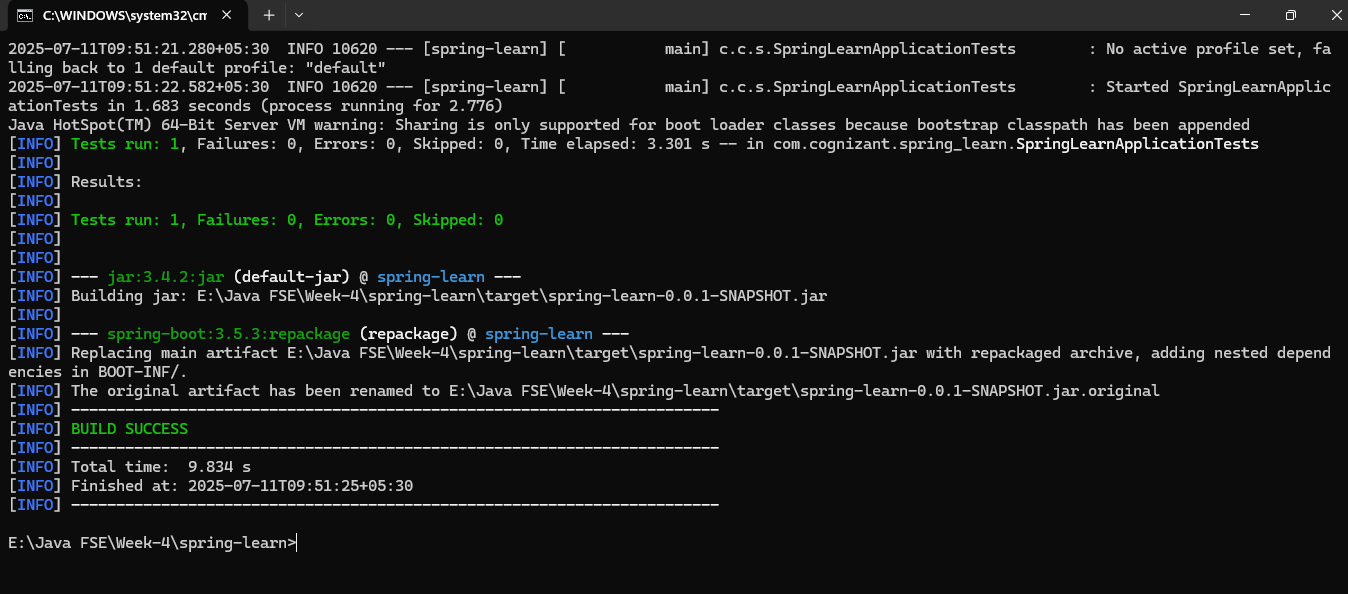
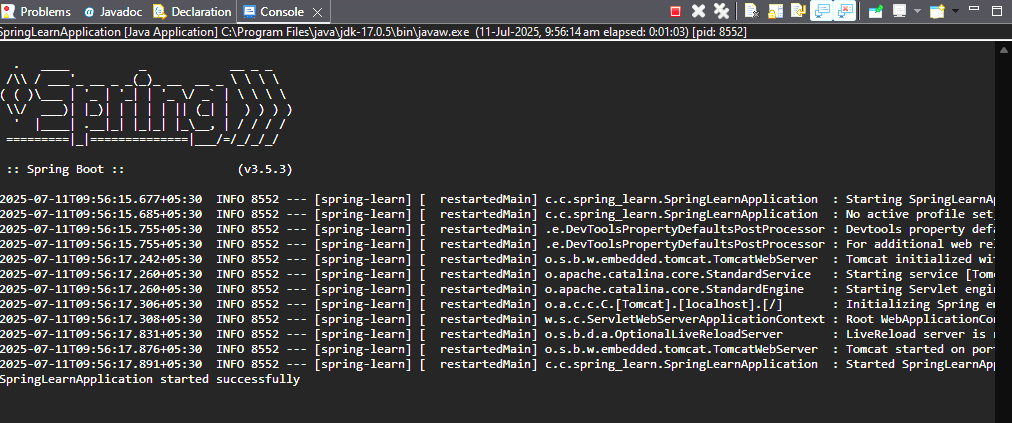
Spring REST using Spring Boot 3

Create a Spring Web Project using Maven

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. Run the SpringLearnApplication class.

