Trường Cao đẳng Công nghệ Thông tin TP.HCM

Khoa Công nghệ Thông tin – Điện tử

Chương 7:

Các Lớp Trừu Tượng (Abstract) và Interfaces

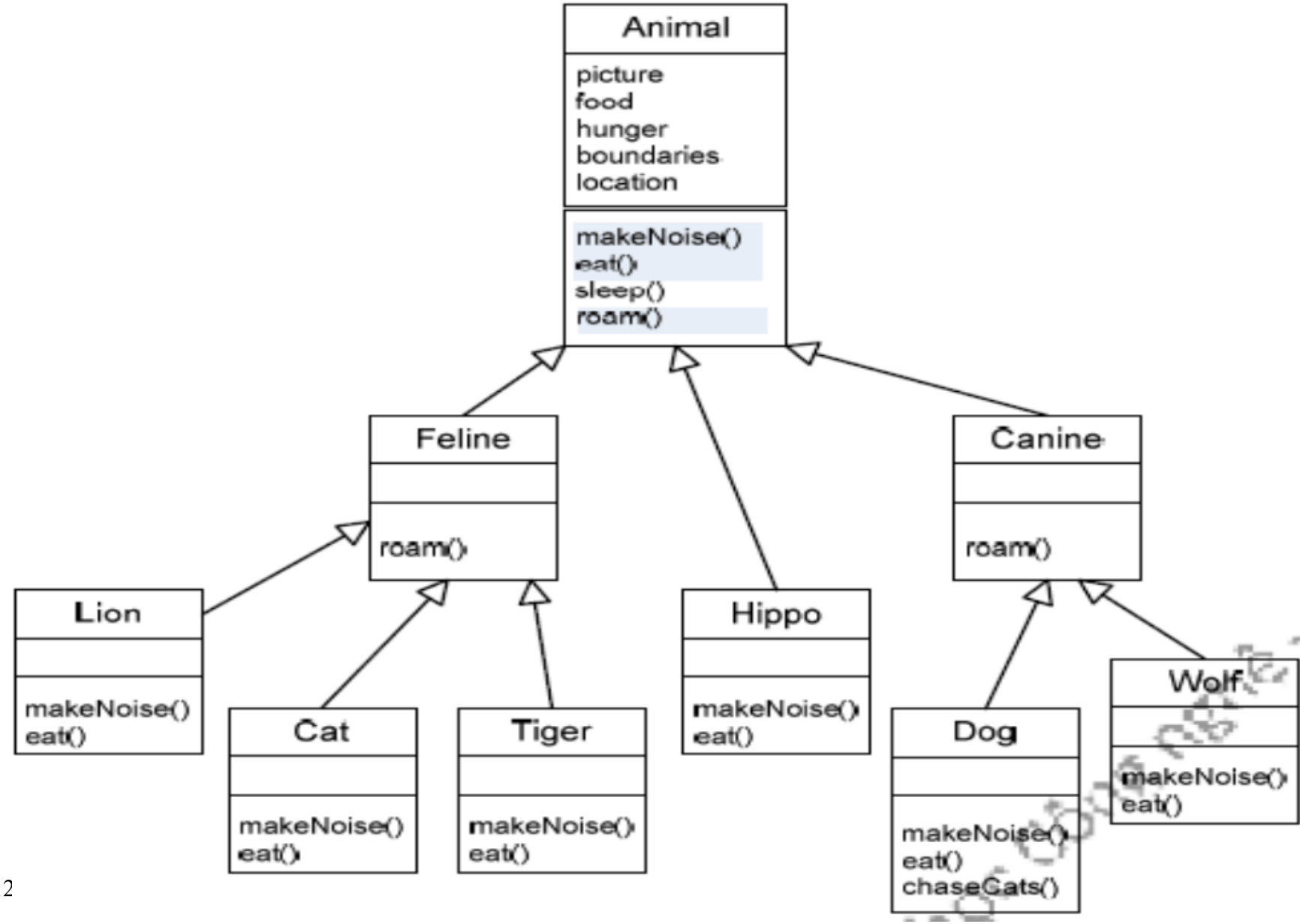
Giảng viên: Hà Mỹ Trinh

Email: trinhhm@itc.edu.vn

Nội dung

1. Một số lớp không nên tạo thực thể
2. Lớp trừu tượng và lớp cụ thể
3. Phương thức trừu tượng
4. Phương thức finalize
5. Giao diện (interface)

