🌱 Spring Bean Lifecycle & Scopes – Final Notes

1️⃣ Bean Lifecycle Overview

Spring beans follow a well-defined lifecycle:

Step	Theory	Practical Example (FeesStatusChecker / StudentService)
1. Package Scanning	Spring scans packages (`@SpringBootApplication`) and detects annotations (`@Component`, `@Service`, etc.)	`@Component FeesStatusChecker`, `@Service StudentService` are detected and registered as beans
2. Bean Instantiation	Spring creates bean instance using constructor	`new FeesStatusChecker()` internally; you **don't call new manually**
3. Dependency Injection (Populate Properties)	Spring injects dependencies after object creation	Constructor injection: `public StudentService(StudentRepository repo, FeesStatusChecker checker)`

4. Aware Interfaces (Optional)	`BeanNameAware`, `ApplicationContextAware` allow access to bean name or context	Not used in project; optional
5. BeanPostProcessor (Before Init)	Spring applies logic (e.g., proxies, AOP) **before** init	Happens internally (e.g., Spring Security); no custom code yet
6. @PostConstruct	Runs **after DI** is complete	`@PostConstruct public void init() { System.out.println("FeesStatusChecker ready"); }`
7. InitializingBean.afterPropertiesSet()	Legacy initialization hook after `@PostConstruct`	Not used; `@PostConstruct` preferred
8. Custom Init Method	Defined in XML / @Bean configuration	Not used in Spring Boot project
9. BeanPostProcessor (After Init)	Logic after init, wrapping/proxying	Internal (AOP, transactions)
10. Bean Ready for Use	Fully initialized and usable by other beans	`feesStatusChecker.isFeesPaid(studentId)`

11. Runtime Usage (Business Logic)	Bean executes actual application logic	`StudentService` uses `FeesStatusChecker` to check fees
12. @PreDestroy	Runs **before bean destruction** (singleton only)	`@PreDestroy public void cleanup() { System.out.println("Fees checker destroyed"); }`
13. DisposableBean.destroy()	Legacy destroy hook	Not used
14. Custom Destroy Method	XML/@Bean-based destroy	Not used in Spring Boot

✅ **Complete Flow (Singleton Bean)**:
Scan → Instantiate → Inject Dependencies → @PostConstruct → Ready → Used → @PreDestroy on shutdown

---

2️⃣ Spring Bean Scopes

Scope	Description	Managed By Spring	Notes
Singleton (default)	One shared instance across the entire application	Full lifecycle: creation → DI → `@PostConstruct` → usage → `@PreDestroy`	Thread-safe if stateless; default scope

Prototype	New instance created every time it's requested	Partial lifecycle: creation + `@PostConstruct` only	`@PreDestroy` not called automatically; safe for multi-threaded use
Request	One instance per HTTP request	Managed per HTTP request	Only valid in web-aware Spring contexts (e.g., `@Controller`); useful for request-scoped beans
Session	One instance per HTTP session	Managed per HTTP session	Only valid in web-aware Spring contexts; preserves state across multiple requests in same session
Application	One instance per ServletContext (shared across all requests & sessions)	Managed per web application lifecycle	Useful for shared, app-wide beans in web applications
Websocket	One instance per WebSocket session	Managed per WebSocket session	Rarely used; for real-time apps requiring WebSocket session-scoped beans

---

3️⃣ Annotation-Based Bean Definitions

Annotation	Layer / Purpose
`@Component`	Generic bean

`@Service`	Service / business layer
`@Repository`	DAO / data access layer
`@Controller / @RestController`	Web / API layer

Dependency Injection:

- Automatic with `@Autowired`
- **Prefer constructor injection** for testability & immutability

---

4️⃣ Other Bean Configuration Methods

Method	Usage / Notes
Java-based	`@Configuration` + `@Bean` Example: `java @Bean public MyBean myBean() { return new MyBean(); }`
XML-based	`<bean id="myBean" class="com.example.MyBean"/>` Less common now
Programmatic / Runtime Registration	Register beans dynamically using `ApplicationContext` Useful for runtime-dependent beans

---

5️⃣ Quick Summary

- **Singleton** → shared instance, fully managed, thread-safe if stateless
- **Prototype** → new instance per request, partially managed
- **Annotations (`@Component`, `@Service`, etc.)** → simple, declarative, automatic DI
- **Java/XML/Programmatic config** → alternative ways to define beans; programmatic useful for dynamic cases
- **Bean Lifecycle Mapping:**
  Scan → Instantiate → Inject → @PostConstruct → Ready → Use → @PreDestroy