**1. src/main/java**

This folder contains all your main Java source code — including:

* Your main Spring Boot application (SpringLearnApplication.java)
* Any controller, service, model, repository classes you add later.

This is your primary codebase.

**2. src/main/resources**

This folder contains configuration files such as:

* application.properties or application.yml: used for configuration like port, logging level, DB properties, etc.
* Any Spring configuration XML files (if you're using XML-based bean config like country.xml)
* Static resources, templates, and messages.

**3. src/test/java**

This folder holds unit test classes. Spring Boot uses JUnit (and optionally Mockito) for writing test cases.

You might see a file like SpringLearnApplicationTests.java here by default.

**4. SpringLearnApplication.java**

**package** com.cognizant;

**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("Inside Main.......");

}

}

* SpringApplication.run() is the **entry point** of the Spring Boot application.
* You can add log statements to confirm it's running.
* It starts the embedded Tomcat server by default.

**5. Purpose of @SpringBootApplication**

This is a meta-annotation that combines:

* @Configuration: Allows Spring to register extra beans or configurations.
* @EnableAutoConfiguration: Tells Spring Boot to automatically configure the application based on dependencies in the classpath.
* @ComponentScan: Scans the package for annotated components like @Controller, @Service, @Repository.

This makes your Spring Boot setup quick and clean.

**6. pom.xml**

This is the **Maven Project Object Model file** which defines:

* Project metadata (groupId, artifactId, version)
* Dependencies like Spring Boot, Web, DevTools
* Build configuration
* Plugin settings

Example excerpt from pom.xml:

<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>

**7. Walkthrough of XML Configuration**

If you're also using Spring XML for bean definition:

Example: country.xml

<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.springlearn.Country">

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

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

</bean>

</beans>

This XML will be loaded using ClassPathXmlApplicationContext in Java code.

**🔹 8. Dependency Hierarchy**

In Eclipse:

* Open pom.xml
* Click on the "Dependency Hierarchy" tab at the bottom

You will see a tree view of all:

* Direct dependencies (Spring Boot Starter Web, DevTools)
* Transitive dependencies (Jackson, Tomcat, SLF4J, etc.)
* Their versions and scopes