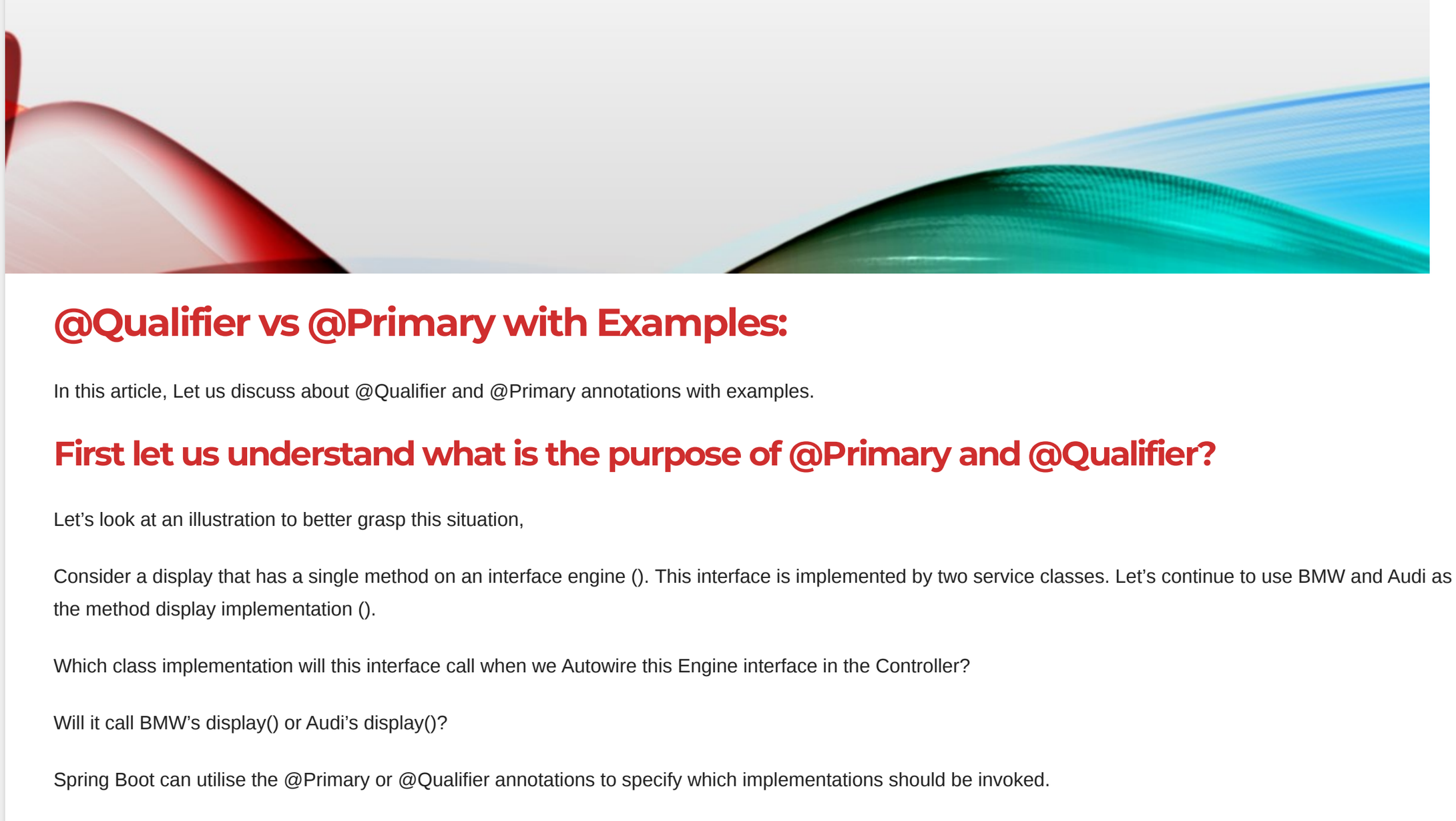
SPRING BOOT

@Qualifier vs @Primary with Examples

By Sri

JUL 11, 2022 @Primary, @Qualifier, @Qualifier and @Primary, @Qualifier and @Primary in Springboot with example, @Qualifier and @primary with example, @Qualifier vs @Primary, @Qualifier vs @Primary in springboot with example, primary vs qualifier, primary vs qualifier with example



@Qualifier vs @Primary with Examples:

In this article, Let us discuss about @Qualifier and @Primary annotations with examples.

First let us understand what is the purpose of @Primary and @Qualifier?

Let's look at an illustration to better grasp this situation,

Consider a display that has a single method on an interface engine (). This interface is implemented by two service classes. Let's continue to use BMW and Audi as the providers of the method display implementation ().

Which class implementation will this interface call when we Autowire this Engine interface in the Controller?

Will it call BMW's display() or Audi's display()?

