Spring @Qualifier Annotation with Example

Last Updated : 05 Aug, 2025

The @Qualifier annotation is used to resolve ambiguity when multiple beans of the same type are available for dependency injection. It helps specify which exact bean should be injected into a class when Spring cannot automatically determine the correct one.

Key features of @Qualifier Annotation:

- @Qualifier annotation is used with @Autowired to resolve ambiguity when multiple beans of the same type are present in the Spring container.
- It helps Spring determine which specific bean to inject by specifying the bean ID (name).
- @Qualifier can be used on Setter methods, Fields and Constructor parameters.
- Without @Qualifier, if more than one matching bean exists, Spring throws a NoUniqueBeanDefinitionException.
- Using @Qualifier("beanName") ensures that the correct and intended bean is injected during autowiring.

Working of @Qualifier Annotation

The @Qualifier annotation is used with @Autowired to specify the exact bean to be injected. We provide the bean name (or ID) as a parameter to the @Qualifier annotation.

Example: This example demonstrates how to use @Autowired with @Qualifier to inject a specific bean when multiple beans of the same type exist.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

public class Human {
    private Heart heart;

    @Autowired
    @Qualifier("humanHeart")
    public void setHeart(Heart heart) {
        this.heart = heart;
    }

    public void startPumping() {
        heart.pump();
    }
}
```

Problem with @Autowired Annotation

To understand the need for the @Qualifier annotation, let's first look at a scenario where the @Autowired annotation can cause issues.

Consider a Human class that depends on a Heart class. The Heart class has a simple method called pump()

1. Heart.java

```
public class Heart {
    public void pump() {
        System.out.println("Heart is Pumping");
    }
}
```

2. Human.java

Human class does has a dependency on another class named Heart.

```
import org.springframework.beans.factory.annotation.Autowired;

public class Human {
    private Heart heart;

    @Autowired
    public void setHeart(Heart heart) {
        this.heart = heart;
```

```
    }

    public void startPumping() {
        heart.pump();
    }
}
```

Now we want to inject the object of the Heart class inside the Human class by using the @Autowired annotation. So for this, we can write the code something like this

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

public class Human {

    private Heart heart;

    @Autowired public void setHeart(Heart heart)
    {
        this.heart = heart;
    }

    public void startPumping() {
        heart.pump();
    }
}
```

3. beans.xml Configuration

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

    <context:annotation-config/>
    <bean id="heartObjValue" class="Heart"></bean>
    <bean id="humanObject" class="Human"></bean>
</beans>
```

4. Main.java

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

public class Main {
    public static void main(String[] args) {
        ApplicationContext context = new
ClassPathXmlApplicationContext("beans.xml");
        Human human = context.getBean("humanObject", Human.class);
        human.startPumping();
    }
}
```

Output:

Heart is Pumping

Explanation: In the above code, the @Autowired annotation is used to autowire the Heart object into the Human class by using the Spring Autowiring 'byType'. When the byType mechanism finds a suitable match, it injects the Heart object without issues.

Note:

How @Autowired Work?

- *It first tries to resolve dependencies by type.*
- *If multiple beans of the same type are found, it tries to resolve by name (matching the bean name to the property name).*

So in our above example, the "byType" Spring Autowiring mechanism happens and the heart object is injected into the Human class. But where does the problem lie?

The Problem: Multiple Beans of the Same Types

Now inside the beans.xml file let's create another bean of the Heart class.

1. Modified bean.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"
        xmlns:context="http://www.springframework.org/schema/context/"
        xsi:schemaLocation="http://www.springframework.org/schema/beans/
         https://www.springframework.org/schema/beans/spring-beans.xsd
         http://www.springframework.org/schema/context/
         https://www.springframework.org/schema/context/spring-context.xsd">

    <context:annotation-config/>

    <bean id="humanHeart" class="Heart"></bean>
    <bean id="octopusHeart" class="Heart"></bean>

    <bean id="humanObject" class="Human"></bean>

</beans>
```

Here, we have two beans of the same class: one is "humanHeart" and another one is "octpusHeart". In this scenario, the "byType" autowiring will fail because there are two beans of the same type (Heart). Spring will then attempt to resolve by name, but since the property name "heart" doesn't match either bean id, this also fails.

When we run your application, you'll get the following exception:

> *Exception in thread "main" org.springframework.beans.factory.UnsatisfiedDependencyException : Error creating bean with name 'humanObject': Unsatisfied dependency expressed through method 'setHeart' parameter.....*

Solution: Using @Qualifier Annotation

When multiple beans of the same type exist, @Qualifier helps specify which one to inject. For example, in the Human class, we can use @Qualifier("humanHeart") to avoid confusion and ensure the correct bean is wired.

Modified Human.java

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

public class Human {

    private Heart heart;
```

```
    @Autowired
    @Qualifier("humanHeart")

    public void setHeart(Heart heart){
        this.heart = heart;
    }

    public void startPumping() {
        heart.pump();
    }
}
```

Output:

Heart is Pumping

Alternative: Using @Qualifier on Field Injection

We can also use the @Qualifier annotation directly on the field, eliminating the need for a setter method.

Example: This example demonstrates how to use @Autowired and @Qualifier to inject a specific bean (humanHeart) into a field.

```
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;

public class Human {
    @Autowired
    @Qualifier("humanHeart")
    private Heart heart;

    public void startPumping() {
        heart.pump();
    }
}
```