**# Note:**

- Định nghĩa

+ Phần mở rộng

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

**1.Variable : class, object, local, final,....**

**Variable:**

+ Chúng ta có thể khai báo đa dạng các kiểu biến như number, string, boolean, etc.

+ Trừ trường hợp được khai báo bên trong 1 function (khi đó biến var sẽ có scope là function/locally scoped), biến var sẽ có scope là globally scoped.

+ Đặc biệt, biến var còn có thêm tính chất hoisting: nghĩa là dù khai báo ở đâu thì biến đều sẽ được đem lên đầu scope trước khi

code được thực hiện.

**Object:** Một thực thể có trạng thái và hành vi được gọi là đối tượng. Ví dụ như máy pha cà phê, xe đạp, cái quạt...

- Một đối tượng có ba đặc điểm:

+ Trạng thái: Đại diện cho dữ liệu (giá trị) của một đối tượng.

+ Hành vi: Đại diện cho hành vi (chức năng) của một đối tượng như gửi tiền, rút tiền, ...

+ Danh tính: Danh tính của một đối tượng thường được cài đặt thông qua một ID duy nhất. ID này được ẩn đối với user bên ngoài.

Tuy nhiên nó được sử dụng trong nội bộ máy ảo JVM để định danh từng đối tượng.

**Class:** Lớp (class) là mô tả về các đối tượng sẽ được tạo ra.

+ Mỗi lớp có tên lớp, trong lớp có các thuộc tính như là dữ liệu của lớp và các ứng xử của lớp đó (các hàm) gọi là các phương thức.

**Local:** Biến local (hay còn gọi là biến cục bộ)

- Đây là loại biến được khai báo bên trong một phương thức, các hàm tạo hoặc một khối.

- Chỉ được sử dụng bên trong phương thức hoặc khối tạo ra nó. Sẽ bị hủy khi kết thúc phương thức hoặc khối lệnh.

- Không có giá trị mặc định khi vừa khởi tạo.

**Instance:** Biến instance

- Là các biến được khai báo trong một lớp nhưng không nằm trong bất kì một phương thức, một hàm tạo hay khối lệnh nào cả.

- Các biến khi khởi tạo nếu không gán giá trị thì nó sẽ có giá trị mặc định.Đối với kiểu số thì mặc định là 0,

đối với kiểu boolean mặc định là false, đối với kiểu trả về đối tượng thì sẽ trả về null.

- Các biến này có thể được sử dụng trong bất kì phương thức, hàm tạo,hay khối lệnh nào thuộc lớp chứa nó.

**Final:**Từ khóa final trong Java được sử dụng để hạn chế người dùng.

Từ khóa final có thể được sử dụng trong nhiều ngữ cảnh:

+ Biến final: bạn không thể thay đổi giá trị của biến final (nó sẽ là hằng số).

+ Phương thức final: bạn không thể ghi đè phương thức final.

+ Lớp final: bạn không thể kế thừa lớp final.

+ Biến static final trống: Một biến final mà không được khởi tạo tại thời điểm khai báo được gọi là biến final trống.

**Static:** Từ khóa static trong Java được sử dụng chính để quản lý bộ nhớ.

- Chúng ta có thể áp dụng từ khóa static với các biến, các phương thức, các khối, các lớp lồng nhau(nested class).

- Từ khóa static thuộc về lớp chứ không thuộc về instance(thể hiện) của lớp.

Trong java, Static có thể là:

+ Biến static: Khi bạn khai báo một biến là static, thì biến đó được gọi là biến tĩnh, hay biến static.

+ Phương thức static: Khi bạn khai báo một phương thức là static, thì phương thức đó gọi là phương thức static.

+ Khối static: Được sử dụng để khởi tạo thành viên dữ liệu static.

Ví Dụ: Giả sử có 1000 sinh viên trong trường đại học, bây giờ instance của các dữ liệu thành viên sẽ sự dụng bộ nhớ mỗi khi đối

tượng được tạo. Tất cả sinh viên có rollno và name là thuộc tính riêng. Tuy nhiên, college là thuộc tính chung của tất cả đối tượng.

