

Bài 7

# Các kỹ thuật kế thừa

# Nội dung

1. Ghi đè
2. Lớp trừu tượng
3. Giao diện

**1**

# Ghi đè

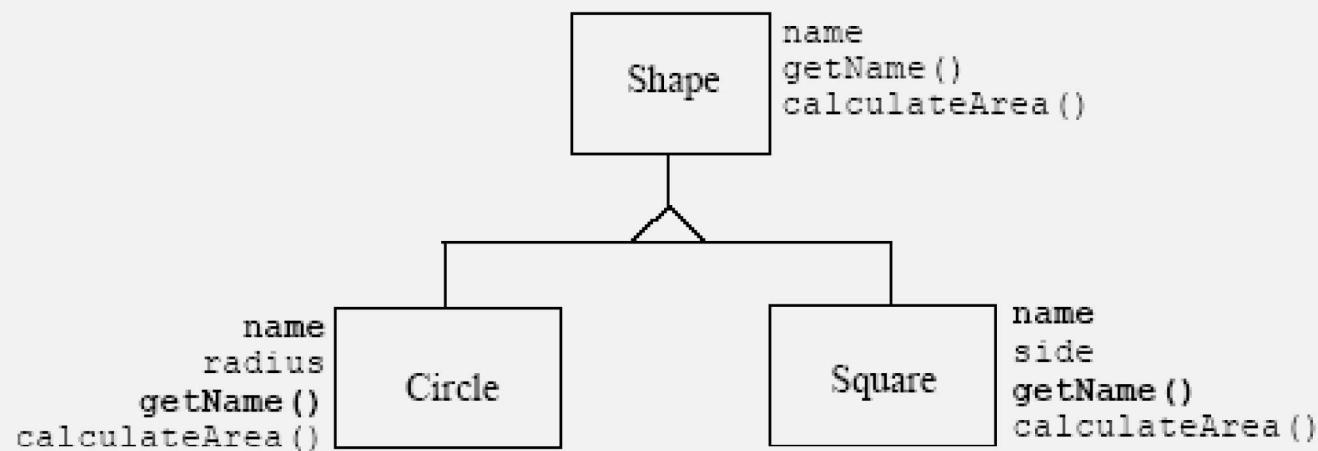
Override

# Mối quan hệ kế thừa

- Lớp con
  - Là một loại (**is-a-kind-of**) của lớp cha
  - Kế thừa các thành phần dữ liệu và các hành vi của lớp cha
  - Chi tiết hóa cho phù hợp với mục đích sử dụng mới
    - + Extension: Thêm các thuộc tính/hành vi mới
    - + **Redefinition (Method Overriding):** **Chỉnh sửa lại các hành vi kế thừa từ lớp cha**