OUTPUT:

**1. src/main/java**

This folder contains **Java source code** for the application.

com.cognizant.spring\_learn.SpringLearnApplication.java

### ****2. src/main/resources****

This folder is for **configuration files and static resources**.

### ****3. src/test/java****

This folder contains **test classes** for application.

### ****4. SpringLearnApplication.java****

This is your **main entry point** for the Spring Boot application.

SpringLearnApplication class

package com.cognizant.spring\_learn;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class SpringLearnApplication {

public static void main(String[] args) {

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

System.***out***.println("SpringLearnApplication started successfully");

}

}

### ****5. Purpose of @SpringBootApplication annotation****

Combines three annotations:

* @Configuration , @EnableAutoConfiguration , @ComponentScan.

6. **pom.xml**

Contains:

* **Project Info**
* **Dependencies**
* **Build Plugins**
* **Parent**

### ****1. Project Root****

<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">

* Declares this as a Maven project.
* Defines XML namespaces and schema for Maven POM (Project Object Model) version 4.0.0.

### ****2. modelVersion****

<modelVersion>4.0.0</modelVersion>

* Specifies the **POM model version**. Always **4.0.0** for Maven 2+ projects.

### ****3. parent****

<parent>

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

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

<version>3.5.3</version>

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

</parent>

* Sets **Spring Boot Starter Parent** as the parent POM.
* Provides default configurations for dependencies and plugins.
* Ensures compatibility of Spring Boot versions with libraries.

### ****4. groupId****

<groupId>com.cognizant</groupId>

It defines the **project group name** (company, organization, or domain).

### ****5. artifactId****

<artifactId>spring-learn</artifactId>

The **name of project artifact** (output .jar file will be named spring-learn.jar).

### ****6. version****

<version>0.0.1-SNAPSHOT</version>

Defines Project version.

### ****7. name****

<name>spring-learn</name>

Human-readable name of the project.

### ****8. Description****

<description>Demo project for Spring Web Project using Maven</description>

Short description for documentation or Maven repositories.

### ****9. URL, licenses, developers, scm****

These are placeholders currently empty.

<url/>

<licenses>

<license/>

</licenses>

<developers>

<developer/>

</developers>

<scm>

<connection/>

<developerConnection/>

<tag/>

<url/>

</scm>

* **Purpose:** Used when publishing artifacts to repositories.
  + <url>: Project homepage.
  + <licenses>: License details.
  + <developers>: Developer information.
  + <scm>: Source code management (Git repository details).

### ****10. Properties****

<properties>

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

</properties>

It sets **Java version compatibility** for your project to Java 17.

### ****11. Dependencies****

#### 🔹 ****spring-boot-starter-web****

<dependency>

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

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

</dependency>

Adds **Spring MVC, embedded Tomcat server, Jackson** for building web applications and REST APIs.

#### 🔹 ****spring-boot-devtools****

<dependency>

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

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

<scope>runtime</scope>

<optional>true</optional>

</dependency>

Provides **hot reload support** for faster development.

#### 🔹 ****spring-boot-starter-test****

<dependency>

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

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

<scope>test</scope>

</dependency>

Provides **JUnit, Mockito, Spring Test** for unit and integration testing.

### ****12. Build****

<build>

<plugins>

<plugin>

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

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

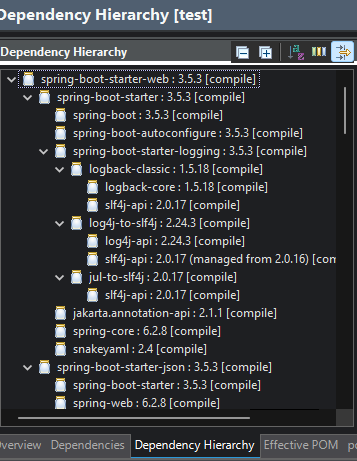
</plugin>

</plugins>

</build>

It Configures **Spring Boot Maven Plugin**.

Enables you to build executable jar files with embedded Tomcat.

