**Java 44. Lớp và phương thức trừu tượng Abstract trong Java**

- Một phương thức trừu tượng bắt buộc phải ở trong class trừu tượng

- Một phương thức trừu tượng không có phần thân

- Chúng ta không thể khởi tạo một biến có kiểu là class abstract nhưng chúng ta có thể khởi tạo một mảng có kiểu class abstract và chúng ta truyền những giá trị có kiểu là class kế thừa từ class abstract đó

- Dưới đây là một ví dụ:

+ Ta có class abstract có tên là Animal, class Canine cũng là class abstract kế thừa từ class Animal, class Dog là class concrete

**Animal**

- String picture

- String food

- String location

**abstract** **public** **void** makeNoise();

**abstract** **public** **void** eat();

**public** **void** sleep() {...}

**abstract** **public** **void** roam();

extends

**abstract** **public** **class** Canine **extends** Animal{

//impletement lại phương thức trừu tượng makeNoise

@Override

**public** **void** roam() {

System.***out***.print("\nDi lang thang trong nha, vuon, duong di");

}

}

extends

**public** **class** MyAnimalList **extends** Animal{

**public** MyAnimalList() {

}

**private** Animal[] animals = **new** Animal[500];

**private** **int** nextIndex = 0;

**public** **void** add(Animal a, **int** n) {

animals[n] = a;

}

**public** **void** xuat(**int** n) {

**for**(**int** i = 0 ; i<n ; i++) {

System.***out***.print("\nThong tin animal "+(i+1));

System.***out***.print("\n"+animals[i].picture+", "+animals[i].food+", "+animals[i].location);

}

}

@Override

**public** **void** makeNoise() {

// **TODO** Auto-generated method stub

}

@Override

**public** **void** eat() {

System.***out***.print("An xuong");

}

**import** java.util.Scanner;

**public** **class** MainAnimal {

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

Scanner ip = **new** Scanner(System.***in***);

String hinhdang, thucan, noio;

**int** n;

System.***out***.print("\nNhap vao so phan tu cua mang animal: ");

n = ip.nextInt();

ip.nextLine();

//Khoi tao bien m de goi ham add

MyAnimalList m = **new** MyAnimalList();

**for**(**int** i=0 ; i<n ; i++) {

System.***out***.print("\nNhap vao animal thu "+(i+1));

System.***out***.print("\nNhap vao hinh dang: ");

hinhdang = ip.nextLine();

System.***out***.print("Nhap vao thuc an: ");

thucan = ip.nextLine();

System.***out***.print("Nhap vao noi o: ");

noio = ip.nextLine();

Dog d = **new** Dog();

d.setPicture(hinhdang);

d.setFood(thucan);

d.setLocation(noio);

//Goi ham them vao thong tin

m.add(d, i);

}

//Goi ham xuat ra thong tin

m.xuat(n);

}

}

**Java 46. Hiểu rõ về INTERFACE trong lập trình Java**

**<<Khái niệm và đặc điểm của INTERFACE:>>**

- interface là một bản thiết kế của một class

- Một interface trong Java không thể chứa constructor (hàm khởi tạo). Mục tiêu chính của một interface là định nghĩa các phương thức mà các lớp khác sẽ implement (thực thi). Interface không thể chứa mã cài đặt cho các phương thức, cũng như không thể chứa bất kỳ hàm khởi tạo nào.

- Một interface có thể kế thừa từ các interface khác.

- Khi một lớp implement một interface, nó phải cung cấp mã cài đặt cho tất cả các phương thức trong interface đó.

- Tuy nhiên, khi một lớp implement một interface, lớp đó vẫn có thể có các hàm khởi tạo của riêng mình, không liên quan gì đến việc implement interface. Những hàm khởi tạo này là những phương thức để khởi tạo đối tượng của lớp đó, không phải là phần của interface.

