# Java Spring

**28/12**

Spring IOC (invert of control) cốt lỗi trong spring framework. IoC container chịu trách nhiệm quản lý vòng đời và các phụ thuộc của các đối tượng trong ứng dụng.

Có 2 loại container:

+ BeanFactory: root interface, Container cơ bản, chỉ cung cấp các tính năng cốt lõi của IoC.

+ ApplicationContext child interface of BeanFactory, Mở rộng từ BeanFactory, bổ sung nhiều tính năng như event handling, internationalization, và tích hợp với Spring AOP.

Spring Bean: Spring Bean is nothing special,

any object in the Spring framework that we initialize through Spring container is called Spring Bean

@Bean: Dùng trên phương thức, Không tự động, cần cấu hình thủ công

@Component: Dùng trên lớp, Được phát hiện tự động qua component scanning



**29/12**

@Autowired: Tự động inject bean dựa trên kiểu dữ liệu.

@Qualifier: Chỉ định cụ thể bean nào cần inject khi có nhiều bean cùng kiểu.

**31/12**

**Todo:Hiểu đầy đủ về IoC và DI**

Kết luận

* Spring IoC là nguyên tắc lập trình trong đó luồng điều khiển ứng dụng ko được quyết định bởi ứng dụng mà được quyết định bởi một framework bên ngoài.
* Spring IoC Container là công cụ hiện thực hóa nguyên tắc này.
* Bạn có thể hiểu rằng Spring IoC là "cái gì" (What), còn Spring IoC Container là "cách" thực hiện (How).

**Tight Coupling** (Kết nối chặt chẽ) -> Khó mở rộng và bảo trì.

**Loose Coupling** (Kết nối lỏng lẻo) giảm sự phụ thuộc -> Dễ mở rộng và bảo trì.

DI(Dependency Injection) giúp thực hiện **loose coupling** bằng cách inject các dependency vào đối tượng thông qua constructor, setter, hoặc interface injection.

**03/01**

**Annotation:** chú thích

**@Component annotation** đánh dấu trên các class để cho biết chúng là các bean được quản lý bởi Spring Boot

**@Autowired annotation:** inject các dependency vào các thành phần khác EX:  
@Autowired

Private MessageService messageService

Spring Bean: những object được quản lý bởi Spring IoC gọi là Bean

Cách tạo bean:

* Dùng @ Component: @Repository, @Controller, @Service
* Dùng @Bean trong class có @Configuration

**@Configuration**

* Là một annotation đánh dấu **trên 1 class** chứa **các thông tin cấu hình** cho ứng dụng.
* Spring Boot sẽ tìm và quét các class được đánh dấu @Configuration để tạo và quản lý các bean.

**@Bean**: là một annotation đánh dấu **trên một method** trong **class được đánh dấu @Configuration**. Nó cho biết method đó tạo và trả về một bean và Spring Boot nên quản lý bean đó trong ứng dụng

Spring Bean Scopes:

+ Singleton (Mặc định)

Mô tả: Chỉ một instance của bean được tạo và chia sẻ trong toàn bộ IoC Container.

Khi nào sử dụng: Khi bean cần dùng chung cho toàn bộ ứng dụng

+ Prototype

Mô tả: Mỗi lần yêu cầu bean, một instance mới sẽ được tạo.

Khi nào sử dụng: Khi cần các instance độc lập, không chia sẻ.

--------------- EX---------------

@Scope("prototype")

@Component

public class MyPrototypeBean {

// Một instance mới sẽ được tạo mỗi lần yêu cầu

}

------------------------------

+ Request (Chỉ dùng trong ứng dụng web)

Mô tả: Một bean được tạo cho mỗi HTTP request.

Khi nào sử dụng: Khi một bean cần gắn với vòng đời của một request cụ thể.

+ Session: Khởi tạo instance cho một HTTP Session

+ Application: Khởi tạo instance cho một vòng đời của ServletContext

+ WebSocket: Khởi tạo instance cho một Websocket Session

* Dùng @Scope vd: @Scope(“singleton”)

