**SPRING**

* Introduced by Rod Thompson in 2003
* Framework is a structure with reusable codes.
* Industry best coding practices.
* Light weight
* IOC Container (Dependency Injection)
* Spring is a layered architecture

**Dependency Injection (Inversion of control):**

Allows modules to be as independent as possible. Loose Coupling.

Example: Building (Interface)

Implements Both have abstract method getPlanDetails()

Apartment Villa

Building obj = new Apartment();

Building obj = new Villa(); Object reference is generic. (But here also tight coupling exists)

**Spring Container or IOC Container:**

Code will give the required info to 3rd part API, and this API will create the object and inject that object to the class. This is called dependency injection. It internally implemented a factory pattern.

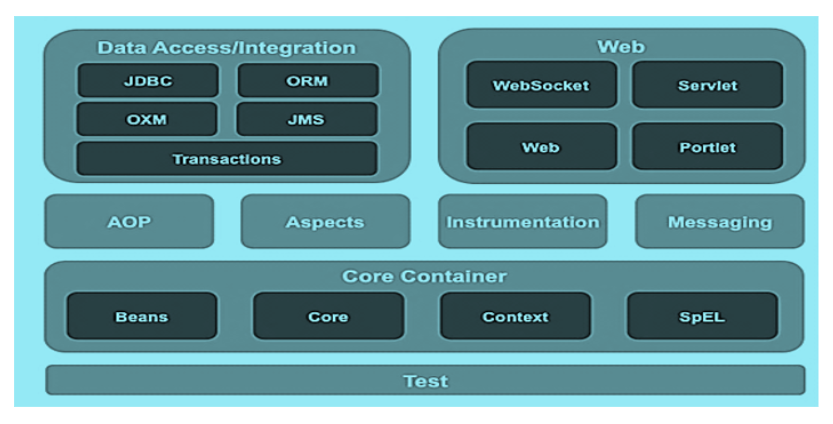
**AOP**: Aspect Oriented Programming

Aspects => Cross cutting concerns

Separate the code based on functionality (ex: Logging features, transaction management)

AOP is used to reduce the boiler plate code.

**Spring Architecture:**



* In Spring, we will not create the object; instead the container creates it.
* The container will create the objects, wire them together, configure them, and manager their complete life cycle from creation till destruction.
* The Container gets its instructions on what objects to instantiate, configure and assemble by reading the configuration metadata provided.
* The configuration metadata can be represented either by XML, Java Annotations, Mixed Configuration (XML and Annotation)

DEMO:

Every object that is managed by the container has to be configured in the configuration file with the bean tag.

Every bean has the following attributes: **id** => to uniquely identify and refer the bean

**Class** => to specify the fully qualified class name of the bean object

The bean configurations are specified in the XML (beans.xml).

ClassPathXmlApplicationContext factory class will read the XML file and provide the configuration information to the application.

Ex:

public class TestApplication {

public static void main(String[] args){

ApplicationContext context = new ClassPathXmlApplicationContext(“beans.xml”);

Welcome obj =(Welcome) context.getBean(“Welcome”);

}

}

**Dependency Injection:**

The technology that actually defines Spring (Heart of Spring).

Keeps classes as independent as possible.

Increase reuse by loose coupling

Easy testing

More maintainable code

Dependency injection is a pattern where container passes objects by name to other objects, via either constructors, properties, or factory methods.

The IoC is a general concept, and it can be expressed in many different ways and dependency injection is merely one concrete example of Inversion of Control.

**Types:**

* Constructor- based dependency injection

The dependent object is injected by the container through constructor.

* Setter-based dependency injection

The dependent object is injected by the container through setter.

<bean id=”wheel” class = “model.Wheel”></bean>

<bean id=”car” class = “model.Car”>

<property ref=”wheel” name=”wheelobj”></property>

</bean>

(The value id id attribute given in the <bean> tag && Name of the attribute we have given in the car class).

**Property Tag attributes:**

Ref: The id of another bean to wire to the property

Name: The name of the property to be wired

Value: the value to assign to the property

**Constructor-arg Attributes**:

Ref: the id of another bean to wire to the property

Name: the exact name of the argument in the constructor argument list

Value: the value to assign to the property

Index: the exact index of the argument in the constructor argument list

Type: the exact type of the constructor argument

Example:

<bean id =”car” class=”model.car”>

<constructor-arg ref=”wheel”></constructor-arg>

<constructor-arg ref=”carNo” value=”AP28CA9959”></constructor-arg>

<constructor-arg ref=”carName” value=”Amaze”></constructor-arg>

<constructor-arg ref=”color” value=”White”></constructor-arg>

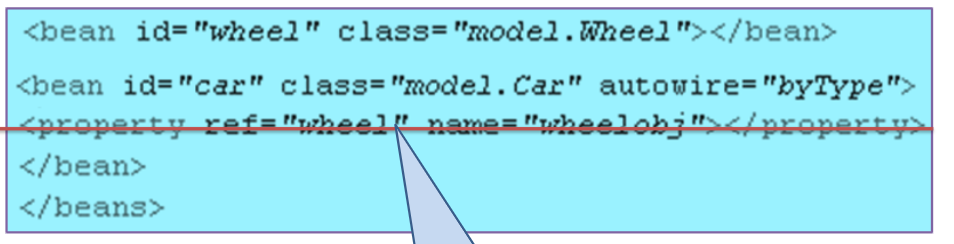
**AutoWiring:**

Allows dependent object to be injected implicitly.

