●09.17_面向对象(接口的成员特点)

- A:接口成员特点

```
interface Inter {
    int num = 10;
}

class Demo implements Inter {
    public void print() {
        System.out.println(num);
    }
}
```

如果;

```
class Demo implements Inter {
    public void print() {
        num = 20;
        System.out.println(num);
    }
}
```

默认是常量 final 不加自动会加

- 成员变量；只能是常量，并且是静态的并公共的。
    - 默认修饰符：public static final
    - 建议：自己手动给出。

```
Demo d = new Demo();
d.print();
System.out.println(Inter.num);
```

静态的 也可以 其实是这样的

```
public static final int num = 10;
```

- 构造方法：接口没有构造方法。

```
//public Inter(){}
```

```
class Demo extends Object implements Inter {
    public void print() {
        //num = 20;
        System.out.println(num);
    }

    public Demo() {
        super();
    }
}
```

接口是干爹 不是亲爹 默认访问是 Object 中的 super

```
//一个类不写继承任何类,默认继承Object类
```

- 成员方法：只能是抽象方法。

```
/*public void print() {                    接口中不能定义非抽象方法

}*/
```

- 默认修饰符：public abstract
- 建议：自己手动给出。

```
public abstract void print();
```

- B:案例演示
  - 接口成员特点

```
public class Demo2 {
    public static void main(String[] args) {

    }
}
interface A{
    public void fly();
}
interface B extends A{
    public void run();
}
interface C{
    public void cry();
}
interface D extends A,C{  //接口可以多继承,不能实现接口
   public void jump();
}
class Dog implements D{
```

```
    @Override
    public void fly() {
        System.out.println("飞");
    }

    @Override
    public void cry() {
        System.out.println("旺旺");
    }

    @Override
    public void jump() {
        System.out.println("狗急跳墙");
    }
}
//适配器的应用场景:如果要使用接口中的某些功能,可写一个适配器,实现接口,然后把自己的类继承这个适配器类就可以了
class K_Adapter implements D{

    @Override
    public void fly() {

    }

    @Override
    public void cry() {

    }

    @Override
    public void jump() {

    }
}
class Dog2 extends K_Adapter{
    @Override
    public void cry() {
        System.out.println("旺旺");
    }

    @Override
    public void jump() {
        System.out.println("狗跳");
    }
}
```

●09.18_面向对象(类与类,类与接口,接口与接口的关系)

- A:类与类,类与接口,接口与接口的关系
    - a:类与类:
        - 继承关系,只能单继承,可以多层继承。
    - b:类与接口:
        - 实现关系,可以单实现,也可以多实现。
        - 并且还可以在继承一个类的同时实现多个接口。
    - c:接口与接口:
        - 继承关系,可以单继承,也可以多继承。
- B:案例演示
    - 类与类,类与接口,接口与接口的关系

```
public class Demo2 {
    public static void main(String[] args) {

    }
}
interface A{
    public void fly();
}
interface B extends A{
    public void run();
}
interface C{
    public void cry();
}
interface D extends A,C{  //接口可以多继承,不能实现接口
   public void jump();
}
class Animal{
    public void eat(){
        System.out.println("动物吃东西");
    }
}
class Dog extends Animal implements D {  //一个类既可以继承父类,又可以实现接口

   @Override
    public void fly() {
        System.out.println("飞");
    }
```

```
    @Override
    public void cry() {
        System.out.println("旺旺");
    }

    @Override
    public void jump() {
        System.out.println("狗急跳墙");
    }
}
```

●09.19_面向对象(抽象类和接口的区别)

- A:成员区别
    - 抽象类：
        - 成员变量：可以变量，也可以常量
        - 构造方法：有 （被子类初始化使用的）
        - 成员方法：可以抽象，也可以非抽象(可以被子类继承下来使用)

    结论：抽象类和普通类差不多该怎么定义就怎么定义，只是增加了强制子类去重写的抽象方法
    - 接口：
        - 成员变量：是常量
        - 成员方法：只可以抽象

- B:关系区别
    - 类与类
        - 继承，单继承
    - 类与接口
        - 实现，单实现，多实现
    - 接口与接口

        - 继承，单继承，多继承

- C:设计理念区别
    - 抽象类 被继承体现的是:”is it”的关系。抽象类中定义的是该继承体系的共同功能。
    - 接口 被实现体现的是：”like it”的关系。接口中定义的是该继承体系的扩展功能。

●09.20_面向对象(猫狗案例)

- A:案例演示
    - 动物类：姓名，年龄，吃饭，睡觉。(共同功能)
    - 猫和狗
    - 动物实现接口：跳高 (扩展功能)

```
public class Demo1 {
    public static void main(String[] args) {

        Dog d = new Dog("小黄",2);
        d.eat();
        d.jump();
        d.sleep();
    }

}
/*
动物类：姓名，年龄，吃饭，睡觉。(共同功能)
    - 猫和狗
    - 动物实现接口：跳高 (扩展功能)
 */
class Animal{
    String name;
    int age;
    public Animal(){}
    public Animal(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public void eat(){
        System.out.println(name+"吃东西");
    }
    public void sleep(){
```

```
        System.out.println(age+"睡觉");
    }
}
interface Sport{
    public void jump();

}
class Dog extends Animal implements Sport{
    public Dog(String name,int age){
        super(name,age);
    }
    @Override
    public void jump() {
        System.out.println("爱跳高");
    }
}
```