BỘ MÔN CÔNG NGHỆ PHẦN MỀM
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
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.

Nội dung

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

Nội dung

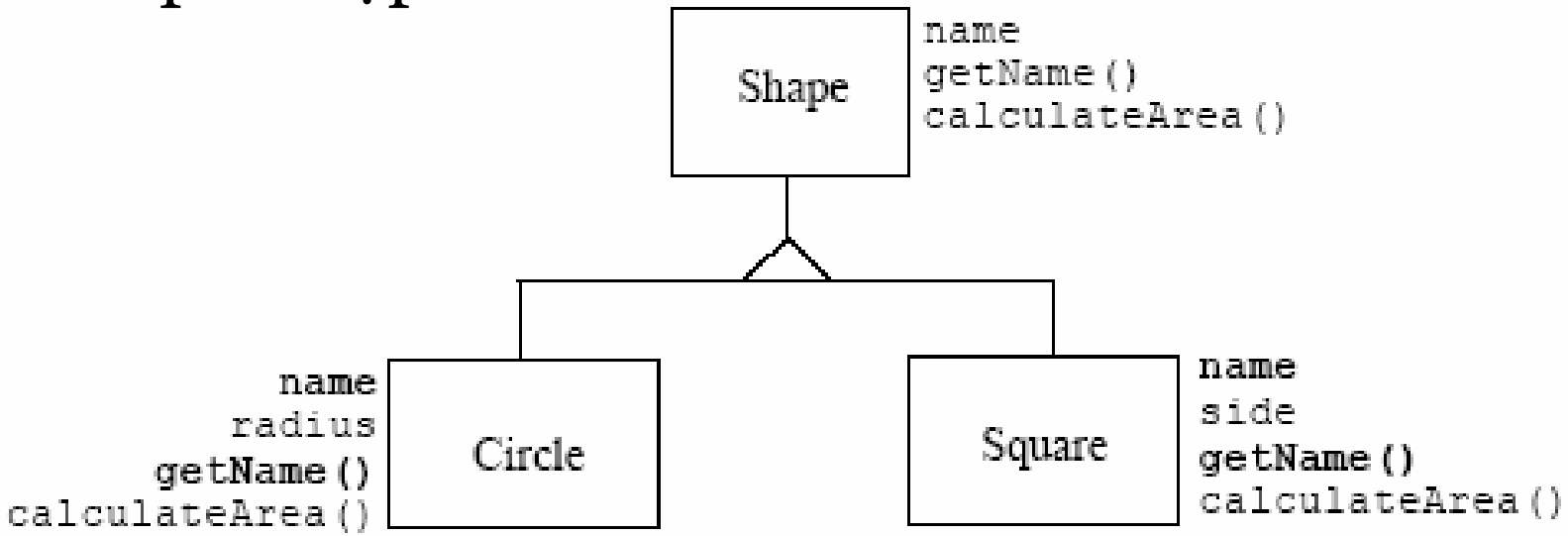
1. Định nghĩa lại (Redefine/Overiding)
2. Lớp trừu tượng (Abstract class)
3. Đơn kế thừa và đa kế thừa
4. Giao diện (Interface)

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)

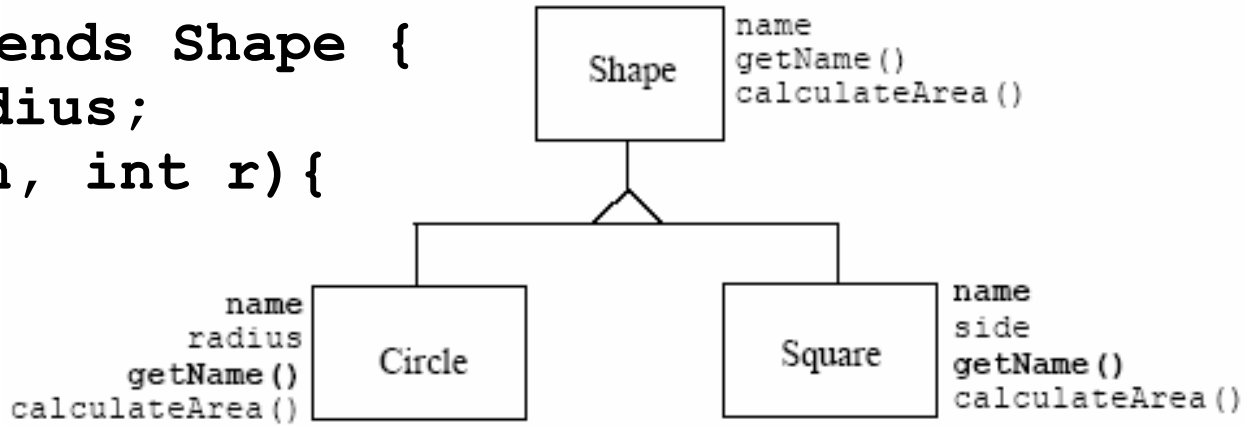
1. Định nghĩa lại hay ghi đè (2)

