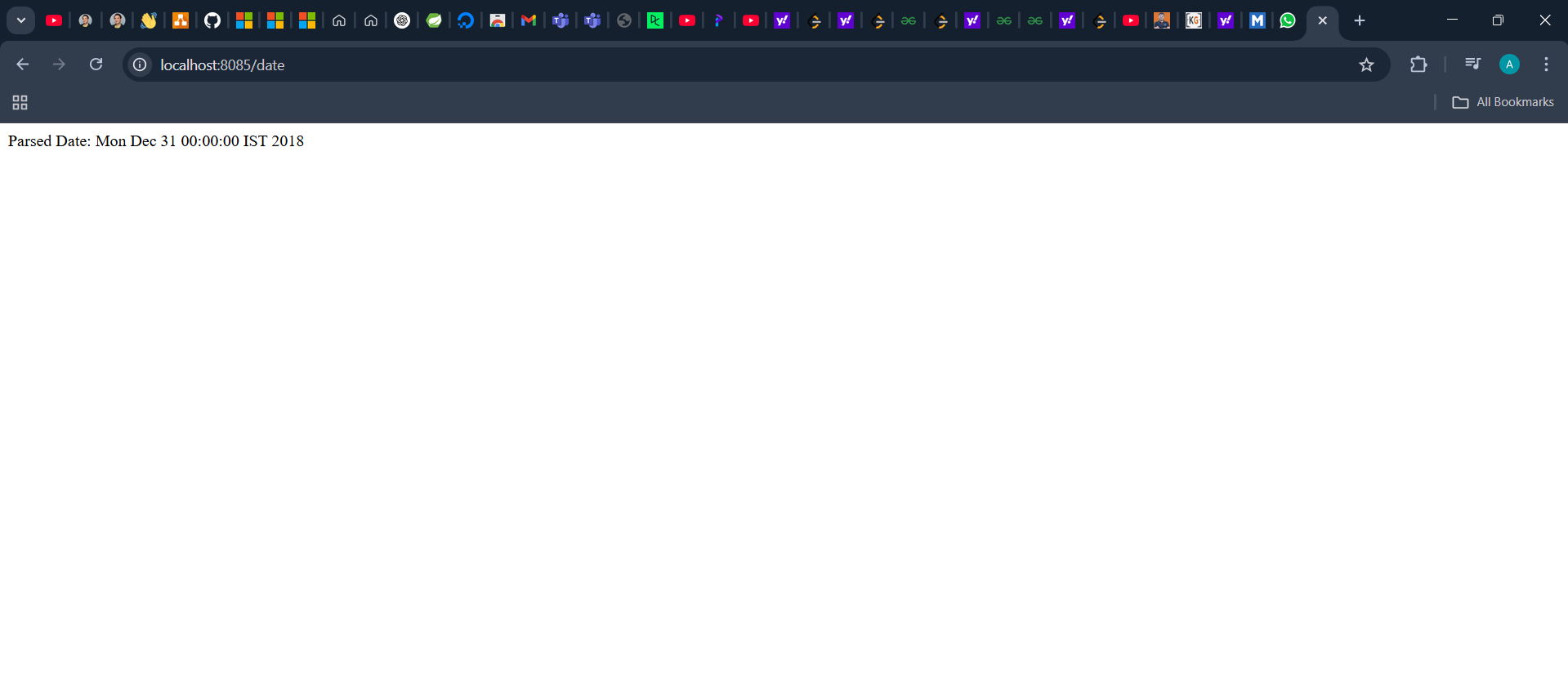
* Using the format variable try to parse string '31/12/2018' to Date class and display the result using System.out.println.
* Run the application as 'Java Application' and check the result in console log output.

