

Bài 6  
Kế thừa

# Nội dung

1. Khái niệm kế thừa
2. Biểu diễn quan hệ kế thừa trong biểu đồ lớp
3. Nguyên lý kế thừa
4. Khởi tạo và hủy bỏ đối tượng lớp con

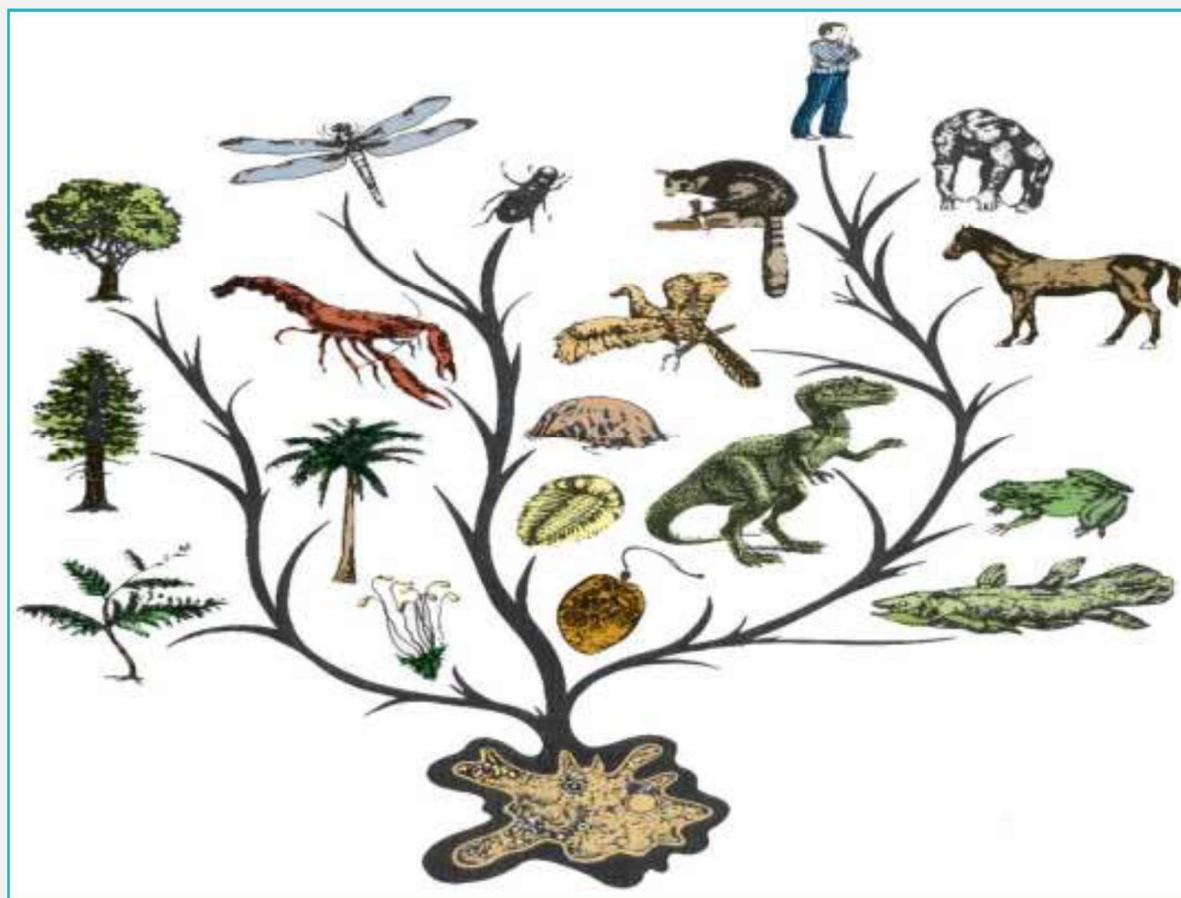
1

# Khái niệm kế thừa

Inheritance

# Kế thừa

- Kế thừa?