Feature provided by the spring framework to reduce the configuration

Types of auto wiring => byName, byType, constructor

By default, there is no auto wiring.



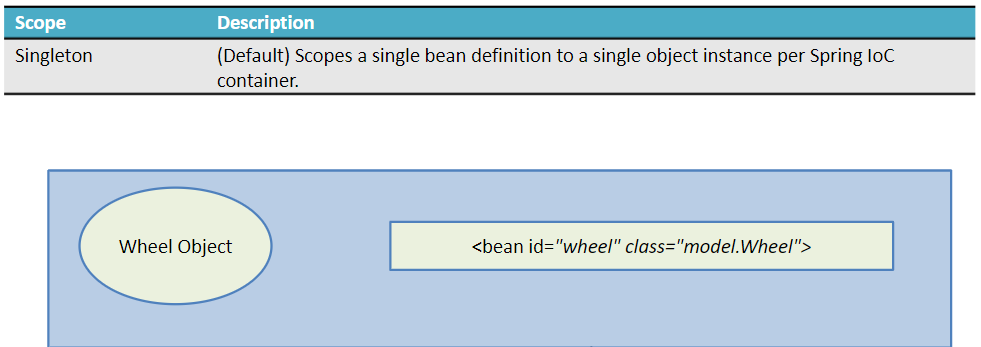
So property tag is not needed.

Points to be noted:

1. ApplicationContext acts as IoC container.
2. Modules of core container are: Beans, Core, Context, SpEL

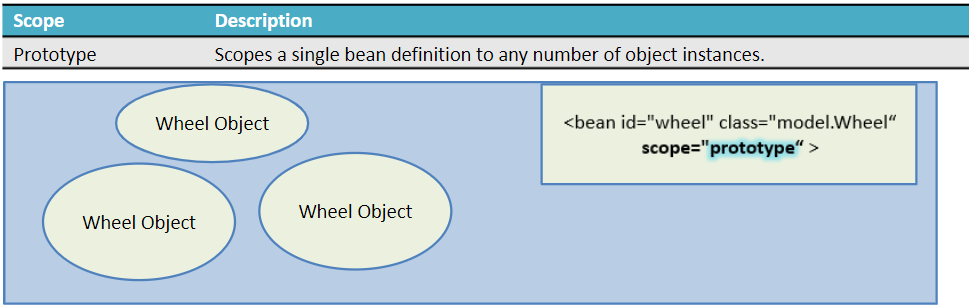
**Bean Scope:**

Singleton and Prototype.



In singleton, only one instance is created and it is shared by all the collaborating objects.

Prototype:



In prototype, for each request a new object is created.

|  |  |
| --- | --- |
| **Early Loading** | **Lazy Loading** |
| Beans get loaded and initialized during container startup | Beans get loaded and initialized only when requested |
| Default | Lazy-init should be set to true in the configuration file for lazy loading. |
| Lazy-int should be set to false in the configuration file for early loading | Ex: context.getBean(“car”); |
| Ex: ApplicationContext context = new ClassPathCmlApplicationContext(“beans.xml”); |  |
| When the request is made for a bean object, the already created object will be returned. |  |

**Annotations:**

Annotations are the meta-data for classes.

Spring uses annotations as an alternative to XML for declarative configuration

Works only from java 1.5 or greater.

**Annotations in Spring:**

* @Component => Marks a java class as a bean and indicates an auto-scan component
* @Scope => To specify the scope of the bean.
* @Autowored => Used to autowire bean on the setter method, constructor or property
* @Value => To set a value.
* @Required => Applied to bean property setter methods; it indicated that the effected bean property must be populated in XML configuration file at configuration time.

Annotation wiring is not turned on by default in the Spring Container.

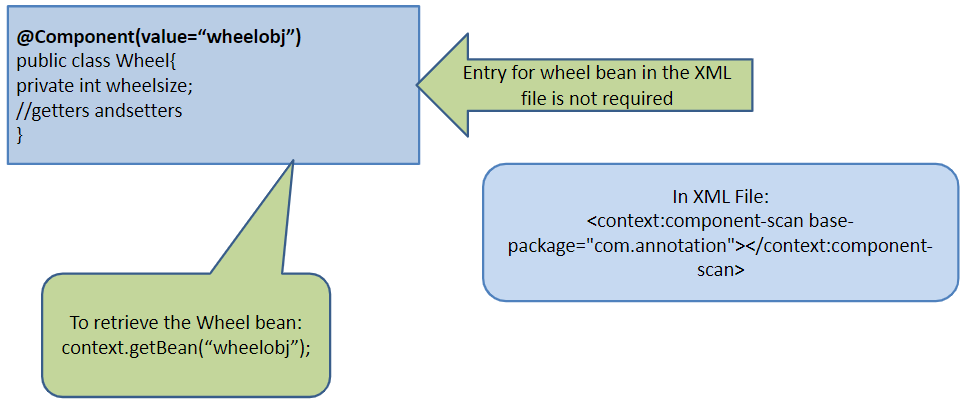
Ex: <context:annotation-config/>

* Activates the annotations in already registered beans in the application context.

