DAY-4 [Part-I]

ANNOTATIONS

A form of metadata that provides supplemental info about a program.

Let’s see an example to understand this concept→

```
package in.sp.beans;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component  // Registering Student as a Spring Bean
public class Student {

    private int roll_no;
    private String name;

    @Autowired  // Injecting Address Bean
    @Qualifier("address")  // If multiple Address beans exist, specify the exact one
    private Address address;

    // Setter methods (Spring will use these if needed)
    public void setRollno(int rollno) {
        this.roll_no = rollno;
    }

    public void setName(String name) {
        this.name = name;
    }
```

```
    public void setAddress(Address address) {
        this.address = address;
    }

    // Method to print attributes
    public void display() {
        System.out.println("Roll_no: " + roll_no);
        System.out.println("Name: " + name);
        System.out.println("Address: " + address);
    }
}
```

```
package in.sp.beans;

import org.springframework.stereotype.Component;

@Component("address")  // Registering Address as a Spring Bean
public class Address {

    private int houseno;
    private String city;
    private int pincode;

    public void setHouseno(int houseno) {
        this.houseno = houseno;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public void setPincode(int pincode) {
        this.pincode = pincode;
    }
```

```
    @Override
    public String toString() {
        return "#" + houseno + ", " + city + " -" + pincode;
    }
}
```

1. `@Component` (for Bean Creation)

- Use `@Component` on both `Student` and `Address` classes to register them as Spring Beans.

2. `@Autowired` (for Dependency Injection)

- Use `@Autowired` in the `Student` class to automatically inject the `Address` bean.

3. `@Qualifier` (to Specify a Particular Bean)

- If there are multiple `Address` beans, you can use `@Qualifier` to specify which one to inject.

Let's modify your code to handle multiple `Address` beans and see how `@Qualifier` comes into play.

Changes Made:

1. **Created multiple `Address` beans (`@Component("homeAddress")` and `@Component("officeAddress")`).**
2. **Modified `Student` class to inject a specific `Address` using `@Qualifier`.**
3. **Demonstrated how `@Qualifier` helps in selecting the right bean.**

Updated Code:

`Student.java` (Injecting a Specific Address)

```
java
CopyEdit
package in.sp.beans;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component // Mark Student as a Spring Bean
public class Student {

    private int roll_no;
    private String name;

    @Autowired  // Injecting Address Bean
    @Qualifier("homeAddress")  // Specify which Address to inject
    private Address address;

    // Setter methods
    public void setRollno(int rollno) {
        this.roll_no = rollno;
    }

    public void setName(String name) {
        this.name = name;
    }

    public void setAddress(Address address) {
        this.address = address;
    }

    // Method to print attributes
    public void display() {
        System.out.println("Roll_no: " + roll_no);
        System.out.println("Name: " + name);
```

```
        System.out.println("Address: " + address);
    }
}
```

> Here’s how it works:

> **Spring scans for a matching bean of type `Address`.**

> Since both `HomeAddress` and `OfficeAddress` extend `Address`, Spring finds **multiple candidates**.

> If **no `@Qualifier` is used**, Spring **throws an error** because it doesn't know which one to inject.

> When you use `@Qualifier("homeAddress")`, Spring **only injects `HomeAddress`**, ignoring `OfficeAddress`.

> Similarly, if you use `@Qualifier("officeAddress")`, Spring injects `OfficeAddress` instead.

> So, **only one child class gets injected**, depending on the `@Qualifier` used.

`Address.java` (Base Class)

```
java
CopyEdit
package in.sp.beans;

public class Address {
    private int houseno;
    private String city;
    private int pincode;
```

```
    public void setHouseno(int houseno) {
        this.houseno = houseno;
    }

    public void setCity(String city) {
        this.city = city;
    }

    public void setPincode(int pincode) {
        this.pincode = pincode;
    }

    @Override
    public String toString() {
        return "#" + houseno + ", " + city + " -" + pincode;
    }
}
```

💡

The reason
why `@Component` was not used in `Address.java` in this example is that **`Address` is an abstract/generalized parent class** that is **not meant to be directly instantiated as a Spring bean**. Instead, its child classes (`HomeAddress` and `OfficeAddress`) are the ones registered as Spring beans using `@Component("homeAddress")` and `@Component("officeAddress")`.

Why Not Use `@Component` in `Address.java`?

1. `Address` **is a general base class** (it just holds common properties).
   - If we mark `Address` with `@Component`, Spring will create a bean for it, which is unnecessary because we always want to work with either `HomeAddress` or `OfficeAddress`, not a generic `Address` object.
2. **If `@Component` were used in `Address.java`, it would create ambiguity.**
   - Spring would register an extra `Address` bean alongside `homeAddress` and `officeAddress`.
   - When we use `@Autowired`, Spring might get confused about which bean to inject (since we would have three beans of type `Address` instead of two).
3. **Using `@Component` in `HomeAddress` and `OfficeAddress` is enough**
   - These two child classes are the actual implementations, so only they should be instantiated as beans.

