

# Spring Annotation Bean

## Contents

1 Spring Bean Autowiring – @Autowire .....	2
1.1 Enable annotaion config .....	2
1.2 sử dụng @Autowired annotaion.....	3
1.2 @Qualifier cho đưng độ các nguồn.....	4
1.3 Lỗi An Toàn autowired với 'required=false' .....	5
1.4 Loại trừ một bean ra khỏi autowiring .....	6
2 Spring annotations.....	7
2.1 Bean annotation.....	7
2.2 Context configuration annotations.....	13
3 Sử dụng @Component, @Repository, @Service và @Controller .....	14
4 Sự khác biệt giữa Component và bean .....	14
5 Constructor working Autowired .....	15
Refernce:.....	15

# 1 Spring Bean Autowiring - @Autowire

Trong Spring framework, Việc khởi tạo các dependency trong file configuration là good practices, nhưng Spring Container cũng có thể autowire mối quan hệ giữa các collaborating beans. Điều đó có nghĩa là nó có thể tự động để Spring giải quyết kết nối collaborators với bean của bạn bằng BeanFactory.

**Lưu ý rằng:** Khi @Autowired Class đã phải định nghĩa là 1 bean (có ứng cử là 1 bean kiểu instance của interface @Repository chẳng hạn xem vd hoặc là @Bean trong cấu hình java hoặc <bean> trong file xml), nếu chưa định nghĩa sẽ ném ra 1 lỗi `No qualifying bean of type 'name class' available`

Spring autowiring có 4 mode: là No, byName, byType, và constructor.

Default autowire trong file xml là No, Default config by Java is ByType.

No: Không cần phải tự động cho tất cả bean, những bean phải được chỉ định chính xác thông qua ref element hoặc sử dụng <property> tag để định nghĩa bean.

ByName: Option này cho phép dependency injection dựa trên name của bean. Khi autowiring một property trong bean, property name được sử dụng để tìm và khớp với bean đã được định nghĩa sẵn với tên đó trong file config. Nếu tìm không thấy sẽ xuất ra 1 lỗi.

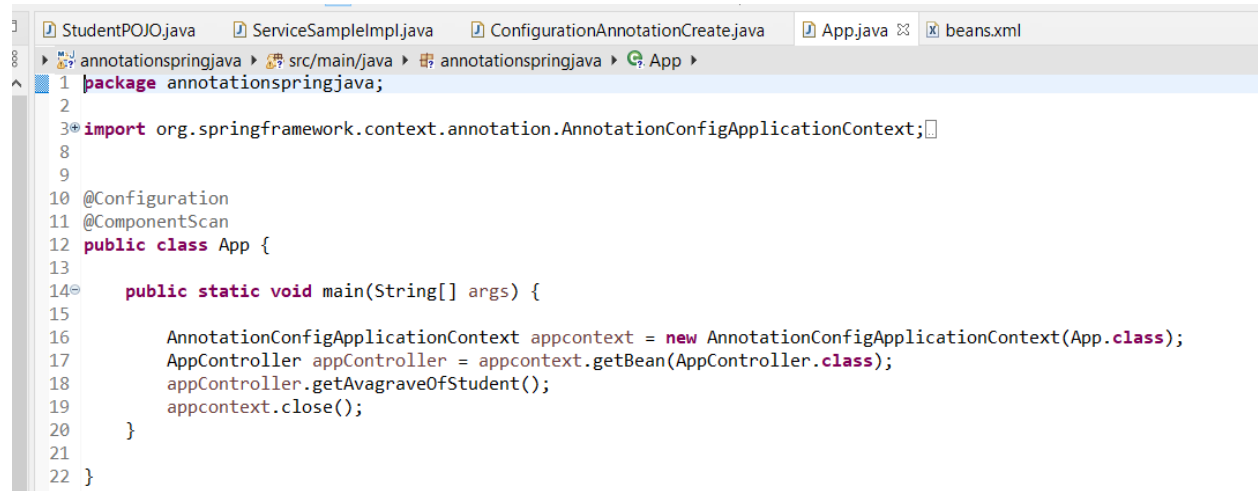
ByType: Option này cho phép dependency injection dựa trên types của bean (type class). Khi autowiring một property trong bean, property' class type được sử dụng để tìm kiếm kiểm tra xem trong config có định nghĩa chưa nếu có sẽ injection bean không thì sẽ xuất hiện lỗi .

