接口（interface）

接口

- 接口是抽象方法和常量值的定义的集合
- 从本质上讲，接口是一种特殊的抽象类，这种抽象类只包含常量和方法的定义，没有变量和方法的实现
- 多个类可以实现同一接口
- 一个类可以实现多个接口
- 与继承关系相似，接口与实现类之间存在多态关系

接口声明

```
interface name{
    成员变量;
    方法声明;
}
```

- 接口中声明属性默认为public static final，也只能是public static final
    - static
        - 如果接口中的成员变量是非静态的，那么每一个实现了该接口的类都会有一个变量。由于接口是可以多继承，如果另一个接口也有一个同样名字的变量，那么在使用的时候就必须通过 接口.名字或对象.名字 调用。
    - final
        - 如果不是final，那么意味着每一个实现该接口的子类都可以修改这个变量。如果随便修改的话，那么其他也继承了该接口的类就会受到影响。
- 接口中只能定义抽象方法，而这些方法默认为public，也只能是public
    - 如果一个类实现两个接口，并且这两个接口中具有的方法名并且返回类型一致，则编译运行正确，这两个接口都可以使用这个方法。
    - 如果一个类实现两个接口，并且这两个接口中具有的方法名并且返回类型一致，则编译运行报错。
- 接口可以继承其他的接口，并添加新的属性和抽象方法

```
interface Singer{
    public void sing();
    public void sleep();
}
interface Painter{
    public void paint();
    public void eat();
}
```

```
class Student implements Singer{
```

```
    private String name;

    @Override
    public void sing() {
        System.out.println("student is singing");
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    @Override
    public void sleep() {
        System.out.println("student is sleeping");

    }
}
```

```
class Teacher implements Singer,Painter{

    @Override
    public void sing() {
        System.out.println("teacher is singing");
    }

    @Override
    public void sleep() {
        System.out.println("teacher is sleeping");
    }

    @Override
    public void paint() {
        System.out.println("teacher is patiting");
    }

    @Override
    public void eat() {
        System.out.println("teacher is eating");
    }
}
```

```
public class test02 {
    public static void main(String[] args) {
        /**
```

```
        * 接口引用指向父类对象
        * 思考:
        *      singer能否访问paint(),eat()方法,
        */
        Singer singer = new Student();
        singer.sing();
        singer.sleep();

        Teacher teacher = new Teacher();
        teacher.eat();
        teacher.paint();
        teacher.sing();
        teacher.sleep();


        Singer singer1 = new Teacher();
        singer1.sing();
        singer1.sleep();

        /**
        * 思考:
        *      painter能否访问sing(),sleep()方法
        */
        Painter painter = new Teacher();
        painter.eat();
        painter.paint();

        /**
        * 向下转型
        */
        ((Teacher) singer).sleep();

    }
}
```

实例二:

```
interface Valuable{
    public double getMoney();
}
interface Protectable{
    public void beProtected();
    public double getMoney();
}
abstract class Animal{
    private String name;
    abstract void enjoy();
}
```

```
class GoldenMonkey extends Animal implements Valuable,Protectable{

    @Override
    public double getMoney() {
        return 666;
    }


    @Override
    public void beProtected() {
        System.out.println("the animal is be protected");
    }

    @Override
    void enjoy() {
        System.out.println("the animal live happlily");
    }
}
```

```
public class test03 {
    public static void main(String[] args) {
//        GoldenMonkey goldenMonkey = new GoldenMonkey();

        /**
         * 以下皆是父类引用指向子类对象
         */
        Valuable valuable = new GoldenMonkey();
        System.out.println(valuable.getMoney());
        System.out.println("transfer the same name by object valuable:  " + valuable.num);

        Protectable protectable = new GoldenMonkey();
        protectable.beProtected();
        System.out.println( protectable.getMoney());
        System.out.println("transfer the same by object protectable:  " + protectable.num);

        Animal animal = new GoldenMonkey();
        animal.enjoy();
//        System.out.println(num);

        System.out.println("transfer the same by className Valuable:  " + Valuable.num);
        System.out.println("transfer the same by className Protectable:  " + Protectable.num);

    }
}
```

输出:

```java
package 面向对象编程.接口;
interface Valuable{
    public double getMoney();
    public static final int num = 1;
}
interface Protectable{
    public void beProtected();
    public double getMoney();
    public static final int num = 2;
}
abstract class Animal{
    private String name;
    abstract void enjoy();
}

class GoldenMonkey extends Animal implements Valuable,Protectable{

    @Override
    public double getMoney() { return 666; }


    @Override
    public void beProtected() { System.out.println("the animal is be protected"); }

    @Override
    void enjoy() { System.out.println("the animal live happlily"); }
}
public class test03 {
    public static void main(String[] args) {
//        GoldenMonkey goldenMonkey = new GoldenMonkey();

        /**
         * 以下皆是父类引用指向子类对象
         */
        Valuable valuable = new GoldenMonkey();
        System.out.println(valuable.getMoney());
        System.out.println("transfer the same name by object valuable:  " + valuable.num);

        Protectable protectable = new GoldenMonkey();
        protectable.beProtected();
        System.out.println( protectable.getMoney());
        System.out.println("transfer the same by object protectable:  " + protectable.num);

        Animal animal = new GoldenMonkey();
        animal.enjoy();
//        System.out.println(num);

        System.out.println("transfer the same by className Valuable:  " + Valuable.num);
        System.out.println("transfer the same by className Protectable:  " + Protectable.num);

    }
```

正确，方法名一样，返回类型也一样

```java
package 面向对象编程.接口;
interface Valuable{
    public double getMoney();
    public static final int num = 1;
}
interface Protectable{
    public void beProtected();
    public int getMoney();
    public static final int num = 2;
}
abstract class Animal{
    private String name;
    abstract void enjoy();
}

class GoldenMonkey extends Animal implements Valuable,Protectable{

    @Override
    public double getMoney() { return 666; }


    @Override
    public void beProtected() { System.out.println("the animal is be protected"); }

    @Override
    void enjoy() { System.out.println("the animal live happlily"); }
}
```

报错，方法名一样，但是返回类型不一样