ĐẠI HỌC BÁCH KHOA HÀ NỘI
VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG

IT3100

LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Bài 06. Một số kỹ thuật trong kế thừa

Mục tiêu bài học

- Trình bày nguyên lý ghi đè 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. Ghi đè (Redefine/Overriding)
2. Lớp trừu tượng (Abstract class)
3. Đa kế thừa và Giao diện (Interface)

1/ GHI ĐÈ (OVERRIDING)

1.1 Khái niệm ghi đè

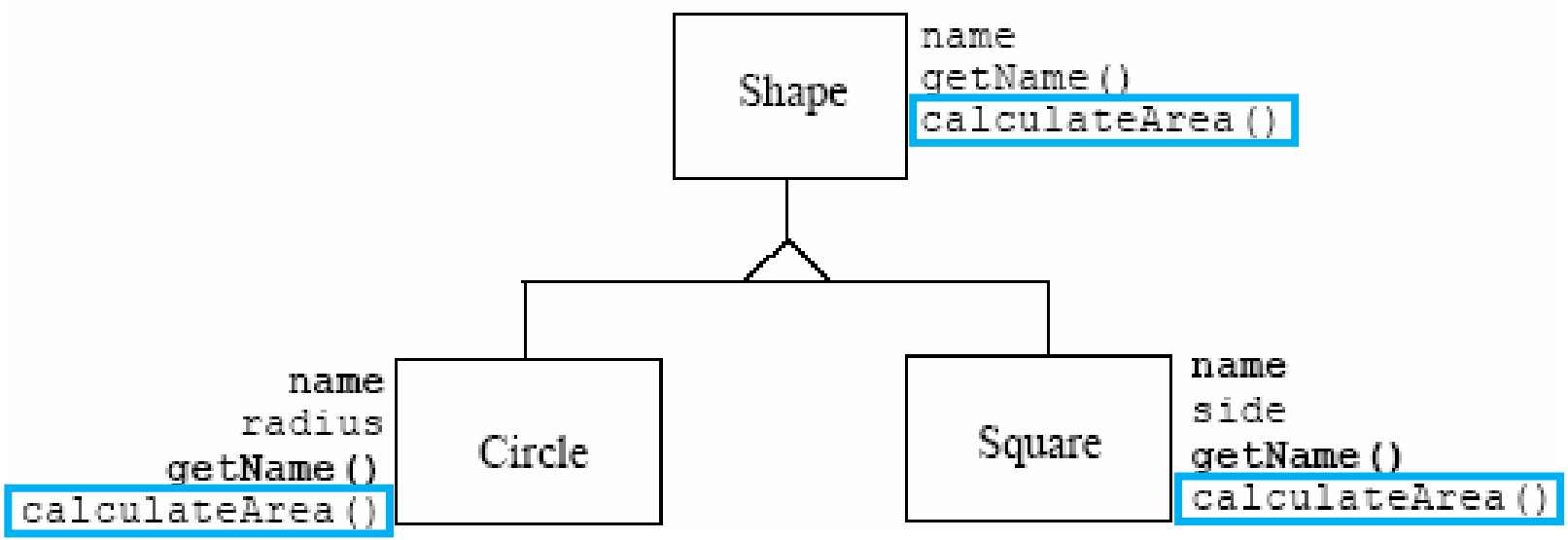
- Lớp con có thể định nghĩa phương thức trùng tên với phương thức trong lớp cha:
  - Nếu phương thức mới chỉ trùng tên và khác chữ ký (số lượng hay kiểu dữ liệu của đối số)

    → Chồng phương thức (Method Overloading) (Bài 04)
  - Nếu phương thức mới hoàn toàn giống về giao diện (chữ ký)

    → Định nghĩa lại hoặc ghi đè (Method Redefine/Override)

1.1 Khái niệm 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 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;
    }
}

public class Demo {
 public static void main(String[] args) {
  Circle c = new Circle("Hinh tron",10);
  Square s = new Square("Hinh vuong",15);

  System.out.println("Dien tich hinh tron: " + c.calculateArea());
  System.out.println("Dien tich hinh vuong: " + s.calculateArea());
}
```

1.2 `this` và `super`

- `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
- `this` và `super` có thể sử dụng cho các phương thức/thuộc tính non-static và phương thức khởi tạo

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;
    }
}
```

