Dependency Injection

Using Annotation & AutoWiring

What is Spring Autowiring?

- Spring will look for a class that matches the property
  - Matches by type : class or interface
- Spring will inject in AUTOMATICALLY… hence it is autowiring

Autowiring Injection Types

- Constructor Injection
- Setter Injection
- Field Injection

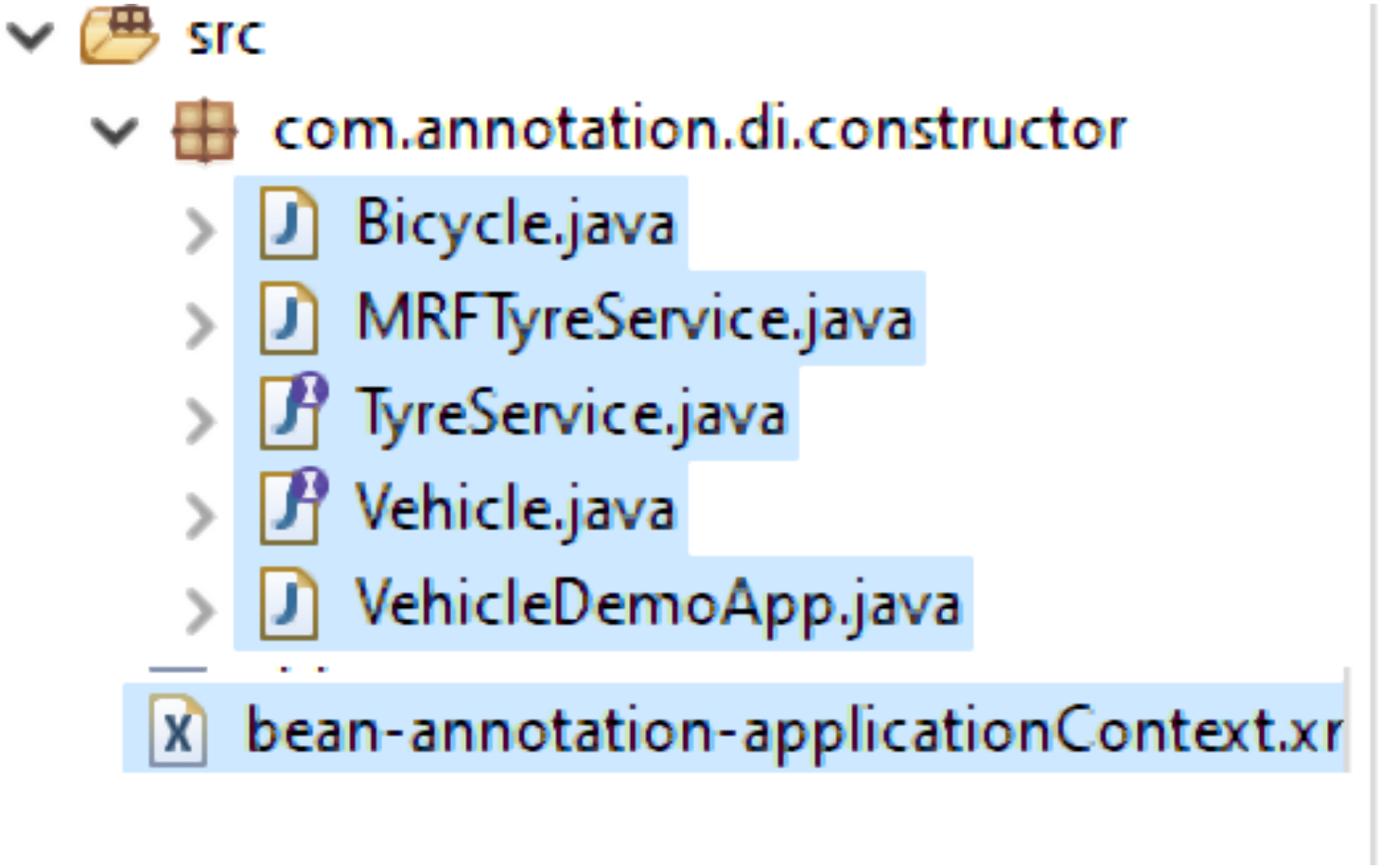
Note: @Autowired annotation used for autowiring

DI using Constructor

Development Process

- Create a new package com.annotation.di.constructor
- Create Dependency class & Interface
- Add an interface and a Class with Constructor for Dependency Injection
  - Annotate class with @Component and the constructor with @Autowired
- Add content.xml with <context:component-scan…>
- Add the main class and invoke the methods in the object and from dependency object.

Development Process



1. Add TyreService interface

TyreService.java

```java
package com.annotation.di.constructor;

public interface TyreService {
    public String getTyre();
}

```

2. Add MRFTyreService class

MRFTyreService.java

```java
package com.annotation.di.constructor;

import org.springframework.stereotype.Component;

@Component
public class MRFTyreService implements TyreService {
    @Override
    public String getTyre() {
        return "In my vehicle im using MRF Tyre";
    }
}
```

3. Add Vehicle interface

Vehicle.java

```java
package com.annotation.di.constructor;

public interface Vehicle {
    public String getVehicle();
    public String getVehicleTyre();
}

```

4. Add Bicycle class

Bicycle.java

```java
1  package com.annotation.di.constructor;
2  import org.springframework.beans.factory.annotation.Autowired;
3  import org.springframework.stereotype.Component;
4
5  @Component
6  public class Bicycle implements Vehicle{
7
8      public TyreService tyreService;
9
10     @Autowired
11     public Bicycle(TyreService tyreService) {
12         this.tyreService = tyreService;
13     }
14     public String getVehicle() {
15         return "Hi, Im using a Bicycle !!!";
16     }
17     public String getVehicleTyre() {
18         return tyreService.getTyre();
19     }
20 }
```

5. Add config file (...context.xml)

bean-annotation-applicationContext.xml

```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:component-scan base-package="com.annotation.di.constructor" />

</beans>
```

6. Add Main class

VehicleDemoApp.java

```java
package com.annotation.di.constructor;
import org.springframework.context.support.ClassPathXmlApplicationContext;

public class VehicleDemoApp {
    public static void main(String[] args) {

        ClassPathXmlApplicationContext context = new
                ClassPathXmlApplicationContext("bean-annotation-applicatic

        Vehicle theVehicle = context.getBean("bicycle", Vehicle.class);
        System.out.println(theVehicle.getVehicle());
        System.out.println(theVehicle.getVehicleTyre());

        context.close();
    }
}
```

Output

Problems @ Javadoc Declaration Console Terminal

<terminated> VehicleDemoApp (2) [Java Application] E:\Softwares\sts-4.12.1.RELEASE\

```
Hi, Im using a Bicycle !!!
In my vehicle im using MRF Tyre
```

DI using setter method

Replace the constructor with setter method

Bicycle.java

```
5  @Component
6  public class Bicycle implements Vehicle{
7
8      public TyreService tyreService;
9
10 //   @Autowired
11 //   public Bicycle(TyreService tyreService) {
12 //       this.tyreService = tyreService;
13 //   }
14
15     @Autowired
16     public void setTyreService(TyreService tyreService) {
17         this.tyreService = tyreService;
18     }
19
20     public String getVehicle() {
```

DI using any method

With @Autowired

Replace the setter with custom method name

Bicycle.java

```
7  public class Bicycle implements Vehicle{
8
9      public TyreService tyreService;
10
11 //   @Autowired
12 //   public void setTyreService(TyreService tyreService) {
13 //       this.tyreService = tyreService;
14 //   }
15
16     @Autowired
17     public void myCustomMethod(TyreService tyreService) {
18         this.tyreService = tyreService;
19     }
20     public String getVehicle() {
21         return "Hi, Im using a Bicycle !!!";
22     }
23     public String getVehicleTyre() {
24         return tyreService.getTyre();
25     }
```

DI using field

With @Autowired (even private fields)

Just add @Autowired with field

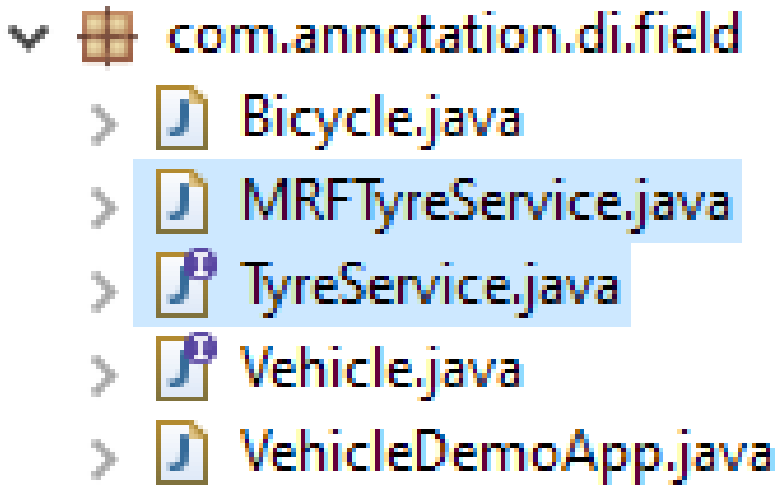
```
1  package com.annotation.di.field;
2  import org.springframework.beans.factory.annotation.Autowired;
3  import org.springframework.stereotype.Component;
4
5
6  @Component
7  public class Bicycle implements Vehicle{
8
9      @Autowired
10     public TyreService tyreService;
11
12     public String getVehicle() {
13         return "Hi, Im using a Bicycle !!!";
14     }
15     public String getVehicleTyre() {
16         return tyreService.getTyre();
17     }
18 }
```

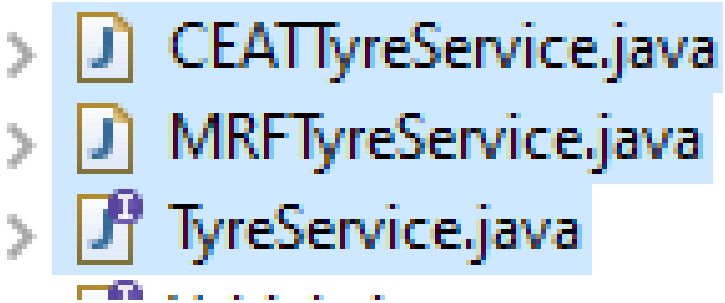
Autowiring & Qualifiers

@Qualifier

While injecting dependency, If the interface is implemented by only one class. Then, it works perfectly !!!

```
@Autowired
public TyreService tyreService;
```





What happens if we have multiple dependency classes implementing the same interface?

It throws NoUniqueBeanDefinitionException

@Qualifier(“bean-id”)

- While creating bean class, we can set custom bean-id with @Component annotation.
- If we fail to have a custom bean id, then spring will make the class name as bean-id by having the first character in lower case.

```
//@Component("myVehicle")

@Component
public class Van implements Vehicle {
```

- Now in this example, we had Custom bean id “myVehicle”, which is comment and then Spring set “van” as bean-id

Class names and bean-id's

- **CLASS NAME -> BEAN NAME (by Spring)**

- AppolloTyreService -> appolloTyreService

If the second character also in the class name is in uppercase. Then the same classname will be considered as bean name.

