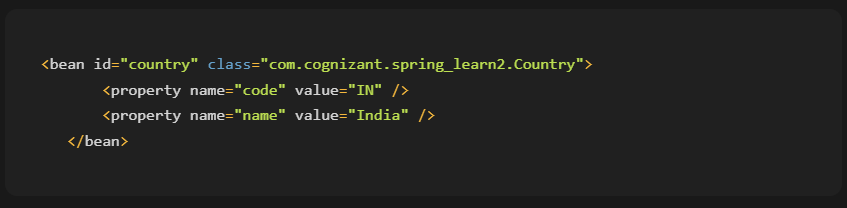
### **Spring Core-Load Country from Spring Configuration XML**

#### **1. Tags & Attributes**

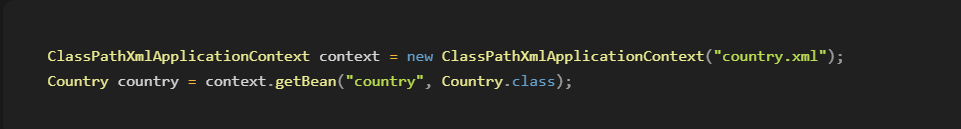
* **<bean> tag**:  
   Declares a bean (an object) that the Spring container should manage. It defines how a specific object is created, configured, and assembled by Spring.
* **id attribute**:  
   Uniquely identifies the bean within the Spring container. It is used later to retrieve this specific bean using methods like getBean("id").
* **class attribute**:  
   Specifies the fully qualified name of the Java class to be instantiated and managed as a bean. This tells Spring which class to load for this bean.
* **<property> tag**:  
   Used to inject values into the bean’s fields through setter methods. This is a part of Spring’s Dependency Injection mechanism.
* **name attribute** (within <property>):  
   Refers to the name of the property in the Java class (usually private member variable). It should match the property for which the setter method exists.
* **value attribute** (within <property>):  
   The actual value to be assigned to the corresponding property. Spring uses this to call the appropriate setter method and inject the value.

Sample Bean :   


#### **2. ApplicationContext and ClassPathXmlApplicationContext**

* **ApplicationContext**:  
   It is a central interface in Spring for providing configuration information to the application. It represents the Spring IoC container and is responsible for instantiating, configuring, and assembling beans.
* **ClassPathXmlApplicationContext**:  
   A concrete implementation of ApplicationContext that loads the configuration from an XML file located in the classpath.  
   It parses the XML, instantiates the defined beans, and wires the dependencies accordingly.

Sample code



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

When the method context.getBean("beanId", ClassName.class) is called:

1. **Spring retrieves the bean** with the given id from the configuration file (e.g., country.xml).
2. It **instantiates the bean** if it hasn’t been created already (based on the scope).
3. It **injects the configured property values** using setter methods (like setCode() and setName()).
4. Finally, it returns the fully initialized object to the caller for use.

If debug logs are enabled, we can trace which constructor and methods (like getters and setters) are invoked during this process.