**封装、继承和多态**

1. 实验目的
2. 构造一个类，把对象的属性封装起来，同时提供一些可以被外界访问属性的方法。

2. 掌握继承的基本概念以及怎么使用继承。

3. 掌握super关键字的使用

4. 掌握方法的重写与重载。

5. 掌握抽象类的定义及用法。

6. 理解并能正确使用final关键字。

7. 掌握接口相关的知识。

1. 实验内容
2. 根据提示，在右侧编辑器Begin-End处补充代码：

声明一个Animal类，将属性name和age封装起来，提供对外的公共访问方法；

声明一个Cat类和Dog类，都继承Animal类，分别定义各自的voice方法和eat方法；

在main方法中分别实例化一个Cat对象和Dog对象，设置各自的属性并调用这两个方法，再打印出名字和年龄信息；

具体具体输出要求请看测试说明。

测试输入：无 预期输出： 姓名：张三，年龄：18，学校：哈佛大学

1. 根据提示，在右侧编辑器补充代码。

声明一个名为Person的类，里面声明name与age两个属性，定义talk()方法返回姓名和年龄信息；

声明一个名为Student的类，此类继承自Person类，添加school属性，声明带三个参数的构造方法，复写talk()方法，在该方法中调用父类的talk()方法，返回姓名、年龄和学校信息；

实例化子类对象s，调用talk()方法打印我是：张三，今年：18岁，我在哈佛大学上学。

3． 根据提示，在右侧编辑器补充代码：

声明一个名为Person的抽象类，在Person类中声明了三个属性name、age和occupation和一个抽象方法talk()；

声明一个Student类和一个Worker类，都继承自Person类，添加带有三个参数的构造方法；

1. 分别实例化Student类与Worker类的对象，分别调用各自类中被复写的talk()方法打印输出信息；

具体输出要求请看测试说明。

1. 根据提示，在右侧编辑器Begin-End处补充代码:

仔细阅读代码，在右侧编辑器中调整代码使程序能正确编译运行;

具体输出要求请看测试说明。

1. 根据提示，在右侧编辑器补充代码：

声明一Person接口，并在里面声明三个常量：name、age、occupation，并分别赋值；

声明一Student类，此类实现Person接口，并复写Person中的talk()方法；

实例化一Student的对象s，并调用talk()方法，打印信息；

具体输出要求请看测试说明。

1. 根据提示，在右侧编辑器Begin-End处补充代码：

声明一个Person类，私有化属性name和age，并将字段封装起来；

在Person类中定义一个talk()方法，打印姓名和年龄信息；

在main方法中声明并实例化一Person对象p，给p中的属性赋值，调用talk()方法打印 我是：张三，今年：18岁。

1. 实验结果

public class TestPersonDemo {

    public static void main(String[] args) {

        /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

        // 声明并实例化一Person对象p

        Person p = new Person();

        // 给p中的属性赋值

        p.setName("张三");

        p.setAge(18);

        // 调用Person类中的talk()方法

        p.talk();

        /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

    }

}

// 在这里定义Person类

class Person {

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    private String name;

    private int age;

    public void talk(){

        System.out.println("我是："+name+"，今年："+age+"岁");

    }

    public String getName() {

        return name;

    }

    public void setName(String name) {

        this.name = name;

    }

     public int getAge() {

        return age;

    }

    public void setAge(int age) {

        this.age = age;

    }

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

public class extendsTest {

    public static void main(String args[]) {

        // 实例化一个Cat对象，设置属性name和age，调用voice()和eat()方法，再打印出名字和年龄信息

        /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

        Cat c = new Cat();

        c.setName("大花猫");

        c.setAge(6);

        c.voice();

        c.eat();

        System.out.println("大花猫"+c.getAge()+"岁");

        /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

        // 实例化一个Dog对象，设置属性name和age，调用voice()和eat()方法，再打印出名字和年龄信息

        /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

        Dog d = new Dog();

        d.setName("大黑狗");

        d.setAge(8);

        d.voice();

        d.eat();

        System.out.println("大黑狗"+d.getAge()+"岁");

        /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

    }

}

class Animal {

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    private String name;

    private int age;

    public String getName(){

        return name;

    }

    public void setName(String name){

        this.name = name;

    }

    public int getAge(){

        return age;

    }

    public void setAge(int age){

        this.age = age;

    }

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

class Cat extends Animal {

    // 定义Cat类的voice()和eat()方法

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    public void voice(){

        System.out.println("大花猫喵喵叫");

    }

    public void eat(){

        System.out.println("大花猫吃鱼");

    }

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

class Dog extends Animal {

    // 定义Dog类的voice()和eat()方法

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    public void voice(){

        System.out.println("大黑狗汪汪叫");

    }

    public void eat(){

        System.out.println("大黑狗吃骨头");

    }

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

public class superTest {

    public static void main(String[] args) {

        // 实例化一个Student类的对象s，为Student对象s中的school赋值，打印输出信息

        /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

        Student s = new Student();

        /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

    }

}

class Person {

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    public String name = "张三";

    public int age=18;

    public Person(){

        System.out.print("姓名："+name+"，年龄："+age);

    }

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

class Student extends Person {

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    String school="哈佛大学";

    public Student(){

        super();

        System.out.println("，学校："+school);

    }

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

public class overridingTest {

    public static void main(String[] args) {

        // 实例化子类对象s，调用talk()方法打印信息

        /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

        Student s = new Student();

        s.talk("张三",18,"哈佛大学");