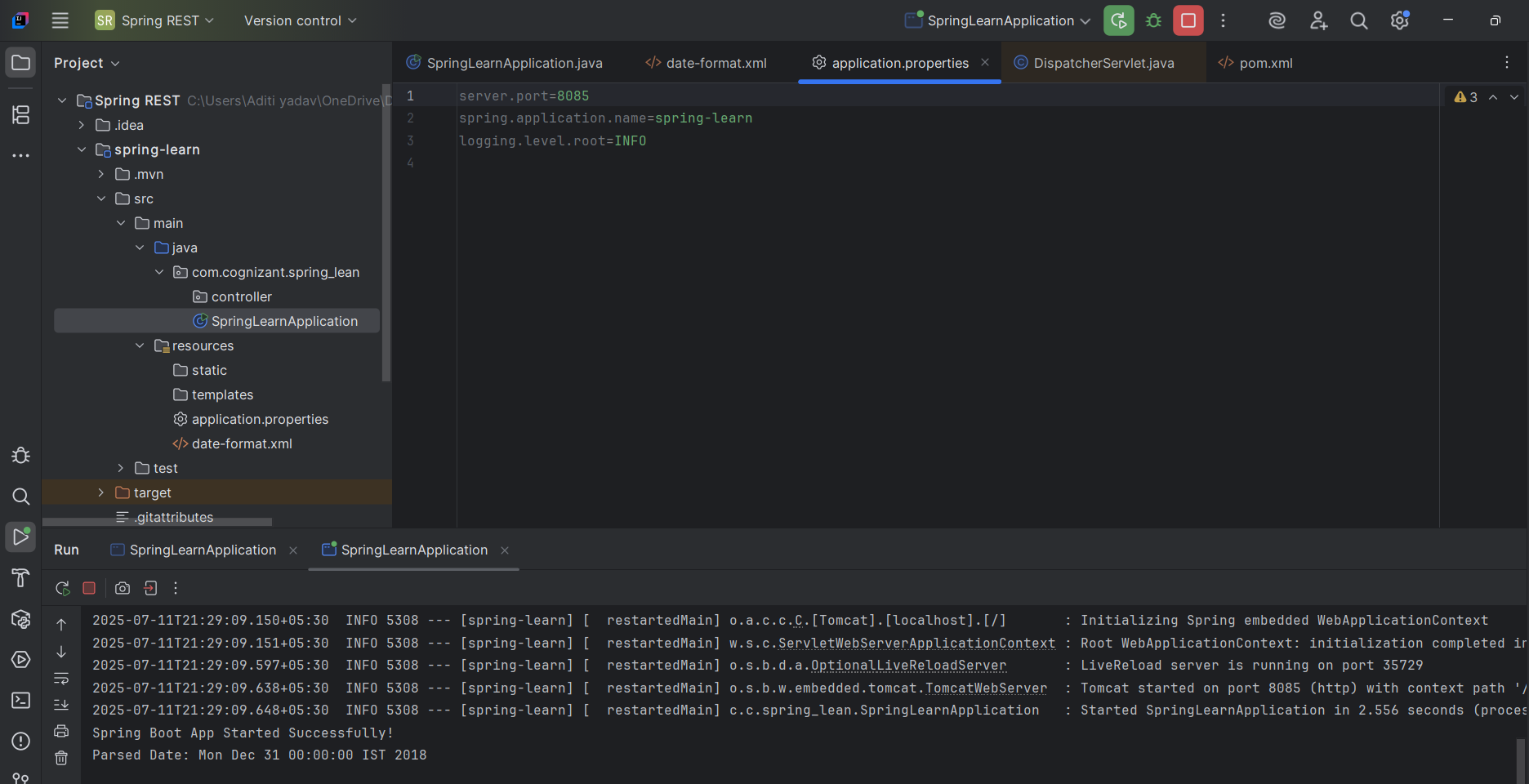
**Troubleshooting Tips**   
  
If the tomcat port has a conflict and the server is not starting include the below property in application.properties file in src/main/resources folder.

Code:

import java.text.SimpleDateFormat;  
import java.util.Date;  
  
@RestController  
public class DateController {  
  
 @GetMapping("/date")  
 public String getFormattedDate() throws Exception {  
 ApplicationContext context = new ClassPathXmlApplicationContext("date-format.xml");  
 SimpleDateFormat format = context.getBean("dateFormat", SimpleDateFormat.class);  
 Date date = format.parse("31/12/2018");  
 return "Parsed Date: " + date.toString();  
 }  
}

<?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="dateFormat" class="java.text.SimpleDateFormat">  
 <constructor-arg value="dd/MM/yyyy"/>  
 </bean>  
  
</beans>

package com.cognizant.spring\_lean;  
  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
@SpringBootApplication  
public class SpringLearnApplication {  
  
 public static void main(String[] args) throws Exception {  
 SpringApplication.*run*(SpringLearnApplication.class, args);  
 System.*out*.println("Spring Boot App Started Successfully!");  
  
 *displayDate*();   
 }  
   
   
 public static void displayDate() throws Exception {  
 ApplicationContext context = new ClassPathXmlApplicationContext("date-format.xml");  
  
 SimpleDateFormat format = context.getBean("dateFormat", SimpleDateFormat.class);  
  
 Date date = format.parse("31/12/2018");  
  
 System.*out*.println("Parsed Date: " + date);  
 }  
}

****

**Hands on 4**

**Spring Core – Load Country from Spring Configuration XML**   
  
An airlines website is going to support booking on four countries. There will be a drop down on the home page of this website to select the respective country. It is also important to store the two-character ISO code of each country. 

|  |  |
| --- | --- |
| **Code** | **Name** |
| US | United States |
| DE | Germany |
| IN | India |
| JP | Japan |

Above data has to be stored in spring configuration file. Write a program to read this configuration file and display the details.  
  
