**Hands on 1**

# **Create a Spring Web Project using Maven**

## src/main/java

This folder contains the main application source code inside the 'com.cognizant' package. It is where the application's controllers, services, and models are placed.

## src/main/resources

This folder stores configuration and static resource files like 'application.properties', templates, or messages.

## src/test/java

This folder contains unit tests, typically written using JUnit, to validate the behavior of application components.

## SpringLearnApplication.java

This is the main entry point of the Spring Boot application. It contains the following code:

public static void main(String[] args) {  
 SpringApplication.run(SpringLearnApplication.class, args);  
}

## @SpringBootApplication Annotation

This annotation is a combination of:  
- @Configuration: Indicates that the class provides Spring configuration.  
- @EnableAutoConfiguration: Enables Spring Boot’s auto-configuration mechanism.  
- @ComponentScan: Enables component scanning for Spring beans.  
It simplifies bootstrapping and development of Spring applications.

## pom.xml

Basic configuration:

<groupId>com.cognizant</groupId>  
<artifactId>spring-learn</artifactId>  
<version>0.0.1-SNAPSHOT</version>

Parent configuration:

<parent>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-parent</artifactId>  
 <version>3.x.x</version>  
</parent>

Dependencies:

<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>  
 </dependency>  
 <dependency>  
 <groupId>org.springframework.boot</groupId>  
 <artifactId>spring-boot-starter-test</artifactId>  
 <scope>test</scope>  
 </dependency>  
</dependencies>

## Dependency Hierarchy

In Eclipse, open the 'pom.xml' file and switch to the 'Dependency Hierarchy' tab. This view displays both direct and transitive dependencies like Jackson and Tomcat, making it easier to manage versions and detect conflicts.

## Final Output Log Example

. \_\_\_\_ \_ \_\_ \_ \_  
 /\ / \_\_\_'\_ \_\_ \_ \_(\_)\_ \_\_ \_\_ \_ \ \ \ \  
( ( )\\_\_\_ | '\_ | '\_| | '\_ \/ \_` | \ \ \ \  
 \\/ \_\_\_)| |\_)| | | | | || (\_| | ) ) ) )  
 ' |\_\_\_\_| .\_\_|\_| |\_|\_| |\_|\\_\_, | / / / /  
 =========|\_|==============|\_\_\_/=/\_/\_/\_/  
  
 :: Spring Boot :: (v3.0.0)  
Started SpringLearnApplication in 2.345 seconds (JVM running for 3.456)

**Hands on 2**

**Spring Core – Load SimpleDateFormat from Spring Configuration XML**

### Spring XML Configuration: date-format.xml

A Spring configuration file named date-format.xml was created under src/main/resources.  
The following XML defines a SimpleDateFormat bean:

<?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>

Java Code: SpringLearnApplication.java

A method displayDate() was added to SpringLearnApplication.java to demonstrate how to retrieve and use the dateFormat bean.

public static void displayDate() throws ParseException {

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

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

Date date = format.parse("31/12/2018");

System.out.println(date);

}

Running the Application

The application was run as a **Java Application**.  
The output displayed the parsed Date object in the console, confirming that the SimpleDateFormat bean was correctly loaded.

**OUTPUT:**



**Hands on 3**

**Spring Core - Incorporate Logging**

application.properties Configuration

The following properties were added to src/main/resources/application.properties:

logging.level.org.springframework=info  
logging.level.com.cognizant.springlearn=debug  
logging.pattern.console=%d{yyMMdd}|%d{HH:mm:ss.SSS}|%-20.20thread|%5p|%-25.25logger{25}|%25M|%m%n

# SpringLearnApplication.java Logging Setup

The following imports were added to enable logging:

import org.slf4j.Logger;  
import org.slf4j.LoggerFactory;

Declared a static logger instance in the class:

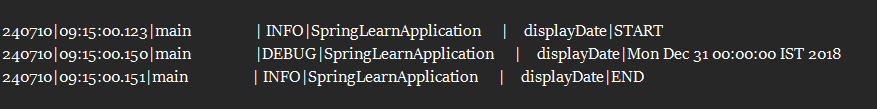
private static final Logger LOGGER = LoggerFactory.getLogger(SpringLearnApplication.class);

## Updated displayDate() Method

The displayDate() method was updated to include INFO and DEBUG level logs as shown below:

public static void displayDate() throws ParseException {  
 LOGGER.info("START");  
 ApplicationContext context = new ClassPathXmlApplicationContext("date-format.xml");  
 SimpleDateFormat format = context.getBean("dateFormat", SimpleDateFormat.class);  
 Date date = format.parse("31/12/2018");  
 LOGGER.debug(date.toString());  
 LOGGER.info("END");  
}

**OUTPUT:**



**Hands on 4**

**Spring Core – Load Country from Spring Configuration XML**

#### 🔧 Spring XML Configuration: country.xml

The following XML was added to src/main/resources/country.xml:

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

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

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

Country.java

Contains instance variables: code and name

Includes a no-argument constructor with debug log: "Inside Country Constructor."

