**CÔNG TY TMA SOLUTION BÌNH ĐỊNH**

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

**BÁO CÁO NHỮNG GÌ ĐÃ HỌC VỀ JAVA + SPRING**

**Mã intern** : 735

**Intern** : Võ Ngọc Khánh

**Mentor** : Nguyễn Văn Tiến

Quy Nhơn, ngày 27 tháng 3 năm 2023

**MỤC LỤC**

[**PHẦN 1 : TỔNG QUAN** 4](#_Toc130767000)

[**PHẦN 2 : TÌM HIỂU JAVA CORE** 7](#_Toc130767001)

[**1. Java là gì ?** 7](#_Toc130767002)

[**2. Kiểu dữ liệu** 7](#_Toc130767003)

[**3. Biến** 8](#_Toc130767004)

[**4. Toán tử** 9](#_Toc130767005)

[**5. Câu lệnh điều kiện** 9](#_Toc130767006)

[**6. Ép kiểu dữ liệu** 9](#_Toc130767007)

[**7. Chuỗi** 9](#_Toc130767008)

[**8. Access modifier** 10](#_Toc130767009)

[**9. Biểu thức chính quy** 10](#_Toc130767010)

[**10. File I/O Stream** 10](#_Toc130767011)

[**11. Exception Handing** 10](#_Toc130767012)

[**12. Java Collection && Generics** 11](#_Toc130767013)

[**13. Mutithreading** 12](#_Toc130767014)

[**14. Design pattern** 13](#_Toc130767015)

[**PHẦN 3 : TÌM HIỂU JAVA OOP** 15](#_Toc130767016)

[**1.** **Tính kế thừa** 15](#_Toc130767017)

[**2.** **Tính đóng gói** 16](#_Toc130767018)

[**3.** **Tính đa hình** 16](#_Toc130767019)

[**4.** **Tính trừu tượng** 17](#_Toc130767020)

[**5.** **Nguyên tắc SOLID** 18](#_Toc130767021)

[**PHẦN 4 : TÌM HIỂU SPRING FRAMEWORK** 19](#_Toc130767022)

[**1.** **Spring là gì ? Ưu nhược điểm của Spring ?** 19](#_Toc130767023)

[**2.** **Nguyên tắc xây dựng Spring** 19](#_Toc130767024)

[**3.** **Spring Architecture** 20](#_Toc130767025)

[**4.** **Spring core** 20](#_Toc130767026)

[**PHẦN 5 : TÌM HIỂU SPRING BOOT** 23](#_Toc130767027)

[**1.** **Spring boot là gì?** 23](#_Toc130767028)

[**2.** **Spring boot code structure** 24](#_Toc130767029)

[**3.** **Denpendency Injection trong Spring Boot** 25](#_Toc130767030)

[**4.** **Bean && Application context** 25](#_Toc130767031)

[**5.** **Spring boot ModelWrapper** 25](#_Toc130767032)

[**6.** **JPA && JDBC** 25](#_Toc130767033)

[**7.** **RESTful API with JPA + mySQL** 25](#_Toc130767034)

# **PHẦN 1 : TỔNG QUAN**

|  |  |  |
| --- | --- | --- |
| **Tuần** | **Nội dung đã học** | **Tổng quan** |
| 1 | Java core, OOP | * **Tìm hiểu Java basic :** * Java là gì ? Ứng dụng của java trong thực tế ? * Biến và kiểu dữ liệu trong java ? * Các loại toán tử trong java ? * Câu lệnh điều kiện * Chuỗi & các hàm làm việc với chuỗi ? * Lớp và đối tượng ? * Phạm vi truy cập khi khai báo ? * Regex ? Các regex thường dùng ? * Exception là gì ? Có mấy loại Exception ? * File I/O ? * **Tìm hiểu về Java OOP :** * Các tính chất của Java OOP ? * Tính kế thừa ? Các kiểu kế thừa ? Nguyên lý kế thừa trong Java ? Lợi ích ? Overriding ? * Từ khóa new, this, super ? * Tính đa hình ? Các kiểu đa hình ? Nguyên lý đa hình ? Lợi ích ? Overloading ? * So sánh Overloading vs Overriding trong đa hình? * Sử dụng toán tử instance of ? * Tính đóng gói ? Nguyên lý ? Lợi ích ? * Tính trừu tượng ? Phương thức trừu tượng ? Lớp trừu tượng ? Lợi ích ? * Interface ? |
|
|
|
|
|
|
|
|
|
| 2 | Java OOP  Java Data Structure  Collection - Generics  Mutithreading  Design pattern | * **Tìm hiểu về OOP** : SOLID là gì ? Nội dung ? * **Tìm hiểu Data Structure trong Java** * **Tìm hiểu Java Collection và Generics** * Collection là gì ? * Collection và Colections ? * Những loại Collection phổ biến ? * Generics là gì ? * **Tìm hiểu về đa luồng** * **Tìm hiểu về Design pattern** * Design pattern là gì ? Lợi ích ? * Các loại Design pattern ? Các loại design pattern phổ biến ? * Mẫu tạo dựng: Singleton pattern? Fatory Pattern ? Abstract Fatory Pattern ? Builder pattern ? Prototype pattern ? * Mẫu cấu trúc, mẫu hành vi ? * Dependency Injection ? |
| 3 | Spring Framework  Spring Boot | * **Tìm hiểu Spring Framework** * Spring là gì ? * Nguyên tắc thiết kế ? DI ? AOP ? * Kiến trúc Spring ? * Các thành phần phát triển ứng dụng sử dụng Spring Framework? * Tìm hiểu Spring core ? * **Tìm hiểu Spring Boot** * Spring boot là gì ? * Code Structure của Spring boot? * Dependency Injection trong Spring boot * Bean & ApplicationContext ? * ModelWrapper là gì ? Lợi ích ? * Sử dụng dependency : Lombok, Thymeleaf * Cơ bản về HTTP response * JPA && JDBC ? * Code ví dụ RESTful API với JPA + mySQL |

**PHẦN 2 : TÌM HIỂU JAVA CORE**

1. **Java là gì ?**

- Java là một ngôn ngữ lập trình phổ biến, độc lập đa nền tảng được phát triển bởi Oracle Corporation…

- Đặc điểm của Java :

+ Đa nền tảng : Thay vì biên dịch mã nguồn của chương trình thành mã máy cho từng nền tảng như C++ , Java được biên dịch thành bytecode, nên Java có thể viết 1 lần và chạy trên nhiều nền tảng khác nhau dễ dàng.

**Note** : Bytecode là một dạng mã trung gian giữa mã nguồn và mã máy, có thể chạy trên nhiều nền tảng khác nhau mà không cần sửa đổi mã nguồn ban đầu. Bytecode được thực thi trên máy ảo JVM và dịch được JVM biên dịch thành mã máy.

+ Quản lý bộ nhớ tự động : Sử dụng thuật toán Garbage Collection để quản lý và giải phóng tự động bộ nhớ.

+ Tính bảo mật cao : Sử dụng cơ chế phân quyền, kiểm soát truy cập để ngăn chặn các cuộc tấn công từ bên ngoài.

+ Hỗ trợ đa luồng : Giúp thực hiện đồng thời nhiều tác vụ hiệu quả.

+ Cú pháp rõ ràng dễ đọc : Giúp dễ tiếp cận và phát triển ứng dụng nhanh chóng.

- Java được sử dụng để : Phát triển ứng dụng android, web, ứng dụng desktop,…

- Các công cụ phát triển ứng dụng Java :

+ JVM (Java Virtual Machine) có nhiệm vụ tải, kiểm tra và thực thi mã bytecode.

+ JRE (Java Runtime Environment) là môi trường thực thi cho các ứng dụng Java.

+ JDK (Java Development Kit) bao gồm JRE và các công cụ cần thiết để phát triển ứng dụng Java, bao gồm trình biên dịch, trình gỡ lỗi, các thư viện và các công cụ khác.

1. **Kiểu dữ liệu**

Gồm 2 kiểu :

+ Kiểu dữ liệu nguyên thủy ( byte, char, short, int, long, float, double, boolean)

+ Kiểu dữ liệu đối tượng ( array, class, interface).

+ array : Một mảng của các dữ liệu cùng kiểu.

+ class : Do người dùng định nghĩa, chứa thuộc tính và phương thức

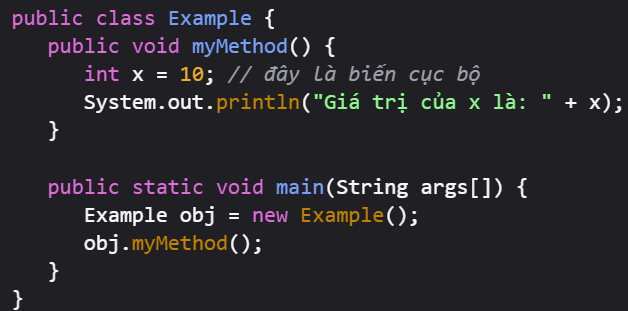
**+** interface : Do người dùng định nghĩa, chứa các phương thức giao tiếp

**Note :** Các phương thức giao tiếp là các phương thức abstract không có định nghĩa, các lớp khác có thể triển khai ( implement) interface để sử dụng các phương thức đó.

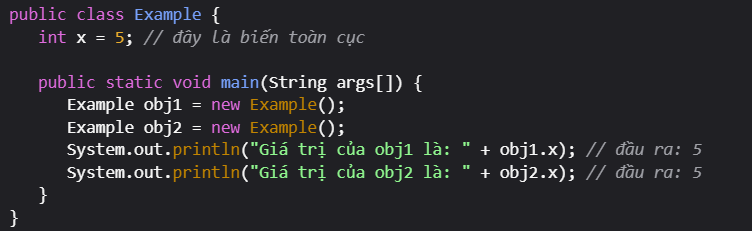
1. **Biến**

Gồm 3 loại :

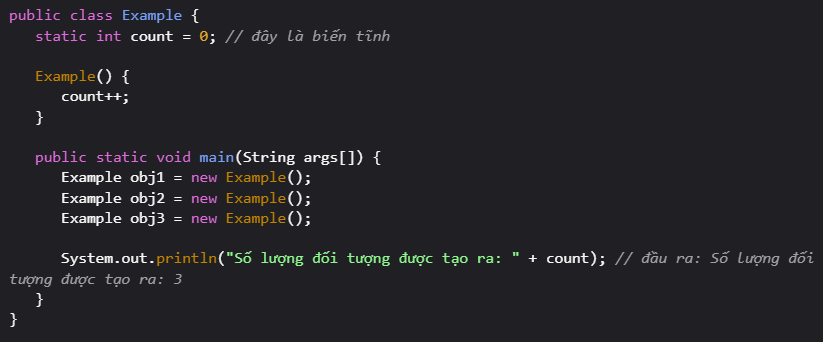
+ Biến cục bộ: Có hiệu lực trong phạm vi khai báo của nó .



+ Biến toàn cục: Có thể truy cập bởi phương thức và đối tượng trong lớp đó



+ Biến tĩnh: Tương tự như biến toàn cục nhưng không thể ghi đè bởi các đối tượng khác.



1. **Toán tử**

Gồm : số học, quan hệ, logic, bitwise, gán, điều kiện

1. **Câu lệnh điều kiện**

- Mệnh đề if…else

- Mệnh đề Switch…case : được sử dụng để thực thi 1 hoặc nhiều khối lệnh từ nhiều điều kiện.

- Vòng lặp while, do…while : Nếu điều kiện là true thì nó sẽ trở thành vòng lặp vô tận

- Vòng lặp for : 3 loại : đơn giản, cải tiến và gán nhãn

- Break : được sử dụng để stop thực thi lệnh trong vòng lặp hoặc trong mệnh đề switch tại điều kiện đã được chỉ định. Đối với vòng lặp bên trong vòng lặp khác, thì nó chỉ stop vòng lặp bên trong đó.

- Continue : được sử dụng để tiếp tục vòng lặp tại điều kiện đã được xác định, với điều kiện đó khối lệnh phía sau từ khóa continue sẽ không được thực thi.

1. **Ép kiểu dữ liệu**

- Ép kiểu cơ bản trong java là việc gán giá trị của một biến có kiểu dữ liệu này tới biến khác có kiểu dữ liệu khác.

- Ép kiểu đối tượng : áp dụng tính đa hình của java

1. **Chuỗi**

- StringBuffer xử lý đa luồng ( mutireading)

- StringBuilder xử lý 1 luồng : công dụng như nhau đáp ứng nhu cầu sử dụng chuỗi có thể thay đổi được như ( insert, replace, delete, append..)

- Có 2 cách khai báo chuỗi đó là thông thường và khởi tạo kiểu “đối tượng”

- Các hàm hay dùng với chuỗi : length(), trim(), concat(), substring(), equal(),…

**Note** : Không thể dùng toán tử “==” để so sánh chuỗi, thay vào đó dùng equal để so sánh bởi vì String là một đối tượng.

1. **Access modifier**

- Access modifier là phạm vi truy cập của một class, một thuộc tính, một phương thức và có 4 loại : public, protected, default, private

- Đặc tính truy xuất của 4 loại modifier :

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | Private | Default | Protected | Public |
| Cùng class | Yes | Yes | Yes | Yes |
| Cùng package, khác class | No | Yes | Yes | Yes |
| Class con trong cùng package với class cha | No | Yes | Yes | Yes |
| Khác package, khác class | No | No | No | Yes |
| Class con khác package với class cha | No | No | Yes | Yes |

**Note** : Class chỉ có 2 Access modifier là public và default.

1. **Biểu thức chính quy**

- Biểu thức chính quy ( regular expression) :Là 1 dãy các kí tự liên tục, nó giúp người dùng tìm kiếm hoặc so khơp hoặc các thao tác liên quan tuân theo những quy tắc và cú pháp nhất định :

- Ví dụ : Định dạng ngày dd/mm/yyyy hoặc dd-mm-yyyy:



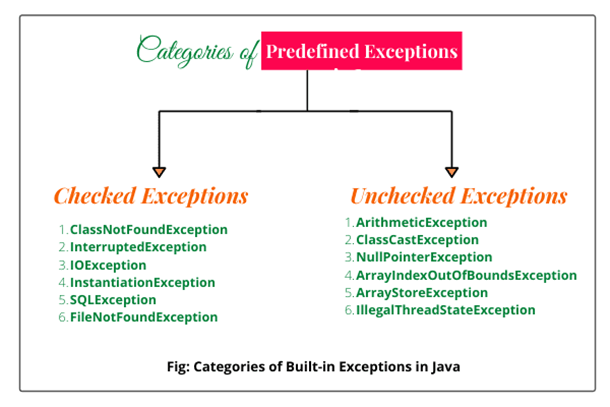
1. **File I/O Stream**

Java sử dụng stream để quá trình input và output được thực hiện nhanh chóng. InputStream được sử dụng để đọc dữ liệu từ 1 nguồn, OutStream dùng để ghi dữ liệu đến 1 điểm đích.

1. **Exception Handing**

- Exception Handing : Xử lý ngoại lệ là một cơ chế xử lý các lỗi runtime để có thể duy trì luồng bình thường của ứng dụng. Xử lý ngoại lệ gồm 3 hoạt động chính là đặc tả ngoại lệ, ném ra ngoại lệ và bắt ngoại lệ.

- Xử lý ngoại lệ được phân làm 2 loại : Checked Exceptions (compile) Unchecked Exceptions (runtime):



- Các khối lệnh sử dụng để exception :

+ Try…catch

+ Try… finally

+ Try..catch…finally

Note : Khối lệnh finally luôn được thực thi, trừ khi chương trình đột ngột dừng lại or die.

- Từ khóa throw && throws :

+ Throw : Dùng để ném ra ngoại lệ do người dùng định nghĩa ( checked || unchecked)

+ Throws : Dùng để khai báo ngoại lệ ( checked)

Trong lập trình chúng ta thường có thể gặp một trong các lỗi sau:

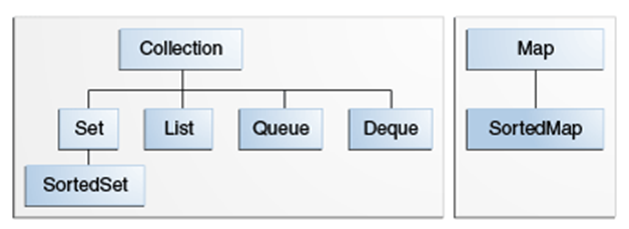
+ Lỗi cú pháp (syntac error)

+ Lỗi logic thuật toán

+ Lỗi lúc thực thi ( runtime error)

1. **Java Collection && Generics**

- Collection trong Java là một nhóm các đối tượng được tổ chức theo một cách nhất định để quản lý và lưu trữ dữ liệu.



- Một số collection phổ biến :

+ List: các phần tử được sắp xếp theo thứ tự và có thể chứa các phần tử trùng lặp.

+ Set: các phần tử không theo thứ tự và không chứa các phần tử trùng lặp.

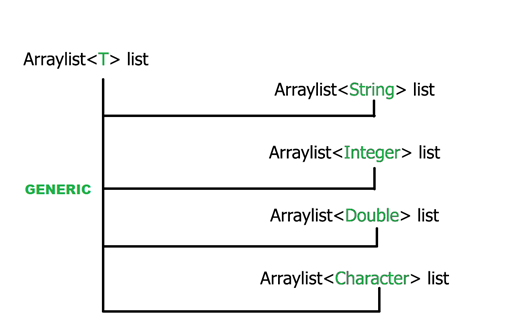
+ Map: được sử dụng để lưu trữ và truy xuất dữ liệu theo cặp key-value, không theo thứ tự và không chứa các key trùng lặp.

- Generics: Thuật ngữ “generics” được hiểu là tham số hóa kiểu dữ liệu.

- Việc tham số hóa kiểu dữ liệu giúp dễ bắt lỗi các kiểu dữ liệu không hợp lệ, đồng thời giúp dễ dàng hơn cho việc tạo và sử dụng các class, interface, method với nhiều kiểu dữ liệu khác nhau.

- Vì Generics kiểm tra dữ liệu chặt chẽ ở compile-time nên chúng ta sẽ dễ dàng kiểm soát lỗi hơn.

- Vd :



1. **Mutithreading**

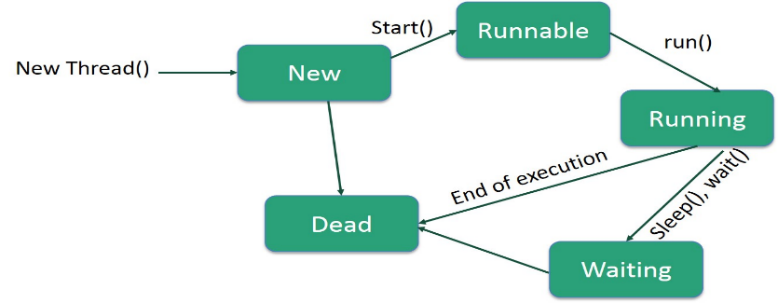
- Multithreading (hay còn gọi là đa luồng) là một kỹ thuật được sử dụng trong lập trình máy tính để thực hiện đa nhiệm song song.

- Thông qua việc tạo nhiều luồng xử lý đồng thời, các ứng dụng có thể đạt được khả năng xử lý đa tác vụ.

**Note** : Mỗi luồng sẽ có thể thực hiện công việc độc lập với các luồng khác, giúp tiết kiệm thời gian xử lý.

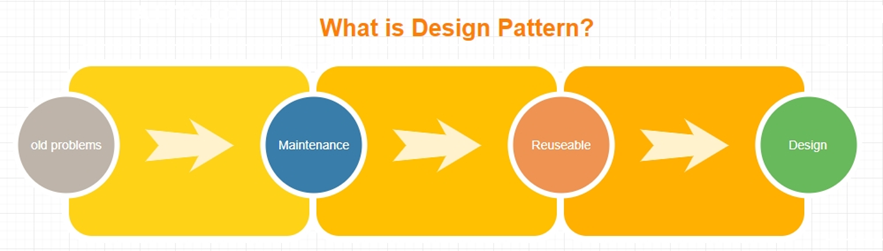
Ví dụ : Khi tải nhiều tệp dữ liệu từ internet. Thay vì tải dữ liệu một cách tuần tự từng tệp một, có thể sử dụng multitheading để tải nhiều tệp cùng một lúc. Điều này giúp tiết kiệm thời gian xử lý và cải thiện hiệu suất của ứng dụng.

- Life cycle của luồng :



1. **Design pattern**

- Desgn pattern ( hay còn gọi là mẫu thiết kế) là một giải pháp lặp lại nói chung cho việc giải quyết một vấn đề thường gặp trong thiết kế phần mềm.



- Lợi ích của desgn pattern:

+ Các mẫu thiết kế đã được xác định và cung cấp cách tiếp cận chuẩn theo ngành để giải quyết vấn đề lặp lại, vì vậy sẽ tiết kiệm thời gian hơn nếu chúng ta sử dụng mẫu thiết kế hợp lý.

+ Tăng khả năng tái sử dụng dẫn đến phát triển phần mềm nhanh hơn.

+ Giúp giảm tổng chi phí của sản phẩm phần mềm.

+ Giúp code dễ hiểu và dễ gỡ lỗi.

- Các loại design pattern :

+ Mẫu tạo dựng ( Creational design pattern) : cung cấp giải pháp tạo một đối tượng theo cách tốt nhất phù hợp cho từng tình huống.