Spring Boot can utilise the @Primary or @Qualifier annotations to specify which implementations should be invoked.

@Primary:

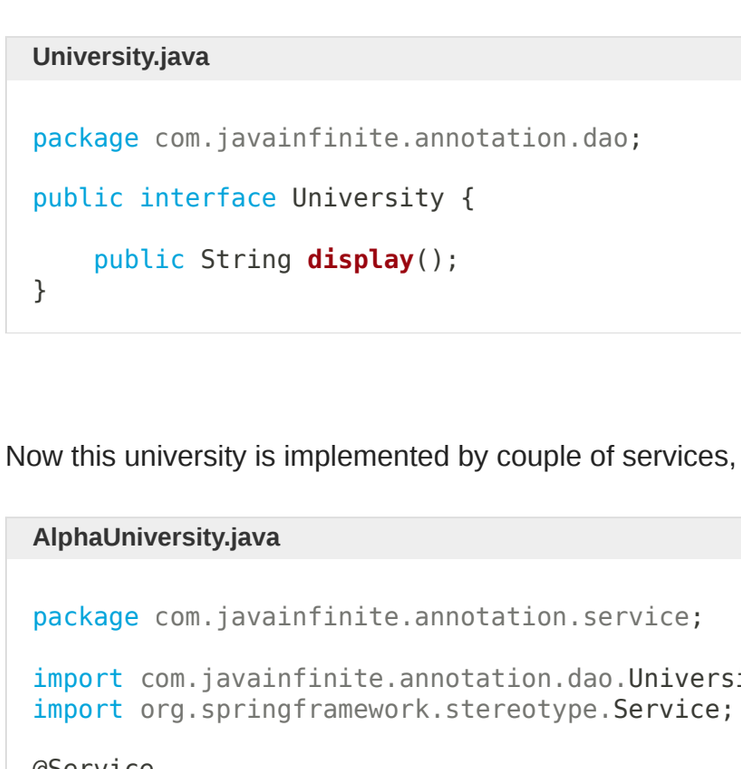
The **@Primary annotation indicates which bean should be given precedence** when more than one bean meets the requirements for the dependency to be autowired. There should only be one principal bean among the acceptable beans.

@Qualifier:

We may use a **@qualifier annotation to clearly indicate the bean name** and tell spring which dependency should be called for when more than one bean satisfies the conditions for the dependency.

Let us try to understand in a more simpler way with an example,

Project Structure:



Let us create an interface University,

University.java

```
package com.javainfinite.annotation.dao;

public interface University {

    public String display();
}
```

University.java

Now this university is implemented by couple of services, Alpha University and Beta University,

AlphaUniversity.java

```
package com.javainfinite.annotation.service;

import com.javainfinite.annotation.dao.University;
import org.springframework.stereotype.Service;

@Service
public class AlphaUniversity implements University {

    @Override
    public String display() {

        return "This is a message from Alpha University";
    }
}
```

AlphaUniversity.java

BetaUniversity

```
package com.javainfinite.annotation.service;

import com.javainfinite.annotation.dao.University;
import org.springframework.stereotype.Service;

@Service
public class BetaUniversity implements University {

    @Override
    public String display() {

        return "This is a message from Beta University";
    }
}
```

BetaUniversity.java

Now let us have a controller,

UniversityController.java

```
package com.javainfinite.annotation.controller;

import com.javainfinite.annotation.dao.University;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UniversityController {

    @Autowired
    private University AlphaUniversity;

    @GetMapping ("/university")
    public String getUniversity() {

        return AlphaUniversity.display();
    }
}
```

UniversityController.java

What have we done here?

We have created an interface university and that has been implemented by AlphaUniversity and BetaUniversity.

We have autowired that interface in our UniversityController class.

Now when we call "/univeristy" API which dependency will be actually called? will it be AlphaUniversity or BetaUniversity?

Let us try to run this code and understand the behaviour,

Error

We will get an error like above mentioned to use either Primary or Qualifier annotation.

Springboot is unsure which dependency to use because both AlphaUniversity and BetaUniversity have implemented University.

Now let us use @Primary annotation for resolving this issue,

Now I am going to mark BetaUniversity with @Primary annotation,

BetaUniversity.java

```
package com.javainfinite.annotation.service;

import com.javainfinite.annotation.dao.University;
import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Service;

@Service
@Primary
public class BetaUniversity implements University {

    @Override
    public String display() {

        return "This is a message from Beta University";
    }
}
```

BetaUniversity.java

Now we have included @Primary annotation, Now springboot will know which bean should be given preference when we autowire University,

let us try to call the API endpoint,

Spring boot has considered BetaUniversity as a preferred bean when University is Autowired.

