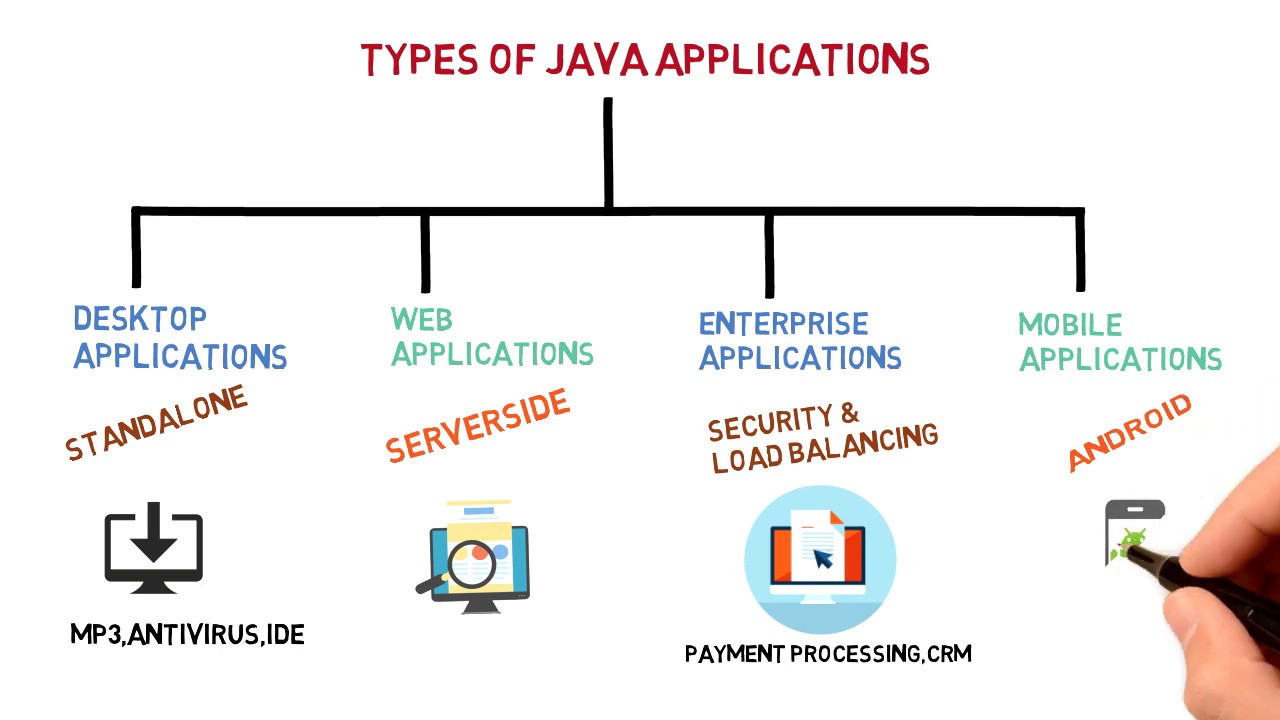
Lí thuyết Java

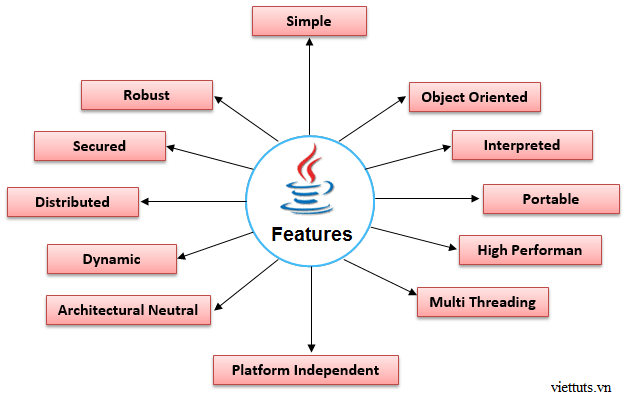
1. Phương châm Java: “Write Once, Run Anywhere”
2. 4 kiểu ứng dụng chính của Java App:



1. 4 Java Platforms:

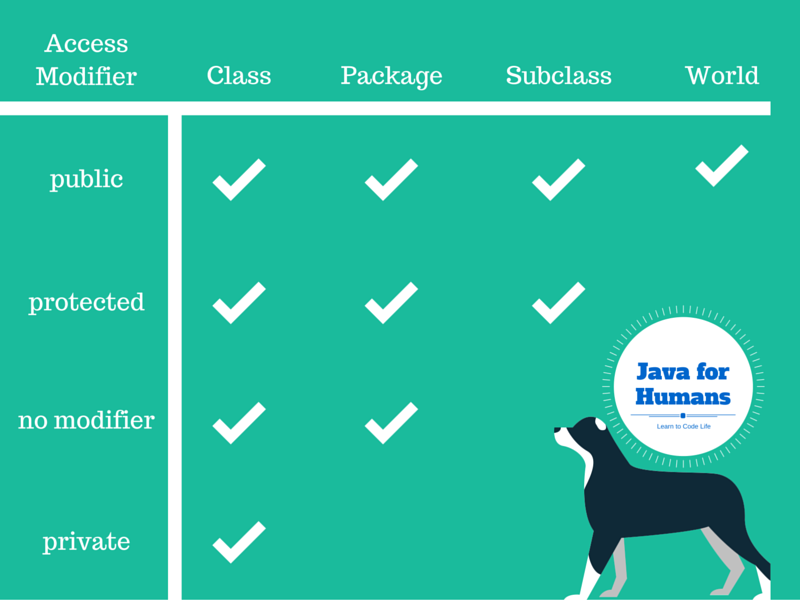
* Java SE (Java Standard Edition): Nền tảng lập trình Java bao gồm các API: java.lang, java.io, java.net, …
* Java EE (Java Enterprise Edition): Phát triển ứng dụng web và doanh nghiệp: Servlet, JSP, Web services, …
* Java ME (Java Micro Edition): Phát triển ứng dụng di động
* JavaFX: phát triển ứng dụng Internet phong phú (Rich Internet Applications - RIAs), tạo giao diện đồ họa dựa trên Swing, Java 2D

1. Java Features:

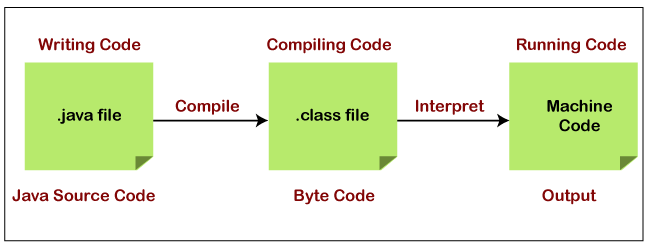


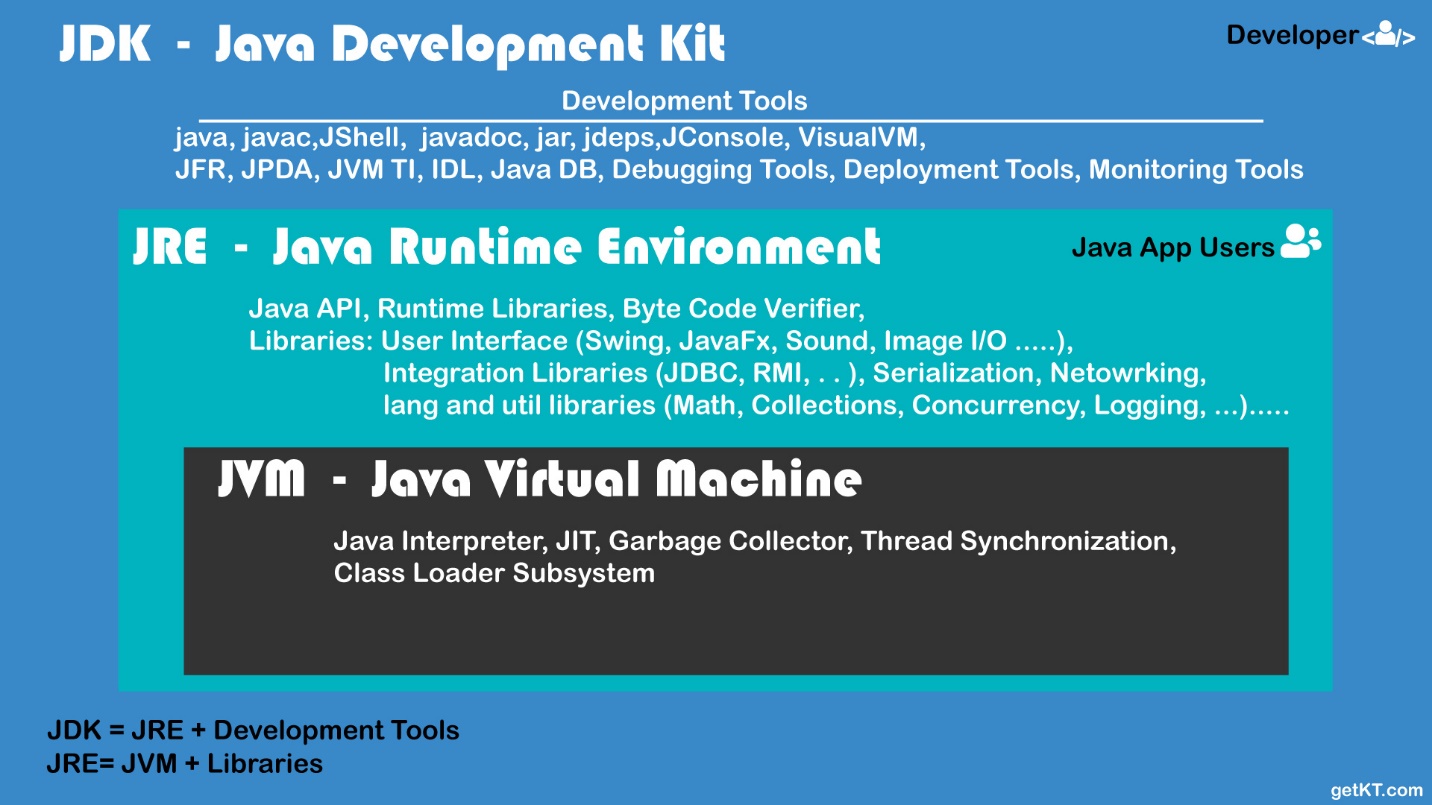
* Portable (Khả chuyển): phần mềm giải nén ra rồi sử dụng, không cần cần cài đặt
* Architechtural Neutral (Kiến trúc tập trung): Chạy được trên phần cứng này thì sẽ chạy được trên phần cứng khác
* Robust (Mạnh mẽ)