1.3 Từ khóa `instanceof`

- `instanceof`: kiểm tra đối tượng có phải là thể hiện của một lớp nào đó hay không. Trả về `true/false`

```
class A {

}
class B extends A {

}

public class Demo {
    public static void main(String[] args) {
        B b = new B();
        System.out.println("b is an instance of B: " + (b instanceof B));
        System.out.println("b is an instance of A: " + (b instanceof A));
    }
}
```

1.4 Ghi đè PT `equals()`

```
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));
    }
}
```

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

1.5 Một số quy định khi ghi đè

- Phương thức ghi đè trong lớp con
  - phải 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
- 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() {
  private
   return 0;
  }
}
class Child extends Parent {
  protected void doSomething() {}
  public
  protected void doSomething2() {}
}
```

attempting to use incompatible return type

attempting to assign weaker access privileges

2/ LỚP TRỪU TƯỢNG

2.1 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.
  - Có thể chứa các phương thức trừu tượng (*abstract method*)*:* chỉ có chữ ký mà không có cài đặt cụ thể
  - 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`.

2.2. Cú pháp

```
abstract class lớp_Cha {
    public abstract kieu ten_PT_TT();
    ...
}
```

- Lớp con bắt buộc phải ghi đè tất cả các phương thức trừu tượng của lớp cha
- Nếu một lớp có một hay nhiều phương thức trừu tượng thì nó phải là lớp trừu tượng

Ví dụ

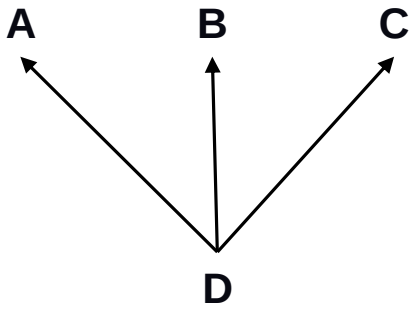
```
abstract class Shape {
  protected String name;
  Shape(String n) { name = n; }
  public String getName() { return name; }
  public abstract float calculateArea();
}
class Circle extends Shape {
  private int radius;
  Circle(String n, int r){
    super(n);
    radius = r;
  }
  public float calculateArea()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