Now let us use @Qualifier annotation to differentiate beans,

AlphaUniversity.java

```
package com.javainfinite.annotation.service;

import com.javainfinite.annotation.dao.University;
import org.springframework.stereotype.Service;

@Service("Alpha")
public class AlphaUniversity implements University {

    @Override
    public String display() {

        return "This is a message from Alpha University";
    }
}
```

AlphaUniversity.java

BetaUniversity.java

```
package com.javainfinite.annotation.service;

import com.javainfinite.annotation.dao.University;
import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Service;

@Service("Beta")
@Primary
public class BetaUniversity implements University {

    @Override
    public String display() {

        return "This is a message from Beta University";
    }
}
```

BetaUniveristy.java

UniversityController.java

```
package com.javainfinite.annotation.controller;

import com.javainfinite.annotation.dao.University;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UniversityController {

    @Autowired
    @Qualifier("Alpha")
    private University AlphaUniversity;

    @GetMapping ("/university")
    public String getUniversity() {

        return AlphaUniversity.display();
    }
}
```

UniversityController.java

On calling the API end point – "/university",

Which is of higher priority @Primary or @Qualifier?

@Qualifier has higher priority than @Primary annotation. In the above example we have used both @Primary and @Qualifier together – Qualifier has taken higher priority than @Primary.

Can we declare both @Primary and @Qualifier together?

Yes, Both annotations can be used together but only one can be executed at a time. (Qualifier first, if qualifier not present it will execute primary)

Complete Code:

pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <parent>
        <groupId>org.springframework.boot</groupId>
        <artifactId>spring-boot-starter-parent</artifactId>
        <version>2.7.0</version>
        <relativePath/> <!-- lookup parent from repository -->
    </parent>
    <groupId>com.javainfinite</groupId>
    <artifactId>annotation</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <name>annotation</name>
    <description>Qualifier vs Primary</description>
    <properties>
        <java.version>17</java.version>
    </properties>
    <dependencies>
        <dependency>
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-web</artifactId>
        </dependency>
    </dependencies>

    <build>
        <plugins>
            <plugin>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-maven-plugin</artifactId>
            </plugin>
        </plugins>
    </build>

</project>
```

pom.xml

UniversityController.java

UniversityController.java

```
package com.javainfinite.annotation.controller;

import com.javainfinite.annotation.dao.University;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UniversityController {

    @Autowired
    @Qualifier("Alpha")
    private University AlphaUniversity;

    @GetMapping ("/university")
    public String getUniversity() {

        return AlphaUniversity.display();
    }
}
```

University.java

Univeristy.java

```
package com.javainfinite.annotation.dao;

public interface University {

    public String display();
}
```

AlphaUniversity.java

AlphaUniversity.java

```
package com.javainfinite.annotation.service;

import com.javainfinite.annotation.dao.University;
import org.springframework.stereotype.Service;

@Service("Alpha")
public class AlphaUniversity implements University {

    @Override
    public String display() {

        return "This is a message from Alpha University";
    }
}
```

BetaUniversity.java

BetaUniversity.java

```
package com.javainfinite.annotation.service;

import com.javainfinite.annotation.dao.University;
import org.springframework.context.annotation.Primary;
import org.springframework.stereotype.Service;

@Service("Beta")
@Primary
public class BetaUniversity implements University {

    @Override
    public String display() {

        return "This is a message from Beta University";
    }
}
```

AnnotationApplication.java

AnnotationApplication.java

```
package com.javainfinite.annotation;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
public class AnnotationApplication {

    public static void main(String[] args) {
        SpringApplication.run(AnnotationApplication.class, args);
    }

}
```

Complete code here

« Custom Actuator Endpoint – Spring Boot actuators with Example

Spring Boot Security with JWT Example – Token Generate, Validate and Refresh »

By Sri

RELATED POST

SPRING BOOT Configuring Multiple Data Sources with Spring Boot – With Example FEB 15, 2023 SRI

SPRING BOOT ChatGPT with Spring Boot: Improving English Grammar FEB 14, 2023 SRI

SPRING BOOT Streamlining Language Translation with Spring Boot and FEB 13, 2023 SRI

Leave a Reply

SEARCH

Search

FOLLOW ME @

Facebook

Twitter

YouTube

RECENT POSTS

Configuring Multiple Data Sources with Spring Boot – With Example

ChatGPT with Spring Boot: Improving English Grammar with AI

Streamlining Language Translation with Spring Boot and ChatGPT API Integration

Mastering Method References: A Deep Dive into Java 8

Integrating Swagger 3 (OpenAPI Specification) with Spring Boot: A Comprehensive Guide

YOU MISSED

SPRING BOOT Configuring Multiple Data Sources with Spring Boot – With FEB 15, 2023 SRI

SPRING BOOT ChatGPT with Spring Boot: Improving English Grammar with FEB 14, 2023 SRI

SPRING BOOT Streamlining Language Translation with Spring Boot and FEB 13, 2023 SRI

JAVA Mastering Method References: A Deep Dive into Java 8 FEB 8, 2023 SRI