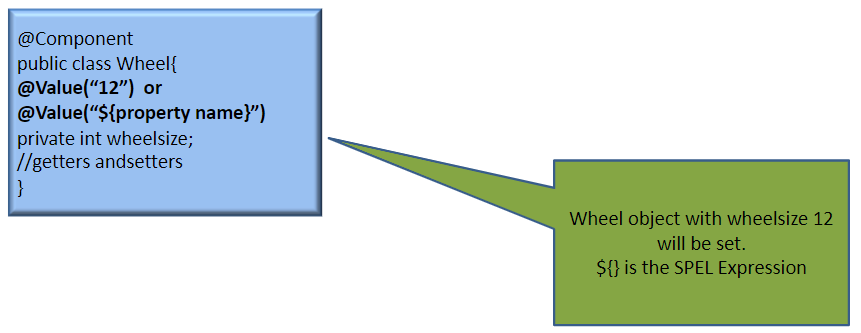
<context:component-scan base-package=”package-name”>

* Scans the packages and registers the beans in application context.

@Component

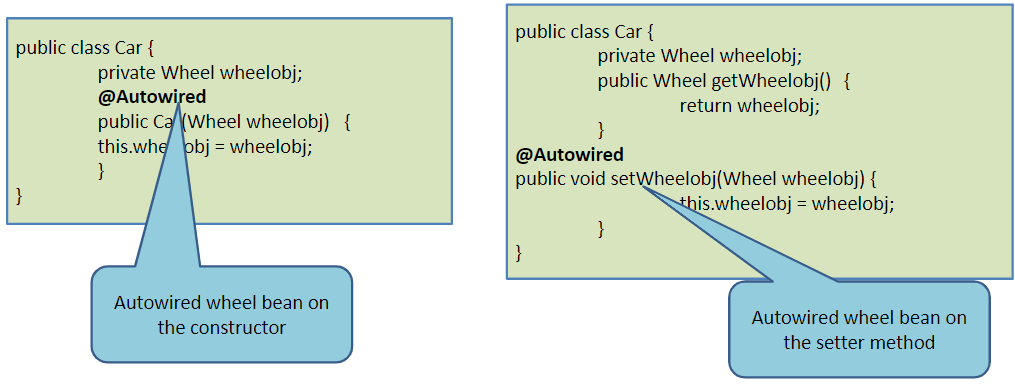


@Value



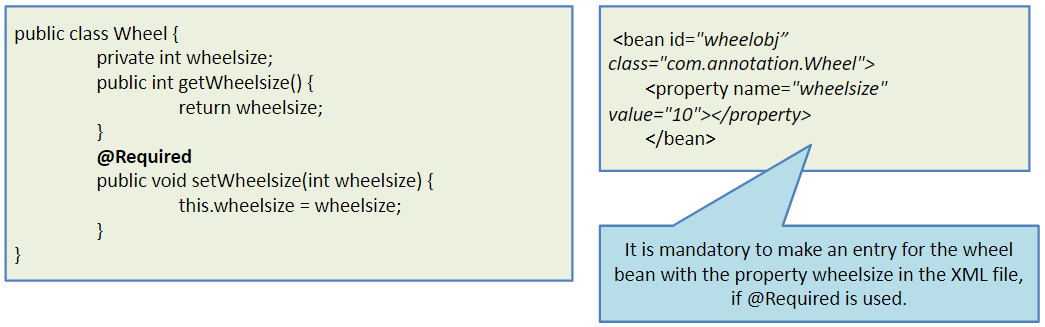
@Autowired

The @Autowired annotation can be used to autowire bean on the setter method, constructor, or a property.



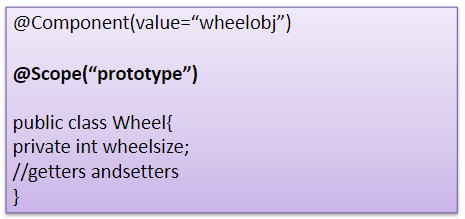
@Required

Specifies that a particular property must be injected or else the configuration will fail.



@Scope

* @Scope is used to specify the scope of the bean
* By default the scope will be a singleton; to change the default, @Scope annotation can be used.



@Configuration:

Indicates that the class can be used by the Spring IoC container as a source of bean definitions.

For Spring to know the packages to be scanned for annotated components.

* @ComponentScan specifies the base packages to scan.
* Any class annotated with @Component will be scanned and registered.