





Bộ môn Công nghệ Phần mềm  
Viện CNTT & TT  
Trường Đại học Bách Khoa Hà Nội

---

## LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG


### Bài 06. Một số kỹ thuật trong kế thừa



## Mục tiêu của bài học

- Trình bày nguyên lý định nghĩa lại trong kế thừa
- Đơn kế thừa và đa kế thừa
- Giao diện và lớp trừu tượng
- Sử dụng các vấn đề trên với ngôn ngữ lập trình Java.


2



## Nội dung

1. Định nghĩa lại (Redefine/Overriding)
2. Lớp trừu tượng (Abstract class)
3. Đơn kế thừa và đa kế thừa
4. Giao diện (Interface)

3



## Nội dung

1. **Định nghĩa lại (Redefine/Override)**
2. Lớp trừu tượng (Abstract class)
3. Đơn kế thừa và đa kế thừa
4. Giao diện (Interface)

4

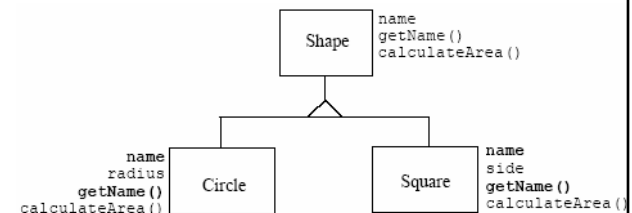
## 1. Định nghĩa lại hay ghi đè

- Lớp con có thể định nghĩa phương thức trùng tên với phương thức trong lớp cha:
  - Nếu phương thức mới chỉ trùng tên và khác chữ ký (số lượng hay kiểu dữ liệu của đối số)
    - Chồng phương thức (Method Overloading)
  - Nếu phương thức mới hoàn toàn giống về giao diện (chữ ký)
    - Định nghĩa lại hoặc ghi đè (Method Redefine/Override)

5

## 1. Định nghĩa lại hay ghi đè (2)

- Phương thức ghi đè sẽ thay thế hoặc làm rõ hơn cho phương thức cùng tên trong lớp cha
- Đối tượng của lớp con sẽ hoạt động với phương thức mới phù hợp với nó



6

```

class Shape {
    protected String name;
    Shape(String n) { name = n; }
    public String getName() { return name; }
    public float calculateArea() { return 0.0f; }
}
class Circle extends Shape {
    private int radius;
    Circle(String n, int r) {
        super(n);
        radius = r;
    }
    public float calculateArea() {
        float area = (float) (3.14 * radius * radius);
        return area;
    }
}

```

7

```

class Square extends Shape {
    private int side;
    Square(String n, int s) {
        super(n);
        side = s;
    }
    public float calculateArea() {
        float area = (float) side * side;
        return area;
    }
}

```

8

## Thêm lớp Triangle

```
class Triangle extends Shape {
    private int base, height;
    Triangle(String n, int b, int h) {
        super(n);
        base = b; height = h;
    }
    public float calculateArea() {
        float area = 0.5f * base * height;
        return area;
    }
}
```

9

## this và super

- this và super có thể sử dụng cho các phương thức/thuộc tính non-static và phương thức khởi tạo
  - this**: tìm kiếm phương thức/thuộc tính trong lớp hiện tại
  - super**: tìm kiếm phương thức/thuộc tính trong lớp cha trực tiếp
- Từ khóa **super** cho phép tái sử dụng các đoạn mã của lớp cha trong lớp con

10

```
package abc;
public class Person {
    protected String name;
    protected int age;
    public String getDetail() {
        String s = name + "," + age;
        return s;
    }
}

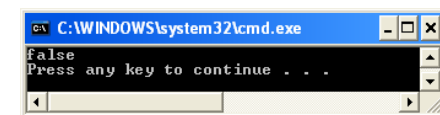
import abc.Person;
public class Employee extends Person {
    double salary;
    public String getDetail() {
        String s = super.getDetail() + "," + salary;
        return s;
    }
}
```

11

## Bài tập: Sửa lại cho kết quả là true

```
class Value {
    int i;
}

public class EqualsMethod2 {
    public static void main(String[] args) {
        Value v1 = new Value();
        Value v2 = new Value();
        v1.i = v2.i = 100;
        System.out.println(v1.equals(v2));
    }
}
```



12

## Solution

```
class Value {
    int i;
    public boolean equals(Object obj){
        if (obj instanceof Value)
            return ((Value)obj).i == this.i;
        return false;
    }
}
```