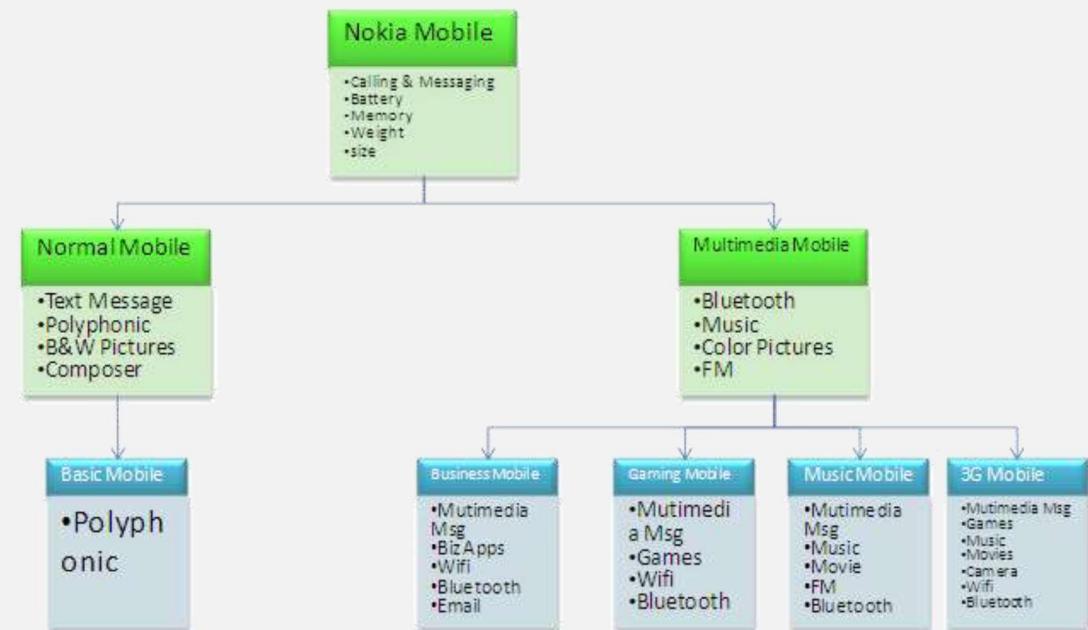
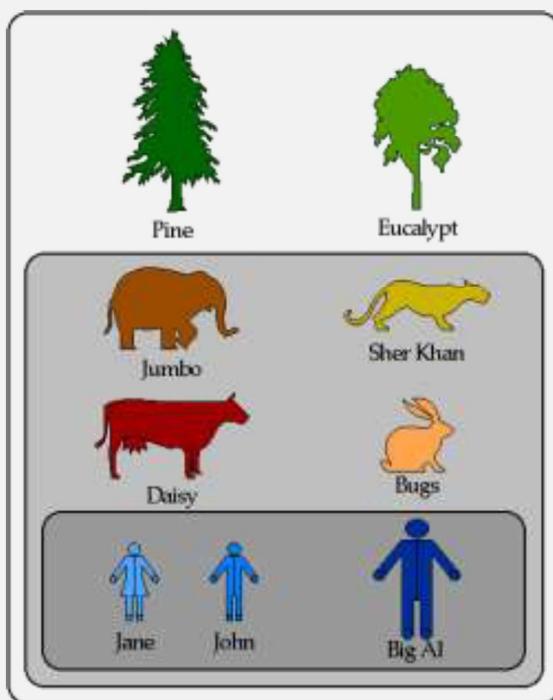
"Xây dựng các lớp mới có sẵn các đặc tính của lớp cũ,  
đồng thời chia sẻ hay mở rộng các đặc tính sẵn có"

# Bản chất kế thừa

- Phát triển lớp mới dựa trên các lớp đã có
- Ví dụ
  - Lớp Người có các thuộc tính như tên, tuổi, chiều cao, cân nặng...; các phương thức như ăn, ngủ, chơi...
  - Lớp Sinh Viên thừa kế từ lớp Người, thừa kế được các thuộc tính tên, tuổi, chiều cao, cân nặng...; các phương thức ăn, ngủ, chơi...
  - Bổ sung thêm các thuộc tính như mã số sinh viên, số tín chỉ tích lũy..., các phương thức như tham dự lớp học, thi...

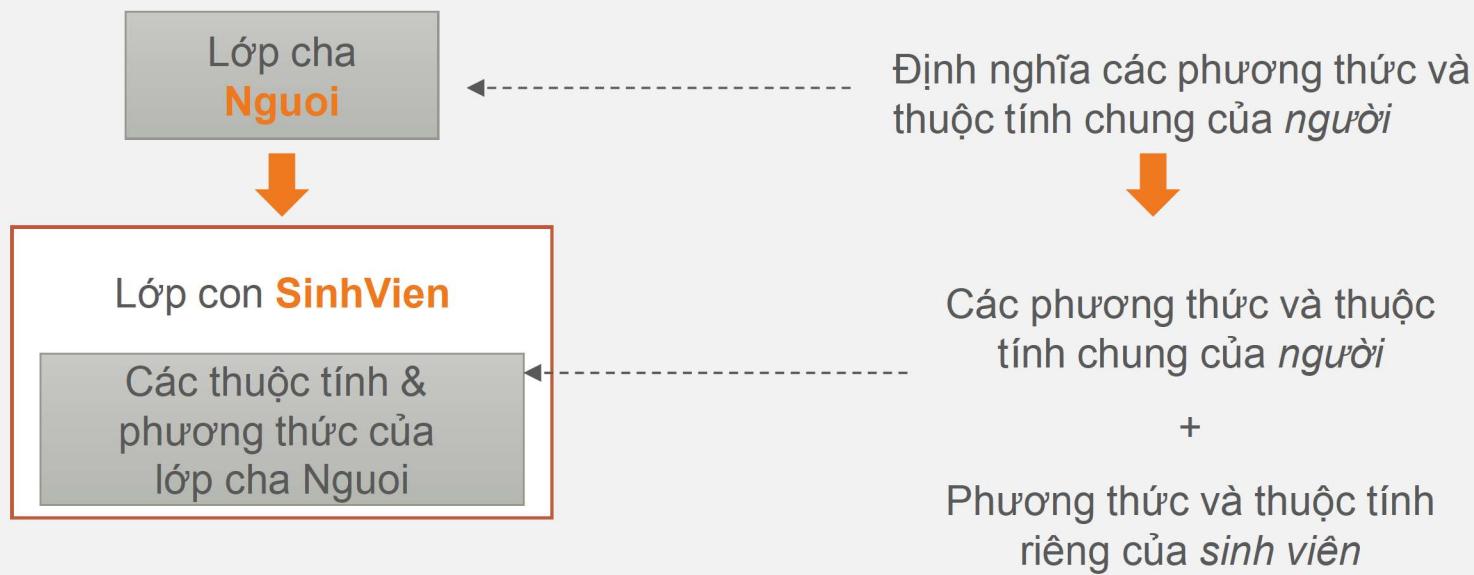
# Bản chất kế thừa

- Chính là nguyên lý phân cấp trong trừu tượng hóa



# Ví dụ

- Khái niệm
  - Lớp cũ: Lớp cha (parent, superclass), lớp cơ sở (base class)
  - Lớp mới: Lớp con (child, subclass), lớp dẫn xuất (derived class)
- Ví dụ: SinhVien thừa kế (dẫn xuất) từ lớp Nguoi



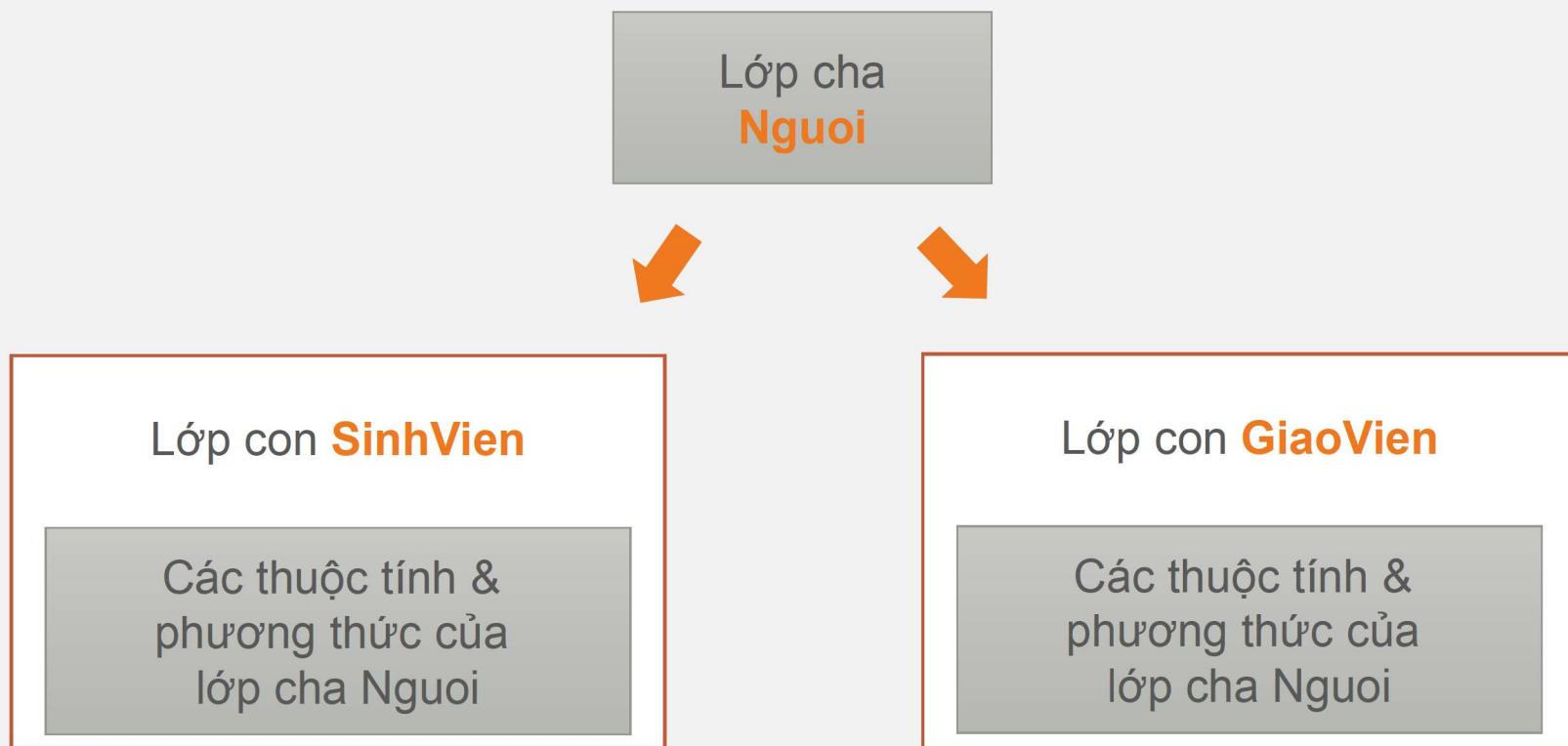
# Mối quan hệ kế thừa

- Lớp con và lớp cha có tính tương đồng
  - Lớp SinhVien kế thừa từ lớp Nguoi.
  - Một sinh viên là một người



