Day06

java课程-李彦伯

抽象类

> 当我们定义一个父类的时候,有些方法可能需要子类去做具体的功能,而在父类中并不能确定做哪些功能,那么我们就可以将这些方法定义成抽象方法.所谓的抽象方法就是没有方法体的方法.那么我们的类也必须定义这个类为抽象的类.

抽象类的格式

- 抽象方法和抽象类需要使用特定的修饰符进行修饰,这个修饰符就是`abstract` - 抽象类的格式为 `public abstract class 类名 { }`
- 抽象方法的格式为 `public abstract 返回值类型 方法名(参数);`

抽象类的特点

- 抽象类和抽象方法都需要被abstract修饰。抽象方法一定要定义在抽象类中。
- 抽象类不可以直接创建对象，原因：调用抽象方法没有意义。
- 只有覆盖了抽象类中所有的抽象方法后，其子类才可以创建对象。否则该子类得定义成抽象类。
- 子类重写父类的抽象方法的时候要去掉abstract关键字

抽象类的注意问题

- 抽象类因为需要子类去实现,所以肯定得是一个父类
- 抽象类可以不定义抽象方法,意义在于,不让该类创建对象,方法可以让子类使用
- 一个方法如果没有方法体,则必须使用`abstract`进行修饰

> 抽象关键字abstract不可以和哪些关键字共存？
>
> 1. private：私有的方法子类是无法继承到的，也不存在覆盖，而abstract和private一起使用修饰方法，abstract既要子类去实现这个方法，而private修饰子类根本无法得到父类这个方法。互相矛盾。
> 2. final:因为被final定义的方法不能被重写,而抽象方法需要子类去重写进行实现功能,所以矛盾
> 3. static:因为static不用实例化可直接调用,即当前类就直接能够调用了,而abstract需要子类实现后才能调用,所以矛盾

- 作业

定义一个类是员工,员工有id和name属性,有一个方法是eat调用eat所有员工输出吃饭,有一个抽象方法是work,不同的员工工作的内容不同,定义3个类,程序员,老师,网管,分别调用work的时候”id为xx的name写代码”,”讲课”,”拉网线”

```java
public abstract class Staff {
    private int id;
    private String name;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

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

    public abstract void work();

}
class Teacher extends Staff{

    //因为父类的get方法是public,所以子类也有get方法,在本类中可以直接调用方法
    public void work() {
        System.out.println("id为"+getI
```

```
d()+getName()+"讲课");
    }


}
class Developer extends Staff{
    public  void work(){
        System.out.println("id为"+getI
d()+getName()+"写代码");
    }

}
class NetManager extends Staff{
    public  void work(){
        System.out.println("id为"+getI
d()+getName()+"拉网线");
    }
}
```

```
public class StaffTest {
    public static void main(String[] arg
s) {
        Developer de = new Developer();
        Teacher te = new Teacher();
        NetManager nm = new NetManager();
        de.setId(1);
        de.setName("李雷");
        te.setId(2);
        te.setName("韩梅梅");
        nm.setId(3);
        nm.setName("露西");
        de.work();
        te.work();
        nm.work();
    }
}
```

接口

1. 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”。
2. 接口中只允许出现抽象方法,不能出现非抽象方法
3. 接口的源文件也是java文件,编译后的文件也是.class,所以可以把接口看作一个特殊的类

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
- 定义格式:

```
public interface 接口名 {
    抽象方法1;
    抽象方法2;
    抽象方法3;
}
```

- 因为接口是供别的类去实现的,并且要实现他的抽象方法的,所以接口中方法的访问修饰符必须定义成public
- 接口中不能定义普通的成员变量,只能定义成public static final 成员变量

> 注意如果定义方法的时候,不写public abstract,语法不会报错,但是系统会默认添加;定义静态最终成员变量的时候如果不写public static final,语法不会报错,但是系统同样会默认添加

接口的实现

- 一个类实现接口的格式为:

```
class 类 implements 接口1,接口2 {
    重写接口中方法
}
```

- 一个类可以同时实现多个接口,但是需要将抽象方法进行实现,创建接口的时候修饰符可以省略, 系统会自动加上,但是实现的时候,就必须要添加public

接口的特点

- 接口不能创建对象
- 接口的变量使用public static final修饰,如果不写默认添加
- 接口的方法为public abstrict,如果不写默认添加
- 子类必须重写接口中所有的抽象方法后,才能创建对象,如果子类不不能够重写所有的抽象方法,那么子类必须定义成抽象类