Steps to implement

* Pick any one of your choice country to configure in Spring XML configuration named country.xml.
* Create a bean tag in spring configuration for country and set the property and values

    <bean id="country" class="com.cognizant.springlearn.Country">

        <property name="code" value="IN" />

        <property name="name" value="India" />

    </bean>

* Create Country class with following aspects:
  + Instance variables for code and name
  + Implement empty parameter constructor with inclusion of debug log within the constructor with log message as “Inside Country Constructor.”
  + Generate getters and setters with inclusion of debug with relevant message within each setter and getter method.
  + Generate toString() method
* Create a method displayCountry() in SpringLearnApplication.java, which will read the country bean from spring configuration file and display the country details. ClassPathXmlApplicationContext, ApplicationContext and context.getBean(“beanId”, Country.class). Refer sample code for displayCountry() method below.

ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");

Country country = (Country) context.getBean("country", Country.class);

LOGGER.debug("Country : {}", country.toString());

* Invoke displayCountry() method in main() method of SpringLearnApplication.java.
* Execute main() method and check the logs to find out which constructors and methods were invoked.

SME to provide more detailing about the following aspects:

* bean tag, id attribute, class attribute, property tag, name attribute, value attribute
* ApplicationContext, ClassPathXmlApplicationContext
* What exactly happens when context.getBean() is invoked

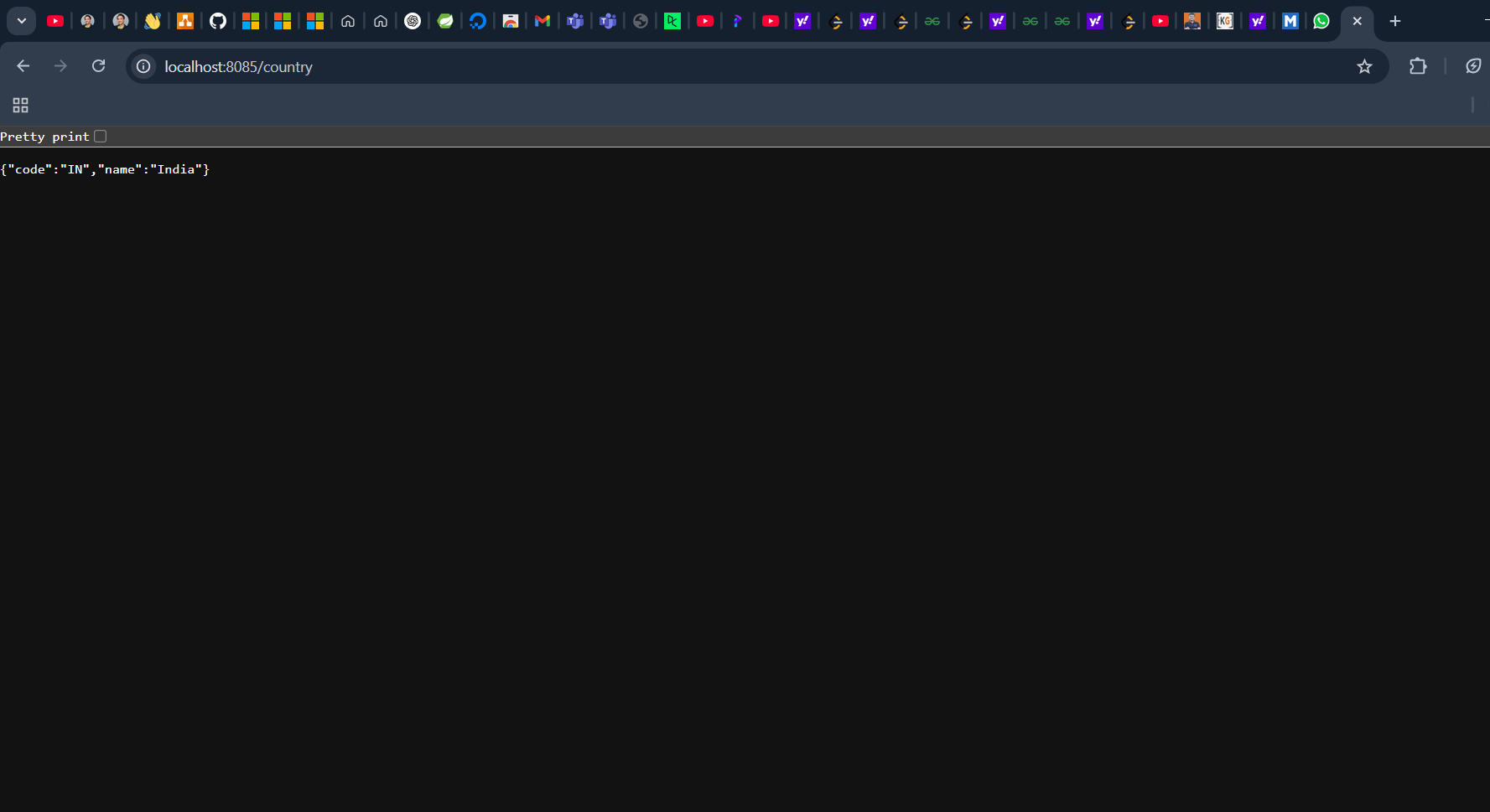
Code:

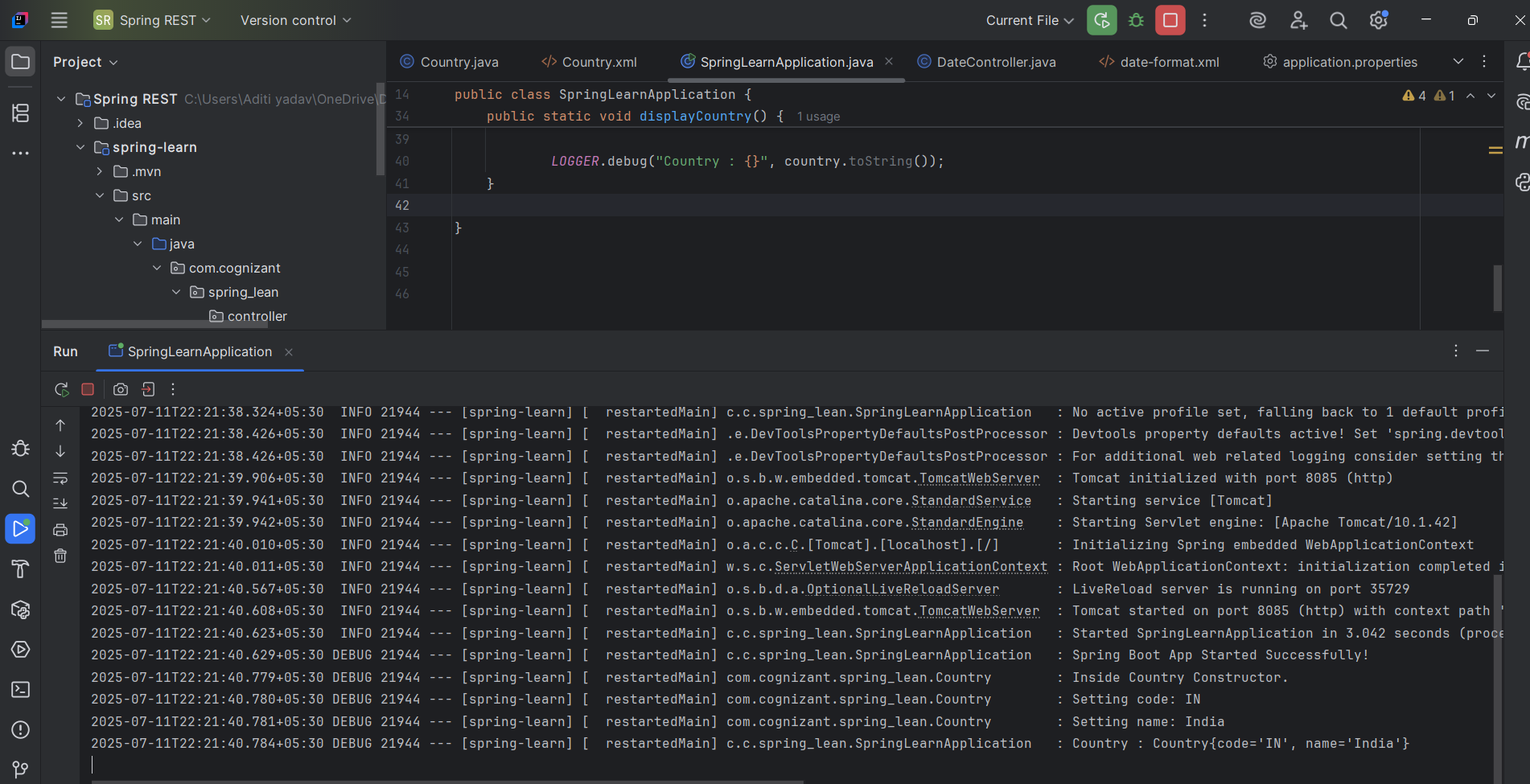
package com.cognizant.spring\_lean;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
import org.springframework.boot.SpringApplication;  
import org.springframework.boot.autoconfigure.SpringBootApplication;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
import java.text.SimpleDateFormat;  
import java.util.Date;  
  
