Khách Inbound và Outbound

User tách riêng 1 DB

Db dùng: Redis hoặc kafka

* Buid trên docker

Mọi thứ đều đi qua Gateway

-header

+ osaptype

+ osapversion

Bản tin trả ra:

+ Http Status

+ Errorcode: “1000” do mk tạo ra

+ message

+ data

Phase1: 2 month: hình thành luồng cơ bản

Sprint 1:

* User: xem tour, đặt tour, thanh toán, rating …
* Agency: đăng ký, đăng tour, quản lý tour – lịch tour/ ….
* Admin: quản trị user/agency

Phase 2: nâng cấp, mobile ….

Quản lý: aglle / trello…

Mỗi ngày nói ra được:

+ done

+ problem

+ plan

# .Primitive & Object datatype

Kiểu dữ liệu là dùng để lưu trữ thông tin, giá trị mà chương trình biên dịch ra

1. Thế nào là kiểu dữ liệu nguyên thủy?

* Kiểu dữ liệu nguyên thủy là kiểu dữ liệu được chương trình cung cấp và hỗ trợ sẵn, nó được dùng để lưu trữ các giá trị đơn giản
* Kiểu dữ liệu nguyên thủy là tham chiếu

Có 2 kiểu chính là kiểu boolen và kiểu số

Kiểu logic: boolean

Kiểu số:

+ Kiểu nguyên:

+ Kiểu số thực: Float, Double

Mặc định Java xem kiểu số nguyên là int và số thực là double, nên khi ta khai báo kiểu dữ liệu float thì them f hoặc F, long thì thêm l hoặc L vào cuối số. Với kiểu float thì chắc chắn phải thêm f (hoặc F) vào cuối số vì nó có phạm vi nhỏ hơn mặc định double. Còn long thì giả sử nó vượt quá phạm vi của int thì them l (hoặc L) ở cuối

Overflow (tràn ra): đó là khi giá trị vượt quá mức cho phép mà chương trình cho ra, giả sử long là kiểu dữ liệu cho phép lớn nhất rồi mà có phép cộng, cộng thêm 1 đơn vị nữa chẳng hạn nó sẽ vượt quá mức cho phép của long thì lúc này Java sẽ đảo ngược chiều về hàng âm nhỏ nhất và bắt đầu đếm lại từ đó

Underflow (tràn xuống): ngược lại với overflow, nghĩa là khi nhỏ quá vượt mức lưu trữ, phạm vi của chương trình thì nó sẽ xoay chiều sang số dương lớn nhất và giảm xuống từ đó

1. Thế nào là kiểu dữ liệu đối tượng

* Kiểu dữ liệu đối tượng là kiểu dữ liệu mang tính chất tham chiếu, biến của nó sẽ tham chiếu đến đối tượng. Biến của nó sẽ không lấy giá trị mà lấy địa chỉ bộ nhớ của đối tượng đó
* Trong java có 3 kiểu dữ liệu đối tượng:

+ Array: đó là 1 mảng động chứa các phần tử có cùng kiểu dữ liệu

+ Class: do người dùng tự định nghĩa

+ interface: dữ liệu kiểu lớp giao tiếp do người dùng tự định nghĩa

1. Chuyển đổi kiểu dữ liệu giữa 2 dạng này như thế nào ?

* Thông qua Wrapper class, cụ thể là autoboxing và unboxing

Java cung cấp 1 Wrapper class bao gồm các đối tượng Integer, Double, Long, Boolean …

nó sẽ tự động chuyển đổi 2 kiểu dữ liệu đó lẫn nhau

Integer a = Integer.valueOf(9); // boxing

Integer a = 9; // autoboxing

int b = a.intValue(); // unboxing

int c = a; // auto unboxing

* Chuyển đổi thủ công

int a = 10;

Integer b = Integer.valueOf(a);

Interger c = new Integer(10);

int d = c.intValue;

Mỗi kiểu điều có ưu nhược điểm riêng

* Kiểu nguyên thủy:

+ Ưu điểm: tốn ít bộ nhớ, hiệu suất tốt hơn, đoạn mã code nhìn rõ ràng

+ Nhược điểm: cứng nhắc, không có tính linh hoạt, không thể tùy biến được

* Kiểu đối tượng:

+Ưu điểm: đa dạng, linh hoạt, dễ tùy biến, tái sử dụng

+ Nhược điểm: tốn bộ nhớ khi ta không sử dụng hợp lý

Autoboxing: là quá trình tự biến đổi kiểu dữ liệu đối tượng sang kiểu dữ liêu tham chiếu

Ví dụ: Integer = 2; // autoboxing

Integer a = new Integer(2)

Integer a = Interger.valueOf(2)

Unboxing: là quá trình biến đổi từ kiểu dữ liệu đối tượng sang nguyên thủy

Integer b = 2;

int c = b.intValue()

# Java-OOP-1

Hướng đối tượng là mô hình hóa hệ thống trong cuộc sống trong phần mềm dựa trên các đối tượng

Các tính chất hướng đối tượng:

* Tính đóng gói: gói gém và che giấu dữ liệu, bảo mật dữ liệu trong đối tượng, và chỉ được truy cập dữ liệu bên trong đối tượng qua các phương thức và không cho phép gọi trực tiếp chúng

Thể hiện qua access modifiers, khi 1 biến hoặc 1 phương thức khai báo dưới dạng:

+ public(công khai): thì chúng ta có thể truy cập chúng ở khắp mọi nơi

+ private(riêng tư): chỉ được gọi trong class mà chúng được định nghĩa

+ defaut: được gọi trong cùng 1 package

+ protected: được truy cập trong cùng 1 package, và các lớp con của nó

public class Animal {  
 private String name;  
  
 public Animal(String name) {  
 this.name = name;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

như trên ta thấy khi khởi tạo đối tượng Animal, khi xét các thuộc tính ta phải gọi thông qua phương thức trung gian đó là setter và getter. 2 phương thức đó sẽ kiểm soát được giá trị mà ta muốn đưa vào xem có hợp lệ không

* Tính kế thừa: khi một lớp con kế thừa 1 lớp cha thì nó sẽ được thừa kế toàn bộ những phương thức và thuộc tính của lớp cha. Nhưng nếu lớp cha đặt trạng thái ở private thì nó sẽ không kế thừa được trừ khi trong cùng 1 class
* Tính đa hình: trên cùng 1 phương thức nhưng mỗi đối tượng khác nhau thì nó là khác nhau, điều này trong java thường thể hiện qua:

+ override(ghi đè): khi 1 lớp con kế thừa 1 lớp cha, mà nó không muốn kế thừa hành động của của lớp cha thì nó sẽ ghi đè nội dung lên theo ý của nó

+ overrloading(nạp chồng): là ta có thể định nghĩa nhiều phương thức có cùng tên nhưng khác kiểu dữ liệu, kiểu trả về và số lượng đối số truyền vào

public class Overload {  
  
 public int tinhTong(int a, int b) {  
 return a+b;  
 }  
 public double tinhTong(double a, double b){  
 return a+b;  
 }  
   
 public double tinhTong(double a, double b, double c){  
 return a+b+c;  
 }  
  
 public static void main(String[] args) {  
 Overload overload = new Overload();  
  
 System.*out*.println(overload.tinhTong(3,2,1));  
 }  
}

+ Sử dụng đối tượng đa hình (Polymorphic Object): biến của lớp cha có thể tham chiếu đến đối tượng của lớp con

* Tính trừu tượng: cho phép chúng ta tạo ra 1 lớp mà không cần cung cấp cách triển khai trong phương thức, nó giúp tạo ra 1 cái tiêu chuẩn, 1 mô tả mà không cần cung cấp nội dung, để khi các lớp con kế thừa nó thì sẽ phải triển khai tuân theo nguyên tắc của mô tả đó và triển khai phương thức bên trong