1. Access Modifier:

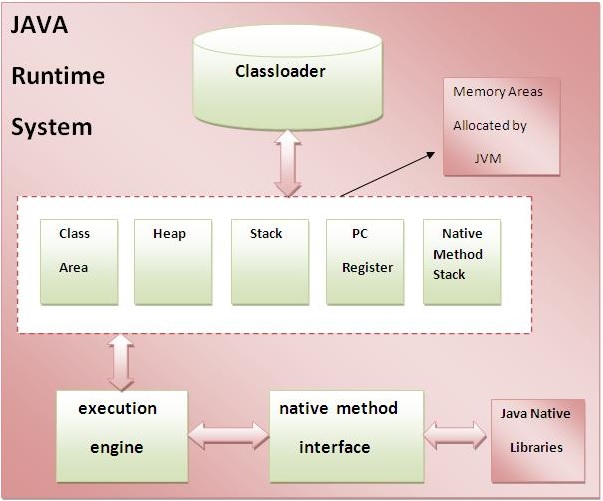


1. JDK, JRE và JVM





1. Cấu trúc của JVM:



* **Classloader**: Là một hệ thống con của JVM được sử dụng để tải class file.
* **Class (method) Area**: Lưu trữ cấu trúc mỗi lớp, chẳng hạn như hằng, trường, dữ liệu phương thức, code của phương thức, ...
* **Heap**: Nó là khu vực dữ liệu runtime mà trong đó đối tượng được cấp phát.
* **Stack:** Nó giữ các biến cục bộ và các kết quả cục bộ, và thực hiện một phần nhiệm vụ trong phần triệu hồi và trả về phương thức

1. 3 loại biến trong java:

**public** **class** Bien {

**public** **static** **float** PI = 3.14f;   // Đây là biến static

**int** n;                            // Đây là biến instance

**public** Bien () {

**char** c = 'c';                 // Đây là biến local

    }

}

1. Phân loại ép kiểu:

* Nới rộng(widening): byte 🡪 short 🡪 int 🡪 long 🡪 float 🡪 double. Đây là dạng ép kiểu không gây mất dữ liệu. VD:
* Thu hẹp(narrowing): double 🡪 float 🡪 long 🡪 int 🡪 short 🡪 byte. Ép kiểu dạng này sẽ gây mất dữ liệu

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

**float** c = 35.8f;

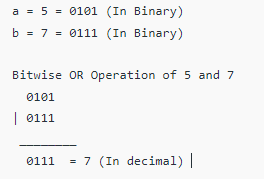
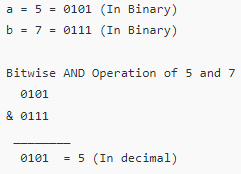
**int** b = (**int**) c;

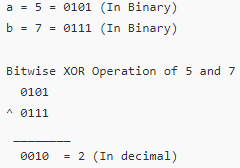
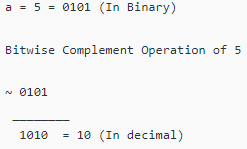
System.***out***.println(b);

