

# Spring Framework Annotations

# DI-Related Annotations

@Autowired

@Bean

@Qualifier

@Required

@Value

@DependsOn

@Lazy

@Lookup

@Primary

@Scope

# @Autowired annotation

**@Autowired** annotation can be used to mark a dependency which Spring is going to resolve and inject. This annotation can be used with a constructor, setter, or field injection.

**@Autowired** has a boolean argument called *required* with a default value of true. It tunes Spring's behavior when it doesn't find a suitable bean to wire. When true, an exception is thrown, otherwise, nothing is wired.

Note, that if we use constructor injection, all constructor arguments are mandatory.

Starting with version 4.3, we don't need to annotate constructors with **@Autowired** explicitly unless we declare at least two constructors.

## Constructor injection:

```
class Car {  
    Engine engine;  
  
    @Autowired  
    Car(Engine engine) {  
        this.engine = engine;  
    }  
}
```

## Setter injection:

```
class Car {  
    Engine engine;  
  
    @Autowired  
    void setEngine(Engine engine) {  
        this.engine = engine;  
    }  
}
```

## Field injection:

```
class Car {  
    @Autowired  
    Engine engine;  
}
```

# @Qualifier annotation

In a situation when you create more than one bean of the same type and want to wire only one of them with a property, you can use the **@Qualifier** annotation along with **@Autowired** to remove the confusion by specifying which exact bean will be wired.

For example, the following two beans implement the same interface:

```
class Bike implements Vehicle {}  
class Car implements Vehicle {}
```

In such cases, we can provide a bean's name explicitly using the **@Qualifier** annotation.

```
@Autowired  
@Qualifier("bike")  
void setVehicle(Vehicle vehicle) {  
    this.vehicle = vehicle;  
}
```

# @Required annotation

The **@Required** annotation applies to bean property setter methods and it indicates that the affected bean property must be populated in XML configuration file at configuration time.

Otherwise, the container throws a `BeanInitializationException` exception.

```
@Required
void setColor(String color) {
    this.color = color;
}
```

```
<bean class="com.test.annotations.Bike">
    <property name="color" value="green" />
</bean>
```

# @Bean annotation

**@Bean** marks a factory method which instantiates a Spring bean:

```
@Bean
Engine engine() {
    return new Engine();
}
```

Spring calls these methods when a new instance of the return type is required.

The resulting bean has the same name as the factory method. To name it differently the name or the value arguments of this annotation can be used (the argument value is an alias for the argument name)

```
@Bean("engine")
Engine getEngine() {
    return new Engine();
}
```

Note, that all methods annotated with **@Bean** must be in `@Configuration` classes.

# @Value annotation

Spring **@Value** annotation is used to assign default values to variables and method arguments.

We can read spring environment variables as well as system variables using **@Value** annotation.

**@Value** annotation also supports SpEL.

Constructor injection:

```
Engine(@Value("8") int cylinderCount) {  
    this.cylinderCount = cylinderCount;  
}
```

Setter injection:

```
@Value("8")  
void setCylinderCount(int cylinderCount) {  
    this.cylinderCount = cylinderCount;  
}
```

Field injection:

```
@Value("8")  
int cylinderCount;
```

Wire values defined in external sources:

\*.properties file: engine.fuelType=petrol

inject the value of engine.fuelType with the following:

```
@Value("${engine.fuelType}")  
String fuelType;
```

# @Primary annotation

Sometimes we need to define multiple beans of the same type. If we mark the most frequently used bean with **@Primary** it will be chosen on unqualified injection points.

```
@Component
@Primary
class Car implements Vehicle {}
```

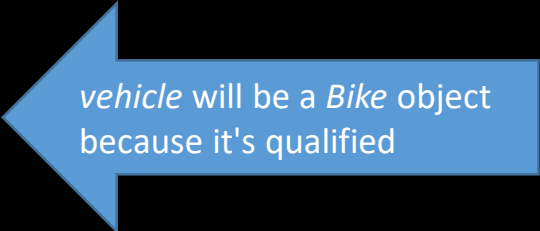
```
@Component
class Bike implements Vehicle {}
```

```
@Component
class Driver {
    @Autowired
    Vehicle vehicle;
}
```



Spring injects a *Car* bean

```
@Component
class Biker {
    @Autowired
    @Qualifier("bike")
    Vehicle vehicle;
}
```



vehicle will be a *Bike* object because it's qualified

# @DependsOn annotation

This annotation can be used to make Spring initialize other beans before the annotated one.

We only need this annotation when the dependencies are implicit, for example, JDBC driver loading or static variable initialization.

**@DependsOn** on the dependent class specifying the names of the dependency beans.

```
@DependsOn("engine")
class Car implements Vehicle {}
```

If we define a bean with the **@Bean** annotation, the factory method should be annotated with **@DependsOn**

```
@Bean
@DependsOn("fuel")
Engine engine() {
    return new Engine();
}
```

# @Lazy annotation

By default, Spring creates all singleton beans eagerly at the startup/bootstrapping of the application context. However, there are cases when we need to create a bean when we request it, not at application startup.

**@Lazy** annotation behaves differently depending on its location place:

- a `@Bean` annotated bean factory method, to delay the method call (hence the bean creation)
- a `@Configuration` class and all contained `@Bean` methods will be affected
- a `@Component` class, which is not a `@Configuration` class, this bean will be initialized lazily
- an `@Autowired` constructor, setter, or field, to load the dependency itself lazily (via proxy)

**@Lazy** annotation has an argument named `value` with the default value of `true`. It is useful to override the default behavior.

For example, marking beans to be eagerly loaded when the global setting is lazy, or configure specific `@Bean` methods to eager loading in a `@Configuration` class marked with **@Lazy**

```
@Configuration
@Lazy
class VehicleFactoryConfig {

    @Bean
    @Lazy(false)
    Engine engine() {
        return new Engine();
    }
}
```



# @Lookup annotation

A method annotated with **@Lookup** tells Spring to return an instance of the method's return type when we invoke it.

Essentially, Spring will override annotated method and use method's return type and parameters as arguments to `BeanFactory#getBean`.

**@Lookup** is useful for:

- Injecting a prototype-scoped bean into a singleton bean (similar to Provider)
- Injecting dependencies procedurally

**@Lookup** is the Java equivalent of the XML element `lookup-method`.

# @Scope annotation

**@Scope** uses to define the scope of a `@Component` class or a `@Bean` definition. It can be either singleton, prototype, request, session, `globalSession` or some custom scope.

```
@Component
@Scope("prototype")
class Engine {}
```