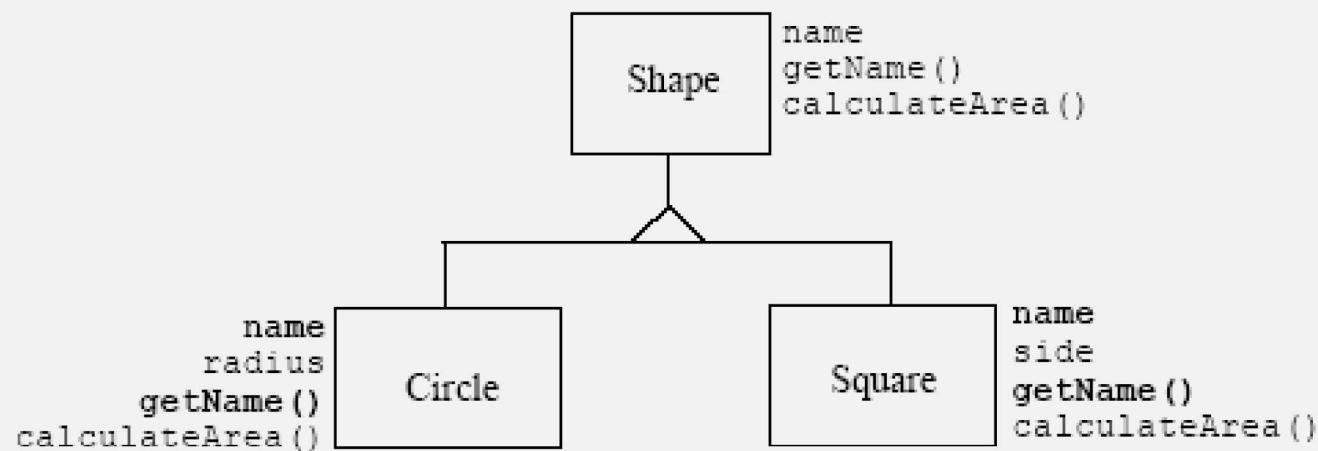
# Định nghĩa lại hay ghi đè

- 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



# Định nghĩa lại hay ghi đè

- Cú pháp: Phương thức ở lớp con hoàn toàn giống về chữ ký với phương thức ở lớp cha
  - Trùng tên & danh sách tham số
  - Mục đích: Để thể hiện cùng bản chất công việc

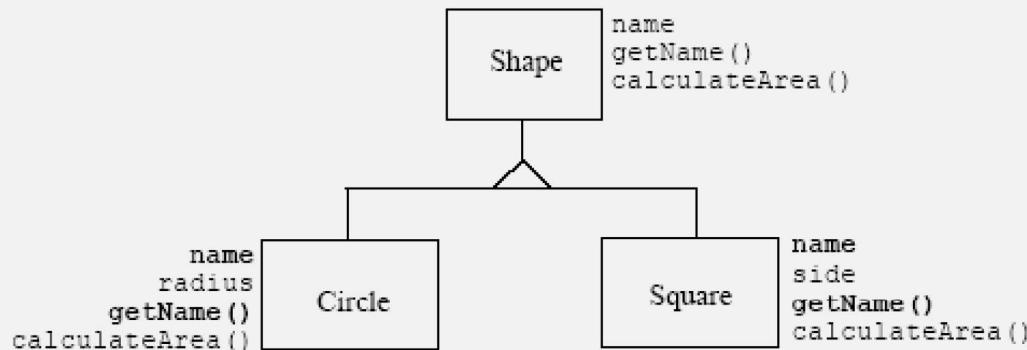


# Ví dụ

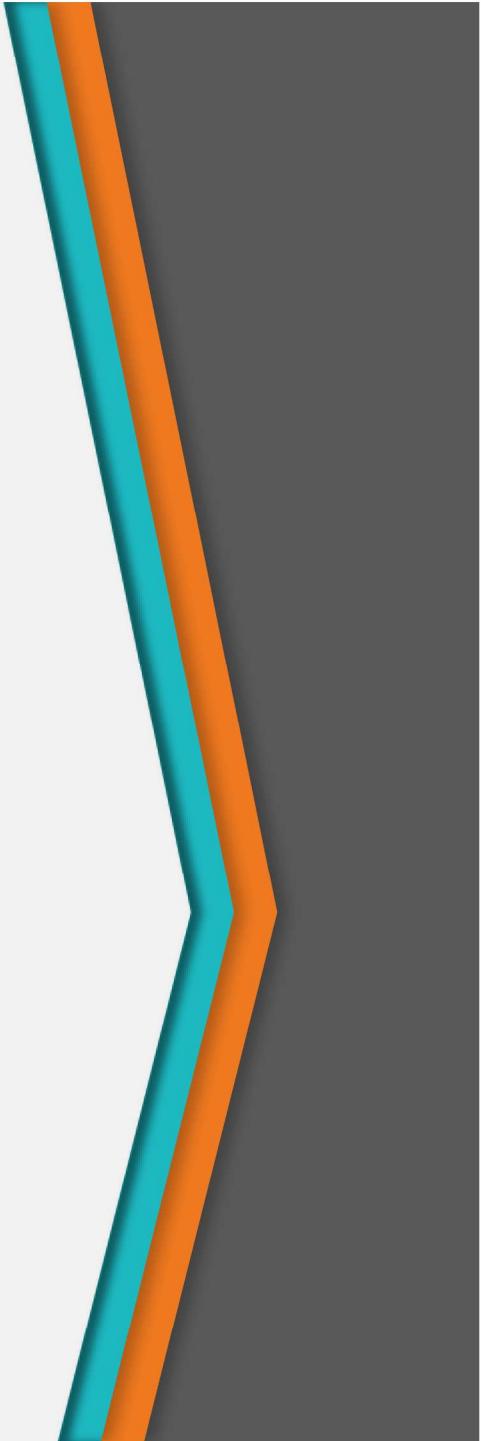
```
class Shape {  
    protected String name;  
    Shape(String n) { name = n; }  
    public String getName() { return name; }  
    public float calculateArea() { return 0.0f; }  
}  
  
class Circle extends Shape {  
    private int radius;  
    Circle(String n, int r){  
        super(n);  
 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

# Lớp Square

```
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;  
    }  
}
```



# Viết lớp Triangle



# Sử dụng từ khóa *super*

- Tái sử dụng các đoạn mã của lớp cha trong lớp con
- Gọi phương thức khởi tạo `super(danh sách tham số);`
  - Bắt buộc nếu lớp cha không có phương thức khởi tạo mặc định
- Gọi các phương thức của lớp cha `super.tênPt(danh sách tham số);`

# Ví dụ

```
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;
    }
}
```

# Quy định trong ghi đè

- 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
- 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.

# Ví dụ

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

*Không ghi đè được do không  
cùng kiểu trả về*

```
class Child extends Parent {  
    protected void doSomething() {}  
    protected void doSomething2() {}  
}
```

*Không ghi đè được do chỉ định truy cập  
yếu hơn (public -> protected)*

# Quy định trong ghi đè (tiếp)

- Không được phép ghi đè:
  - Các phương thức **static** trong lớp cha
  - Các phương thức **private** trong lớp cha
  - Các phương thức hằng (**final**) trong lớp cha

# Hạn chế ghi đè

- Đôi lúc ta muốn hạn chế việc định nghĩa lại vì các lý do sau:
  - Tính đúng đắn: Định nghĩa lại một phương thức trong lớp dẫn xuất có thể làm sai lạc ý nghĩa của nó
  - Tính hiệu quả: Cơ chế kết nối động không hiệu quả về mặt thời gian bằng kết nối tĩnh. Nếu biết trước sẽ không định nghĩa lại phương thức của lớp cơ sở thì nên dùng từ khóa final đi với phương thức

```
public final String basename () {  
    return "Person";  
}
```

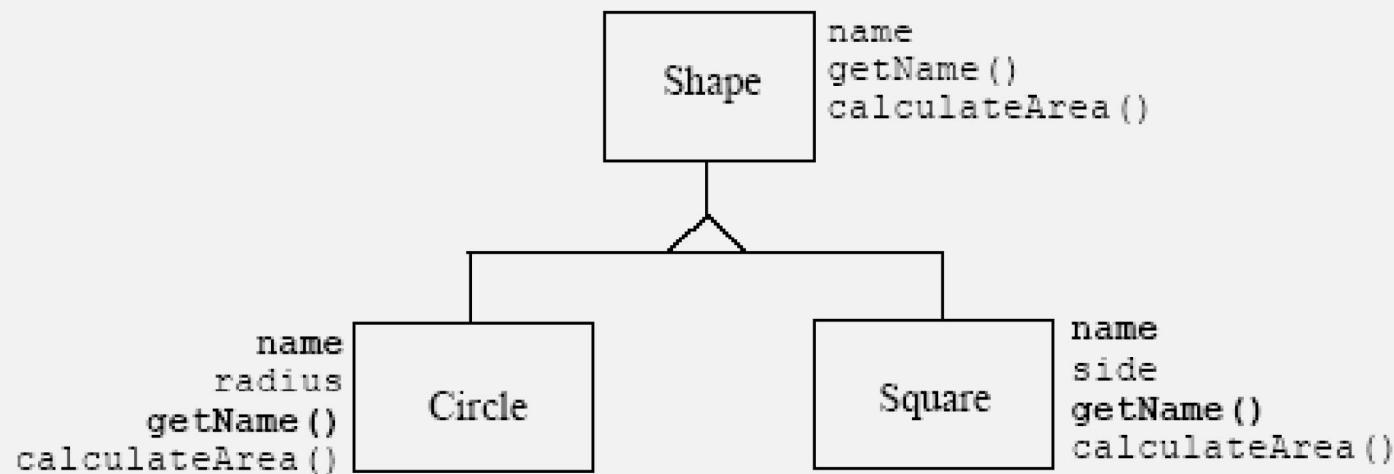
# 2

## Lớp trừu tượng

Abstract class

# Lớp trừu tượng

- Xét ví dụ: Lớp Shape



- Là một lớp "không rõ ràng", khó hình dung ra các đối tượng cụ thể
- Có những phương thức như `calculateArea()` khó cài đặt cụ thể

# Lớp trừu tượng

- Đặc điểm của lớp trừu tượng
  - Không thể thể hiện hóa (stantiate – 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.
    - + Lớp trừu tượng có thể chứa các phương thức trừu tượng không được định nghĩa
    - + Các lớp dẫn xuất có trách nhiệm định nghĩa lại (overriding) các phương thức trừu tượng này

# Lớp trừu tượng

- Không thể tạo đối tượng trực tiếp từ các lớp trừu tượng
- Thường lớp trừu tượng được dùng để định nghĩa các "khái niệm chung", đóng vai trò làm lớp cơ sở cho các lớp "cụ thể" khác.

# Lớp trừu tượng

- Cú pháp
  - Khai báo với từ khóa **abstract**

