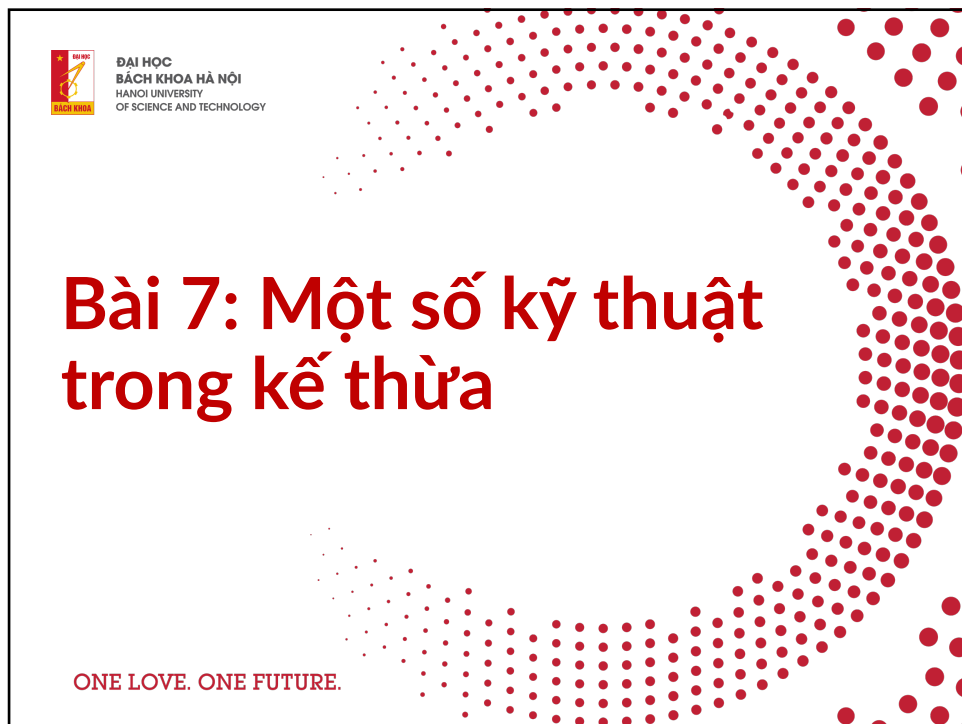




1



2

## Mục tiêu

- Trình bày nguyên lý định nghĩa lại trong kế thừa
- Phân biệt khái niệm đơn kế thừa và đa kế thừa
- Giới thiệu về giao diện, lớp trừu tượng và vai trò của chúng
- Ví dụ và bài tập về các vấn đề trên với ngôn ngữ lập trình Java



## Nội dung

1. Định nghĩa lại/ghi đè (Overriding)
2. Lớp trừu tượng
3. Đơn kế thừa & Đa kế thừa
4. Giao diện (Interface)
5. Vai trò của lớp trừu tượng và giao diện
6. Ví dụ và bài tập



## Nội dung

1. Định nghĩa lại/ghi đè (Overriding)
2. Lớp trừu tượng
3. Đơn kế thừa & Đa kế thừa
4. Giao diện (Interface)
5. Vai trò của lớp trừu tượng và giao diện
6. Ví dụ và bài tập



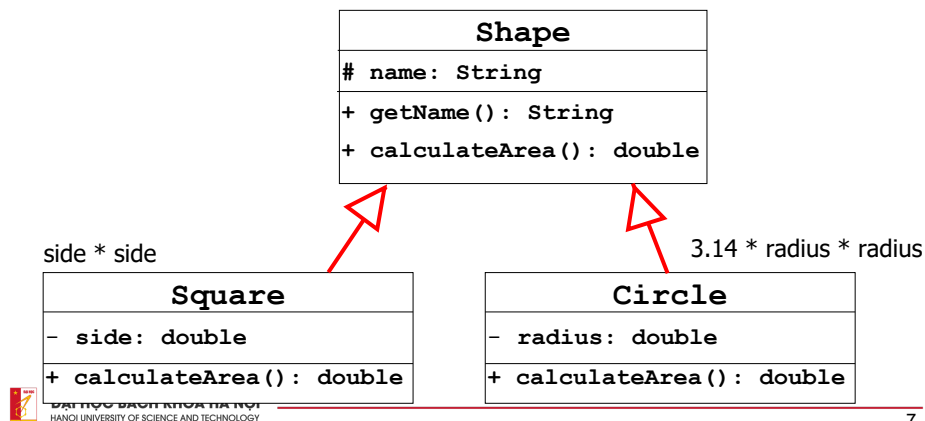
## 1. Định nghĩa lại/ghi đè (Overriding)

