@Value annotation get us rid of injecting dependencies by using <property> tag.

Instead of injecting dependencies through <property> tag we can simply specify what value needs to be inject using @Value annotation.

Below project let you make use of @Value annotation,

**Student.java :**

public class Student {

private int rollno;

private String name;

private double fee;

@Value("101")

public void setRollno(int rollno) {

this.rollno = rollno;

}

@Value("Sourabh")

public void setName(String name) {

this.name = name;

}

@Value("500")

public void setFee(double fee) {

this.fee = fee;

}

@Override

public String toString() {

return "Student [rollno=" + rollno + ", name=" + name + ", fee=" + fee + "]";

}

}

**config.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"

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

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

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

<context:annotation-config />

d

<bean id="stu" class="com.annotation.value.Student" />

</beans>

**School.java :**

public class School {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("com/annotation/value/config.xml");

Student stu = context.getBean("stu", Student.class);

System.out.println(stu);

}

}

Injecting Dynamic Values Through Properties File Using @Value

This can done with the following steps :

**Step 1 :** Create a .properties file with the following data

student.rollno = 105

student.name = Sourabh

student.fee = 500

**Step 2 :** Add the below tag inside your config.xml file

<context:property-placeholder location="classpath:com/spring/propertiesfileusage/student-info.properties" />

**Step 2 :** Now annotate your setters like below :

**@Value(“${student.rollno}”)**

**@Value(“${student.name}”)**

**@Value(“${student.fee}”)**

Annotating The Data Members

We can annotate our dependency as well. Like this, we can skip creating Setters. So our Student.java file can be like below :

**Student.java**

public class Student {

@Value("${student.rollno}")

private int rollno;

@Value("${student.name}")

private String name;

@Value("${student.fee}")

private double fee;

@Override

public String toString() {

return "Student [rollno=" + rollno + ", name=" + name + ", fee=" + fee + "]";

}

}

The code will run perfectly this time as well.

@Required Annotation

@Required annotation had used for mark a dependency that this dependency should be initialized at any cost else Spring will throw Exception. But it is **deprecated** in **Spring 5.1**.

We can still use it by mentioning a tag for it in our config.xml file. But as it deprecated so will only learn the alternate of @Required annotation.

Before move ahead we have to must clear what actually @Required does.

**So, @Required force to end user that this dependency is very important to initialized. It can’t be uninitialized.**

Now, to learn about the alternate of @Required Annotation. We must know the Core Java rule which is explained below :

Suppose we have a following class :

class MyClass {

private final int id;

public int getId() {

return id;

}

}

class Main {

public static void main(String []args) {

MyClass obj = new MyClass();

System.out.println(obj.getId());

}

}

This code will give us a compile error saying :

error: variable id not initialized in the default constructor

It is because, java force us to initialize final members.

So to remove this error, we have two solution.

**Solution 1 :** Initialize final member at the time of declare it.

**Solution 2 :** Initialize final member in constructor.

We don’t want to hard code so we will go with Solution 2.

class MyClass {

private final int id;

public MyClass(int id) {

this.id = id;

}

public int getId() {

return id;

}

}

class Main {

public static void main(String []args) {

MyClass obj = new MyClass(105);

System.out.println(obj.getId());

}

}

Now the above code will work fine.

So, using this concept spring come up with the alternate of @Required annotation.

No one can create object of the class without initialize the “final” members.

To make a dependency Required Dependency. We should do the following thing :

* Make that dependency as **final**.
* Initialize it through constructor.

**What spring say in terms of alternate for a @Required annotation ?**

You should use constructor injection for **REQUIRED** dependencies and setter injections for **OPTIONAL** dependencies instead of field injection. Some reasons why:

* It makes it clear to everybody which dependencies are required
* It makes testing easier
* You can make your objects immutable

Here is the demo project for understanding :

**student-info.properties :**

student.rollno = 105

student.name = Sourabh

student.fee = 500

**Student.java :**

public class Student {

private final int rollno;

private String name;

private double fee;

public Student(int rollno) {

this.rollno = rollno;

}

public void setName(String name) {

this.name = name;

}

public void setFee(double fee) {

this.fee = fee;

}

@Override

public String toString() {

return "Student [rollno=" + rollno + ", name=" + name + ", fee=" + fee + "]";

}

}

**config.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>

xmlns:context=<http://www.springframework.org/schema/context>

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

http://www.springframework.org/schema/context [http://www.springframework.org/schema/context/spring- context.xsd](http://www.springframework.org/schema/context/spring-%20context.xsd)">

<context:property-placeholder location="classpath:com/annotation/required/student-info.properties" />

<constructor-arg name="rollno" value="${student.rollno}" type="int" />

<property name="name" value="${student.name}" />

<property name="fee" value="${student.fee}" />

</bean>

</beans>

**School.java :**

public class School {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("com/annotation/required/config.xml");

Student stu = context.getBean("stu", Student.class);

System.out.println(stu);

}

}