# Mỗi quan hệ kế thừa

- Cả GiaoVien và SinhVien đều có quan hệ **là (is-a)** với lớp Nguoi
- Cả giáo viên và sinh viên đều có một số hành vi thông thường của con người



# 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): Chính sửa lại các hành vi kế thừa từ lớp cha



# Cú pháp (Java)

- Cú pháp (Java):

<Lớp con> **extends** <Lớp cha>

- Ví dụ

```
class Nguoi {  
    String name; int age;  
}  
  
class SinhVien extends Nguoi {  
    int studentId;  
}
```

- Lớp con *mở rộng* các đặc tính của lớp cha

Lớp cha **Nguoi**  
*name, age*



Lớp con **SinhVien**  
*studentId*

*name, age*



# Cú pháp (C++/C#)

- Cú pháp (Java):

<Lớp con> : <Lớp cha>

- Ví dụ

```
class Nguoi {  
    String name; int age;  
};  
  
class SinhVien : Nguoi {  
    int studentId;  
};
```

- Lớp con là một đối tượng lớp cha

Lớp cha **Nguoi**  
*name, age*



Lớp con **SinhVien**  
*studentId*  
  
*name, age*

# Bản chất kế thừa

- Là một kỹ thuật *tái sử dụng mã nguồn*
  - Tái sử dụng mã nguồn thông qua **lớp**
- Ví dụ: Lớp SinhViên tái sử dụng được các thuộc tính như tên, tuổi... và các phương thức của lớp Người.

# Kế thừa và kết tập

## Kế thừa

- Tái sử dụng thông qua *lớp*
- Tạo lớp mới bằng cách phát triển lớp đã có sẵn
- Quan hệ “*là một loại*” (is a kind of)

## Kết tập

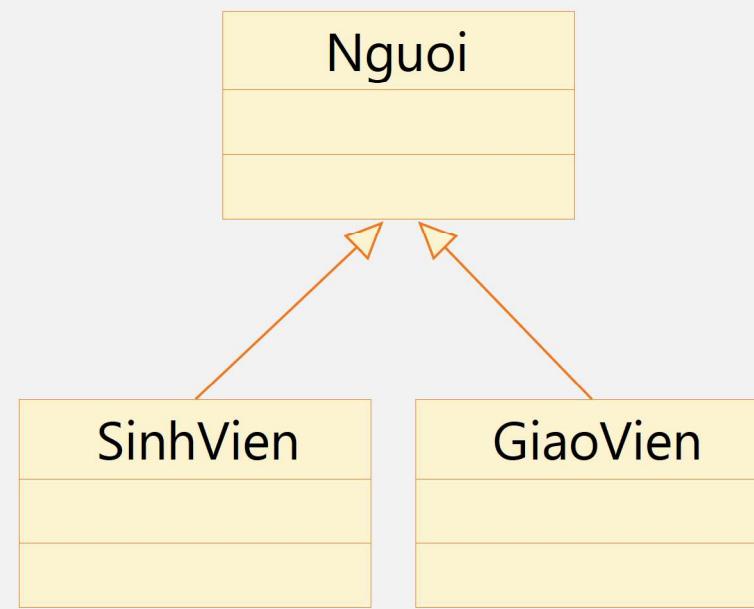
- Tái sử dụng thông qua *đối tượng*
- Tạo ra tham chiếu đến các đối tượng của các lớp có sẵn trong lớp mới
- Quan hệ “*là một phần*” (is a part of)

# 2

Biểu diễn quan hệ kế thừa  
trong biểu đồ lớp

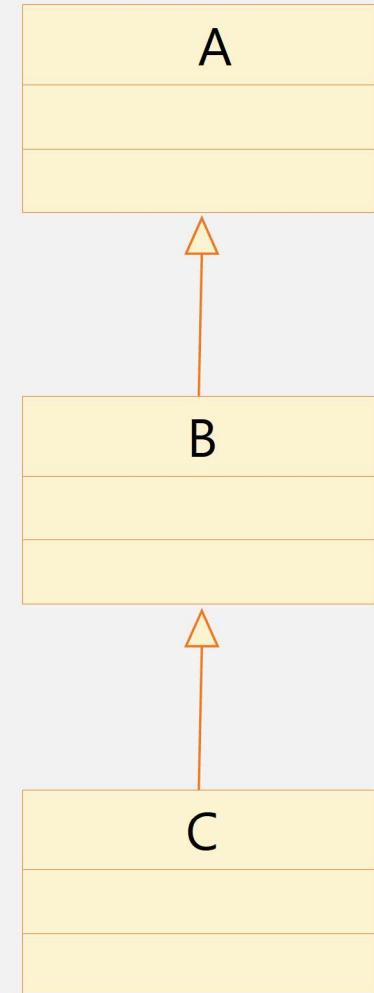
# Cây phân cấp kế thừa

- Dùng mũi tên với *tam giác rỗng* ở đầu



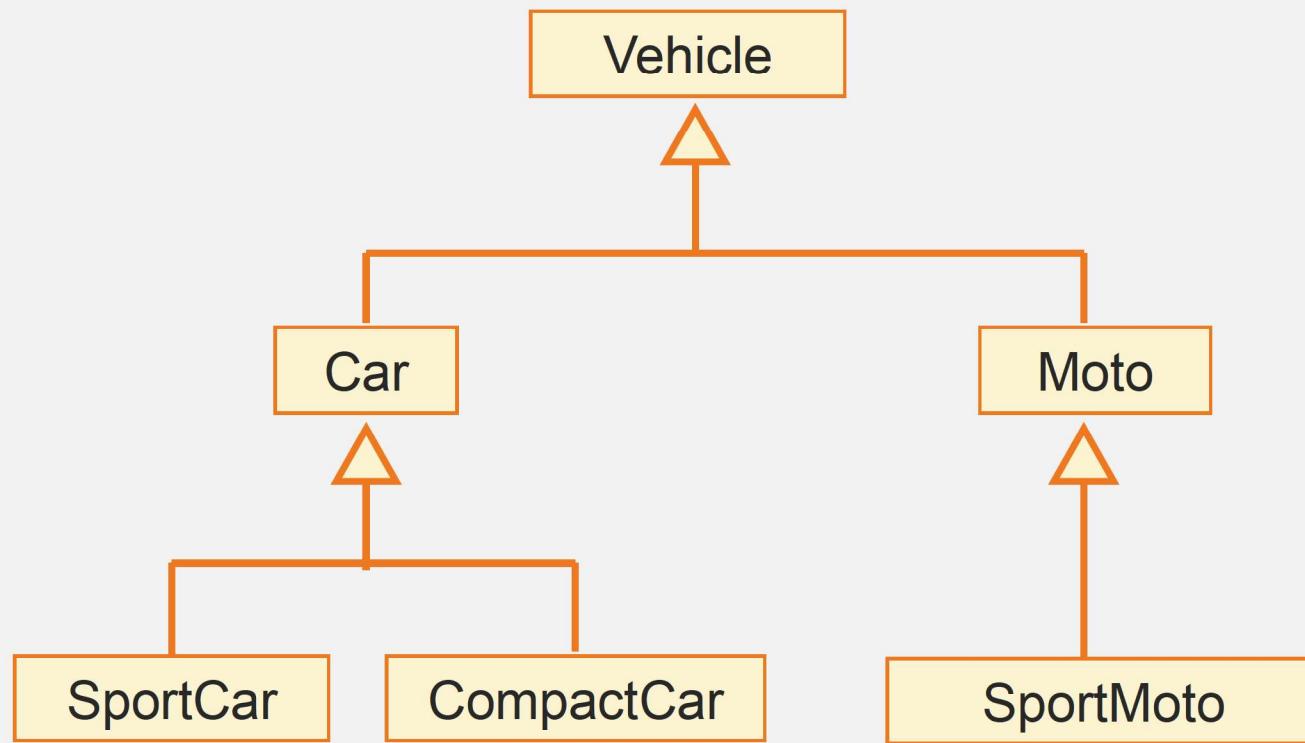
# Cây phân cấp kế thừa

- Cấu trúc phân cấp hình cây, biểu diễn mối quan hệ kế thừa giữa các lớp.
- Dẫn xuất trực tiếp
  - B dẫn xuất trực tiếp từ A
- Dẫn xuất gián tiếp
  - C dẫn xuất gián tiếp từ A



# Cây phân cấp kế thừa

- Các lớp con có cùng lớp cha gọi là các lớp anh chị em (siblings)