- Quan hệ kế thừa (inheritance)
  - 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:
    - Mở rộng lớp cha (Extension): Thêm các thuộc tính/hành vi mới
    - Định nghĩa lại (Redefinition): Chính sửa lại các hành vi kế thừa từ lớp cha → Ghi đè (Method Overriding)



## 1. Định nghĩa lại/ghi đè (Overriding)

- 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



7

7

## 1. Định nghĩa lại/ghi đè (Overriding)

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

8

8

## Ví dụ (1)

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




ĐẠI HỌC BÁCH KHOA HÀ NỘI  
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

9

9

## Ví dụ (2)

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



ĐẠI HỌC BÁCH KHOA HÀ NỘI  
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

10

10

## Thêm lớp Triangle

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

Muốn gọi lại các phương thức của lớp cha đã bị ghi đè ?

11

## Ví dụ sử dụng từ khóa super

```
public class Person {  
    protected String name;  
    protected int age;  
  
    public String getDetail() {  
        String s = this.name + "," + this.age;  
        return s;  
    }  
}  
  
public class Employee extends Person {  
    double salary;  
  
    public String getDetail() {  
        String s = super.getDetail() + "," + this.salary;  
        return s;  
    }  
}
```

12

## Sử dụng từ khóa super

- 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.tên_Phương_thức(danh sách tham số);`

13

## 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
  - Có 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`

14

## 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

## Quy định trong ghi đè

- 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 – Từ khóa final

- 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 lệch 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. Ví dụ:

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

## Hạn chế ghi đè – Từ khóa final

- Các phương thức được khai báo là final không thể ghi đè

```
class A {  
    final void method() { }  
}  
class B extends A {  
    void method() { // Báo lỗi!!!  
    }  
}
```

## Hạn chế ghi đè – Từ khoá final

- Từ khoá **final** được dùng khi khai báo lớp:
  - Lớp được khai báo là lớp hằng (không thay đổi), lớp này không có lớp con thừa kế
  - Được sử dụng để hạn chế việc thừa kế và ngăn chặn việc sửa đổi một lớp

```
public final class A {  
    //...  
}
```

## 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 khoá **super** cho phép tái sử dụng các đoạn mã của lớp cha trong lớp con

## Bài tập 1

- Cho đoạn mã dưới đây:

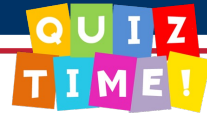
```
1. class BaseClass {
2.     private float x = 1.0f;
3.     float getVar() { return x; }
4. }
5. class SubClass extends BaseClass {
6.     private float x = 2.0f;
7.     // insert code here
8. }
```

- Lựa chọn nào có thể chèn tại dòng 7?

```
1. public double getVar() { return x; }
2. public float getVar(float f){ return f; }
3. float getVar() { return x; }
4. public float getVar() { return x; }
5. private float getVar() { return x; }
```



## Bài tập 2



- Cho đoạn mã dưới đây:

```
1. class Super {
2.     public String getName() { return "Super"; }
3. }
4. class Sub extends Super {
5.
6. }
```

- Lựa chọn nào khi đặt vào dòng 5 trong đoạn mã trên gây ra lỗi biên dịch?

```
1. public String getTen () { }
2. public void getName(String str) { }
3. public String getName() {return "Sub"; }
4. public void getName() {}
```



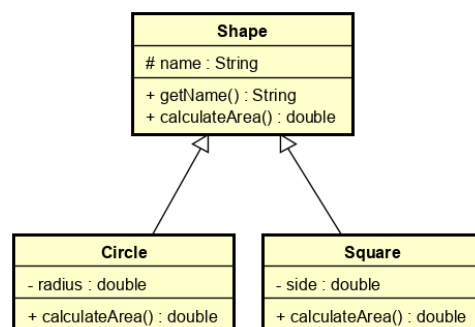
## Nội dung

1. Định nghĩa lại/ghi đè (Overriding)
2. **Lớp trừu tượng**
3. Đơn kế thừa & Đa kế thừa
4. Giao diện (Interface)
5. Vai trò của lớp trừu tượng và giao diện
6. Ví dụ và bài tập