****

Spring Core – Load Country from Spring Configuration XML

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

http://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 Class

package com.cognizant.spring\_learn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

public class Country {

private String code;

private String name;

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

// Empty constructor

public Country() {

***LOGGER***.debug("Inside Country Constructor.");

}

public String getCode() {

***LOGGER***.debug("Inside getCode");

return code;

}

public void setCode(String code) {

***LOGGER***.debug("Inside setCode");

this.code = code;

}

public String getName() {

***LOGGER***.debug("Inside getName");

return name;

}

public void setName(String name) {

***LOGGER***.debug("Inside setName");

this.name = name;

}

*@Override*

public String toString() {

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

}

}

SpringLearnApplication Class

package com.cognizant.spring\_learn;

import org.slf4j.Logger;

import org.slf4j.LoggerFactory;

import org.springframework.context.ApplicationContext;

import org.springframework.context.support.ClassPathXmlApplicationContext;

import org.springframework.boot.SpringApplication;

import org.springframework.boot.autoconfigure.SpringBootApplication;

*@SpringBootApplication*

public class SpringLearnApplication {

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

public static void main(String[] args) {

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

*displayCountry*();

}

*@SuppressWarnings*("resource")

public static void displayCountry() {

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

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

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

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

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

}

}

**id attribute**

It is used as unique **identifier for the bean**.

**class attribute**

It defines fully qualified **class name of the bean**.

**property tag**

It is used to **set values to fields** of the bean via setters **name attribute (of property tag)**.

**value attribute (of property tag)**

It is the **value to assign** to the property.

**ApplicationContext**

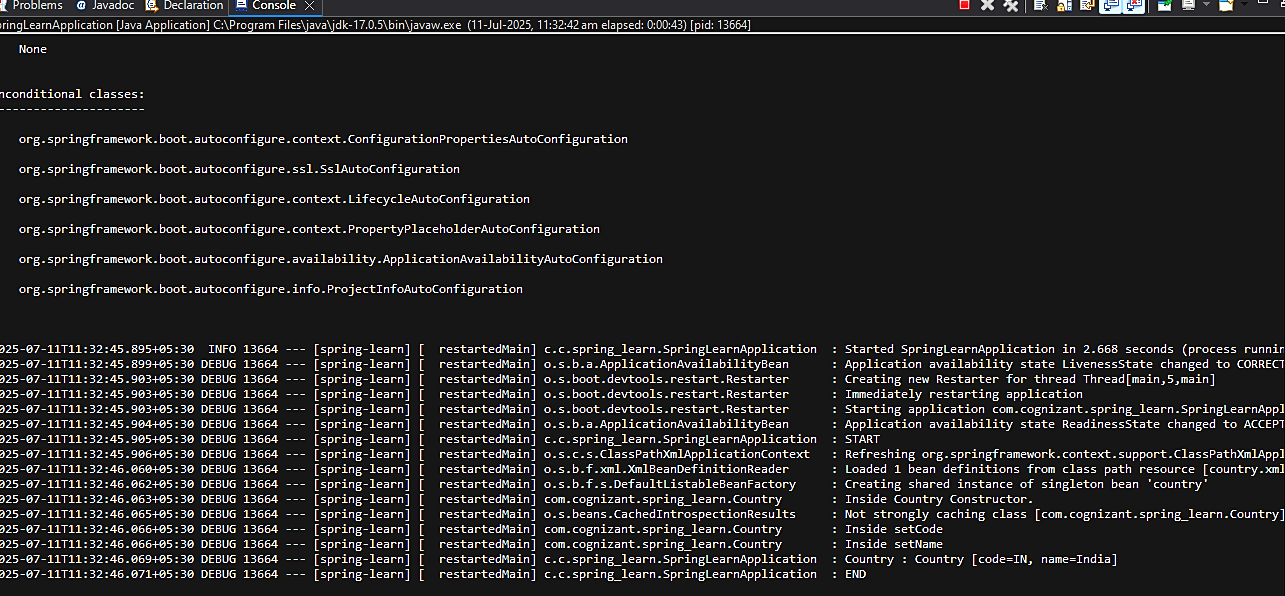
It is a **Spring Container interface** to access beans and configurations.

**ClassPathXmlApplicationContext**

It helps in implementation of ApplicationContext and loads configuration from **classpath resource (e.g. country.xml)**.

### ****What happens when**** context.getBean() ****is invoked?****

1. Spring looks up the bean with specified id.
2. Instantiates the bean if not already created.
3. Injects properties defined in XML via setter methods.
4. Returns the bean reference.

OUTPUT: