

**ĐA HÌNH**

# **Nội dung chính**

- 1. Lớp trừu tượng**
- 2. Giao diện**
- 3. Giao diện Comparable**
- 4. Giao diện Cloneable**
- 5. Biểu thức lambda**
- 6. Đa hình**
- 7. Lập trình tổng quan**

# Lớp trừu tượng

- Sự trừu tượng hoá thể hiện thông qua lớp trừu tượng và giao diện.
- Để khai báo lớp trừu tượng dùng từ khóa **abstract** trước tên lớp.

```
abstract class TênLớp {  
  
}
```

- Lớp trừu tượng **không thể** tạo đối tượng.

# Phương thức trừu tượng

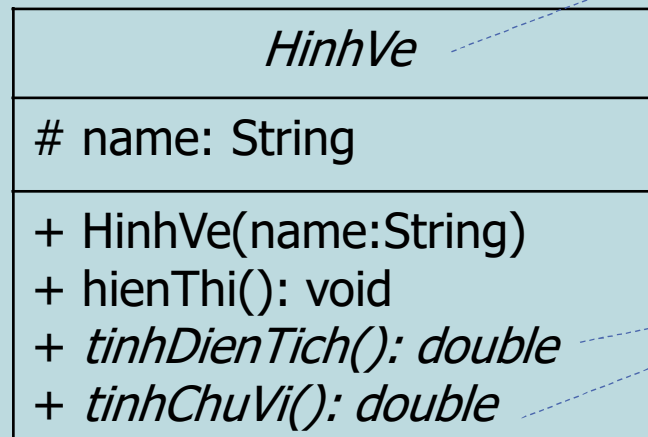
- Phương thức trừu tượng là phương thức **không có phần thân** thực thi, mà nó được quyết định ở lớp con.
- Để khai báo phương thức trừu tượng sử dụng từ khoá **abstract** trước kiểu dữ liệu trả về.

# Phương thức trừu tượng

- Lớp có **ít nhất** một phương thức trừu tượng thì phải là lớp trừu tượng.
- Lớp con **không hiện thực tất cả** các phương thức trừu tượng của lớp cha thì cũng trở thành lớp trừu tượng.

# Lớp trừu tượng

## ▪ Ký hiệu UML



***In nghiên:*** lớp trừu tượng

***In nghiên:*** phương thức  
trừu tượng

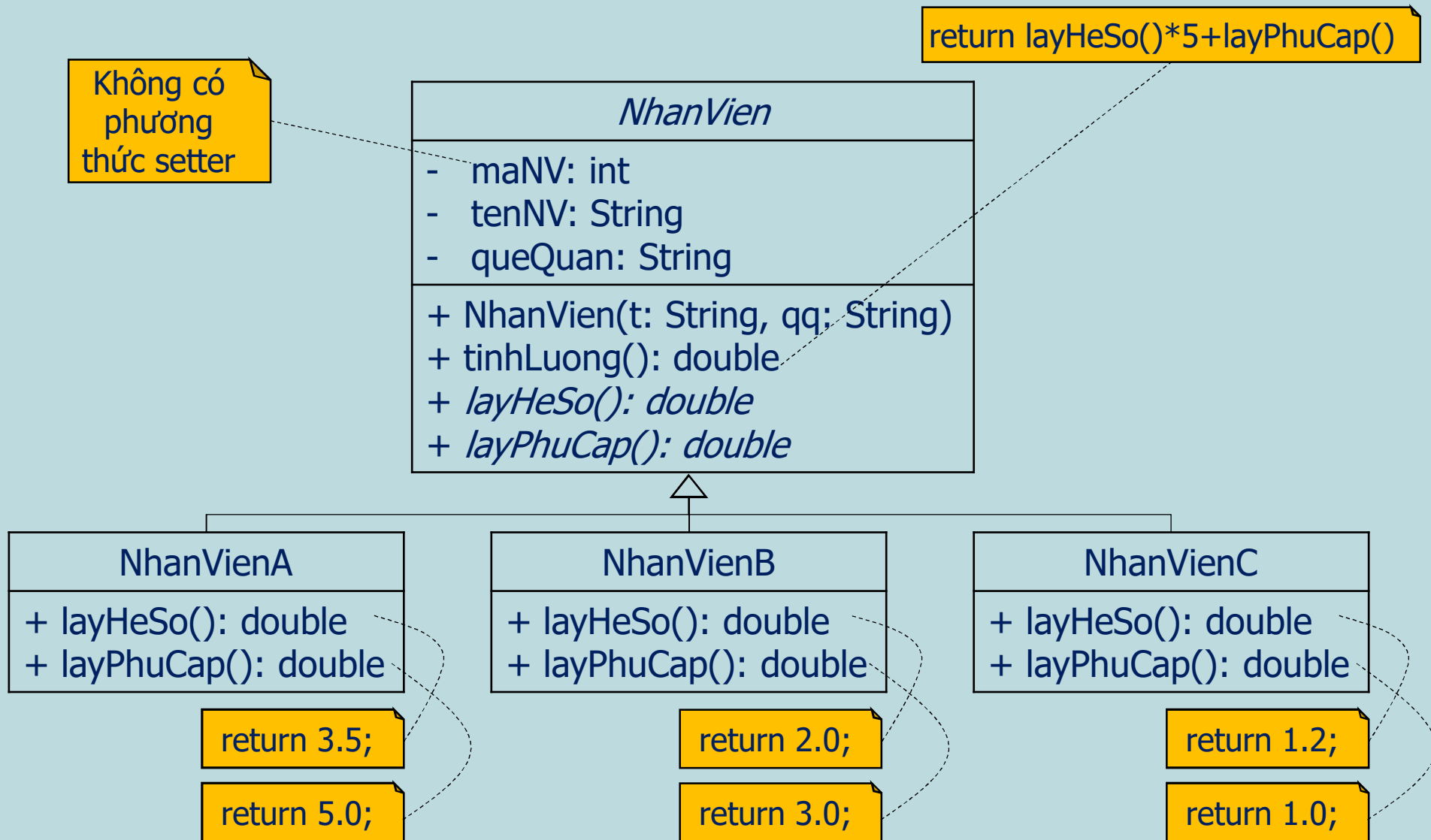
# Lớp trừu tượng

- Một hệ thống quản lý nhân viên có chức năng tính lương cho 3 loại nhân viên A, B, C:

Loại nhân viên	Tiền lương
A	Hệ số * Lương cơ bản + phụ cấp (5 triệu)
B	Hệ số * Lương cơ bản + phụ cấp (3 triệu)
C	Hệ số * Lương cơ bản + phụ cấp (1 triệu)

- Lương cơ bản tất cả nhân viên là 5 triệu. Hệ số lương loại A, B, C lần lượt 3.5, 2.0 và 1.2. Mỗi nhân viên bao gồm thông tin mã nhân viên, họ tên, ngày sinh, quê quán. Trong đó mã số nhân viên là số nguyên tự động tăng.

# Lớp trừu tượng





# Lớp trừu tượng

- Công ty quản lý thêm nhân viên thời vụ (loại D) không có hệ số lương và phụ cấp. Lương được trả theo số giờ làm việc của nhân viên.
- Viết lớp **NhanVienD** hiện thực lớp **Nhanvien** để tính lương cho nhân viên thời vụ?

# Phát biểu B tương ứng khái báo nào A?

A	B
<b>1</b> public final class PhanSo { }	<b>a</b> Lớp này không thể tạo đối tượng và có thể truy cập từ bất cứ lớp nào.
<b>2</b> public abstract class Nguoi { }	<b>b</b> Lớp này chỉ được truy cập từ các lớp trong cùng gói, nhưng không được phép kế thừa từ lớp này.
<b>3</b> final class SinhVien { }	<b>c</b> Lớp này chỉ truy cập từ các lớp trong cùng gói và các lớp đó có thể kế thừa từ lớp này.
<b>4</b> public class Diem { }	<b>d</b> Lớp này có thể truy cập từ bất kỳ lớp nào và các lớp có thể kế thừa từ lớp này.
<b>5</b> class DienThoai { }	<b>e</b> Lớp này có thể truy cập từ bất kỳ lớp nào, nhưng không được phép kế thừa từ lớp này.