Ví dụ : Singleton pattern, Factory pattern, Abstract Factory Pattern, Builder pattern ,Prototype pattern.

+ Mẫu cấu trúc ( Structural design pattern) : cung cấp giải pháp tạo ra cấu trúc của một lớp

Ví dụ : Apdaper Pattern, Composite Pattern, Facade Pattern, Bridge Pattern

+ Mẫu hành vi (Behavioral design pattern) : cung cấp giải pháp cho sự tương tác giữa các class. Làm sao để tăng tính mở rộng.

+ Một số mẫu khác :

Dependency Injection : là một kỹ thuật, một design pattern cho phép xóa bỏ sự phụ thuộc hard-code, giúp phần mềm dễ mở rộng và maintain hơn.

Phương pháp thực hiên Dependency Injection :

+ Contructor Injection : Các dependency sẽ được truyền ( inject) vào 1 class thông qua constructor của class đó .

+ Setter Injection : Các dependency sẽ được truyền vào 1 class thông qua hàm Setter/Getter.

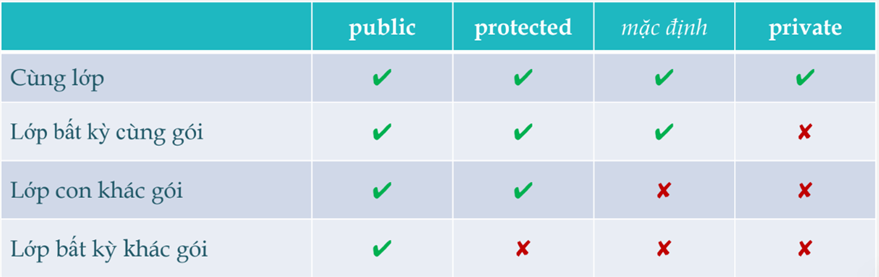
**PHẦN 3 : TÌM HIỂU JAVA OOP**

1. **Tính kế thừa**

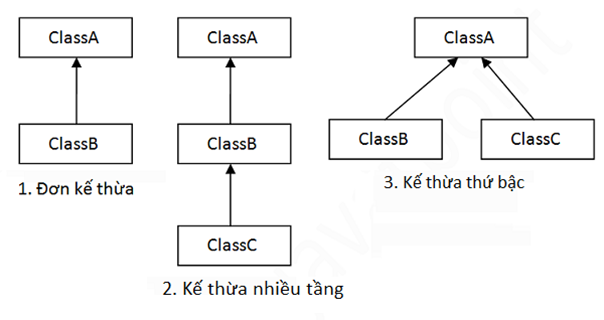
**-** Tính kế thừa ( Inheritance) : Kế thừa cho phép tạo ra một lớp mới bằng cách sử dụng các thuộc tính và phương thức của lớp đã có sẵn. Lớp kế thừa có thể mở rộng hoặc thay đổi các tính năng của lớp cha để đáp ứng nhu cầu của ứng dụng.

**Note :** 1 class cha có thể có nhiều class con, nhưng 1 class con chỉ có 1 class cha

**-** Nguyên lý kế thừa :



- Có 3 kiểu kế thừa : đơn kế thừa, kế thừa nhiều tầng và kế thừa thứ bậc



**Note** : đa kế thừa không được support thông qua class, nó chỉ được support thông qua interface.

- Lợi ích của kế thừa :

+ Giúp tái sử dụng mã một cách dễ dàng bằng cách sử dụng các tính năng đã có của lớp cha trong lớp con.

+ Tạo ra các lớp có cấu trúc hợp lý và dễ dàng bảo trì.

+ Giúp tăng tính linh hoạt của ứng dụng.

- Trong kế thừa, "super" và "this" là từ khóa được sử dụng để truy cập các thuộc tính và phương thức của lớp cha và lớp con, tương ứng.

- Overriding trong kế thừa : lớp con có phương thức giống lớp cha.

- Nguyên tắc khi overriding :

+ Phương thức phải có tên giống lớp cha.

+ Phương thức phải có tham số giống lớp cha

+ Lớp con và lớp cha có mỗi quan hệ kế thừa, nên phải cùng kiểu trả về.

**Note** : Phương thức static, private, final không được phép overriding.

1. **Tính đóng gói**

- Tính đóng gói (encapsulation) : Đóng gói là một cách để che giấu thông tin và bảo vệ dữ liệu trong một lớp. Nó cho phép truy cập vào các thành phần của lớp thông qua các phương thức công khai mà không cần biết cách thức thực hiện bên trong của lớp.

- Với các access modifier, tính đóng gói sẽ có thể giúp ngăn chặn những lớp bên ngoài truy cập, thay đổi thuộc tính và phương thức của một lớp. Từ đó, giúp cho việc che giấu dữ liệu (data hiding).

- Nguyên tắc để đạt tính đóng gói :

+ Khai báo các biến của một lớp là private.

+ Cung cấp phương thức setter và getter là public để có thể sửa đổi và xem các giá trị biến.

- Lợi ích của tính đóng gói :

+ Bảo vệ dữ liệu của lớp, tránh được truy cập trực tiếp từ các đối tượng bên ngoài lớp và đảm bảo tính toàn vẹn của dữ liệu.

+ Giúp giảm sự phụ thuộc giữa các lớp với nhau, tăng tính độc lập và tái sử dụng mã.

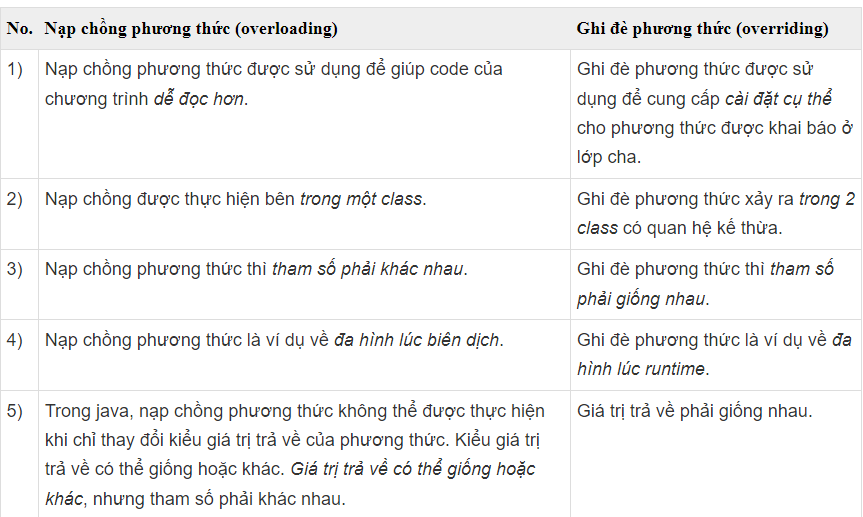
+ Cung cấp sự kiểm soát truy cập cho các thành phần của lớp.

1. **Tính đa hình**

- Tính đa hình ( Polymorphism) : Đa hình cho phép sử dụng cùng một phương thức với các đối tượng khác nhau để đạt được kết quả khác nhau. Điều này giúp tăng tính linh hoạt và tái sử dụng mã trong chương trình.

- Có 2 loại đa hình là compile time polymorphism và runtime polymorphism

- Ví dụ overriding là đa hình lúc thực thi, overloading là đa hình lúc biên dịch



- Lợi ích của đa hình :

+ Giúp tăng tính linh hoạt và tái sử dụng mã trong chương trình.

+ Cho phép chương trình thực hiện các phương thức khác nhau với cùng một tên phương thức.

+ Giúp tăng tính tương thích giữa các lớp khác nhau trong ứng dụng.

1. **Tính trừu tượng**

- Trừu tượng (Abstraction): Trừu tượng là một cách để tập trung vào cách thức hoạt động của một đối tượng mà không quan tâm đến chi tiết bên trong. Nó cho phép giảm sự phức tạp của chương trình và tăng tính dễ bảo trì.

- Tính trừu tượng bao gồm lớp trừu tượng và phương thức trừu tượng

- Nguyên tắc :

+ Không thể khởi tạo trực tiếp 1 lớp trừu tượng

+ Lớp con bắt buộc phải triển khai ( implement) tất cả phương thức trừu tượng của lớp cha.

- Lợi ích của tính trừu tượng :

+ Giúp giảm sự phức tạp của chương trình và tăng tính dễ bảo trì.

+ Cung cấp sự che giấu chi tiết bên trong của lớp và tập trung vào các tính năng quan trọng của đối tượng.

+ Giúp tách biệt trách nhiệm của các lớp và giảm sự phụ thuộc giữa chúng.

- Interface là một thành phần của Java được sử dụng để định nghĩa một tập hợp các phương thức trừu tượng và hằng số. Interface chỉ định các phương thức và hằng số mà một lớp cần triển khai để đạt được tính năng cụ thể.

1. **Nguyên tắc SOLID**

**Solid** : là 5 nguyên lý thiết kế trong OOP , Những nguyên tắc này đã được đưa ra bởi Michael Feathers và Robert C.Martin bao gồm:

|  |  |  |  |
| --- | --- | --- | --- |
| **Nguyên lý** | | **Nội dung** | **Ghi chú** |
| **S** | **Single responsibility** | Mỗi lớp chỉ nên chịu trách nhiệm về một nhiệm vụ cụ thể nào đó mà thôi. | Đơn giản nhất, nhưng khó dùng nhất |
| Ví dụ : Ngày mai công ty có 1 buổi phỏng vấn ứng viên ( Schedule Interview). Để xử lý công việc này chúng ta cần lưu thông tin phỏng vấn và gửi email kết quả về cho ứng viên. Thay vì thiết kế 1 class ScheduleInterview để xử lý tất cả công việc trên, ta vận dụng nguyên lý SRP tạo 1 class interview để lưu thông tin buổi phỏng vấn, 1 class EmailSending để thực hiện công việc gửi email. | | | |
| **O** | **Open/Closed** | Không được sửa đổi một Class có sẵn, nhưng có thể mở rộng bằng kế thừa. | Open: Dễ mở rộng Closed: Khó sửa đổi |
| Ví dụ: một lớp cơ sở PaymentGatewayBase chứa tất cả các thuộc tính và phương thức thanh toán cơ bản. Lớp này có thể được mở rộng bởi các lớp PaymentGateway khác nhau cho các cổng thanh toán khác nhau để phục vụ các chức năng của họ. Do đó, nó được mở rộng thoải mái nhưng hạn chế việc thay đổi. | | | |
| **L** | **Liskov Substitution** | Các đối tượng của lớp con có thể thay thế các đối tượng của lớp cha mà vẫn đảm bảo tính đúng đắn của chương trình. | Nguyên lý LSP dễ bị vi phạm nhất, nguyên nhân chủ yếu là do sự thiếu kinh nghiệm khi thiết kế class |
| Nói một cách dễ hiểu: chỉ cho class A kế thừa class B khi class A thay thế được class B.  Nguyên lý này đảm bảo rằng kế thừa được sử dụng đúng cách. | | | |
| **I** | **Interface Segregation** | Thay vì dùng 1 interface lớn, ta nên tách thành nhiều interface nhỏ, với nhiều mục đích cụ thể. | Một lớp không nên chứa các method mà không sử dụng. |
| Áp dụng nguyên lý ISP sẽ làm hệ thống linh hoạt hơn, đồng thời giảm thiểu code thừa (do phải implement những tính năng không cần thiết). Tuy nhiên, trong thực tế vẫn có nhiều trường hợp bất khả kháng mà chúng ta phải tạo 1 interface lớn. | | | |
| **D** | **Dependency Inversion** | Một lớp không nên phụ thuộc trực tiếp vào các lớp khác, mà nên phụ thuộc vào các abstraction và interface. |  |
| Dependency Injection pattern là một implementation của nguyên lý này. | | | |

**NOTE** : Về bản chất, nguyên lý chỉ là nguyên lý, nó chỉ là hướng dẫn chứ không phải là quy tắc tuyệt đối bất di bất dịch. Tuy vậy, việc hiểu rõ chúng vẫn giúp code ta viết ra dễ đọc, dễ hiểu, dễ quản lý hơn.