```
public abstract class Shape {  
    // Nội dung lớp  
}
```
- Phương thức trừu tượng
  - Chỉ có chữ ký mà không có cài đặt cụ thể
  - Khai báo với từ khóa **abstract**

```
public abstract float calculateArea();
```

# Lớp trừu tượng

- Lớp trừu tượng cần chứa ít nhất một phương thức trừu tượng (abstract method)
- Nếu một lớp có một hay nhiều phương thức trừu tượng thì nó phải là lớp trừu tượng
- 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.

# Ví dụ 1

```
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

## Ví dụ 2

```
abstract class Point {  
    private int x, y;  
    public Point(int x, int y) {  
        this.x = x; this.y = y;  
    }  
    public void move(int dx, int dy) {  
        x += dx; y += dy;  
        plot();  
    }  
    public abstract void plot();  
    // phương thức trừu tượng không có  
    // phần code thực hiện  
}
```

## Ví dụ 2 (tiếp)

```
abstract class ColoredPoint extends Point {  
    int color;  
    public ColoredPoint(int x, int y, int color) {  
        super(x, y); this.color = color;  
    }  
}  
  
class SimpleColoredPoint extends ColoredPoint {  
    public SimpleColoredPoint(int x, int y,  
                             int color) {  
        super(x, y, color);  
    }  
    public void plot() { ... }  
    // code to plot a SimplePoint  
}
```