@SpringBootApplication  
public class SpringLearnApplication {  
  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(SpringLearnApplication.class);  
  
 public static void main(String[] args) throws Exception {  
 SpringApplication.*run*(SpringLearnApplication.class, args);  
 *LOGGER*.debug("Spring Boot App Started Successfully!");  
  
// displayDate();  
 *displayCountry*();  
 }  
  
// public static void displayDate() throws Exception {  
// ApplicationContext context = new ClassPathXmlApplicationContext("date-format.xml");  
// SimpleDateFormat format = context.getBean("dateFormat", SimpleDateFormat.class);  
// Date date = format.parse("31/12/2018");  
// System.out.println("Parsed Date: " + date);  
// }  
  
  
 public static void displayCountry() {  
 ApplicationContext context = new ClassPathXmlApplicationContext("Country.xml");  
  
 // Correct usage of your Country class  
 Country country = context.getBean("country", Country.class);  
  
 *LOGGER*.debug("Country : {}", country.toString());  
 }  
  
}

package com.cognizant.spring\_lean.controller;  
  
import com.cognizant.spring\_lean.Country;  
import org.springframework.web.bind.annotation.GetMapping;  
import org.springframework.web.bind.annotation.RestController;  
import org.springframework.context.ApplicationContext;  
import org.springframework.context.support.ClassPathXmlApplicationContext;  
  
@RestController  
public class CountryController {  
  
 @GetMapping("/country")  
 public Country getCountry() {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
 return context.getBean("country", Country.class);  
 }  
}

package com.cognizant.spring\_lean;  
  
import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;  
  
public class Country {  
 private static final Logger *LOGGER* = LoggerFactory.*getLogger*(Country.class);  
  
 private String code;  
 private String name;  
  
 public Country() {  
 *LOGGER*.debug("Inside Country Constructor.");  
 }  
  
 public String getCode() {  
 *LOGGER*.debug("Getting code: {}", code);  
 return code;  
 }  
  
 public void setCode(String code) {  
 *LOGGER*.debug("Setting code: {}", code);  
 this.code = code;  
 }  
  
 public String getName() {  
 *LOGGER*.debug("Getting name: {}", name);  
 return name;  
 }  
  
 public void setName(String name) {  
 *LOGGER*.debug("Setting name: {}", name);  
 this.name = name;  
 }  
  
 @Override  
 public String toString() {  
 return "Country{" +  
 "code='" + code + '\'' +  
 ", name='" + name + '\'' +  
 '}';  
 }  
}

<?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="country" class="com.cognizant.spring\_lean.Country">  
 <property name="code" value="IN"/>  
 <property name="name" value="India"/>  
 </bean>  
  
  
</beans>





**Hands on 5**

**Spring Core – Demonstration of Singleton Scope and Prototype Scope**   
  
The Country bean done in the previous hands on will be used to demonstrate the scopes in Spring. Implement the steps below.  
  
**Follow steps below to demonstrate Singleton Scope**

* Include a line in displayCountry() to get country bean reference one more time from the same application context. Only the third line of the below code snippet should be copied and pasted.

ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");

Country country = context.getBean("country", Country.class);

Country anotherCountry = context.getBean("country", Country.class);

* The constructor will be called only once, which means that only one instance of Country bean is created

**Follow steps below to demonstrate Prototype Scope**

* Include scope="prototype" attribute in bean definition xml.

<bean id="country" class="com.cognizant.springlearn.Country" scope="prototype">

* Run the application
* Constructor will be called twice, which means that two instances of country is created.

Code:

public static void displayCountry() {  
 ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");  
  
 Country country = context.getBean("country", Country.class);  
 Country anotherCountry = context.getBean("country", Country.class);  
  
 *LOGGER*.debug("Country: {}", country);  
 *LOGGER*.debug("Another Country: {}", anotherCountry);  
}

<?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="country" class="com.cognizant.spring\_lean.Country" scope="prototype">  
 <property name="code" value="IN"/>  
 <property name="name" value="India"/>  
 </bean>  
  
  
</beans>

s