1. Một số lớp không nên tạo thực thể



1. Một số lớp không nên tạo thực thể

- VD
- Một thư viện đồ họa:
  - cho phép vẽ (draw), xóa (erase), di chuyển (move) các hình đồ họa.
  - có các lớp Circle (hình tròn), Rectangle (hình chữ nhật)… và để có thể tận dụng quan hệ thừa kế và khi cần có thể xử lý đồng loạt các thành phần của một bản vẽ chẳng hạn, thư viện có thêm lớp tổng quát Shape (hình) là lớp cha chung của các hình đồ họa đó.
- → Liệu có khi nào ta cần tạo một đối tượng thuộc lớp Shape?
- → Nó có hình dạng như thế nào?
- → Làm thế nào để vẽ/xóa nó?
- → Viết nội dung gì cho các phương thức draw và erase của lớp Shape?
  - Chẳng lẽ để trống ?
  - + Hoặc thông báo gì đó?
- → Lỡ có ai tạo một đối tượng Shape rồi gọi phương thức mà đáng ra nó không nên làm gì ?

1. Một số lớp không nên tạo thực thể

- Một lớp cha không bao giờ được dùng để tạo đối tượng được gọi là lớp cơ sở trừu tượng, hay ngắn gọn là lớp trừu tượng (abstract class). Với những lớp thuộc diện này, trình biên dịch sẽ báo lỗi bất cứ đoạn mã nào định tạo thực thể của lớp đó.
- Khi ta thiết kế cấu trúc thừa kế, ta cần quyết định lớp nào trừu tượng, lớp nào cụ thể. Các lớp cụ thể (concrete) là các lớp đủ đặc trưng để có thể tạo thực thể. Trong phạm vi lập trình, một lớp cụ thể có nghĩa đơn giản là: ta được phép tạo đối tượng thuộc loại đó.

1. Một số lớp không nên tạo thực thể

- Các lớp ta vẫn thấy trong các slide bài giảng này đều là các lớp được khai báo là lớp cụ thể. Để quy định một lớp là trừu tượng, ta đặt từ khóa abstract vào đầu khai báo lớp.
- Cú pháp:

abstract class TênClass{

...

}

Ví dụ:

- abstract class Animal{

…

}

- Kết quả là trình biên dịch sẽ không cho phép ta tạo thực thể của lớp đó nữa.

1. Một số lớp không nên tạo thực thể

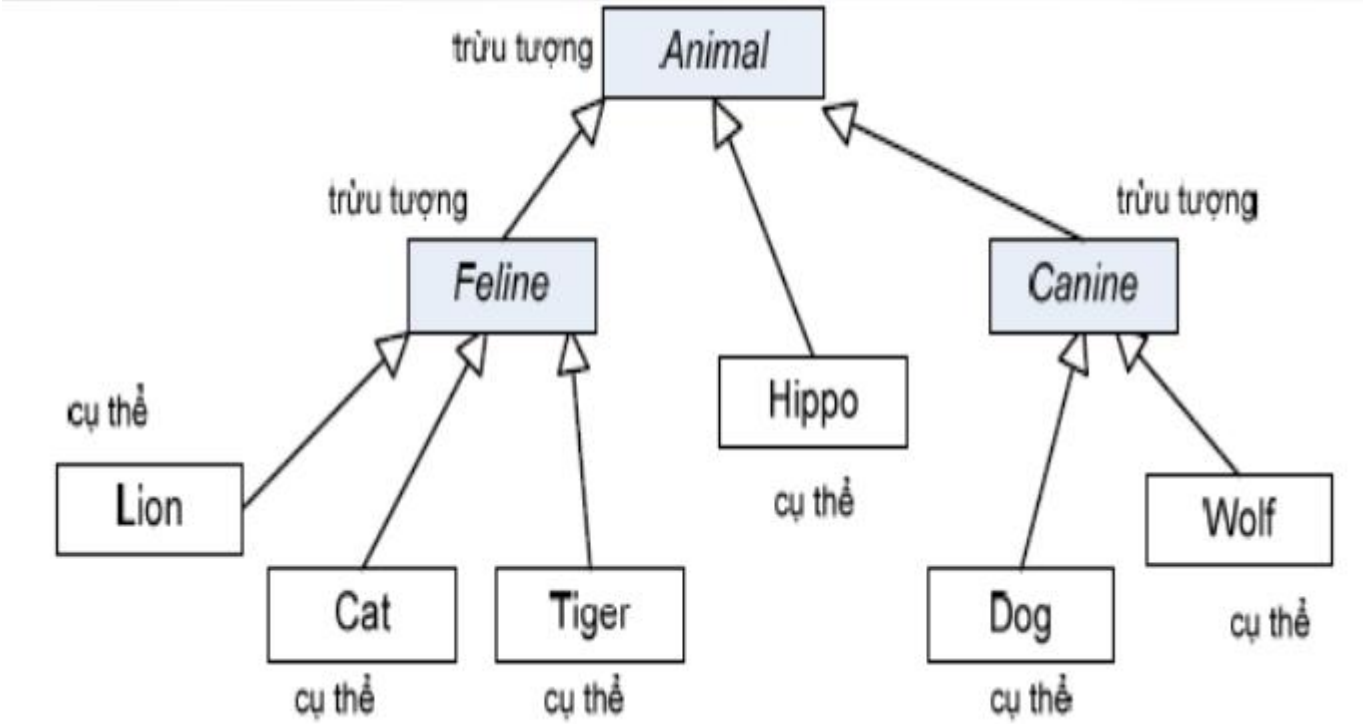
```
public class CanineTestDrive {
  public static void main(String [] args) {
    Canine c;
    c = new Dog();
    c = new Canine();
    c.roam();
  }
}
```