// kết quả: 35

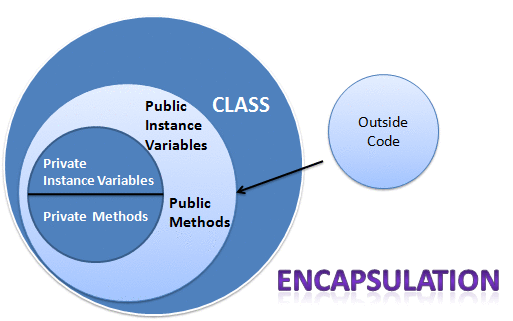
}

1. Các toán tử bit:

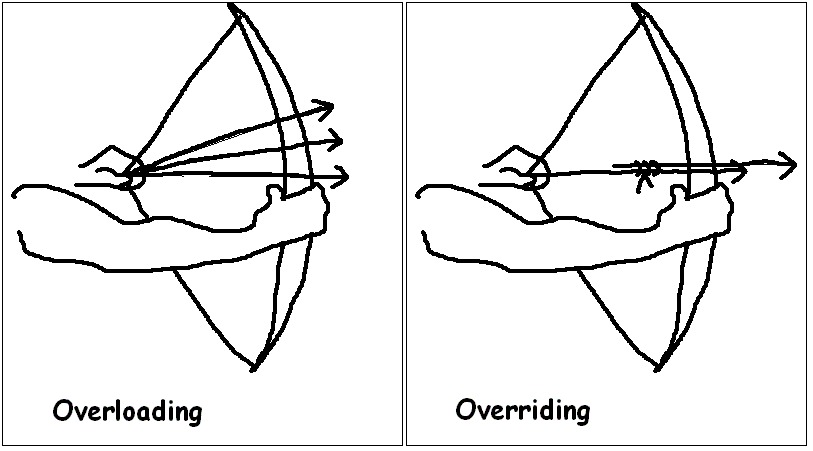
 

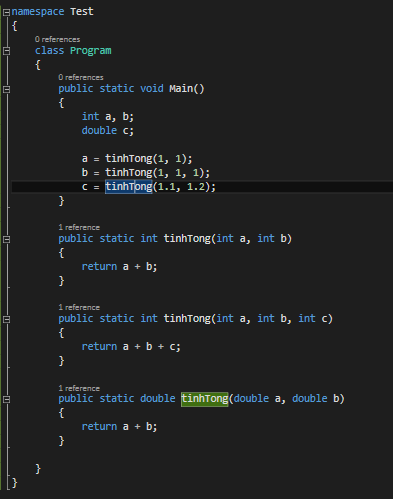
1. Tính đóng gói trong Java: Bảo vệ trạng thái, cấu trúc bên trong một đối tượng khỏi những truy cập không được phép. Giảm tải khớp nối có thể có trong một ứng dụng



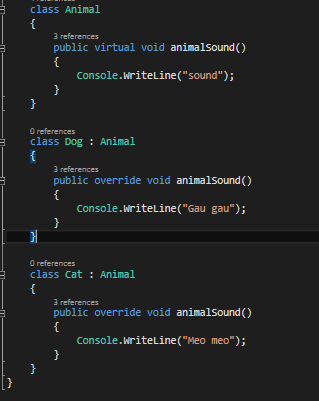
1. Overriding và Overloading:



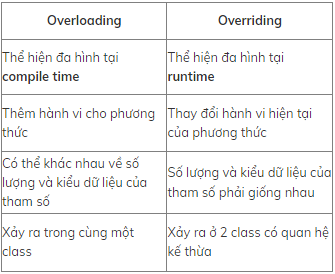
* Overloading:



* Overriding:



* Sự khác nhau:



1. Lớp Wrapper trong Java:

Boolean, Character, Byte, …

1. Toán tử instanceof:

  Dog dog = **new** Dog();

  System.out.println(dog **instanceof** Animal);// true

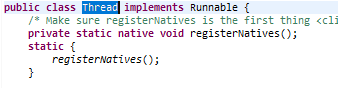
  Dog2 d = **null**;

