**Hands on 4**

**Spring Core – Load Country from Spring Configuration XML**   
  
**pom.xml**

<?xml version="1.0" encoding="UTF-8"?>

<project xmlns="http://maven.apache.org/POM/4.0.0"

xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">

<modelVersion>4.0.0</modelVersion>

<parent>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-parent</artifactId>

<version>3.5.3</version>

<relativePath/> <!-- lookup parent from repository -->

</parent>

<groupId>com.cognizant</groupId>

<artifactId>spring-learn</artifactId>

<version>0.0.1-SNAPSHOT</version>

<packaging>jar</packaging>

<name>spring-learn</name>

<description>Demo project for Spring Boot</description>

<properties>

<java.version>17</java.version>

</properties>

<dependencies>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-web</artifactId>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-devtools</artifactId>

<scope>runtime</scope>

<optional>true</optional>

</dependency>

<dependency>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-starter-test</artifactId>

<scope>test</scope>

</dependency>

</dependencies>

<build>

<plugins>

<plugin>

<groupId>org.springframework.boot</groupId>

<artifactId>spring-boot-maven-plugin</artifactId>

</plugin>

</plugins>

</build>

</project>

**Country.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=*"country"* class=*"com.cognizant.spring\_learn.Country"*>

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

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

</**bean**>

</**beans**>

**Country.java**

package com.cognizant.spring\_learn;

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 Country code");

return code;

}

public void setCode(String code) {

***LOGGER***.debug("Setting Country code");

this.code = code;

}

public String getName() {

***LOGGER***.debug("Getting Country name");

return name;

}

public void setName(String name) {

***LOGGER***.debug("Setting Country name");

this.name = name;

}

*@Override*

public String toString() {

return "Country [code=" + code + ", name=" + name + "]";

}

}

**SpringLearnApplication.java**

package com.cognizant.spring\_learn;

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;

*@SpringBootApplication*

public class SpringLearnApplication {

private static final Logger ***LOGGER*** = LoggerFactory.*getLogger*(SpringLearnApplication.class);

public static void main(String[] args) {

SpringApplication.*run*(SpringLearnApplication.class, args);

***LOGGER***.info("START");

*displayCountry*();

***LOGGER***.info("END");

}

public static void displayCountry() {

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

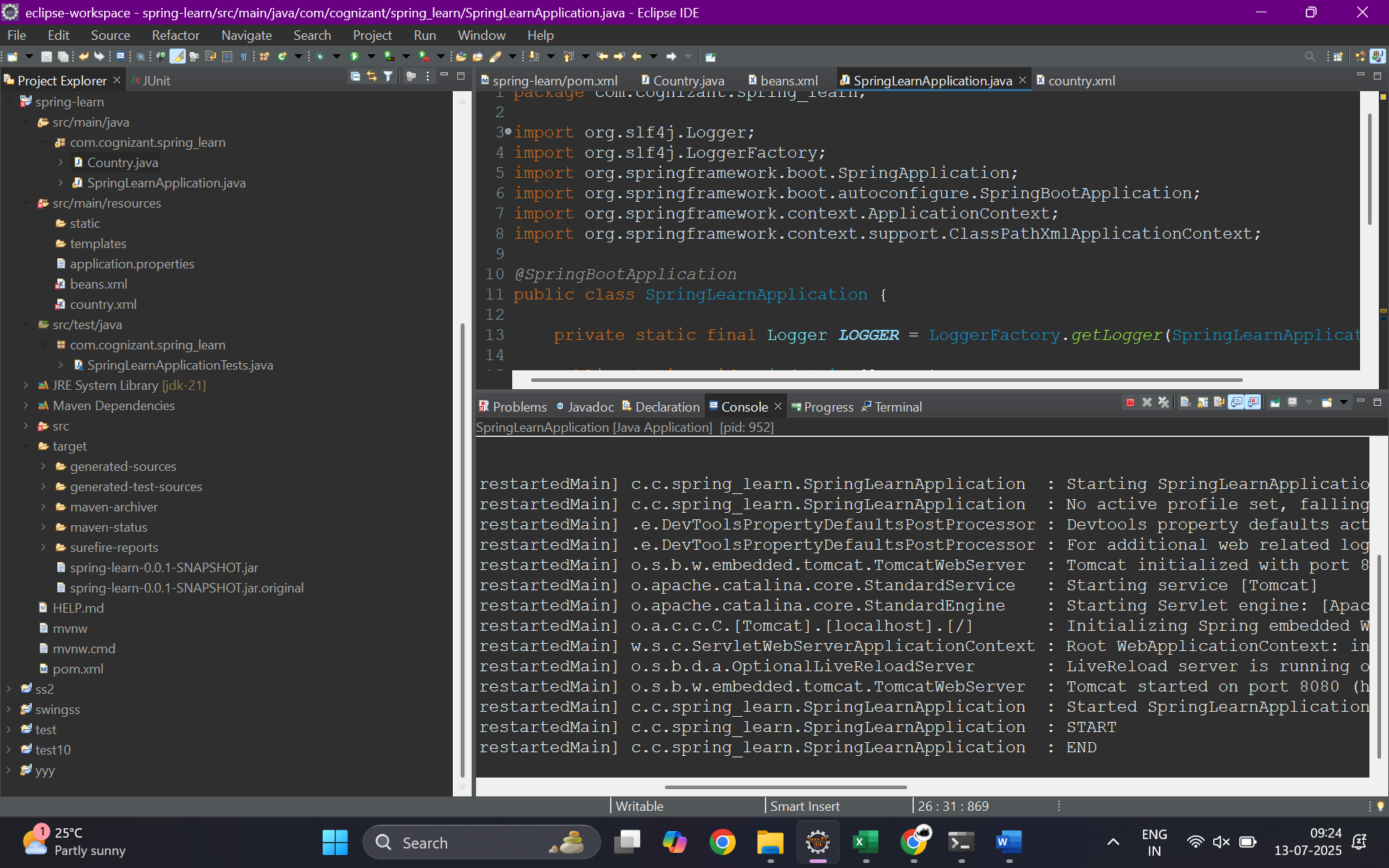
Country country = (Country) context.getBean("country", Country.class);

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

}

}

**Output:**



**SME Notes**

In this hands-on, we used the Spring XML configuration to define a bean representing a country. The <bean> tag in Spring’s XML file is used to declare a Spring-managed object. The id attribute specifies the unique name of the bean, which is used to retrieve it from the application context. The class attribute provides the fully qualified class name of the bean. Inside the bean tag, <property> tags are used to set the values for the bean’s properties. The name attribute of the property corresponds to the setter method name in the Java class (e.g., setCode for name="code"), and the value attribute is the actual value being assigned (e.g., "IN" for the code of the country).

Spring’s ApplicationContext is a central interface for providing configuration information to the application. It loads bean definitions and manages their lifecycle. The specific implementation we used, ClassPathXmlApplicationContext, reads the XML configuration file from the classpath.

When the context.getBean("country", Country.class) method is called, Spring returns the bean instance named "country" by locating it in the configuration, instantiating the class, injecting the specified property values, and returning the fully initialized object. This process ensures loose coupling and centralized configuration management.