Nếu chúng ta tạo nó là static, thì trường này sẽ chỉ sử dụng bộ nhớ một lần để lưu biến này.

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

**2. Data types, Access modifier, constructor, supper, this**

**Data types:** Kiểu dữ liệu

+ Chúng ta có thể khai báo đa dạng các kiểu biến như number, string, boolean, etc.

+ Mỗi kiểu có cách sử dụng và bộ nhớ khác nhau

Ví dụ:

+ String : "abc"

+ Boolearn: True or False

+ Int: 123

....

**Access modifier:**

- Các access modifiers trong java xác định độ truy cập (Phạm vi) vào dữ liệu của của các trường, phương thức, cấu tử hoặc class.

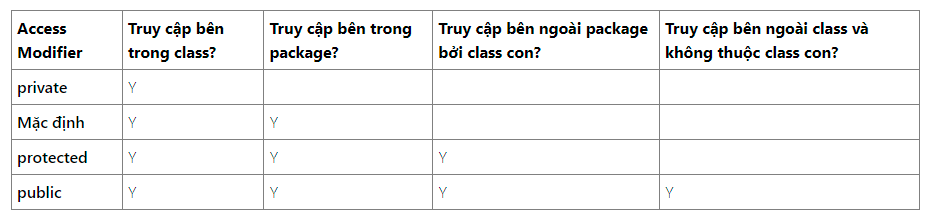
- Có 4 kiểu của java access modifiers:

+ private

+ (Mặc định)

+ protected

+ public



**Constructor:**

- Constructor trong java là một dạng đặc biệt của phương thức được sử dụng để khởi tạo các đối tượng.

- Java Constructor được gọi tại thời điểm tạo đối tượng. Nó khởi tạo các giá trị để cung cấp dữ liệu cho các đối tượng,

đó là lý do tại sao nó được gọi là constructor.

Có 2 quy tắc cơ bản cho việc tạo constructor:

+ Tên constructor phải giống tên lớp chứa nó.

+ Constructor không có kiểu trả về tường minh.

**Supper:** Từ khóa super trong java là một biến tham chiếu được sử dụng để tham chiếu trực tiếp đến đối tượng của lớp cha gần nhất.

- Bất cứ khi nào bạn tạo ra instance(thể hiển) của lớp con, một instance của lớp cha được tạo ra ngầm định,

nghĩa là được tham chiếu bởi biến super.

- Trong java, từ khóa super có 3 cách sử dụng như sau:

+ Từ khóa super được sử dụng để tham chiếu trực tiếp đến biến instance của lớp cha gần nhất.

+ super() được sử dụng để gọi trực tiếp Constructor của lớp cha.

+ Từ khóa super được sử dụng để gọi trực tiếp phương thức của lớp cha.

**This:**

- Trong java, Từ khóa this có 6 cách sử dụng như sau:

+ Từ khóa this có thể được dùng để tham chiếu tới biến instance của lớp hiện tại.

+ this() có thể được dùng để gọi Constructor của lớp hiện tại.

+ Từ khóa this có thể được dùng để gọi phương thức của lớp hiện tại.

+ Từ khóa this có thể được truyền như một tham số trong phương thức.

+ Từ khóa this có thể được truyền như một tham số trong Constructor.

+ Từ khóa this có thể được dùng để trả về instance của lớp hiện tại.

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

**3.Four principle of OOP:** Bốn tính chất trong lập trình hướng đối tượng

**Encapsulation:** Tính đóng gói

- “đóng gói” là việc đưa tất cả thông tin, dữ liệu quan trọng vào bên trong một đối tượng, nhằm che đậy và ngăn chặn các tác

động trực tiếp làm thay đổi trạng thái

**Abstraction:** Tính trừu tượng

- Tính trừu tượng dùng để chỉ quá trình ẩn việc triển khai thực tế của một ứng dụng khỏi người dùng.