- 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

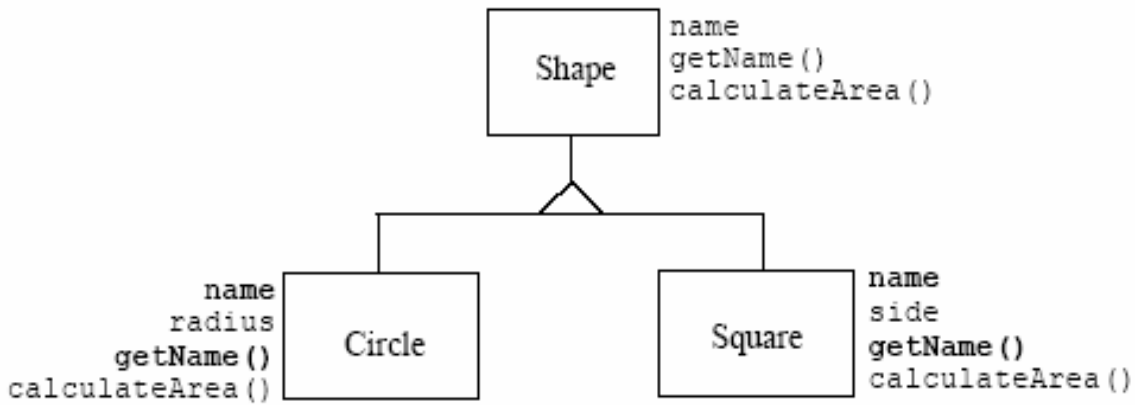


```
class Shape {
  protected String name;
  Shape(String n) { name = n; }
  public String getName() { return name; }
  public float calculateArea() { return 0.0f; }
}
class Circle extends Shape {
  private int radius;
  Circle(String n, int r){
      super(n);
      radius = r;
  }

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



```
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;
  }
}
```



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;
  }
}
```

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

```
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;
   }
}
```

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

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.

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

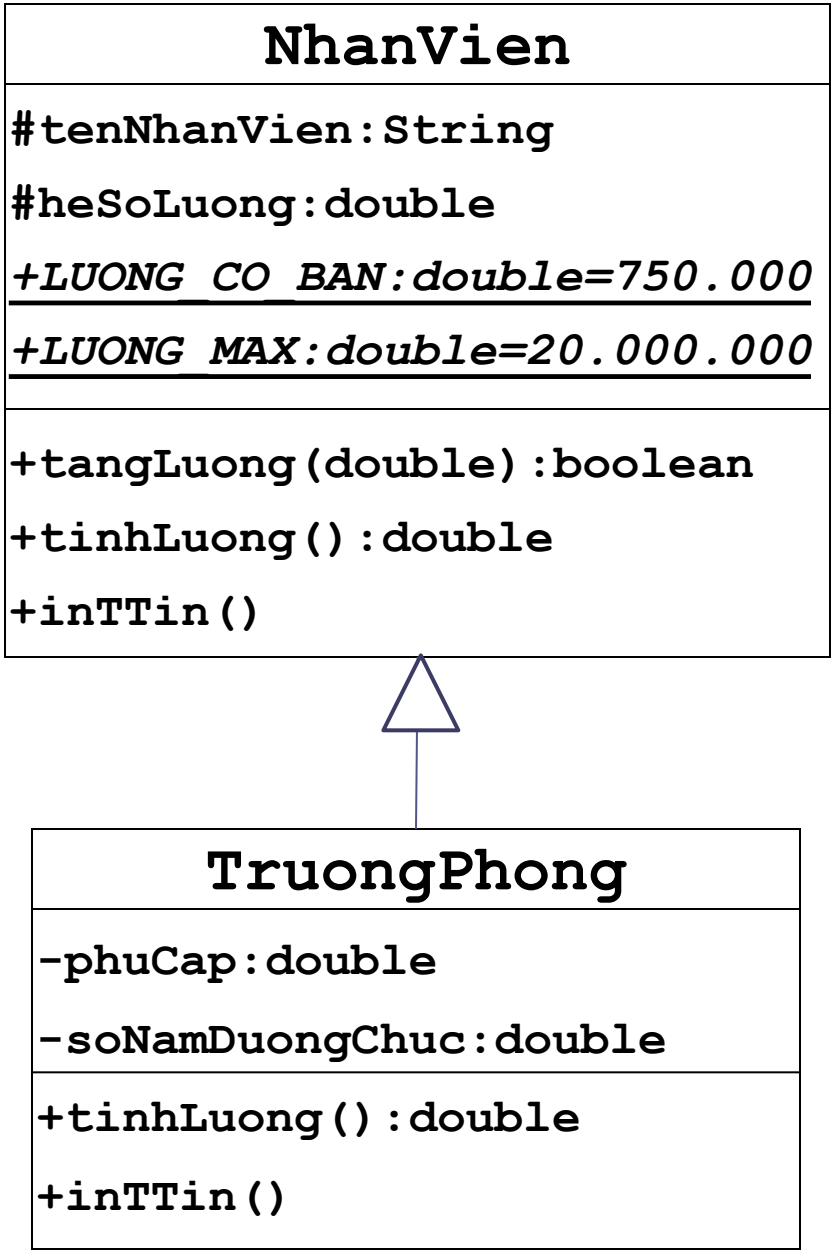
cannot override: attempting to assign weaker access privileges; was public