# 04/01

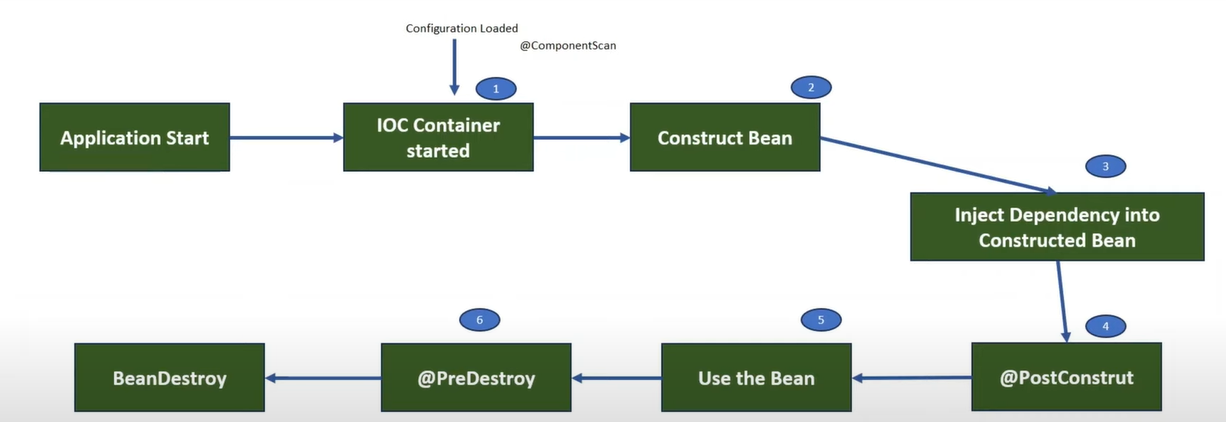
1. Component scan: cho phép Spring Boot tự động tìm kiếm và quản lý các bean trong ứng dụng. Default Spring Boot sẽ quét toàn bộ các package và các package con của package chứa class chứa hàm main
2. Thay đổi phạm vi component scan
   1. @ComponentScan(“…”) chỉ quét các bean nằm trong package được khai báo trong …
   2. SpringApplication(scanBasePackage =”…”)
3. Lazy-initialized Beans
   1. Eager(háo hức): Bean được tạo ngay khi run app

Vd: singleton scope

* 1. Lazy(lười): Bean được tạo ra khi chúng ta gọi tới nó.

Vd: prototype scope hoặc sử dụng (@Lazy)

1. Lifecycle của Bean

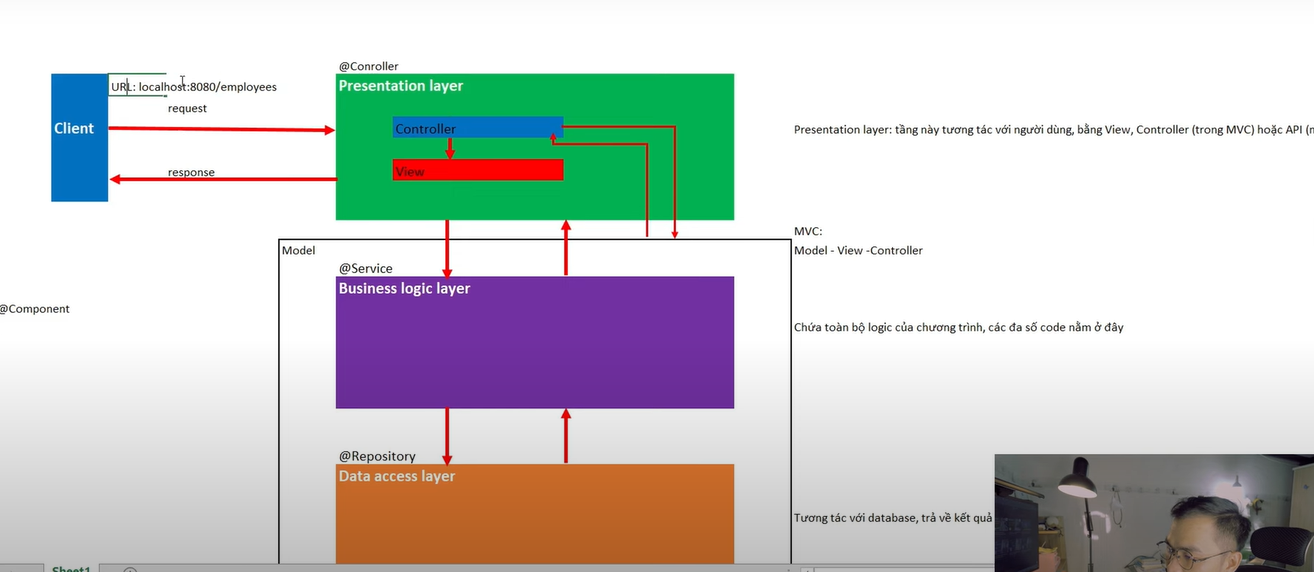


1. Bean name:
   1. context.getBeanDefinitionNames();
   2. @Primary: đánh dấu 1 bean là ưu tiên và luôn được chọn trong trường hợp có nhiều bean cùng loại trong context
   3. @Qualifier: xác định tên của 1 bean chỉ định để inject

# 06/01

**Kiến trúc Spring boot, Hiểu luồng chạy của ứng dụng**

@Controller, @Service, @Repository là biến thể của @Component và đều tạo Bean



# 07/01

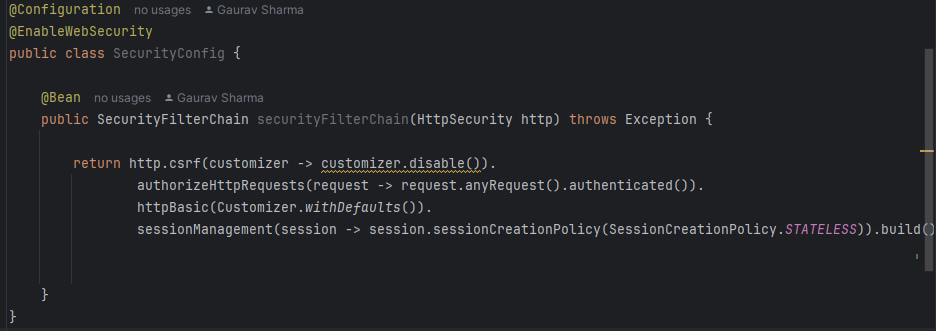
**Spring AOP (Aspect-Oriented Programming) lập trình hướng khía cạnh**

