# Spring Annotations

## Core Spring Annotations

These annotations are used by Spring to guide creation and injection of beans.

| **ANNOTATION** | **USE** | **DESCRIPTION** |
| --- | --- | --- |
| **@Autowired** | Constructor, Field, Method | Declares a constructor, field, setter method, or configuration method to be autowired by type. Items annotated with @Autowired do not have to be public. |
| **@Configurable** | Type | Used with <context:springconfigured> to declare types whose properties should be injected, even if they are not instantiated by Spring. Typically used to inject the properties of domain objects. |
| **@Order** | Type, Method, Field | Defines ordering, as an alternative to implementing the org. springframework.core.Ordered interface. |
| **@Qualifier** | Field, Parameter, Type, Annotation Type | Guides autowiring to be performed by means other than by type. |
| **@Required** | Method (setters) | Specifies that a particular property must be injected or else the configuration will fail. |
| **@Scope** | Type | Specifies the scope of a bean, either singleton, prototype, request, session, or some custom scope. |

#### Example:

//File: Student.java

public class Student {

private int sno;

private String name;

@Autowired

private Address address; //this property is Autowiring

//Setters & getters

@Autowired

public Student(Address address) {

System.*out*.println("CONSTRCUTOR Injection");

this.address = address;

}

@Autowired

public void setAddress(Address address) {

this.address = address;

System.*out*.println("Setter Injection");

}

}

* @Autowired annotation is auto wire the bean by matching data type.
* **@Autowired** can be applied on setter method, constructor or a field.in above we applied at 3 places, we need to place at one of the places.

To activiate Spring core annotations in our application, we have to configure *AutowiredAnnotationBeanPostProcessor* bean in SpringConfig.xml

<!-- File : SpringConfig.xml -->

<beans>

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

<bean id=*"student"* class=*"anno.Student"*>

<property name=*"sno"* value=*"101"*></property>

<property name=*"name"* value=*"Satya Kaveti"*></property>

<!-- This property will Autowires

<property name="address">

<ref bean="addr" />

</property> -->

</bean>

<bean id=*"address"* class=*"anno.Address"*>

<property name=*"hno"* value=*"322"*></property>

<property name=*"city"* value=*"HYDERABAD"*></property>

</bean>

</beans>

//File : AutowireAnnotationExample.java

public class AutowireAnnotationExample {

public static void main(String[] args) {

ApplicationContext context = new ClassPathXmlApplicationContext("anno/SpringConfig.xml");

Object ob = context.getBean("student");

Student st = (Student) ob;

Address addr = st.getAddress();

System.*out*.println("Sno : "+st.getSno());

System.*out*.println("Name : "+st.getName());

System.*out*.println("City : "+addr.getCity());

}

}

Sno : 101

Name : Satya Kaveti

City : HYDERABAD

**The @Qualifier** annotation us used to control which bean should be autowire on a field. For example, bean configuration file with two similar person beans**.**

<bean id="address1" class="anno.Address">

<property name="hno" value="322"></property>

<property name="city" value="HYDERABAD"></property>

</bean>

<bean id="address2" class="anno.Address">

<property name="hno" value="322"></property>

<property name="city" value="HYDERABAD"></property>

</bean>

public class Student {

private int sno;

private String name;

@Autowired

@Qualifier("address1")

private Address address;

}

## Stereotyping Annotations

| **ANNOTATION** | **USE** | **DESCRIPTION** |
| --- | --- | --- |
| **@Component** | Type | Generic stereotype annotation for any Spring-managed component. |
| **@Controller** | Type | Stereotypes a component as a Spring MVC controller. |
| **@Repository** | Type | Stereotypes a component as a repository. Also indicates that SQLExceptions thrown from the component's methods should be translated into Spring DataAccessExceptions. |
| **@Service** | Type | Stereotypes a component as a service. |

## Spring MVC Annotations

These were introduced in Spring 2.5 to make it easier to create Spring MVC applications with minimal XML configuration and without extending one of the many implementations of the Controller interface.