Ví dụ

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

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



Nội dung

1. Định nghĩa lại (Redefine/Overiding)
2. Lớp trừu tượng (Abstract class)
3. Đa kế thừa và đơn kế thừa
4. Giao diện (Interface)

2. Lớp trừu tượng (Abstract Class)

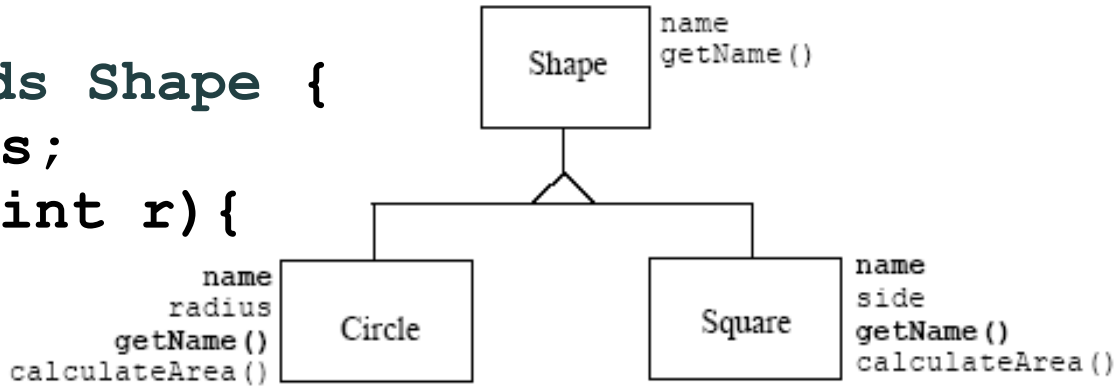
- Không thể thể hiện hóa (instantiate – tạo đối tượng của lớp) trực tiếp
- Chưa đầy đủ, thường được sử dụng làm lớp cha. Lớp con kế thừa nó sẽ hoàn thiện nốt.

2. Lớp trừu tượng (2)

- Để trở thành một lớp trừu tượng, cần:
  - Khai báo với từ khóa abstract
  - Chứa ít nhất một 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

```
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;
  }
}
```