3/ ĐA KẾT THỪA VÀ GIAO DIỆN

3.1. Đa kế thừa

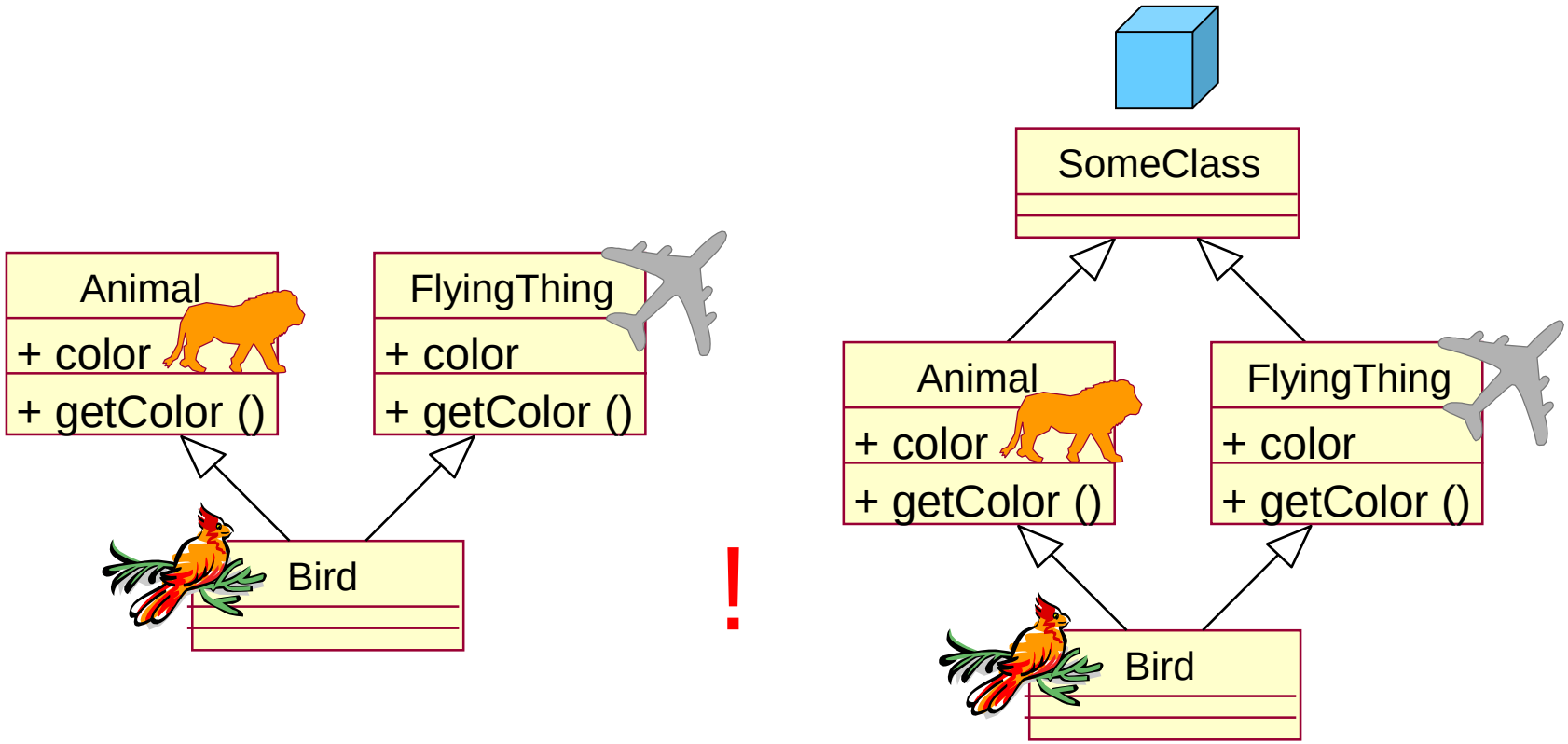
- Đơn kế thừa (Single Inheritance)
  - Một lớp chỉ được kế thừa từ một lớp khác
- Đa kế thừa (Multiple Inheritance)
  - Một lớp có thể kế thừa nhiều lớp khác
- Java chỉ hỗ trợ đơn kế thừa

=> Đưa thêm khái niệm Giao diện (Interface)



3.2. Vấn đề trong Đa kết thừa

- Xung đột tên
- Thừa kế bị lặp (diamond problem)



3.3 Giao diện (Interface)

- Interface: đặc tả cho các cách cài đặt/triển khai.
- Interface định nghĩa một "kiểu" chỉ chứa định nghĩa hằng và phương thức trừu tượng
  - Không thể thể hiện hóa (instantiate) trực tiếp
- 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
- Dùng để cho phép một lớp có thể "kế thừa" (triển khai, thực thi) nhiều cấu trúc một lúc
- Interface có thể kế thừa từ nhiều interface khác

3.3 Giao diện (2)

- Cú pháp giao diện
  - 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 thực.

3.4. Thực thi giao diện

Kế thừa trước rồi mở rộng !