# Tìm lỗi đoạn chương trình và giải thích?

- Đoạn chương trình có lỗi gì và giải thích?

```
public class HìnhVe {  
    public abstract double tinhDienTich();  
    public void hienThi() {  
        System.out.println("Dien tich = "  
            + this.tinhDienTich());  
    }  
}
```

```
HìnhVe hv1 = new HìnhVe();  
hv1.hienThi();
```

# Cho biết kết quả đoạn chương trình?

```
abstract class A {  
    public abstract int tinhToan(int a, int b);  
    public abstract void hienThi();  
}  
class B extends A {  
    @Override  
    public int tinhToan(int a, int b) {  
        return a + b;  
    }  
}
```

```
B b = new B();  
System.out.println(b.tinhToan(2, 5));
```

# **Phát biểu nào sau đây đúng?**

- A. Lớp trừu tượng chỉ chứa các phương thức trừu tượng.**
- B. Lớp trừu tượng không có phương thức khởi tạo.**
- C. Các lớp trừu tượng không thể kế thừa lẫn nhau.**
- D. Trong lớp trừu tượng không thể khai báo phương thức tĩnh.**
- E. Lớp không có phương thức trừu tượng nào không thể khai báo là lớp trừu tượng.**

# Lớp trừu tượng

- Cho hai lớp A, B và đoạn chương trình trong **main**, viết lớp C để kết quả xuất ra "ABC".

```
abstract class A {  
    abstract void show1();  
}  
  
abstract class B extends A {  
    @Override  
    void show1() {  
        System.out.print("B");  
    }  
  
    abstract void show2();  
}
```

```
A a1 = new C();  
a1.show1();  
B b1 = new C();  
b1.show2();
```

# **Các phát biểu sau đúng hay sai?**

- A. Phương thức trừu tượng có thể khai báo là `private` hoặc `final` hoặc `static`.**
- B. Lớp trừu tượng bắt buộc phải có ít nhất một phương thức trừu tượng.**
- C. Các lớp trừu tượng có thể kế thừa nhau.**
- D. Lớp con không thể truy cập vào phương thức `public abstract` của lớp cha.**
- E. Lớp trừu tượng có thể kế thừa lớp thường.**

# Lớp trừu tượng

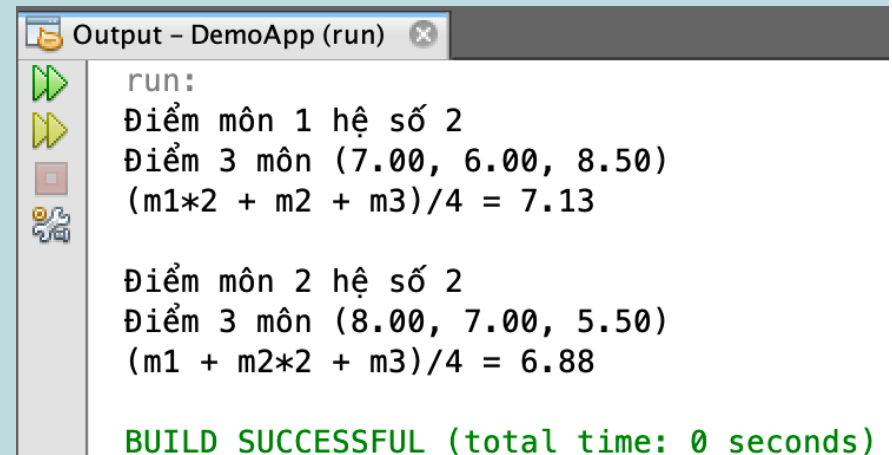
- Cho lớp M và đoạn chương trình trong **main**, viết các lớp M, A, B để kết quả hiển thị như trong hình?

