## JAVA CORE

## Khái niệm Static

Static được cấp phát bộ nhớ cố định trong class (method hoặc biến)

Static thì ko cần this;

## 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;

## Collections Class (1 lớp hỗ trợ các Collection FrameWork) và Interface Collection Trong java

<https://gpcoder.com/2783-lop-collections-trong-java-collections-utility-class/#:~:text=Collection%20l%C3%A0%20m%E1%BB%99t%20interface%20c%E1%BA%A5p,Map%2C%20Set%2C%20%E2%80%A6>).

* <https://gpcoder.com/2783-lop-collections-trong-java-collections-utility-class/>

Các hàm cần biết:

* Thuộc tính **EMPTY\_LIST**,**EMPTY\_SET**,**EMPTY\_MAP nó là final của class này ko thể thay đổi**

**Nếu cố tình thêm =>** throw UnsupportedOperationException

* **addAll()** để thêm tất cả các phần tử của một danh sách này vào một danh sách khác

|  |
| --- |
| * **public** **static** **void** main(String a[]) { * List<String> list = **new** ArrayList<String>(); * list.add("C"); * list.add("Core Java"); * list.add("Advance Java"); * System.out.println("Initial collection value:" + list); * Collections.addAll(list, "Servlet", "JSP"); * System.out.println("After adding elements collection value:" + list); * String[] strArr = { "C#", ".Net" }; * Collections.addAll(list, strArr); * System.out.println("After adding array collection value:" + list); * } |

## Sự khác nhau giữa SET và MAP, ArrayList và LinkedList trong java

<https://viettuts.vn/interview/list-cau-hoi-phong-van-java-collection>

## Iterator và ListInterator Trong java

<https://openplanning.net/13443/java-listiterator#:~:text=ListIterator%20l%C3%A0%20m%E1%BB%99t%20interface%20con,t%E1%BB%AD%20trong%20qu%C3%A1%20tr%C3%ACnh%20l%E1%BA%B7p>.

## WildCard trong java

<https://www.geeksforgeeks.org/wildcards-in-java/?ref=lbp>

nhưng ko thể sử dụng các hàm phía bên dưới List<? Supper| extends A> Nghĩa là ko xác định trước dữ liệu.

*Use extend wildcard when you want to get values out of a structure and super wildcard when you put values in a structure. Don’t use wildcard when you get and put values in a structure. You can specify an upper bound for a wildcard, or you can specify a lower bound, but you cannot specify both.*

*=>*

Sử dụng ký tự đại diện mở rộng khi bạn muốn lấy các giá trị ra khỏi cấu trúc và ký tự đại diện siêu cấp khi bạn đặt các giá trị vào một cấu trúc. Không sử dụng ký tự đại diện khi bạn lấy và đặt các giá trị trong một cấu trúc. Bạn có thể chỉ định giới hạn trên cho ký tự đại diện hoặc bạn có thể chỉ định giới hạn dưới, nhưng bạn không thể chỉ định cả hai.

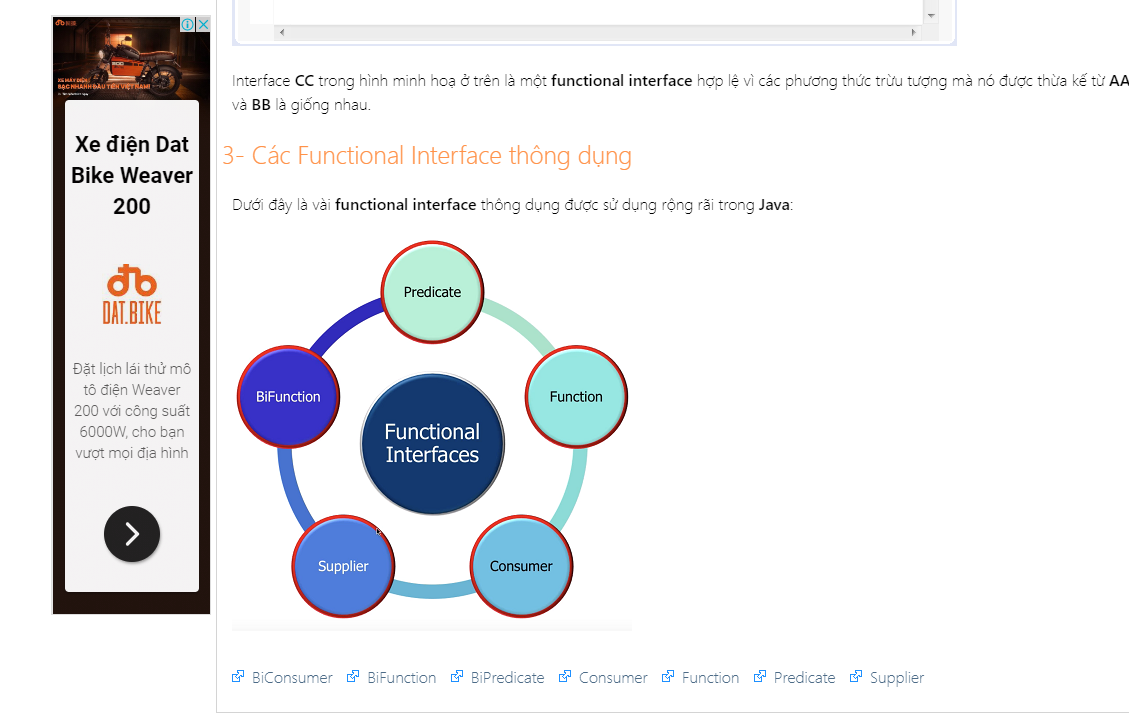
## Lớp Object;

Lớp Object trong gói java.lang là lớp nằm tại gốc của cây phân cấp lớp. Tất cả các lớp khác được tạo ra đều là hậu duệ trực tiếp hoặc gián tiếp của lớp Object. Và do đó sẽ kế thừa tất cả các phương thức của lớp Object, mặc dù chúng ta cũng ít khi dùng các phương thức của lớp này, tuy nhiên nếu phải dùng thì chúng ta nên override lại các phương thức đó.