- MRFTyreService -> MRFTyreService
- CEATTyreService -> CEATTyreService

Injection Types

- @Qualifier can be applied with
  - Constructor Injection
  - Setter Injection
  - Filed Injection

@Qualifier with Field Injection

Bicycle.java

```java
1  package com.annotation.di.qualifier;
2  import org.springframework.beans.factory.annotation.Autowired;
5
6  @Component
7  public class Bicycle implements Vehicle{
8
9      @Autowired
10     @Qualifier("appolloTyreService")
11     public TyreService tyreService;
12
13     public String getVehicle() {
14         return "Hi, Im using a Bicycle !!!";
15     }
16     public String getVehicleTyre() {
17         return tyreService.getTyre();
18     }
19 }
```

com.annotation.di.qualifier

- AppolloTyreService.java
- Bicycle.java
- CEATTyreService.java
- MRFTyreService.java
- TyreService.java
- Vehicle.java
- VehicleDemoApp.java

@Qualifier with Constructor Injection

Bicycle.java

```java
1  package com.annotation.di.qualifier;
2  import org.springframework.beans.factory.annotation.Autowire
5
6  @Component
7  public class Bicycle implements Vehicle{
8
9      public TyreService tyreService;
10
11     @Autowired
12     public Bicycle(@Qualifier("MRFTyreService") TyreService tyreService) {
13         this.tyreService = tyreService;
14     }
15     public String getVehicle() {
16         return "Hi, Im using a Bicycle !!!";
17     }
18     public String getVehicleTyre() {
19         return tyreService.getTyre();
20     }
21 }
```

com.annotation.di.qualifier

- AppolloTyreService.java
- Bicycle.java
- CEATTyreService.java
- MRFTyreService.java
- TyreService.java
- Vehicle.java
- VehicleDemoApp.java

@Qualifier with Setter Injection

com.annotation.di.qualifier
- AppolloTyreService.java
- Bicycle.java
- CEATTyreService.java
- MRFTyreService.java
- TyreService.java
- Vehicle.java
- VehicleDemoApp.java

Bicycle.java

```java
6  @Component
7  public class Bicycle implements Vehicle{
8
9      public TyreService tyreService;
10
11     @Autowired
12     @Qualifier("CEATTyreService")
13     public void setTyreService(TyreService tyreService) {
14         this.tyreService = tyreService;
15     }
16     public String getVehicle() {
17         return "Hi, Im using a Bicycle !!!";
18     }
19     public String getVehicleTyre() {
20         return tyreService.getTyre();
21     }
22 }
```

Bean Scope Annotation

Scopes : singleton & prototype using annotation instead XML bean

Main Class

```
public class VehicleDemoApp {
    public static void main(String[] args) {

        ClassPathXmlApplicationContext context = new
                ClassPathXmlApplicationContext("bean-annotation-appl

        Vehicle v1 = context.getBean("bicycle", Vehicle.class);
        Vehicle v2 = context.getBean("bicycle", Vehicle.class);

        System.out.println("Are Same Objects : "+(v1 == v2));
        System.out.println("v1 memory loaction : "+v1);
        System.out.println("v2 memory loaction : "+v2);

        context.close();
    }
}
```

Singleton Scope

Bicycle.java | VehicleDemoApp.java

```java
1  package com.annotation.di.beanscope;
2  import org.springframework.context.annotation.Scope;
3  import org.springframework.stereotype.Component;
4
5  @Component
6  @Scope("singleton")
7  public class Bicycle implements Vehicle{
8      public String getVehicle() {
9          return "Hi, Im using a Bicycle !!!";
10     }
11 }
```

Problems | Javadoc | Declaration | Console | Terminal

```
<terminated> VehicleDemoApp (7) [Java Application] E:\Softwares\sts-4.12.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_1
Are Same Objects : true
v1 memory loaction : com.annotation.di.beanscope.Bicycle@14028087
v2 memory loaction : com.annotation.di.beanscope.Bicycle@14028087
```