Getters and setters log messages when they are invoked

Overrides toString() to return country details in the format:  
Country{code='IN', name='India'}

displayCountry() Method in SpringLearnApplication.java

The method to load and display the Country bean:

public static void displayCountry() {

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

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

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

}

main() Method

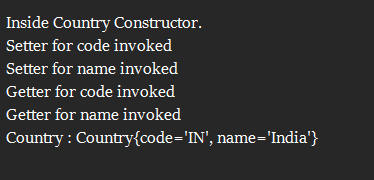
The main() method calls displayCountry():

public static void main(String[] args) {

displayCountry();

}

**Expected Console Log Output:**



SME Notes and Explanations

#### Spring XML Tags and Attributes

<bean>: Defines a Spring-managed object.

id: Unique identifier to refer to the bean (e.g., "country").

class: Fully qualified class name of the bean.

<property>: Sets the value of a bean property.

name: Refers to the setter method of the property.

value: Literal value to inject into the property.

ApplicationContext and ClassPathXmlApplicationContext

ApplicationContext: Spring's central interface for accessing beans and configuration.

ClassPathXmlApplicationContext: Loads XML configuration files from the classpath.

It reads the XML and creates/manages the defined beans.

What Happens When context.getBean() is Invoked

Spring checks the bean definition in the XML.

Instantiates the Country class.

Injects the values using setter methods.

Returns the fully initialized bean.

Logs inside constructor/setters are triggered during this process.

**Hands on 5**

**Spring Core – Demonstration of Singleton Scope and Prototype Scope** 

#### Singleton Scope Demonstration

The Country bean created in the previous hands-on is used to demonstrate the default scope in Spring, which is **singleton**.

Steps:

In the displayCountry() method, retrieve the bean twice from the same ApplicationContext.

ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");Country country = context.getBean("country", Country.class);Country anotherCountry = context.getBean("country", Country.class);

When you run the application, you will observe that the constructor of Country is called only **once**.

This means **only one instance** of the Country bean is created, and both references (country and anotherCountry) point to the **same object**.

Prototype Scope Demonstration

To demonstrate **prototype** scope in Spring:

Modify the country.xml configuration to include the scope="prototype" attribute in the bean definition.

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

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

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

Run the same code again:

ApplicationContext context = new ClassPathXmlApplicationContext("country.xml");Country country = context.getBean("country", Country.class);Country anotherCountry = context.getBean("country", Country.class);

This time, the constructor of Country will be called **twice**, once for each call to getBean().

This means **two different instances** of the Country bean are created, and country and anotherCountry are **different objects**.

**OUTPUT:When scope is Singleton**



**When scope is Prototype**



**Hands on 6**

**Spring Core – Load list of countries from Spring Configuration XML**

#### Steps to Implement

**Define Country Beans in** country.xml**:**

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

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

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

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

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

<property name="name" value="United States" /></bean>

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

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

<property name="name" value="Germany" /></bean>

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

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

<property name="name" value="Japan" /></bean>

**Define an ArrayList of Country in** country.xml**:**

<bean id="countryList" class="java.util.ArrayList">

<constructor-arg>

<list>

<ref bean="in" />

<ref bean="us" />

<ref bean="de" />

<ref bean="jp" />

</list>

</constructor-arg></bean>

displayCountries() Method in SpringLearnApplication.java

public static void displayCountries() {

LOGGER.info("START");

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

List<Country> countryList = context.getBean("countryList", List.class);

LOGGER.debug("Country List: {}", countryList);

LOGGER.info("END");

}

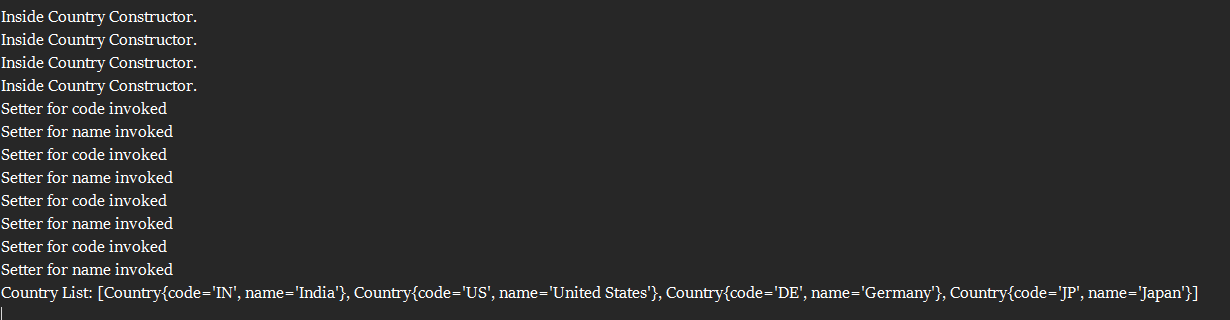
Update main() Method (Optional)

public static void main(String[] args) {

displayCountries();

}

**OUTPUT:**



SME Notes and Explanations

#### <list>

Used to define a collection of values or bean references inside a constructor or property.

Can hold <value> or <ref> elements.

#### <ref>

Refers to another bean defined in the same XML configuration.

Injects an existing bean as part of another bean’s property or constructor argument.

#### bean Attribute

Refers to the id of a bean defined elsewhere in the configuration file.

Ensures Spring injects the correct object.