

Spring Framework

Beans Lifecycle and Annotations



Table of Content

- Dependency Injection
- Injection Type
- Dependency Injection with Annotations
- What is the @Autowired
- Autowiring injection types
- Dev Process - Constructor injection with annotations
- Dev Process - Setter injection with annotations
- Dev Process - Field injection with annotations
- Bean Scopes (annotations)
- Bean Lifecycle (annotations)
- Spring Config with Java Class
- Questions
- Thanks

Dependency Injection

is a technique where the dependencies of a class (i.e., the objects it relies on) are injected into the class rather than the class creating or managing its dependencies





give me a Shape

Spring Bean
Container

config file

config file which
contains the beans
defination

Shape

Circle

Rectangle

Triangle

DrawShape

Spring container

give me an object

Object



Injection Types

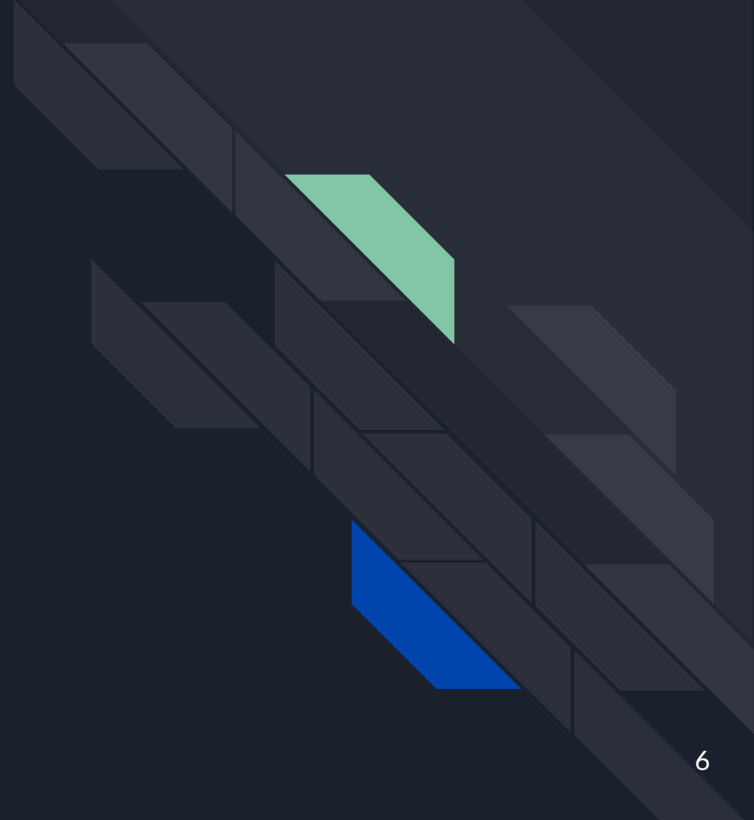
We will cover the two most common (without annotations)

- Constructor Injection
- Setter Injection

We will talk about “auto-wiring” in the Annotations section later



Spring Dependency Injection with Annotations and Autowiring





**Let's see the old code of
constructor injection using XML**



What is Spring AutoWiring?

Spring will look for a class that matches the property

(matches by type: class or interface)

Spring will inject it automatically ... hence it is autowired



Autowiring Injection Types

- Constructor Injection
- Setter Injection
- Field Injections



Dev Process - Constructor Injection (annotations)

1. Define the dependency class
2. Create a constructor in your class for injections
3. Configure the dependency injection with `@Autowired`
Annotation



Step 1: Define the dependency class

```
@Component
public class Draw2D
{
    public void draw(String shapeName) { System.out.println("Drawing a 2d for " + shapeName); }
}
```



Step 2: Create a constructor in your class for injections

```
@Component
public class CircleShape implements Shape
{

    public Draw2D draw2D;

    //constructor injection
    public CircleShape(Draw2D draw2D) { this.draw2D = draw2D; }

    @Override
    public void drawShape2d() { draw2D.draw( shapeName: "circle"); }

}
```



Step 3: Configure the dependency injection with @Autowired Annotation

```
@Component
public class CircleShape implements Shape
{

    public Draw2D draw2D;


    //constructor injection
    @Autowired
    public CircleShape(Draw2D draw2D) { this.draw2D = draw2D; }

    @Override
    public void drawShape2d() { draw2D.draw( shapeName: "circle"); }

}
```



