Spring

Terminology

→ Tight coupling & Loose coupling

→ IOC container

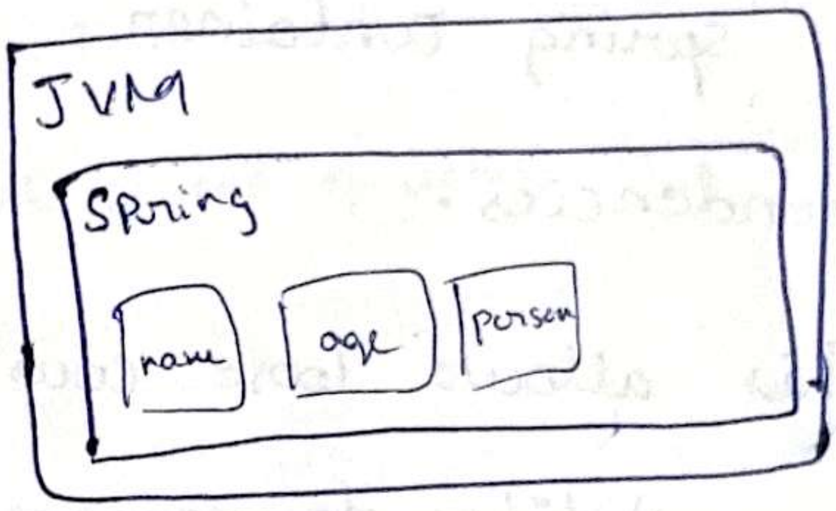
→ Application Context

→ Component Scan

→ Dependency Injection

→ Spring Beans

→ Auto wiring

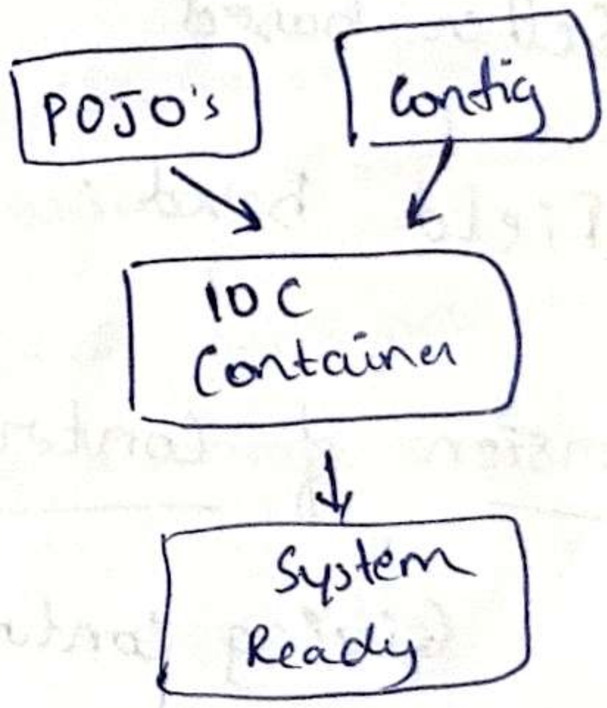


Spring Container / IOC Container.

which manages spring Beans and their life cycle.

1. Bean Factory → Basic spring container.

2. Application Context → advanced spring container with enterprise-specific features.



Annotations

@Bean → used over a method to declare the return object as bean
→ Default bean name is method name.

@Primary → declare as primary bean when more than 1 bean found with same name or type

@Qualifier → when a specific bean is supposed to be autowired
→ Takes priority over @primary

@Component → Class level annotation which creates a ~~be~~ spring bean for the class.

@Configuration → class level annotation indicating the source of bean definitions.

Dependency Injection:

→ It is a fundamental aspect of spring framework through which spring container injets objects into other objects as dependencies.

→ This allows loose coupling of components and moves the responsibility of managing components onto the container.

3 Types of dependency injection

1. Constructa based.
2. Setter based
3. Field based.

Inversion of Control:

Giving control of creating and instantiating the spring beans to the IOC container.

→ In an inversion of control scenaria, framework is incharge of creating instances of a class.

Bean Initialization

1. Eager → Default
   → errors in configuration are identified early during startup.
   → ~~Creates~~ initializes beans at the start of application

2. Lazy → can be configured using @lazy annotation.
   → ~~Creates~~ beans when the bean is called or used.
   ↳ initiates.
   Can be used over @Configuration class. Then all beans inside configuration class will be lazy initialized.
   Lazy resolution proxy will be injected instead of actual dependency.