# 3

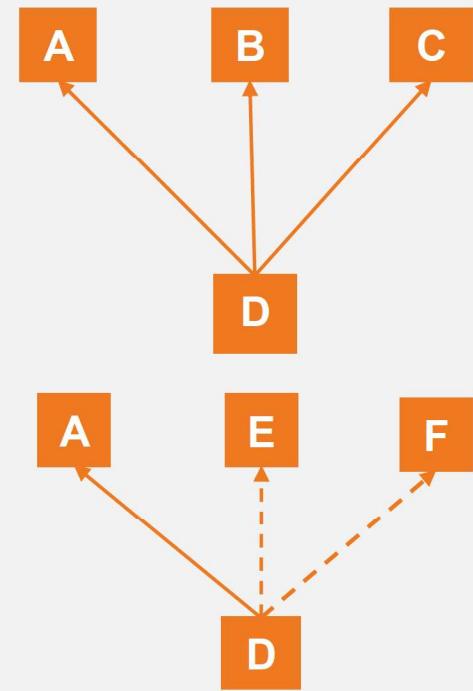
## Giao diện

Interface



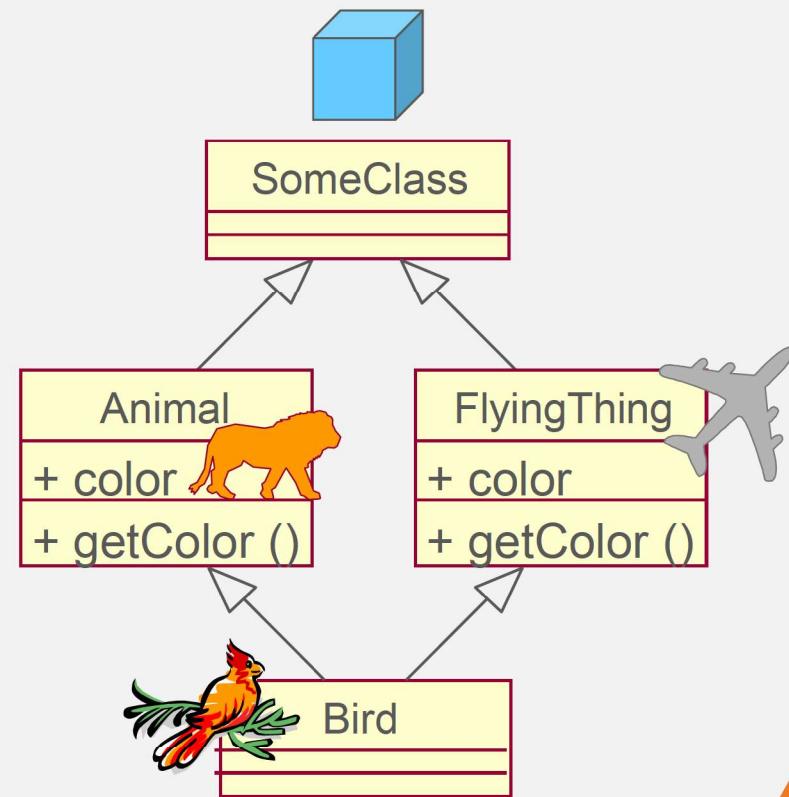
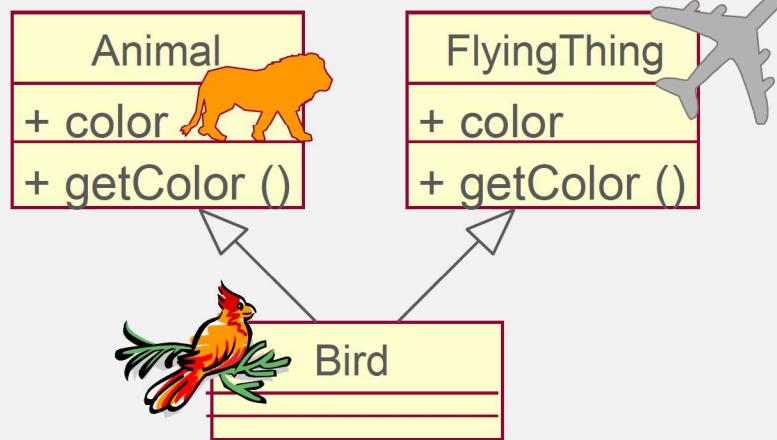
# Đa kế thừa và đơn kế thừa

- 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)



# Vấn đề gặp phải trong đa kế thừa

- Name collision
- "Diamond shape" problem



# Giao diện

- 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.

# Giao diện

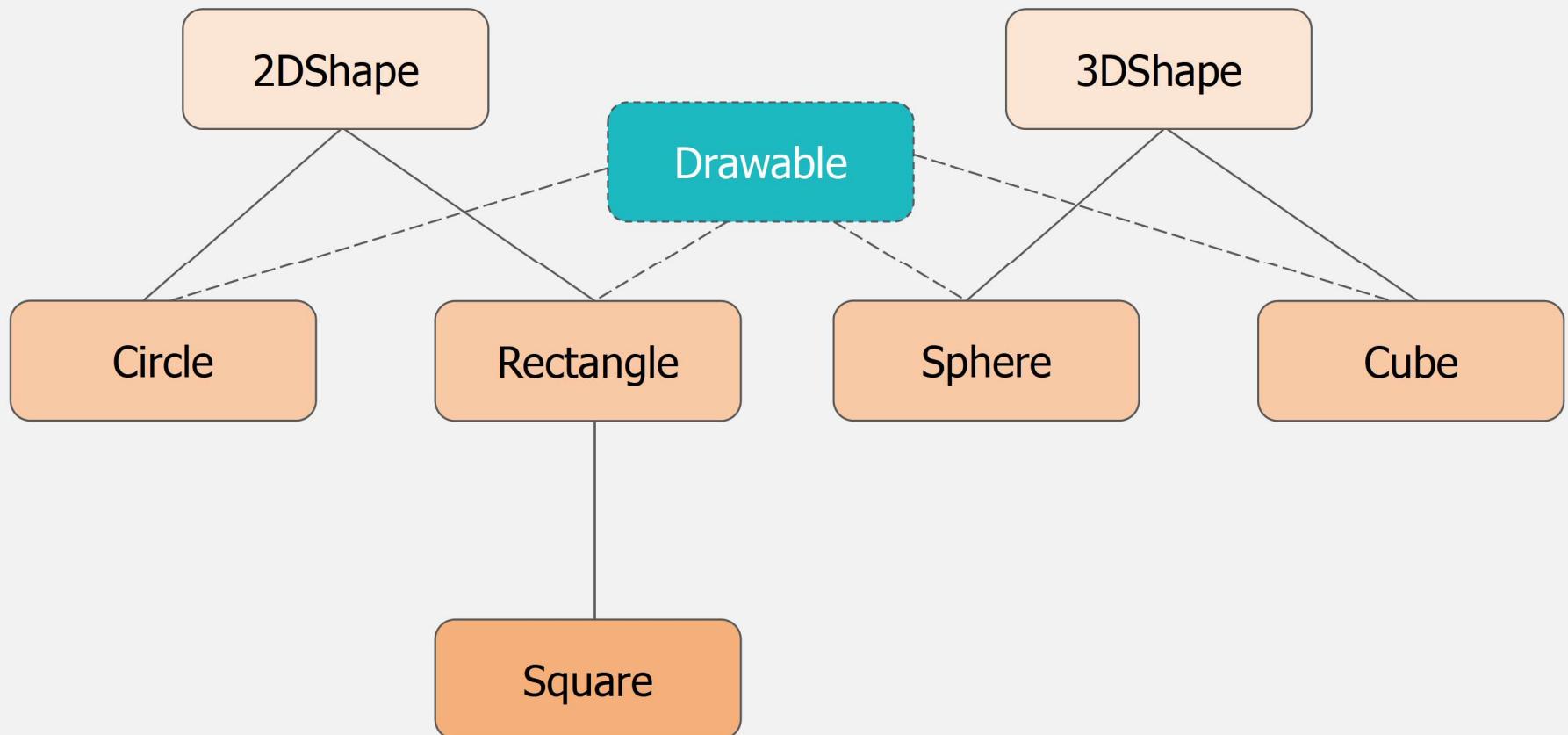
- Cú pháp khai báo giao diện trên Java:  
`interface <Tên giao diện>`  
`<Giao diện con> extends <Giao diện cha>`
- Ví dụ:  
`public interface DoiXung {...}`  
`public interface Can extends DoiXung {...}`  
`public interface DiChuyen {...}`

