Chương 3

Thừa kế

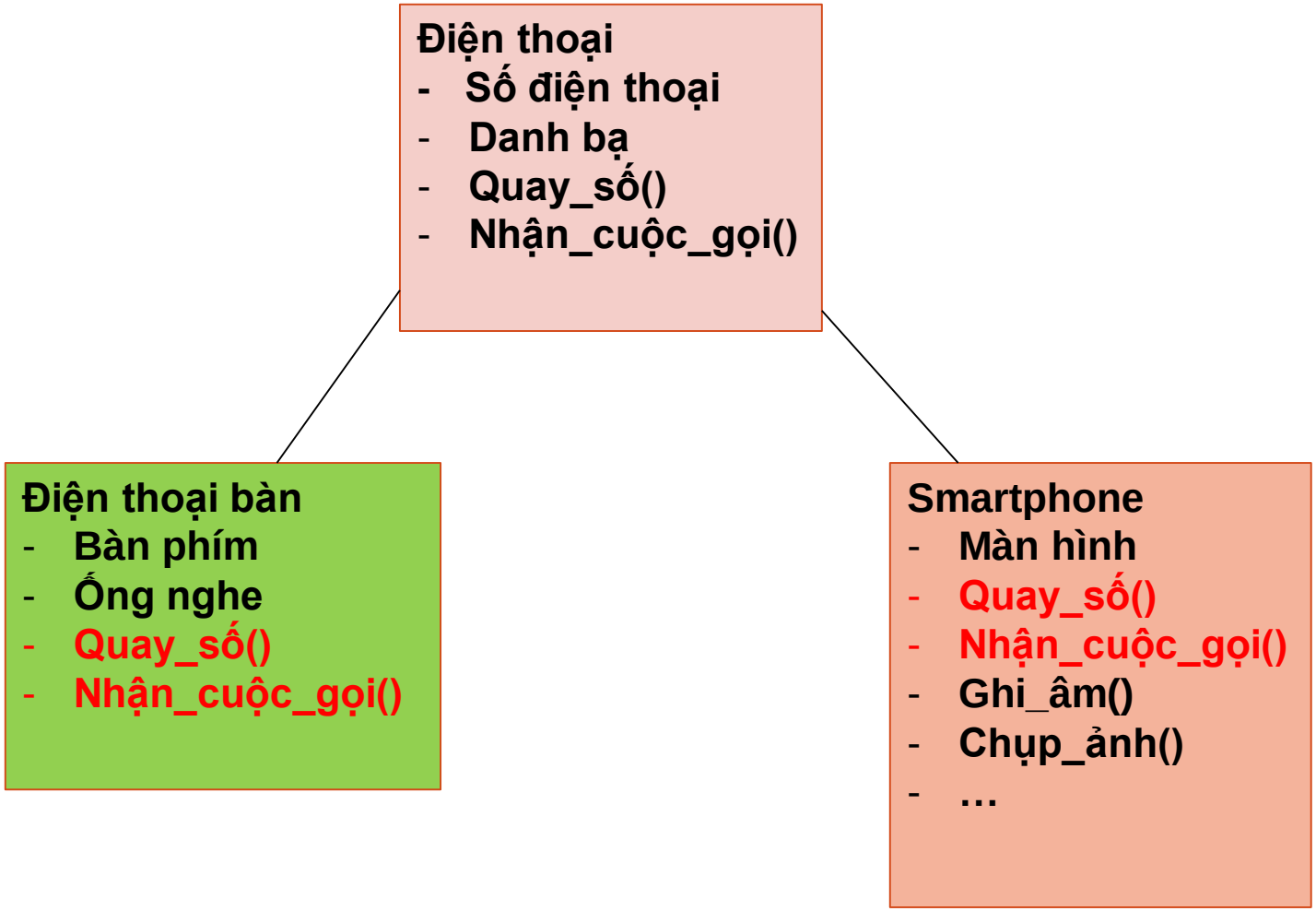
Giáo viên : ThS. Trần Văn Thọ
Đơn vị : Bộ môn KTHT & MMT

