Spring Core – Load Country from Spring Configuration XML

An airlines website is going to support booking on four countries. There will be a drop down on the home page of this website to select the respective country. It is also important to store the two-character ISO code of each country.

Code	Name
US	United States
DE	Germany
IN	India
JP	Japan

Above data has to be stored in spring configuration file. Write a program to read this configuration file and display the details

Steps to implement

- Pick any one of your choice country to configure in Spring XML configuration named country.xml.
- Create a bean tag in spring configuration for country and set the property and values

```
<bean id="country" class="com.cognizant.springlearn.Country">
    <property name="code" value="IN" />
```

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

- Create Country class with following aspects:

o Instance variables for code and name

o Implement empty parameter constructor with inclusion of debug log within the constructor with log message as “Inside Country Constructor.”

o Generate getters and setters with inclusion of debug with relevant message within each setter and getter method.

o Generate toString() method

- Create a method displayCountry() in SpringLearnApplication.java, which will read the country bean from spring configuration file and display the country details. ClassPathXmlApplicationContext, ApplicationContext and context.getBean(“beanId”, Country.class). Refer sample code for displayCountry() method below.

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

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

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

- Invoke displayCountry() method in main() method of SpringLearnApplication.java.
- Execute main() method and check the logs to find out which constructors and methods were invoked.

1. Country Class

```
package com.cognizant.springlearn.model;

public class Country {
   private String code;
   private String name;

   // Default constructor (required for Spring)
   public Country() {}

   // Parameterized constructor
   public Country(String code, String name) {
      this.code = code;
      this.name = name;
   }

   // Getters and setters
   public String getCode() {
      return code;
   }
```

```
    public void setCode(String code) {
        this.code = code;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public String toString() {
        return "Country [code=" + code + ", name=" + name + "]";
    }
}
```

2. Spring Configuration XML

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

```
    xsi:schemaLocation="http://www.springframework.org/schema/beans
                        http://www.springframework.org/schema/beans/spring-
beans.xsd">

   <!-- India Bean Definition -->
   <bean id="in" class="com.cognizant.springlearn.model.Country">
      <property name="code" value="IN"/>
      <property name="name" value="India"/>
   </bean>

   <!-- Other Country Beans -->
   <bean id="us" class="com.cognizant.springlearn.model.Country">
      <property name="code" value="US"/>
      <property name="name" value="United States"/>
   </bean>

   <bean id="jp" class="com.cognizant.springlearn.model.Country">
      <property name="code" value="JP"/>
      <property name="name" value="Japan"/>
   </bean>

   <bean id="de" class="com.cognizant.springlearn.model.Country">
      <property name="code" value="DE"/>
```

```
    <property name="name" value="Germany"/>
  </bean>
</beans>
```

3. Main Application to Load Configuration

```
package com.cognizant.springlearn;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
import com.cognizant.springlearn.model.Country;

public class MainApp {
  public static void main(String[] args) {
    // Load the Spring configuration file
    ApplicationContext context = new
ClassPathXmlApplicationContext("country.xml");

    // Retrieve beans
    Country india = (Country) context.getBean("in");
    Country usa = (Country) context.getBean("us");

    // Print country details
```

```
        System.out.println("India: " + india);
        System.out.println("USA: " + usa);

        // Close the context (optional)
        ((ClassPathXmlApplicationContext) context).close();
    }}
```

SME to provide more detailing about the following aspects:

- **bean tag, id attribute, class attribute, property tag, name attribute, value attribute**

<bean> **Tag**

- **Purpose**: Defines a Spring-managed object (bean)
- **Lifecycle**: By default, beans are singleton-scoped (one instance per container)
- **Attributes**:
    - id: Unique identifier for bean retrieval (e.g., "in")
    - class: Fully qualified class name (e.g., com.cognizant.springlearn.model.Country)

<property> **Tag**

- **Purpose**: Dependency injection via setter methods
- **Attributes**:

- name: Matches the Java bean property name (case-sensitive, e.g., "code" → setCode())
- value: Injects literal values (strings are automatically converted to property type)

- **ApplicationContext, ClassPathXmlApplicationContext**

ApplicationContext **Interface**

- **Role**: Central Spring IoC container interface
- **Key Capabilities**:
  - Bean instantiation/wiring
  - Configuration source agnostic (XML, annotations, Java config)
  - Event propagation
  - Resource loading

ClassPathXmlApplicationContext

- **Specialization**: Loads XML configs from classpath
- **Initialization Flow**:
  - Parses XML using SAX parser
  - Creates BeanDefinition registry
  - Pre-instantiates singletons (if configured)
  - Validates dependency graph

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

1. **Check Registry**
    - Looks for existing instance (singleton scope only)
2. **Create Instance**
    - Loads class → creates object via reflection (constructor/factory)
3. **Inject Dependencies**
    - Sets properties (XML <property> → setter methods)
    - Handles autowiring if configured
4. **Initialize**
    - Calls @PostConstruct, InitializingBean, init methods
    - Applies AOP proxies if needed
5. **Return Bean**
    - Caches singleton (if applicable)
    - Returns fully configured instance

SUPERSET ID: 6407636

KANMANI MURUGHAIYAN