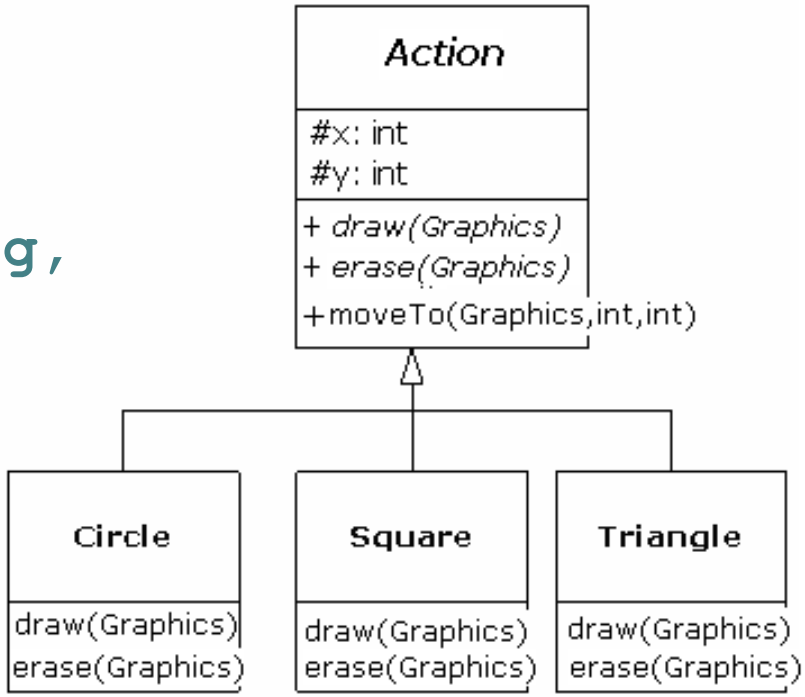


Lớp con bắt buộc phải override tất cả các phương thức abstract của lớp chả

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);
}
```



Ví dụ lớp trừu tượng (2)

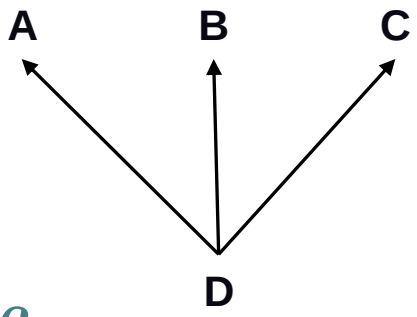
```
class Circle extends Action {
  int radius;
  public Circle(int x, int y, int r) {
    super(x, y); radius = r;
  }
  public void draw(Graphics g) {
    System out println("Draw circle at ("
                              + x + "," + y + ")");
    g.drawOval(x-radius, y-radius,
                  2*radius,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

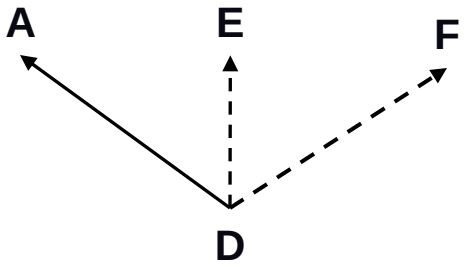
Nội dung

1. Định nghĩa lại (Redefine/Overiding)
2. Lớp trừu tượng (Abstract class)
3. Đa kế thừa và đơn kế thừa
4. Giao diện (Interface)

Đa kế thừa và đơn kế thừa

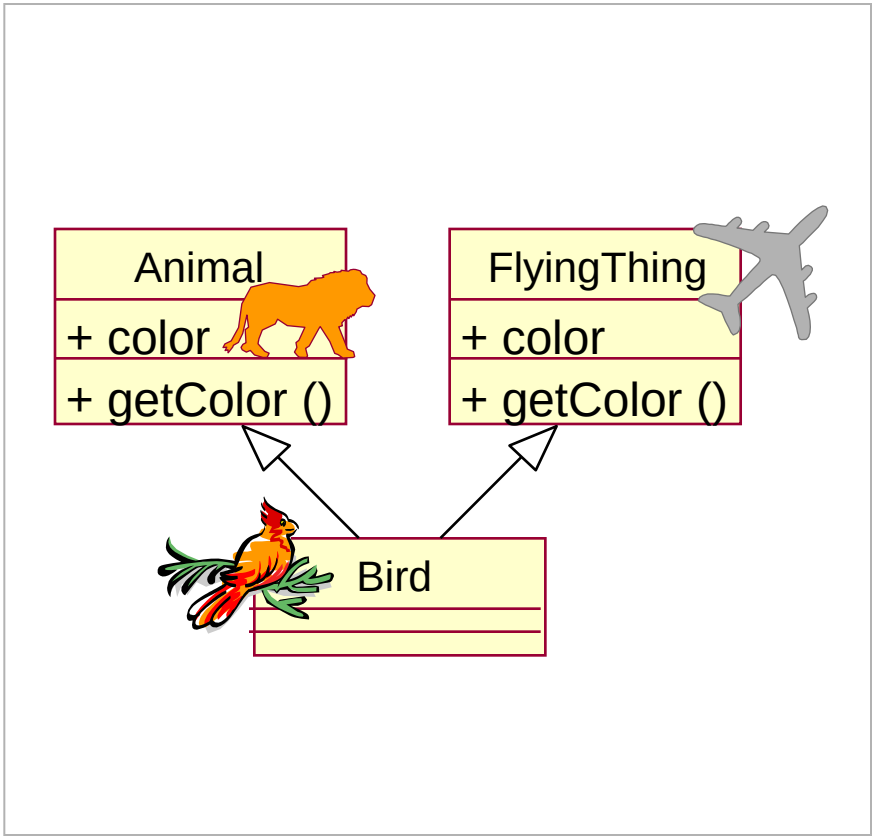
- Đa kế thừa (Multiple Inheritance)
  - Một lớp có thể kế thừa nhiều lớp khác
  - C++ hỗ trợ đa kế thừa
- Đơn kế thừa (Single Inheritance)
  - Một lớp chỉ được kế thừa từ một lớp khác
  - Java chỉ hỗ trợ đơn kế thừa
  - → Đưa thêm khái niệm Giao diện (Interface)