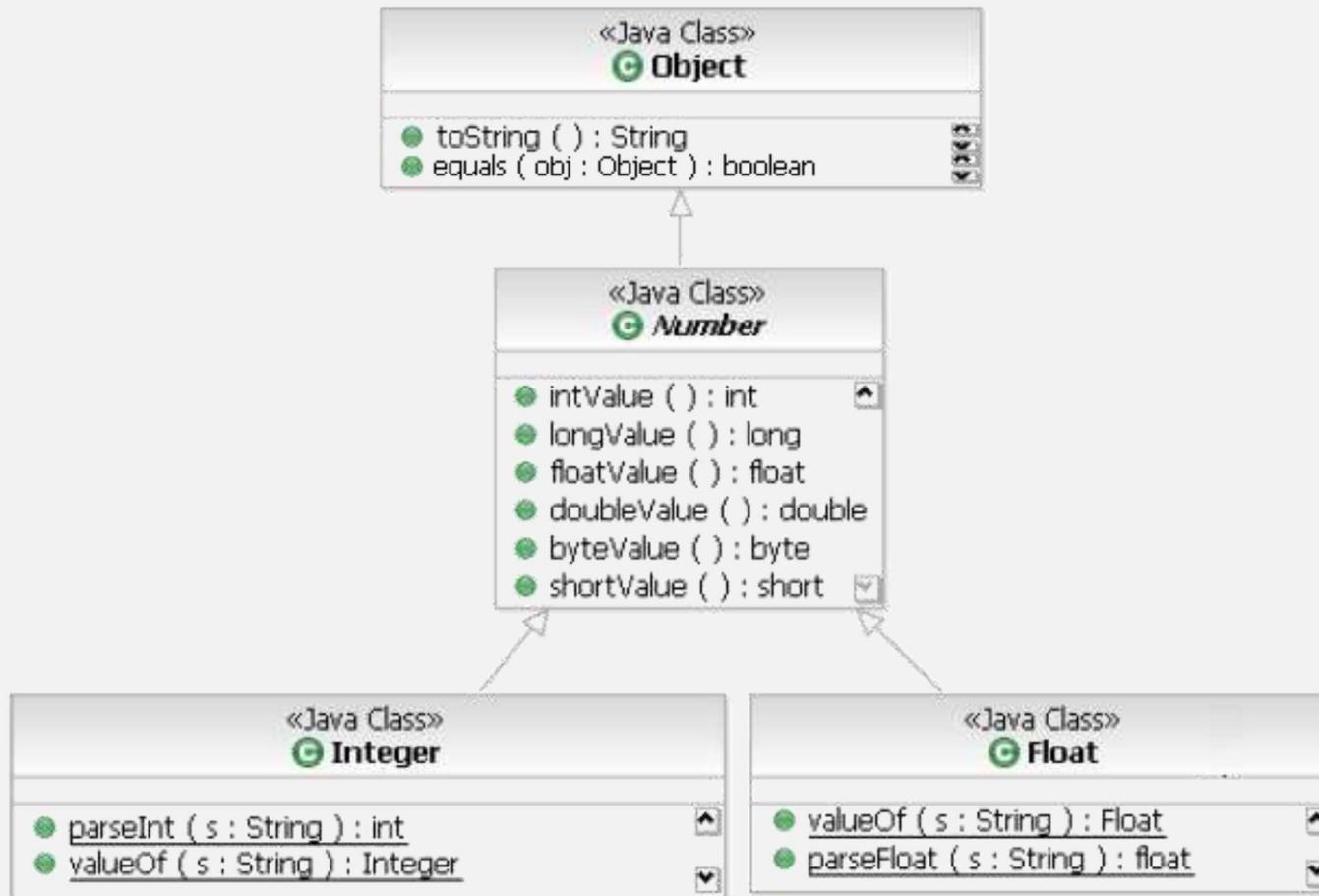
# Lớp Object

- Lớp **Object** là lớp gốc trên cùng của tất cả các cây phân cấp kế thừa
  - Nếu một lớp không được định nghĩa là lớp con của một lớp khác thì mặc định nó là lớp con trực tiếp của lớp **Object**.
- Được định nghĩa trong package chuẩn **java.lang**



# Lớp Object

- Chứa một số phương thức hữu ích kế thừa lại cho tất cả các lớp, ví dụ: **toString()**, **equals()**...



# 3

## Nguyên lý kế thừa

Lớp con kế thừa được những gì từ lớp cha?

# Nguyên lý kế thừa

- Lớp con có thể thừa kế được gì từ lớp cha?
  - Kế thừa được các thành viên được khai báo là **public** và **protected** của lớp cha.
  - Không kế thừa được các thành viên **private**.

*Thành viên **protected** trong lớp cha được truy cập trong:*

- *Các thành viên lớp cha*
- *Các thành viên lớp con*
- *Các thành viên các lớp cùng thuộc 1 package với lớp cha*

# Nguyên lý kế thừa

	public	protected	mặc định	private
Cùng lớp	✓	✓	✓	✓
Lớp bất kỳ cùng gói	✓	✓	✓	✗
Lớp con khác gói	✓	✓	✗	✗
Lớp bất kỳ khác gói	✓	✗	✗	✗

# Nguyên lý kế thừa

- Các phương thức không được phép kế thừa:
  - Các phương thức khởi tạo và hủy
    - + Làm nhiệm vụ khởi đầu và gỡ bỏ các đối tượng
    - + Chúng chỉ biết cách làm việc với từng lớp cụ thể
  - Toán tử gán =
    - + Làm nhiệm vụ giống như phương thức khởi tạo

# Ví dụ 1

```
public class TuGiac {  
    protected Diem d1, d2, d3, d4;  
    public void setD1(Diem _d1) {d1=_d1;}  
    public Diem getD1(){return d1;}  
    public void printTuGiac(){...}  
}
```

```
public class HinhVuong extends TuGiac {  
    public HinhVuong(){  
        d1 = new Diem(0,0); d2 = new Diem(0,1);  
        d3 = new Diem(1,0); d4 = new Diem(1,1);  
    }  
}
```

