Spring Annotations Cheat Sheet

Core Annotations You've Learned

Annotation	What it does	When to use	Target
`@Autowired`	Automatically injects dependencies into your class	When you need Spring to assign an instance of another class	Field/Method
`@Service`	Marks a class as a business logic component	For classes that contain your main application logic.	Class
`@Bean`	Creates and configures objects (factory methods!) Replaces the factory classes you wrote that do manual injection	In `@Configuration` classes to define custom objects for Spring to manage	Method
`@Qualifier`	Specifies which bean to inject when multiple options exist	When you have multiple beans of the same type (usually inheriting from the same class or implementing the same interface)	Field/Method
`@Value`	Injects values from properties files or environment variables	For configuration values like URLs, server names, ports, passwords, or other settings that may change depending on the environment (test, dev, QA, production)	Field/Method
`@Configuration`	Marks a class as a source of bean definitions	For classes that configure your application and contain `@Bean` methods. These replace the XML file if you're using annotations instead of XML.	Class

Annotation	What it does	When to use	Target
`@Primary`	Marks a bean as the preferred choice when multiple candidates exist	When you have multiple beans of the same type and want one as default	Method

Additional Useful Annotations

Annotation	What it does	When to use	Target
`@Component`	Generic stereotype for any Spring-managed component	For general-purpose classes that don't fit @Service, @Repository, or @Controller	Class
`@Repository`	Marks a class as a data access component	For classes that interact with databases or external data sources	Class
`@Controller`	Marks a class as a web controller component	For classes that handle web requests (you'll use this in web apps later)	Class
`@PostConstruct`	Runs a method after all dependencies are injected	For initialization code that needs injected dependencies. We don't create constructors when using annotations, this allows us to specify a void method to handle setup code that would normally be in a constructor.	Method
`@PreDestroy`	Runs a method before the bean is destroyed	For cleanup code (closing files, connections, etc.). This is a void method that will be executed before an object is removed from the heap.	Method

Key Concept: Beans vs Factory Classes

Remember: `@Bean` methods are basically **factory methods** that replace the factory classes you created earlier!

Before Spring (manual factory):

```
public class UserServiceFactory {
    public static UserService createUserService() {
        UserRepository repo = new UserRepository();
        return new UserService(repo);  // Manual injection
    }
}
```

With Spring (@Bean factory method):

```
@Configuration
public class AppConfig {
    @Bean
    public UserService userService() {
        return new UserService(userRepository());  // Spring manages this
    }
}
```

Consuming the bean:

```
@Service
public class OrderService {
    @Autowired
    private UserService userService;  // Spring automatically injects the bean
}
```

Note that in our examples, we gave the Bean a name `@Bean("userService")` and a qualifier with the same name `@Qualifier("userService")`. If we don't provide a name, it will use the name of the method it is attached to as the bean name. Similarly, if the @Autowired field has the same name, in this case, `userService`, Spring will match on the name and figure it out.

I prefer being explicit, but if you want to be lazy, this does work. This is because if we use explicit names, if someone refactors and renames either the field or the method, it won't break.

With Spring (@Bean factory method, explicit):

```
@Configuration
public class AppConfig {
    @Bean("userService")
    public UserService userService() {
        return new UserService(userRepository());  // Spring manages this
    }
}
```

Consuming the bean (explicit):

```
@Service
public class OrderService {
    @Autowired
    @Qualifier("userService")
    private UserService userService;  // Spring automatically injects the bean
}
```

Key Concept: Stereotype Annotations

`@Component`, `@Service`, `@Repository`, and `@Controller` all do the same basic thing - **they tell Spring to manage these classes as beans**.

The difference is just **semantic** - which one you choose depends on what the class is intended to do:

- `@Component` = Generic "this is a Spring-managed class"
- `@Service` = "this class contains business logic"
- `@Repository` = "this class handles data access"
- `@Controller` = "this class handles web requests"

They're all essentially `@Component` with different names for clarity:

java

```java
@Component
public class EmailValidator { ... }  // Generic utility

@Service
public class UserService { ... }     // Business logic

@Repository
public class UserRepository { ... }  // Database access

@Controller
public class UserController { ... }  // Web requests (later)
```

Spring treats them all the same - it creates beans for all of them and allows `@Autowired` injection.