ok, có thể dùng tham chiếu kiểu trừu tượng

trình biên dịch sẽ báo lỗi,
lớp Canine trừu tượng nên không thể
tạo đối tượng Canine

2. Lớp trừu tượng và lớp cụ thể

- Một lớp không phải là lớp trừu tượng thì nó là lớp cụ thể
- Trong cây phả hệ Animal, nếu ta cho Animal, Feline, và Canine là các lớp trừu tượng, thì còn lại sẽ là các lớp cụ thể.



2. Lớp trừu tượng và lớp cụ thể

- Có rất nhiều lớp trừu tượng, đặc biệt trong thư viện giao diện đồ họa GUI
- Vậy khi nào một lớp nên là lớp trừu tượng, khi nào thì nên là lớp cụ thể? Bút chắc là lớp trừu tượng. Bút bi và Bút máy có lẽ cũng nên là các lớp trừu tượng. Vậy đến khi nào thì các lớp trở thành lớp cụ thể? Bút máy Parker liệu có thành lớp cụ thể hay vẫn là lớp trừu tượng? Có vẻ như Bút máy Hồng Hà nét hoa 2008 chắc chắn là lớp cụ thể. Nhưng làm thế nào để chắc chắn?
- Lớp trừu tượng là một dạng lớp đặc biệt, trong đó có phương thức chỉ được khai báo ở dạng khuôn mẫu (template) mà không được cài đặt chi tiết. Việc cài đặt chi tiết các phương thức chỉ được thực hiện ở các lớp con kế thừa lớp trừu tượng đó.
- Lớp trừu tượng được sử dụng khi muốn định nghĩa một lớp mà không thể biết và định nghĩa ngay được vài phương thức của nó.
- Lớp trừu tượng chỉ thiết lập cơ sở cho các lớp kế thừa => bên trong nó sẽ không có bất kỳ cài đặt nào khác

2. Lớp trừu tượng và lớp cụ thể

- Cú pháp khai báo lớp trừu tượng:

[public] abstract class TênLớp

- Lớp trừu tượng là công cụ để ta định nghĩa các hành vi và các thuộc tính mà một nhóm lớp nào đó bắt buộc phải có. Bởi vì các lớp con của nó bắt buộc phải hiện thực các phương thức trừu tượng của nó. Ví dụ: Lớp máy nghe nhạc bắt buộc phải có các hành vi chung đó là: Play, Stop, Pause, Resume, Forward, Backward.
- Lưu ý: Lớp trừu tượng cũng có thể kế thừa một lớp khác, nhưng lớp cha cũng phải là một lớp trừu tượng. (Khai báo kế thừa thông qua từ khoá extends như khai báo kế thừa thông thường).