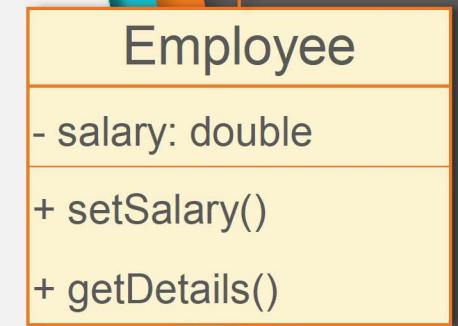
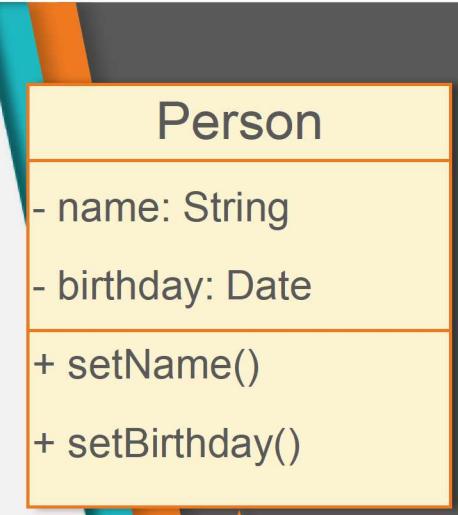
```
public class Test{  
    public static void main(String args[]){  
        HinhVuong hv = new HinhVuong();  
        hv.printTuGiac();  
    }  
}
```

Sử dụng các thành phần  
*protected* của lớp cha  
trong lớp con

Gọi phương thức *public* của lớp  
cha trong đối tượng lớp con

# Ví dụ 2

```
class Person {  
    private String name;  
    private Date bithday;  
    public String getName() {return name;}  
}  
  
class Employee extends Person {  
    private double salary;  
    public boolean setSalary(double sal) {  
        salary = sal;  
        return true;  
    }  
    public String getDetail() {  
        String s = name + ", " + birthday +  
                  ", " + salary; // ERROR  
    }  
}
```



## Ví dụ 2 (tiếp)

```
public class Test{  
    public static void main(String args[]){  
        Employee e = new Employee();  
        e.setName("John");  
        e.setSalary(3.0);  
    }  
}
```



# Ví dụ 3

- Cùng gói
- Khác gói

```
package abc;  
public class Person {  
    Date birthday;  
    String name;  
}
```

```
package abc.Person;  
public class Employee extends Person {  
    double salary;  
    public String getDetail() {  
        String s;  
        s = name + "," + birthday + "," + salary;  
        return s;  
    }  
}
```

# Ví dụ 3

```
package abc;  
public class Person {  
    protected Date birthday;  
    protected String name;  
}
```

```
package abc.Person;  
public class Employee extends Person {  
    double salary;  
    public String getDetail() {  
        String s;  
        s = name + "," + birthday + "," + salary;  
        return s;  
    }  
}
```

# 4

## Khởi tạo và hủy bỏ đối tượng

Thứ tự khởi tạo, gọi phương thức của lớp cha

# Khởi tạo và hủy bỏ đối tượng

- Khởi tạo và huỷ bỏ đối tượng trong kế thừa
- Khởi tạo đối tượng:
  - Lớp cha được khởi tạo trước lớp con.
  - Các phương thức khởi tạo của lớp con luôn gọi phương thức khởi tạo của lớp cha ở câu lệnh đầu tiên
    - + Tự động gọi (ngầm định - implicit): Khi lớp cha **CÓ** phương thức khởi tạo mặc định (hoặc ngầm định)
    - + Gọi trực tiếp (tường minh - explicit)
- Hủy bỏ đối tượng:
  - Ngược lại so với khởi tạo đối tượng

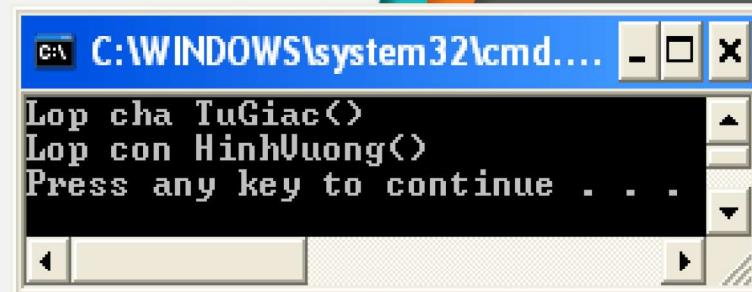
*Sử dụng từ khóa **super***

# Gọi phương thức của lớp cha

- 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
  - Phải được khai báo *tại dòng lệnh đầu tiên* trong phương thức khởi tạo của lớp con
- Gọi các phương thức của lớp cha `super.tênPt(danh sách tham số);`