**PHẦN 4 : TÌM HIỂU SPRING FRAMEWORK**

1. **Spring là gì ? Ưu nhược điểm của Spring ?**

- Spring là một framework phát triển các ứng dụng Java được sử dụng bởi hàng triệu lập trình viên.Nó giúp tạo các ứng dụng có hiệu năng, dễ kiểm thử, sử dụng lại code…

- Spring là một mã nguồn mở, được phát triển, chia sẻ và có cộng đồng người dùng lớn.

- Spring thường sử dụng để phát triển Java Desktop, ứng dụng mobile, Java Web. Mục tiêu chính của spring là giúp phát triển các ứng dụng J2EE một cách đễ dàng hơn :

+ Dựa trên mô hình sử dụng POJO( Plain Old Java Oject) để giảm sự phức tạp trong phát triển các ứng dụng và tăng tính tái sử dụng

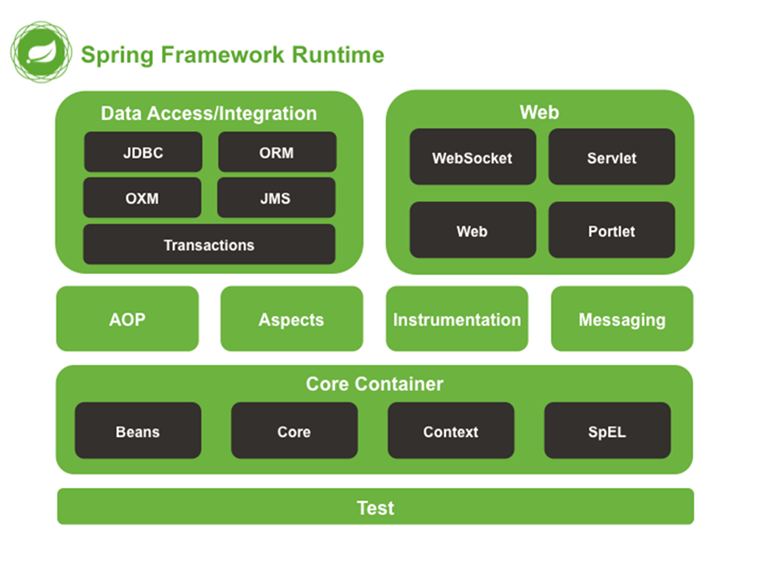
+ Nới lỏng ràng buộc giữa các thành phần thông qua việc sử dụng Dependency Injection và viết các interface

+ Tiếp cận lập trình khai báo bằng cách sử dụng các quy tắc (convention) và các khía cạnh (aspect) chung

1. **Nguyên tắc xây dựng Spring**

Spring framework được xây dựng dựa trên 2 nguyên tắc design chính là : DI và AOP ( lập trình hướng khía cạnh)

1. **Spring Architecture**

****

Spring là một framework phát triển ứng dụng Java. Kiến trúc của Spring bao gồm nhiều thành phần khác nhau, được chia thành các mô-đun riêng biệt với các chức năng cụ thể. Các thành phần chính của kiến trúc Spring bao gồm:

- **Core Container:** Đây là thành phần cốt lõi của Spring, chứa các lớp cơ bản để xử lý các yêu cầu phức tạp của ứng dụng. Core Container chứa hai phần chính:

* **Spring Bean**: Là một đối tượng được quản lý bởi Spring IoC (Inversion of Control) container. Spring Bean cung cấp các phương thức khởi tạo và phương thức hủy, giúp quản lý tài nguyên tốt hơn.
* **Spring IoC:** Là một bộ điều khiển Inversion of Control (IoC) cung cấp khả năng quản lý các Spring Bean và phân phối các Bean trong ứng dụng.

- **AOP (Aspect Oriented Programming):** Là một phần mở rộng của OOP, AOP cung cấp các cơ chế để tách riêng các quan tâm không liên quan với logic chính của ứng dụng, giúp giảm độ phức tạp và tăng tính tái sử dụng.

- **Data Access/Integration:** Là thành phần cung cấp các khả năng để truy cập dữ liệu và tích hợp ứng dụng với các nguồn dữ liệu khác nhau. Spring Data là một mô-đun dễ sử dụng cho việc tương tác với các hệ quản trị cơ sở dữ liệu khác nhau.

- **Web**: Là thành phần cung cấp khả năng phát triển ứng dụng web bằng cách sử dụng các kỹ thuật như MVC (Model-View-Controller) và Restful Web Services. Spring MVC là một phần của Spring Web cung cấp khả năng phát triển các ứng dụng web theo mô hình MVC.

- **Test**: Là một mô-đun hỗ trợ cho việc viết các bài kiểm tra (test) cho ứng dụng. Spring Test Framework cung cấp khả năng kiểm tra các đơn vị và tích hợp các bài kiểm tra với các môi trường phát triển khác nhau.

**Note** : Kiến trúc của Spring có tính linh hoạt và có khả năng mở rộng tốt, cho phép các lập trình viên có thể tùy chỉnh để phù hợp với yêu cầu cụ thể của ứng dụng.

1. **Spring core**
   1. **Spring IoC**

- Ioc( Inversion of Control) : giúp làm thay đổi luồng điều khiển của chương trình 1 cách linh hoạt, thường dùng với Dependency Injection

- Spring IoC Container là thành phần thực hiện IoC, Spring IoC Container sẽ cấu hình và quản lý vòng đời của đối tượng từ lúc tạo cho đến lúc bị hủy

- Gồm 2 kiểu : BeanFactory và ApplicationContext.

### **4.2 Định nghĩa Annotation trong Spring Framework**

Trong Spring Framework, annotation được sử dụng để cung cấp thông tin bổ sung cho các đối tượng trong ứng dụng. Annotation là các thẻ đánh dấu được đặt trực tiếp trước một phần tử trong mã nguồn Java, chẳng hạn như một class, method, hoặc field.

Một số annotation phổ biến trong Spring Framework bao gồm:

* + @Controller: Đánh dấu một class là một controller trong Spring MVC.
  + @Autowired: Tự động liên kết các dependency cho một bean trong Spring.
  + @RequestMapping: Xác định các URL mà một method trong một controller xử lý.
  + @Service: Đánh dấu một class là một service trong Spring.
  + @Repository: Đánh dấu một class là một repository trong Spring.
  + @Transactional: Xác định các phương thức sử dụng transaction.

Các annotation này giúp đơn giản hóa việc cấu hình ứng dụng và giảm thiểu lượng mã cần phải viết thủ công. Ngoài ra, Spring cũng cho phép tạo các annotation tuỳ chỉnh để đáp ứng các nhu cầu đặc biệt của ứng dụng.

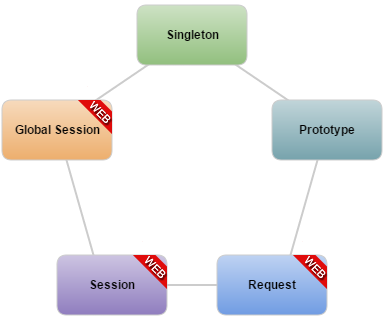
* 1. **Spring bean**

**-** Spring Bean là 1 đối tượng được quản lý bởi Spring Ioc Container, được tạo khi sử dụng và hủy khi ứng dụng không cần đến nữa.

**-** Annotation trong Spring là các thẻ đánh đấu được đặt trực tiếp trước một class, method hoặc field, Ví dụ : @Component, @Controller, @Service,…

**-** Mỗi 1 thẻ khi đánh dấu sẽ được IoC container hiểu là 1 Bean, và lấy ra khi cần sử dụng.

**-** **Bean Scope**

****

- Singleton: Bean mặc định sử dụng scope này. Nó được tạo ra duy nhất một lần và sử dụng chung cho tất cả các yêu cầu từ các class khác.

- Prototype: Khi sử dụng scope này, mỗi lần bean được yêu cầu, nó sẽ tạo ra một instance mới cho mỗi yêu cầu.

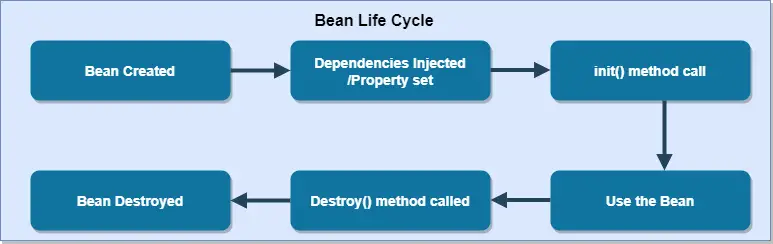
- Request : Bean sẽ được tạo mới cho mỗi request HTTP

- Session : Bean sẽ được tạo ra một lần duy nhất cho mỗi phiên làm việc

- Global Session : Bean sẽ tạo ra một lần duy nhất cho mỗi phiên người dùng

**Note :**  Các scope session và global session cần sử dụng trong môi trường web với các session và phiên làm việc được hỗ trợ

- **Bean Life Cycle**

****

Bean life cycle trong Spring là quá trình mà một đối tượng bean được tạo ra, khởi tạo, sử dụng và cuối cùng bị hủy bỏ. Quy trình này bao gồm các bước sau:

* **Khởi tạo**: Khi một đối tượng bean được tạo ra, Spring sẽ tìm kiếm bean trong context, tạo một instance của bean và khởi tạo các thuộc tính của nó.
* **Thiết lập thuộc tính**: Sau khi bean được khởi tạo, Spring sẽ thiết lập các thuộc tính của bean, bao gồm các giá trị được cấu hình trong tệp cấu hình và các bean khác được yêu cầu.
* **Gọi các phương thức cần thiết**: Sau khi tất cả các thuộc tính của bean đã được thiết lập, Spring sẽ gọi các phương thức cần thiết để chuẩn bị cho việc sử dụng bean.
* **Sử dụng**: Sau khi bean được khởi tạo và chuẩn bị, nó sẵn sàng để sử dụng trong ứng dụng của chúng ta.
* **Huỷ**: Khi ứng dụng kết thúc hoặc bean không còn được sử dụng, Spring sẽ giải phóng tài nguyên được sử dụng bởi bean và hủy bỏ bean.