Constructor: Autowiring bởi constructor cũng giống như byType, nhưng nó ứng dụng vào các đối số của hàm constructor. Lúc đó nó sẽ tìm kiếm class type của những đối số hàm khởi tạo và sau đó nó autowire byType trên tất cả đối số hàm constructor. Lưu ý rằng nếu không có chính xác 1 bean của đối số hàm constructor trong container, nó sẽ xảy ra 1 lỗi nghiêm trọng

## 1.1 Enable annotation config

```
<context:annotation-config />
Hoặc
<bean class
="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostPr
ocessor"/>
```

## Java config



```
1 package annotationspringjava;
2
3 import org.springframework.context.annotation.AnnotationConfigApplicationContext;
4
5
6
7
8
9
10 @Configuration
11 @ComponentScan
12 public class App {
13
14     public static void main(String[] args) {
15
16         AnnotationConfigApplicationContext appcontext = new AnnotationConfigApplicationContext(App.class);
17         AppController appController = appcontext.getBean(AppController.class);
18         appController.getAvagradeOfStudent();
19         appcontext.close();
20     }
21
22 }
```

@ComponentScan sẽ Tự động tìm các annotaion @Controller, @sevice ... để định nghĩa bean với Spring container.

### 1.2 sử dụng @Autowired annotaion

Bây giờ, sử dụng annotaion configuration đã được bật, nên bạn có thể tự do autowire bean dependency sử dụng @Autowired, có 3 cách để sử dụng

@Autowired trên properties

Khi sử dụng trên properties nó tương đương với việc autowired bằng byType trong file configuraion.

```
public class EmployeeBean
{
    @Autowired
    private DepartmentBean departmentBean;

    public DepartmentBean getDepartmentBean() {
        return departmentBean;
    }
    public void setDepartmentBean(DepartmentBean departmentBean) {
        this.departmentBean = departmentBean;
    }
    //More code
}
```

## @Autowired trên property setter

Khi sử dụng trên setter nó tương đương với việc autowired bằng byType trong file configuraion.

```
public class EmployeeBean
{
    private DepartmentBean departmentBean;

    public DepartmentBean getDepartmentBean() {
        return departmentBean;
    }

    @Autowired
    public void setDepartmentBean(DepartmentBean departmentBean) {
        this.departmentBean = departmentBean;
    }
    //More code
}
```

## @Autowired trên constructor

Khi sử dụng trên constructor nó tương đương với việc autowired bằng byType , nó sẽ tìm các đối số của hàm constructor để injection vào.

```
package com.howtodoinjava.autowire.constructor;

public class EmployeeBean
{
    @Autowired
    public EmployeeBean(DepartmentBean departmentBean)
    {
        this.departmentBean = departmentBean;
    }

    private DepartmentBean departmentBean;

    public DepartmentBean getDepartmentBean() {
        return departmentBean;
    }
    public void setDepartmentBean(DepartmentBean departmentBean) {
        this.departmentBean = departmentBean;
    }
    //More code
}
```

## 1.2 @Qualifier cho đúng độ các nguồn

Khi đã tìm hiểu được cách sử dụng byType autowire mode và các dependency tìm kiếm dựa vào type class nếu không tìm thấy sẽ có error ,

nếu tìm thấy 2 hoặc nhiều type class của properties thì sẽ xuất lỗi để giải quyết vấn đề này ta sử dụng @Qualifier

Chúng ta cần sử dụng @Qualifier cùng với @Autowired annotation và truyền Name của bean cần được đưa vào.

```
public class EmployeeBean
{
    @Autowired
    @Qualifier("finance")
    private DepartmentBean departmentBean;

    public DepartmentBean getDepartmentBean() {
        return departmentBean;
    }
    public void setDepartmentBean(DepartmentBean departmentBean) {
        this.departmentBean = departmentBean;
    }
    //More code
}
```

File xml

```
?xml version="1.0" encoding="UTF-8"?>
<beans>
    <context:annotation-config />

    <bean id="employee"
class="com.howtodoinjava.autowire.constructor.EmployeeBean"
autowire="constructor">
        <property name="fullName" value="Lokesh Gupta"/>
    </bean>

    <!--First bean of type DepartmentBean-->
    <bean id="humanResource"
class="com.howtodoinjava.autowire.constructor.DepartmentBean" >
        <property name="name" value="Human Resource" />
    </bean>

    <!--Second bean of type DepartmentBean-->
    <bean id="finance"
class="com.howtodoinjava.autowire.constructor.DepartmentBean" >
        <property name="name" value="Finance" />
    </bean>
</beans>
```

### 1.3 Lỗi An Toàn autowired với 'required=false'

Ngay khi hết sức cẩn thận khi sử dụng các phụ thuộc bean autowired, bạn vẫn có thể tìm thấy các lỗi kì lạ. Nên để giải quyết vấn đề đó, bạn sẽ cần phải đánh dấu 1 sự lựa chọn cho nó nếu không tìm thấy 1

bean nào phù hợp application cũng sẽ không xuất 1 exception mà nó chỉ cần bỏ qua.