Factor	Lazy	Eager
Initialization	when its 1st made use in the application	At the startup of the application
Default	not default	Default
~~Usage~~ Code	@Lazy (a) @Lazy(value=true)	@Lazy(value=false) a absence of annotation
If errors	Errors will result in runtime exceptions	Errors will prevent application startup
Usage	rarely used	frequently
Memory Consumption	Less (until bean is initialized)	All beans initialized at startup
Recommended Scenario	Beans that are rarely used	Most of the beans

Bean Scopes

1. Singleton → default - one object instance per spring IOC container

2. Prototype - Possibly many object instances per spring IOC container.

Factor	Prototype	Singleton
Instances	Many per spring IOC container.	One per spring IOC container.
Beans	New bean instance created every time bean is refered	Same Bean instance re used
Default	non-default	Default
Code	@Scope(value = ConfigurableBeanFactory.SCOPE_PROTOTYPE	@scope(value = ConfigurableBeanFactory.SCOPE_SINGLETON)
Usage	Rarely used	frequently.
Recommended scenario	Stateful Beans	stateless beans

Spring Framework contains multiple Spring Modules.

1. Fundamental Features
   → Core (IOC container, Dependency injection, autowiring---)
2. Web
   → Spring MVC (web applications, REST API).
3. Web Reactive
   → Spring webflux etc
4. Data Access
   → JDBC, JPA etc
5. Integration
   → JMS etc
6. Testing
   → Mock Objects, Spring MVC test etc

Spring Evolves through spring projects

1. First Project - Spring Framework.
2. Spring Security - secure application
3. Spring Data - integrate with databases
4. Spring Integration - Address challenges with integration with other applications.
5. Spring Boot - popular framework to build MVC
6. Spring Cloud - build cloud native applications.

Annotations

@Configuration — Indicates that a class declares one or more @Bean methods and may be processed by the spring container to generate bean definitions

@ComponentScan - Defines specific packages to scan for component or beans. If package not defined, scanning starts from current package of the annotated class.

@Bean - Indicates that a method produces a bean to be managed by the spring container

@Component - Indicates an annotated class as a Bean or component.

@Service - Specilization of @component indicated the class has business logic

@Controller - Specilization of @Component indicating the class is a controller.
Used to define controllers in your web applications and REST API

@Repository - Specilization of @Component indicating the class is used to retreive and manipulate data.

→ @Service, @Controller, @Repository are called stereotype annotations

@Primary - Indicates that a Bean should be given preference when multiple candidates are qualified to autowire a single valued dependency

@Qualifier - Used on a field & parameter as a qualifier for candidate beans when autowiring
- This takes priority over @Primary while autowiring

@Lazy - Indicates that a bean has to be lazily initialized
Absence of @Lazy will lead to eager initialization

@Scope (value = ConfigurableBeanFactory.SCOPE_PROTOTYPE)
- Defines a bean to be prototype ie. new instance will be created every time bean is referred.

@PostConstruct - Identifies the method that should be executed after dependency injection is done.

@PreDestroy - Identifies the method that will recieve callback notification to signal that the instance is in the process of being removed by the container. It is typically used to release resources that it has been holding

@Named - Jakarta Contexts & Dependency injection (CDI)
Annotation similar to @Component

@Inject - Jakarta Contexts & Dependency Injection (CDI) annotation
similar to @autowired

SPRING BOOT

Spring Boot helps in building Production Ready application quickly

→ Build Quickly

* Spring Initializer
* Spring Boot starter projects — Convinient dependency descriptors for diff features
* Spring Boot Auto Configuration
* Spring Boot Devetools

→ Be Production Ready

* Logging
* Different configurations for different env's
  - Profiles, configuration properties
* Monitoring (spring Boot Actuator)

Auto Configuration: (spring Boot - autoConfigure .jar)

Automated configuration for your application.

Decided based on

→ which frameworks are in the classpath

→ existing configurations (Annotations etc)

Dev tools

→ <artifactId> spring-boot-devtools </artifactId>

→ Auto restart for every code-change without manual restart the server

→ If the change is to pom-xml, developer has to restart the server manually.

Profiles

→ Configurations differ for each env.

→ profiles enables you to provide environment specific configeration

→ when no profile enabled, application makes use of default properties ie application.properties.

→ Spring.profiles.active = prod

→ If the above property is configured, spring ~~search~~. takes properties from application-prod.properties file

Actuator

→ To monitor & mangge your application in production

→ Provides a number of end points. Few of them

* beans - list of spring beans in the app
* health - app health
* metrics - app metrics
* mappings - details around request mappings.

→ <artifactId> spring-boot-starter-actuator </artifactId>

→ To configure all a more endpoints, add a property in application.properties.

→ management.endpoints.web.exposure.include = *

Spring Boot Components:

1. Spring Boot starters
2. Spring Boot Auto-configurator
3. Spring Boot CLI
4. Spring Boot Actuator.