3. Phương thức trừu tượng

- Khai báo phương thức của lớp trừu tượng

[public] abstract Kiểu_dữ_liệu_trả_về Tên_phương_thức([Các tham số]) [throws <các ngoại lệ>];

- Cú pháp Java quy định rằng phương thức trừu tượng không có thân phương thức.
- Dòng khai báo phương thức kết thúc tại dấu chấm phẩy và không có cặp ngoặc { }.

```
public abstract void makeNoise();
```

- Nếu ta khai báo một phương thức là abstract, ta phải đánh dấu lớp đó cũng là

abstract. Ta không thể đặt một phương thức trừu tượng ở bên trong một lớp cụ thể.

- Tuy nhiên, ta có thể có phương thức không trừu tượng bên trong một lớp trừu tượng.

3. Phương thức trừu tượng

- Tính chất: tính chất của một phương thức trừu tượng của lớp trừu tượng luôn là public. Nếu không khai báo tường minh thì giá trị mặc định cũng là public.
- Kiểu dữ liệu trả về: có thể là các kiểu cơ bản của Java, cũng có thể là kiểu do người dùng tự định nghĩa (kiểu đối tượng).
- Tên phương thức: tuân thủ theo quy tắc đặt tên phương thức của lớp
- Các tham số: nếu có thì mỗi tham số được xác định bằng một cặp <kiểu tham số> <tên tham số>. Các tham số được phân cách nhau bởi dấu phẩy.
- Các ngoại lệ: nếu có thì mỗi ngoại lệ được phân cách nhau bởi dấu phẩy.
- Lớp trừu tượng là lớp có khai báo các phương thức (là trừu tượng).
- Các lớp dẫn xuất của nó sẽ cài đặt cụ thể các phương thức của lớp trừu

3. Phương thức trừu tượng

- abstract class LopCha_TruuTuong(){

```
    int ThuocTinh_1;
    String ThuocTinh_2;
    abstract void Xuat();
void HienThi(){
    System.out.println("Cha: tôi là phương thức bình thường của lớp cha trừu
tượng");
  }
}
```

3. Phương thức trừu tượng

- class LopCon extends LopCha_TruuTuong(){

```
void Xuat(){
    System.out.println("Con: tôi là phương thức triển khai của phương thức trong lớp cha trừu tượng");
  }
void HienThi(){
    System.out.println("Con: tôi là phương thức bình thường của lớp con");
   }
}
```

3. Phương thức trừu tượng

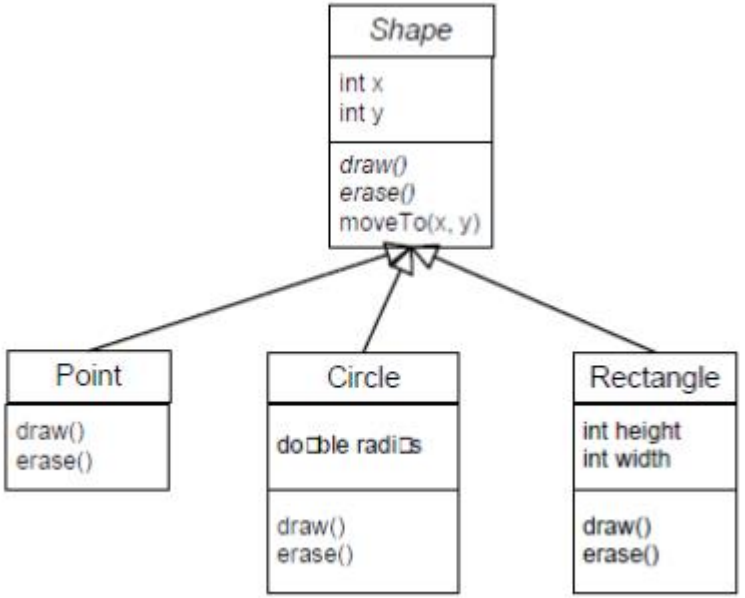
- public class ViDuTruuTuong {
  public static void main(String[] args) {
  ~~//Báo lỗi, vì ko thể tạo đối tượng cho lớp trừu tượng~~
  ~~LopCha_TruuTuong cha = new LopCha_TruuTuong();~~
  LopCha_TruuTuong cha = new LopCon();//upcasting
  cha.Xuat();
  cha.HienThi();