  System.out.println(d **instanceof** Dog2);// false

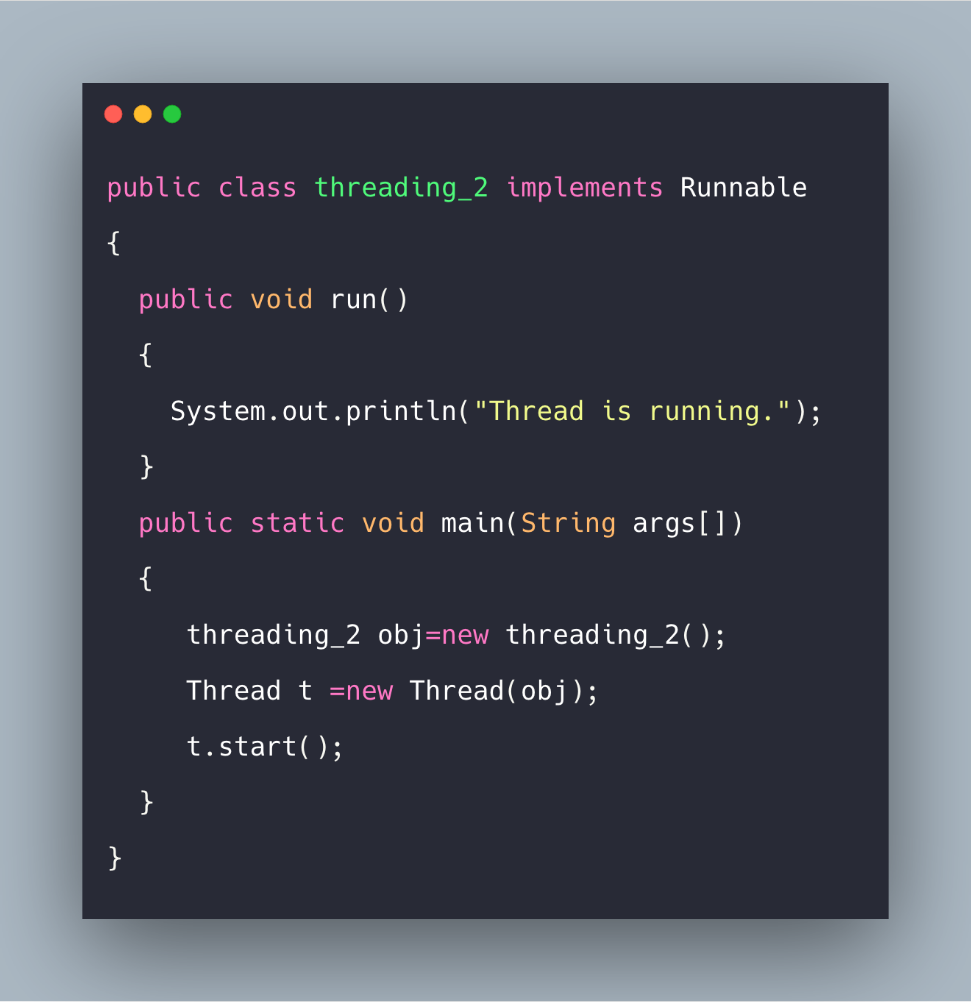
1. Giới hạn của các kiểu dữ liệu:



1. Class Thread implements từ Runnable



Cách dùng Thread:



1. StringBuffer vs StringBuilder

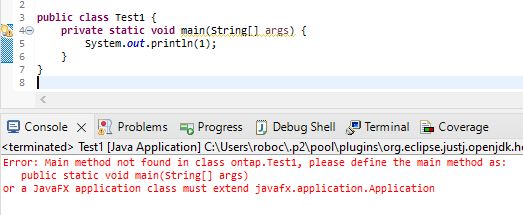
ArrayList vs Vector

Async vs Sync

1. IS-A relationship is wholly related to Inheritance. Quan hệ kế thừa giữa 2 class
2. Has-A relationship is a special form of Association. Composit

Lưu ý trong Java

1. Chỉ có modifier public cho hàm main thì nó mới chạy được nếu không sẽ sinh lỗi khi chạy:



1. Hàm main hợp lệ:

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

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

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

**public** **static** **void** main(String... args)

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

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

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

**final** **strictfp** **public** **static** **void** main(String[] args)

1. Hàm main không hợp lệ:

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

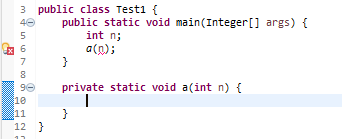
**static** **void** main(String[] args)

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

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

**public** **static void** main(Integer[] args)  // lỗi khi chạy

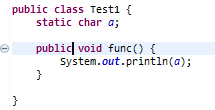
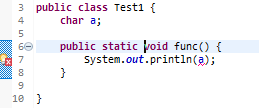
1. Tên biết có thể bắt đầu $: **int $a = 3;**
2. Biến local cần được khai báo trước khi truyền vào làm gì với biến đó:



1. Biến instance không cần phải khai báo trước, default của các kiểu dữ liệu:

* Int, short, byte: 0
* Double: 0.0
* String: null

1. Phương thức static không gọi được biến instance, nhưng phương thức không phải static thì gọi được biến static



1. Float so sánh với float và để ý 2 các viết hợp lệ:

**float** c = 35.8f;

**float** a = 35.8f;

System.***out***.println(c == a == **true**);

// System.out.println((c == a) == true);

// Kết quả true

1. Kết quả phép gán boolean là boolean vì vậy:

**boolean** a = **false**;

**boolean** b;

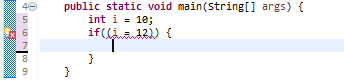
**if**((a = b = **true**)) {

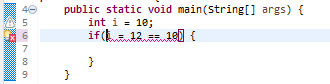
System.***out***.println(a);