`HomeAddress.java` (One Address Bean)

java
CopyEdit
package in.sp.beans;

```
import org.springframework.stereotype.Component;

@Component("homeAddress") // Register Home Address as a Bean
public class HomeAddress extends Address {
    public HomeAddress() {
        setHouseno(101);
        setCity("Bangalore");
        setPincode(560001);
    }
}
```

`OfficeAddress.java` (Another Address Bean)

```
java
CopyEdit
package in.sp.beans;

import org.springframework.stereotype.Component;

@Component("officeAddress") // Register Office Address as a Bean
public class OfficeAddress extends Address {
    public OfficeAddress() {
        setHouseno(202);
        setCity("Mumbai");
        setPincode(400001);
    }
}
```

How `@Qualifier` Comes into Picture

- Since we have **two** `Address` **beans** (`homeAddress` and `officeAddress`), Spring doesn't know which one to inject.
- We use `@Qualifier("homeAddress")` inside `Student.java` to tell Spring **which** `Address` **bean to inject**.

- If we wanted the **office address instead**, we'd replace:with

```
java
CopyEdit
@Qualifier("homeAddress")
```

```
java
CopyEdit
@Qualifier("officeAddress")
```

How to Use This in Spring Boot?

1. **Enable Component Scanning:** Ensure your main application scans the package.
2. **Run the Application and Fetch the Bean:**

```
java
CopyEdit
ApplicationContext context = new AnnotationConfigApplicationContext("in.sp.beans");
Student student = context.getBean(Student.class);
student.display();
```

Expected Output (With `homeAddress` Chosen)

```
yaml
CopyEdit
Roll_no: 0
Name: null
Address: #101, Bangalore -560001
```

If you switch `@Qualifier("officeAddress")`, the output changes to:

```
yaml
CopyEdit
Roll_no: 0
Name: null
Address: #202, Mumbai -400001
```

This is how `@Qualifier` helps resolve conflicts when multiple beans of the same type exist.

Q. suppose there were 2-3 student class who are depending upon the address class like in the above code i provided. so now how will these three annotations work?

Scenario:

- We have **three different** `Student` **classes** (`EngineeringStudent`, `MedicalStudent`, and `ArtsStudent`).
- Each student needs an **Address**, but different students may have different addresses.
- We use `@Autowired` **to inject Address** and `@Qualifier` **to specify which Address to inject**.

Updated Code:

`Address.java` (Parent Class)

```
java
CopyEdit
package in.sp.beans;

public class Address {
    private int houseno;
```

```
    private String city;
    private int pincode;

    public void setHouseno(int houseno) { this.houseno = houseno; }
    public void setCity(String city) { this.city = city; }
    public void setPincode(int pincode) { this.pincode = pincode; }

    @Override
    public String toString() {
        return "#" + houseno + ", " + city + " -" + pincode;
    }
}
```

HomeAddress.java (First Child Bean)

```
java
CopyEdit
package in.sp.beans;

import org.springframework.stereotype.Component;

@Component("homeAddress")  // Register Home Address as a Bean
public class HomeAddress extends Address {
    public HomeAddress() {
        setHouseno(101);
        setCity("Delhi");
        setPincode(110001);
    }
}
```

HostelAddress.java (Second Child Bean)

```
java
CopyEdit
```

```
package in.sp.beans;

import org.springframework.stereotype.Component;

@Component("hostelAddress")  // Register Hostel Address as a Bean
public class HostelAddress extends Address {
    public HostelAddress() {
        setHouseno(302);
        setCity("Bangalore");
        setPincode(560001);
    }
}
```

OfficeAddress.java (Third Child Bean)

```
java
CopyEdit
package in.sp.beans;

import org.springframework.stereotype.Component;

@Component("officeAddress")  // Register Office Address as a Bean
public class OfficeAddress extends Address {
    public OfficeAddress() {
        setHouseno(505);
        setCity("Mumbai");
        setPincode(400001);
    }
}
```

Multiple Student Classes Depending on Address

EngineeringStudent.java

```
java
CopyEdit
package in.sp.beans;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component  // Register EngineeringStudent as a Spring Bean
public class EngineeringStudent {

    private String name = "John Doe";

    @Autowired
    @Qualifier("homeAddress")  // Engineering students live at home
    private Address address;

    public void display() {
        System.out.println("Engineering Student: " + name);
        System.out.println("Address: " + address);
    }
}
```