23

## 2. Lớp trừu tượng

- Các ngôn ngữ lập trình hướng đối tượng cung cấp các cơ chế kiểu trừu tượng (abstract type)
  - Các kiểu trừu tượng có cài đặt không đầy đủ hoặc không có cài đặt
  - Nhiệm vụ chính của chúng là giữ vai trò kiểu tổng quát hơn của một số các kiểu khác
- Xét ví dụ: lớp Shape
  - Là một lớp "không rõ ràng", khó hình dung ra các đối tượng cụ thể
    - Không thể thể hiện hóa (instantiate – tạo đối tượng của lớp) trực tiếp



24

## 2. Lớp trừu tượng

- Đặc điểm của 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 (base class) cho các lớp "cụ thể" khác (concrete class)
  - Chưa đầy đủ, thường được sử dụng làm lớp cha. Lớp con kế thừa nó sẽ hoàn thiện nốt.
    - Lớp trừu tượng thường chứa các *phương thức trừu tượng* (phương thức không được cài đặt)



## 2. Lớp trừu tượng

- Phương thức trừu tượng
  - Là các phương thức "không rõ ràng" / chưa hoàn thiện, khó đưa ra cách cài đặt cụ thể
  - **Chỉ có chữ ký** mà không có cài đặt cụ thể
  - Các lớp dẫn xuất có thể làm rõ - định nghĩa lại (overriding) các phương thức trừu tượng này



## Từ khoá abstract

- Lớp trừu tượng
  - Khai báo với từ khóa `abstract`

```
public abstract class Shape {  
    // Nội dung lớp  
}
```
- Phương thức trừu tượng
  - Khai báo với từ khóa `abstract`

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

```
Shape = new Shape(); //Compile error
```

## Ví dụ Lớp trừu tượng

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

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

## 2. Lớp trừu tượng

- 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
  - Nếu không ghi đè các phương thức này thì lớp con cũng trở thành một lớp trừu tượng → Phương thức trừu tượng không thể khai báo là **final** hoặc **static**

**Kết hợp cho phép**  
**abstract public**  
**abstract protected**

**Kết hợp KHÔNG cho phép**  
**abstract private**  
**abstract static**  
**abstract final**



ĐẠI HỌC BÁCH KHOA HÀ NỘI  
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

29

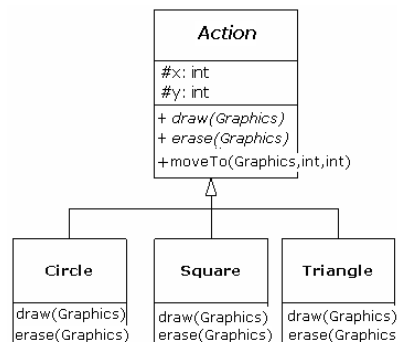
29

## 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);  
    }  
}
```

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

```
..Circle c = new Circle();  
c.moveTo(...);
```



ĐẠI HỌC BÁCH KHOA HÀ NỘI  
HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

30

## Ví dụ lớp trừu tượng

```
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...
    }
}
```



## Ví dụ lớp trừu tượng

```
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 Lớp trừu tượng

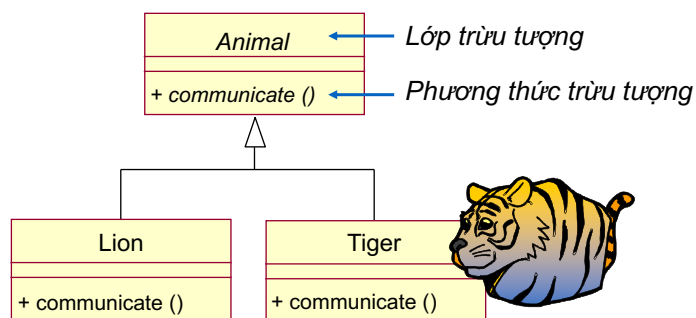
```
abstract class ColoredPoint extends Point {  
    int color;  
    public ColoredPoint(int x, int y, int color) {  
        super(x, y); this.color = color;  
    }  
}  
  
class SimpleColoredPoint extends ColoredPoint {  
    public SimpleColoredPoint(int x, int y, int color) {  
        super(x,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



33

## 2. Lớp trừu tượng

- Biểu diễn trong UML
  - Lớp trừu tượng (không thể tạo đối tượng cụ thể)
    - Chứa phương thức trừu tượng
    - Tên lớp / tên phương thức: Chữ nghiêng



Tất cả các đối tượng là sư tử hoặc hổ



34

## Bài tập 3

- 1. Đoạn mã dưới đây có lỗi gì không?