Phân biệt giữa lớp và đối tượng trong Java

* Lớp là 1 bản thiết kế hoặc như 1 bản mô tả đến đối tượng, nó cung cấp các thuộc tính và hành vi để định nghĩa lên đối tượng
* Đối tượng là 1 thực thể cụ thể của 1 lớp, có hành vi và được tạo ra dựa trên thiết kế của lớp
* Như vậy ta có thể hiểu đối tượng được tạo ra dựa trên class, đối tượng là 1 thực thể còn class là 1 logic được viết ra, trong java khi runtime thì nó sẽ không cấp vùng nhớ cho class mà cấp cho đối tượng được tạo ra từ class

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | Trong class | Trong package | Ngoài package không thuộc class con | Ngoài package thuộc class con | World |
| private | Y | X | X | X | X |
| default | Y | Y | X | X | X |
| protected | Y | Y | X | Y | X |
| public | Y | Y | Y | Y | Y |

# 2. Static & final

Thế nào là static trong Java?

* Khi 1 static được khai báo nghĩa là nó thuộc của lớp đó chứ không thuộc về đối tượng cụ thể nào của lớp đó. Các đối tượng sẽ dùng chung những gì khai báo static

Ý nghĩa từ khóa static trong Java?

* Static khi khai báo biến thì biến này sẽ được coi là của lớp, mỗi đối tượng sẽ không phải khởi tạo nó và dùng luôn nó, các đối tượng sẽ chia sẻ giá trị của biến được gán static này

Ưu điểm:

+ chia sẻ giá trị cho nhau giữa các đối tượng

+ tiết kiệm bộ nhớ

+ kết hợp với từ khóa final để khai báo hằng

+ theo dõi được đối tượng tạo ra

* Static trong phương thức thì chỉ có class tạo ra nó mới được sử dụng, nó không phụ thuộc vào đối tượng được tạo ra hay không và các đối tượng cũng không được dùng nó. Nó không phụ thuộc vào tham số truyền vào của đối tượng

Ưu điểm

+ Thực hiện các hành động không liên quan đến đối tượng, nhưng vẫn liên quan đến lớp

+ Tạo các tiện ích chung như là: tính tổng, max, min, trung bình ….

+ Xây dựng phương thức main, vì phương thức main được chạy ngay từ đầu khi bắt đầu câu lệnh, thì với static nó không cần khởi tạo đối tượng nên JVM không cần phải khởi tạo đối tượng cho class chứa nó

+ Tạo ra các factory method, đây là các phương thức tạo ra các đối tượng mới dựa trên tham số truyền vào

* Chúng ta có thể ghi đè 1 hàm là static hoặc private hay không?

Không thể ghi đè static hoặc private

+ static: vì chúng thuộc của lớp không phải của đối tượng. Bình thường khi kế thừa, lớp con thừa hưởng của lớp cha và có thể ghi đè, nó dành cho đối tượng

+ private: do phạm vi của nó chỉ được truy cập khi trong cùng 1 class nên không thể ghi đè được

Có thể truy cập biến non-static trong 1 lớp static không

* Truy cập trực tiếp thì không, còn gián tiếp thì có. Dùng biến tham chiếu của đối tượng, bằng cách ta phải khởi tạo đối tượng của lớp đó và dùng biến tham chiếu đó gọi ra

Thế nào là final trong Java? Có thể kế thừa được phương thức final không?

* Final trong java có những loại sau:

+ Biến final khi được khai báo sẽ có giá trị không đổi

+ Phương thức final: khi lớp con kế thừa phương thức này thì nó chỉ được kế thừa chứ không được ghi đè chúng

+ class final: thì không thể kế thừa

+ biến static final trống: được gọi là biến static final trống, sẽ được khai báo sau trong block static {}

* Có thể kế thừa phương thức final nhưng không thể ghi đè chúng vì khi 1 phương thức được khai báo là final nó sẽ không đổi, nên lớp con kế thừa nó chỉ được phép dùng chứ không được khi đè

Có thể khi báo biến final không khởi tạo giá trị ban đầu tại thời điểm khai báo không?

* Có thể không khai báo, nhưng phải thông qua hàm constructor hoặc với static final thì ta có thể khai báo sau qua khối block static

Thế nào là tham số final?

* Khi 1 tham số final được khai báo trong 1 phương thức hay 1 constructor, thì nó sẽ không được gán giá trị trong phương thức hay constructor đó. Mà nó chỉ nhận giá trị khi 1 đối tượng gọi đến hàm đó và truyền đối số vào
* Khi một final khai báo với 1 collection thì có thể hiểu là biến tham chiếu đến collection đó không được thay đổi, size và giá trị của collection vẫn thay đổi được nhưng biến tham chiếu đến nó không được cho giá trị khác

# 3. This & Super

Thế nào là **this** và **super** trong java? 2 từ khóa trên được sử dụng trong các trường hợp nào?

* this nó tham chiếu đến lớp hiện tại như các biến istance, hàm, constructor, trả về instance của lớp hiện tại, được truyền như 1 tham số trong phương thức và constructor

ví dụ: + this tham chiếu tới biến instance trong lớp hiện tại

public class Test {  
  
 private String name;  
 private int age;  
  
 public Test(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
  
 public void run(){  
 System.*out*.println("Name: " + this.name);  
 System.*out*.println("Age: " + this.age);  
 }  
  
 public static void main(String[] args) {  
 Test test = new Test("nam", 18);  
 test.run();  
 }  
  
}

**+** gọi đến phương thức đến lớp hiện tại

public class Test {  
  
 private String name;  
 private int age;  
  
 public Test(String name, int age) {  
 this.name = name;  
 this.age = age;  
 }  
  
 public void printWebsite(){  
 System.*out*.println("Welcome ");  
 this.run();  
 }  
  
 public void run(){  
 System.*out*.println("Name: " + this.name);  
 System.*out*.println("Age: " + this.age);  
 }  
  
 public static void main(String[] args) {  
 Test test = new Test("nam", 18);  
 test.printWebsite();  
 }  
  
}

+ gọi đến constructor của lớp hiện tại. Khi bạn có nhiều constructor và bạn muốn sử dụng lại

public class Test {  
 private int id;  
 private String name;  
 private int age;  
  
 public Test() {  
 System.*out*.println("Ok");  
 this.id = 1;  
 this.name = "Nam";  
 }  
  
 public Test(String name) {  
 this();  
 this.name = name;  
 }  
  
 public Test(String name, int age){  
 this(name);  
 this.age = age;  
 }  
  
 public void printWebsite(){  
 System.*out*.println("Welcome ");  
 this.run();  
 }  
  
 public void run(){  
 System.*out*.println("Id: " + this.id);  
 System.*out*.println("Name: " + this.name);  
 System.*out*.println("Age: " + this.age);  
 }  
  
 public static void main(String[] args) {  
 Test test = new Test("nam", 18);  
 test.printWebsite();  
 }  
  
}

+ trả về instance cho lớp hiện tại

public class Test {  
 private int id;  
 private String name;  
 private int age;  
   
 public Test(String name, int age){  
 this.name = name;  
 this.age = age;  
 }  
  
 public void printWebsite(){  
 System.*out*.println("Welcome ");  
 this.run();  
 }  
  
 public void run(){  
 System.*out*.println("Id: " + this.id);  
 System.*out*.println("Name: " + this.name);  
 System.*out*.println("Age: " + this.age);  
 }  
  
 public static void main(String[] args) {  
 Test test = new Test("nam", 18);  
 test.printWebsite();  
 }  
  
 public Test setNameTest(String name) {  
 this.name = name;  
 return this;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
}

+ cũng có thể dùng this truyền vào như 1 tham số

public class Test {  
 private int id;  
 private String name;  
 private int age;  
  
 public Test(String name, int age){  
 this.name = name;  
 this.age = age;  
 }  
  
 public void printWebsite(){  
 System.*out*.println("Welcome ");  
 this.run();  
 }  
  
 public void run(){  
 TestCon testCon = new TestCon();  
 testCon.getAll(this);  
 }  
  
 public static void main(String[] args) {  
 Test test = new Test("nam", 18);  
 test.run();  
 }  
  
 public Test setNameTest(String name) {  
 this.name = name;  
 return this;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public int getAge() {  
 return age;  
 }  
}  
  
class TestCon {  
  
 public void getAll(Test test){  
 System.*out*.println("Name: " + test.getName());  
 System.*out*.println("Age: " + test.getAge());  
 }  
}

+ Được truyền như 1 tham số trong constructor

public class Test {  
 private int id;  
 private String name;  
 private int age;  
  
 public Test(String name, int age){  
 this.name = name;  
 this.age = age;  
 }  
  
 public void printWebsite(){  
 TestCon testCon = new TestCon(this);  
 testCon.print();  
 }  
  
 public static void main(String[] args) {  
 Test test = new Test("nam", 18);  
 test.printWebsite();  
 }  
  
 public Test setNameTest(String name) {  
 this.name = name;  
 return this;  
 }  
  
 public void setName(String name) {  
 this.name = name;  
 }  
  
 public int getId() {  
 return id;  
 }  
  
 public String getName() {  
 return name;  
 }  
  
 public int getAge() {  
 return age;  
 }  
}  
  
class TestCon {  
 private Test test;  
 public TestCon(Test test) {  
 this.test = test;  
 }  
  
 public void print(){  
 System.*out*.println("Name: " + test.getName());  
 System.*out*.println("Age: " + test.getAge());  
 }  
}

Nói tóm lại This sẽ có những mục đích như sau:

this thực ra nó là 1 biến tham chiếu và nó được sử dụng để tham chiếu đến đối tượng hiện tại của lớp đó

+ tham chiếu đến biến instance của lớp hợp tại ( this.age = age )

+ gọi đến phương thức hoặc constructor của lớp hiện tại ( this.run() hoặc this(), this(name) )

+ có thể trả về 1 instance (return this;)

+ được truyền vào như 1 tham số trong phương thức hoặc construtor ( testCon.getAll(this), Testcon testcon = new Testcon(this) )

* super: là 1 biến tham chiếu nó được sử dụng để tham chiếu đến đối tượng cha gần nhất của lớp con
* Bất cứ khi nào 1 instance lớp con tạo ra sẽ có 1 intance của lớp cha được tạo ra ngầm định, nó được gọi là biến super.
* Nó sẽ gọi constructor, thuộc tính và phương thức của lớp cha gần nhất

+ goi đến constructor của lớp cha gần nhất của nó:

class Parents {  
 private String name;  
  
 public Parents() {  
 System.*out*.println("This is parents");  
 print();  
 }  
 public Parents(String name){  
 this.name = name;  
 System.*out*.println("This is parents");  
 print();  
 }  
 public void print(){  
 System.*out*.println("Hello world: " + name);  
 }  
}  
  
class Child extends Parents {  
 public Child(){  
 System.*out*.println("This is Child");  
 }  
 public Child(String name){  
 super(name);  
 System.*out*.println("This is Child");  
 }  
  
 public static void main(String[] args) {  
 Child child1 = new Child();  
 System.*out*.println("---");  
 Child child2 = new Child("Hai");  
 }  
}