<i>M</i>
# m1, m2, m3: double
+ M(m1:double, m2:double, m3:double)
+ <i>cal(): double</i>
+ <i>formula(): String</i>
+ show(): void

Trả về kết quả điểm

Trả về chuỗi công thức tính điểm

```
M v1 = new A(7, 6, 8.5);  
M v2 = new B(8, 7, 5.5);  
v1.show();  
v2.show();
```



```
run:  
Điểm môn 1 hệ số 2  
Điểm 3 môn (7.00, 6.00, 8.50)  
(m1*2 + m2 + m3)/4 = 7.13  
  
Điểm môn 2 hệ số 2  
Điểm 3 môn (8.00, 7.00, 5.50)  
(m1 + m2*2 + m3)/4 = 6.88  
  
BUILD SUCCESSFUL (total time: 0 seconds)
```



# Giao diện

- Giao diện (interface) là một kiến trúc giống như lớp nhưng chỉ chứa
  - Các thuộc tính **static** hoặc **final**.
  - Các phương thức **trừu tượng, tĩnh**.
- Từ Java 8, giao diện cho phép khai báo các phương thức **mặc định**.
- Từ Java 9, giao diện cho phép khai báo phương thức **private** chỉ được truy cập trong interface, thường là từ các phương thức mặc định của interface.

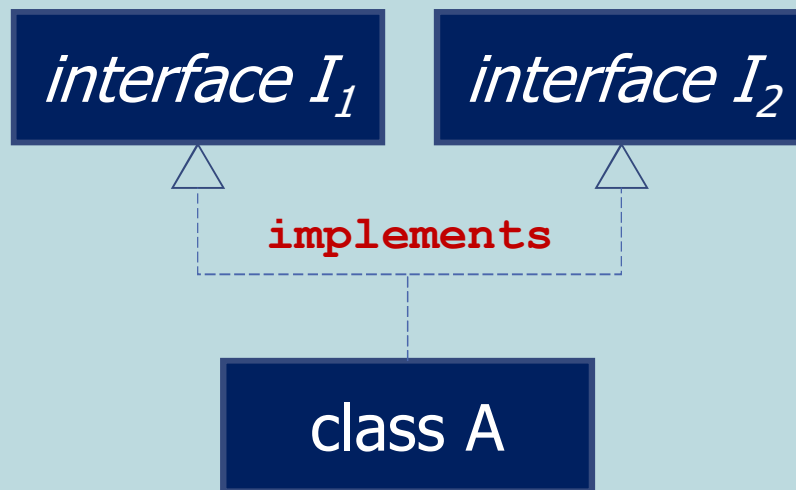
# Giao diện

- Cú pháp khai báo giao diện

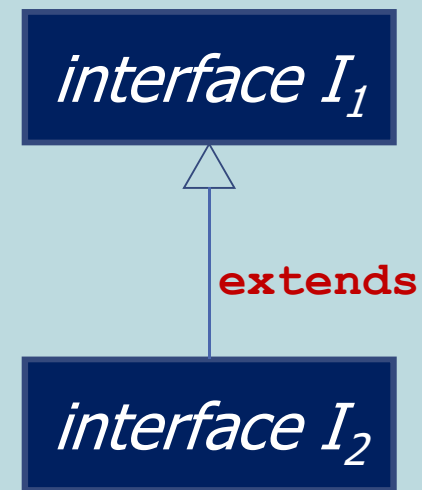
```
AccessModifier interface TênInterface {  
    [Các thuộc tính];  
    [Các phương thức];  
}
```

- Lớp hiện thực hoá giao diện **phải** hiện thực tất cả các phương thức của giao diện, ngược lại nó sẽ thành lớp trừu tượng.