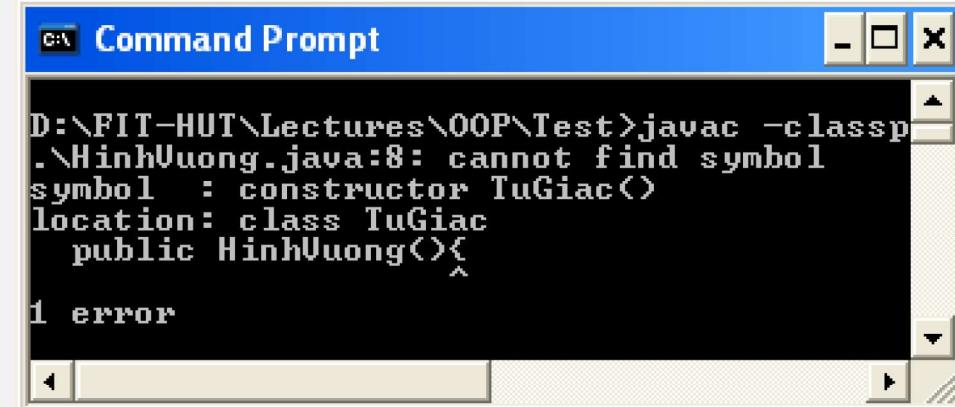
# Ví dụ

```
public class TuGiac {  
    protected Diem d1, d2;  
    protected Diem d3, d4;  
    public TuGiac() {  
        System.out.println("Lop cha TuGiac()");  
    }  
}  
  
public class HinhVuong extends TuGiac {  
    public HinhVuong() {  
        // Tu dong goi TuGiac()  
        System.out.println("Lop con HinhVuong()");  
    }  
}  
  
public class Test {  
    public static void main(String arg[]) {  
        HinhVuong hv = new HinhVuong();  
    }  
}
```



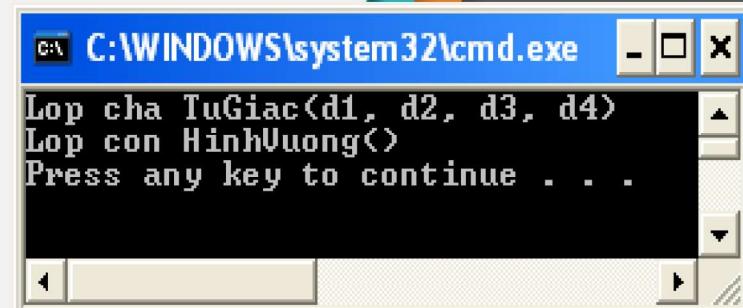
# Ví dụ

```
public class TuGiac {  
    protected Diem d1, d2;  
    protected Diem d3, d4;  
    public TuGiac(Diem d1, Diem d2, Diem d3, Diem d4) {  
        System.out.println("Lop cha TuGiac(d1, d2, d3, d4)");  
        this.d1 = d1; this.d2 = d2;  
        this.d3 = d3; this.d4 = d4;  
    }  
}  
  
public class HinhVuong extends TuGiac {  
    public HinhVuong() {  
        System.out.println("Lop con HinhVuong()");  
    }  
}  
  
public class Test {  
    public static void main(String arg[]) {  
        HinhVuong hv = new HinhVuong();  
    }  
}
```



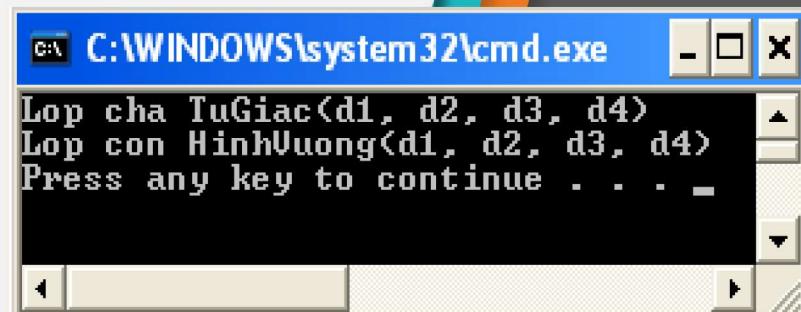
# Ví dụ

```
public class TuGiac {  
    protected Diem d1, d2, d3, d4;  
    public TuGiac(Diem d1, Diem d2, Diem d3, Diem d4) {  
        System.out.println("Lop cha TuGiac(d1, d2, d3, d4)");  
        this.d1 = d1; this.d2 = d2;  
        this.d3 = d3; this.d4 = d4;  
    }  
}  
  
public class HinhVuong extends TuGiac {  
    public HinhVuong() {  
        super(new Diem(0,0), new Diem(0,1),  
              new Diem(1,1),new Diem(1,0));  
        System.out.println("Lop con HinhVuong()");  
    }  
}  
  
public class Test {  
    public static void main(String arg[]) {  
        HinhVuong hv = new HinhVuong();  
    }  
}
```



# Ví dụ

```
public class TuGiac {  
    protected Diem d1, d2, d3, d4;  
    public TuGiac(Diem d1, Diem d2, Diem d3, Diem d4) {  
        System.out.println("Lop cha TuGiac(d1, d2, d3, d4)");  
        this.d1 = d1; this.d2 = d2;  
        this.d3 = d3; this.d4 = d4;  
    }  
}  
  
public class HinhVuong extends TuGiac {  
    public HinhVuong(Diem d1, Diem d2, Diem d3, Diem d4) {  
        super(d1, d2, d3, d4);  
        System.out.println("Lop con HinhVuong(d1, d2, d3, d4)");  
    }  
}  
  
public class Test {  
    public static void main(String arg[]) {  
        HinhVuong hv = new HinhVuong(  
            (new Diem(0,0), new Diem(0,1),  
             new Diem(1,1),new Diem(1,0)));  
    }  
}
```



# Thank you!

Any questions?