**SpringBeanConfiguration using Annotation**

We can spring configuration through Annotation also.

**Using autowiring with @Autowired annotations**

Apart from the autowiring modes provided in bean configuration file, autowiring can be specified in bean classes also using @Autowired annotation. To use @Autowired annotation in bean classes, you must first enable the annotation in spring application using below configuration.

|  |
| --- |
| <context:annotation-config /> |

Same can be acheived using ‘AutowiredAnnotationBeanPostProcessor’ bean definition in configuration file.

|  |
| --- |
| <bean class ="org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor"/> |

Now, when annotation configuration has been enables, you are free to autowire bean dependencies using @Autowired, the way you like. This is done by three ways:

**1) @Autowired on properties**

When @Autowired is used on properties, it is equivalent to autowiring by ‘byType’ in configuration file.

|  |
| --- |
| public class EmployeeBean  {      @Autowired      private DepartmentBean departmentBean;        public DepartmentBean getDepartmentBean() {          return departmentBean;      }      public void setDepartmentBean(DepartmentBean departmentBean) {          this.departmentBean = departmentBean;      }      //More code  } |

**2) @Autowired on property setters**

When @Autowired is used on setters, it is also equivalent to autowiring by ‘byType’ in configuration file.

|  |
| --- |
| public class EmployeeBean  {      private DepartmentBean departmentBean;        public DepartmentBean getDepartmentBean() {          return departmentBean;      }        @Autowired      public void setDepartmentBean(DepartmentBean departmentBean) {          this.departmentBean = departmentBean;      }      //More code  } |

**3) @Autowired on constructors**

When @Autowired is used on bean’s constructor, it is also equivalent to autowiring by ‘constructor’ in configuration file.

|  |
| --- |
| 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  } |

**Using @Qualifier in case of conflict**

As we learned that if we are using autowiring in ‘byType’ mode and dependencies are looked for property class types. If no such type is found, an error is thrown. But, what if there are two or more beans for same class type.

In this case spring will not be able to choose correct bean to inject into property, and you will need to help the container using qualifiers.

To resolve a specific bean using qualifier, we need to use @Qualifier annotation along with @Autowired annotation and pass the bean name in annotation parameter. Take a look below for example:

|  |
| --- |
| public class EmployeeBean  {      @Autowired      @Qualifier("finance")      private DepartmentBean departmentBean;        public DepartmentBean getDepartmentBean() {          return departmentBean;      }      public void setDepartmentBean(DepartmentBean departmentBean) {          this.departmentBean = departmentBean;      }      //More code  } |

where duplicate beans are as below:

|  |
| --- |
| <?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> |

**Making autowiring error safe using required=false**

Even if you have used utmost care in autowiring bean dependencies, still you may find strange lookup failures. So, solve this issue, you will need to make autowiring optional so that if no dependency is found, application should not throw any exception and autowiring should simpy be ignored.

This can be done in two ways:

1)If you want to make specific bean autowiring non-mandatory for a specific bean property, use required=”false” attribute in @Autowired annoration

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

2) If you want to apply optional autowiring at global level i.e. for all properties in all beans; use below configuration setting.

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

**Excluding a bean from being available for autowiring**

By default, autowiring scan and matches all bean definitions in scope. If you want to exclude some bean definitions so that they can not be injected through autowiring mode, you can do this using ‘autowire-candidate’ set to false.

1) Using ‘**autowire-candidate**‘ as false totally exclude a bean from being an autowire candidate. It totally exclude that specific bean definition from being available to the autowiring infrastructure.

|  |
| --- |
| <?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) Another option is to limit autowire candidates based on pattern-matching against bean names. The top-level element accepts one or more patterns within its ‘**default-autowire-candidates**‘ attribute. For example, to limit autowire candidate status to any bean whose name ends with ‘Impl’, provide a value of ‘\*Impl’. To provide multiple patterns, define them in a comma-separated list.

|  |
| --- |
| <?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> |

Note that an explicit value of ‘true’ or ‘false’ for a bean definition’s ‘autowire-candidate’ attribute always takes precedence, and for such beans, the pattern matching rules will not apply.

**Example Annotation Bean Configuration**

**Body.Java**

**package** com.doj.spring;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** Body {

String bodyType="STEEL";

**public** String getBodyType() {

**return** bodyType;

}

}

**Engine.java**

**package** com.doj.spring;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** Engine {

**final** String engineName="FORD";

**public** String getEngineName() {

**return** engineName;

}

}

**Tyre.java**

**package** com.doj.spring;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** Tyre {

String tyreName="MRF";

**public** String getTyreName() {

**return** tyreName;

}

}

**Car.java**

**package** com.doj.spring;

**import** org.springframework.beans.factory.annotation.Autowired;

**import** org.springframework.stereotype.Component;

@Component

**public** **class** Car {

@Autowired(required=**false**)

Engine engine;

@Autowired

Tyre tyre;

@Autowired

Body body;

**public** Car(Engine engine, Tyre tyre, Body body) {

**super**();

**this**.engine = engine;

**this**.tyre = tyre;

**this**.body = body;

}

**public** Car() {

**super**();

}

**public** **void** driveCar(){

System.*out*.println("You are driving a car which has a engine "+**this**.engine.engineName

+" has a tyre "+**this**.tyre.getTyreName()+" has a body "+**this**.body.getBodyType());

}

}

**CarDemo.java**

**package** com.doj.spring;

**import** org.springframework.context.support.AbstractApplicationContext;

**import** org.springframework.context.support.ClassPathXmlApplicationContext;

**public** **class** CarDemo {

**public** **static** **void** main(String[] args) {

AbstractApplicationContext context = **new** ClassPathXmlApplicationContext("spring.xml");

Car car = (Car) context.getBean("car");

car.driveCar();

context.close();

}

}

**Spring.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"*>

<!-- There are no bean definition here -->

<!-- Component scaning from this base package -->

<context:component-scan base-package=*"com.doj.spring"*/>

</beans>

Points:-

# Put this “context:component” in bean configuration file, it means, enable auto scanning feature in Spring. The **base-package** is indicate where are your components stored, Spring will scan this folder and find out the bean (annotated with @Component) and register it in Spring container.

**#Annotate with @Component to indicate this is class is an auto scan component.**

**#@Autowired annotation – We can use this annotation for spring bean autowiring. This annotation can be applied on variables and methods for autowiring byType. We can also use this annotation on constructor for constructor based autowiring.**

**For this annotation to work, we also need to enable annotation based configuraiton in spring bean configuration file. This can be done by context:annotation-config element or by defining a bean of type org.springframework.beans.factory.annotation.AutowiredAnnotationBeanPostProcessor.**

**# @Qualifier annotation – This annotation is used to avoid conflicts in bean mapping and we need to provide the bean name that will be used for autowiring. This way we can avoid issues where multiple beans are defined for same type. This annotation usually works with the @Autowired annotation. For constructors with multiple arguments, we can use this annotation with the argument names in the method.**