Được thực hiện bằng 2 cách:

Nếu bạn muốn đánh dấu bean autowiring không bắt buộc thì bạn chỉ cần sử dụng `required=false` trên attribute của `@Autowired(required=false)`

```
@Autowired (required=false)
@Qualifier ("finance")
private DepartmentBean departmentBean;
```

Nếu bạn muốn nó áp dụng cho toàn bộ bean trong ứng dụng ta sẽ cài đặt như sau

```
<bean
class="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor">
    <property name="requiredParameterValue" value="false" />
</bean>
```

## 1.4 Loại trừ một bean ra khỏi autowiring

Bởi mặc định, autowiring tìm kiếm và kết nối tất cả các bean định nghĩa trong scope. Nếu bạn muốn loại bỏ 1 vài bean không cho chúng thông qua autowired mode, bạn có thể sử dụng `autowire-candidate` set false.

1 Sử dụng `autowire-candidate` như là false sẽ loại trừ hoàn toàn một bean ra khỏi không trở thành 1 ứng cử viên của bean. Nó sẽ loại trừ hoàn toàn định nghĩa bean ra khỏi tầng cơ sở hạ tầng của bean.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans>
    <context:annotation-config />

    <bean id="employee" class="com.howtodoinjava.autowire.constructor.EmployeeBean"
autowire="constructor">
        <property name="fullName" value="Lokesh Gupta"/>
    </bean>
    <!--Will be available for autowiring-->
    <bean id="humanResource"
class="com.howtodoinjava.autowire.constructor.DepartmentBean" >
        <property name="name" value="Human Resource" />
    </bean>

    <!--Will not participate in autowiring-->
    <bean
id="finance"    class="com.howtodoinjava.autowire.constructor.DepartmentBean"
autowire-candidate="false">
        <property name="name" value="Finance" />
    </bean>
</beans>
```

2 Có 1 cách tiếp cận khác cho việc giới hạn các autowired ứng viên dựa trên pattern-matching trên name của bean. Trên top-level <beans> element chấp nhận 1 hoặc nhiều pattern với default-autowire-candidates' attribute. Cho ví dụ , để giới hạn autowire candidates với bất kì bean nào có name kết thúc bằng 'impl' cung cấp là \*impl, định nghĩa nhiều pattern phân cách bằng dấu phẩy.

```
<?xml version="1.0" encoding="UTF-8"?>
<beans default-autowire-candidates="*Impl,*Dao">
  <context:annotation-config />

  <bean id="employee" class="com.howtodoinjava.autowire.constructor.EmployeeBean"
    autowire="constructor">
    <property name="fullName" value="Lokesh Gupta"/>
  </bean>
  <!--Will be available for autowiring-->
  <bean id="humanResource"
    class="com.howtodoinjava.autowire.constructor.DepartmentBean" >
    <property name="name" value="Human Resource" />
  </bean>

  <!--Will not participate in autowiring-->
  <bean
    id="finance"    class="com.howtodoinjava.autowire.constructor.DepartmentBean"
    autowire-candidate="false">
    <property name="name" value="Finance" />
  </bean>
</beans>
```

## 2 Spring annotations

Học về các spring annotations được sử dụng rộng rãi. Trong bài này chúng tôi sẽ tóm tắt bao phủ các annotation quan trọng được cung cấp bởi spring core để định nghĩa các bean và tạo mới các cấu hình phức tạp trong ứng dụng.

### 2.1 Bean annotation

Để sử dụng được các annotation bean sử dụng bằng cách

Annotation-based-configuration

View github to config:

<https://github.com/nguyenthinhit996/sharefullcode/tree/spring/spring/core/annotation-based-configuration>

Cung cấp 1 list annotations được sử dụng để định nghĩa beans trong spring để điều khiển sự đăng kí tham chiếu trong applocation context.

@Bean: Được đặc ở cấp độ method sử dụng để tạo 1 spring bean. Khi cấu hình được thực thi trên method, nó sẽ đăng kí và trả về giá trị như 1 bean bên trong Beanfactory. Nó cũng giống như đăng kí 1 bean ở dưới file xml thôi. Nếu không chỉ định tên @Bean("name") nó sẽ lấy tên của method đăng kí với BeanFactory.

Có nhiều loại bean:

Nếu @Bean trong class có @Configuration sẽ là bean bình thường cứ gọi trực tiếp bởi BeanFactory

Nếu @Bean trong class not @Configuration sẽ là 1 factory-method pattern của 1 bean. Sử dụng cho mục đích xml thì ok.

Xem ví dụ: Class POJO bình thường.