```
abstract class ABC {  
    void firstMethod() {  
        System.out.println("First Method");  
    }  
    void secondMethod() {  
        System.out.println("Second Method");  
    }  
}
```

- 2. Lớp nào là lớp trừu tượng, lớp nào có thể tạo đối tượng?

```
abstract class A { }  
  
class B extends A {
```



## Nội dung

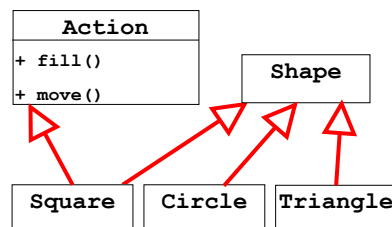
1. Định nghĩa lại/ghi đè (Overriding)
2. Lớp trừu tượng
3. **Đơn kế thừa & Đa kế thừa**
4. Giao diện (Interface)
5. Vai trò của lớp trừu tượng và giao diện
6. Ví dụ và bài tập



### 3. Đơn kế thừa & Đa kế thừa

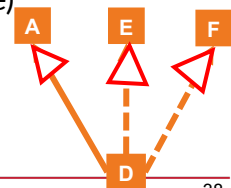
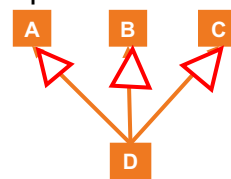
- Giả sử trong bài toán các lớp đối tượng Hình học, lớp Square cần thiết kế bổ sung thêm những hành vi mới Fill (tô màu), Move (di chuyển) mà chỉ có các đối tượng của nó sử dụng
  - Giải pháp 1: thêm các hành vi này vào lớp cha Shape → ảnh hưởng đến các đối tượng của lớp con Circle và Triangle (các đối tượng này không sử dụng đến các hành vi trên)
  - Giải pháp 2: đặt các hành vi này trực tiếp tại lớp Square → tương lai có thể có thêm lớp mới Hình thang cũng sử dụng các hành vi trên → cần phải cài đặt lại, không tái sử dụng

→ Tách các hành vi trên thành một lớp cha riêng, rồi cho lớp Square kế thừa cả HAI lớp cha trong cây thừa kế?



### 3. Đơn kế thừa & Đa kế thừa

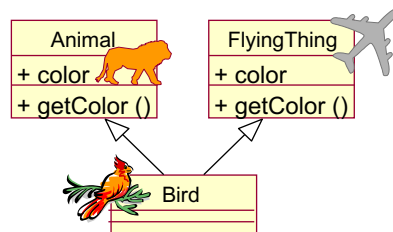
- Đa kế thừa (Multiple Inheritance)
  - Một lớp có thể kế thừa nhiều lớp cha trực tiếp
  - C++ hỗ trợ đa kế thừa
- Đơn kế thừa (Single Inheritance)
  - Một lớp chỉ được kế thừa từ một lớp cha trực tiếp
  - Java chỉ hỗ trợ đơn kế thừa
  - Đưa thêm khái niệm Giao diện (Interface)



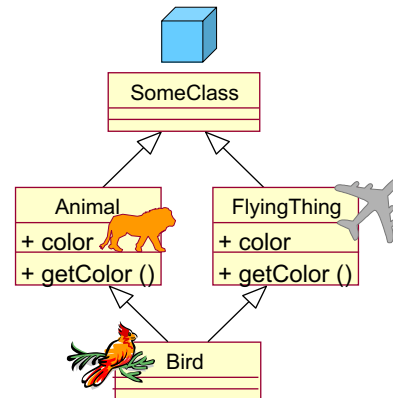
### 3. Đơn kế thừa & Đa kế thừa

- Vấn đề gặp phải trong đa kế thừa

- Name collision



- "Diamond shape" problem



### Nội dung

1. Định nghĩa lại/ghi đè (Overriding)
2. Lớp trừu tượng
3. Đơn kế thừa & Đa kế thừa
4. **Giao diện (Interface)**
5. Vai trò của lớp trừu tượng và giao diện
6. Ví dụ và bài tập

## 4. Giao diện

- Giao diện (interface)
  - Là kiểu dữ liệu trừu tượng, được dùng để đặc tả các hành vi mà các lớp phải thực thi
  - Tương tự như giao thức (protocols)
  - Chứa các chữ ký phương thức (Mọi phương thức đều là phương thức trừu tượng) và các hằng
  - Giải quyết bài toán đa thừa kế, tránh các rắc rối nhập nhằng ngữ nghĩa
- Giao diện trong Java
  - Một cấu trúc lập trình của Java được định nghĩa với từ khóa **interface**
  - Từ Java 8: có thêm phương thức default, static. Từ Java 9, có thêm phương thức private và private static



## 4. Giao diện

- Sử dụng từ khóa **interface** để định nghĩa
  - Một giao diện chỉ được bao gồm:
    - Chữ ký các phương thức (method signature)
    - Các thuộc tính khai báo hằng (static & final)
  - Không có thể hiện
  - Chỉ được thực thi và mở rộng
- 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 {...}
```



