# JAVA CORE

Link học java core:

* <https://v1study.com/java-methods-of-the-patternsyntaxexception-class.html>
* <https://www.phongvanit.com/ky-nang/java-1001>
* <https://niithanoi.edu.vn/hoc-java-trong-7-ngay.html>

## Quản lý bộ nhớ trong java

<https://gochocit.com/ngon-ngu-lap-trinh/quan-ly-bo-nho-trong-java-bo-nho-stack-va-bo-nho-heap#1-1-kien-truc-bo-nho-trong-java>

## Khái niệm Static

<https://topdev.vn/blog/tu-khoa-static-va-final-trong-java/>

* Static được cấp phát bộ nhớ cố định trong class (method, class hoặc biến), lưu ở heap
* Từ khoá static biểu thị cho biến hoặc phương thức có thể được truy cập (sử dụng) mà không cần tạo ra thực thể của lớp chứa nó. Người dùng không thể override phương thức static trong Java, bởi vì kỹ thuật đè (overriding) phương thức được dựa trên quá trình gán (binding) động khi runtime (khi chương trình đang chạy) và những phương thức static  được gán tĩnh trong thời gian biên dịch. Phương thức tĩnh không ràng buộc với thực thể của lớp nên phương thức tĩnh sẽ không thể override.
* Static thì ko sử dụng được từ khóa this, super;
* Không thể truy cập biến non-static trong 1 trong nó, vì static được cấp phát cố định mà các instance Object( chứa static cũng đc) lại không được cấp phát cố định bộ nhớ.

## Kiểu Wrapper {Integer,String,Long,Byte,..}

Nó là final class { m ko thể tạo 1 lớp con kế thừa từ nó, nó là lớp ko có con}

## Từ khóa fetch trong java

LAZY bản chất nó sẽ lấy dữ liệu sau khi hàm getter đc gọi , nên là nếu sout thì sẽ ko nhìn thấy có dữ liệu trong Object cha chứa nó.

EAGER bản chất sẽ lấy dữ liệu luôn ngay khi gọi object cha chứa nó.

## Quan hệ HAS-A, IS-A.

Nếu class A **extends** class B: thì đó là quan hệ **is-a**. Vì sao? Is-a có nghĩa là cái này là cái kia. Khi A kế thừa B thì A sẽ có những tính chất của B, nói cách khác A là một bản sao có các tính chất của B.

Nếu class A **implements** interface B: thì đó là quan hệ **has-a**. Vì sao? Has-a có nghĩa là cái này có khả năng gì đó. Khi A implements (triển khai - từ này dịch hơi ngựa) B thì là lớp A có các khả năng được mô tả trong các hàm của interface B.

## . Collection và Collections khác gì nhau

* Collections là 1 class chứa các static method để thao tác vs tập dữ liệu
* colection thì là 1 interface trong collection framework

## Abtract class và Interface

<https://viblo.asia/p/interface-vs-abstract-class-ke-tam-lang-nguoi-nua-can-07LKX9JeZV4>

Abstract class: nên có ít nhất 1 abstract method, có đẩy đủ tính năng như 1 class bình thường, có các access modifier cho abstract method, chỉ extend đc 1 abstract class, có constructor

[10:05](https://techasiansjsc.slack.com/archives/D03JJ4P6TNE/p1658070309068449)

Interface: chỉ chứa biến static final, chứa các method trìu tượng, mặc định là public, có thể extends nhiều interface khác, ko có constructor, default method đc implement trong interface và override ở class con, static method đc implement nhưng không đc override, ko thể khai báo interface là private và protected

1 class extends 1 class và implement nhiều interface

## Bộ nhớ trong java

<https://gochocit.com/ngon-ngu-lap-trinh/quan-ly-bo-nho-trong-java-bo-nho-stack-va-bo-nho-heap>

lưu trữ các biến local và các method,

* Stack ( Lưu biến cục bộ)

stack có kích thước nhỏ,tự tăng kích thước khi method đc gọi là giảm khi method thực hiện xong,các biến local chỉ tồn tại khi method đang chạy, Truy cập nhanh hơn Heap

* Mỗi thread là 1 stack
* Lỗi là stackoverflowerror
* ưu trữ các đối tượng, biến static, biến toàn cục
* ko tự giải phóng bộ nhớ, phải dùng GC kích thước lớn, Truy cập lâu hơn stack
* chia sẻ cho tất cả các luồng Lỗi outofmemoryerror

Các object mới đc khởi tạo sẽ đc lưu ở heap và các tham chiếu đến chúng đc lưu ở stack

Stack: - lưu trữ các biến local và các method, -  stack có kích thước nhỏ, - tự tăng kích thước khi method đc gọi là giảm khi method thực hiện xong, -  các biến local chỉ tồn tại khi method đang chạy - Truy cập nhanh hơn Heap - Mỗi thread là 1 stack - Lỗi là stackoverflowerror

Heap: - lưu trữ các đối tượng, biến static, biến toàn cục - ko tự giải phóng bộ nhớ, phải dùng GC -  kích thước lớn, - Truy cập lâu hơn stack -  chia sẻ cho tất cả các luồng - Lỗi outofmemoryerror Các object mới đc khởi tạo sẽ đc lưu ở heap và các tham chiếu đến chúng đc lưu ở stack

* HEAP

Lưu biến static

Lưu các các object đc khởi tạo với từ khóa new 🡪 biến global cũng sẽ đc khởi tạo theo sau

Bộ nhớ sẽ lớn hơn stack

Dung lượng sử dụng của heap sẽ tăng giảm phụ thuộc vào các object được tạo ra.