}}

- Con: tôi là phương thức triển khai của phương thức trong lớp cha trừu tượng
- Con: tôi là phương thức bình thường của lớp con

3. Phương thức trừu tượng

- Ví dụ: Lớp cha tổng quát Shape nên là lớp trừu tượng do ứng dụng không cần và không nên tạo đối tượng Shape. Ngoài ra, các phương thức draw và erase của lớp này cũng nên là phương thức trừu tượng do ta không thể nghĩ ra nội dung gì hữu ích cho chúng. Các lớp con cụ thể, Point, Circle, Rectangle, và các lớp mà sau này sẽ bổ sung vào thư viện khi cần, sẽ định nghĩa các phiên bản với nội dung riêng cụ thể phù hợp với chính mình.



3. Phương thức trừu tượng

```
abstract public class Shape {
  protected int x, y;
  protected Shape (int _x, int _y) { x = _x; y = _y; }

  abstract public void draw();
  abstract public void erase();

  public void moveTo(int _x, int _y) {
    erase();
    x = _x;
    y = _y;
    draw();
  }
}
```

```
public class Circle extends Shape {
  private double radius;
  public Circle(int _x, int _y, double _r) {
    super(_x, _y);
    radius = _r;
  }
  public void draw() {
    System.out.println("Draw circle");
  }
  public void erase() {
    System.out.println("Erase circle");
  }
}
```

3. Phương thức trừu tượng

Ví dụ:

```
abstract class A {
        abstract void method_1();
}

public class B extends A {
        public void method_1( ) {
                // cài đặt chi tiết cho phương thức method_1
                // trong lớp con B.
                // …
        }
}
```

```
public class C extends A
{
        public void method_1()
        {
                // cài đặt chi tiết cho
                //method_1 trong lớp con C.
                // …
        }
}
```

- Lưu ý:
  - Các phương thức được khai báo dùng các tiền tố private và static thì không được khai báo là trừu tượng abstract
  - Tiền tố private thì không thể truy xuất từ các lớp dẫn xuất, còn tiền tố static thì chỉ dùng riêng cho lớp khai báo mà thôi.

3. Phương thức trừu tượng

- Ví dụ: lớp trừu tượng và phương thức trừu tượng

```
// lop truu tuong Bike
abstract class Bike{
  abstract void run();  // phuong thuc truu tuong voi tu khoa abstract
}

// lop Honda4 ke thua lop truu tuong Bike
class Honda4 extends Bike{
void run(){
  System.out.println("Dang chay mot cach an toan..");
  }

// phuong thuc main()
public static void main(String args[]){
 Bike obj = new Honda4();
 obj.run();
}
}
```

3. Phương thức trừu tượng

- Ví dụ: kế thừa lớp Abstract trong java

```
abstract class Bank{
   abstract int getRateOfInterest();
}
class SBI extends Bank{
   int getRateOfInterest(){
       return 7;
   } // bat buoc phai cung cap trinh trien khai cua getRateOfInterest
}
class PNB extends Bank{
   int getRateOfInterest(){
     return 8;
   }// bat buoc phai cung cap trinh trien khai cua getRateOfInterest
}
```

3. Phương thức trừu tượng

- Ví dụ: kế thừa lớp Abstract trong java

```
class TestBank{
public static void main(String args[]){
// Tao mot doi tuong SBI moi
Bank b=new SBI(); //Neu doi tuong la PNB, phuong thuc cua PNB se duoc trieu hoi
int interest=b.getRateOfInterest(); //Trieu hoi phuong thuc cua SBI
System.out.println("Ti le lai suat la: "+interest+" %");
}
}
```

4. Ví dụ tính trừu tượng

- Viết chương trình khai báo một lớp trừu tượng là Animal có phương thức eat() và makeSound().
- Xây dựng các lớp Cat và Bird kế thừa lớp Animal trong đó:
  - Lớp Cat có phương thức run()
  - Lớp Bird có phương thức fly()

4. Ví dụ tính trừu tượng

- Viết chương trình khai báo một lớp trừu tượng là Animal có phương thức eat() và makeSound().

```
public abstract class Animal {
    public abstract void eat();

    public abstract void makeSound();
}
```