        /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

    }

}

class Person {

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    String name;

    int age;

    public void talk(String name,int age){

        System.out.println("我是："+name+"，今年："+age+"岁");

    }

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

class Student extends Person {

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    String school;

    public void talk(String name,int age,String school){

        System.out.println("我是："+name+"，今年："+age+"岁"+"，我在"+school+"上学");

    }

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

public class abstractTest {

    public static void main(String[] args) {

        /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

        // 分别实例化Student类与Worker类的对象，并调用各自构造方法初始化类属性。

        Student s=new Student();

        Worker w=new Worker();

        // 分别调用各自类中被复写的talk()方法 打印信息。

        s.talk("张三",20,"学生");

        w.talk("李四",30,"工人");

        /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

    }

}

// 声明一个名为Person的抽象类，在Person中声明了三个属性name age occupation和一个抽象方法——talk()。

abstract class Person {

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    String name;

    int age;

    String occupation;

    abstract void talk(String name,int age,String occupation);

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

// Student类继承自Person类，添加带三个参数的构造方法，复写talk()方法 返回姓名、年龄和职业信息

class Student extends Person {

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    void talk(String name,int age,String occupation){

        System.out.println("学生——>姓名："+name+"，年龄："+age+"，职业："+occupation+"！");

    }

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

// Worker类继承自Person类，添加带三个参数的构造方法，复写talk()方法 返回姓名、年龄和职业信息

class Worker extends Person {

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    void talk(String name,int age,String occupation){

        System.out.println("工人——>姓名："+name+"，年龄："+age+"，职业："+occupation+"！");

    }

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

//不可以修改 final 变量的值

// final方法,不可以重写

//// 不可以扩展 final 类

    //请在此添加你的代码

    /\*\*\*\*\*\*\*\*\*\* Begin \*\*\*\*\*\*\*\*\*/

    class Bike1 {

    int speedlimit = 90;

    void run() {

        speedlimit = 120;

        System.out.println("speedlimit=120");

    }

}

class Bike2 {

    void run() {

        System.out.println("running");

    }

}

class Honda extends Bike2 {

    void run() {

        System.out.println("running safely with 100kmph");

    }

}

class Bike3 {

}

class Yamaha extends Bike3 {

    void run() {

        System.out.println("running safely with 100kmph");

    }

}

    /\*\*\*\*\*\*\*\*\*\* End \*\*\*\*\*\*\*\*\*\*/

public class interfaceTest {

    public static void main(String[] args) {

        // 实例化一Student的对象s，并调用talk()方法，打印信息

        /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

        Student s=new Student();

        s.talk();

        /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

    }

}

// 声明一个Person接口，并在里面声明三个常量：name、age和occupation，并分别赋值，声明一抽象方法talk()

interface Person {

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    final String name= "张三";

    final int age = 18;

    final String occupation="学生";

    abstract void talk();

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

// Student类继承自Person类 复写talk()方法返回姓名、年龄和职业信息

class Student implements Person {

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    public void talk(){

        System.out.println("学生——>姓名："+name+"，年龄："+age+"，职业："+occupation+"！");

    }

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

public class TestPolymorphism {

    public static void main(String[] args) {

        // 以多态方式分别实例化子类对象并调用eat()方法

        /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

        Animal a = new Dog();

        a.eat();

        Animal c = new Cat();

        c.eat();

        Animal l = new Lion();

        l.eat();

        /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

    }

}

// Animal类中定义eat()方法

class Animal {

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    public void eat(){

    }

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

// Dog类继承Animal类 复写eat()方法

class Dog extends Animal {

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    public void eat(){

        System.out.println("eating bread...");

    }

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

// Cat类继承Animal类 复写eat()方法

class Cat extends Animal {

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    public void eat(){

        System.out.println("eating rat...");

    }

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

// Lion类继承Animal类 复写eat()方法

class Lion extends Animal {

    /\*\*\*\*\*\*\*\*\* begin \*\*\*\*\*\*\*\*\*/

    public void eat(){

        System.out.println("eating meat...");

    }

    /\*\*\*\*\*\*\*\*\* end \*\*\*\*\*\*\*\*\*/

}

1. 实验总结
2. 多态的三个条件:

①继承的存在(继承是多态的基础，没有继承就没有多态)；

②子类重写父类的方法(多态下调用子类重写的方法)；

③父类引用变量指向子类对象(子类到父类的类型转换)。

子类转换成父类时的规则:

①将一个父类的引用指向一个子类的对象，称为向上转型(upcasting)，自动进行类型转换。此时通过父类引用调用的方法是子类覆盖或继承父类的方法，不是父类的方法。 此时通过父类引用变量无法调用子类特有的方法；

②如果父类要调用子类的特有方法就得将一个指向子类对象的父类引用赋给一个子类的引用，称为向下转型，此时必须进行强制类型转换。

1. 子类对象在实例化时会默认先去调用父类中的无参构造方法，之后再调用本类中的相应构造方法。
2. 方法重载是指多个方法可以享有相同的名字，但是参数的数量或类型不能完全相同。
3. 抽象类的定义规则：

抽象类和抽象方法都必须用abstract关键字来修饰；

抽象类不能被实例化，也就是不能用new关键字去产生对象；

抽象方法只需声明，而不需实现；

含有抽象方法的类必须被声明为抽象类，抽象类的子类必须复写所有的抽象方法后才能被实例化，否则这个子类还是个抽象类。

5． final关键字修饰类、成员变量和成员方法