Thời gian tồn tại của object phụ thuộc vào Garbage Collector (GC) của Java. Khi một object bị **null**hoặc không còn được sử dụng thì GC sẽ xóa object khỏi bộ nhớ.

Dung lượng vùng **heap** thường lớn hơn vùng nhớ **stack**.

## Hướng đối tượng là gì

<https://topdev.vn/blog/oop-la-gi/>

**Định nghĩa OOP là gì?**

OOP (từ viết tắt của -Object Oriented Programming)\_lập trình hướng đối tượng là một phương pháp lập trình dựa trên các khái niệm về lớp và đối tượng. OOP thường tập trung vào các đối tượng thao tác hơn là logic để có thể thao tác chúng.

OOP là một nền tảng quen thuộc của các design pattern hiện nay. OOP đặt ra mục tiêu quản lý source code giúp gia tăng khả năng tái sử dụng và quan trọng hơn hết là có thể tóm gọn được các thủ tục đã biết trước tính chất thông qua quá trình sử dụng các đối tượng.

## Database là gì

Database là 1 tập hợp các dữ liệu có quan hệ với nhau là các bảng

Bảng là tập hợp dữ liệu có cùng định dạng với nhau, các bảng có thể có quan hệ với nhau 1-1,1-n,n-1,n-n;

## 11 Quy tắc viết Code ( CleanCode )

<https://techmaster.vn/posts/36509/13-quy-uoc-can-thiet-de-viet-clean-code>

## 12 Tính năng mới của java đọc ở đây

<https://viettuts.vn/java-new-features/tuy-chinh-annotation-trong-java>

## 13. Annotation là gì

<https://codelearn.io/sharing/annotation-trong-java-la-gi>

<https://docs.oracle.com/javase/tutorial/java/annotations/index.html>

<https://viettuts.vn/java-new-features/tuy-chinh-annotation-trong-java>

**Annotation trong Java là gì?**

**Annotation** được hiểu là một dạng **chú thích** hoặc  một dạng **siêu dữ liệu** (metadata) được dùng để cung cấp thông tin dữ liệu cho mã nguồn Java. Các chú thích không có ảnh hưởng trực tiếp đến hoạt động của mã mà chúng chú thích. Annotation được thêm vào Java từ Java 5.

Các Annotation được sử dụng trong mã nguồn sẽ được biên dịch thành bytecode và sử dụng kỹ thuật phản chiếu (Reflection) để truy vấn thông tin siêu dữ liệu và đưa ra hành động thích hợp.Chúng ta có thể chú thích các lớp (class), phương thức (method), các biến (variable), các gói (package) và các tham số (prameter) trong Java.

Annotation được sử dụng trong Java cho 3 mục đích chính **(3 Loại):**

* ***Thứ nhất*, chỉ dẫn cho trình biên dịch (Compiler)**

1. @Deprecated

2. @Override

3. @SuppressWarnings

**- Thứ hai, chỉ dẫn trong thời điểm biên dịch (Build-time)**

Các công cụ phần mềm có thể thông qua các chỉ dẫn của chú thích để tạo mã nguồn, tệp XML, nén mã biên dịch và các tâp tin vào một tập tin, v…v

* ***Thứ ba*, chỉ dẫn trong thời gian chạy (Runtime)**

Thông thường, các Annotation không có mặt trong mã Java sau khi biên dịch. Tuy nhiên, có thể xác định trong thời gian chạy bằng cách sử dụng kỹ thuật Reflection và có thể sử dụng để đưa ra những hướng dẫn cho chương trình trong thời gian chạy.

#### 2. Java Annotation được tích hợp sẵn được sử dụng trong Annotation khác.

Một Annotation sẽ đượng định nghĩa bởi các **Meta-Annotations**. Các Meta-Annotations gồm @Retention, @Target, @Documented, @Inherited.  🡪 Cụ thể

* **@Target:** Dùng để chú thích **phạm vi sử dụng** của một Annotation.  Các chú thích này đã được định nghĩa trong enum**java.lang.annotation.ElementType**:

|  |  |
| --- | --- |
| **ElementType.TYPE** | Chú thích trên Class, interface, enum, annotation |
| **ElementType.FIELD** | Chú thích trường (field), bao gồm cả các hằng số enum. |
| **ElementType.METHOD** | Chú thích trên method. |
| **ElementType.PARAMETER** | Chú thích trên parameter. |
| **ElementType.CONSTRUCTOR** | Chú thích trên constructor. |
| **ElementType.LOCAL\_VARIABLE** | Chú thích trên biến địa phương. |
| **ElementType.ANNOTATION\_TYPE** | Chú thích trên Annotation khác. |
| **ElementType.PACKAGE** | Chú thích trên package. |