## 4. Giao diện

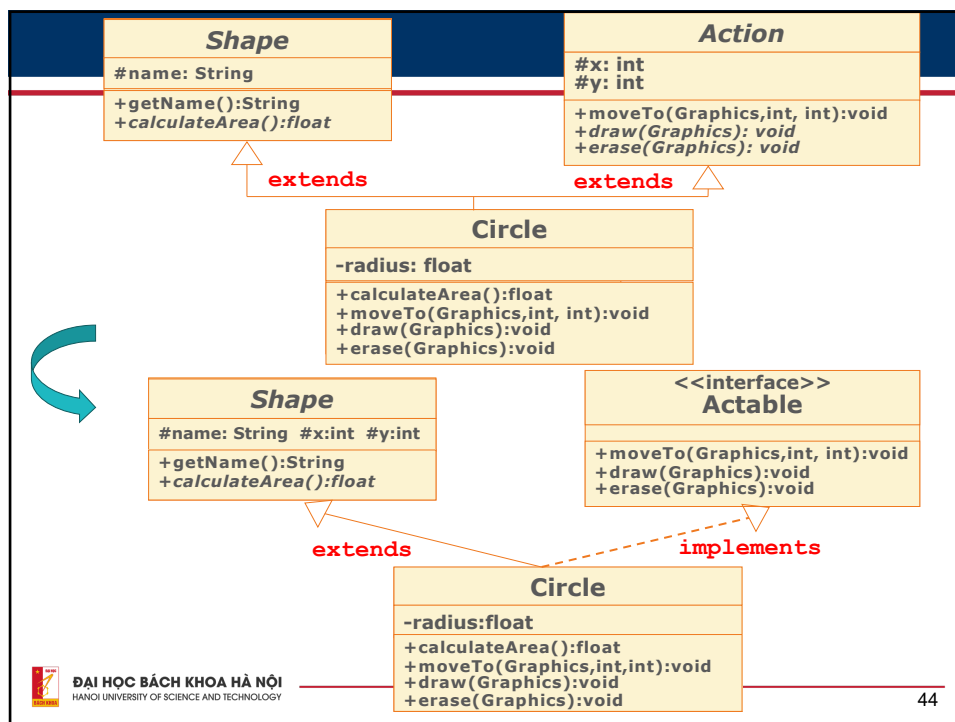
- Lớp thực thi giao diện
  - Hoặc là lớp trừu tượng (abstract class)
  - Hoặc là bắt buộc phải cài đặt chi tiết toàn bộ các phương thức trong giao diện nếu là lớp cụ thể
- Một lớp có thể thực thi nhiều giao diện

<Lớp con> [**extends** <Lớp cha>] **implements** <Danh sách giao diện>
- Ví dụ:

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



43



44

## 4. Ví dụ giao diện

```
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 moveTo(Graphics g, int x1, int y1);
    public void draw(Graphics g);
    public void erase(Graphics g);
}
```



45

```
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 circle with background color...
    }
}
```



46

## 4. Giao diện

- Giao diện có thể được sử dụng như một kiểu dữ liệu
- Các đối tượng gán cho biến tham chiếu giao diện phải thuộc lớp thực thi giao diện

- Ví dụ:

```
public interface I {}  
public class A implements I {}  
public class B {}  
A a = new A();  
B b = new B();  
I i1 = new A(); // ok  
I i2 = new B(); // lỗi
```



## 4. 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





## 4. Giao diện

- 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 – 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 cài đặt chi tiết
  - Interface: đặc tả cho các bản cài đặt (implementation) khác nhau.
  - Phân chia ranh giới:
    - Cái gì (What) và như thế nào (How)
    - Đặc tả và Cài đặt cụ thể.