- interface chỉ chứa các phương thức trừu tượng và các hằng số

- Tất cả các phương thức là ở dạng public

**<<Mục đích, ưu điểm của interface trong Java>**

Dưới đây là một số so sánh về ưu điểm vượt trội của interface trong lập trình hướng đối tượng Java:

|  |  |  |
| --- | --- | --- |
|  | **Interface** | **Extends** |
| 1. Đa kế thừa (Multiple Inheritance) | Cho phép một class triển khai nhiều interface. Điều này cho phép đa kế thừa, có nghĩa là một class có thể có các tính chất từ nhiều nguồn khác nhau thông qua các interface. | Trong Java, một class chỉ có thể kế thừa từ một class cha duy nhất. Nó không hỗ trợ đa kế thừa của các class khác. |
| 2. Mô hình tương tự | Được sử dụng để xác định các hành vi mà các class triển khai phải tuân theo, không cung cấp cài đặt. Nó cung cấp một cách để định nghĩa các phương thức mà các class triển khai cần có. | Khi một class kế thừa từ một class cha, nó kế thừa tất cả các thuộc tính và phương thức của class cha đó. |
| 3. Flexibility(linh hoạt) và Composition(thành phần) | Cho phép sử dụng tính linh hoạt hơn trong việc thiết kế. Các class có thể triển khai nhiều interface, từ đó cung cấp sự phong phú hơn trong việc xác định hành vi và tính năng của một đối tượng. | Khi sử dụng kế thừa, đôi khi có thể dẫn đến mối quan hệ chặt chẽ giữa các class. Composition (sự sắp xếp thành phần) thông qua việc sử dụng interface có thể cho phép một cách tiếp cận linh hoạt hơn trong việc xây dựng các class và tái sử dụng code. |

**<<Vậy nên sử dụng interface hay enxtends>>**

- Nếu muốn định nghĩa hành vi mà các class có thể thực hiện với cùng tên gọi thì nên sử dụng interface.

- Nếu muốn chia sẻ code từ class cha sang class con thì việc sử dụng extends sẽ phù hợp và tiết kiệm được thời gian hơn.

**Sau đây là một số ví dụ để giúp chúng ta hiểu rõ hơn về interface:**

**1. Cách thức để khởi tạo một interface**

Khởi tạo một Project có tên là Java\_46 ==> chuột phải vào

**Java 47. Hiểu rõ về Gói - Package trong lập trình Java**

- Package trong java có thể được phân loại theo hai hình thức, package được dựng sẵn và package do người dùng định nghĩa.

- Có rất nhiều package được dựng sẵn như java, lang, AWT, javax, swing, net, io, util, sql, ...