**Aspect. Log before hoặc after khi. Sử dụng để đo hiệu năng**

Youtube: <https://www.youtube.com/watch?v=jx3kyeANTBc>

# 19/01

**Seciruty config**



**Cấu hình từng phần**

**a) .csrf(customizer -> customizer.disable())**

* **Mô tả**: Vô hiệu hóa **CSRF Protection** (Cross-Site Request Forgery).
* **Lý do sử dụng**: CSRF thường được vô hiệu hóa trong các ứng dụng RESTful API, vì chúng không dựa vào cookies để duy trì phiên người dùng mà thường sử dụng các cơ chế khác như **JWT (JSON Web Token)**. Việc vô hiệu hóa CSRF giúp tránh những lỗi không cần thiết khi xử lý các yêu cầu API.

**b) .authorizeHttpRequests(request -> request.anyRequest().authenticated())**

* **Mô tả**: Cấu hình yêu cầu rằng **tất cả các request** gửi đến ứng dụng phải được **xác thực**.
* **Giải thích chi tiết**:
  + anyRequest() chỉ định tất cả các yêu cầu HTTP (không phân biệt URL hoặc phương thức HTTP).
  + .authenticated() yêu cầu người dùng phải xác thực trước khi truy cập bất kỳ tài nguyên nào.

**c) .httpBasic(Customizer.withDefaults())**

* **Mô tả**: Kích hoạt xác thực **HTTP Basic Authentication**.
* **Giải thích chi tiết**:
  + HTTP Basic Authentication là một cơ chế xác thực đơn giản, trong đó thông tin tài khoản (username và password) được mã hóa base64 và gửi trong tiêu đề HTTP Authorization.
  + Đây là cơ chế thường được dùng trong các API đơn giản hoặc nội bộ.

**d) .sessionManagement(session -> session.sessionCreationPolicy(SessionCreationPolicy.STATELESS))**

* **Mô tả**: Cấu hình quản lý phiên của Spring Security thành **stateless**.
* **Giải thích chi tiết**:
  + SessionCreationPolicy.STATELESS chỉ định ứng dụng sẽ không sử dụng **HttpSession** để lưu trạng thái của người dùng.
  + Điều này phù hợp với các ứng dụng RESTful API, nơi mỗi yêu cầu được xử lý độc lập và không phụ thuộc vào thông tin phiên.
  + Khi kết hợp với httpBasic, ứng dụng sẽ yêu cầu xác thực mỗi khi gửi yêu cầu, vì không có phiên nào được lưu lại.

**3. Tổng kết**

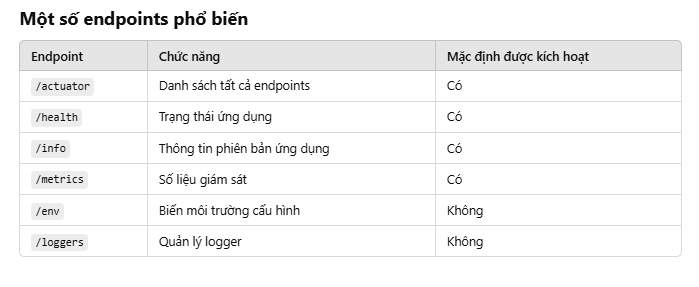
Đoạn mã trên cấu hình bảo mật cho một ứng dụng REST API với các đặc điểm sau:

1. **CSRF bị vô hiệu hóa**: Để phù hợp với các API không sử dụng cookies.
2. **Tất cả các yêu cầu đều cần xác thực**: Bảo vệ toàn bộ tài nguyên.
3. **Sử dụng HTTP Basic Authentication**: Đơn giản hóa việc xác thực (chỉ phù hợp với ứng dụng nhỏ hoặc trong nội bộ).
4. **Stateless Session**: Không lưu trạng thái của người dùng giữa các yêu cầu, thích hợp cho RESTful API.

# 06/02

# Spring Boot Actuator

Giám sát, quản lý và kiểm tra tình trạng hoạt động của ứng dụng một cách dễ dàng. Nó cung cấp các **RESTful endpoints** cho phép bạn thu thập thông tin về hiệu suất, cấu hình, trạng thái và hoạt động của ứng dụng.



**Spring Boot Starters là gì?**

Spring Boot Starters là tập hợp các **gói thư viện (dependencies)** được cấu hình sẵn, giúp bạn dễ dàng tích hợp và sử dụng các tính năng phổ biến của Spring Boot mà không cần tự tay quản lý từng dependency riêng lẻ.