```
StudentPOJO.java  ServiceSampleImpl.java
> annotationspringjava > src/main/java > annotationspringjava.POJO > StudentPOJO
1 package annotationspringjava.POJO;
2
3 public class StudentPOJO {
4     private String name;
5     private Integer age;
6
7     public String getName() {}
10    public void setName(String name) {}
13    public Integer getAge() {}
16    public void setAge(Integer age) {}
19    public StudentPOJO(String name, Integer age) {}
24
25    public StudentPOJO() {
26        super();
27        this.name = "default";
28        this.age = 0;
29    }
30
31    @Override
32    public String toString() {
33        return this.name+","+this.age;
34    }
35
36
```

Gọi Autowired sẽ báo lỗi do chưa đăng kí bean nào với tên là

No qualifying bean of type 'annotationspringjava.POJO.StudentPOJO' available

Có 2 cách để giải quyết 1 là đăng kí bằng @Bean cùng với class @Configuration

```
StudentPOJO.java  ServiceSampleImpl.java  ConfigurationAnnotationCreate.java  App.java
> annotationspringjava > src/main/java > annotationspringjava.beancreate > ConfigurationAnnotationCreate
1 package annotationspringjava.beancreate;
2
3 import org.springframework.context.annotation.Bean;
7
8 @Configuration
9 public class ConfigurationAnnotationCreate {
10
11     @Bean
12     StudentPOJO getObjectStudentPOJOByBean() {
13         return new StudentPOJO("bean",11111);
14     }
15 }
16
```

, 2 là đăng kí bằng xml file.làm giống ví dụ bên dưới



```
StudentPOJO.java ServiceSampleImpl.java ConfigurationAnnotationCreate.java App.java beans.xml
61 <bean id="lifecycleannotation"
62     class="basicioc.pojo.ContractPOJODependencyLifeCycleAnnotation"
63     />
64
65
66 <!-- bean mode lite -->
67
68 <!-- bean mode lite -->
69 <bean id="beanNonStaticUseBeanInClassNormal"
70     class="basicioc.beanannotations.UseBeanInClassNormal"
71     />
72
73 <bean id="useBeanInClassNormal"
74     class="basicioc.beanannotations.UseBeanInClassNormal"
75     factory-method="getBeanInNormalClassOrNotConfigAnnoClass"
76     factory-bean="beanNonStaticUseBeanInClassNormal"
77     />
78 </beans>
79
```

```
@Bean
EmployeeService employeeService()
{
    return new EmployeeServiceImpl();
}
```

**@Component:** Chỉ định cho Class, interface (including annotation type), or enum declaration . Chỉ định chúng như là 1 componet và sẽ được auto-detected khi sử dụng annotation-based-configurartion và scan classpath.

@Component annotation đánh dấu một java class như là bean và the component-scanning của spring có thể nhắc nó lên và đẩy nó vào trong applicaton context.

```
@Component
public class EmployeeDAOImpl
    implements EmployeeDAO {
    ...
}
```

**@Repository:** là tổng môn hóa của @component annotation. Ngoài việc import annotation các class DAO vào DI container. Nó cũng đánh dấu các exception (ném ra từ DAO) đủ điều kiện để dịch sang Spring DataAccessException.

Mặc dù cách sử dụng @Component đã tốt nhưng chúng ta có thể sử dụng cho phù hợp hơn annotation dùng với mục đích xác định lớp DAOs bằng cách sử dụng @Repository

```

Public interface EmployeeDAO
{
    public EmployeeDTO createNewEmployee();
}

@Repository ("employeeDao")
public class EmployeeDAOImpl implements EmployeeDAO
{
    public EmployeeDTO createNewEmployee()
    {
        EmployeeDTO e = new EmployeeDTO();
        e.setId(1);
        e.setFirstName("Lokesh");
        e.setLastName("Gupta");
        return e;
    }
}

```

@Service: là tổng môn hóa của @Component. Nó chỉ định đây là class "Bussiness Service Facade". Phù hợp cho service-layer.