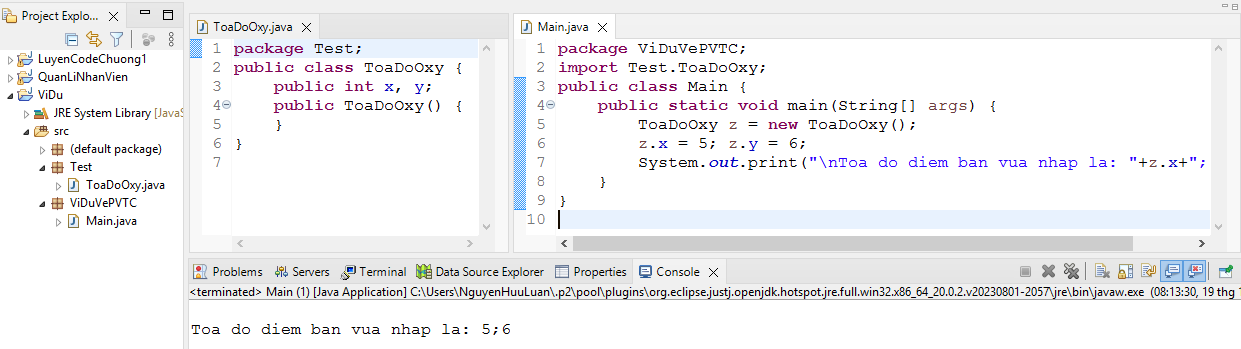
- Một gói có thể có nhiều gói con

- Không được có hai thành viên trong gói trùng tên với nhau

- Tên của gói được viết bằng chữ thường, không được bắt đầu bằng dấu ‘-’hoặc số, không được bắt đầu bằng java hoặc javax(vì đây là tên gói mặt định trong java đã có sẵn). Tên của class có thể trùng với tên của gói chứa nó nhưng khuyến khích là không nên đặt như vậy.

**Ví dụ:**

- Tạo một package có tên là Test(chứa class có tên là ToaDoOxy) và một package khác có tên là ViDuVePVTC( chứa lớp có tên là Main).



Khai báo gói.tên class

**Java 48. Phân biệt điều khiển public, protected, private**

Chúng ta có thể truy cập đến các thuộc tính và phương thức của lớp thông qua đối tượng đã được tạo thông qua 2 cú pháp sau:

Cách truy cập đến thuộc tính: [Tên đối tượng].[Tên thuộc tính];

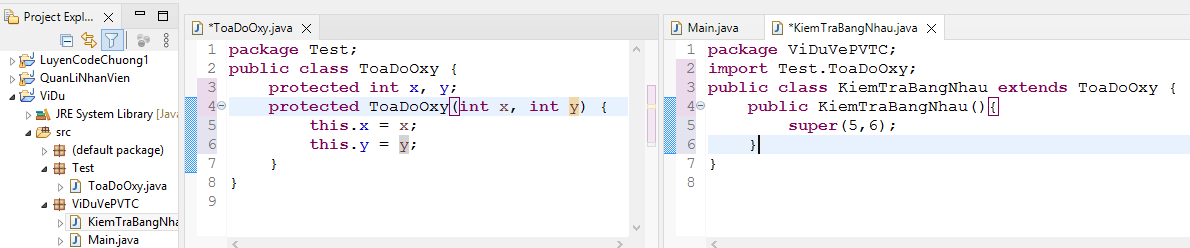
Ví dụ: khởi tạo một đối tượng và gọi tên thuộc tính từ lớp

Cách truy cập đến phương thức của lớp: [Tên đối tượng].[Tên phương thức()];

Như tôi đã trình bày trong bài trước, Trong Java, có 4 phạm vi truy cập sau: **public**, **private**, **protected** và **default** (mặc định).

- **public**: có thể truy cập ở mọi nơi trong project.

- **protected**: truy cập được từ trong lớp khai báo, lớp con(khác gói như trong ví dụ minh họa) của lớp khai báo và các lớp cùng gói với lớp khai báo.



- **default**: truy cập được từ trong lớp khai báo và các lớp cùng gói với lớp khai báo. Lớp con phải nằm trong cùng package với lớp cha để có thể kế thừa. Bất kỳ thành phần có kiểu truy cập default trong lớp cha sẽ không thể truy cập từ lớp con ở package khác.

- **private**: chỉ có thể truy cập bên trong lớp.

**Java\_bổ sung: Hiểu rõ về ArrayList trong lập trình Java**

- Nền tảng Java, phiên bản tiêu chuẩn (SE) cung cấp hàng trăm class được xây dựng sẵn. Trong thư viện Java.util có ArrayList cho phép thêm, sửa, xóa,... với một mảng có kiểu dữ liệu bất kì.