13

## 1. Định nghĩa lại hay ghi đè (3)

### ■ Một số quy định

- Phương thức ghi đè trong lớp con phải
  - Có danh sách tham số giống hệt phương thức kế thừa trong lớp cha.
  - Có cùng kiểu trả về với phương thức kế thừa trong lớp cha
- Không được phép ghi đè:
  - Các phương thức hằng (final) trong lớp cha
  - Các phương thức static trong lớp cha
- (Các phương thức private trong lớp cha, lớp con không kế thừa được)

14

## 1. Định nghĩa lại hay ghi đè (3)

### ■ Một số quy định (tiếp)

- Các chỉ định truy cập không giới hạn chặt hơn phương thức trong lớp cha
  - Ví dụ, nếu ghi đè một phương thức protected, thì phương thức mới có thể là protected hoặc public, mà không được là private.

15

## Ví dụ

```
class Parent {
    public void doSomething() {}
    protected int doSomething2() {
        return 0;
    }
}
class Child extends Parent {
    protected void doSomething() {}
    protected void doSomething2() {}
}
```

cannot override: attempting to use incompatible return type

cannot override: attempting to assign weaker access privileges; was public

16

## Ví dụ

```
class Parent {
    public void doSomething() {}
    private int doSomething2() {
        return 0;
    }
}
class Child extends Parent {
    public void doSomething() {}
    private void doSomething2() {}
}
```

17

## Bài tập

- Sửa lại lớp NhanVien:
  - 3 thuộc tính không hằng của NhanVien kế thừa lại cho lớp TruongPhong
- Viết mã nguồn của lớp TruongPhong như hình vẽ
  - Viết các phương thức khởi tạo cần thiết để khởi tạo các thuộc tính của lớp TruongPhong
  - Lương của trưởng phòng = Lương Cơ bản \* hệ số lương + phụ cấp

NhanVien
#tenNhanVien:String
#heSoLuong:double
#luongCoBan: double
+LUONG_MAX:double=20.000.000
+tangLuong(double):boolean
+tinhLuong():double
+inTTin()



TruongPhong
-phuCap:double
-soNamDuongChuc:double
+tinhLuong():double
+inTTin()

18

## Nội dung

- Định nghĩa lại (Redefine/Overriding)
- Lớp trừu tượng (Abstract class)**
- Đơn kế thừa và đa kế thừa
- Giao diện (Interface)

19

## 2. Lớp trừu tượng (Abstract Class)

- Không thể thể hiện hóa (instantiate – tạo đối tượng của lớp) trực tiếp
- Chưa đầy đủ, thường được sử dụng làm lớp cha. Lớp con kế thừa nó sẽ hoàn thiện nốt.

20

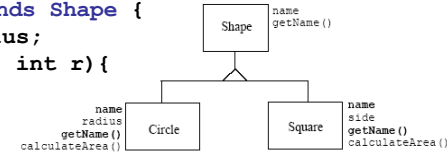
## 2. Lớp trừu tượng (2)

- Để trở thành một lớp trừu tượng, cần:
  - Khai báo với từ khóa `abstract`
  - Có thể chứa các phương thức trừu tượng (abstract method - chỉ có chữ ký mà không có cài đặt cụ thể)
    - `public abstract float calculateArea();`
  - Lớp con khi kế thừa phải cài đặt cụ thể cho các phương thức trừu tượng của lớp cha → Phương thức trừu tượng không thể khai báo là `final` hoặc `static`.
- Nếu một lớp có một hay nhiều phương thức trừu tượng thì nó phải là lớp trừu tượng

21

```
abstract class Shape {
    protected String name;
    Shape(String n) { name = n; }
    public String getName() { return name; }
    public abstract float calculateArea();
}
```

```
class Circle extends Shape {
    private int radius;
    Circle(String n, int r) {
        super(n);
        radius = r;
    }
```



```
public float calculateArea() {
    float area = (float) (3.14 * radius * radius);
    return area;
}
```

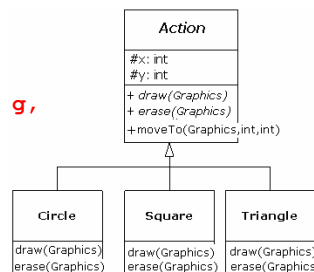
Lớp con bắt buộc phải override tất cả các phương thức abstract của lớp cha