Quy trình này được quản lý bởi Spring Container, nơi tất cả các bean được quản lý và được tạo ra. Khi một bean được yêu cầu, Spring Container sẽ xử lý toàn bộ quy trình Bean Life Cycle và trả về một instance của bean.

* 1. **Java Based Configuration**

**-** Java Based Configuration là một cách cấu hình và tạo các bean trong Spring Framework bằng cách sử dụng mã Java thay vì sử dụng file cấu hình XML như trước đây.

**-** Trong Java Based Configuration, chúng ta sử dụng các class Java để cấu hình và tạo các bean. Các class này được đánh dấu bằng các annotation như @Configuration, @ComponentScan, @Bean, @Import... để Spring biết rằng đây là các class cấu hình và các bean được tạo ra từ đó.

- Java Based Configuration cung cấp một số lợi ích như làm cho mã nguồn dễ đọc hơn, dễ bảo trì hơn và giúp cho việc kiểm tra lỗi trong quá trình compile sớm hơn. Ngoài ra, nó cũng cho phép chúng ta sử dụng các tính năng của ngôn ngữ Java như generics, inheritance, method overloading... để tạo ra các bean phức tạp hơn.

* 1. **Dependency Injection trong Spring**

- Có thể thực hiện DI bằng cách tự code, tự định nghĩa các điều kiện tạo thể hiện.. tuy nhiên trong thực tế người ta thường dùng các thư viện framework để thực hiện DI một cách thuận tiện dễ hiểu hơn

- Trong Spring có 2 cách thực hiện DI là : qua hàm khởi tạo và qua hàm setter.

* 1. **Spring AOP**

- Aspect Oriented Programming là 1 kỹ thuật lập trình dùng để tách logic chương trình thành các phần riêng biệt..

- Trong Spring, AOP có 4 loại advice được hỗ trợ :

+ Before advice : chạy trước khi method được thực thi

+ After returning advice : Chạy sau khi method trả về một kết quả

+ After throwing advice : Chạy khi method ném ra 1 exception

+ Around advice : Chạy khi method được thực thi

* 1. **AspectJ là gì? Spting AOP+ AspectJ**

- AspectJ là một thư viện, đặc tả trong java để thực hiện AOP.

- Tương tự như Spring AOP, thay vì định nghĩa trong Bean, AspectJ cung cấp các annotation để định nghĩa các poincut, advice, target object…

- Các Annotation chung của AspectJ gồm :

* @Before : chạy trước khi method được thực thi
* @After : Chạy sau khi method trả về 1 kết quả
* @AfterReturning : Chạy sau khi method trả về 1 kết quả, lấy kết quả trả về
* @AfterThrowing : Chạy khi method xảy ra exception
* @Around : Gồm các tất cả advice trên
* @Aspect : Đánh dấu đây là 1 aspect
  1. **Spring Autowiring**

**-** Autowiring là một tính năng quan trọng trong Spring Framework cho phép tự động kết nối các đối tượng (objects) với nhau thông qua việc quét (scan) các bean trong container.

- Spring hỗ trợ Autowiring qua các cách sau:

* Autowiring qua thuộc tính (property): sử dụng annotation @Autowired trên thuộc tính cần inject để Spring tự động gán giá trị vào thuộc tính đó.
* Autowiring qua constructor: sử dụng annotation @Autowired trên constructor để Spring tự động tìm kiếm và inject các giá trị phù hợp vào constructor.
* Autowiring qua setter method: sử dụng annotation @Autowired trên phương thức setter của thuộc tính để Spring tự động inject giá trị vào thuộc tính đó.
* Autowiring qua phương thức: sử dụng annotation @Autowired trên phương thức khởi tạo của đối tượng để Spring tự động inject các giá trị phù hợp vào các tham số của phương thức.

Mặc định, Spring sẽ tự động tìm kiếm và inject các bean phù hợp với kiểu dữ liệu của thuộc tính hoặc tham số của phương thức. Tuy nhiên, nếu trong container có nhiều hơn một bean có cùng kiểu dữ liệu, chúng ta phải sử dụng annotation @Qualifier để chỉ định tên của bean cần inject.

**PHẦN 5 : TÌM HIỂU SPRING BOOT**

1. **Spring boot là gì?**

- Spring mặc dù mạnh mẽ, nhưng nó cofig dự án khá phức tạp ví dụ tạo 1 dự án Spring bất kỳ :

+ Tạo Maven hoặc Gradle project, thêm các thư viện cần thiết

+ Tạo XML để cấu hình dự án, cấu hình các bean.

+ Code và build thành file war

- Cấu hình server Tomcat để chạy được file war vừa build

**-** Spring boot ra đời với các ưu điểm :

**+** Auto cofig

**+** Xây dựng các bean dựa trên annotation thay vì XML

**+** Server Tomcat được nhúng ngay trong file jar, không cần cấu hình server như Spring

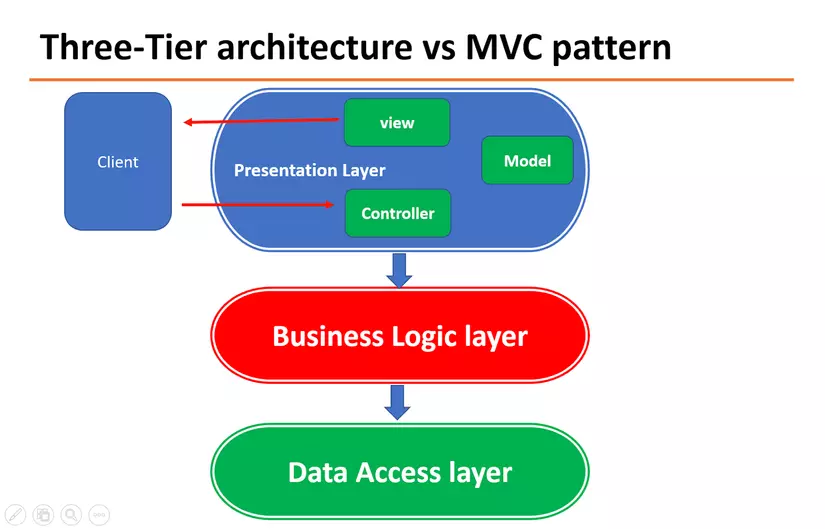
**=>** Spring Boot là một framework của Spring. Nó được phát triển để giảm bớt sự phức tạp trong quá trình cấu hình Spring truyền thống bằng cách tự động cấu hình và cung cấp các giải pháp mặc định cho các vấn đề phổ biến.

**=>** Spring tập trung nhiều vào business logic, nghĩa là mục tiêu của code làm được gì, Spring boot có nhiều thư viện có sẵn và cấu trúc code chuẩn mực.

1. **Spring boot code structure**

**-** Cấu trúc source code của Spring Boot dựa trên 2 mô hình là mô hình MVC và mô hình 3 lớp

* **Mô hình 3 lớp ( Three Tier) :**

****

Đây là mô hình tổ chức source code rất phổ biến trong Spring Boot. Cụ thể, ứng dụng được chia làm 3 tầng như sau :

+ **Presentation layer** : tầng này tương tác với người dùng, bằng View, Controller( trong MVC) hoặc API ( nếu có)

+ **Business logic layer** : Chứa toàn bộ logic của chương trình, các đa số code nằm ở đây

+ **Data access layer** : Tương tác với database, trả về kết quả cho tầng business logic

Trong Spring Boot, thì có một số thành phần đại diện cho từng lớp :

+ **Service** : chứa các business logic code

+ **Repository** : đại diện cho tầng Data Access

Còn Presentation layer thì chúng ta sẽ bàn tiếp qua mô hình MVC

* **Mô hình MVC :**

Do Spring Boot chỉ là wrapper cho Spring, chung ta vẫn sử dụng ngầm các module Spring khác bên dưới, Ví dụ như Spring MVC.

Mô hình MVC chia tầng Presentation làm 3 phần :

+ **Model** : các cấu trúc dữ liệu của toàn chương trình, có thể đại diện cho trạng thái của ứng dụng

+ **View** : lớp giao diện, dùng để hiển thị dữ liệu ra cho client xem và tương tác.

+ **Controller** : kết nối giữa Model và View, điều khiển dòng dữ liệu.

Dữ liệu từ Model qua Controller sau đó được gửi cho View hiển thị ra.Và ngược lại, khi có yêu cầu mới từ View, thì sẽ qua Controller thực hiện thay đổi dữ liệu của Model.

**NOTE :** Để xây dựng ứng dụng Spring Boot hoàn chỉnh cần kết hợp cả mô hình MVC và 3-Tier lại với nhau.

* **Các thành phần trong Spring Boot**

Sau khi kết hợp 2 mô hình lại, chúng ta có được ứng dụng Spring Boot hoàn chỉnh, gồm các thành phần sau :

+ **Controller** : trả về View ( có chứa data sẵn, dạng trang HTML), hoặc Model thể hiện dưới dạng API cho View ( View viết riêng bằng React, Vue hoặc Angular).

+ **Service** : chứa các code tính toán, xử lý. Khi Controller yêu cầu, thì Service tương ứng sẽ tiếp nhận và cho ra dữ liệu trả cho Controller ( trả về Model). Controller sẽ gửi về View như trên

+ **Repository** : Service còn có thể tương tác với service khác, hoặc dùng Repository để gọi DB. Repository là thằng trực tiếp tương tác, đọc ghi dữ liệu trong DB và trả cho service.

**Note :**  Nên giữ cho Controller càng ít code càng tốt, áp dụng nguyên tác SRP không nên để 1 Service thực hiện quá nhiều công việc nên tách ra nhiều Service.