- Dưới đây là một số chức năng cơ bản của ArrayList mà ai cũng phải biết:

**1.Thêm phần tử vào ArrayList:**

a) Thêm vào, lấy ra một phần tử vào ArrayList:

Ví dụ: Tạo một Class gồm các thuộc tính sau:

**package** ArrayList\_trongJava;

**public** **class** CongNhan {

**private** String HoTen, GioiTinh, DiaChi;

**private** **int** Tuoi;

**public** CongNhan(String hoTen, String gioiTinh, String diaChi, **int** tuoi) {

HoTen = hoTen;

GioiTinh = gioiTinh;

DiaChi = diaChi;

Tuoi = tuoi;

}

**public** String getHoTen() {

**return** HoTen;

}

**public** **void** setHoTen(String hoTen) {

HoTen = hoTen;

}

**public** String getGioiTinh() {

**return** GioiTinh;

**public** **void** setGioiTinh(String gioiTinh) {

GioiTinh = gioiTinh;

}

**public** String getDiaChi() {

**return** DiaChi;

}

**public** **void** setDiaChi(String diaChi) {

DiaChi = diaChi;

}

**public** **int** getTuoi() {

**return** Tuoi;

}

**public** **void** setTuoi(**int** tuoi) {

Tuoi = tuoi;

}

}

}

Cú pháp khởi tạo một ArrayList có tên là DSCongNhan có kiểu dữ liệu là CongNhan

ArrayList<CongNhan> DSCongNhan = **new** ArrayList<CongNhan>();

Hàm thêm một phàn tử vào ArrayList DSCongNhan

**public** **void** ThemMotPhanTu() {

CongNhan cn = **new** CongNhan("Nguyen Huu Luan","Nam","Tra Vinh",19);

DSCongNhan.add(cn);

}

Hàm lấy ra một phần tử từ ArrayList DSCongNhan

**public** **void** LayMotPhanTu() {

System.***out***.print("\nThong tin cong nhan 1: ");

System.***out***.print("\nHo ten: "+DSCongNhan.get(0).getHoTen());

System.***out***.print("\nGioi tinh: "+DSCongNhan.get(0).getGioiTinh());

System.***out***.print("\nDia chi: "+DSCongNhan.get(0).getDiaChi());

System.***out***.print("\nTuoi: "+DSCongNhan.get(0).getTuoi());

}

Hàm thêm n phần tử nhập từ bàn phím:

**public** **void** ThemNPhanTu(**int** n) {

String Name, Sex, Location;

**int** Age;

**for**(**int** i=0 ; i<n ; i++) {

System.***out***.print("\nNhap vao thong tin cong nhan thu"+(i+1));

System.***out***.print("\nNhap ho ten: ");

Name = ip.nextLine();

System.***out***.print("Nhap gioi tinh: ");

Sex = ip.nextLine();

System.***out***.print("Nhap dia chi: ");

Location = ip.nextLine();

System.***out***.print("Nhap tuoi: ");

Age = ip.nextInt();

CongNhan cb = **new** CongNhan(Name,Sex,Location,Age);

DSCongNhan.add(cb);

ip.nextLine();

}

}

Hàm xuất n phần tử

**public** **void** LayNPhanTu() {

**int** i=0;

**for**(CongNhan cb : DSCongNhan) {

System.***out***.print("\nThong tin cong nhan thu : "+(i+1));

System.***out***.print("\nHo ten: "+cb.getHoTen());

System.***out***.print("\nGioi tinh: "+cb.getGioiTinh());

System.***out***.print("\nDia chi: "+cb.getDiaChi());

System.***out***.print("\nTuoi: "+cb.getTuoi());

i++;

}

}

Hàm xóa n phần tử dựa theo thuộc tính

**public** **void** XoaTheoTen() {

System.***out***.print("\nNhap ten can xoa: ");

String TenCanXoa = ip.nextLine();

**for**(**int** i=0 ; i< DSCongNhan.size() ; i++) {

//Lay ra phan tu thu i gan vao bien cb

CongNhan cb = DSCongNhan.get(i);

**if**(cb.getHoTen().equalsIgnoreCase(TenCanXoa)) {

DSCongNhan.remove(i);

}

**else** {

System.***out***.print("\nKhong co ten ban can xoa!");