4. Ví dụ tính trừu tượng

- Xây dựng các lớp Cat và Bird kế thừa lớp Animal trong đó:
  - Lớp Cat có phương thức run()
  - Lớp Bird có phương thức fly()

```
public class Cat extends Animal{
    @Override
    public void eat() {
        System.out.println("Mèo đang ăn cá");
    }
                                    public void makeSound() {
                                        System.out.println("meow");
                                    }

                                    public void run(){
                                        System.out.println("Mèo có thể chạy");
                                    }
                                }
```

4. Ví dụ tính trừu tượng

- Xây dựng các lớp Cat và Bird kế thừa lớp Animal trong đó:
  - Lớp Cat có phương thức run()
  - Lớp Bird có phương thức fly()

```
public class Bird extends Animal {
    @Override
    public void eat() {
        System.out.println("Con chim đang ăn sâu");
    }
    @Override
    public void makeSound() {
        System.out.println("Con chim đang hót");
    }

    public void fly() {
        System.out.println("Con chim có thể bay");
    }
}
```

4. Ví dụ tính trừu tượng

- Viết chương trình khai báo một lớp Animal có phương thức eat() và makeSound().

```
public abstract class Animal {
   public abstract void eat();

   public abstract void makeSound();
}
```

Bỏ trừu tượng

```
public class Animal {
   public void eat(){

   }

   public void makeSound(){

   }
}
```

4. Ví dụ tính trừu tượng

- Xây dựng các lớp Cat và Bird kế thừa lớp Animal trong đó:
  - Lớp Cat có phương thức run()
  - Lớp Bird có phương thức fly()

```
public class Cat extends Animal{
    @Override
    public void eat() {
        System.out.println("Mèo đang ăn cá");
    }
                                public void makeSound() {
                                    System.out.println("meow");
                                }

                                public void run(){
                                    System.out.println("Mèo có thể chạy");
                                }
                            }
```

4. Ví dụ tính trừu tượng

- Xây dựng các lớp Cat và Bird kế thừa lớp Animal trong đó:
  - Lớp Cat có phương thức run()
  - Lớp Bird có phương thức fly()

```
public class Bird extends Animal {
    @Override
    public void eat() {
        System.out.println("Con chim đang ăn sâu");
    }
    @Override
    public void makeSound() {
        System.out.println("Con chim đang hót");
    }

    public void fly() {
        System.out.println("Con chim có thể bay");
    }
}
```

5. Phương thức Finalize

- Trong java không có kiểu dữ liệu con trỏ như trong C, người lập trình không cần phải quá bận tâm về việc cấp phát và giải phóng vùng nhớ, sẽ có một trình dọn dẹp hệ thống đảm trách việc này. Trình dọn dẹp hệ thống sẽ dọn dẹp vùng nhớ cấp phát cho các đối tượng trước khi hủy một đối tượng.
- Phương thức finalize() là một phương thức đặc biệt được cài đặt sẵn cho các lớp. Trình dọn dẹp hệ thống sẽ gọi phương thức này trước khi hủy một đối tượng. Vì vậy việc cài đặt một số thao tác giải phóng, dọn dẹp vùng nhớ đã cấp phát cho các đối tượng dữ liệu trong phương thức finalize() sẽ giúp cho người lập trình chủ động kiểm soát tốt quá trình hủy đối tượng thay vì giao cho trình dọn dẹp hệ thống tự động. Đồng thời việc cài đặt trong phương thức finalize() sẽ giúp cho bộ nhớ được giải phóng tốt hơn, góp phần cải tiến tốc độ chương trình.

5. Phương thức Finalize

Ví dụ:

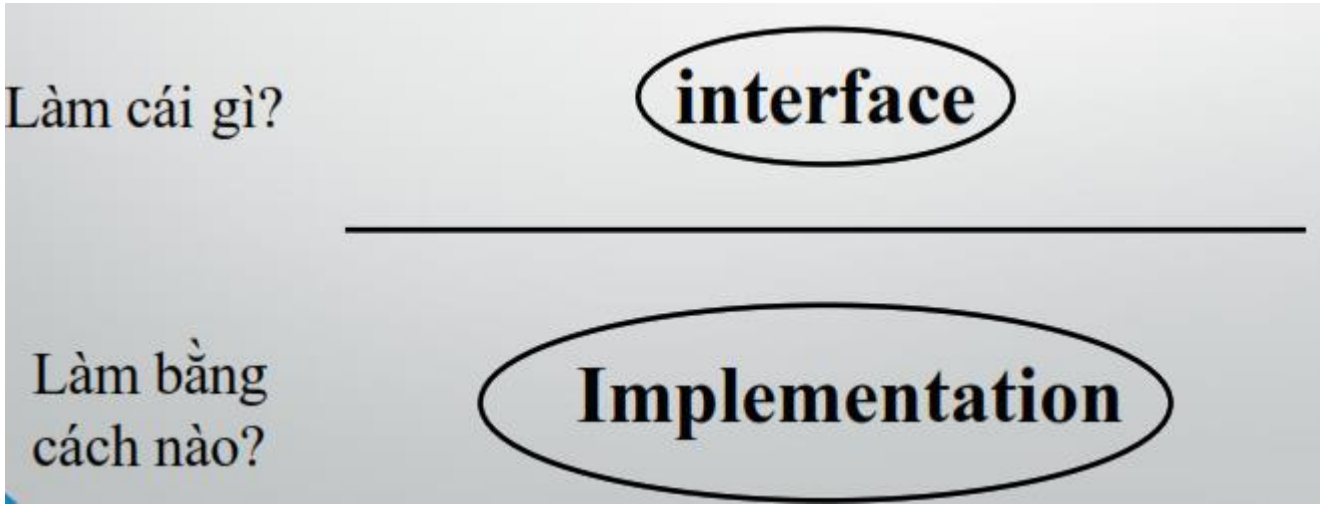
```
class A  {
        // Khai báo các thuộc tính
        public void method_1( ) { // ...  }
        protected void finalize()
        {
                // Có thể dùng để đóng tất cả các kết nối
                // vào cơ sở dữ liệu trước khi hủy đối tượng.
                // ...
        }
}
```

6. Giao diện (Interface)

- Interface là một dạng của lớp trừu tượng được sử dụng với mục đích hỗ trợ tính đa hình
- Trong interface không có bất cứ một cài đặt nào, chỉ có các nguyên mẫu phương thức, thuộc tính, chỉ mục được khai báo trong interface.
- Tất cả các thành phần khai báo trong interface mặc định là public (nên không có từ khóa về mức độ truy cập trong khai báo các thuộc tính và phương thức)
- Khi một lớp kế thừa một interface ta nói rằng lớp đó thực thi (Implement) interface

6. Giao diện (Interface)

- Nhằm tránh những nhập nhằng của tính chất đa kế thừa của C++, Java không cho phép kế thừa trực tiếp từ nhiều hơn một lớp cha. Nghĩa là Java không cho phép đa kế thừa trực tiếp, nhưng cho phép cài đặt nhiều Interface để có thể thừa hưởng thêm các thuộc tính và phương thức của các Interface đó.
- Tách biệt giữa giao tiếp và cài đặt cụ thể



6. Giao diện (Interface)

```
interface Printable {
    void print();
}
```

- Khai báo giao diện

MứcĐộTruyCập Interface TênInterface[extends danh sách Interface]

{

  //Nội dung Interface

}

```
interface Showable extends Printable{
    void show();
}
```

- Trong đó:
  - MứcĐộTruyCập: thường là public; nếu không khai báo tường minh thì mặc định là public
  - danh sách Interface: danh sách các Interface cha đã được định nghĩa để kế thừa, các Interface được phân cách nhau bởi dấu phẩy
  - Lưu ý: một Interface chỉ có thể kế thừa từ các Interface khác mà không thể được kế thừa từ các lớp sẵn có.

6. Giao diện (Interface)

- Khai báo Interface
- Interface được khai báo như một lớp. Nhưng các thuộc tính của interface là các hằng (khai báo dùng từ khóa final) và các phương thức của interface là trừu tượng (mặc dù không có từ khóa abstract).
- Trong các lớp có cài đặt các interface ta phải tiến hành cài đặt cụ thể các phương thức này.