+ gọi đến thuộc tính (field) cha gần nhất

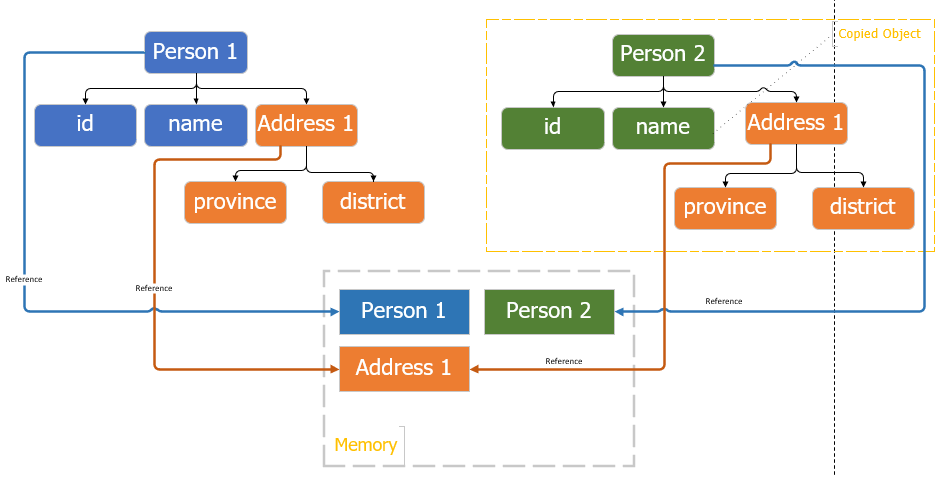
Lưu ý: nếu như thuộc tính của lớp con khác tên thuộc tính của lớp cha thì không cần dùng từ super

+ gọi đến phương thức của lớp cha gần nhất. Giải quyết trường hợp ghi đè, và trong trường hợp bạn vẫn gọi hàm từ lớp cha ta sẽ dùng super

**Phân biệt Reference Copy (copy kiểu tham chiếu) và Object Copy (copy toàn bộ đối tượng ) ? Nêu ví dụ cụ thể**

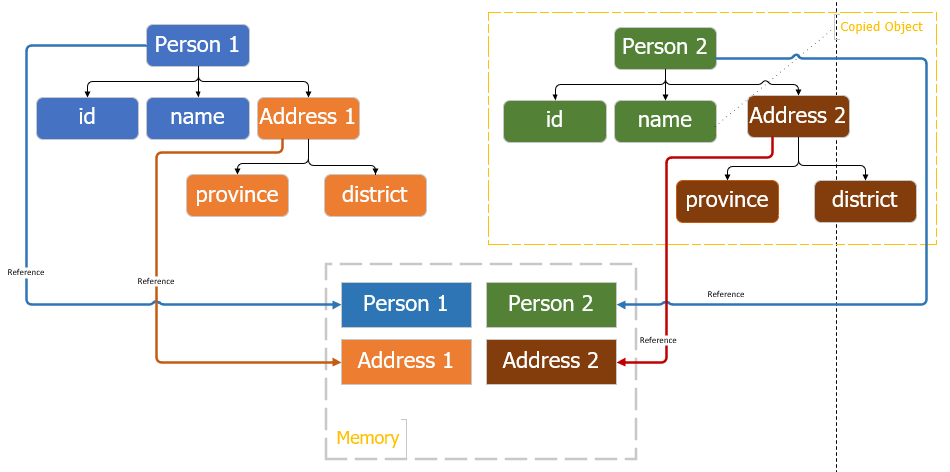
Trong Java có 2 kiểu copy đó là copy kiểu tham chiếu và copy đối tượng

* Reference Copy (copy tham chiếu): khi chúng ta thay đổi thuộc tính bên trong 1 đối tượng thì đối tượng copy đối tượng đố qua toán tử ‘=’ thì đối tượng copy đó cũng sẽ thay đổi theo. Vì bản chất đây là copy địa chỉ chứ không phải giá trị, 2 đối tượng này sẽ cùng có 1 địa chỉ
* public class Student {  
   int id;  
   String name;  
    
   public Student() {  
   }  
    
   public Student(int id, String name) {  
   this.id = id;  
   this.name = name;  
   }  
    
   public void show(){  
   System.*out*.println("Id: " + id);  
   System.*out*.println("Name: " + name);  
   }  
  }  
    
  class Test {  
   public static void main(String[] args) {  
   Student s1 = new Student(1, "Hai");  
   Student s2 = s1;  
   s1.show();  
   s2.show();  
    