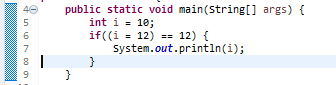
}

* a và b sau khi thực hiện đều là **true**

1. Kết quả của phép gán int là int:







* Kết quả in ra 12

1. Xét 2 TH ép về String:

* Hợp lệ

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

**int** a = 1;

String str = a + "";

}

* Không hợp lệ:



1. Toán tử bitwise:

**int** a = 01001;

**int** b = 1001;

System.***out***.println(a);

// a = 513

System.***out***.println(b);

// b = 1001

1. Kết quả phép dịch chuyển bit:

**int** a = 0001;

**int** left = a << 1;

**int** right = a >> 1;

System.***out***.println(a);

System.***out***.println(left);

System.***out***.println(right);

// a = 1, left = 2, left = 2, right = 0

1. Kết quả chỉ in ra 3:

**int** a = 2;

**if**(a < 3) {

System.out.println(3);

} **else** **if**(a < 5) {

System.out.println(5);

} **else** **if**(a < 7) {

System.out.println(7);

} **else** {

System.out.println(10);

}

1. Kết quả chỉ in ra 3:

**int** a = 2;

**if**(a < 3) {

a = 4;

System.***out***.println(3);

} **else** **if**(a < 5) {

System.***out***.println(5);

} **else** **if**(a < 7) {

System.***out***.println(7);

} **else** {

System.***out***.println(10);

}

1. Kết quả chỉ in ra 7:

**int** a = 2;

**if**((a = 4) < 3) {

System.***out***.println(3);

} **else** **if**((a = 6) < 5) {

System.***out***.println(5);

} **else** **if**(a < 7) {

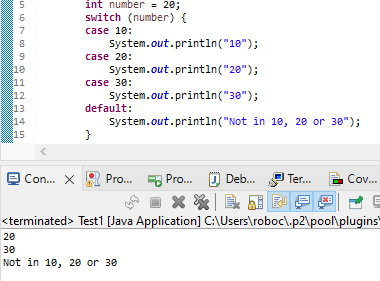
System.***out***.println(7);

} **else** {

System.***out***.println(10);

}

1. Kết quả của switch bắt đầu từ case đúng:



1. Hợp lệ

**long** a = 3L;

**long** b = 3l;

**long** b1 = 3;

**float** c = 3f;

**float** d = 3F;

**float** a = 3L;

**float** t = 3.0f;

**float** t1 = 3;

* Không hợp lệ:

**long** b = 3F;

**float** t = 3.0;

1. Khai báo mảng Java:

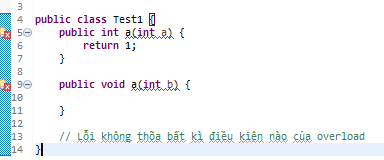
**float**[] a = **new** **float**[3];

**float**[] b = **new** **float**[] {3.0f, 2.0f};

**float** c[] = {3.0f};

// a[0] = a[1] = a[2] = 0.0

1. Overloading thõa điều kiện khác số lượng hoặc tham số và cũng tên phương thức nên:



1. Phương thức trong Interface luôn luôn là **public abstract** và biến là **public static final**
2. Lỗi Switch:

**float** f = **new** ~~Float~~("12");

**switch**(f) {

**case** 1: **break**;

**default**: **break**;

}

// switch không nhận tham số là long float và double, case trong switch chỉ nhận 1 tham số

1. Trong toán tử so sánh không được = =, > =, < = mà là ==, >=, <= và không có toán tử <> hay ><
2. Abstract class có thể viết:

**abstract** **public** **class** Test1 {

**abstract** **boolean** m1();

**abstract** **byte** m2(**short** s);

}

**public** **abstract** **class** Test1 {

**abstract** **boolean** m1();

**abstract** **byte** m2(**short** s);

}

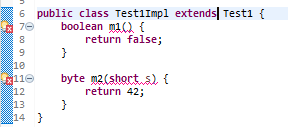
1. Hiện thực phương thức trong class abstract hay interface thì modifier của Class phải giữa nguyên, điều này vẫn đúng khi dùng phương thức **static**

**public** **abstract** **class** Test1 {

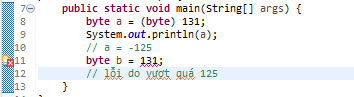
**abstract** **public** **boolean** m1();

**abstract** **public** **byte** m2(**short** s);

}



1. Final class không thể được kế thừa, final function không thể override nhưng có thể overload
2. Ép kiểu byte:



1. Lưu ý trong khi so sánh:

Float f1 = 9.0F;

Float f2 = 9.0F;