**Lợi ích của Spring Boot Starters**

1. **Tiết kiệm thời gian:** Không cần tìm kiếm và cấu hình từng dependency.
2. **Quản lý dependency dễ dàng:** Giảm thiểu xung đột giữa các thư viện.
3. **Tích hợp liền mạch:** Các Starter được kiểm thử kỹ và hoạt động tốt cùng nhau.
4. **Cấu hình tự động:** Spring Boot có khả năng tự động phát hiện các Starter và cấu hình mặc định cho ứng dụng.

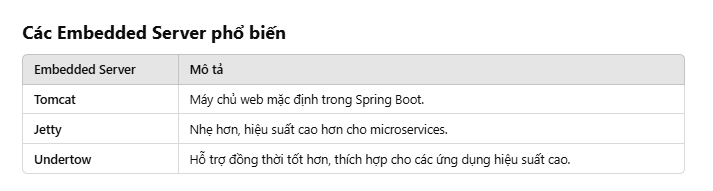


**Embedded Server**

**Embedded Server (Máy chủ nhúng)** trong Spring Boot là một máy chủ web được tích hợp trực tiếp bên trong ứng dụng, cho phép bạn chạy ứng dụng Spring Boot như một tệp **JAR** độc lập mà không cần cài đặt thêm máy chủ bên ngoài như Tomcat, Jetty hoặc Undertow.

**Lợi ích của Embedded Server**

1. **Triển khai dễ dàng:** Không cần cấu hình và cài đặt riêng máy chủ web.
2. **Độc lập:** Ứng dụng có thể được đóng gói và chạy trực tiếp bằng lệnh java -jar.
3. **Tính linh hoạt:** Hỗ trợ nhiều loại máy chủ (Tomcat, Jetty, Undertow).
4. **Tùy chỉnh đơn giản:** Có thể cấu hình cổng, thread pool hoặc thêm SSL một cách dễ dàng.



# 08/02

# Spring Data JPA

Spring Data JPA là một **module của Spring Data** giúp đơn giản hóa việc truy xuất, thao tác và quản lý dữ liệu trong ứng dụng Java thông qua JPA (Java Persistence API). Thay vì phải viết thủ công các câu lệnh SQL phức tạp, Spring Data JPA cung cấp một **API trừu tượng** giúp bạn thao tác với cơ sở dữ liệu dễ dàng hơn.

**Các tính năng chính của Spring Data JPA**

1. **CRUD tự động:** Hỗ trợ các phương thức mặc định như save(), findById(), delete() mà không cần viết SQL.
2. **Query Methods:** Cho phép định nghĩa truy vấn theo tên phương thức, ví dụ: findByUsername().
3. **Hỗ trợ JPA Query Language (JPQL):** Có thể viết các truy vấn phức tạp bằng JPQL hoặc Native SQL.
4. **Pagination & Sorting:** Dễ dàng phân trang và sắp xếp dữ liệu với Pageable và Sort.
5. **Auditing:** Hỗ trợ tự động ghi nhận thông tin tạo, cập nhật bản ghi.
6. **Tích hợp Hibernate:** Mặc định Spring Boot sử dụng Hibernate làm implementation của JPA.

# 09/02

# Spring Data Mongodb

Spring Data MongoDB là một module trong hệ sinh thái **Spring Data** giúp bạn tương tác dễ dàng với cơ sở dữ liệu **MongoDB**.

**Tính năng chính của Spring Data MongoDB**

1. **CRUD tự động:** Hỗ trợ sẵn các phương thức như save(), findById(), delete(), v.v.
2. **Query Methods:** Tự động tạo truy vấn từ tên phương thức như findByName().
3. **Hỗ trợ Native Queries:** Cho phép viết truy vấn MongoDB gốc.
4. **Mapping linh hoạt:** Chuyển đổi tự động giữa tài liệu MongoDB và đối tượng Java (POJOs).
5. **Tích hợp dễ dàng:** Tương thích tốt với các module khác như Spring Boot, Spring Security.

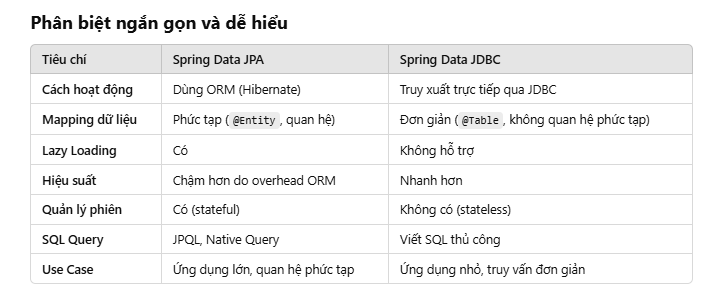
# Spring Data JDBC

**Spring Data JDBC là gì?**

Spring Data JDBC là một module trong **Spring Data** cung cấp giải pháp đơn giản và hiệu quả để làm việc với **cơ sở dữ liệu quan hệ (RDBMS)** mà không cần sử dụng ORM phức tạp như Hibernate hoặc JPA.