## 4. Giao diện

- |   |   |
|---|---|
| <ul style="list-style-type: none"><li>• Lớp trừu tượng<ul style="list-style-type: none"><li>• Cần có ít nhất một phương thức abstract, có thể chứa các phương thức instance</li><li>• Có thể chứa các phương thức protected và static</li><li>• Có thể chứa các thuộc tính final và non-final</li><li>• Một lớp chỉ có thể kế thừa một lớp trừu tượng</li></ul></li></ul> | <ul style="list-style-type: none"><li>• Giao diện<ul style="list-style-type: none"><li>• Chỉ có thể chứa chữ ký phương thức (danh sách các phương thức)</li><li>• Chỉ có thể chứa các phương thức public mà không có mã nguồn</li><li>• Chỉ có thể chứa các thuộc tính hằng</li><li>• Một lớp có thể thực thi (kế thừa) nhiều giao diện</li></ul></li></ul> |
|---|---|



## 4. 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 nhằm tăng tái sử dụng mã nguồn nhưng giao diện không làm được điều này

51

## Bài tập 4

- 1. Khai báo nào là hợp lệ trong một interface?
  - a. `public static int answer = 42;`
  - b. `int answer;`
  - c. `final static int answer = 42;`
  - d. `public int answer = 42;`
  - e. `private final static int answer = 42;`
- 2. Một lớp có thể kế thừa chính bản thân nó không?
- 3. Chuyện gì xảy ra nếu lớp cha và lớp con đều có thuộc tính trùng tên?
- 4. Phát biểu "Các phương thức khởi tạo cũng được thừa kế xuống các lớp con" là đúng hay sai?
- 5. Có thể xây dựng các phương thức khởi tạo cho lớp trừu tượng không?
- 6. Có thể khai báo phương thức protected trong một giao diện không?

52

## Nội dung

1. Định nghĩa lại/ghi đè (Overriding)
2. Lớp trừu tượng
3. Đơn kế thừa & Đa kế thừa
4. Giao diện (Interface)
5. **Vai trò của lớp trừu tượng và giao diện**
6. Ví dụ và bài tập



53

## 5. Lớp trừu tượng & Giao diện

- 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 quan hệ **IS-A** đối với bất cứ loại nào khác
  - Một lớp nên là lớp con nếu cần cho nó làm một phiên bản chuyên biệt hơn của một lớp khác và cần ghi đè hành vi có sẵn hoặc bổ sung hành vi mới



54

## 5. Lớp trừu tượng & Giao diện

- 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 cha nếu muốn định nghĩa một khuôn mẫu cho một nhóm các lớp con, và có mã cài đặt mà tất cả các lớp con kia có thể sử dụng
    - Cho lớp đó làm lớp trừu tượng nếu muốn đảm bảo rằng không ai được tạo đối tượng thuộc lớp đó
  - Dùng một interface nếu muốn định nghĩa một vai trò mà các lớp khác có thể nhận, bất kể các lớp đó thuộc cây thừa kế nào

## Mở rộng

- Khái niệm giao diện trong các phiên bản của Java

JAVA 7	JAVA 8	JAVA 9
<ul style="list-style-type: none"><li>• Chữ ký các phương thức (method signature)</li><li>• Các thuộc tính khai báo hằng (static &amp; final)</li></ul>	<ul style="list-style-type: none"><li>• Chữ ký các phương thức (method signature)</li><li>• Các thuộc tính khai báo hằng (static &amp; final)</li><li>• Phương thức mặc định (default method)</li><li>• Phương thức tĩnh (Static method)</li></ul>	<ul style="list-style-type: none"><li>• Chữ ký các phương thức (method signature)</li><li>• Các thuộc tính khai báo hằng (static &amp; final)</li><li>• Phương thức mặc định (default method)</li><li>• Phương thức tĩnh (Static method)</li><li>• Private methods</li></ul>