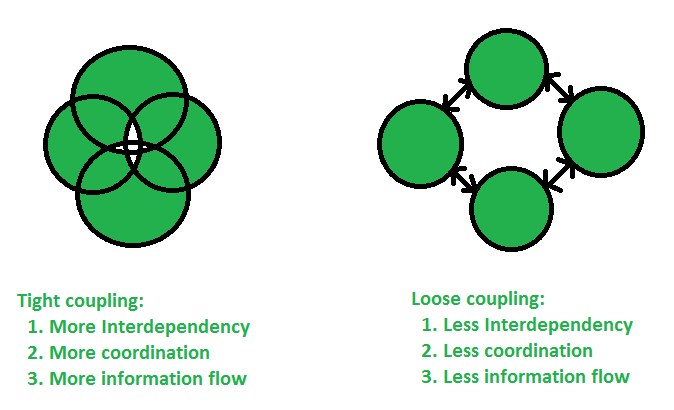
1. **Denpendency Injection trong Spring Boot**

Trước khi tìm hiểu về DI trong Spring chúng ta cần nắm vài khái niệm sau :

* **Module couping** : là mối quan hệ giữa hai module ( Object) với nhau, giữa chúng có sự phụ thuộc lẫn nhau. Gồm hai loại :

+ Tight Coupling : hai module liên kết chặt chẽ, khó tách rời

+ Loose Coupling : Hai module liên kết yếu, rời rạc



Để code dễ bảo trì và sửa đối, theo nguyên tắc giảm sự phục thuộc giữa hai module, chúng ta cần biến mối quan hệ giữa hai module từ tight coupling thành loose coupling.

* **Nguyên lý Dependency Inversion :**  DI là nguyên lý số năm trong SOLID, được đưa ra để thiết kế các module trong chương trình sao cho ít sự phụ thuộc nhất có thể, nội dung nguyên lý gồm :

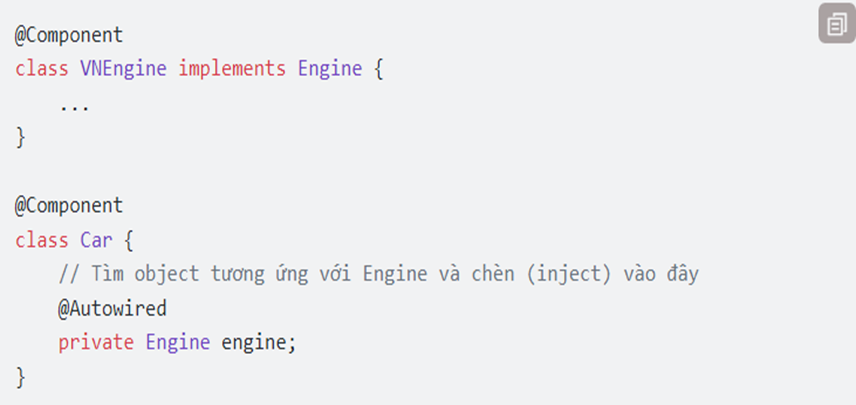
+ Các module cấp cao không nên trực tiếp phụ thuộc vào module cấp thấp. Cả hai nên phụ thuộc vào abstraction.

+ Abstraction không nên phụ thuộc vào chi tiết mà ngược lại : tức là Abstraction chỉ lấy những thuộc tính, những hành động chung nhất, mà không cần quan tâm chi tiết bên trong chúng hoạt động thế nào.

* **Khái niệm IoC ( Inversion of Control)**

- Khi thực hiện nguyên lý DI, khi chương trình của chúng ta có nhiều module ( Object) thì sẽ gặp các tình huống như : gắn thiếu module vào module khác, phải quan tâm đến thứ tự khởi tạo module, phụ thuộc vòng,…=> IoC ra đời nhằm mục đích đơn giản hóa quá trình tạo đối tượng và liên kết giữa chúng chúng bằng cách tuân theo nguyên tắc : **Không tạo đối tượng chỉ mô tả cách chúng sẽ được tạo ra**.

- Ví dụ :



**Giải thích** : Mỗi class được đánh dấu @Component sẽ được IoC hiểu là 1 module :

**+ @Component** : IoC Container sẽ tạo ra một module duy nhất ( Singleton)

**+ @Autowired** : là một tính năng của Spring, có nhiệm vụ tìm module tương ứng ( tạo từ trước) và inject vào đó.

**Note** : Nhờ có IoC , việc tạo và liên kết các module dễ dàng hơn nhiều, tuy nhiên IoC chỉ nên áp dụng trên các module, object lớn, còn object nhỏ nhỏ, linh tinh dùng tạm thời thì không nên dùng

* **Dependency Injection**

- Từ các khái niệm vừa tìm hiểu trên dễ dàng thấy Dependency Injection là một dạng thực hiện của IoC, bằng cách inject vào một module cần nó.

- Ví dụ : Khi xây dựng một ứng dụng, ta sẽ cần nhiều module liên kết với nhau để hoàn thành chức năng của ứng dụng đó.

+ Nếu một module cần sử dụng các thành phần của một module khác, chúng ta có thể sử dụng IoC để quản lý các thành phần này trong một IoC container. Khi một module cần sử dụng module khác, IoC sẽ tìm kiếm trong IoC container xem module đã được tạo chưa.

+ Nếu module đã được tạo IoC sẽ inject module đó vào module cần sử dụng và tiếp tục thực hiện chức năng của ứng dụng.

+ Ngược lại, nếu module chưa được tạo, IoC sẽ tạo mới module đó, bỏ vào IoC container và tiếp tục thực hiện chức năng của ứng dụng. Việc này được gọi là Dependency Injection, nó giúp cho việc quản lý các thành phần của ứng dụng trở nên dễ dàng hơn, giảm thiểu sự phụ thuộc giữa các module và giúp ứng dụng dễ dàng mở rộng hơn trong tương lai.

- Có hai loại injection :

+ **Constructor-based injection** : Dùng inject các module bắt buộc. Các module được inject nằm trong constructor, và được gán lần lượt vào các field.

+ **Setter-based injection** : Dùng inject các module tùy chọn. Mỗi module sẽ được inject thông qua setter, nằm ở tham số và cũng gán cho field nào đó.

* **Dependency Injection trong Spring Boot**

- Spring là một framework được xây dựng dựa trên nguyên lý DI. Bản thân Spring có chứa IoC container, có nhiệm vụ tạo và quản lý các module :

+ IoC container của Spring gọi là Application context

+ Các module chứa trong IoC container được Spring gọi là các **Bean**

- Spring Boot sử dụng các annotation dạng như **@Component** để đánh dấu lên class, chỉ ra rằng class đó cần tạo một module. Ngoài ra, còn có các annotation khác như **@Repository**, **@Controller**, **@Service**... cũng được đánh dấu là module

- Khi ứng dụng Spring boot chạy, thì IoC container sẽ thực hiện quá trình như sau :

+ Quét tìm (scan) các class được đánh dấu là Bean, và tạo một object singleton, bỏ vào IoC container

+ Khi có một Bean phụ thuộc vào Bean khác, thì IoC sẽ tìm trong container, nếu chưa có thì tạo, nếu đã có thì lấy ra và inject vào bean cần nó.

1. **Bean && Application context trong Spring Boot**

* **Bean là gì ?**

Như những gì đã học về DI trong Spring Boot dễ dàng định nghĩa được Bean là những module chính của chương trình, được tạo và quản lý bởi Spring IoC Container.

Vd : mỗi class được đánh dấu là @Component, @Controller, @Service,… sẽ được IoC hiểu là một Bean.

* **ApplicationContext là gì ?**

- Trong Spring Boot ApplicationContext dùng để đại diện cho Spring IoC Container, tương tự như Bean là đại diện cho các Dependency.

- Khi ứng dụng Spring chạy, Spring IoC container sẽ quét toàn bộ packages, tìm ra các bean và đưa vào ApplicationContext. Cơ chế đó là **Component scan.**

* **ComponentScan là gì ?**

- Khi ứng dụng Spring Boot bắt đầu chạy, thì nó sẽ tìm hết các class đánh dấu là bean trong chương trình và tạo bean. Quá trình tìm kiếm các bean này gọi là component scan.

- Componet sẽ tìm class đánh dấu @SpringBootApplication có chứa main method sẽ là nơi bắt đầu. Spring Boot sẽ tìm từ package này (package gốc) tìm xuống để tạo các Bean.

**Note** : Component scan sẽ tìm toàn bộ class ở package cùng cấp hoặc các package thấp hơn.

* **Khởi tạo @Bean bên trong @Configuration**

- Cách này dùng cho trường hợp bean cần thực hiện nhiều thao tác phức tạp để khởi tạo, hoặc có nhiều bean liên quan với nhau. Do đó, thay vì khởi tạo riêng rẽ từng class là từng bean, thì gom chung các bean cần khởi tạo lại bỏ vào class chứa là **@Configuration**.

- Thường thì các class đánh dấu **@Configuration** có hậu tố là Config.

- Khi Spring tìm thấy class **@Configuration**, nó sẽ tạo bean của class này trước (do @Configuration cũng là @Component). Trong khi tạo thì các logic khởi tạo cũng được thực thi, để chuẩn bị sẵn sàng tạo các **@Bean** bên trong.

- Sau đó Spring Boot sẽ tìm các method được đánh dấu @Bean bên trong @Configuration để tạo bean. Thường thì các bean dạng này ngắn và return ngay object chứ không phải để Spring Boot tạo ra.

1. **Spring boot ModelWrapper**

- ModelMapper là một thư viện Java, giúp đơn giản hóa code mapping các object. Mapping có thể hiểu là việc convert qua lại giữa hai object có cấu trúc gần giống nhau. - Trong Spring Boot có các dạng khác nhau của data, nhưng cấu trúc gần tương tự nhau, nên cũng dùng được thư viện này với chúng.

- Ví dụ : map giữa entity và model, entity và DTO,...

1. **Spring Boot Database – Spring Boot JPA**

* **Spring Data JPA là gì ?**

- JPA là viết tắt của **Java Persistence API**, nó là một đặc tả Java cho việc ánh xạ giữa các đối tượng Java với cơ sở dữ liệu quan hệ sử dụng công nghệ phổ biến là ORM (Object Relational Mapping) dược đề xuất bởi Oracle.

- JPA cung cấp đầy đủ các công cụ cho phép chúng ta có thể thao tác với cơ sở dữ liệu một cách đơn giản và nhanh chóng. JPA có thể dùng để persist một đối tượng Java (POJO – Plain Old Java Object) vào trong cơ sở dữ liệu hoặc lấy dữ liệu từ cơ sở dữ liệu và ánh xạ (mapping) ra các đối tượng Java một cách đơn giản.

JPA hoạt động như một cầu nối giữa các table/ các mối quan hệ giữa các table trong database và các class / mối quan hệ giữa các object.

