

Annotation Based Configuration:

In Spring Framework annotation-based configuration instead of using XML for describing the bean wiring, you have the choice to **move the bean configuration into component class**. It is done by using annotations on the relevant class, method or the field declaration.

Below is the configuration file in case you want to use annotation in your application:

```
<?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=""></context:component-scan>
</beans>
```

<context:component-scan> detects the annotations by package scanning. To put it differently, it tells Spring which packages need to be scanned to look for the annotated beans or components.

@Component, @Repository, @Service, @Controller, @RestController, and @Configuration are several ones that `<context:component-scan>` can detect. In base-package attribute we need to specify the package it need to scan for annotations

Now let's see Inversion Of control using annotations

Steps to Make annotation based Configuration

Step 1: Mention the base package in xml file

```
<context:component-scan
base-package="com.tapacad.spring"></context:component-scan>
// com.tapacad.spring is the package name where all the
classes are available
```

annotationApplicationContext.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.tapacad.spring"></context:component-scan>

</beans>
```

Step 2: Now we need to make class as bean using @component annotation

@Component is an annotation that allows Spring to automatically detect our custom beans.

Add @component(mention the id that is specified in the configuration file) to the class to represent it as bean to avoid xml configuration that we used to do for bmw class as shown below

```
<bean id="bmw" class="com.tapacad.spring.BMW">
    <constructor-arg ref ="engine"></constructor-arg>
</bean>
```

BMW.java

```
@Component("bmw") // Now BMW class is a bean
public class BMW implements Car
{
    //Logic
}
```

Bean represented for nano in xml is replaced by @component("nano") as shown below

```
<bean id="nano" class="com.tapacad.spring.Nano">
    <constructor-arg ref ="engine"></constructor-arg>
</bean>
```

Nano.java

```
@Component("nano")
public class Nano implements Car
{

}
```

Bean represented for audi in xml is replaced by @component("audi") as shown below

```
<bean id ="audi" class="com.tapacad.spring.Audi" >
    <property name="rocketEngine" ref =
"engine"></property>
    <property name="colour" value =
"${colour}"></property>
    <property name="price" value = "${price}"></property>
</bean>
```

Audi.java

```
@Component("audi")
public class Audi implements Car
{
    //Logic
}
```

Bean represented for Rocket Engine in xml is replaced by @component("engine") as shown below

```
<bean id ="engine" class="com.tapacad.spring.RocketEngine">
</bean>
```

RocketEngine.java

```
@Component("engine")
public class RocketEngine implements Engine
{
    //Logic
}
```

Step 3: Now let's try to load annotationApplicationContext.xml file and print object reference

MyApp7.java

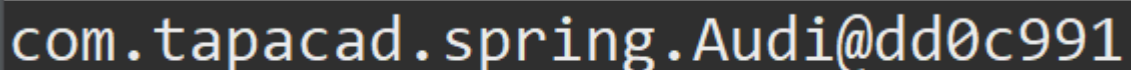
```
package com.tapacad.spring;
import
org.springframework.context.support.ClassPathXmlApplication
Context;
public class MyApp7 {
    public static void main(String[] args) {
        //      Load annotation Application context
        ClassPathXmlApplicationContext context = new
ClassPathXmlApplicationContext("annotateApplicationContext.
xml");

        //      Get bean
        Car car = context.getBean("audi", Car.class);

        //      call getCarDetails()
        System.out.println(car);

        //      close context
        context.close();
    }
}
```

Output:



```
com.tapacad.spring.Audi@dd0c991
```

If you observe from the above output you have got object reference of audi. This is only **inversion of control in case of annotations**

Dependency Injection Using Annotations

Scenario:

Now let's try to get bmw bean and rocket engine bean that we have declared in bmw

MyApp8.java

```
package com.tapacad.spring;

import
org.springframework.context.support.ClassPathXmlApplication
Context;

public class MyApp8 {

    public static void main(String[] args) {
//        Load Application context
        ClassPathXmlApplicationContext context =
            new
ClassPathXmlApplicationContext("annotateApplicationContext.
xml");

//        Get bean
        Car car = context.getBean("bmw", Car.class);

//        call getCarDetails() and getEngineDetail
        System.out.println(car.getCarDetails());
        System.out.println(car.getEngineDetail());

//        close context
        context.close();

    }

}
```

Output:

```
BMW  
Rocket Engine gives good performance
```

If you observe from the above output `getCarDetail()` and `getEngineDetail()` executed successfully .

The only reason for this is we have declared the attribute as `RocketEngine` in `bmw` class so it is creating a bean of `RocketEngine` as well.