# Giao diện

- Cú pháp thực thi giao diện  
<Lớp con> [*extends* <Lớp cha>] **implements** <Đanh sách giao diện>
- Ví dụ:  

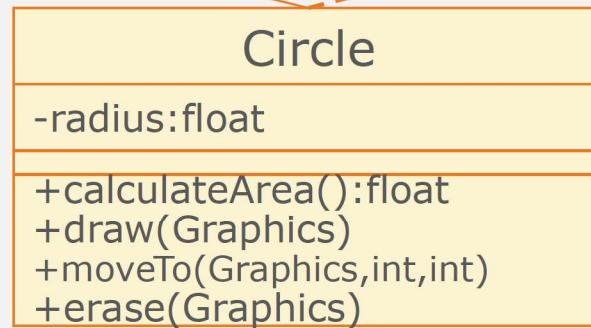
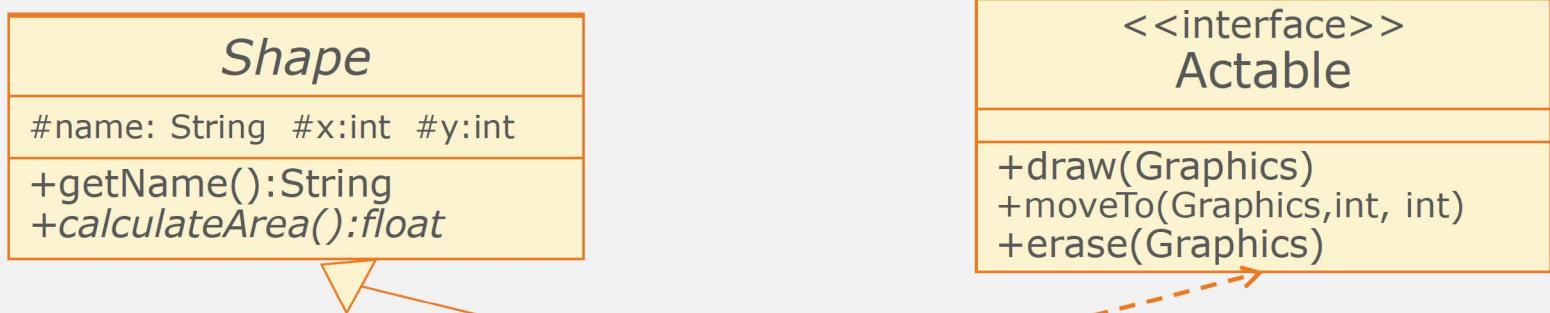
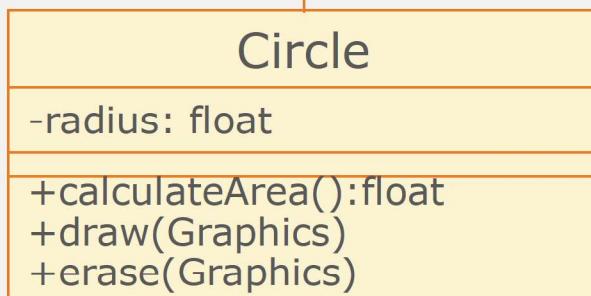
```
public class HinhVuong extends TuGiac
    implements DoiXung, DiChuyen { ...
}
```