**float** f3 = 9;

System.***out***.println(f2.equals(f1)); // true

System.***out***.println(f2 == f1); // false

System.***out***.println(f3 == f1); // true

String str1 = **new** String("H");

String str2 = **new** String("H");

String str3 = "H";

String str4 = "H";

System.***out***.println(str1 == str2); // false

System.***out***.println(str1.equals(str2)); // true

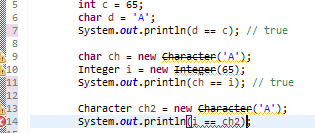
System.***out***.println(str3 == str4); // true

System.***out***.println(str3.equals(str4)); // true

System.***out***.println(str2 == str3); // false

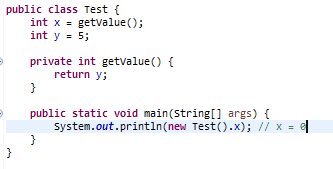
System.***out***.println(str2.equals(str4)); // true

1. char có thể so sánh với int nhưng Character thì không thể với Integer:



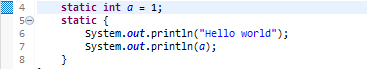
1. Quá trình tạo biến thuộc tính trong class:

* Khởi tạo với giá trị mặc định
* set giá trị sau

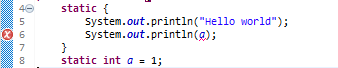


1. Đối với static thì run code từ trường xuống, ta xét 2 trường hợp:

* Không lỗi:

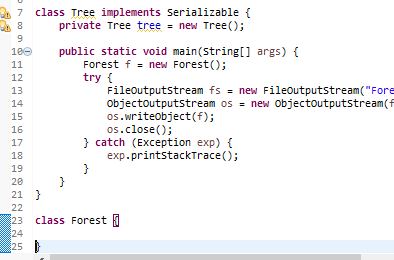


* Lỗi:

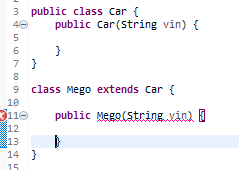


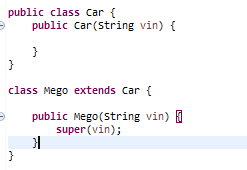
1. Trong một file .java thì **public** **class** trùng với tên file còn class không public thì không cần:

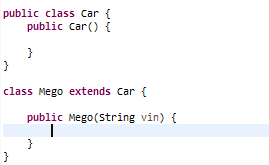
Trong file Forest.java:



1. Thông thường constructor class con nếu không có tham số thì mặc định nó lấy không có đối số của class cha còn khi nếu có tham số thì phải ghi ra, ta xét 3 trường hợp:







1. Tất cả các List đều implements Iterable và chứa phương thức iterator()
2. Không thể sort cho Object có những dữ liệu có kiểu dữ liệu khác nhau:

Object[] myObjects = { 10, "Jaja"};

Arrays.*sort*(myObjects);

1. Xét VD sau:

**int**[] array = {1,2,3,4,5};

List l = Arrays.*asList*(array, array);

System.***out***.println(l.size()); // size = 2

1. Split //d:

String test = "1a2b3c4d5";

String[] tokens = test.split("\\d");

System.***out***.println(Arrays.*toString*(tokens));

// print: [, a, b, c, d]

1. Giá trị key và value null:

Map<String, String> map = **new** HashMap<>();

map.put(**null**, **null**);

map.put(**null**, **null**);

System.***out***.println(map);

// print {null=null}

1. Mảng 3 chiều có thể khai báo như sau:

**int**[] []x[]

1. Ưu tiên khởi tạo biến final trước:

**private** **static** **int** *x* = *getValue*();

**private** **static** **final** **int** ***y*** = 5;

**private** **static** **int** getValue() {

**return** ***y***;

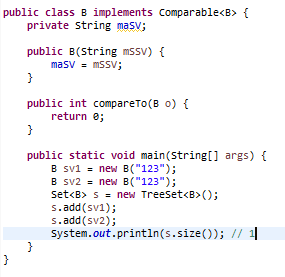
}

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

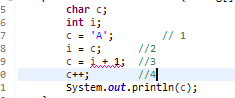
System.***out***.println(*x*);

}

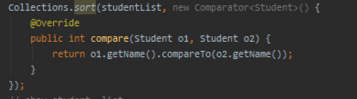
1. Dùng Comparable để so sánh cho Set:

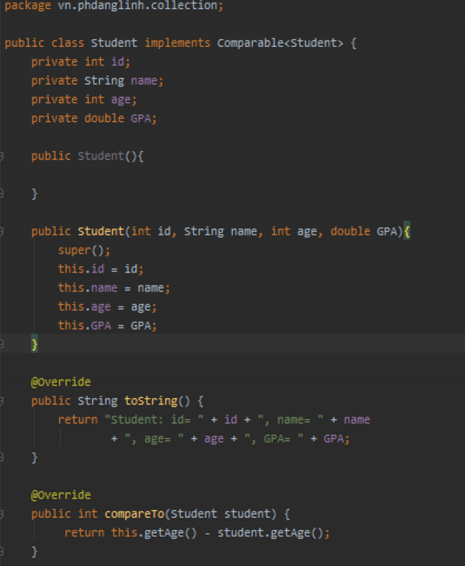


1. Không thể gán char = int + number:

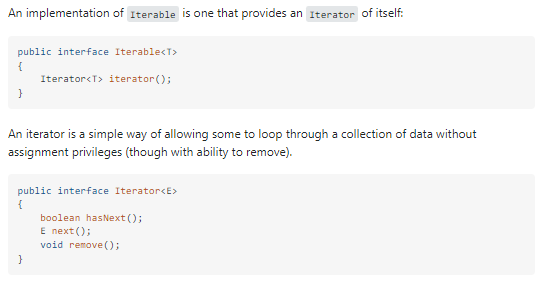


1. Null có thể cộng với chuỗi
2. Comparable vs Comparator





1. Iterable vs Iterator



1. Tính chất của Java:

* Tính đóng gói (encapsulation): Che giấu thông tin (information hiding), ngăn chặn những truy cập không được phép
* Tính kế thừa (Inheritance)
* Tính đa hình (polymorphism)
* Tính trừu tượng (abstraction): ẩn các chi tiết trình triển khai và chỉ hiển thị tính năng tới người dùng

1. Java 8
   1. Lambda Expression:

Java 8 gọi các Interface có duy nhất một method trừu tượng là các **Functional Interface**

@FunctionalInterface

**interface** MyInterface

{

**void** abstract\_func(**int** x,**int** y);

**default** **void** default\_Fun()

{

System.***out***.println("This is default method");

}

}

**class** Main

{

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

{

//lambda expression

MyInterface fobj = (**int** x, **int** y)->System.***out***.println(x+y);

System.***out***.print("The result = ");

fobj.abstract\_func(5,5);

fobj.default\_Fun();

}

}

* 1. Method reference

@FunctionalInterface

**interface** ExecuteFunction {

**public** **int** execute(**int** a, **int** b);

}

**class** MathUtils {

**public** **static** **int** sum(**int** a, **int** b) {

**return** a + b;

    }

**public** **static** **int** minus(**int** a, **int** b) {

**return** a - b;

    }

}

**public** **class** DemoMethodReference1 {

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

**int** a = 10;

**int** b = 7;

**int** sum = doAction(a, b, MathUtils::sum);

        System.out.println(a + " + " + b + " = " + sum); // 10 + 7 = 17

**int** minus = doAction(a, b, MathUtils::minus);

        System.out.println(a + " - " + b + " = " + minus); // 10 - 7 = 3

    }

**public** **static** **int** doAction(**int** a, **int** b, ExecuteFunction func) {

**return** func.execute(a, b);

    }

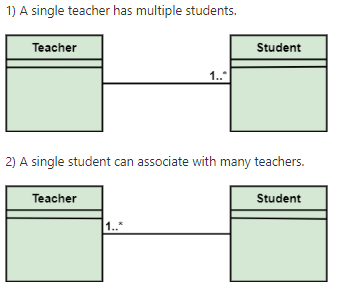
}

1. Từ khóa final

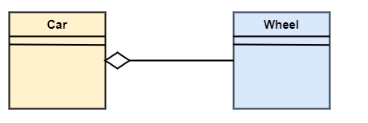
* Biến final: bạn không thể thay đổi giá trị của biến final (nó sẽ là hằng số).
* Phương thức final: bạn không thể ghi đè phương thức final.
* Lớp final: bạn không thể kế thừa lớp final.
* Biến static final trống: Một biến final mà không được khởi tạo tại thời điểm khai báo được gọi là biến final trống.

1. Không thể static cho constructor
2. Liên hệ các class

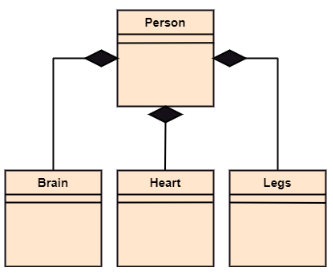
* Association: Có liên hệ nhau



* Aggregation: A car cannot move without a wheel. But the wheel can be independently used with the bike, scooter, cycle, or any other vehicle. The wheel object can exist without the car object, which proves to be an aggregation relationship.



* Composition: If the person is destroyed, the brain, heart, and legs will also get discarded.



Không rõ nguyên nhân

1. Expected: 3, Actual: 9

**int** a = 0011;

System.***out***.println(a);