Thay vào đó, lập trình viên chỉ nhấn mạnh vào cách sử dụng ứng dụng.

**Inheritance :** Tính kế thừa

- Là kỹ thuật cho phép kế thừa lại những tính năng mà một đối tượng khác đã có, giúp tránh việc code lặp dư thừa mà chỉ xử lý

công việc tương tự.

**Polymorphism :** Tính đa hình

- Thuật ngữ này dùng để ám chỉ đến việc một biến, một hàm hoặc một phương thức có thể tồn tại ở nhiều dạng khác nhau.

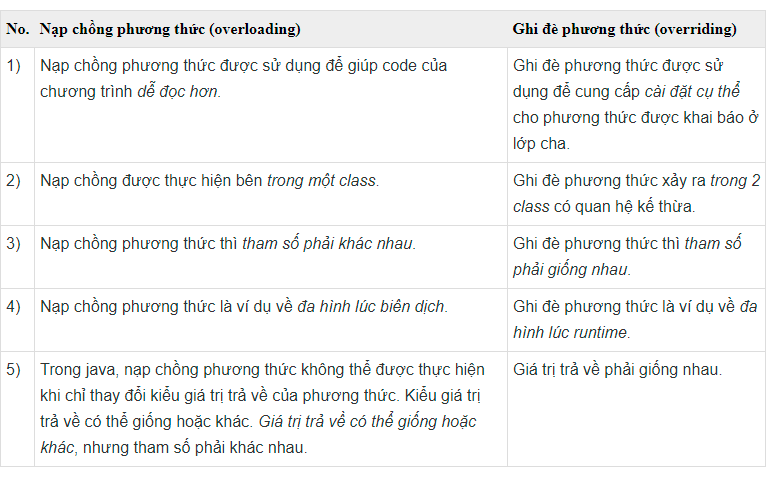
Tức là nhiều hàm hoặc phương thức có thể cùng một tên nhưng chức năng thực sự của chúng lại khác nhau.

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

**4 : Overload & Overrided**

Sự khác nhau giữa overloading và overriding trong java

Sự khác nhau giữa overloading và overriding phương thức trong java được thể hiện trong bảng sau:



**5. Abstract class & Interface**

### **Các cách để đạt được sự trừu tượng hóa**

Có 2 cách để đạt được sự trừu tượng hóa trong java

1. **Sử dụng lớp abstract**
2. **Sử dụng interface**

**5.1. abstract**

## Phương thức trừu tượng trong Java

Một phương thức được khai báo là abstract và không có trình triển khai thì đó là phương thức trừu tượng.

Nếu bạn muốn một lớp chứa một phương thức cụ thể nhưng bạn muốn triển khai thực sự phương thức đó để được quyết định bởi các lớp con, thì bạn có thể khai báo phương thức đó trong lớp cha ở dạng abstract.

Từ khóa abstract được sử dụng để khai báo một phương thức dạng abstract. Một phương thức abstract không có thân phương thức.

Phương thức abstract sẽ không có định nghĩa, được theo sau bởi dấu chấm phảy, không có dấu ngoặc nhọn theo sau:

[?](https://viettuts.vn/java/abstract-class-trong-java)

|  |  |
| --- | --- |
| 1  2 | // Khai bao phuong thuc voi tu khoa abstract va khong co than phuong thuc  **abstract** **void** printStatus(); |

### **Ví dụ về lớp trừu tượng và phương thức trừu tượng trong Java**

**abstract** **class** Bike{

**abstract** **void** run();

}

**class** Honda4 **extends** Bike{

**void** run() {

        System.out.println("running safely..");

    }

**public** **static** **void** main(String args[]) {

        Bike obj = **new** Honda4();

        obj.run();