| **ANNOTATION** | **USE** | **DESCRIPTION** |
| --- | --- | --- |
| **@Controller** | Type | Stereotypes a component as a Spring MVC controller. |
| **@InitBinder** | Method | Annotates a method that customizes data binding. |
| **@ModelAttribute** | Parameter, Method | When applied to a method, used to preload the model with the value returned from the method. When applied to a parameter, binds a model attribute to the parameter. table |
| **@RequestMapping** | Method, Type | Maps a URL pattern and/or HTTP method to a method or controller type. |
| **@RequestParam** | Parameter | Binds a request parameter to a method parameter. |
| **@SessionAttributes** | Type | Specifies that a model attribute should be stored in the session. |

## AOP Annotations

| **ANNOTATION** | **USE** | **DESCRIPTION** |
| --- | --- | --- |
| **@Aspect** | Type | Declares a class to be an aspect. |
| **@After** | Method | Declares a method to be called after a pointcut completes. |
| **@AfterReturning** | Method | Declares a method to be called after a pointcut returns successfully. |
| **@AfterThrowing** | Method | Declares a method to be called after a pointcut throws an exception. |
| **@Around** | Method | Declares a method that will wrap the pointcut. |
| **@Before** | Method | Declares a method to be called before proceeding to the pointcut. |
| **@DeclareParents** | Static Field | Declares that matching types should be given new parents,that is, it introduces new functionality into matching types. |
| **@Pointcut** | Method | Declares an empty method as a pointcut placeholder method. |

# Spring Testing

# References

* Introductions: <http://www.java4s.com/spring/>
* IoC,DI : <https://www.javatpoint.com/ioc-container>
* Setter: java4s.com
* Autowire : [http://www.java4s.com/](http://www.java4s.com/spring/example-on-spring-autowiring-by-autodetect/) javatpoint.com, mkyong
* Spring JDBC: javatponit.com
* Spring AOP: mkyong.com, JavaTpoint.com
* SpringEL : <http://www.baeldung.com/spring-expression-language>
* RequestMapping : <http://javabeat.net/requestmapping-spring-mvc/>
* Controllers, view resolevrs: mkyong.com
* Form : [http://www.codejava.net/](http://www.codejava.net/frameworks/spring/spring-mvc-form-validation-example-with-bean-validation-api)
* <http://www.beingjavaguys.com/p/spring-framework.html>
* <https://howtodoinjava.com/java-tutorials-list-howtodoinjava/>



# References

<https://docs.spring.io/spring/docs/5.0.0.RELEASE/spring-framework-reference/core.html#spring-core>

1. **Core:** <https://docs.spring.io/spring/docs/5.0.0.RC3/spring-framework-reference/core.html#beans>
2. **AOP:** <https://docs.spring.io/spring/docs/5.0.0.RC3/spring-framework-reference/core.html#aop-introduction>
3. **SpringData**: <https://docs.spring.io/spring/docs/5.0.0.RC3/spring-framework-reference/data-access.html#jdbc-introduction>
4. **SpringWeb**: <https://docs.spring.io/spring/docs/5.0.0.RC3/spring-framework-reference/web.html#mvc-introduction>
5. **Spring Test**: https://docs.spring.io/spring/docs/5.0.0.RC3/spring-framework-reference/testing.html#unit-testing

## Core

Spring Life Cycle : <http://javainsimpleway.com/spring-bean-life-cycle/>

<https://javabeginnerstutorial.com/spring-framework-tutorial/java-spring-bean-lifecycle/>

Annotations

<https://springframework.guru/spring-framework-annotations/>

<https://dzone.com/articles/a-guide-to-spring-framework-annotations>

## Spring Data

<https://www.infoq.com/articles/spring-data-intro>

<http://spring.io/projects/spring-data>

<http://www.springboottutorial.com/introduction-to-spring-data-with-spring-boot>