6. Giao diện (Interface)

```
public class VD {
    public static void main(String[] args) {
        QuanLy abc = new QuanLy();
        abc.show();
    }
}

interface Nguoi{
    public static final int TUOI_TB = 100;
    int CANNANG_TB = 60;

    public abstract void show();
    void study();
}
```

```
class QuanLy implements Nguoi{
    @Override
    public void show() {
        System.out.println("Tuoi trung bình: "+ Tuoi_TB);

    }

    @Override
    public void study() {
        // TODO Auto-generated method stub
    }
}
```

Java tự động hiểu

6. Giao diện (Interface)

- Khai báo phương thức của giao diện

[public] kiểu_giá_trị_trả_về Tên_phương_thức ([các tham số]) [throws danh sách ngoại lệ];

- Tính chất: tính chất của một thuộc tính hay phương thức của Interface luôn là public. Nếu không khai báo tường minh thì giá trị mặc định cũng là public.
- Đối với thuộc tính, thì luôn phải thêm là hằng (final) và tĩnh (static).
- Kiểu giá trị trả về: có thể là các kiểu cơ bản của Java, cũng có thể là kiểu do người dùng tự định nghĩa (kiểu đối tượng).
- Tên phương thức: tuân thủ theo quy tắc đặt tên phương thức của lớp

6. Giao diện (Interface)

- Khai báo phương thức của Interface
- Các tham số: nếu có thì mỗi tham số được xác định bằng một cặp kiểu tham số tên tham số. Các tham số được phân cách nhau bởi dấu phẩy.
- Các ngoại lệ: nếu có thì mỗi ngoại lệ được phân cách nhau bởi dấu phẩy.
- Để khai báo rằng một lớp cài đặt một interface, ta dùng từ khóa implements thay vì extends, theo sau là tên của interface.
- Một lớp có thể cài đặt một vài interface và đồng thời là lớp con của một lớp khác. Chẳng hạn lớp Dog vừa là lớp con của Canine, vừa là lớp cài đặt interface Pet:

```
class Dog extends Canine implements Pet {...}
```

6. Giao diện (Interface)

- Lưu ý:
  - Các phương thức của interface chỉ được khai báo dưới dạng mẫu mà không có cài đặt chi tiết (có dấu ; ngay sau khai báo và không có phần cài đặt trong dấu { }).
  - Phần cài đặt chi tiết của các phương thức chỉ được thực hiện trong các lớp sử dụng interface đó.
  - Các thuộc tính của interface luôn có tính chất là final, static và public. Do đó, cần gán giá trị khởi đầu ngay khi khai báo thuộc tính của interface.

6. Giao diện (Interface)

- Phân biệt Abstract và Interface

Lớp trừu tượng	Interface
Lớp trừu tượng có thể có các phương thức abstract và non-abstract	Interface chỉ có thể có phương thức abstract
Lớp trừu tượng không hỗ trợ đa kế thừa	Interface hỗ trợ **đa kế thừa**
Lớp trừu tượng có thể có các biến **final, non-final, static và non-static**	Interface chỉ có các biến **static và final**
Lớp trừu tượng có thể có phương thức static, phương thức main và constructor	Interface không thể có phương thức static, main hoặc constructor.
Từ khóa abstract được sử dụng để khai báo lớp trừu tượng	Từ khóa interface được sử dụng để khai báo Interface
Lớp trừu tượng có thể cung cấp trình triển khai của Interface	Interface không cung cấp trình triển khai cụ thể của lớp abstract
Ví dụ: public abstract class Shape{ public abstract void draw(); }	Ví dụ: public interface Drawable{ void draw(); }

6. Giao diện (Interface)

- Khi nào nên cho một lớp là lớp độc lập, lớp con, lớp trừu tượng, hay nên biến nó thành interface?
- Một lớp nên là lớp độc lập, nghĩa là nó không thừa kế lớp nào (ngoại trừ Object) nếu nó không thỏa mãn kiểm tra IS-A đối với bất cứ loại nào khác.
- Một lớp nên là lớp con nếu ta cần cho nó làm một phiên bản chuyên biệt hơn của một lớp khác và cần cài đè hành vi có sẵn hoặc bổ sung hành vi mới.
- Một lớp nên là lớp cha nếu ta muốn định nghĩa một khuôn mẫu cho một nhóm các lớp con, và ta có một chút muốn cài đặt mà tất cả các lớp con kia có thể sử dụng.
- Cho lớp đó làm lớp trừu tượng nếu ta muốn đảm bảo rằng không ai được tạo đối tượng thuộc lớp đó.
- Dùng một interface nếu ta muốn định nghĩa một vai trò mà các lớp khác có thể nhận, bất kể các lớp đó thuộc cây thừa kế nào.


Q
&
A