    }

}

## Kế thừa lớp Abstract trong Java

// lop truu tuong Shape

**abstract** **class** Shape{

**abstract** **void** draw();

}

//Trong tinh huong nay, trinh trien khai duoc cung cap boi ai do,

// vi du: nguoi su dung cuoi cung nao do

**class** Rectangle **extends** Shape{

**void** draw(){

  System.out.println("Ve hinh chu nhat");

  }

}

**class** Circle1 **extends** Shape{

**void** draw(){

   System.out.println("Ve hinh tron");

}

}

//Trong tinh huong nay, phuong thuc duoc goi boi lap trinh vien hoac nguoi dung

**class** TestAbstraction1{

**public** **static** **void** main(String args[]) {

   // Trong tinh huong nay, doi tuong duoc cung cap thong qua phuong thuc,

   // chang han nhu getShape()

   Shape s=**new** Circle1();

   s.draw();

   }

}

**5.2. Interface**

Một **Interface trong Java** là một bản thiết kế của một lớp. **Nó chỉ có các phương thức trừu tượng**. Interface là một kỹ thuật để thu được tính trừu tượng hoàn toàn và đa kế thừa trong Java.

Interface trong Java cũng biễu diễn mối quan hệ IS-A. Nó không thể được khởi tạo giống như lớp trừu tượng.

**Nói cách khác, các trường của Interface là public, static và final theo mặc định và các phương thức là public và abstract.**

Một Interface trong Java là một tập hợp các phương thức trừu tượng (abstract). Một class triển khai một interface, do đó kế thừa các phương thức abstract của interface.

Một **interface không phải là một lớp**. Viết một interface giống như viết một lớp, nhưng chúng có 2 định nghĩa khác nhau. **Một lớp mô tả các thuộc tính và hành vi của một đối tượng. Một interface chứa các hành vi mà một class triển khai.**

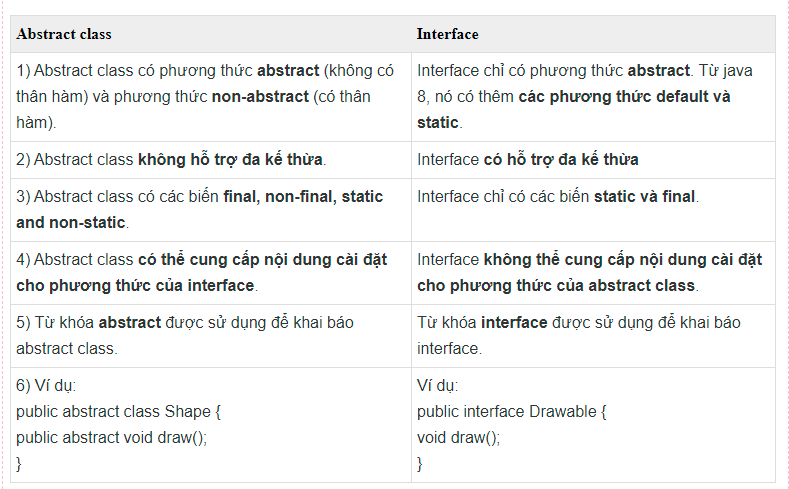
## Sự khác nhau giữa Abstract class và Interface

Một interface tương tự với một class bởi những điểm sau đây:

* Một interface được viết trong một file với định dạng .java, với tên của interface giống tên của file.
* Bytecode của interface được lưu trong file có định dạng .class.
* Khai báo interface trong một package, những file bytecode tương ứng cũng có cấu trúc thư mục có cùng tên package.

Một interface khác với một class ở một số điểm sau đây:

* Bạn không thể khởi tạo một interface.
* Một interface không chứa bất cứ hàm Contructor nào.
* Tất cả các phương thức của interface đều là abstract.
* Một interface không thể chứa một trường nào trừ các trường vừa static và final.
* Một interface không thể kế thừa từ lớp, nó được triển khai bởi một lớp.
* Một interface có thể kế thừa từ nhiều interface khác.



Parcelable

Tài liệu tham khảo

[ 1 ] : https://viettuts.vn/java/tu-khoa-this-trong-java#goto-h2-4