**SME Explanation: Spring Core - Load Country from Spring Configuration XML**

**Project Overview:**  
This hands-on task demonstrates the use of Spring Core to load a bean configuration from an XML file and fetch data using ApplicationContext. The goal is to model country data and display it through a Spring-managed bean.

**Spring Configuration Details:**

**1. bean tag**:

* Declares a Spring bean to be managed by the Spring container.

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

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

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

</bean>

**2. id attribute**:

* A unique identifier for the bean. Used to fetch the bean using context.getBean("country").

**3. class attribute**:

* Fully-qualified name of the class that Spring will instantiate.

**4. property tag**:

* Used to inject values into bean properties.

**5. name attribute (in property)**:

* Matches the name of the setter method or instance variable in the class.

**6. value attribute (in property)**:

* The value that will be passed to the setter method.

**Country.java:**  
Contains:

* code and name variables
* Constructor with debug log: "Inside Country Constructor."
* Getter/Setter with logs
* toString() method for output display

**ApplicationContext:**

* Central interface for providing configuration information to an application.

**ClassPathXmlApplicationContext:**

* A concrete implementation of ApplicationContext that loads the context definition from an XML file located in the classpath.

**Example:**

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

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

LOGGER.debug("Country : {}", country.toString());

**What happens when context.getBean() is invoked:**

1. Spring reads the country.xml file from the classpath.
2. It locates the <bean> definition with id="country".
3. Spring creates an instance of com.cognizant.springlearn.Country.
4. It injects the property values (code = "IN", name = "India") using setters.
5. The constructed and fully initialized object is returned.
6. If logging is enabled, logs will show constructor and method calls.

**Conclusion:**  
This hands-on demonstrates core Spring features like XML-based bean configuration, dependency injection using setter methods, and use of ApplicationContext for retrieving beans. It helps in understanding the foundation of Spring's IoC (Inversion of Control) mechanism.