# Interface



# Góc nhìn quan niệm

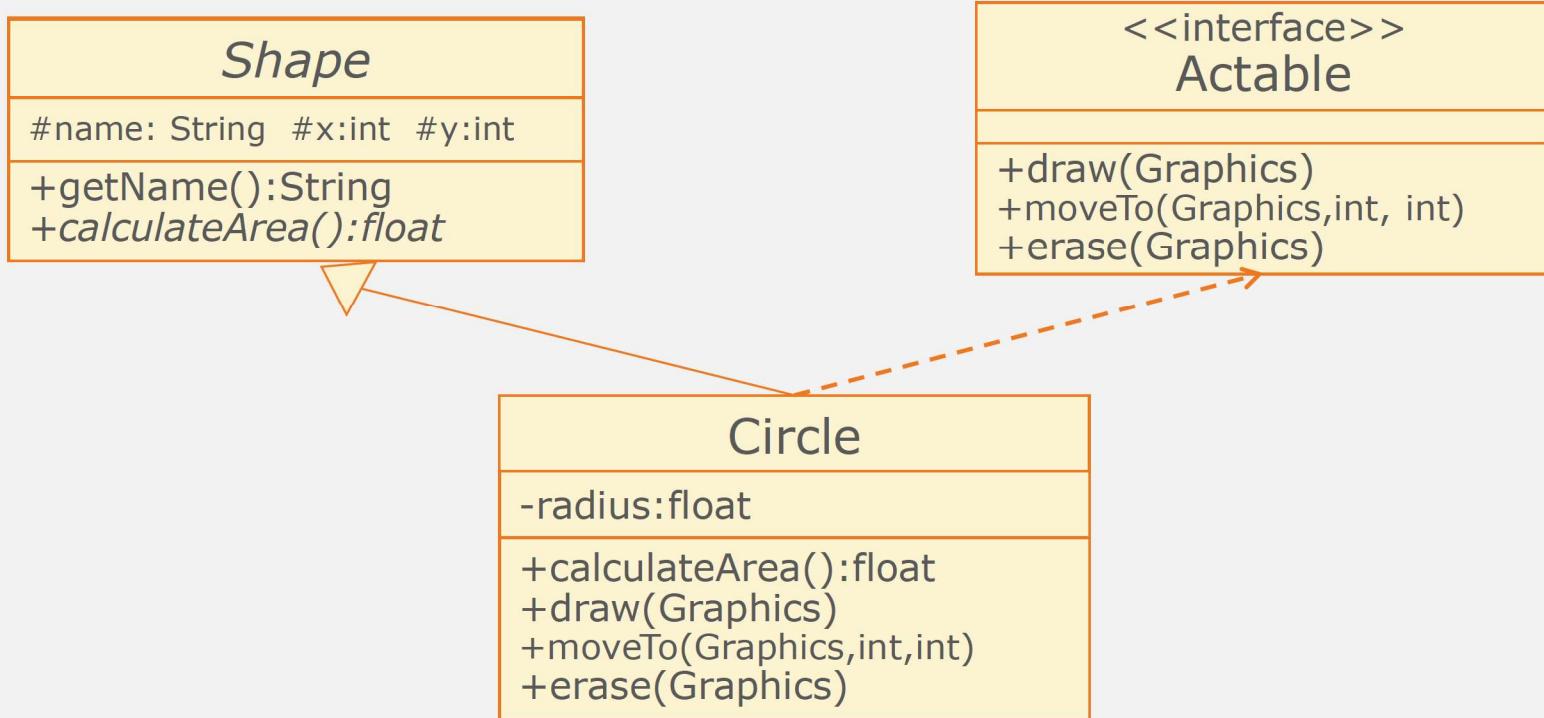
- Interface không cài đặt bất cứ một phương thức nào nhưng để lại cấu trúc thiết kế trên bất cứ lớp nào sử dụng nó
- Một interface: 1 contract – mà trong đó các nhóm phát triển phần mềm thống nhất sản phẩm của họ tương tác với nhau như thế nào, mà không đòi hỏi bất cứ một tri thức về cách thức tiến hành của nhau.