## 16.Các phương thức của lớp Object

Lớp Object cung cấp các phương thức như trong bảng sau:

|  |  |
| --- | --- |
| **Phương thức** | **Mô tả** |
| public final Class getClass() | trả về đối tượng lớp Class của đối tượng hiện tại. Từ lớp Class đó có thể lấy được các thông tin metadata của class hiện tại. |
| public int hashCode() | trả về số hashcode cho đối tượng hiện tại. |
| public boolean equals(Object obj) | so sánh đối tượng đã cho với đối tượng hiện tại. |
| protected Object clone() throws CloneNotSupportedException | tạo và trả về bản sao chép (clone) của đối tượng hiện tại. |
| public String toString() | trả về chuỗi ký tự đại diện của đối tượng hiện tại. |
| public final void notify() | đánh thức một luồng, đợi trình giám sát của đối tượng hiện tại. |
| public final void notifyAll() | đánh thức tất cả các luồng. đợi trình giám sát của đối tượng hiện tại. |
| public final void wait(long timeout)throws InterruptedException | làm cho Thread hiện tại đợi trong khoảng thời gian là số mili giây cụ thể, tới khi Thread khác thông báo (gọi phương thức notify() hoặc notifyAll()). |
| public final void wait(long timeout,int nanos)throws InterruptedException | làm cho Thread hiện tại đợi trong khoảng thời gian là số mili giây và nano giây cụ thể, tới khi Thread khác thông báo (gọi phương thức notify() hoặc notifyAll()). |
| public final void wait()throws InterruptedException | làm Thread hiện tại đợi, tới khi Thread khác thông báo (gọi phương thức notify() hoặc notifyAll()). |
| protected void finalize()throws Throwable | Được gọi bởi Garbage Collector trước khi đối tượng bị dọn rác. |

\\ tìm hiểu them mấy cái ở trên đi nhé @@

## Comparator trong java

Nên nhớ là các kiểu biến Số, Ngày tháng đều có hộ trợ:

**phương thức so sánh compareTo Vì nó kế thừa lớp :: Compareable**

Cách 1: Dùng Annonyous Class ( thực ra cái Comparator kia chính là 1 functionInterface)

index = *BinarySearch*(*listInteger*, 13, **new** Comparator<Integer>() {

@Override

**public** **int** compare(Integer o1, Integer o2) {

**return** o1.compareTo(o2);

}

Cách 2: Dùng lamda expression trong java 8

Với cách này vì ko khai báo generic trước nên ko dùng được phương thức compareTo như cách 1. Phải xem lại lamda8 .

**int** index = *BinarySearch*(*listInteger*, 10, (i1, i2) -> {

**if**(i1 > i2) **return** 1;

**else** {

**if**(i1 == i2) **return** 0;

**else** **return** -1;

}

});

Cách 3: Thực ra Comparator còn hỗ trợ cả phương thức compareInt

index = *BinarySearch*(*listInteger*, 12,Comparator.*comparingInt*(Integer :: **new**));

## Stream trong java 8

<https://www.tutorialspoint.com/java8/java8_streams.htm>

## Sự khác nhau giữ iterator và iterable

<https://www.wake-up-neo.net/vi/java/su-khac-biet-giua-iterator-va-iterable-la-gi-va-lam-nao-de-su-dung-chung/940579806/>

* Iterator là một giao diện, có triển khai để lặp qua các phần tử.
* Iterable là một giao diện cung cấp Iterator.

Trong interface Iteratable chứa interface Iterator.

## Castting và Instaneof

Upcasting và Downcastting

<https://gpcoder.com/2406-co-che-upcasting-va-downcasting-trong-java/>

* 1 lớp con là biểu hiện (instance của chính nó và lớp cha) vì cơ chế casting yêu cầu đúng biểu hiện instance

Instanceof

<https://gpcoder.com/2318-toan-tu-instanceof-trong-java/>

## Custom Lombol với Accesstor

<https://shareprogramming.net/custom-getter-setter-voi-accessors-trong-lombok/>

*@Accessors* annotation được sử dụng để tùy biến cách lombok generate getter và setter method với 3 option:

* **fluent**
* **chain**
* **prefix**
  1. **Khi fluent option là true lombok sẽ loại bỏ tiền tố get, set cho getter và setter method.**
  2. **Khi chain option là true giống như @Builder có thể set nhiều giá trị**

Oprion **chain** được bật sẽ khiến setter method trả về **this** thay vì **void**. Cách này khá giống với Builder pattern, chúng ta có thể gán giá trị cho các thuộc tính liên tiếp trong cùng một mệnh đề.

* 1. **Prefix** option cho phép bỏ qua các chuỗi bắt đầu bằng prefix liệt kê trong mảng string tự định nghĩa.