**return**;

}

}

}

Hàm xoa toan bo ArrayList

**public** **void** XoaToanBoArrayList() {

System.***out***.print("\nBan co chac chan muon xoa DSCongNhan ?");

System.***out***.print("\nAn Y de xoa An N de thoat");

System.***out***.print("\nY bang phim 1; N bang phim 0: ");

**int** key = ip.nextInt();

**if**(key==1) {

DSCongNhan.removeAll(DSCongNhan);

System.***out***.print("Xoa thanh cong DSCongNhan");

}

**else** {

**return**;

}

}

Hàm tìm kiếm thông tin theo thuộc tính(trong ví dụ là tên):

**public** **void** TimKiemTheoTen() {

**boolean** Check = **false**;

System.***out***.print("\nNhap ten cong nhan can tim thong tin: ");

System.***out***.print("\nNhap ho ten: ");

String Name = ip.nextLine();

**for**(CongNhan cb : DSCongNhan) {

**if**(cb.getHoTen().equalsIgnoreCase(Name)) {

Check = **true**;

System.***out***.print("\nHo ten: "+cb.getHoTen());

System.***out***.print("\nGioi tinh: "+cb.getGioiTinh());

System.***out***.print("\nDia chi: "+cb.getDiaChi());

System.***out***.print("\nTuoi: "+cb.getTuoi());

}

}

**if**(Check == **false**) {

System.***out***.print("\nKhong ton tai thong tin cua "+Name);

}

}

**Java\_bổ sung: Tìm hiểu các phương thức so sánh chuỗi trong lớp String của thư viện java.lang**

- Trong lớp String của thư viện java.lang có các phương thức so sánh giữa hai chuỗi với nhau.Thư viện chuẩn này tự động import vào các mã nguồn nên ta không cần khai báo

**1. equals()**

Hàm concrete này có kiểu boolean, trả về đúng khi hai chuỗi i hệt nhau(có phân biệt chữ hoa, chữ thường, kể cả space), ngược lại trả về false

Ví dụ:

**package** Test;

**public** **class** ViDuPTString {

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

String s1="A",s2="A ";

System.***out***.print(s1.equals(s2)); //So sánh hai chuỗi có giống nhau hay không

//Có phân biệt chữ hoa chữ thường

}

}

Kết quả trả về: false

**2.compareTo()**

Lưu ý làm hàm này nó hoàn toàn khác với hàm compareTo() trong interface comparable. Hàm này trả về kiểu số nguyên và có 3 kiểu giá trị trả về.<0; >0;=0

Và nó sẽ so sánh từng kí tự trong chuỗi, nếu hai chữ cái đầu tiên như nhau thì nó sẽ so sánh chữ cái tiếp theo. Kết quả sẽ dựa theo kí tự của mã ASCII trong lập trình:

Nếu trả về giá trị <0 thì ta biết là s1 đứng trước s2

Nếu trả về giá trị =0 thì ta biết s1 và s2 tương đương

Nếu trả về giá trị >0 thì ta biết s1 đứng sau s2

Ví dụ:

String s1= "A1", s2="A2";

**int** result = s1.compareTo(s2);

**if** (result < 0) {

System.***out***.println("s1 đứng trước s2 trong từ điển");

}**else** **if** (result > 0) {

System.***out***.println("s1 đứng sau s2 trong từ điển");

} **else** {

System.***out***.println("s1 và s2 giống nhau");

}

Kết quả trả về là s1 đứng trước s2 trong từ điển

**3. equalsIgnoreCase():** tương tự như equals nhưng nó không phân biệt chữ hoa với chữ thường.

**4. compareToIgnoreCase():** tương tự như compareTo nhưng nó không phân biệt chữ hoa với chữ thường

**5. Toán tử ==:** dùng để kiểm tra xem 2 chuỗi có chiếu đến cùng ô nhớ hay không