   System.*out*.println("After changed: ");  
   s1.id = 4;  
   s1.name = "Nam";  
   s1.show();  
   s2.show();  
    
   }  
  }
* Object copy: khi đối tượng 1 copy đối tượng 2 thì khi object 2 thay đổi giá trị thuộc tính thì đối tượng 1 sẽ không ảnh hưởng, thực ra là 2 đối tượng cùng tạo trên 1 class
* Shallow copy/ Shallow cloning: đối tượng copy sẽ copy những thuộc tính kiểu dữ liệu nguyên thủy (int, float, …) và String, nhưng các thuộc tính thuộc dạng Object vẫn tham chiếu tới địa chỉ cũ. Nghĩa là khi thuộc tính Object thuộc đối tượng gốc thay đổi thì nhưng đối tượng copy nó sẽ thay đổi theo



class Address {  
 String name;  
  
 public Address() {  
 }  
  
 public Address(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public String toString() {  
 return "Address{" +  
 "name='" + name + '\'' +  
 '}';  
 }  
}  
public class Person implements Cloneable{  
 int id;  
 String name;  
 Address address;  
  
 public Person(int id, String name, Address address) {  
 this.id = id;  
 this.name = name;  
 this.address = address;  
 }  
  
 @Override  
 protected Person clone() throws CloneNotSupportedException {  
 return (Person) super.clone();  
 }  
  
 @Override  
 public String toString() {  
 return "Person{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", address=" + address +  
 '}';  
 }  
}  
  
class TestCopy {  
 public static void main(String[] args) throws CloneNotSupportedException{  
 Address address = new Address("Hải Phòng");  
 Person person = new Person(1, "Hai", address);  
 Person personCopy = person.clone();  
 System.*out*.println(person);  
 System.*out*.println(personCopy);  
  
 System.*out*.println("----------------");  
  
 person.id = 2;  
 person.name = "Duy";  
 person.address.name = "Hà nội";  
 System.*out*.println(person);  
 System.*out*.println(personCopy);  
 }  
}

* Deep cloning



class Address implements Cloneable{  
 String name;  
  
 public Address() {  
 }  
  
 public Address(String name) {  
 this.name = name;  
 }  
  
 @Override  
 public String toString() {  
 return "Address{" +  
 "name='" + name + '\'' +  
 '}';  
 }  
  
 @Override  
 protected Address clone() throws CloneNotSupportedException {  
 return(Address) super.clone();  
 }  
}  
public class Person implements Cloneable{  
 int id;  
 String name;  
 Address address;  
  
 public Person(int id, String name, Address address) {  
 this.id = id;  
 this.name = name;  
 this.address = address;  
 }  
  
 @Override  
 protected Person clone() throws CloneNotSupportedException {  
 Person person = (Person) super.clone();  
 person.address = this.address.clone();  
 return person;  
 }  
  
 @Override  
 public String toString() {  
 return "Person{" +  
 "id=" + id +  
 ", name='" + name + '\'' +  
 ", address=" + address +  
 '}';  
 }  
}  
  
class TestCopy {  
 public static void main(String[] args) throws CloneNotSupportedException{  
 Address address = new Address("Hải Phòng");  
 Person person = new Person(1, "Hai", address);  
 Person personCopy = person.clone();  
 System.*out*.println(person);  
 System.*out*.println(personCopy);  
  
 System.*out*.println("----------------");  
  
 person.id = 2;  
 person.name = "Duy";  
 person.address.name = "Hà nội";  
 System.*out*.println(person);  
 System.*out*.println(personCopy);  
 }  
}

**Bạn hiểu gì về từ khóa instance of trong Java**

Biến là 1 vị trí trong bộ nhớ dùng để lưu trữ dữ liệu

Integer a = 1;

+ Instance dùng để kiểm tra 1 đối tượng có thể hiện của 1 kiểu dữ liệu cụ thể hay không

+ Trả về True khi nó đúng

+ Với biến có giá trị null thì nó trả về false