1. Giới thiệu

- Thừa kế là một trong 4 nguyên tắc cơ sở của Lập trình hướng đối tượng
- Tính sử dụng lại mã nguồn -> cần có cơ chế thâu tóm các đặc điểm chung ở các cấu trúc tương tự nhau.
- Tính mở rộng -> mỗi loại cấu trúc trong một nhóm tương tự nhau cũng có thể có các điểm khác biệt, cần có cơ chế kết hợp và mô tả các khác biệt này

-> Một lớp có thể là mở rộng, cá biệt hóa hoặc kết hợp từ các lớp khác -> **Thừa kế**

1. Giới thiệu


Điện thoại
- Số điện thoại
- Danh bạ
- Quay_số()
- Nhận_cuộc_gọi()
Điện thoại bàn
- Bàn phím
- Ống nghe
- Quay_số()
- Nhận_cuộc_gọi()
Smartphone
- Màn hình
- Quay_số()
- Nhận_cuộc_gọi()
- Ghi_âm()
- Chụp_ảnh()
- …

2. Đơn thừa kế

- Là dạng thừa kế mà một lớp chỉ thừa kế từ một (và chỉ một) lớp khác.
  - Lớp được thừa kế: lớp cơ sở (lớp cha)
  - Lớp thừa kế : lớp dẫn xuất (lớp con)
- Nguyên tắc:
  - Lớp dẫn xuất thừa hưởng toàn bộ các thuộc tính và phương thức từ lớp cơ sở ngoại trừ hàm thiết lập
  - Định nghĩa hàm thành phần trong lớp dẫn xuất **không được phép** truy cập đến các thành phần **private** của lớp cơ sở.

2. Đơn thừa kế

Cú pháp:

```
class D extends B{
….
};
```

B: lớp cơ sở

D: lớp dẫn xuất

2. Đơn thừa kế - Ví dụ

```
public class Person {
    private long id;
    private String name;
    public Person() {
        this.id=0;this.name="";
    }
    public Person(long id, String
  name) {
        this.id = id;
        this.name = name;
    }
    public void display(){

  System.out.println(id+","+name);
    }
}
```

```
public class Employee extends Person{
    private float salary;
  public Employee() {
        this.salary=0;
    }
    public Employee(float salary, long
  id, String name) {
        super(id, name);
        this.salary = salary;
    }

    public void display(){
        super.display();
        System.out.println("has
  salary:"+this.salary+"$");
    }
}
```

2. Đơn thừa kế

Từ khóa **super:**

- Đại diện cho thể hiện của lớp cha bên trong lớp con: cho phép truy cập đến các thành phần của lớp cha bên trong lớp con
    - + super.<tên thuộc tính> : truy cập đến thuộc tính của lớp cha
    - + super.<tên phương thức>() : truy cập đến hàm của lớp cha
    - + super(tham số) **:** gọi đến hàm khởi tạo của lớp cha

2.1. Định nghĩa lại các thành phần lớp cơ sở trong lớp dẫn xuất

- Định nghĩa lại Hàm thành phần (Overriding)

- Một hàm thành phần ở lớp cơ sở có thể được định nghĩa lại trong lớp dẫn xuất
- Trong lớp dẫn xuất, định nghĩa một hàm trùng tên và danh sách tham số với hàm ở lớp cơ sở
- Khi gọi hàm ở đối tượng thuộc lớp dẫn xuất, hàm định nghĩa lại sẽ **thay thế** hàm của lớp cơ sở.
- Hàm định nghĩa lại có thể gọi tới hàm thành phần cùng tên trong lớp cơ sở bằng từ khóa **super**

VD: hàm display() của lớp Employee là hàm định nghĩa lại hàm display() trong lớp cơ sở Person, trong định nghĩa có gọi đến hàm display() của lớp cơ sở thông qua lời gọi super.display()

2.1. Định nghĩa lại các thành phần lớp cơ sở trong lớp dẫn xuất

- **Biến thành phần:**
  - Cũng có thể định nghĩa lại ở lớp dẫn xuất -> khai báo các biến cùng tên với các biến ở lớp cơ sở (có thể khác kiểu)
  - Các biến định nghĩa lại ở lớp dẫn xuất sẽ thay thế các biến cùng tên ở lớp cơ sở. Có thể truy cập tới biến ở lớp cơ sở bằng từ khóa **super**

2.1. Định nghĩa lại các thành phần lớp cơ sở trong lớp dẫn xuất

VD:

```
public static void
  main(String[] args) {
  Person person=new
       Person(1,"An");
  person.display();
  Employee emp=new
       Employee(2,"Binh",1000);
  emp.display();
}
```

```
Kết quả:
1,An
2,Binh
has salary:1000.0$
```

2.2. Hàm thiết lập trong lớp dẫn xuất

- Lớp dẫn xuất không thừa hưởng hàm thiết lập ở lớp cơ sở, nó có hàm thiết lập riêng
- Khi hàm thiết lập ở lớp con được gọi, nó sẽ gọi **hàm thiết lập ngầm định** ở lớp cơ sở trước -> các thành phần dữ liệu thừa hưởng ở lớp cơ sở được khởi tạo trước, sau đó mới đến các thành phần dữ liệu mới định nghĩa
- Làm thế nào để gọi đến một hàm thiết lập cụ thể của lớp cơ sở ở định nghĩa hàm thiết lập lớp dẫn xuất -> sử dụng từ khóa **super** để chủ động gọi đến các hàm thiết lập ở lớp cơ sở

2.2. Hàm thiết lập trong lớp dẫn xuất - Ví dụ

```
public class Person {
    private long id;
    private String name;
    public Person() {

  System.out.println("constructor(
  ) of person");
        this.id=0;this.name="";
    }
    public Person(long id, String
  name) {
        this.id = id;
        this.name = name;
    }
    public void display(){

  System.out.println(id+","+name);
    }
}
```

```
public class Employee extends Person{
    private float salary;
  public Employee() {
        this.salary=0;
    }
    public Employee(float salary, long
  id, String name) {
        super(id, name);
        this.salary = salary;
    }

    public void display(){
        super.display();
        System.out.println("has
  salary:"+this.salary+"$");
    }
}
```

2.2. Hàm thiết lập trong lớp dẫn xuất - Ví dụ

VD:

```
public static void
  main(String[] args) {
  Employee emp1=new
  Employee();
        emp1.display();
        Employee emp2=new
  Employee(1,"An",1000);
        emp2.display();
}
```

```
Kết quả:
constructor() of person
0,
has salary:0.0$
1,An
has salary:1000.0$
```

3. Tính đa hình

3.1. Khái niệm:

Đa hình có nghĩa là một cấu trúc chương trình mang một diện mạo nào đó có thể có những ý nghĩa khác nhau tùy vào kiểu của toán hạng (hoặc tham số) liên quan.

Có 2 dạng đa hình:

- Định nghĩa chồng hàm: hàm được gọi tương ứng với danh sách tham số
- Sử dụng thừa kế

3. Tính đa hình

3.2 Tính đa hình trong phân cấp thừa kế:

Chuyện gì xảy ra nếu gán đối tượng thuộc lớp con cho đối tượng thuộc lớp cha?

Ví dụ:

```
Person person= new Employee(1,"An",1000);
person.display();
```

hoặc:

```
Employee emp=new Employee(1,"Bình",2000);
Person person=emp;
person.display();
//đối tượng person ở đây sẽ thể hiện nó là đối tượng lớp con hay lớp cha?
```

Ví dụ

```
public static void
main(String[] args) {
        Employee emp=new
Employee(1,"Bình",2000);
        Person person=emp;
        person.display();
    }
```

```
1,Bình
has salary:2000.0$
```

3. Tính đa hình

3.2 Tính đa hình trong phân cấp thừa kế:

- Việc gán đối tượng lớp con cho lớp cha thực chất là thay tham chiếu đến đối tượng lớp cha bằng tham chiếu đối tượng lớp con
- Khi gọi các hàm được định nghĩa lại ở lớp con, một biến thể hiện nó thuộc lớp nào phụ thuộc vào việc nó đang tham chiếu đến đối tượng nào trong bộ nhớ
  - Chú ý: danh sách thuộc tính và các hàm có thể gọi vẫn dựa trên khai báo của lớp mà đối tượng khai báo
- Một biến có thể tham chiếu đến bất kỳ đối tượng nào thuộc lớp nó khai báo hoặc thuộc lớp con của lớp khai báo.