**Ví dụ**: Table User với các column (Id, username, password) sẽ tương ứng với class User.java với các field Id, username, password. Từ đó mỗi khi truy vấn table hay các column ta sẽ gọi trực tiếp các phương thức trên các class, các field của class mà không cần quan tâm tới việc đang dùng loại database nào, kiểu dữ liệu database ra sao, …

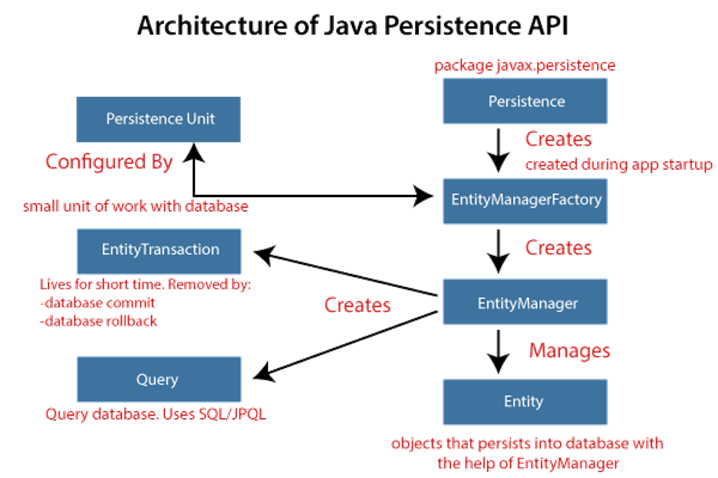
* **ORM là gì ?**

- ORM là viết tắt của Object Relational Mapping, là một công nghệ/ khái niệm/ quá trình chuyển đổi dữ liệu từ ngôn ngữ hướng đối tượng sang Database quan hệ và ngược lại. Ví dụ, trong Java nó được thực hiện với sự trợ giúp của Reflection và JDBC.

- ORM có khả năng xử lý các thao tác của nhiều loại cơ sở dữ liệu khác nhau một cách dễ dàng mà không quan tâm đến loại database sử dụng (SQL Server, MySQL, PostgreSQL, …) hay loại thao tác sử dụng (INSERT, UPDATE, DELETE, SELECT, …).

- JPA chỉ là một API định nghĩa các đặc tả cần thiết và không có code hiện thực từ những đặc tả đó. Nó chỉ chứa những hướng dẫn để hiện thực ORM. Do đó cần phải có một cài đặt ORM để hoạt động và persist các đối tượng Java. Các ORM Framework có thể sử dụng cho JPA như: Hibernate, iBatis, Eclipse Link, OpenJPA, ….

* **JPA Architecture ?**

****

+ **Entity Class**: Đây là các lớp Java được đánh dấu bằng các annotation của JPA để chỉ định rằng chúng là các đối tượng của JPA và được lưu trữ trong cơ sở dữ liệu. Mỗi entity class tương ứng với một bảng trong cơ sở dữ liệu.

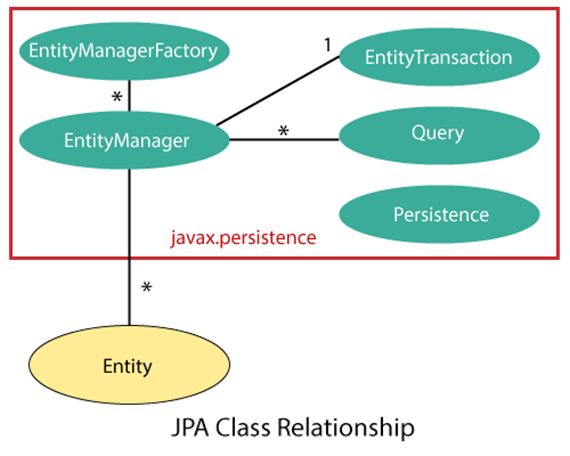
**EntityManager**: Đây là một đối tượng quản lý vòng đời của các entity object. Nó cung cấp các phương thức để thêm, xóa, sửa đổi và truy xuất các entity object.

**Persistence Unit**: Là đơn vị cấu hình JPA. Nó bao gồm thông tin về cơ sở dữ liệu, các entity class, các mapping của chúng với các bảng trong cơ sở dữ liệu và các thông số cấu hình khác.

**EntityManagerFactory**: Đây là một đối tượng được tạo ra từ Persistence Unit. Nó cung cấp các EntityManager.

**JPQL (Java Persistence Query Language)**: Là một ngôn ngữ truy vấn trừu tượng độc lập cơ sở dữ liệu sử dụng để tìm kiếm các đối tượng trong cơ sở dữ liệu.

* **JPA class relationships**

****

* **Lợi ích của JPA**
* JPA hỗ trợ plugable. (tức là hỗ trợ cho việc sử dụng nhiều third-party frameworks khác nhau, chẳng hạn như Hibernate hay MyBatis. Điều này cho phép người dùng có thể lựa chọn framework phù hợp để thực hiện các chức năng tương ứng với yêu cầu của ứng dụng)
* Hỗ trợ cấu hình thông qua annotation và xml.
* Giảm bớt số lớp yêu cầu cho việc phát triển persistence.
* Không cần phải viết các mô tả triển khai trong XML. Các Annotation dựa trên metadata đã hỗ trợ trong các ứng dụng JPA.
* Đã chuẩn hóa ORM và dễ dàng phát triển hơn.
* JPA hỗ trợ truy vấn động và tĩnh.
* Nhiều IDE hỗ trợ phát triển ứng dụng JPA và có thể tự động sinh code ánh xạ từ cơ sở dữ liệu thành các entity và ngược lại.
* **JPA trong Spring Boot**

**- Cài đặt ( dependency)** : spring-boot-starter-data-jpa.

**- JpaRepository** :

* Để sử dụng JPA chúng ta cần sử dụng interface JpaRepository
* Để sử dụng JpaRepository chúng ta cần cung cấp 2 thông tin : Entity ( tương ứng với table trong DB) và Primary key ( kiểu dữ liệu khóa chính).

**- Có thể tạo các câu Query với JPA ( Query creation)**

Trong Spring JPA, có một cơ chế giúp chúng ta tạo ra các câu Query mà không cần viết thêm code. Cơ chế này xây dựng Query dựa trên tên của method.

**Note** : Cơ chế xây dựng Query từ tên method này giúp chúng ta tiết kiệm thời gian với những query có logic đơn giản, và cũng đặc biệt hữu ích là nó giống ngôn ngữ con người thường nói hơn là SQL. (human-readable)

**- Quy tắc đặt tên method trong Spring JPA :**

***“Trong Spring JPA, cơ chế xây dựng truy vấn thông qua tên của method được quy định bởi các tiền tố sau : find…By, read…By, query…By, count…By, và get…By. Phần còn lại sẽ được parse theo tên của thuộc tính (viết hoa chữ cái đầu). Nếu chúng ta có nhiều điều kiện, thì các thuộc tính có thể kết hợp với nhau bằng chữ And hoặc Or”.***

- Ví dụ : chúng ta có 1 đối tượng là Person gồm : id, name, age

+ Tạo query mức cơ bản : tìm 1 person theo name

**Person findByName(String name) or Person findPersonByName(String name)**

+ Tạo query mức trung bình :

Tìm danh sách person theo tên và tuổi

**List<Person> findByNameAndAge(String name, Int age)**

Dùng findDistinct : tìm kiếm và loại bỏ các đối tượng trùng nhau

**List<Person> findDistinctByName (String name)**

Dùng IgnoreCase or AllIgnoreCase: để tìm kiếm không phân biệt hoa thường

**List<Person> findByNameIgnoreCase(String name)**

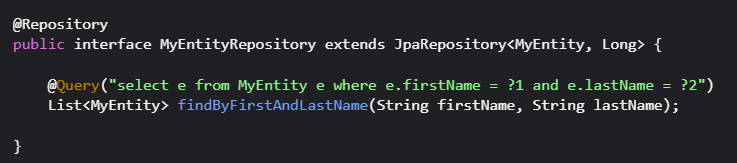
Dùng OrderBy ( Asc, Desc) : sắp xếp tăng dần, giảm dần

**List<Person> findByIdOrderbyNameDesc(Int id)**

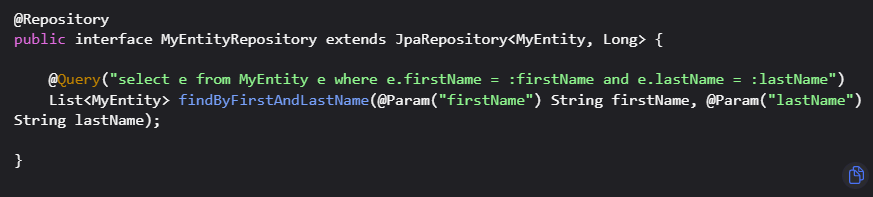
- Nếu khó khăn trong việt tạo query bằng tên method như trên, Spring Jpa còn hỗ trợ 1 cách nguyên thủy khác với cách sử dụng @Query, chúng ta có thể sử dụng câu truy vấn JPQL (Hibernate) hoặc raw SQL.

- Tham số có thể truyền vào câu lệnh query nguyên thủy qua 2 cách :

+ Theo thứ tự method ?1, ?2 …



+ Đặt tên cho tham số



* **Tổng thể**

Tổng thể, JPA là một công nghệ quản lý cơ sở dữ liệu quan hệ, được đề xuất bởi Oracle (trước đây là Sun), cung cấp các công cụ hỗ trợ cho việc lấy và lưu trữ dữ liệu. Nó cũng hỗ trợ kết nối và tương tác với các third-party frameworks khác như Hibernate hay MyBatis, giúp cho việc phát triển ứng dụng trở nên dễ dàng hơn.

1. **Spring Boot RESTful API with JPA + mySQL**

- API là phương thức giao tiếp giữa các thiết bị

- REST API ( hay còn gọi là RESTful API) là một kiểu thiết kế API, được quản lý và truyền tải dưới dạng tài nguyên độc lập.

- Spring Boot là một framework phát triển ứng dụng web của Java, được thiết kế để giảm thiểu khó khăn trong việc triển khai và cấu hình ứng dụng web. Nó có thể giúp cho việc triển khai các ứng dụng RESTful API trở nên dễ dàng hơn với các annotation và thư viện tích hợp sẵn.

**7.1 Các Annotation sử dụng trong Spring Boot để xây dựng RESTful API**

* @RestController: được sử dụng để đánh dấu một lớp là một RESTful Controller. Lớp được đánh dấu bởi @RestController sẽ trả về các tài nguyên dưới dạng JSON/XML thay vì các trang HTML.
* @GetMapping: được sử dụng để đánh dấu một phương thức là một endpoint cho phép phương thức GET. Phương thức được đánh dấu bởi @GetMapping sẽ xử lý các yêu cầu GET từ client và trả về các tài nguyên tương ứng.
* @PostMapping: được sử dụng để đánh dấu một phương thức là một endpoint cho phép phương thức POST. Phương thức được đánh dấu bởi @PostMapping sẽ xử lý các yêu cầu POST từ client và lưu trữ các tài nguyên mới.
* @PutMapping: được sử dụng để đánh dấu một phương thức là một endpoint cho phép phương thức PUT. Phương thức được đánh dấu bởi @PutMapping sẽ xử lý các yêu cầu PUT từ client và cập nhật các tài nguyên tương ứng.
* @DeleteMapping: được sử dụng để đánh dấu một phương thức là một endpoint cho phép phương thức DELETE. Phương thức được đánh dấu bởi @DeleteMapping sẽ xử lý các yêu cầu DELETE từ client và xóa các tài nguyên tương ứng.