* **@Retention:**Dùng để chú thích **mức độ tồn tại** của một Annotation nào đó. Cụ thể có 3 mức nhận thức tồn tại của vật được chú thích, và được định nghĩa trong enum **java.lang.annotation.RetentionPolicy**:

|  |  |
| --- | --- |
| **RetentionPolicy.SOURCE** | Tồn tại trên mã nguồn, và không được trình biên dịch nhận ra. |
| **RetentionPolicy.CLASS** | Mức tồn tại được trình biên dịch nhận ra, nhưng không được nhận biết bởi máy ảo tại thời điểm chạy (Runtime). |
| **RetentionPolicy.RUNTIME** | Mức tồn tại lớn nhất, được trình biên dịch nhận biết, và máy ảo (JVM) cũng nhận ra khi chạy chương trình. |

* **@Inherited: C**hú thích chỉ ra rằng loại chú thích có thể được kế thừa từ lớp cha và có giá trị mặc định là **false**. Khi người dùng truy vấn kiểu Annotation của lớp con và lớp con không có chú thích cho kiểu này thì lớp cha của lớp được truy vấn cho loại chú thích sẽ được gọi. Chú thích này chỉ áp dụng cho các khai báo class.
* Theo mặc định, các annotation không được kế thừa cho các lớp con. Annotation @Inherited được thừa kế đánh dấu annotation này được kế thừa cho các lớp con.

|  |
| --- |
| import java.lang.annotation.Inherited;    @Inherited  @interface ForEveryone {      int value();  }    @ForEveryone(10)  class Superclass {}    class Test extends Superclass {} |

* **@Documented:** Chú thích này chỉ ra rằng chú thích mới nên được bao gồm trong tài liệu Java được tạo ra bởi các công cụ tạo tài liệu Java.

## 14 Fail Fast và Safe Fast trong java

* Tóm tắt thfi nó là ( khi dugf Collection trong vòng lặp thì ko thể modify element ( FailFast), trừ khi dùng iterator để xóa thì nó là SafeFast

<https://www.geeksforgeeks.org/fail-fast-fail-safe-iterators-java/>

## 15 Iterator, và Iterable trong java

<https://stackoverflow.com/questions/6863182/what-is-the-difference-between-iterator-and-iterable-and-how-to-use-them#:~:text=An%20Iterable%20is%20a%20simple,the%20object%20with%20iteration%20state>.

Iterator và Iterable có sự liên quan chặt chẽ với nhau.

Ta thử Coi **Iterator** là một giao diện giúp chúng ta duyệt qua bộ sưu tập với sự trợ giúp của một số phương thức không xác định như *hasNext (), next () và remove ()*

Mặt khác, **Iterable** là một giao diện khác, nếu được triển khai bởi một lớp sẽ buộc lớp đó có thể lặp và là mục tiêu cho cấu trúc For-Each. Nó chỉ có một phương thức có tên iterator () xuất phát từ chính giao diện Iterator.

<https://codegym.vn/blog/2020/04/29/huong-dan-su-dung-iterator-trong-java/#:~:text=Iterator%20v%C3%A0%20Iterable%20c%C3%B3%20s%E1%BB%B1%20li%C3%AAn%20quan%20ch%E1%BA%B7t%20ch%E1%BA%BD%20v%E1%BB%9Bi%20nhau.&text=M%E1%BA%B7t%20kh%C3%A1c%2C%20Iterable%20l%C3%A0%20m%E1%BB%99t,t%E1%BB%AB%20ch%C3%ADnh%20giao%20di%E1%BB%87n%20Iterator>.

## 16. Comparetor và Comparable trong java

<https://viettuts.vn/java-collection/su-khac-nhau-giua-comparable-va-comparator>

theo t thấy là Compartor là function interface, Còn Comparable là interface chứa các function CompareTo, ..

|  |  |
| --- | --- |
| 1) Bạn phải implements giao tiếp Comparable cho lớp đối tượng cần được so sánh. | Bạn **KHÔNG** phải implements giao tiếp Comparator cho lớp đối tượng cần được so sánh. |
| 2) Comparable cung cấp **phương thức compareTo()**để sắp xếp các phần tử. | Comparator cung cấp **phương thức compare()**để sắp xếp các phần tử. |
| 3) Comparable thuộc về **java.lang** package. | Comparator thuộc về **java.util** package. |
| 4) Chúng ta có thể sắp xếp các phần tử của kiểu Comparable bởi **phương thức Collections.sort(List)** . | Chúng ta có thể sắp xếp các phần tử của kiểu Comparator bởi **phương thức Collections.sort(List,Comparator)**. |