# Giao diện



Lớp hiện thực giao diện



Giao diện kế thừa  
giao diện khác

# Giao diện

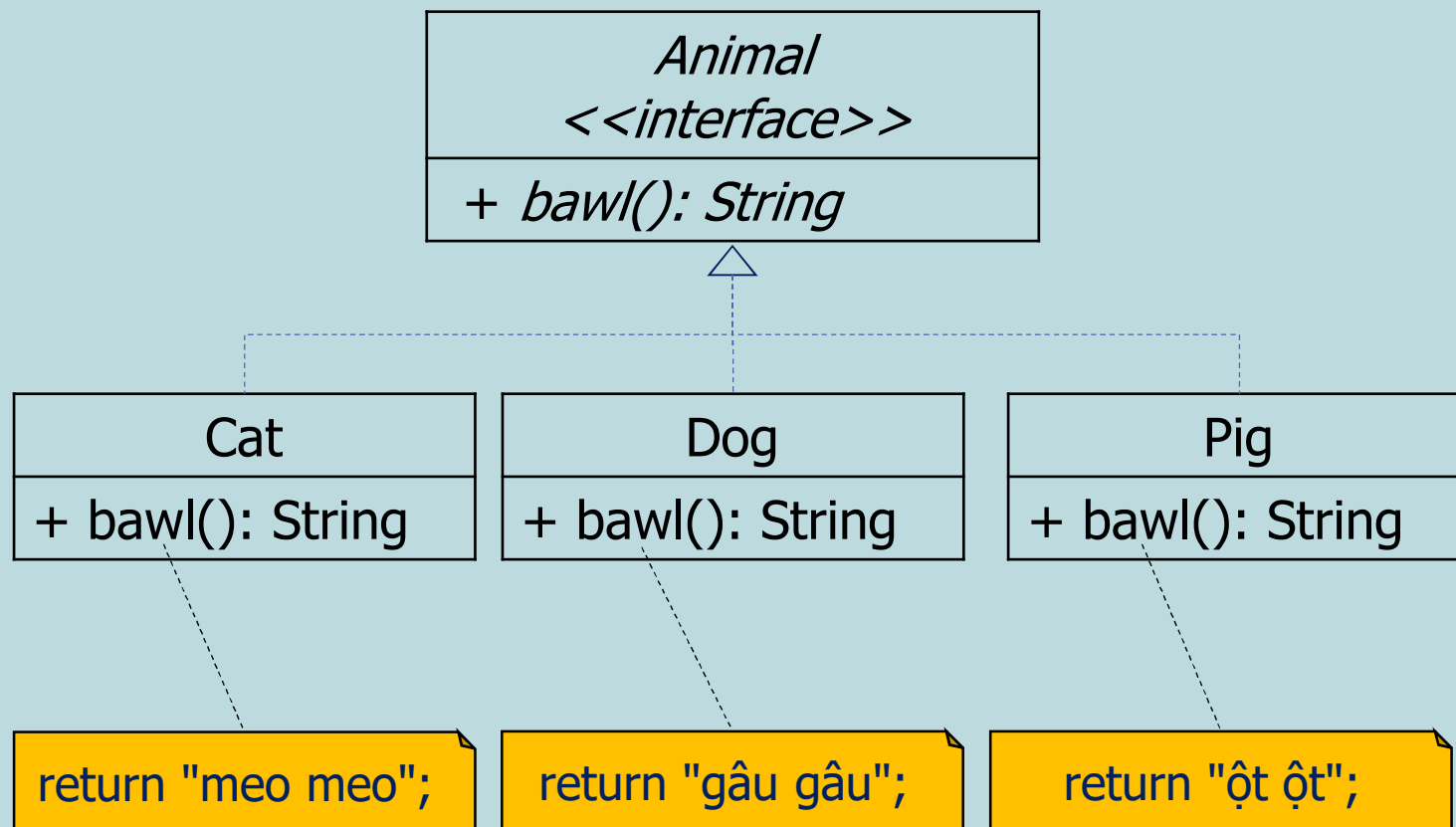
- Giao diện kế thừa giao diện

```
interface TênInterface extends TênInterface {  
    [<thực-thi-các-phương-thức-interface>];  
}
```