**Đặc điểm nổi bật của Spring Data JDBC**

1. **Nhẹ nhàng hơn JPA:** Không sử dụng Hibernate, tránh các chi phí phức tạp liên quan đến ORM.
2. **Mapping đơn giản:** Ánh xạ dữ liệu trực tiếp giữa bảng cơ sở dữ liệu và đối tượng Java (Entity) mà không cần sử dụng annotations phức tạp.
3. **Hỗ trợ CRUD tự động:** Các thao tác như save(), findById(), delete() đều có sẵn.
4. **Tập trung vào truy vấn thuần JDBC:** Hỗ trợ viết truy vấn SQL thủ công dễ dàng.
5. **Không quản lý phiên (Session Management):** Mỗi truy vấn là độc lập, không giống như cách Hibernate hoạt động.



# JPA Test

**JPA Test** trong Spring Boot được sử dụng để kiểm thử các thao tác CRUD và logic liên quan đến **persistence layer** (tầng lưu trữ) khi làm việc với JPA (Java Persistence API). Điều này giúp đảm bảo dữ liệu được lưu trữ và truy xuất đúng cách trong cơ sở dữ liệu.

# Mock MVC

**MockMvc** là một công cụ quan trọng trong **Spring Boot** dùng để kiểm thử các API mà không cần khởi động toàn bộ ứng dụng (không cần chạy server thực tế).

# @SpringBootTest

@SpringBootTest là một annotation trong Spring Boot dùng để **kiểm thử toàn bộ ứng dụng**, khởi chạy cả **Spring Context** và tích hợp đầy đủ các bean, giống như khi ứng dụng chạy thực tế.

**Khi nào nên dùng @SpringBootTest?**

* Khi bạn muốn **kiểm thử tích hợp (Integration Test)** thay vì chỉ kiểm thử đơn vị (Unit Test).
* Khi cần kiểm thử **nhiều tầng (Repository, Service, Controller) cùng lúc**.
* Khi bạn cần Spring Boot tự động cấu hình môi trường kiểm thử.

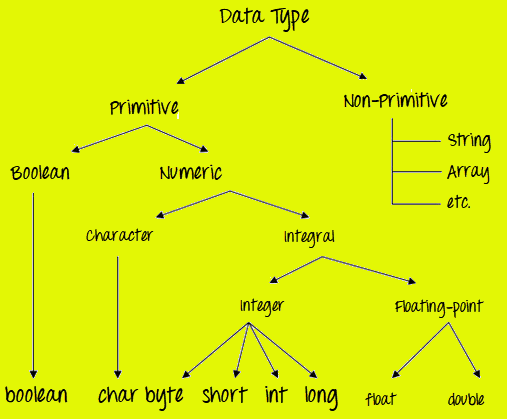
✅ **@SpringBootTest phù hợp khi cần kiểm thử tích hợp toàn bộ ứng dụng, bao gồm Service, Repository và Controller.**  
❌ **Không nên dùng nếu chỉ kiểm thử đơn vị vì nó khởi chạy toàn bộ ứng dụng, tốn nhiều tài nguyên.**

# **@MockBean Annotation**

@MockBean là một annotation trong **Spring Boot Test** được sử dụng để tạo **mock objects** cho các bean trong Spring Context. Nó giúp kiểm thử dễ dàng hơn bằng cách thay thế **bean thực** bằng **bean giả (mock)** mà không cần khởi chạy toàn bộ ứng dụng.

✅ **Dùng @MockBean khi kiểm thử Service hoặc Controller mà không muốn gọi đến Database hoặc API thật.**  
✅ **Hữu ích khi dùng với @SpringBootTest hoặc @WebMvcTest để kiểm thử Spring Bean.**  
❌ **Không nên dùng nếu chỉ kiểm thử Unit Test đơn giản (dùng @Mock thay thế).**

# Java basic



# Static variable

 **Biến static thuộc về class, không thuộc về object.**

 **Tất cả object dùng chung 1 biến static, không có bản sao riêng.**

** Có thể truy cập biến static mà không cần tạo object.**

** Dùng khi cần lưu giá trị chung cho tất cả object.**

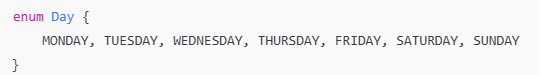


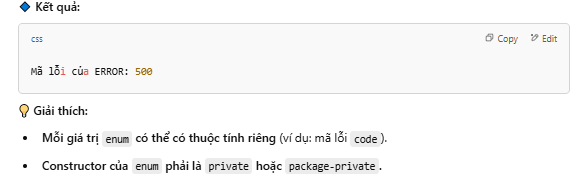
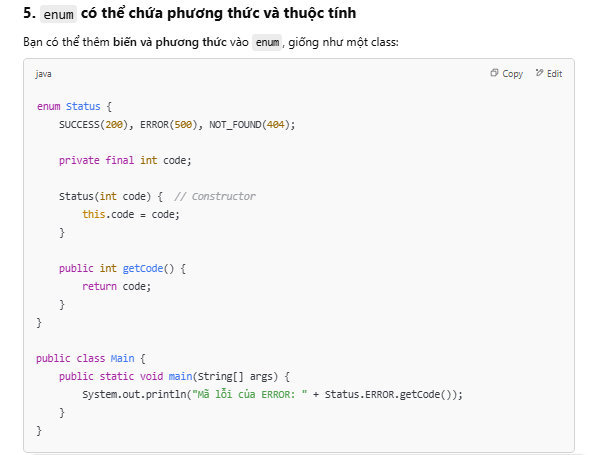
# Enum

 enum giúp **định nghĩa nhóm giá trị cố định** dễ dàng và an toàn.