VD:

```
Person p=new Person();
Employee e=new Employee(1,”An”,1000);
Person ptr=p;
…
ptr=e;
…
```

- Lưu ý: không thể gán đối tượng lớp cha cho đối tượng lớp con

3.3 Liên kết tĩnh – liên kết động

- **Liên kết tĩnh**: việc gắn kết giữa lời gọi hàm và hàm được gọi là cố định, không thay đổi, được quy định ngay khi biên dịch chương trình.
  - VD: định nghĩa chồng hàm, chồng toán tử,…
- **Liên kết động:** việc gắn kết giữa lời gọi hàm và hàm được gọi có thể thay đổi trong quá trình thực thi chương trình
- **Để tạo ra liên kết động cần:**
  - Phân cấp thừa kế
  - Sử dụng biến đối tượng của **lớp cha** để *tham chiếu* đến các đối tượng thuộc **lớp con**
  - Hàm thành phần phải được **định nghĩa lại** ở lớp con

4 Lớp trừu tượng

- **Hàm trừu tượng:**
  - Là hàm không có phần định nghĩa
  - Đóng vai trò đánh dấu cho hàm thực sự sẽ được khai báo ở các lớp con.
  - Cú pháp:

```
abstract <kiểu> tên_hàm(danh sách tham số);
```

- **Lớp trừu tượng:**
  - Không thể khai báo đối tượng thuộc lớp trừu tượng
  - Lớp con nếu không định nghĩa lại tất cả các hàm trừu tượng của lớp cha thì cũng phải trở thành lớp trừu tượng.
  - Cú pháp khai báo:

```
abstract class <tên lớp> {
    //abstract functions
}
```

4 Lớp trừu tượng

- **Lớp giao diện (interface):**
  - Là kiểu lớp trừu tượng đặc biệt:
  - Chỉ chứa các hàm không có phần định nghĩa
  - Thực chất là các hàm trừu tượng, tuy nhiên không cần sử dụng từ khóa abstract
  - Có thể chứa hằng (các thuộc tính có từ khóa **final**)
  - Tất cả các thành phần đều có phạm vi mặc định là **public**
  - Không thể khởi tạo đối tượng từ lớp giao diện
  - Sử dụng từ khóa **implements** khi muốn thừa kế từ lớp giao diện
  - Cho phép đa thừa kế : một lớp con có thể thừa kế từ nhiều lớp interface
  - Cú pháp khai báo:

```
interface <tên lớp giao diện>{
 //các hàm trừu tượng
}
```

Ví dụ

```
public abstract class
Organization {
    private String name;
    public abstract void
display();
}
…
public interface
ManagementAction {
    public void meeting();
    public void
addMember(Employee e);
    public void
removeMember(Employee e);
}
```

```
public class Director extends
Organization implements
ManagementAction {
  Employee members[];
  public void display() {
    System.out.println("display...");
  }

  public void meeting() {
    System.out.println("meeting...");
  }

  public void addMember(Employee e) {
   System.out.println("addmember...");
  }

 public void removeMember(Employee e){
System.out.println("deletemember...");
}

}
```

Bài tập

- Tạo lớp trừu tượng **Animal** có thuộc tính Name, các phương thức là: hàm tạo, hàm trừu tượng sayHello() và hàm trả về thuộc tính Name.
- Tạo 2 lớp **Cat** và **Dog** thừa kế từ lớp **Animal**. Viết hàm tạo và định nghĩa hàm trừu tượng sayHello() để chào hỏi theo các riêng của từng lớp.
- Tạo lớp **Zoo** để quản lý nhiều Animal, có thuộc tính là mảng các Animal, các phương thức là: thêm, xóa, hiển thị (các đối tượng trong mảng Animal gọi phương thứcsayHello).
- Viết hàm main tạo các đối tượng Cat, Dog và thêm chúng vào đối tượng Zoo, gọi phương thức hiển thị để minh họa.