- Hiện thực hóa giao diện

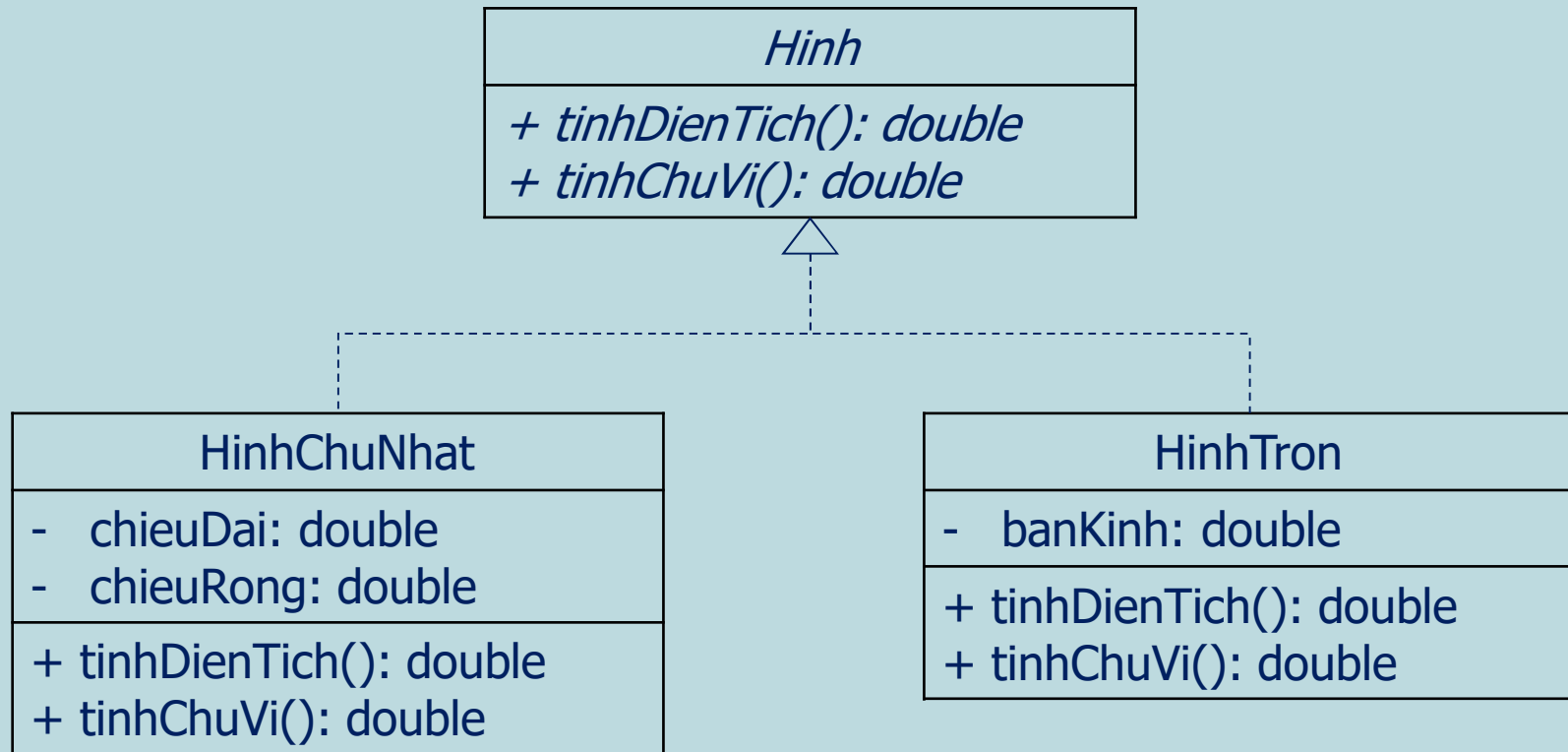
```
class TênLớp implements TênInterface {  
    [<thực-thi-các-phương-thức-interface>];  
}
```

# Giao diện



# Giao diện

## ▪ Ví dụ



# Giao diện

## ▪ Giao diện Hình

```
public interface Hình {  
    // Các phương thức trừu tượng  
    double tinhDienTich();  
    double tinhChuVi();  
    // Phương thức mặc định  
    default void hienThi() {  
        String s = "Chu vi: %.2f\nDien tich: %.2f\n";  
        s = String.format(s, this.tinhChuVi(),  
                           this.tinhDienTich());  
        System.out.println(s);  
    }  
}
```

# Giao diện

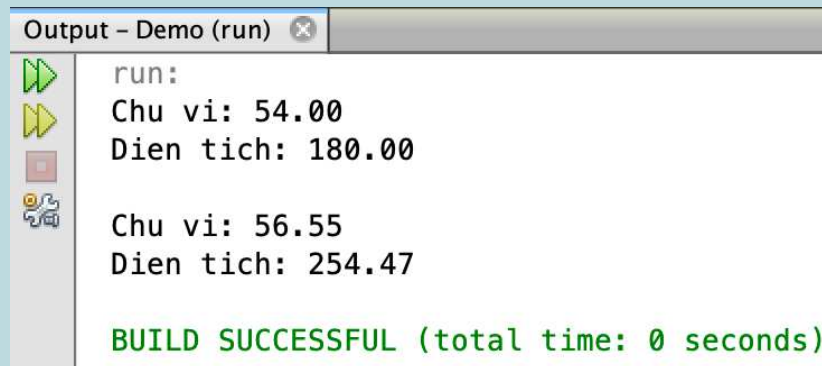
```
class HìnhTron implements Hình {  
    private double banKinh;  
    public HìnhTron(double bk) {  
        this.banKinh = bk;  
    }  
    @Override  
    public double tinhDienTich() {  
        return Math.pow(this.banKinh, 2)*Math.PI;  
    }  
    @Override  
    public double tinhChuVi() {  
        return 2*this.banKinh*Math.PI;  
    }  
}
```



