第3天 面向对象

今日内容介绍

* 接口
* 多态
* 笔记本案例

**今日学习目标**

* 写出定义接口的格式
* 写出实现接口的格式
* 说出接口中成员的特点
* 接口和抽象类的区别\*\*\*\*\*\*\*\*
* 能够说出使用多态的前提条件
* 理解多态的向上转型
* 理解多态的向下转型
* 能够完成笔记本电脑案例（方法参数为接口）

# 接口

## 接口概念

类:具有相同属性和功能的事物集合

接口是功能的集合，同样可看做是一种数据类型，是比抽象类更为抽象的”类”。

接口只描述所应该具备的方法，并没有具体实现，具体的实现由接口的实现类(相当于接口的子类)来完成。这样将功能的定义与实现分离，优化了程序设计。

请记住：一切事物均有功能，即一切事物均有接口。

## 接口的定义

与定义类的class不同，接口定义时需要使用interface关键字。

定义接口所在的仍为.java文件，虽然声明时使用的为interface关键字的编译后仍然会产生.class文件。这点可以让我们将接口看做是一种只包含了功能声明的特殊类。

定义格式：

public interface 接口名 {

抽象方法1;

抽象方法2;

抽象方法3;

}

使用interface代替了原来的class，其他步骤与定义类相同：

* 接口中的方法均为公共访问的抽象方法
* 接口中无法定义普通的成员变量

## 类实现接口

类与接口的关系为实现关系，即类实现接口。实现的动作类似继承，只是关键字不同，实现使用implements。

其他类(实现类)实现接口后，就相当于声明：”我应该具备这个接口中的功能”。实现类仍然需要重写方法以实现具体的功能。

格式：

class 实现类 implements 接口 {

重写接口中所有方法

}

在类实现接口后，该类就会将接口中的抽象方法继承过来，此时该类需要重写该抽象方法，完成具体的逻辑。

* 接口中定义功能，当需要具有该功能时，可以让类实现该接口，只声明了应该具备该方法，是功能的声明。
* 在具体实现类中重写方法，实现功能，是方法的具体实现。

于是，通过以上两个动作将功能的声明与实现便分开了。(此时请重新思考：类是现实事物的描述，接口是功能的集合。)

## 接口中成员的特点

1、接口中可以定义成员变量，但是变量必须有固定的修饰符修饰，public static final 所以接口中的变量也称之为常量，其值不能改变。后面我们会讲解static与final关键字

2、接口中可以定义方法，方法也有固定的修饰符，public abstract

3、接口不可以创建对象。

4、实现类必须覆盖掉接口中所有的抽象方法后，实现类才可以实例化。否则实现类是一个抽象类。

interface Demo { ///定义一个名称为Demo的接口。

public static final int NUM = 3;// NUM的值不能改变

public abstract void show1();

public abstract void show2();

}

//定义子类去覆盖接口中的方法。类与接口之间的关系是 实现。通过 关键字 implements

class DemoImpl implements Demo { //子类实现Demo接口。

//重写接口中的方法。

public void show1(){}

public void show2(){}

}

## 接口特点

* 接口可以继承接口

如同类继承类后便拥有了父类的成员，可以使用父类的非私有成员。A接口继承B接口后，A接口便拥有了A、B两个接口中所有的抽象方法。

* Java支持一个类同时实现多个接口，或一个接口同时继承多个接口。
* 类可以在继承一个类的同时，实现多个接口。
* 接口与父类的功能可以重复，均代表要具备某种功能，并不冲突。

## 接口和抽象类的区别\*\*\*

明白了接口思想和接口的用法后，接口和抽象类的区别是什么呢？接口在生活体现也基本掌握，那在程序中接口是如何体现的呢？

通过实例进行分析和代码演示抽象类和接口的用法。

1、举例：

犬：

行为：

吼叫；

吃饭；

缉毒犬：

行为：

吼叫；

吃饭；

缉毒；

2、思考：

由于犬分为很多种类，他们吼叫和吃饭的方式不一样，在描述的时候不能具体化，也就是吼叫和吃饭的行为不能明确。当描述行为时，行为的具体动作不能明确，这时，可以将这个行为写为抽象行为，那么这个类也就是抽象类。

