Setter Injection with Collection Example

We can inject collection values by setter method in spring framework. There can be used three elements inside the **property** element.

It can be:

1. **list**
2. **set**
3. **map**

Each collection can have string based and non-string based values.

Each collection can have string based and non-string based values.

1. **Question.java**
2. **applicationContext.xml**
3. **Test.java**

In this example, we are using list that can have duplicate elements, you may use set that have only unique elements. But, you need to change list to set in the applicationContext.xml file and List to Set in the Question.java file.

**Question.java**

This class contains three properties with setters and getters and displayInfo() method that prints the information. Here, we are using List to contain the multiple answers.

**package** com.javatpoint;

**import** java.util.Iterator;

**import** java.util.List;

**public** **class** Question {

**private** **int** id;

**private** String name;

**private** List<String> answers;

//setters and getters

**public** **void** displayInfo(){

    System.out.println(id+" "+name);

    System.out.println("answers are:");

    Iterator<String> itr=answers.iterator();

**while**(itr.hasNext()){

        System.out.println(itr.next());

    }

}

}

**applicationContext.xml**

The list element of constructor-arg is used here to define the list.

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

<beans

    xmlns="http://www.springframework.org/schema/beans"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xmlns:p="http://www.springframework.org/schema/p"

    xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="q" **class**="com.javatpoint.Question">

<property name="id" value="1"></property>

<property name="name" value="What is Java?"></property>

<property name="answers">

<list>

<value>Java is a programming language</value>

<value>Java is a platform</value>

<value>Java is an Island</value>

</list>

</property>

</bean>

</beans>

**Test.java**

This class gets the bean from the applicationContext.xml file and calls the displayInfo method.

**package** com.javatpoint;

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.beans.factory.xml.XmlBeanFactory;

**import** org.springframework.core.io.ClassPathResource;

**import** org.springframework.core.io.Resource;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

    Resource r=**new** ClassPathResource("applicationContext.xml");

    BeanFactory factory=**new** XmlBeanFactory(r);

    Question q=(Question)factory.getBean("q");

    q.displayInfo();

}

}

# Setter Injection with Non-String Collection (having Dependent Object) Example

If we have dependent object in the collection, we can inject these information by using the **ref** element inside the **list**, **set** or **map**. Here, we will use list, set or map element inside the **property** element.

In this example, we are taking the example of Forum where **One question can have multiple answers**. But Answer has its own information such as answerId, answer and postedBy. There are four pages used in this example:

1. **Question.java**
2. **Answer.java**
3. **applicationContext.xml**
4. **Test.java**

In this example, we are using list that can have duplicate elements, you may use set that have only unique elements. But, you need to change list to set in the applicationContext.xml file and List to Set in the Question.java file.

**Question.java**

This class contains three properties, two constructors and displayInfo() method that prints the information. Here, we are using List to contain the multiple answers.

**package** com.javatpoint;

**import** java.util.Iterator;

**import** java.util.List;

**public** **class** Question {

**private** **int** id;

**private** String name;

**private** List<Answer> answers;

//setters and getters

**public** **void** displayInfo(){

    System.out.println(id+" "+name);

    System.out.println("answers are:");

    Iterator<Answer> itr=answers.iterator();

**while**(itr.hasNext()){

        System.out.println(itr.next());

    }

}

}

**Answer.java**

This class has three properties id, name and by with constructor and toString() method.

**package** com.javatpoint;

**public** **class** Answer {

**private** **int** id;

**private** String name;

**private** String by;

//setters and getters

**public** String toString(){

**return** id+" "+name+" "+by;

}

}

**applicationContext.xml**

The **ref** element is used to define the reference of another bean. Here, we are using **bean** attribute of **ref** element to specify the reference of another bean.

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

<beans

    xmlns="http://www.springframework.org/schema/beans"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

    xmlns:p="http://www.springframework.org/schema/p"

    xsi:schemaLocation="http://www.springframework.org/schema/beans

http://www.springframework.org/schema/beans/spring-beans-3.0.xsd">

<bean id="answer1" **class**="com.javatpoint.Answer">

<property name="id" value="1"></property>

<property name="name" value="Java is a programming language"></property>

<property name="by" value="Ravi Malik"></property>

</bean>

<bean id="answer2" **class**="com.javatpoint.Answer">

<property name="id" value="2"></property>

<property name="name" value="Java is a platform"></property>

<property name="by" value="Sachin"></property>

</bean>

<bean id="q" **class**="com.javatpoint.Question">

<property name="id" value="1"></property>

<property name="name" value="What is Java?"></property>

<property name="answers">

<list>

<ref bean="answer1"/>

<ref bean="answer2"/>

</list>

</property>

</bean>

</beans>

**Test.java**

This class gets the bean from the applicationContext.xml file and calls the displayInfo method.

**package** com.javatpoint;

**import** org.springframework.beans.factory.BeanFactory;

**import** org.springframework.beans.factory.xml.XmlBeanFactory;

**import** org.springframework.core.io.ClassPathResource;

**import** org.springframework.core.io.Resource;

**public** **class** Test {

**public** **static** **void** main(String[] args) {

    Resource r=**new** ClassPathResource("applicationContext.xml");

    BeanFactory factory=**new** XmlBeanFactory(r);

    Question q=(Question)factory.getBean("q");

    q.displayInfo();

}

}