**Spring XML Configuration and Bean Instantiation Details**

**SME Detailed Explanations**

**1. <bean> Tag and Attributes**

In Spring, the <bean> tag is used in the XML configuration file to define a bean — a Java object that Spring will manage. It informs the Spring IoC (Inversion of Control) container to create and manage the lifecycle of that object.

* **id**: This is the unique name of the bean. It is used to retrieve the bean using context.getBean("id").
* **class**: Specifies the fully qualified class name of the bean (e.g., com.cognizant.springlearn.Country). Spring uses this to instantiate the class via reflection.
* **<property>**: Used to inject values into the fields of the class using setter-based injection.
  + **name** (attribute): This must match the Java property name (i.e., the variable name or the setter method name).
  + **value** (attribute): The actual data you want to set for that property, like "IN" for code or "India" for name.

**2. ApplicationContext and ClassPathXmlApplicationContext**

* **ApplicationContext** is a central Spring interface for managing beans. It provides advanced features like event propagation, internationalization, and bean lifecycle management.
* **ClassPathXmlApplicationContext** is an implementation of ApplicationContext that loads the Spring configuration XML file from the classpath. When you write:

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

you're telling Spring to look for country.xml in the classpath and load all beans defined inside.

**3. What Happens When context.getBean() is Invoked?**

When you call context.getBean("country"), Spring performs the following steps:

1. **Locate the bean definition** in the configuration file using the specified id.
2. **Load the class** mentioned in the class attribute using Java Reflection.
3. **Instantiate the object** of that class.
4. **Inject property values** using setter methods, as configured via <property> tags.
5. **Return the fully initialized object** to the caller.

If the bean scope is singleton (which is the default), this object is cached and the same instance is returned for all future calls to getBean().