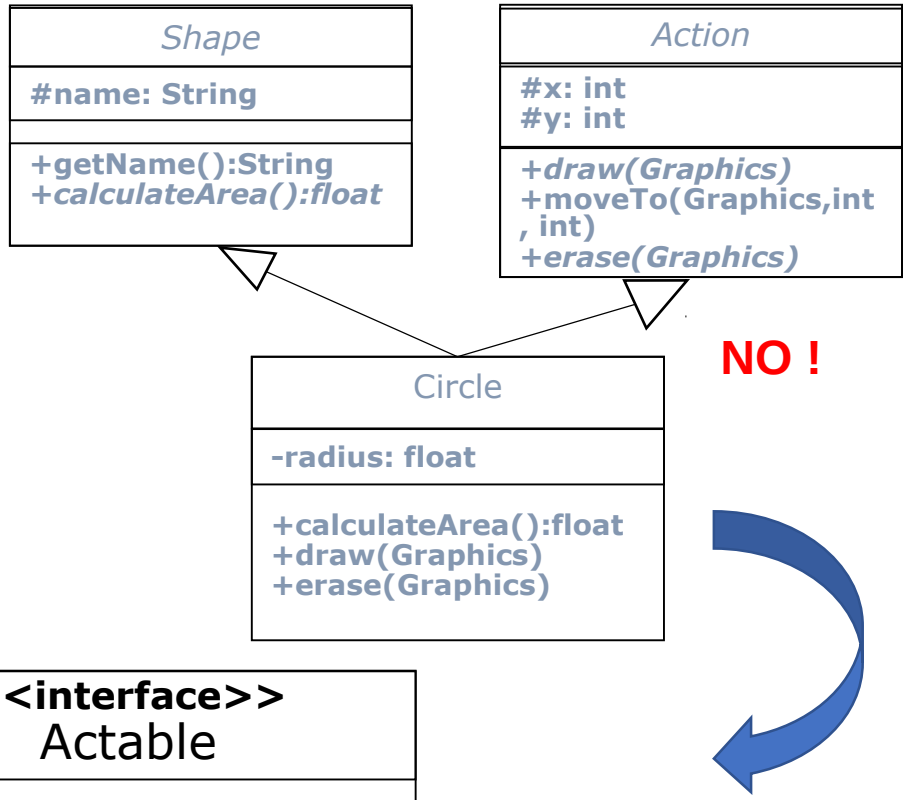
- Cú pháp thực thi trên Java:
  - <Lớp con> [***extends*** <Lớp cha>] ***implements*** <Danh sách giao diện>
  - <Giao diện con> ***extends*** <Giao diện cha>
- Ví dụ:

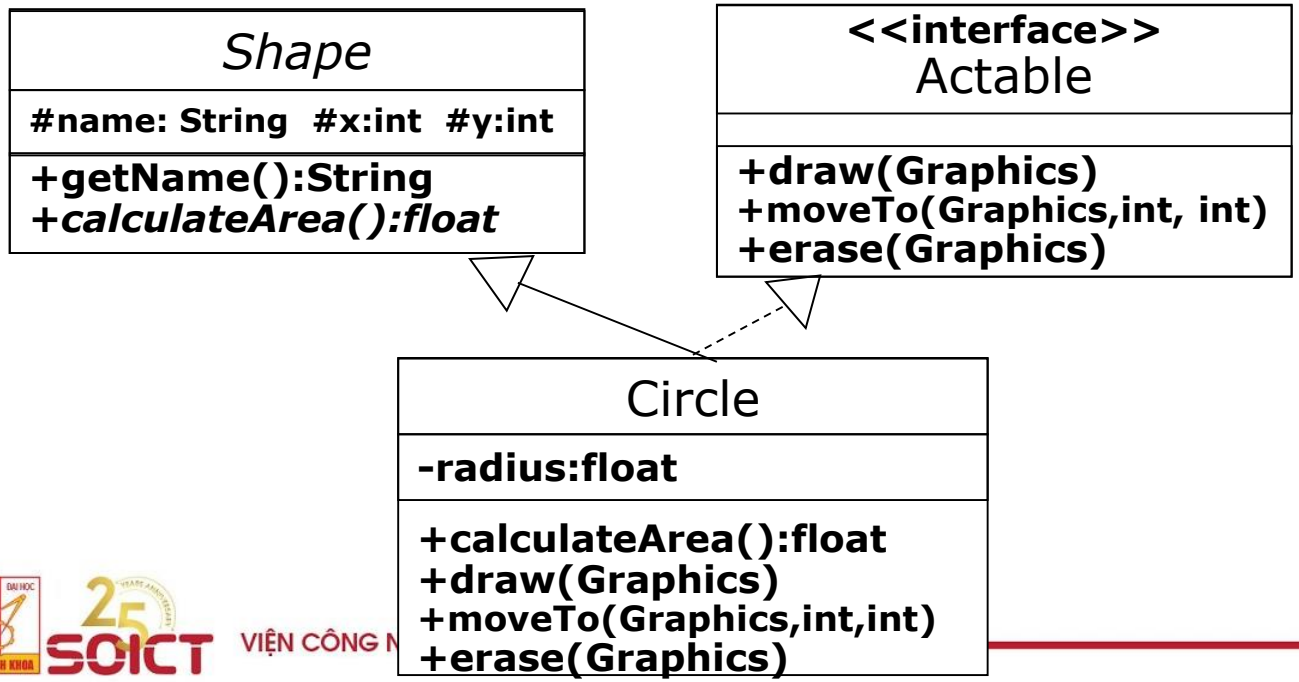
```
public interface DoiXung {…}
public interface DiChuyen {…}
public class HinhVuong extends TuGiac
                    implements DoiXung, DiChuyen {
    ...
}
```