接口可以多继承(了解即可)

- 因为接口中的方法都是没有实现的,所以就算有两个父接口的方法名称相同,子类实现接口的时候也不会有什么影响,重写即可,所以接口与接口之间是存在多继承的

```
interface MyInterface1{
    void method01();
}
interface MyInterface2{
    void method02();
}
interface MyInterface3{
    void method03();
}
interface MyInterface4 extends MyInterfac
e1,MyInterface2,MyInterface3{
    void method04();
}
```

接口的意义

- 扩展原有类的功能
- 设定了规则
- 降低耦合性

接口和抽象类区别(面试题)!!!

- 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:
  - 抽象类可以定义非抽象方法,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)
  - 类与类之间只能是单继承(包括抽象类),接口与接口之间可以是多实现
  - 抽象类可以定义普通的成员变量,接口只能定义public static final的成员变量

- 作业

```java
public abstract class Staff {
    private int id;
    private String name;
    public int getId() {
        return id;
    }
    public void setId(int id) {
        this.id = id;
    }
    public String getName() {
        return name;
    }
    public void setName(String name) {
        this.name = name;
    }

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

    public abstract void work();

}
class Teacher extends Staff implements Coding{

    //因为父类的get方法是public,所以子类也有get方法,在本类中可以直接调用方法
    public void work() {
```

```
            System.out.println("id为"+getI
d()+getName()+"讲课");
        }

        @Override
        public void codeProject() {
            System.out.println("老师在写项目");
        }
}
class Developer  extends Staff implements
Coding{
        public  void work(){
            System.out.println("id为"+getI
d()+getName()+"写代码");
        }

        @Override
        public void codeProject() {
            System.out.println("程序员在写项
目");
        }
}

class NetManager extends Staff{
        public  void work(){
            System.out.println("id为"+getI
d()+getName()+"拉网线");
        }
}
```

```
interface Coding{
    public abstract void codeProject();
}
```

```
public class StaffTest {
    public static void main(String[] arg
s) {
        Developer de = new Developer();
        Teacher te = new Teacher();
        NetManager nm = new NetManager();
        de.setId(1);
        de.setName("李雷");
        te.setId(2);
        te.setName("韩梅梅");
        nm.setId(3);
        nm.setName("露西");
        de.work();
        te.work();
        nm.work();
        de.codeProject();
        te.codeProject();
    }
}
```

多态

- 通过继承我们实现了,男人和女人都是人,一个人的类,出

现了多种形态,我们称之为这种现象就是java的多态

> 多态的表现形式为父类的变量指向子类的对象
> 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。
> 父类类型的变量调用方法的时候,实际上会调用子类重写的方法

定义格式

- 普通类多态定义的格式 `父类 变量名 = new 子类();`

```
class Fu {}
class Zi extends Fu {}
//类的多态使用
Fu f = new Zi();
```

- 抽象类多态定义的格式:`抽象类 变量名 = new 抽象类子类();`

```
abstract class Fu {
    public abstract void method();
        }
class Zi extends Fu {
    public void method(){
            System.out.println(“重写父类抽象
方法”);
         }
}
//抽象类的多态使用
Fu fu= new Zi();
```

- 接口多态定义的格式:`接口 变量名 = new 接口实现类();`

```
interface Fu {
       public abstract void method();
}
class Zi implements Fu {
      public void method(){
               System.out.println(“重写接口
抽象方法”);
         }
}
//接口的多态使用
Fu fu = new Zi();
```

> 注意:同一个父类的方法会被不同的子类重写。在调用方法时，调用的为各个子类重写后的方法。

多态的特点

- 对于成员方法:编译的时候看=左边,运行的时候看=右边
- 对于成员变量:编译的时候看=左边,运行的时候看=左边

多态的转型

- 如果是多态的话,程序在进行编译的时候只会看变量的类型,而实际运行的时候则是看具体的对象,那么如果想要调用父类中没有而子类中有的方法怎么办?
- 可以使用强制转换,也叫做多态的向下转型,将父类类型再转换为子类类型`子类类型 变量 = （子类类型） 变量名称`
- 注意:如果强制转换的话,必须要保证变量原始就是什么类型,否则就会出现异常

instanceof关键字

- 如果直接进行强制转换的时候有可能会出现问题,所以可以使用instanceof进行判断如`对象 instanceof 类名`

```
if(a instanceof Zi){
    Zi z = (Zi) a;
}
```