```

public interface EmployeeManager
{
    public EmployeeDTO createNewEmployee();
}

@Service ("employeeManager")
public class EmployeeManagerImpl implements EmployeeManager
{
    @Autowired
    EmployeeDAO dao;

    public EmployeeDTO createNewEmployee()
    {
        return dao.createNewEmployee();
    }
}

```

@Controller: là chuyên môn hóa của @Component, để chú thích điều kiện (web controller). Nó sử dụng kết hợp với annotated handler methods dựa trên RequestMapping annotaion.

```

@Controller ("employeeController")
public class EmployeeController
{
    @Autowired
    EmployeeManager manager;

    public EmployeeDTO createNewEmployee()
    {
        return manager.createNewEmployee();
    }
}

```

Trong thực tế, chúng ta sẽ rất ít trường hợp ở đó sử dụng @Component. Hầu hết các trường hợp, chúng tôi sẽ sử dụng @repository, @Service và @Controller. @Component sẽ được dùng nếu ko rơi vào 3 trường hợp trên.

@Qualifier: Trong lúc autowiring, nếu có nhiều hơn 1 bean cùng với kiểu của nó có giá trị trong container thì nó sẽ ném ra 1 lỗi runtime exception. Để fix điều đó, chúng ta cần chỉ định cho autowire sử dụng cái nào để trả về cho yêu cầu bean để injection. Bằng các sử dụng @Qualifier

```
public class EmployeeService {  
  
    @Autowired  
    @Qualifier("fsRepository")  
    private Repository repository;  
  
}
```

@Require: dùng cho method (thông thường thì dùng ở hàm setter), mặc định bean autowired chỉ check dependency đã được cài đặt, nó không check liệu ở đó cài đặt giá trị là gì null hoặc non-null. Nên sử dụng @Require để check giá trị được set là not null. Nhưng bây giờ nó đã được deprecated kể từ spring 5.1 vì từ đây ủng hộ việc sử dụng constructor injection hơn là dùng setter.

@Value: Áp dụng cho field, method, constructor parameter level. Chỉ định giá trị default cho đối tượng được set , có thể sử dụng spring expression language

```
public class SomeService {  
  
    @Value("${ENV}")  
    private String environment;  
  
}
```

@Lazy: Chỉ định rằng một bean sẽ được lazily(lười biếng) khởi tạo. Mặc định in spring DI, eager(hăng hái) khởi tạo được chỉ định.

Khi áp dụng vào bất kì bean nào, việc khởi tạo của nó sẽ không xảy ra cho đến khi có sự tham chiếu từ bean khác hoặc có sự truy xuất rõ ràng từ BeanFactory.

```
public class SomeService {  
  
    @Autowired  
    @Lazy  
    private RemoteService remotng;  
  
}
```

@DependsOn: Trong quá trình componet-scanning, nó được sử dụng để xác định các bean mà bean hiện tại phụ thuộc vào. Bean được chỉ định phải được đảm bảo rằng nó đã được tạo trong container trước bean hiện tại

```
public class SomeService {  
  
    @Autowired  
    @DependsOn ("pingService")  
    private RemoteService remotng;  
  
}
```

@Lookup: Chỉ định một method như là 'lookup' method. Sử dụng tốt nhất cho việc injecting một prototype-scope vào trong một singleton bean.

```
@Component  
@Scope("prototype")  
public class AppNotification {  
    //prototype-scoped bean  
}  
  
@Component  
public class NotificationService {  
  
    @Lookup  
    public AppNotification getNotification() {  
        //return new AppNotification();  
    }  
  
}
```

@Primary: Chỉ định rằng 1 bean sẽ được ưu tiên khi có nhiều ứng cử viên đủ điều kiện để tự động tạo một phụ thuộc single-value.

Khi không sử dụng @Primary, chúng ta cần phải cung cấp @Qualifier annotation để chỉ định chính xác inject các bean.

Trong ví dụ, khi FooRepository sẽ được autowired, the instance của HibernateFooRepository sẽ được injected

```
@Component  
public class JdbcFooRepository  
    extends FooRepository {  
  
}  
  
@Primary  
@Component  
public class HibernateFooRepository  
    extends FooRepository {  
  
}
```

## 2.2 Context configuration annotations

Những annotations giúp liên các bean khác với nhau để tạo thành lúc runtime application context

**@ComponentScan:** @ComponentScan cùng với @Configuration được sử dụng để cho phép và cấu hình tìm kiếm các component. Mặc định, nếu không chỉ định path nó sẽ tìm kiếm component ở current package và tất cả các package con của nó.

Sử dụng component scanning, spring có thể auto-scan tất cả class có annotation với stereotype-annotation @Component, @Controller, @Service, @Repository và cấu hình chúng với BeanFactory.