- Để xử lý các tham số đầu vào của RESTful API, Spring Boot hỗ trợ nhiều annotation khác nhau, trong đó bao gồm:

* @PathVariable: @PathVariable được sử dụng để lấy các giá trị path variable từ URL của request trong Spring Boot.

**Note** : Tên của tham số trong @PathVariable phải khớp với tên của tham số trong URL.

* @RequestBody: @RequestBody được sử dụng trong Spring Boot để lấy thông tin được gửi đến từ phía client qua HTTP request body và chuyển đổi nó thành một đối tượng Java tương ứng.

**Note** : đối tượng được tạo ra từ request body cần phải có các thuộc tính và phương thức getter/setter tương ứng để Spring Boot có thể chuyển đổi dữ liệu từ request body thành đối tượng Java.

* @RequestParam: @RequestParam được sử dụng trong Spring Boot để lấy các tham số (parameters) được gửi đến từ phía client qua HTTP request URL.

**Note** : Tên của tham số trong @RequestParam phải khớp với tên của tham số trong URL.

**7.2 Demo RESTful API with Spring Boot**

Xây dựng một RESTful API đơn giản để quản lý danh sách sản phẩm (id, name, price) và lưu trữ các sản phẩm vào cơ sở dữ liệu MySQL

* Bước 1 : Cấu hình cơ sở dữ liệu MySQL trong file application.properties
* Bước 2 :Tạo một model để đại diện cho đối tượng sản phẩm
* Bước 3 : Tạo một repository để thao tác với cơ sở dữ liệu MySQL
* Bước 4 : Tạo một service để thực hiện các thao tác liên quan đến sản phẩm
* Bước 5 : Tạo một controller để xử lý các yêu cầu tới RESTful API
* Bước 6 : Kiểm tra API có thể kiểm tra API bằng cách sử dụng các công cụ như Postman hoặc curl.

**8. Spring boot – Exception**

Exception cơ bản là đối tượng đại diện cho một lỗi xảy ra khi chạy chương trình. Lỗi này có thể do tác động bên ngoài hoặc chính trong code ném ra. Nhiệm vụ của chúng ta là phải "bắt" và "xử lý" chúng một cách thích hợp (đưa ra thông báo lỗi,...). Nếu exception không được xử lý, nó có thể làm chương trình bị dừng.

**Note :**

- Dùng Spring AOP để inject đoạn code logging vào trước và sau mỗi Controller method

- Loại bỏ try catch finally, để việc xử lý exception cho @ExceptionHandler

- Tránh lạm dụng class ResponseEntity<T> quá nhiều

**9. Spring boot – Validation**

- Validation là hành động kiểm tra tính hợp lệ của dữ liệu. Việc này giúp cho hệ thống hoạt động an toàn, đảm bảo, tránh xử lý dữ liệu lỗi,... và cũng giúp ngăn chặn một số cuộc tấn công phổ biến như SQL injection hoặc XSS.

- Ví dụ khi nhập dữ liệu vào form đăng kí, cần thực hiện validate dữ liệu form để đảm bảo:

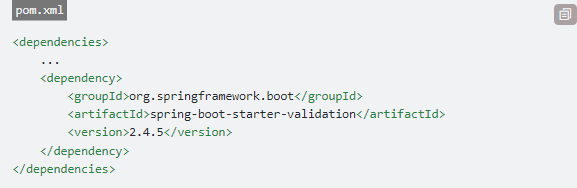
* Trường username không được trống
* Password phải đủ độ khó
* Email phải đúng định dạng email
* **Nên thực hiện validation ở client hay server?**

Nên validate ở cả hai, chúng có lợi ích riêng như :

* Validate ở client thì có thể nhanh chóng thông báo cho user, giúp giảm việc chờ đợi server phản hồi
* Validate ở server cung cấp thêm một lớp nữa để đảm bảo rằng dữ liệu luôn hợp lệ.
* Nếu không có validation ở server, thì hacker có thể gửi những request độc hại đến, gây lỗi, sập server,... (không thông qua web, app,... thì không bị dính validate ở client).
* **Các cách thực hiện Validation**

- Thêm validation thủ công : hầu hết các framework như Java, C#, PHP đều có thư viện hỗ trợ cho việc validate dữ liệu, chúng ta chỉ thực hiện thêm thủ công với trường hợp thư viện không hỗ trợ như kiểm tra thêm mới sản phẩm với điều kiện tên sản phẩm nhập vào không được trùng.

- Dùng thư viện dependencies **spring-boot-starter-validation**, nó cung cấp các validation API dưới dạng annotation để tiện sử dụng.

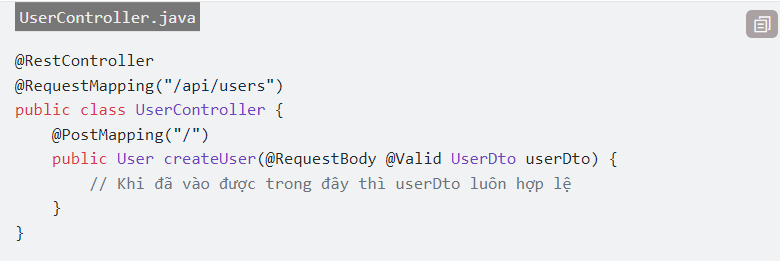


- Quy trình thực hiện validation gồm 2 bước :

* Thêm các annotation ràng buộc trên các field của class nào đó. Mỗi annotation có ý nghĩa riêng, ví dụ @NotNull, @NotEmpty, @Email,...



* Class đó được dùng ở vị trí nào cần đảm bảo hợp lệ, ví dụ truyền cho method làm tham số, mà tham số phải hợp lệ rồi mới thực hiện method, thì thêm @Valid hoặc @Validated trên tham số (thuộc class đó).



Ở đoạn code trên, nếu dữ liệu không hợp lệ, thì method createUser sẽ không được gọi. Lúc này là validation đã bị fail.

**- Xử lý lỗi khi validation fail**

Spring Boot có hai cách để xử lý việc này :

* Dùng thêm 1 tham số cuối cùng là BindingResult. Nếu validation fail, method vẫn sẽ được gọi vào, và chúng ta có thể check tham số BindingResult kia có chứa lỗi hay không, từ đó xử lý phù hợp.



* Nếu không, chương trình ném ra lỗi BindException. Chúng ta chỉ cần bắt lỗi lại với @ExceptionHandler (ở bài trước) và xử lý phù hợp.



**8. Spring Boot – Thymeleaf**

Thymeleaf là một Java Template Engine. Có nhiệm vụ xử lý và generate ra các file HTML, XML, v.v..

* **Syntax :**

- Cú pháp của **Thymeleaf** sẽ là một **attributes** (Thuộc tính) của thẻ HTML và bắt đầu bằng chữ **th:**

- Với cách tiếp cận này, chúng ta sẽ chỉ cần sử dụng các thẻ HTML cơ bản đã biết mà không cần bổ sung thêm syntax hay thẻ mới như JSP truyền thống.

* **Model & View**

Model là đối tượng lưu trữ thông tin và được sử dụng bởi Template Engine để generate ra webpage. Có thể hiểu nó là Context của Thymeleaf.

**Note** : Model lưu giữ thông tin dưới dạng key-value.

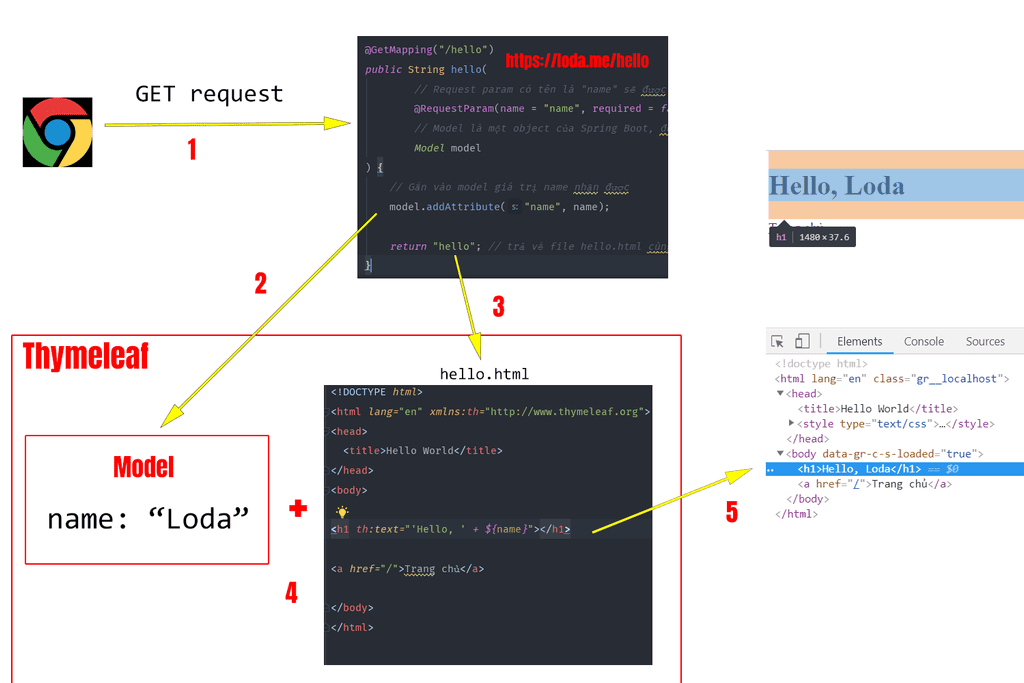
Trong template thymeleaf, để lấy thông tin trong Model cần sử dụng các Thymeleaf Standard Expression.

+ ${..} : Gía trị của một biến

+ \*{…} Gía trị của một biến được chỉ định ( dùng trong th:each )

+ #{…} : lấy message

+ @{…} : Lấy đường dẫn URL dựa theo context của server



**9. Spring boot SB Caching**

SB Caching là một tính năng được cung cấp bởi Spring Boot để giúp ứng dụng của chúng ta cải thiện hiệu suất bằng cách lưu trữ dữ liệu tạm thời trong bộ nhớ trong thay vì truy cập vào cơ sở dữ liệu hoặc tính toán lại dữ liệu mỗi lần yêu cầu. Điều này giúp giảm thiểu thời gian phản hồi của ứng dụng và tăng tốc độ xử lý.