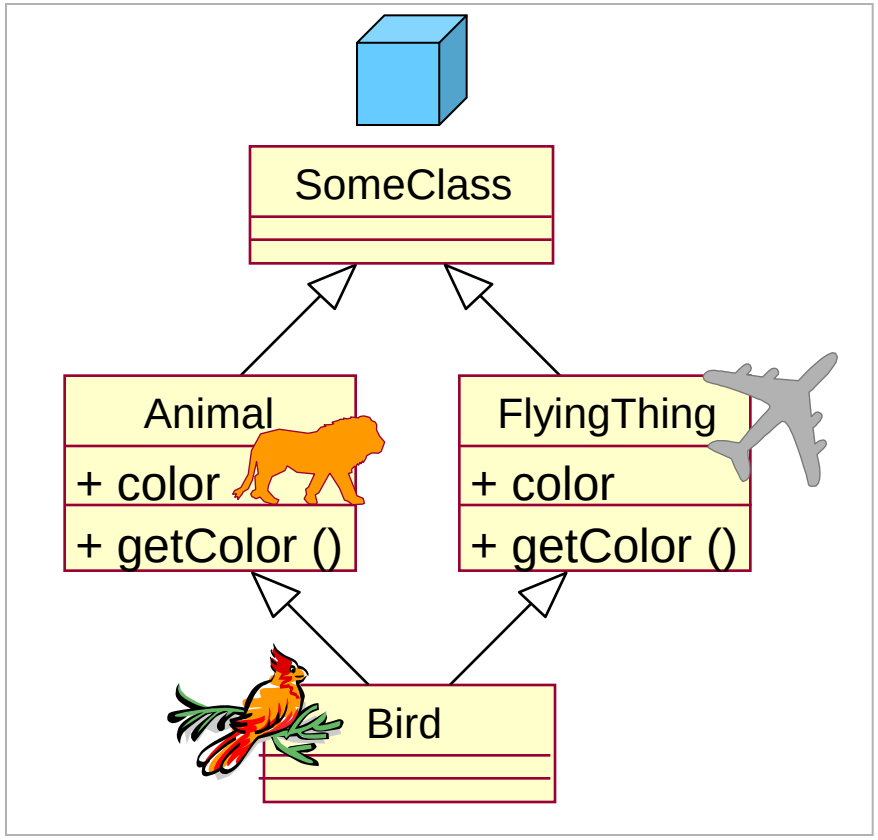


Vấn đề gặp phải trong Đa kế thừa

Name clashes on
attributes or operations



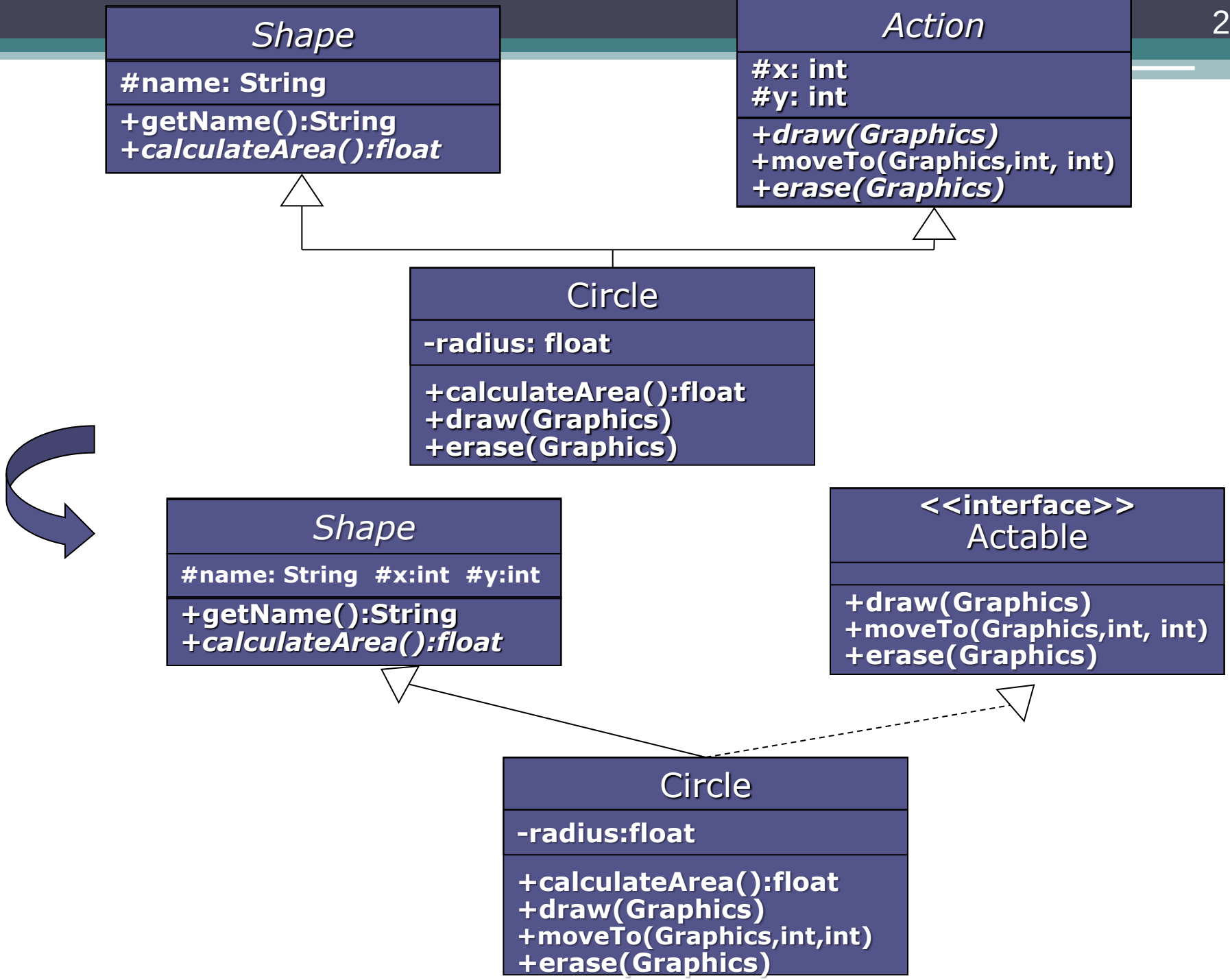
Repeated inheritance



Resolution of these problems is implementation-dependent.

Nội dung

1. Định nghĩa lại (Redefine/Overiding)
2. Lớp trừu tượng (Abstract class)
3. Đa kế thừa và đơn kế thừa
4. Giao diện (Interface)


Shape
#name: String
+getName():String
+calculateArea():float
Action
#x: int
#y: int
+draw(Graphics)
+moveTo(Graphics,int, int)
+erase(Graphics)
Circle
-radius: float
+calculateArea():float
+draw(Graphics)
+erase(Graphics)
Shape
#name: String #x:int #y:int
+getName():String
+calculateArea():float
<<interface>>
Actable
+draw(Graphics)
+moveTo(Graphics,int, int)
+erase(Graphics)
Circle
-radius:float
+calculateArea():float
+draw(Graphics)
+moveTo(Graphics,int,int)
+erase(Graphics)

