Spring - Setter Injection with Map

Last Updated : 23 Jul, 2025

Spring Framework is a powerful tool for Java developers because it offers features like Dependency Injection (DI) to simplify application development. One of the key features of Spring is **Setter Injection**, which allows us to inject dependencies into our beans using setter methods. In this article, we will discuss **Spring Setter Injection with Map**, It is a very useful approach for managing key-value pairs in our spring applications.

Dependency Injection

Dependency injection (DI) is a process where the objects define their dependencies i.e., the other objects they work with. It happens only through constructor arguments, arguments to a factory method, or properties that are set on the object instance after it is constructed or returned from a factory method. Then the container is responsible for injecting those dependencies when it creates the bean. This process is fundamentally the inverse and is hence also called Inversion of Control (IoC).

Dependency injection exists in two ways,

1. Constructor-based dependency injection – This is done by the container invoking a constructor with a number of arguments, each representing a dependency. We need to use **<constructor-arg>** sub-element of the <bean> tag for constructor injection.

```
<bean id="beanId" class="BeanClass">
    <constructor-arg type="String" value="test"/>
    <constructor-arg type="int" value="1002"/>
```

```
</bean>
```

2. Setter-based dependency injection – This is done by the container calling setter methods on beans after invoking a no-argument constructor or no-argument static factory method to instantiate the bean. We need to use the **<property>** sub-element of the <bean> tag for setter injection.

```
<bean id="beanId" class="BeanClass">
    <property name="secondBean" ref="SecondBean"/>
    <property name="message" value="This is message."/>
</bean>
```

Setter Injection with Map

Setter Injection with Map is a technique in Spring Framework that allows us to inject a collection of key-value pairs into a bean using setter methods. This is particularly useful when we need to manage dynamic or configurable data, such as framework details, environment properties, or any other key-value-based information. Instead of hardcoding these values, we can inject them via Spring's configuration, making our application more flexible and maintainable.

This approach is ideal for scenarios where dependencies are optional or need to be injected dynamically at runtime.

Implementation

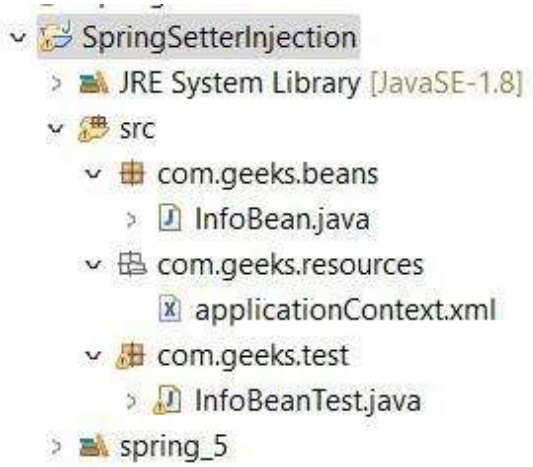
We will create a simple Spring application in Eclipse to display the Name, Welcome Message, and Framework details as the output in the console.

Step 1: Project Structure

- **Create a new Java project:** SpringSetterInjection.
- Add all the required Spring Jar files to the project.

- **Create a bean class:** InfoBean.java, to define all the properties and getter/setter methods of those properties.
- Create XML configuration file: applicationContext.xml, for the bean configurations.
- **Create bean test class:** InfoBeanTest.java, to define the application context container and get the bean objects.
- Run the application to get the output.

After creating the project, bean classes, and the XML configuration file, below will be the project structure.



We can download the spring jar files from Maven Repository.

Step 2: Files to be created

1. InfoBean.java:

```
package com.geeks.beans;

import java.util.Map;

public class InfoBean {

    private String name;
    private String msg;
    private Map<String, String> frameworks;


    public Map<String, String> getFrameworks() {
        return frameworks;
    }
```

```
    public void setFrameworks(Map<String, String> frameworks) {
        this.frameworks = frameworks;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }
    public String getMsg() {
        return msg;
    }
    public void setMsg(String msg) {
        this.msg = msg;
    }

    public void display() {
        System.out.println("Hi "+name+", "+msg);
        System.out.println("Framework Names: Description");
        for (Map.Entry<String,String> entry : frameworks.entrySet())
            System.out.println(entry.getKey() + ": " + entry.getValue());
    }

}
```

- In the bean class, we defined properties 'name' and 'msg' as String type and 'frameworks' as Map.
- Generate the getter/setter methods of all the properties.
- Define a method - display(), to print the values of the properties.
- As the field 'framework' is a type map, we need to iterate the map to print the key values.
- So, here we are using a for-each loop to iterate the 'frameworks' and print the values in it.

2. applicationContext.xml:

```
<?xml version="1.0" encoding="UTF-8"?>

<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">

    <bean id="info" class="com.geeks.beans.InfoBean">
        <property name="name" value="Lakshmi" />
        <property name="msg" value="Welcome to GeeksforGeeks!!" />
        <property name="frameworks">
```

```
            <map>
                <entry key="Struts"
                    value="Struts is an open source framework used to develop
Java MVC based web applications."></entry>
                <entry key="Spring"
                    value="Spring is an application framework used to develop
Java Enterprise applications
                            and designed on concept called Dependency
injection."></entry>
                <entry key="Hibernate"
                    value="Hibernate is an object-relational mapping (ORM)
framework that works with relational
                            databases to manage the data."></entry>
            </map>
        </property>
    </bean>

</beans>
```

- In the XML file, we need to include the bean XML schema location for the configuration.
- We are setting the property values of Map using **<map>** and **<entry key="" value="">** tags inside the **<property>** tag.

3. InfoBeanTest.java:

```
package com.geeks.test;

import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;

import com.geeks.beans.InfoBean;

public class InfoBeanTest {

    public static void main(String[] args) {

        ApplicationContext con = new
ClassPathXmlApplicationContext("com/geeks/resources/applicationContext.xml");
        InfoBean infoBean = (InfoBean) con.getBean("info");
        infoBean.display();
    }

}
```

Note: *The application context container creates the bean object and we are getting that bean object to call the display() method.*

Step 3: Run the Application

- Run the test class as a Java project.
- The Spring container will create the 'InfoBean' object and it sets all the property values and make the bean object available to the developer.
- Using the bean object, it calls the display() method and prints the below output in the console.