Spring - Constructor Injection with Non-String Map

Last Updated : 23 Jul, 2025

Constructor Injection is a widely used technique in the Spring Framework for injecting dependencies through a class constructor. This method ensures that all required dependencies are provided at the time of object creation, making the class immutable and easy to test. In this article, we will explore Constructor Injection with a Map, where both the keys and values are non-String objects. We will configure the Map using Spring XML and inject it into a class.

Constructor Injection with Collection

Spring Framework provides the flexibility to inject collection values via a constructor. The following types of collections can be used inside the <constructor-arg> tag:

- List
- Set
- Map

Constructor Injection with Non-String Map

In the example below, we will demonstrate how to perform constructor injection with a Map where both the key and value are non-String objects

The Map will have:

- **Key**: An Employee object with the fields (Name, Employee ID, Department).
- **Value**: An Address object with the fields (House No, Pincode, State, Country).

Implementation of Constructor Injection with Non-String Map in Spring

Below is the implementation of constructor injection with a non-String Map in a Spring application.

Step 1: Create the Company.java Class

The Company class contains a Map of Employee-Address pairs, which is injected via the constructor.

```
import java.util.Map;

public class Company {
    private Map<Employee, Address> employeeAddressMap;

    // Constructor for dependency injection
    public Company(Map<Employee, Address> employeeAddressMap) {
        this.employeeAddressMap = employeeAddressMap;
    }

    // Method to display employee and address details
    public void display() {
        for (Map.Entry<Employee, Address> entry :
employeeAddressMap.entrySet()) {
            System.out.println("Employee Data -> " + entry.getKey() + ",
Address -> " + entry.getValue());
        }
    }
}
```

Explanation:

- This class holds a Map<Employee, Address>, which will be injected using Spring.
- The constructor is used for dependency injection.
- The display() method iterates over the map and prints the Employee-Address pairs.

Step 2: Create the Employee.java Class

This class represents an employee with attributes: name, employeeId, and department.

```java
public class Employee {
    private String name;
    private int employeeId;
    private String department;

    // Constructor
    public Employee(String name, int employeeId, String department) {
        this.name = name;
        this.employeeId = employeeId;
        this.department = department;
    }

    // Override toString() method
    @Override
    public String toString() {
        return "[" + name + ", " + employeeId + ", " + department + "]";
    }
}
```

Explanation:

- This class has three fields: name, employeeId, and department.
- The constructor initializes these values.
- The toString() method is overridden to provide a meaningful representation.

Step 3: Create the Address.java Class

The Address class represents an address with fields: house number, pincode, state, and country.

```java
public class Address {
    private String houseNo;
    private String pincode;
    private String state;
    private String country;

    // Constructor
    public Address(String houseNo, String pincode, String state, String
country) {
        this.houseNo = houseNo;
        this.pincode = pincode;
        this.state = state;
        this.country = country;
    }
    // Override toString() method
    @Override
    public String toString() {
        return "[" + houseNo + ", " + pincode + ", " + state + ", " + country
+ "]";
    }
}
```

Explanation:

- The Address class defines attributes for an address.
- The constructor initializes the attributes.
- The toString() method is overridden to return a readable string format.

Step 4: Configure applicationContext.xml

This XML file configures Spring beans and sets up Constructor Injection with a Map.

applicationContext.xml:

```
<beans xmlns="http://www.springframework.org/schema/beans/"
       xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="http://www.springframework.org/schema/beans/
           http://www.springframework.org/schema/beans//spring-beans.xsd">

    <!-- Employee Beans -->
    <bean id="employee1" class="Employee">
        <constructor-arg value="Ram" />
        <constructor-arg value="101" />
        <constructor-arg value="Software Testing" />
    </bean>

    <bean id="employee2" class="Employee">
        <constructor-arg value="Shyam" />
        <constructor-arg value="102" />
        <constructor-arg value="Development" />
    </bean>

    <!-- Address Beans -->
    <bean id="address1" class="Address">
        <constructor-arg value="110/4" />
        <constructor-arg value="128933" />
        <constructor-arg value="Delhi" />
        <constructor-arg value="India" />
    </bean>

    <bean id="address2" class="Address">
        <constructor-arg value="201/5" />
        <constructor-arg value="560001" />
        <constructor-arg value="Bangalore" />
        <constructor-arg value="India" />
    </bean>

    <!-- Map Configuration -->
    <bean id="employeeAddressMap" class="java.util.HashMap">
        <constructor-arg>
            <map>
                <entry key-ref="employee1" value-ref="address1" />
                <entry key-ref="employee2" value-ref="address2" />
            </map>
        </constructor-arg>
    </bean>

    <!-- Company Bean -->
    <bean id="company" class="Company">
        <constructor-arg ref="employeeAddressMap" />
    </bean>
</beans>
```

Explanation:

- Two Employee beans (employee1, employee2) are created with sample data.
- Two Address beans (address1, address2) are created.
- A HashMap is configured, mapping Employee objects to Address objects.
- The Company bean is injected with the employeeAddressMap.

Step 5: Create the Test.java Class

```
import org.springframework.beans.factory.BeanFactory;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class Test {
    public static void main(String[] args) {
        // Load the Spring configuration file
        BeanFactory factory = new
ClassPathXmlApplicationContext("applicationContext.xml");

        // Retrieve the Company bean
        Company company = (Company) factory.getBean("company");

        // Display employee and address details
        company.display();
    }
}
```

Explanation:

- The ClassPathXmlApplicationContext loads the applicationContext.xml file.
- The company bean is retrieved from the Spring container.
- The display() method prints the Employee-Address details.

Output:

```
Employee Data -> [Ram, 101, Software Testing], Address -> [110/4,
128933, Delhi, India]

Employee Data -> [Shyam, 102, Development], Address -> [201/5,
560001, Bangalore, India]
```

Conclusion

- We used Constructor Injection in Spring to inject a Map<Employee, Address>.
- The applicationContext.xml file was used to define the dependencies.
- This approach ensures loose coupling and better testability of the application.