# Giao diện

- Tương tự cho lớp **HìnhChuNhat**
- Sử dụng các lớp này

```
Hinh h1 = new HinhChuNhat(15, 12);  
Hinh h2 = new HinhTron(9);  
  
h1.hienThi();  
h2.hienThi();
```



```
Output - Demo (run) x  
run:  
Chu vi: 54.00  
Dien tich: 180.00  
  
Chu vi: 56.55  
Dien tich: 254.47  
  
BUILD SUCCESSFUL (total time: 0 seconds)
```

# Chú ý sử dụng giao diện

- **Không** thể tạo thể hiện từ giao diện.
- Giao diện **không** có phương thức khởi tạo.
- Giao diện **không** được phép kế thừa lớp.
- Các giao diện **được phép** kế thừa nhau.

# Giao diện

- Cho biết lỗi đoạn chương trình và giải thích?

```
interface A {  
    void show();  
}  
  
class B implements A {  
    @Override  
    void show() {  
        System.out.println("Hello");  
    }  
}
```

# Giao diện

- Cho biết kết quả đoạn chương trình?

```
interface IA {  
    String show();  
}  
interface IB {  
    void show(String s);  
}
```

```
class AB implements IA, IB {  
    @Override  
    public String show() {  
        return "TEST";  
    }  
}
```

```
AB a = new AB();  
System.out.print(a.show());
```

# Giao diện

- Cho hai interface như sau

```
interface IA {  
    String getMessage();  
}  
  
interface IB {  
    void getMessage();  
}
```

- Viết lớp hiện thực cả hai giao diện trên?

# Giao diện Comparable

- Giao diện **Comparable** dùng so sánh 2 đối tượng

```
public interface Comparable<E> {  
    public int compareTo(E obj);  
}
```

- Phương thức **compareTo()** chỉ định cách so sánh đối tượng hiện tại **this** và **obj**, phương thức này trả về
  - Số dương: **this > obj**
  - Số 0: **this = obj**
  - Số âm: **this < obj**

# Giao diện Comparable

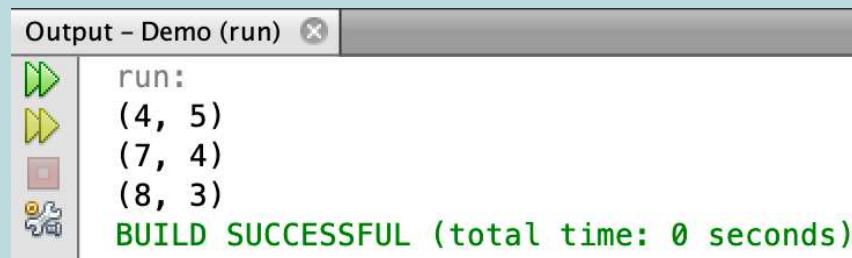
- Ví dụ sắp xếp dãy phân số.
- Lớp PhanSo hiện thực giao diện Comparable

```
public class PhanSo implements Comparable<PhanSo> {  
    ...  
    @Override  
    public int compareTo(PhanSo o) {  
        PhanSo p = this.tru(o);  
        return p.tuSo * p.mauSo;  
    }  
}
```

# Giao diện Comparable

- Tạo dãy phân số, sắp xếp và xuất kết quả

```
PhanSo[] ds = {  
    new PhanSo(8, 3),  
    new PhanSo(4, 5),  
    new PhanSo(7, 4)  
};  
Arrays.sort(ds);  
for (PhanSo p:ds)  
    System.out.printf("( %d, %d) \n",  
        p.getTuSo(), p.getMauSo());
```



```
Output - Demo (run) x  
run:  
(4, 5)  
(7, 4)  
(8, 3)  
BUILD SUCCESSFUL (total time: 0 seconds)
```