可是当缉毒犬有其他额外功能时，而这个功能并不在这个事物的体系中。这时可以让缉毒犬具备犬科自身特点的同时也有其他额外功能，可以将这个额外功能定义接口中。

如下代码演示：

interface 缉毒{

public abstract void 缉毒();

}

//定义犬科的这个提醒的共性功能

abstract class 犬科{

public abstract void 吃饭();

public abstract void 吼叫();

}

// 缉毒犬属于犬科一种，让其继承犬科，获取的犬科的特性，

//由于缉毒犬具有缉毒功能，那么它只要实现缉毒接口即可，这样即保证缉毒犬具备犬科的特性，也拥有了缉毒的功能

class 缉毒犬 extends 犬科 implements 缉毒{

public void 缉毒() {

}

void 吃饭() {

}

void 吼叫() {

}

}

class 缉毒猪 implements 缉毒{

public void 缉毒() {

}

}

3、通过上面的例子总结接口和抽象类的区别：

**相同点:**

* 都位于继承的顶端,用于被其他类实现或继承;
* 都不能直接实例化对象;
* 都可以包含抽象方法,其子类都必须覆写这些抽象方法;

**区别:**

* 抽象类为部分方法提供实现,避免子类重复实现这些方法,提高代码重用性;
* 接口只能包含抽象方法;
* 一个类只能继承一个直接父类(可能是抽象类),却可以实现多个接口;(接口弥补了Java的单继承)
* 抽象类为继承体系中的共性内容,接口为继承体系外的扩展功能

二者的选用:

* 优先选用接口,尽量少用抽象类;
* 需要定义子类的行为,又要为子类提供共性功能时才选用抽象类;

# 多态

## 多态概述

多态是继封装、继承之后，面向对象的第三大特性。

现实事物经常会体现出多种形态，如学生，学生是人的一种，则一个具体的同学张三既是学生也是人，即出现两种形态。

Java作为面向对象的语言，同样可以描述一个事物的多种形态。如Student类继承了Person类，一个Student的对象便既是Student，又是Person。

Java中多态的代码体现在一个子类对象(实现类对象)既可以给这个子类(实现类对象)引用变量赋值，又可以给这个子类(实现类对象)的父类(接口)变量赋值。

如Student类可以为Person类的子类。那么一个Student对象既可以赋值给一个Student类型的引用，也可以赋值给一个Person类型的引用。

最终多态体现为父类引用变量可以指向子类对象。

多态的前提是必须有子父类关系或者类实现接口关系，否则无法完成多态。

在使用多态后的父类引用变量调用方法时，会调用子类重写后的方法。

## 多态代码体现

Java中多态的代码体现在一个子类对象(实现类对象)既可以给这个子类(实现类对象)引用变量赋值，又可以给这个子类(实现类对象)的父类(接口)变量赋值。

如Student类可以为Person类的子类。那么一个Student对象既可以赋值给一个Student类型的引用，也可以赋值给一个Person类型的引用。

最终多态体现为父类引用变量可以指向子类对象。

多态的前提是必须有子父类关系或者类实现接口关系，否则无法完成多态。

在使用多态后的父类引用变量调用方法时，会调用子类重写后的方法。

具体格式如下：

父类引用指向子类对象就是多态的定义格式。同一个父类的方法会被不同的子类重写为各自的具体实现。在调用方法时，调用的为各个子类重写后的方法。

父类类型 变量名 = new 子类类型();

变量名.方法名();

此时，虽然该变量指向的是子类对象，但表现为一个父类的形态，可以调用一切父类的方法，子类特有的方法将不能调用。

## 多态调用注意事项

* 成员变量编译看父类中是否存在,不存在编译失败
* 成员变量运行父类中的变量
* 成员方法编译看父类中是否存在,不存在编译失败
* 成员方法运行子类重写的方法

## 多态的好处和弊端(猫狗的案例)\*\*

当变量名指向不同的子类对象时，由于每个子类重写父类方法的内容不同，所以会调用不同的方法。

如：

在Boss类中，有叫员工去工作的方法，当该方法的参数定义为接口时，可以传入任意的子类对象。相比定义多个子类参数，定义多个方法，这样大大提高了代码复用性与扩展性。

class Boss{

public void goToWork(Empolyee e){

e.work();