MedicalStudent.java

```
java
CopyEdit
package in.sp.beans;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component  // Register MedicalStudent as a Spring Bean
```

```
public class MedicalStudent {

    private String name = "Alice Smith";

    @Autowired
    @Qualifier("hostelAddress")  // Medical students live in a hostel
    private Address address;

    public void display() {
        System.out.println("Medical Student: " + name);
        System.out.println("Address: " + address);
    }
}
```

ArtsStudent.java

```
java
CopyEdit
package in.sp.beans;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component  // Register ArtsStudent as a Spring Bean
public class ArtsStudent {

    private String name = "Bob Williams";

    @Autowired
    @Qualifier("officeAddress")  // Arts students live near an office area
    private Address address;

    public void display() {
        System.out.println("Arts Student: " + name);
```

```
        System.out.println("Address: " + address);
    }
}
```

How to Use This in Spring Boot

```
java
CopyEdit
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationC
ontext;

public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new AnnotationConfigApplicationContext("i
n.sp.beans");

        EngineeringStudent engStudent = context.getBean(EngineeringStudent.c
lass);
        engStudent.display();

        MedicalStudent medStudent = context.getBean(MedicalStudent.class);
        medStudent.display();

        ArtsStudent artsStudent = context.getBean(ArtsStudent.class);
        artsStudent.display();
    }
}
```

Expected Output

```
yaml
CopyEdit
Engineering Student: John Doe
```

```
Address: #101, Delhi -110001

Medical Student: Alice Smith
Address: #302, Bangalore -560001

Arts Student: Bob Williams
Address: #505, Mumbai -400001
```

Example: Using an Interface with Multiple Implementations

Scenario

- We create an `Address` **interface**.
- We have **two implementations**: `HomeAddress` and `OfficeAddress`.
- Multiple `Student` classes depend on `Address`, and we use `@Qualifier` to specify the implementation.

Step 1: Create an Interface

`Address.java` (Interface)

```
java
CopyEdit
package in.sp.beans;

public interface Address {
    String getAddressDetails();
}
```

Step 2: Implement Multiple Address Classes

HomeAddress.java

```
java
CopyEdit
package in.sp.beans;

import org.springframework.stereotype.Component;

@Component("homeAddress")  // Register as a Spring Bean
public class HomeAddress implements Address {
    @Override
    public String getAddressDetails() {
        return "Home Address: #101, Delhi - 110001";
    }
}
```

OfficeAddress.java

```
java
CopyEdit
package in.sp.beans;

import org.springframework.stereotype.Component;

@Component("officeAddress")  // Register as a Spring Bean
public class OfficeAddress implements Address {
    @Override
    public String getAddressDetails() {
        return "Office Address: #505, Mumbai - 400001";
    }
}
```

Step 3: Inject the Interface into Student Class

EngineeringStudent.java

```
java
CopyEdit
package in.sp.beans;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component
public class EngineeringStudent {
    private String name = "John Doe";

    @Autowired
    @Qualifier("homeAddress")  // Explicitly specify HomeAddress
    private Address address;

    public void display() {
        System.out.println("Engineering Student: " + name);
        System.out.println(address.getAddressDetails());
    }
}
```

MedicalStudent.java

```
java
CopyEdit
package in.sp.beans;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.stereotype.Component;

@Component
```

```
public class MedicalStudent {
    private String name = "Alice Smith";

    @Autowired
    @Qualifier("officeAddress")  // Explicitly specify OfficeAddress
    private Address address;

    public void display() {
        System.out.println("Medical Student: " + name);
        System.out.println(address.getAddressDetails());
    }
}
```

Step 4: Spring Boot Main Class

```
java
CopyEdit
import org.springframework.context.ApplicationContext;
import org.springframework.context.annotation.AnnotationConfigApplicationC
ontext;

public class MainApp {
    public static void main(String[] args) {
        ApplicationContext context = new AnnotationConfigApplicationContext("i
n.sp.beans");

        EngineeringStudent engStudent = context.getBean(EngineeringStudent.c
lass);
        engStudent.display();

        MedicalStudent medStudent = context.getBean(MedicalStudent.class);
        medStudent.display();
    }
}
```

Expected Output

```
yaml
CopyEdit
Engineering Student: John Doe
Home Address: #101, Delhi - 110001

Medical Student: Alice Smith
Office Address: #505, Mumbai - 400001
```

🔍 Key Differences Between XML and Annotation-Based Configuration

Feature	Annotation-Based	XML-Based
Bean Definition	`@Component`	`<bean>`
Dependency Injection	`@Autowired`	`<property>`
Bean Selection	`@Qualifier("beanName")`	`ref="beanName"`
Configuration File	No XML required (uses `@ComponentScan`)	Uses `spring-config.xml`
Readability	Less configuration, easier to read	More verbose

💡 **Generally, annotation-based configuration (`@Component`, `@Autowired`, `@Qualifier`) is preferred** because it's cleaner and reduces XML complexity. 🚀