# Giao diện Comparable

- Cho biết lỗi của đoạn chương trình và giải thích?

```
class Student {  
    private int id;  
    public Student(int id) {  
        this.id = id;  
    }  
}
```

```
Student[] students = {  
    new Student(1), new Student(5),  
    new Student(7), new Student(3)  
};  
Arrays.sort(students);
```

# Giao diện Cloneable

- Giao diện **Cloneable** dùng tạo bản sao các đối tượng.

```
public interface Cloneable {  
  
}
```

- Đây là giao diện có phần thân rỗng dùng đánh dấu (marked interface) khả năng tạo bản sao bằng phương thức **clone()** của lớp **Object**.

# Giao diện Cloneable

```
class Khoa {  
    private String ten;  
    public Khoa(String t) {  
        this.ten = t;  
    }  
}
```

```
class SinhVien implements Cloneable {  
    private String ten;  
    private Khoa khoa;  
    public SinhVien(String t, Khoa k) {  
        this.ten = t;  
        this.khoa = k;  
    }  
    @Override  
    protected Object clone()  
        throws CloneNotSupportedException {  
        return super.clone();  
    }  
}
```

# Giao diện Cloneable

- Cho biết kết quả xuất ra đoạn chương trình?

```
Khoa k = new Khoa("CNTT");  
  
SinhVien sv1 = new SinhVien("A", k);  
SinhVien sv2 = (SinhVien) sv1.clone();  
sv2.getKhoa().setTen("Công nghệ thông tin");  
System.out.println(sv1.getKhoa().getTen());  
System.out.println(sv2.getKhoa().getTen());
```



shallow copy và  
deep copy

# Giao diện Cloneable

- Lớp Khoa cũng hiện thực Cloneable.
- Phương thức trong lớp SinhVien sửa lại:

```
@Override
protected Object clone()
    throws CloneNotSupportedException {
    SinhVien sv = (SinhVien) super.clone();
    sv.khoa = (Khoa) this.khoa.clone();
    return sv;
}
```

# Biểu thức lambda

- Cú pháp biểu thức lambda có dạng

```
( [<kiểu1>] p1, [<kiểu2>] p2, ...) -> <một-câu-lệnh>

( [<kiểu1>] p1, [<kiểu2>] p2, ...) -> {
    <nhiều-câu-lệnh>;
}
```

- Biểu thức lambda giúp cho việc lập trình trở nên **ngắn gọn, đơn giản và dễ hiểu hơn**.
- Biểu thức lambda sử dụng chính để hiện thực các giao diện có **duy nhất** một phương thức trừu tượng.

# Biểu thức lambda

```
interface SayHello {  
    void say(String name);  
}  
  
class Demo {  
    public static void main(String[] args) {  
        SayHello s = (name) -> {  
            System.out.println("Hello " + name);  
        };  
        s.say("A");  
        s.say("B");  
    }  
}
```

# Đa hình

- Đa hình là khả năng một đối tượng **có nhiều hình thức**, nói cách khác các đối tượng khác nhau sử dụng cùng phương thức sẽ cho kết quả khác nhau.
- Tính đa hình thường đạt phổ biến nhất là đối tượng **lớp cha tham chiếu đến đối tượng lớp con**.



# Lập trình tổng quan

- Generic là cách **tham số hóa kiểu dữ liệu**.
- Kiểu Generic **bắt buộc là kiểu tham chiếu**.
- Quy ước dùng các ký hiệu E và T để khai báo generic.
- Lớp hay giao diện đều có thể định nghĩa generic.

# Lập trình tổng quan

```
public class GenericStack<T> {  
    private ArrayList<T> stack = new ArrayList<>();  
  
    public void push(T x) {  
        this.stack.add(x);  
    }  
  
    public T pop() {  
        int size = this.stack.size();  
        T x = this.stack.get(size - 1);  
        this.stack.remove(size - 1);  
        return x;  
    }  
}
```

# Tổng kết

- Đa hình là gì? Nó được thể hiện thông qua cơ chế nào?
- Đặc điểm của interface trong Java?
- Phân biệt lớp trừu tượng và interface?