}

}

所以多态的存在意义(优点)为：

配合继承与方法重写提高了代码的复用性与扩展性，如果没有方法重写，则多态同样没有意义。

多态的弊端: 不能调用子类的特有方法

## 向上向下类型转换

多态本身是子类类型向父类类型向上转型的过程。

多态的转型分为向上转型与向下转型两种：

* 向上转型：当有子类对象赋值给一个父类引用时，便是向上转型，多态本身就是向上转型的过程。

使用格式：

父类类型 变量名 = new 子类类型();

如：Animal p = new Cat();

* 向下转型：一个已经向上转型的子类对象可以使用强制类型转换的格式，将父类引用转为子类引用，这个过程是向下转型。如果是直接创建父类对象，是无法向下转型的！

使用格式：

子类类型 变量名 = (子类类型) 父类类型的变量;

如:Cat c = (Cat) a; //变量p 实际上指向Cat对象

* instanceof关键字

使用格式：

boolean b = 引用变量 instanceof 类;

if(a instanceof Dog){

Dog d = (Dog)a;

}

# 笔记本电脑案例

## 案例介绍

定义USB接口（具备开启功能、关闭功能），笔记本要使用USB设备，即笔记本在生产时需要预留可以插入USB设备的USB接口，即就是笔记本具备使用USB设备的功能，但具体是什么USB设备，笔记本并不关心，只要符合USB规格的设备都可以。鼠标和键盘要想能在电脑上使用，那么鼠标和键盘也必须遵守USB规范，不然鼠标和键盘的生产出来无法使用

进行描述笔记本类，实现笔记本使用USB鼠标、USB键盘

* USB接口，包含开启功能、关闭功能
* 笔记本类，包含运行功能、关机功能、使用USB设备功能
* 鼠标类，要符合USB接口
* 键盘类，要符合USB接口

## 案例需求分析

阶段一：

使用笔记本，笔记本有运行功能，需要笔记本对象来运行这个功能

阶段二：

想使用一个鼠标，又有一个功能使用鼠标，并多了一个鼠标对象。

阶段三：

还想使用一个键盘 ，又要多一个功能和一个对象

问题：每多一个功能就需要在笔记本对象中定义一个方法，不爽，程序扩展性极差。

降低鼠标、键盘等外围设备和笔记本电脑的耦合性。

## 实现代码步骤

* 定义鼠标、键盘，笔记本三者之间应该遵守的规则

interface USB {

void open();// 开启功能

void close();// 关闭功能

}

* 鼠标实现USB规则

class Mouse implements USB {

publicvoid open() {

System.*out*.println("鼠标开启");

}

publicvoid close() {

System.*out*.println("鼠标关闭");

}

}

* 键盘实现USB规则

class KeyBoard implements USB {

publicvoid open() {

System.*out*.println("键盘开启");

}

publicvoid close() {

System.*out*.println("键盘关闭");

}

}

* 定义笔记本

class NoteBook {

// 笔记本开启运行功能

publicvoid run() {

System.*out*.println("笔记本运行");

}

// 笔记本使用usb设备，这时当笔记本对象调用这个功能时，必须给其传递一个符合USB规则的USB设备

publicvoid useUSB(USB usb) {

// 判断是否有USB设备

if (usb != null) {

usb.open();

usb.close();

}

}

publicvoid shutDown() {

System.*out*.println("笔记本关闭");

}

}

publicclass Test {

publicstaticvoid main(String[] args) {

// 创建笔记本实体对象

NoteBook nb = new NoteBook();

// 笔记本开启

nb.run();

// 创建鼠标实体对象

Mouse m = new Mouse();

// 笔记本使用鼠标

nb.useUSB(m);

// 创建键盘实体对象

KeyBoard kb = new KeyBoard();

// 笔记本使用键盘

nb.useUSB(kb);

// 笔记本关闭

nb.shutDown();

}

}