- 作业

> 定义一个类是People,定义两个类,一个是Man,一个是Woman,Man中有个属性是老婆,有一个方法lol,Woman中有个属性是老公,有一个方法shopping,还有一个方法生孩子,先判断是否有老公,如果有,就创建一个对象50%概率是man,50%概率是woman,有一个返回值,最后如果是男的调用lol,如果是nv的调用shopping

```
public class People {
}
```

```java
public class Man extends People{
//定义属性的位置
//定义属性的类型
Woman wife;
//定义方法的位置?还是定义方法格式?
    public void lol(){
        System.out.println("lol");
    }
}
```

```
public class Woman extends People{
    Man husband;

    public void shopping(){
        System.out.println("购物");
    }
    //返回值类型是Man或者Woman，所以定义成People
    //根据逻辑我们需要在方法中创建一个Man的对象或者Woman的对象，所以不需要传递参数
    /*
     * 1.判断有没有老公
     * 2.如果条件成立执行生孩子的逻辑
     * 3. 如果不成立，程序结束  return  null;
     * 4.50%搞不定，问老师
     *
     *
     *
     */

    public People havaBaby(){
        People pe = null;
        if(husband != null){
            Random ran = new Random();
            if(ran.nextInt(2) == 0){
                pe = new Man();
            }else{
                pe = new Woman();
```

```
                }
            }
            return pe;
        }

    }
```

```
public class TestHomework {
    public static void main(String[] arg
s) {

        Woman wm = new Woman();
        Man man = new Man();
        wm.husband = man;

        People pe = wm.havaBaby();
        if(pe == null){
            System.out.println("没老公，生
个毛线啊");
            return;
        }

        if(pe instanceof Man){
            Man m = (Man)pe;
            m.lol();

        }else{
            Woman m = (Woman)pe;
            m.shopping();
        }
    }

}
```

构造方法

> 如果我们希望在创建对象的时候直接就能对对象的成员变量进行赋值,就需要创建构造函数,构造函数的目的就是在对象创建的时候给对象的成员变量进行赋值

构造方法的格式

```
修饰符  构造方法名(参数列表){

}
```

构造方法的特点

- 构造方法名称必须和类型保持一致。
- 构造方法没有具体的返回值。

默认构造方法

- 如果我们没有定义过构造方法,编译器会在编译的时候添加默认空参的构造方法,这就是我们之前从未定义过构造方法,而能一直使用的原因
- 如果我们定义了新的构造方法,那么编译器就不会在给

我们添加空参的构造方法.

- 构造方法的细节：
  1. 一个类中可以有多个构造方法，多个构造方法是以重载的形式存在的
  2. 构造方法是可以被private修饰的，作用：其他程序无法创建该类的对象。

子父构造方法的调用

- 子类创建对象的时候,会调用父类的构造方法,在子类构造方法中的第一行总有一个隐式的调用`super();`会默认去调用父类的无参的构造方法.

```
public class TTTTTT {
    public static void main(String[] arg
s) {
        new Zi();
    }

}
class Fu{
    int a;
    Fu(){
        System.out.println("父类构造方法"+n
um);
        a = 250;
    }
}
class Zi extends Fu{
    Zi(){
        System.out.println("子类构造方法"+n
um);
    }
}
//输出的结果是
//父类构造方法0
//子类构造方法250
```

- 思考1

  > 为什么子类的构造方法会去调用父类的构造方法呢?

因为构造方法本身就是初始化对象的成员变量的,父类的构造方法也是这个目的,如果不去调用,我们就不知道父类做了什么操作,所以才会隐式的去调用super();
子类会继承父类中的内容，所以子类在初始化时，必须先到父类中去执行父类的初始化动作。这样，才可以使用父类中的内容。

- 思考2

  > 如果子类会默认调用父类的无参构造方法,如果父类中又没有无参的构造方法,程序会出现什么问题?编译器会自动去找到对应的有参的构造方法么?

- 如果父类中没有空参构造方法,那么子类的的构造方法就会报错,需要我们显示的调用父类的有参数的构造方法