**WEEK\_4**

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

SME to explain the following aspects:

**1] bean tag, id attribute, class attribute, property tag, name attribute, value attribute**

**Ans:**  
The Spring configuration file (country.xml) defines a bean representing a country using the <bean> tag. It allows Spring to manage the lifecycle and dependencies of the object.

**Example from your code:**

xml

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

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

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

</bean>

* **<bean> tag**: Defines a bean that Spring should manage.
* **id attribute**: This is the unique identifier for the bean in the Spring container. In this case, it's "country".
* **class attribute**: Specifies the fully qualified class name of the bean to instantiate — here it is com.cognizant.springlearn.Country.
* **<property> tag**: Used to inject values into the bean via **setter injection**.
* **name attribute (inside <property>)**: Matches the setter method — for example, name="code" maps to setCode().
* **value attribute (inside <property>)**: The literal value passed to the setter. For instance, value="IN" calls setCode("IN").

In summary, this configuration creates a Country object and sets its code and name properties using Spring's XML-based dependency injection.

**2] ApplicationContext, ClassPathXmlApplicationContext**

**Ans:**  
In your application, the following line is used to load the Spring configuration:

java

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

* **ApplicationContext**: This is the central interface for accessing Spring beans. It represents the Spring container and is responsible for instantiating, configuring, and assembling beans.
* **ClassPathXmlApplicationContext**: A concrete implementation of ApplicationContext that reads configuration from an XML file located in the classpath.

When this line executes:

* Spring loads the country.xml file.
* It parses all <bean> definitions.
* It instantiates the Country bean using the default constructor.
* It injects the values for code and name using the setter methods.

So, in your app, this context is used in displayCountry() to load and fetch the Country bean defined in XML.

**3] What exactly happens when context.getBean() is invoked**

**Ans:**  
In this code:

java

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

Spring performs several steps when getBean() is called:

1. **Lookup**: It searches for the bean with ID "country" in the XML (country.xml).
2. **Instantiation**: It creates an object of com.cognizant.springlearn.Country using the no-argument constructor. You'll see the log:

scss

DEBUG Inside Country Constructor.

1. **Dependency Injection**: It sets the properties using the setter methods:
   * setCode("IN") → logs: Setting Country Code.
   * setName("India") → logs: Setting Country Name.
2. **Return**: The fully initialized bean is returned and stored in the country variable.
3. **Logging**: The following debug message is printed:

pgsql

DEBUG Country : Country [code=IN, name=India]

In conclusion, context.getBean() triggers a complete lifecycle: object creation, dependency injection, and returning a fully initialized bean ready to use in the application.

**Hello World RESTful Web Service**

SME to explain following aspects

**1] In network tab of developer tools show the HTTP header details received**

Ans: Details are shown in output (Chrome Developer Tools → Network tab → Headers section).

**2] In Postman click on "Headers" tab to view the HTTP header details received**

Ans: Details are shown in output (Postman → Response Headers tab after sending request).

**REST – Country Web Service**

SME to explain following aspects

**1] What happens in the controller method?**

**Ans:**  
When a GET request is made to the URL http://localhost:8083/country, the method getCountryIndia() in CountryController is executed. Inside this method:

* A Spring application context is created using applicationContext.xml.
* The bean with ID "in" is retrieved from the context.
* This bean is an instance of the Country class with values code = "IN" and name = "India".
* The Country object is then returned as the response.
* The controller also logs "START" and "END" using SLF4J Logger for debugging.

**2] How the bean is converted into JSON response?**

**Ans:**  
The returned Country object is automatically converted to a JSON response by Spring Boot. This is done using **Jackson**, which is the default HTTP message converter in Spring Boot.  
Because the Country class has getter methods (getCode() and getName()), Spring uses these methods to serialize the object into JSON format like this:

json

Copy code

{

"code": "IN",

"name": "India"

}

This happens automatically because of the @RestController annotation, which combines @Controller and @ResponseBody, ensuring that the return value is directly written to the HTTP response as JSON.

**3] In network tab of developer tools show the HTTP header details received**

**Ans:**  
Details are shown in output (Chrome Developer Tools → Network tab → Headers section).

**4] In postman click on "Headers" tab to view the HTTP header details received**

**Ans:**  
Details are shown in output (Postman → Response → Headers tab).