Java configuration annotations

You're developing a Spring Boot app using **Java configuration annotations** — like @PostConstruct, @Inject, etc.

But… these annotations are actually part of **Java’s standard libraries** (called JSE or JEE modules), not created by Spring.

**🚀 So how do they work?**

These annotations are just **tags** or **instructions** on your code.

But those tags won’t do anything by themselves — your **framework (like Spring, Hibernate, or Servlet)** is the one that looks at them and decides what to do.

**🔧 Different frameworks act on them differently:**

* In **Spring**, the Spring Framework reads the annotations and does things like injecting objects or calling methods.
* In **Hibernate**, the Hibernate engine might use them for DB entity lifecycle.
* In **Servlet**, the server might use them to manage servlet lifecycle.

**📚 Let’s break down the examples:**

| **Annotation** | **What it does** | **Like in Spring** |
| --- | --- | --- |
| @PostConstruct | Run this method **after the bean is created and dependencies are injected**. | Like init() method |
| @PreDestroy | Run this method **just before the bean is destroyed or app shuts down**. | Like destroy() method |
| @Inject | **Inject a dependency by type**. | Like @Autowired |
| @Named("myBean") | **Give a name to a bean** or **inject a bean using its name**. | Like @Component("myBean") or @Qualifier("myBean") |
| @Resource(name="myBean") | **Inject a bean by name first**, then fallback to type if name not found. | Like @Autowired + @Qualifier |

**🧠 Simple analogy**

Think of annotations as **stickers** you place on methods or fields:

* 📌 @PostConstruct = “Run this once when ready!”
* 📌 @PreDestroy = “Do this before shutdown!”
* 📌 @Inject = “Give me a helper object automatically.”
* 📌 @Named = “This is my official name.”
* 📌 @Resource = “Find this exact helper by name.”

But who follows the stickers? → **Spring does**, if you’re in a Spring Boot app.

**🎯 Final Words**

You don’t have to use these — Spring has its own versions like @Autowired, @Component, etc.  
But these **standard annotations** work across many Java platforms, so Spring supports them too.

## ✅ **Spring Annotations are used more in practice**

| **Reason** | **Explanation** |
| --- | --- |
| 🧩 **Simpler and more powerful** | Spring annotations like @Component, @Autowired, @Service, etc., are tightly integrated with Spring’s full feature set. |
| 📚 **More documentation and tutorials** | Most examples, courses, and real-world projects use Spring’s annotations. |
| ⚙️ **Better support for advanced features** | Spring annotations support things like profiles, conditional beans, lifecycle hooks, scopes, etc. |
| 🛠️ **Easier debugging and customization** | Spring provides more control using annotations like @Qualifier, @Value, @Configuration, etc. |

## 🎯 Conclusion

**In Spring-based applications, 95%+ of the time, developers prefer and use Spring annotations.**

Java standard annotations are still supported by Spring and used in some cases (especially @PostConstruct), but **Spring's own annotations dominate in the real world**.

### ✅ ****Dependency Injection through**** application.yml ****– Notes****

1. **Industry Use:**
   * Widely used in Spring Boot microservices to inject configuration like database URLs, service endpoints, credentials, and feature flags.
   * Helps manage multiple environments (dev, test, prod) using Spring Profiles (application-dev.yml, etc.).
2. **Advantages:**
   * **Externalized & Environment-Specific Config:** You can keep your configuration outside the codebase and easily switch between environments.
   * **Readable & Structured Format:** YAML is cleaner and more readable than .properties for nested configuration.
   * **Loose Coupling:** Injects configuration into beans via @ConfigurationProperties, avoiding hardcoding and improving flexibility.

Eg: SpringBootDependencyInjectionUsingYml