**WEEK\_4**

**REST - Country Web Service**

SME to explain the following aspects:

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

Ans: **When a REST API request hits your controller:**

java

@RestController

public class HelloController {

@GetMapping("/hello")

public String hello() {

return "Hello World!!";

}

}

**Internally:**

1. The Spring container scans for @RestController and registers the class as a handler for HTTP requests.
2. When a GET /hello request is received, Spring matches it to the hello() method.
3. The method executes and returns the string "Hello World!!".
4. Since @RestController includes @ResponseBody, the return value is written directly to the HTTP response body.

If you return a **Java object** instead of a String, Spring automatically converts it to **JSON** using **Jackson** (a JSON processor).

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

**Ans: Example:**

java

@GetMapping("/employee")

public Employee getEmployee() {

return new Employee(1, "John Doe", "Developer");

}

**➤ Steps:**

1. Spring returns an Employee object.
2. The **Jackson** library (included by default in Spring Boot) uses reflection to inspect the object.
3. It serializes the object into JSON:

json

{

"id": 1,

"name": "John Doe",

"designation": "Developer"

}

1. This JSON is set as the **response body** with Content-Type: application/json.

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

**Ans:** details are in output

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

Ans:details are in output

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

SME to provide more detailing about the following aspects:

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

Ans: **1. <bean> Tag in Spring XML**

The <bean> tag is used in Spring's XML configuration file (country.xml) to define and configure a Spring-managed object (also known as a Spring **bean**).

**Structure:**

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

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

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

</bean>

**id Attribute:**

* The **identifier** for the bean.
* It is used to **refer to the bean** inside the Spring context using context.getBean("country").

Think of it like a variable name that stores the bean instance.

**class Attribute:**

* Specifies the **fully qualified class name** of the bean.
* Spring uses **reflection** to instantiate the class using the no-arg constructor.

Example: com.cognizant.springlearn.Country

**<property> Tag:**

Used to **inject values** into the bean’s fields via **setter methods** (Setter Injection).

**name Attribute:**

* Name of the **property** (field) in the class.
* Spring internally calls the corresponding **setter method** (e.g., setCode() for name="code").

**value Attribute:**

* Literal value assigned to the property.

Spring uses reflection to call the setter method with this value after bean instantiation.

**2]ApplicationContext, ClassPathXmlApplicationContext**

Ans: **ApplicationContext**

* The **central interface** of the Spring Container.
* Responsible for:
  + Loading bean definitions.
  + Instantiating and wiring beans.
  + Managing bean life cycle.
  + Handling configuration (XML, annotation, Java config).

It is an advanced version of BeanFactory.

**ClassPathXmlApplicationContext**

* A concrete implementation of ApplicationContext.
* Loads the Spring XML configuration file from the **classpath** (typically src/main/resources).

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

This:

* Loads country.xml
* Parses all <bean> definitions
* Creates and wires beans into memory

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

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

Step-by-step Breakdown:

**Lookup:**

Spring looks for a bean with id="country" in its internal registry.

Instantiation (if not already created):

If the bean is singleton (default), Spring checks if it’s already created.

If not, Spring creates it using the no-argument constructor.

**Dependency Injection:**

Spring scans <property> tags.

**For each property:**

Matches it to a setter method (setCode, setName).

Converts the string value to the appropriate type (e.g., "IN" to String).

Calls the setter with the value.

**Returns the Bean:**

Once the bean is fully initialized, Spring returns it to your code.

Example Flow for country.xml:

xml

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

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

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

</bean>

Step 1: Create object → new Country()

Step 2: Call → setCode("IN")

Step 3: Call → setName("India")

Step 4: Return the fully populated object

**Hello World RESTful Web Service**

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

Ans: details are in output

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

Ans:details are in output