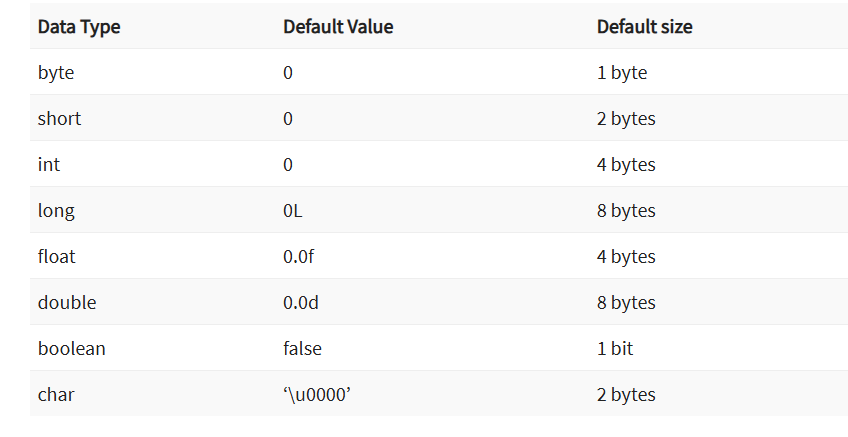
 Có thể dùng với switch-case, có thuộc tính và phương thức như class.

 Giúp mã nguồn **sạch, dễ bảo trì** hơn so với final static.





# Data type



✅ **Giá trị mặc định của int là 0, nhưng chỉ áp dụng cho biến instance (field của class) hoặc static.**  
❌ **Biến cục bộ (local variable) không có giá trị mặc định, phải khởi tạo trước khi dùng.**

# Conditionals

Stream: Dùng để tạo một **luồng xử lý dữ liệu** (Stream API), giúp thực hiện các thao tác như **lọc, biến đổi, gộp dữ liệu**. stream() **cho phép thao tác nâng cao** như map(), filter(), reduce().

Map: Dùng để **biến đổi** từng phần tử trong danh sách mà không làm thay đổi danh sách gốc.

filter() **– Lọc phần tử**

Dùng để **loại bỏ** những phần tử không thỏa mãn điều kiện.

reduce() **– Gộp dữ liệu**

Dùng để **tính toán hoặc gộp tất cả phần tử trong danh sách thành một giá trị duy nhất**.



# Types of Method

Trong Java, có **2 loại phương thức chính**:

* **Predefined Methods** (Phương thức có sẵn) (Math.sqrt(), String.length, System.out.println()…)
* **User-defined Methods** (Phương thức do người dùng tự định nghĩa)

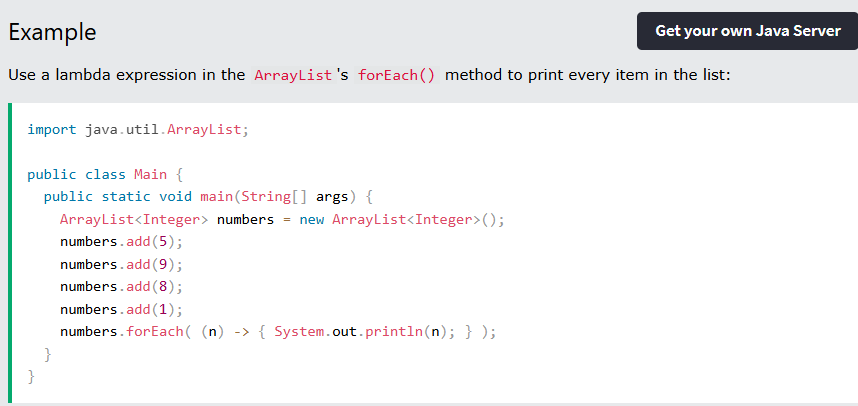
# Lambda Expression

The simplest lambda expression contains a single parameter and an expression:

*parameter* -> *expression*

To use more than one parameter, wrap them in parentheses:

*(parameter1, parameter2)* -> *expression*



# **Method References**

(phương thức tham chiếu) trong Java 8.