## Ví dụ Java 8 Interface – default methods

<https://gpcoder.com/3854-interface-trong-java-8-default-method-va-static-method/>

```
public interface Shape {  
  
    void draw();  
  
    default void setColor(String color) {  
        System.out.println("Draw shape with color " + color);  
    }  
}
```

## Đa thừa kế

```
interface Interface1 {  
    default void doSomething() {  
        System.out.println("doSomething1");  
    }  
}  
  
interface Interface2 {  
    default void doSomething() {  
        System.out.println("doSomething2");  
    }  
}  
  
public class MultiInheritance implements Interface1, Interface2 {  
    @Override  
    public void doSomething() {  
        Interface1.super.doSomething();  
    }  
}
```

## Đa thừa kế

```
interface Interface3 {
    default void doSomething() {
        System.out.println("Execute in Interface3");
    }
}

class Parent {
    public void doSomething() {
        System.out.println("Execute in Parent");
    }
}

public class MultInheritance2 extends Parent implements Interface3 {
    public static void main(String[] args) {
        MultInheritance2 m = new MultInheritance2();
        m.doSomething(); // Execute in Parent
    }
}
```

## Ví dụ Java 8 interface – Static methods

```
interface Vehicle {
    default void print() {
        if (isValid())
            System.out.println("Vehicle printed");
    }
    static boolean isValid() {
        System.out.println("Vehicle is valid");
        return true;
    }
    void showLog();
}

public class Car implements Vehicle {
    @Override
    public void showLog() {
        print();
        Vehicle.isValid();
    }
}
```

## Ví dụ với static method

```
interface Interface3 {
    default void doSomething() {
        System.out.println("Execute in Interface3");
    }
}

abstract class Parent {
    public void doSomething() {
        System.out.println("Execute in Parent");
    }
    public static void test() {
        System.out.println("test");
    }
}

public class MultilInheritance2 extends Parent implements Interface3 {
    public static void main(String[] args) {
        MultilInheritance2 m = new MultilInheritance2();
        m.doSomething(); // Execute in Parent

        m.test(); // OK
    }
}
```

## Ví dụ với static method

```
interface Interface3 {
    default void doSomething() {
        System.out.println("Execute in Interface3");
    }
}

abstract class Parent {
    public void doSomething() {
        System.out.println("Execute in Parent");
    }
    public static void test() {
        System.out.println("test");
    }
}

public class MultilInheritance2 extends Parent implements Interface3 {
    public static void main(String[] args) {
        MultilInheritance2 m = new MultilInheritance2();

        m.test(); // OK
    }
}
```

## Ví dụ với static method

```
interface Interface3 {
    default void doSomething() {
        System.out.println("Execute in Interface3");
    }
}

abstract class Parent {
    public void doSomething() {
        System.out.println("Execute in Parent");
    }
    public static void test() {
        System.out.println("test");
    }
}

public class MultInheritance2 extends Parent implements Interface3 {
    public static void main(String[] args) {
        MultInheritance2 m = new MultInheritance2();

        MultInheritance2.test(); // OK
    }
}
```



## Ví dụ với static method

```
interface Interface3 {
    default void doSomething() {
        System.out.println("Execute in Interface3");
    }
    public static void test() {
        System.out.println("test");
    }
}

abstract class Parent {
    public void doSomething() {
        System.out.println("Execute in Parent");
    }
}

public class MultInheritance2 extends Parent implements Interface3 {
    public static void main(String[] args) {
        MultInheritance2 m = new MultInheritance2();

        m.test(); // ERROR!!!
    }
}
```





## Ví dụ với static method

```
interface Interface3 {  
    default void doSomething() {  
        System.out.println("Execute in Interface3");  
    }  
    public static void test() {  
        System.out.println("test");  
    }  
}  
  
abstract class Parent {  
    public void doSomething() {  
        System.out.println("Execute in Parent");  
    }  
}  
  
public class MultInheritance2 extends Parent implements Interface3 {  
    public static void main(String[] args) {  
        MultInheritance2 m = new MultInheritance2();  
  
        MultInheritance2.test(); // ERROR!!!  
    }  
}
```

## 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

## Nội dung

1. Định nghĩa lại/ghi đè (Overriding)
2. Lớp trừu tượng
3. Đơn kế thừa & Đa kế thừa
4. Giao diện (Interface)
5. Vai trò của lớp trừu tượng và giao diện
6. Ví dụ và bài tập



**HUST**

**THANK YOU !**