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

4. Giao diện (2)

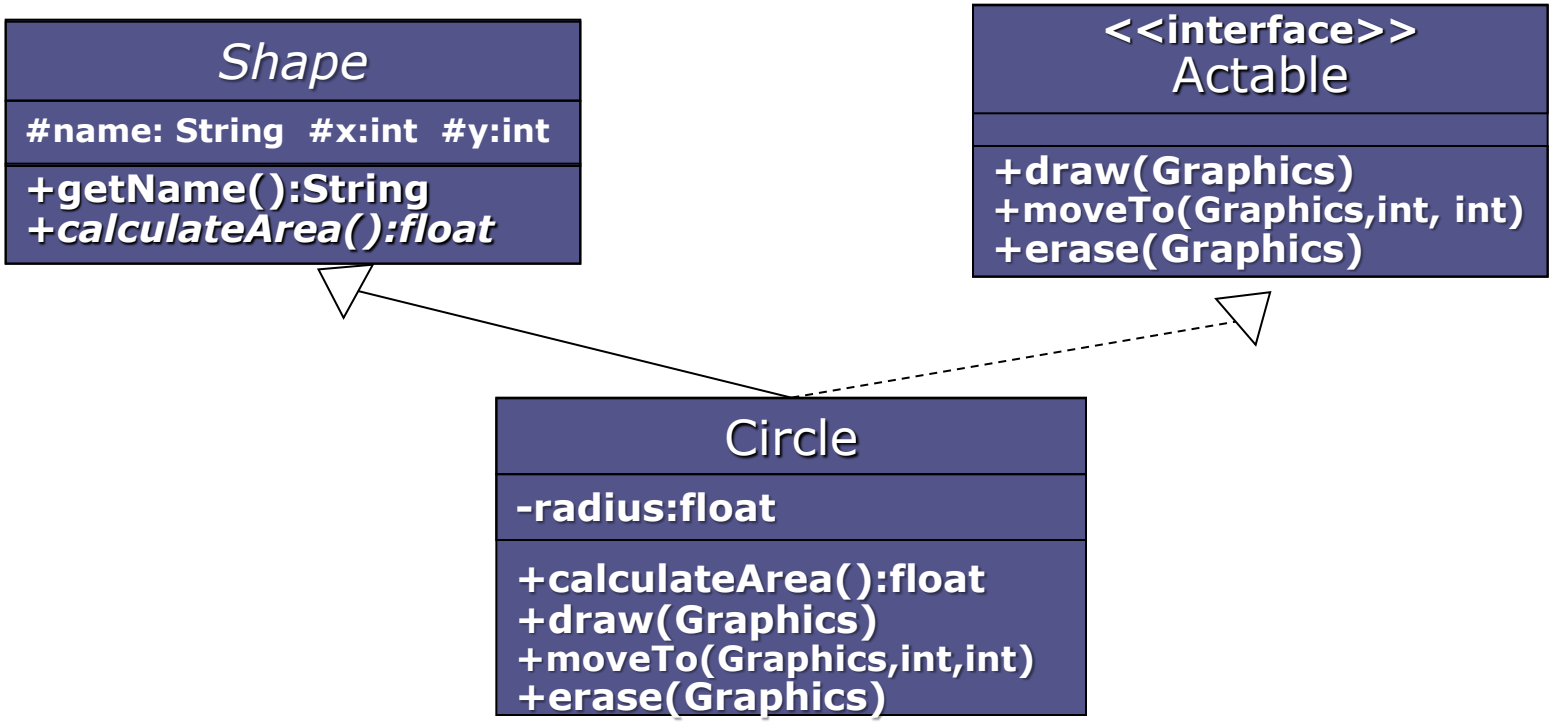
- Để trở thành giao diện, cần
  - Sử dụng từ khóa interface để định nghĩa
  - Chỉ được bao gồm:
    - Chữ ký các phương thức (method signature)
    - Các thuộc tính khai báo hằng (static & final)
- Lớp thực thi giao diện
  - Hoặc là lớp trừu tượng (abstract class)
  - Hoặc là bắt buộc phải cài đặt chi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

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

Ví dụ


Shape
#name: String #x:int #y:int
+getName():String
+calculateArea():float
<<interface>>
Actable
+draw(Graphics)
+moveTo(Graphics,int, int)
+erase(Graphics)
Circle
-radius:float
+calculateArea():float
+draw(Graphics)
+moveTo(Graphics,int,int)
+erase(Graphics)

```
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 trượng vs. Giao diện

- 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

- 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 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