Prototype Scope

Bicycle.java | VehicleDemoApp.java

```java
1  package com.annotation.di.beanscope;
2  import org.springframework.context.annotation.Scope;
3  import org.springframework.stereotype.Component;
4
5  @Component
6  @Scope("prototype")
7  public class Bicycle implements Vehicle{
8      public String getVehicle() {
9          return "Hi, Im using a Bicycle !!!";
10     }
11 }
```

Problems | Javadoc | Declaration | Console | Terminal

<terminated> VehicleDemoApp (7) [Java Application] E:\Softwares\sts-4.12.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_1

```
Are Same Objects : false
v1 memory loaction : com.annotation.di.beanscope.Bicycle@dc7df28
v2 memory loaction : com.annotation.di.beanscope.Bicycle@30f842ca
```

Bean Life Cycle Methods

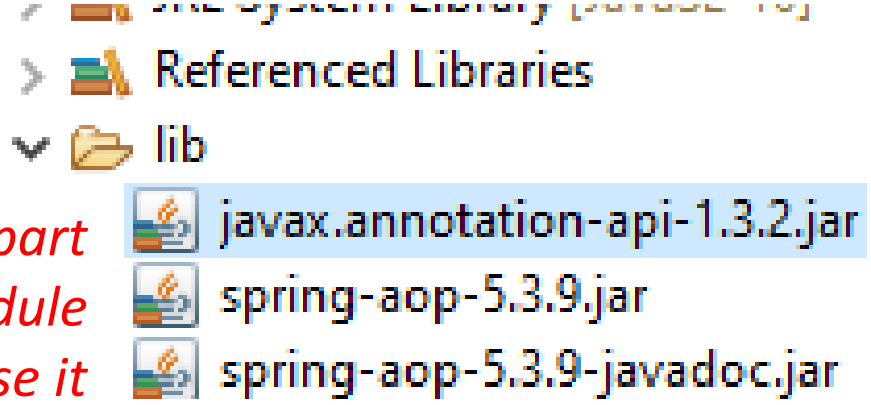
Init-method & destroy-method using annotation instead XML bean

Spring annotations for Lifecycle Methods

- Instead of defining the life cycle methods in XML.

```
<bean id="myVehicle"  class="com.beanlifecycle.Van"
  init-method="method1" destroy-method="method2">
```

- We can define it in the bean class itself using
  - @PostConstruct *(for init-method)*
  - @PreDestroy *(for destroy-method)*



To use @PostConstruct and @PreDestroy annotations are part of Common Annotations API and it's part of JDK module javax.annotation-api. We have to explicitly add this jar to use it in our project

Development Steps

- Create an interface

Van.java | LifeCycleDemo.java | Vehicle.java

```java
package com.annotation.di.beanlifecycle;

public interface Vehicle {
    public String getVehicle();
}

```

- Create a Bean class with @PostConstruct and @PreDestroy
- *Create the main class*
- *Have context:component-scan in configuration .xml*

Van.java | LifeCycleDemo.java | Vehicle.java

```java
package com.annotation.di.beanlifecycle;
import javax.annotation.PostConstruct;

@Component
public class Van implements Vehicle {

    public String getVehicle() {
        return "Hi,im using a Van ";
    }

    @PostConstruct
    public void method1() {
        System.out.println("Initializing Bean ");
    }

    @PreDestroy
    public void method2() {
        System.out.println("Destroying Bean ");
    }
}
```

Main Class

```
4  public class LifeCycleDemo {
5      public static void main(String[] args) {
6          ClassPathXmlApplicationContext context = new
7                  ClassPathXmlApplicationContext("bean-annotation-appli
8
9          Vehicle theVehicle = context.getBean("van", Vehicle.class);
10         System.out.println(theVehicle.getVehicle());
11         context.close();
12     }
13 }
```

Problems | Javadoc | Declaration | Console | Terminal

<terminated> LifeCycleDemo (1) [Java Application] E:\Softwares\sts-4.12.1.RELEASE\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_17.0.0.v20211012-1

```
Initializing Bean
Hi,im using a Van
Destroying Bean
```

Thank You