# Giao diện

- Một interface có thể được coi như một dạng “class” mà
  - Phương thức và thuộc tính là public không tường minh
  - Các thuộc tính là static và final
  - Các phương thức là abstract
- Không thể thể hiện hóa (instantiate) trực tiếp
- Một lớp có thể thực thi nhiều giao diện

# Ví dụ



# Ví dụ

```
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);
}
```

```
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...  
    }  
}
```

# Lớp trừu tượng vs. Giao diện

## *Lớp trừu tượng*

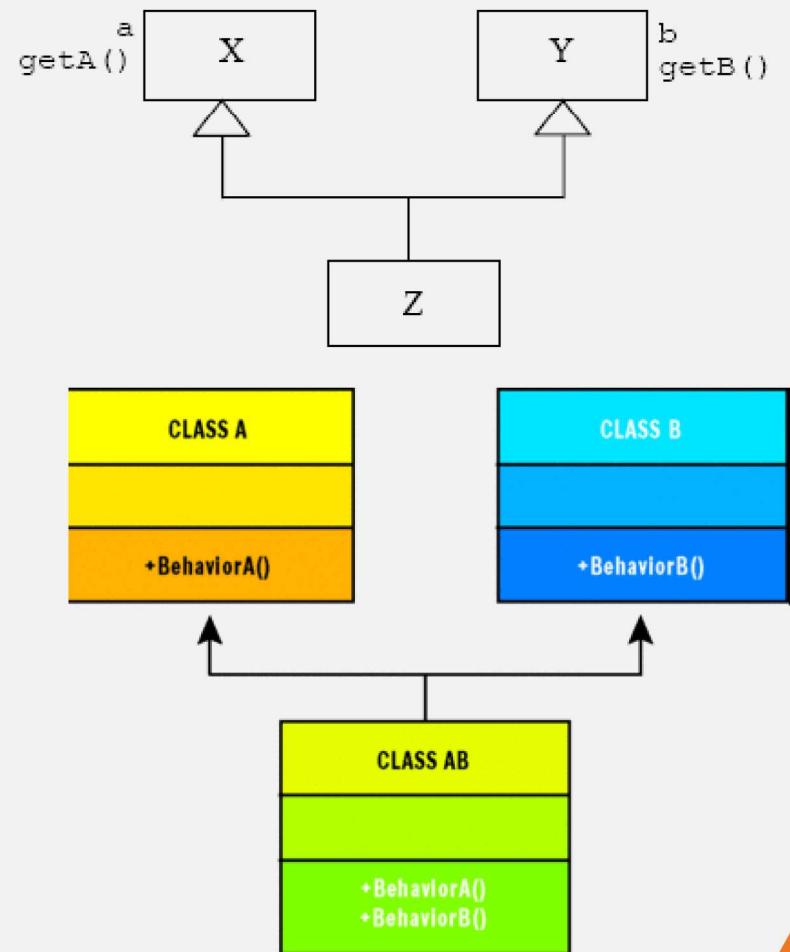
- Cần có ít nhất một phương thức abstract, có thể chứa các phương thức instance
- Có thể chứa các phương thức protected và static
- Có thể chứa các thuộc tính final và non-final
- Một lớp chỉ có thể kế thừa một lớp trừu tượng

## *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

# Nhược điểm

- 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



# Tổng kết

- **Ghi đè**
  - Các phương thức ở lớp con có cùng chữ ký và danh sách tham số với phương thức ở lớp cha, được tạo ra để định nghĩa lại các hành vi ở lớp con
- **Lớp trừu tượng**
  - Các lớp không được khởi tạo đối tượng, được tạo ra làm lớp cơ sở cho các lớp con định nghĩa rõ hơn
  - Có ít nhất một phương thức trừu tượng
- **Giao diện**
  - Định nghĩa các phương thức mà lớp thực thi phải cài đặt
  - Giải quyết vấn đề đa kế thừa

# Thank you!

Any questions?