3.4. Thực thi giao diện

- Nếu GD có các thuộc tính thì chúng đều là `public static final`.
- Các phương thức của GD đều là trừu tượng, nghĩa là không có thân hàm, và đều có modifier là `public abstract`.
- Interface không có Constructor.

Ví dụ


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)
NO !


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)

Ví dụ - Code

```
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...
  }
}
```

3.4 Lớp trừu tượng vs. Giao diện

Lớp trừu trượng

- Có thể có các phương thức abstract, có thể chứa các phương thức thực
- 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

3.4 Lớp trừu tượng vs. Giao diện

Lớp trừu trượng

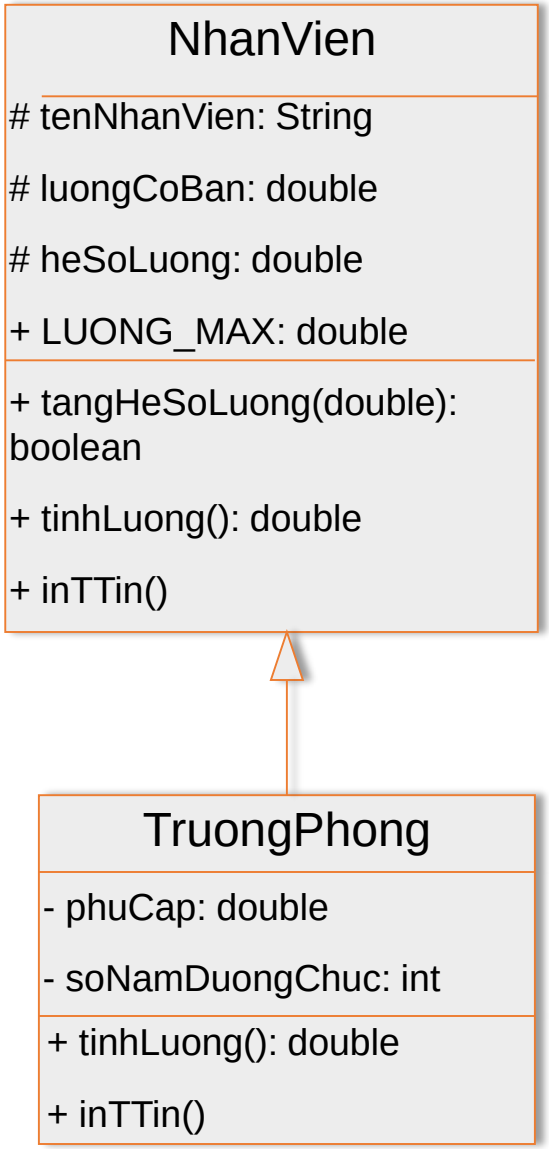
- Có thể chứa PT không trừu tượng, với cài đặt mặc định cho các lớp con kế thừa
- Có thể thêm PT không trừu tượng vào lớp mà không yêu cầu thay đổi các lớp con kế thừa

Giao diện

- Nếu Giao diện chứa nhiều phương thức, phải cài đặt lại PT ở tất cả các lớp thực thi GD
- Nếu thêm một PT vào trong GD, phải cài đặt lại tất cả các lớp thực thi GD đó
- GD không đáp ứng được tính tái sử dụng mã nguồn

Bài tập

- **Bài 1**: Sửa lại lớp NhanVien:
  - Viết mã nguồn của lớp TruongPhong như hình vẽ biết Lương của trưởng phòng = Lương Cơ bản * hệ số lương + phụ cấp



Bài tập

- **Bài 2:** Triển khai xây dựng các lớp, giao diện theo mô tả UML sau (nội dung các PT là in ra 1 dòng chữ chỉ tên PT)