```
@Configuration
@ComponentScan(basePackages = {com.howtodoinjava.data.jpa})
public class JpaConfig {
}
```

**@Configuration:** Chỉ định 1 class có thể có 1 hoặc nhiều @Bean method và có thể được xử lý bởi container để tạo ra định nghĩa bean khi sử dụng với @ComponentScan.

```
@Configuration
public class AppConfig {

    @Bean
    public AppUtils appUtils()
    {
        return new AppUnits();
    }
}
```

**@Profile:** Chỉ định 1 component đủ điều kiện cho đăng ký bean khi 1 hoặc nhiều chỉ định profile đang hoạt động. Một Profile là một tên được đặc theo logical grouping như là dev hoặc prod.

```
@Bean
@Profile("dev")
public AppUtils appUtils()
{
    return new DevAppUnits();
}

@Bean
@Profile("prod")
public AppUtils appUtils()
{
    return new ProdAppUnits();
}
```

@Import: Chỉ định 1 hoặc nhiều component được import, thông thường là class của @Configuration. @Bean định nghĩa khởi tạo 1 import @Configuraion class nên có thể dc truy cập bằng cách sử dụng @Autowired injection.

```
@Configuration
@Import({ JpaConfig.class, SchedulerConfig.class })
public class AppConfig {

}
```

@ImportResouce: chỉ định 1 hoặc nhiều resouce chứa các bean được định nghĩa để import vào. Nó được sử dụng cho file xml định nghĩa các bean cũng giống như @Import sử dụng cho java configuration sử dụng @Bean.

```
@Configuration
@ImportResource( { "spring-context.xml" } )
public class ConfigClass {

}
```

### 3 Sử dụng @Component, @Repository, @Service và @Controller

Chúng ta sẽ sử dụng @Repository, @Service và @Controller cho tương ứng với Dao, Service, Controller classes. Nhưng thực tế Dao và Services chúng tôi thường xuyên sẽ tách biệt chúng bằng interface và classes. Interface cho việc định nghĩa các giao kèo, quy định còn class sẽ định nghĩa triển khai các giao kèo kí ước đó.

Luôn luôn đặt annotation trên class không đặt ở interface.

### 4 Sự khác biệt giữa Component và bean

Trong Spring, cả 2 khá khác biệt nhau

@Component được sử dụng để auto phát hiện và auto cấu hình bean sử dụng classpath scanning. (@ComponentScan) ở đó ngầm hiểu rằng 1 1 mapping giữa 1 class annotaion component với 1 bean.

@Bean được sử dụng bằng cách chỉ rõ ràng tạo ra 1 single bean (singleton scope) hơn là để hệ thống tự động làm đều đó cho chúng ta.

Một sự khác biệt lớn giữa @Bean là áp dụng cho Method còn @Component là cấp class.

## 5 Constructor working Autowired

Khi run spring sẽ call constructor rồi mới call các autowired trong class.

Đăng kí là bean trong spring container :

Dùng @Component ...

Call constructor autowired -> constructor non argument -> constructor argument.

+ Trường hợp không có constructor đánh dấu là autowired

- Tìm constructor() non argument nếu không thấy, sẽ tìm constructor(argument...) lưu ý ở đây chỉ có 1 constructor(argument...) nếu có 2 sẽ xuất hiện error BeanCreationException , và các argument phải bean nếu ko sẽ lỗi No qualifying bean.

+ Trường hợp constructor là 1 autowired

- Nó sẽ được call thay vì call các constructor non autowire, nếu có 2 constructor autowired sẽ báo lỗi BeanCreationException.

Dùng @Bean trong file @Configuration

Ta phải tự định nghĩa nó sử dụng constructor nào

```
@Bean
StudentPOJO getObjectStudentPOJOByBean() {
    return new StudentPOJO(" bean-StudentPOJO",11111);
}
```

## Refernce :

<https://howtodoinjava.com/>