Let's write some code



Dev Process - Setter Injection (annotations)

1. Define the dependency class
2. Create a Setter in your class for injections
3. Configure the dependency injection with `@Autowired`
Annotation



Step 1: Define the dependency class

```
@Component
public class Draw2D
{
    public void draw(String shapeName) { System.out.println("Drawing a 2d for " + shapeName); }
}
```


Step 2: Create a Setter in your class for injections

```
@Component
public class RectangleShape implements Shape
{
    public Draw2D draw2D;

    //setter method for setter injection
    @Autowired
    public void setDrawShapeFor2d(Draw2D drawShape2d) {
        this.draw2D = drawShape2d;
    }

    @Override
    public void drawShape2d() { draw2D.draw( shapeName: "rectangle"); }
}
```



Step 3: Configure the dependency injection with @Autowired Annotation

```
@Component
public class CircleShape implements Shape
{

    public Draw2D draw2D;

    //constructor injection
    @Autowired
    public CircleShape(Draw2D draw2D) { this.draw2D = draw2D; }

    @Override
    public void drawShape2d() { draw2D.draw( shapeName: "circle"); }

}
```



Dev Process - field Injection (annotations)

You can Inject dependencies by setting field values on your

```
@Component
public @Component
    public class TriangleShape implements Shape {
        @Autowired
        public Draw2D draw2D;

        @Override
        public void drawShape2d() { draw2D.draw( shapeName: "triangle"); }
    }
```



Let's write some code



5 minutes Break



Bean Scope with annotations



**Let's see the old code
of bean scope**



Explicitly Specify Bean Scope

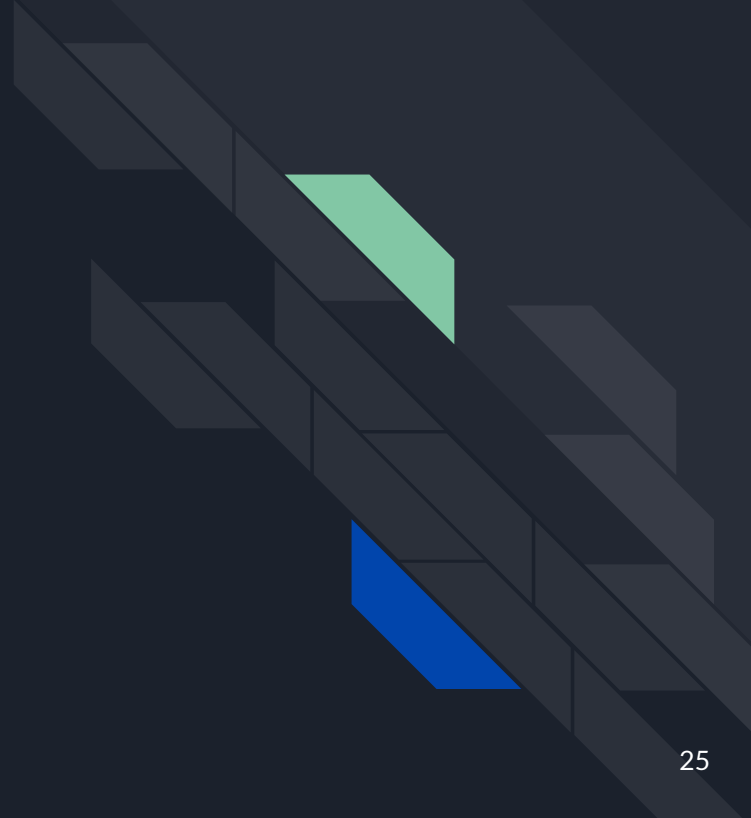
```
@Component
@Scope("singleton")
public class CircleShape implements Shape
{
    int radius;

    public int getRadius() { return radius; }

    public void setRadius(int radius) { this.radius = radius; }
```




Bean Lifecycle with Annotations





Init/destroy: methods configuration

```
//this is the init method  
@PostConstruct  
public void connectToDatabase()  
{  
    System.out.println("the connection to db established .....");  
}  
  
//this is the destroy method  
@PreDestroy  
public void disconnectFromDatabase() { System.out.println("the connection to db ended ....."); }
```