22

## Ví dụ lớp trừu tượng

```
import java.awt.Graphics;
abstract class Action {
    protected int x, y;
    public void moveTo(Graphics g,
        int x1, int y1) {
        erase(g);
        x = x1; y = y1;
        draw(g);
    }
```

```
abstract public void erase(Graphics g);
abstract public void draw(Graphics g);
}
```



23

## Ví dụ lớp trừu tượng (2)

```
class Circle extends Action {
    int radius;
    public Circle(int x, int y, int r) {
        super(x, y); radius = r;
    }
    public void draw(Graphics g) {
        System.out.println("Draw circle at ("
            + x + ", " + y + ")");
        g.drawOval(x-radius, y-radius,
            2*radius, 2*radius);
    }
    public void erase(Graphics g) {
        System.out.println("Erase circle at ("
            + x + ", " + y + ")");
        // paint the circle with background color...
    }
}
```

24

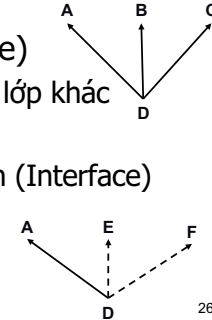
## Nội dung

1. Định nghĩa lại (Redefine/Overriding)
2. Lớp trừu tượng (Abstract class)
3. **Đơn kế thừa và đa kế thừa**
4. Giao diện (Interface)

25

## Đa kế thừa và đơn kế thừa

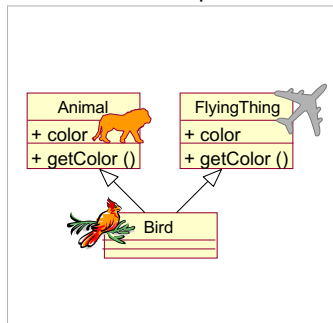
- Đa kế thừa (Multiple Inheritance)
  - Một lớp có thể kế thừa nhiều lớp khác
  - C++ hỗ trợ đa kế thừa
- Đơn kế thừa (Single Inheritance)
  - Một lớp chỉ được kế thừa từ một lớp khác
  - Java chỉ hỗ trợ đơn kế thừa
  - → Đưa thêm khái niệm Giao diện (Interface)



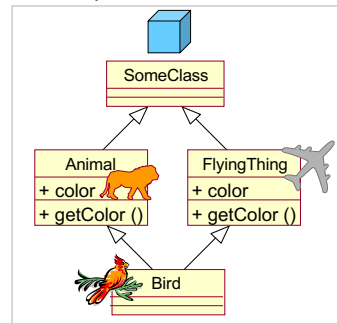
26

## Vấn đề gặp phải trong Đa kế thừa

Name clashes on  
attributes or operations



Repeated inheritance

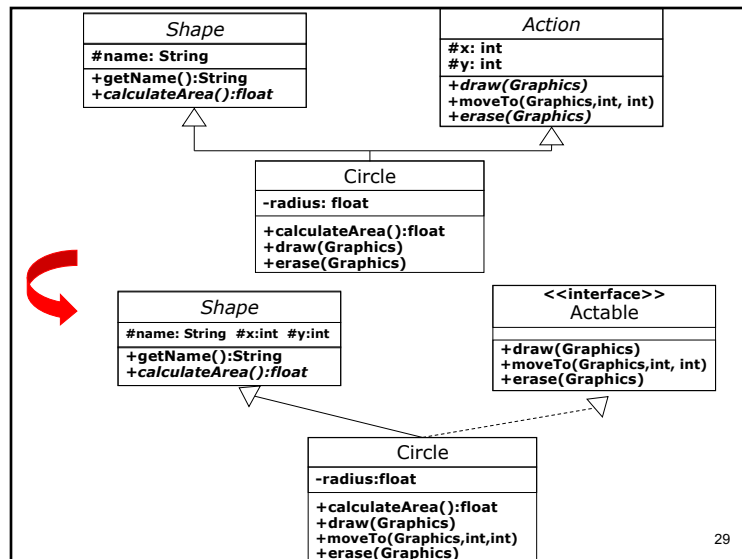


Resolution of these problems is implementation-dependent.

28

## Nội dung

1. Định nghĩa lại (Redefine/Overriding)
2. Lớp trừu tượng (Abstract class)
3. Đơn kế thừa và đa kế thừa
4. **Giao diện (Interface)**



29

## 4. Giao diện-Interface

- Interface: đặc tả cho các bản cài đặt (implementation) khác nhau.
- Interface định nghĩa một "kiểu" chỉ chứa định nghĩa hằng và phương thức trừu tượng
- Interface không cài đặt bất cứ một phương thức nào nhưng để lại cấu trúc thiết kế trên bất cứ lớp nào sử dụng nó.