**Method References** là cú pháp viết tắt của biểu thức **Lambda**để gọi phương thức.



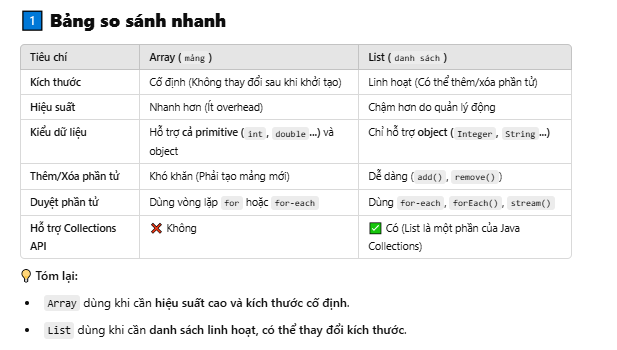


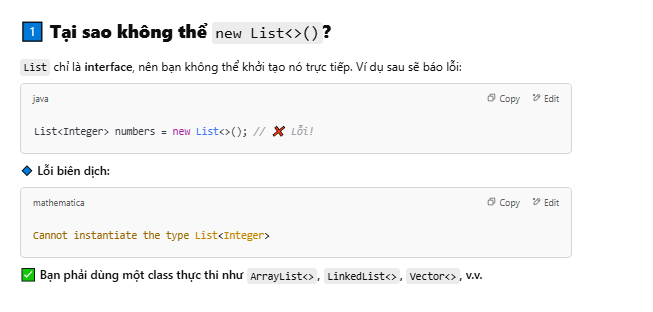
# **Working with Date and Time in Java**

# 

# **Datastructures**

## Array vs List

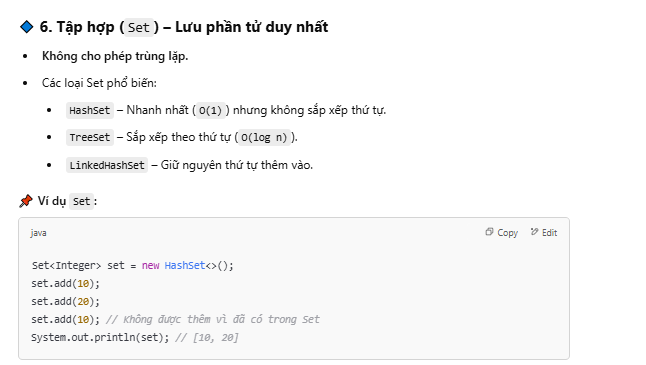




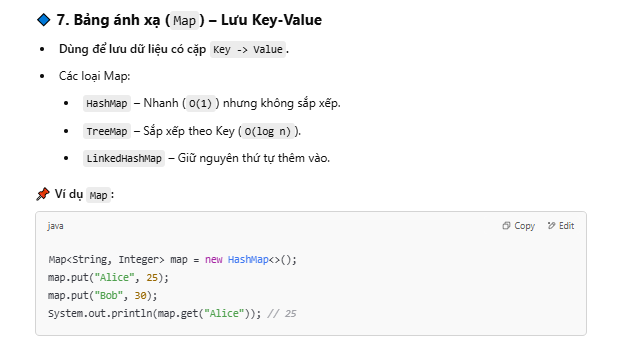
 Dùng **ArrayList** khi cần truy cập phần tử nhanh, danh sách không thay đổi nhiều.

 Dùng **LinkedList** khi cần thêm/xóa nhiều phần tử ở giữa danh sách.

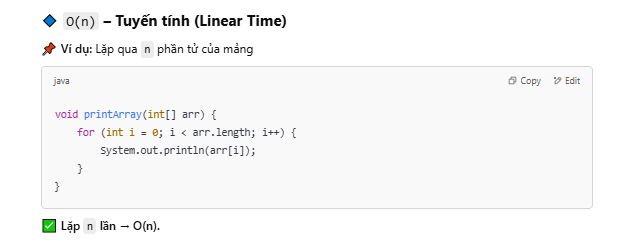
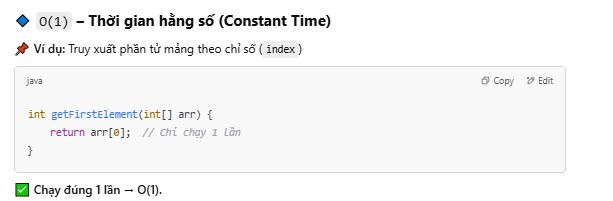
## Set

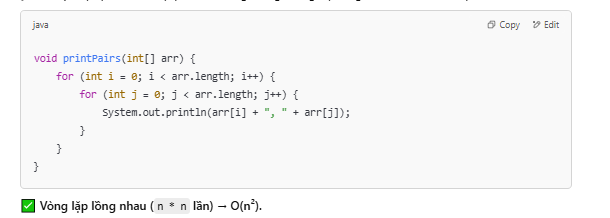


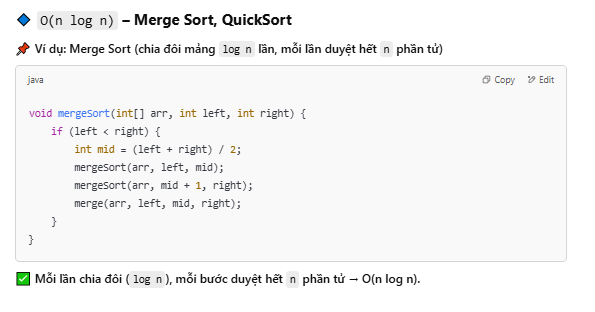
## Map



## Tính độ phức tạp







# OOP object-oriented programming

**4. Polymorphism (Đa hình)**

✔ Một hành động có thể có nhiều cách thực hiện khác nhau.