Spring Configuration

File: applicationContext.xml

```
<beans ...>

  <bean id="circle"
        class="com.example1.CircleShape">
  </bean>

</beans>
```

```
@Configuration
@ComponentScan("com.dependencyInjection")
public class Config
{
}

}
```

```
<!-- Enable Component Scanning-->
<context:component-scan base-package="com.usingAnnotations"/>
```




Dev process - Config class (No more XML)

1. Create a Java class and annotate as `@Configuration`
2. Add component scanning support: `@ComponentScan` (optional)
3. Read Spring Java configuration class
4. Retrieve bean from Spring container



1-Create a Java class and annotate as @Configuration

```
@Configuration  
public class Config  
{  
  
}
```



2-Add component scanning support: `@ComponentScan` (optional)

```
@Configuration
@ComponentScan("com.dependencyInjection")
public class Config
{

}
```



3-Read Spring Java configuration class

```
ApplicationContext context = new AnnotationConfigApplicationContext(Config.class);
```



4-Retrieve bean from Spring container

```
Shape circleShape = context.getBean( name: "circleShape" , Shape.class);  
circleShape.drawShape2d();
```




Read properties from properties file (annotations)



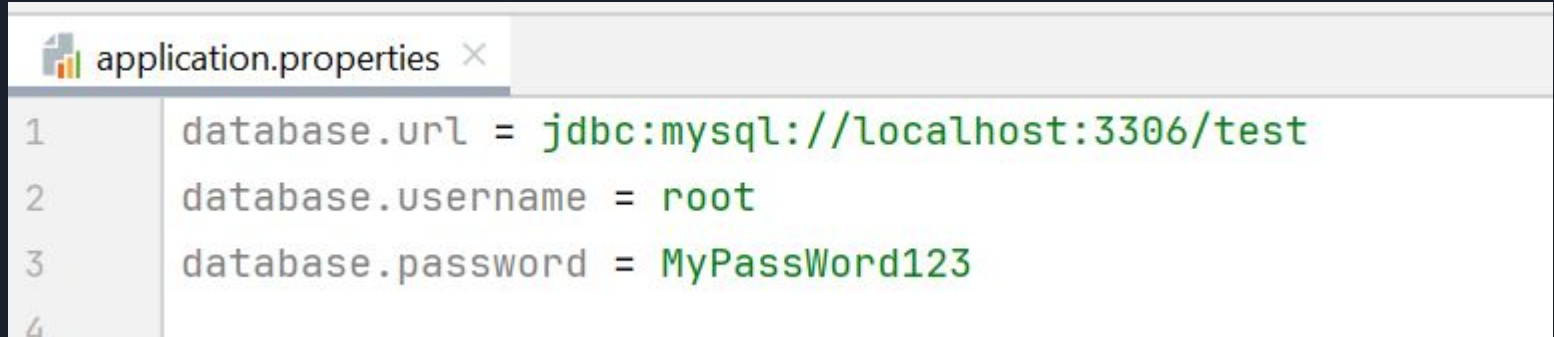
Development Steps

Step 1: Create Properties File

Step 2: Load Properties file in Spring config file

Step 3: Reference Values from Properties File

Step 1: Create Properties File



The screenshot shows a code editor window with a single tab titled 'application.properties'. The editor contains three lines of text, each representing a database connection property. Line numbers 1, 2, 3, and 4 are visible on the left side of the editor. The text is as follows:

```
1 database.url = jdbc:mysql://localhost:3306/test
2 database.username = root
3 database.password = MyPassWord123
4
```




Step 2: Load Properties file in Spring config file

```
@Configuration
@ComponentScan("InjectLiteralsValues")
@PropertySource("classpath:application.properties")
public class Config
{

}
```

Step 3: Reference Values from Properties File

```
@Value("${database.url}")  
private String url;  
  
@Value("${database.username}")  
private String username;  
  
@Value("${database.password}")  
private String password;
```



application.properties

```
1 database.url = jdbc:mysql://localhost:3306/test  
2 database.username = root  
3 database.password = MyPassWord123  
4
```



Questions ?



THANK YOU