30

## 4. Giao diện

Cho phép một lớp có thể kế thừa (thực thi - implement) nhiều giao diện một lúc.  
Không thể thể hiện hóa (instantiate) trực tiếp

31

## 4. Giao diện (2)

- Để trở thành giao diện, cần
  - Sử dụng từ khóa interface để định nghĩa
  - Chỉ được bao gồm:
    - Chữ ký các phương thức (method signature)
    - Các thuộc tính khai báo hằng (static & final)
- Lớp thực thi giao diện
  - Hoặc là lớp trừu tượng (abstract class)
  - Hoặc là bắt buộc phải cài đặt chi tiết toàn bộ các phương thức trong giao diện nếu là lớp instance.

32



## 4. Giao diện (3)

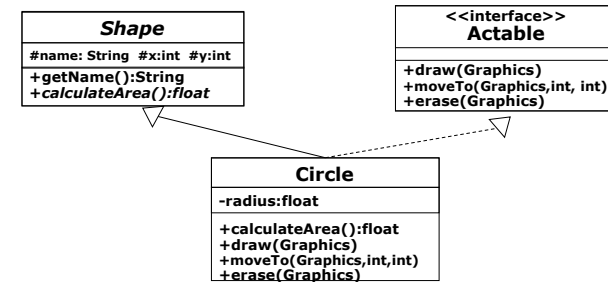
- Cú pháp thực thi trên Java:
  - <Lớp con> [**extends** <Lớp cha>] **implements** <Danh sách giao diện>
  - <Giao diện con> **extends** <Giao diện cha>

### Ví dụ:

```
public interface DoiXung {...}
public interface DiChuyen {...}
public class HìnhVuong extends TuGiac
    implements DoiXung, DiChuyen {
    ...
}
```

33

## Ví dụ



34

```
import java.awt.Graphics;
abstract class Shape {
    protected String name;
    protected int x, y;
    Shape(String n, int x, int y) {
        name = n; this.x = x; this.y = y;
    }
    public String getName() {
        return name;
    }
    public abstract float calculateArea();
}
interface Actable {
    public void draw(Graphics g);
    public void moveTo(Graphics g, int x1, int y1);
    public void erase(Graphics g);
}
```

35

```
class Circle extends Shape implements Actable {
    private int radius;
    public Circle(String n, int x, int y, int r) {
        super(n, x, y); radius = r;
    }
    public float calculateArea() {
        float area = (float) (3.14 * radius * radius);
        return area;
    }
    public void draw(Graphics g) {
        System.out.println("Draw circle at ("
            + x + ", " + y + ")");
        g.drawOval(x-radius, y-radius, 2*radius, 2*radius);
    }
    public void moveTo(Graphics g, int x1, int y1) {
        erase(g); x = x1; y = y1; draw(g);
    }
    public void erase(Graphics g) {
        System.out.println("Erase circle at ("
            + x + ", " + y + ")");
        // paint the region with background color...
    }
}
```

36

## Lớp trừu tượng vs. Giao diện

### Lớp trừu tượng

- Có thể có các phương thức abstract, có thể chứa các phương thức instance
- Có thể chứa các phương thức protected và static
- Có thể chứa các thuộc tính final và non-final
- Một lớp chỉ có thể kế thừa một lớp trừu tượng

### Giao diện

- Chỉ có thể chứa chữ ký phương thức (danh sách các phương thức)
- Chỉ có thể chứa các phương thức public mà không có mã nguồn
- Chỉ có thể chứa các thuộc tính hằng
- Một lớp có thể thực thi (kế thừa) nhiều giao diện

37

## Nhược điểm của Giao diện để giải quyết vấn đề Đa kế thừa

- Không cung cấp một cách tự nhiên cho các tình huống không có sự đụng độ về kế thừa xảy ra
- Kế thừa là để Tái sử dụng mã nguồn nhưng Giao diện không làm được điều này

```

classDiagram
    class X {
        +getA()
    }
    class Y {
        +getB()
    }
    class Z {
    }
    class CLASS_A {
        +BehaviorA()
    }
    class CLASS_B {
        +BehaviorB()
    }
    class CLASS_AB {
        +BehaviorA()
        +BehaviorB()
    }
    Z <|-- X
    Z <|-- Y
    CLASS_AB <|-- CLASS_A
    CLASS_AB <|-- CLASS_B
  
```

38