Spring Bean Lifecycle

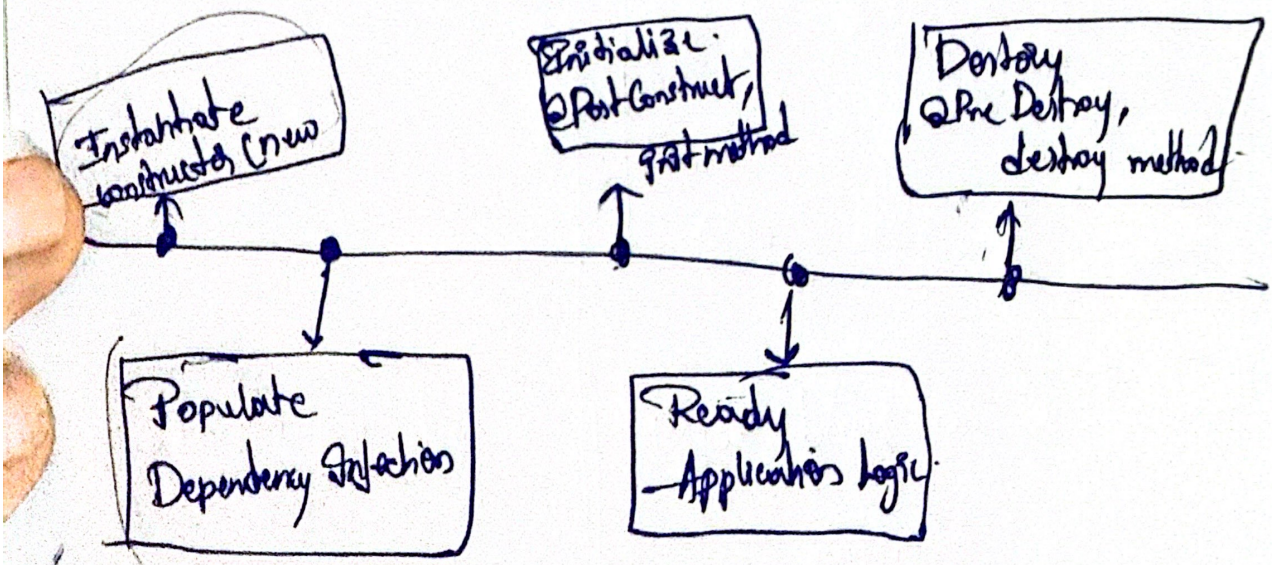
Spring manages beans

Initialization, Injection, Destruction are not "magic"

Knowing the lifecycle helps

- Debugging
- Writing initialization / shutdown logic
- Avoiding creational pitfalls.

Bean Lifecycle Overview



Instantiation

- Constructor called
- Dependencies Injected
- Beans not fully ready yet

Initialization

- container allows bean to run Setup logic.
- options:
  - @PostConstruct
  - init method.
- Initialization bean

Ready State

- Bean fully initialized and ready
- Application logic can use the bean

Triggered on shutdown

options:

@PreDestroy

DisposableBean

destroy method

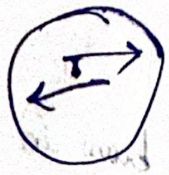
Customizing Beans Before and After Instantiation

why Customize the lifecycle	"Extra" Configuration Adjust or enhance behaviors. Extension points - Before instantiation → change Definitions - After instantiation → enhance beans.
BeanFactoryPostProcessor	Before beans are instantiated. Allows definition modifications Common uses: - change property values - Add missing Configurations

- Dynamically override settings.

Bean Factory Post Processor → Alters before Creation.

Bean Post Processor



- After Beans are instantiated but before use.
- Before initialization and After initialization methods
- Operates on instance

common uses :-

- ↳ Adds Cross Cutting behaviour
- Proxies
- Custom initialization

Hook in creation process of Bean.

Spring Use Case

- AOP
- @Autowired and @Value Inje
- @Configuration
- "Magic" as usually post-proces

Proxies and Bean Creation Order

Why Proxies Matter

- Spring "wraps" beans in a Proxy.
- Proxies enable:
  - Transactions (@Transactional)
  - Security
  - Lazy initialization
  - Custom behavior

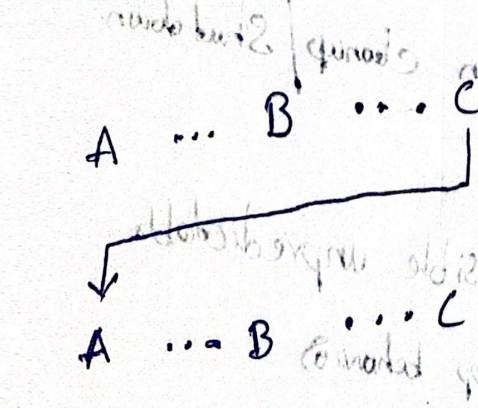
What is a Proxy?

- Wrapper object
- Call interceptor
  - ↳ when you call this bean, the proxy will intercept that call and direct it & shape it upon what you think you want to get back.

Two Strategies!

- JDK Dynamic Proxy
  - → Interface-based
- CGLIB Proxy
  - → Subclass-based.

Why Bean Creation Order Matters



Spring decider order.

Problems when:

- Bean needed before it exist
- cyclic reference.

Controlling Creation Order

- @DependsOn
- @Order
- Lazy initialization

And by default many can use lazy initialization in the code.

Why Injection can be Tricky.

- Multiple beans of same type
- Circular dependencies
- Bean cleanup / Shutdown
- Possible unpredictable app behavior

Multiple Beans of same type

- @Primary → default bean
- @Qualifier → explicit choice.
- Profiles and conditional beans
- Primary and Qualifier tied to the life cycle.

Circular Dependencies

A → B, B → A.

Neither instantiates

Solutions:

- Refactor dependencies
- @Lazy.
- Constructor injection + Circular references - big red flag
- we have to question ourselves if

Cleanup and shutdown

Beans "hold" resources

Cleanup hooks

Options:

- @PreDestroy
- DisposableBean.destroy()
- ContextClosedEvent listener.