📌 Có 2 loại đa hình trong Java:  
1️Method Overloading (Nạp chồng phương thức) – Cùng tên phương thức, khác tham số.  
2️Method Overriding (Ghi đè phương thức) – Ghi đè phương thức của class cha trong class con.

# Package

* **Package** là một cách để nhóm các class có liên quan lại với nhau.
* **Có hai loại package**: Built-in (có sẵn trong Java) và User-defined (do lập trình viên tạo).
* **Sử dụng từ khóa import** để sử dụng class từ package khác.

# Generic

**Generic** trong Java giúp tạo ra **class, interface, hoặc method** có thể làm việc với **nhiều kiểu dữ liệu khác nhau** mà vẫn đảm bảo **tính an toàn kiểu dữ liệu (Type Safety)**.



# Gradle

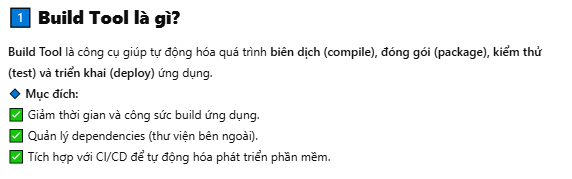
**Gradle** là một **công cụ build** (build tool) mạnh mẽ, linh hoạt, thường được sử dụng để quản lý dependencies và tự động hóa quá trình build trong **Java, Kotlin, Groovy**, đặc biệt là trong **Spring Boot**.

# Maven

Maven là một công cụ xây dựng nguồn mở, được sử dụng chủ yếu cho các dự án Java.

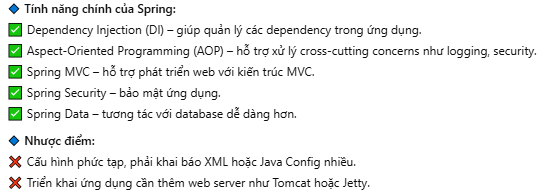
# Apache Ant

**Ant (Another Neat Tool)** là một công cụ **build tự động** cho Java, nhưng **cấu hình bằng XML** thay vì DSL như Gradle hay Maven.



# Spring

Spring là một **framework mạnh mẽ** dành cho Java, cung cấp các tính năng để xây dựng ứng dụng Java dễ dàng hơn.



# Spring Boot

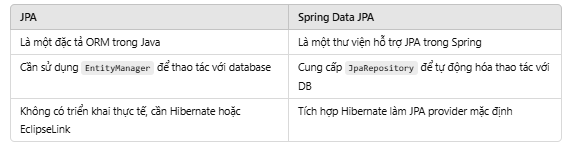
Spring Boot được xây dựng dựa trên nền tảng Spring Framework truyền thống, vốn được sử dụng rộng rãi để phát triển REST API

# 20/02

# JPA

JPA (Java Persistence API) là một chuẩn trong Java giúp quản lý dữ liệu trong cơ sở dữ liệu thông qua các đối tượng Java. Nó cung cấp cách tiếp cận hướng đối tượng để làm việc với dữ liệu mà không cần viết truy vấn SQL thủ công.

## JPA vs Spring Data JPA



# JDBC (Java™ database connectivity)

**JDBC** là một **API** trong Java giúp **kết nối và thao tác với cơ sở dữ liệu (DB**). Nó cung cấp một giao diện tiêu chuẩn để gửi truy vấn SQL, lấy dữ liệu và thực hiện các thao tác CRUD (Create, Read, Update, Delete).

JDBC hoạt động dựa trên **JDBC Driver**, là một phần mềm trung gian giúp Java giao tiếp với các hệ quản trị cơ sở dữ liệu (DBMS) như MySQL, PostgreSQL, SQL Server, Oracle, v.v.

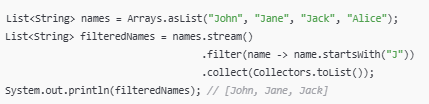
# JDBDI3

 higher level interface than JDBC to access the database.

# Java 8 Stream

một API giúp xử lý tập hợp dữ liệu (Collections, Arrays, v.v.) theo phong cách **functional programming**. Nó cung cấp các thao tác như **lọc (filter), ánh xạ (map), giảm (reduce), duyệt (forEach)** một cách dễ dàng, giúp viết code ngắn gọn, rõ ràng và hiệu quả hơn.

Ví dụ:



# 23/02

# Mybatis

MyBatis là một framework ORM (Object-Relational Mapping) nhẹ giúp tương tác với cơ sở dữ liệu một cách linh hoạt. Không giống như Hibernate (JPA), MyBatis không tự động ánh xạ Java object với bảng mà cho phép viết SQL thủ công nhưng vẫn hỗ trợ ánh xạ kết quả vào object.

