**日期：2020/5/8**

封装、继承和多态的综合练习

1. 实验目的
2. 按要求编写一个Java应用程序，巩固Java面向对象知识。
3. 按要求编写一个Java应用程序，巩固Java封装、继承和多态的知识。
4. 过一个简单实例讲解并自己动手编写一个Java应用程序，全面复习Java面向对象知识。
5. 实验内容
6. 声明一个抽象类Pet，封装属性name和sex，声明一个带有两个参数的构造函数，声明抽象方法void talk()和void eat()；

声明一个Dog类继承自Pet，封装属性color，声明带有三个参数的构造函数，复写talk()和eat()方法；

明一个Cat类继承自Pet，封装属性weight，声明带有三个参数的构造函数，复写talk()和eat()方法；

编写测试类，通过有参构造函数实例化Dog类对象，调用talk()方法和eat()方法；通过有参构造函数实例化Cat类对象 ，调用talk()方法和eat()方法；

1. 定义一个抽象类Person，包含抽象方法eat()，封装属性name、sex、age，声明包含三个参数的构造方法；

定义一个Chinese类，继承自Person类，重写父类的eat()方法，并定义一个自己特有的方法shadowBoxing()；

定义一个English类，继承自Person类，重写父类的eat()方法，并定义一个自己特有的方法horseRiding()；

编写测试类，定义一个showEat()方法，使用父类作为方法的形参，实现多态，分别调用showEat()方法，通过强制类型转换调用各自类特有的方法；

教练和运动员案例：

乒乓球运动员和篮球运动员；

乒乓球教练和篮球教练；

跟乒乓球相关的人员都需要学习英语；

分析，这个案例中有哪些抽象类，哪些接口，哪些具体类。

1. 实验结果

// 通过有参构造函数实例化Dog类对象dog

        // dog调用talk()方法

        // dog调用eat()方法

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

        Dog d = new Dog(dogName,dogSex,dogColor);

        d.talk();

        d.eat();

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

        // 通过有参构造函数实例化Cat类对象cat

        // cat调用talk()方法

        // cat调用eat()方法

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

        Cat c = new Cat(catName,catSex,catWeight);

        c.talk();

        c.eat();

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

    }

}

// 抽象类Pet 封装属性name和sex

// 构造函数初始化name和sex

// 声明抽象方法talk()

// 声明抽象方法eat()

abstract class Pet {

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

    String name;

    String sex;

    public Pet(){

    }

    abstract void talk();

    abstract void eat();

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

}

// Dog类继承自Pet类 封装属性color

// 构造函数初始化name、sex和color

// 实现自己的talk()方法和eat()方法

// talk()输出'名称：name，性别：sex，颜色：color，汪汪叫'

// eat()输出'name吃骨头'

class Dog extends Pet {

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

    String color;

    public Dog(String name,String sex,String color){

        this.name=name;

        this.sex=sex;

        this.color=color;

    }

    void talk(){

        System.out.println("名称："+name+"，性别："+sex+"，颜色："+color+"，汪汪叫");

    }

    void eat(){

        System.out.println(name+"吃骨头！");

    }

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

}

// Cat类继承自Pet类 封装属性weight

// 构造函数初始化name、sex和weight

// 实现自己的talk()方法和eat()方法

// talk()输出'名称：name，性别：sex，体重：weight kg，喵喵叫'

// eat()输出'name吃鱼'

class Cat extends Pet {

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

    double weight ;

    public Cat(String name,String sex,double weight){

        this.name=name;

        this.sex=sex;

        this.weight=weight;

    }

    void talk(){

        System.out.println("名称："+name+"，性别："+sex+"，体重："+weight+"kg，喵喵叫");

    }

    void eat(){

        System.out.println(name+"吃鱼！");

    }

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

}

2. // 创建测试类对象test

// 创建Person类对象person1,引用指向中国人,通过有参构造函数实例化中国人类对象

// 通过showEat()方法调用Chinese的eat()方法

// 创建Person类对象person2,引用指向英国人,通过有参构造函数实例化英国人类对象

// 通过showEat()方法调用English的eat()方法

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

Person person1=new Chinese(cName,cSex,cAge);

showEat(person1);

Person person2 = new English(eName,eSex,eAge);

showEat(person2);

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

// 强制类型转换(向下转型) 调用Chinese类特有的方法shadowBoxing()

// 强制类型转换(向下转型) 调用English类特有的方法horseRiding()

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

Chinese c=(Chinese) person1;

c.shadowBoxing();

English e=(English) person2;

e.horseRiding();

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

}

// 定义showEat方法，使用父类作为方法的形参，实现多态，传入的是哪个具体对象就调用哪个对象的eat()方法

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

public static void showEat(Person p){

if(p instanceof Chinese){

Chinese c= (Chinese) p;

c.eat();

}else if(p instanceof English){

English e= (English) p;

e.eat();

}

}

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

}

// 抽象类Person 封装属性name、sex和age

// 构造函数初始化name、sex和age

// 声明抽象方法eat()

abstract class Person {

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

String name;

String sex;

int age;

public Person(){

}

abstract void eat();

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

}

// Chinese类继承自Person类

// 构造函数初始化name、sex和age

// 重写父类方法eat() 输出'姓名：name，性别：sex，年龄：age，我是中国人，我喜欢吃饭！'

// 定义子类特有方法shadowBoxing(),当父类引用指向子类对象时无法调用该方法 输出'name在练习太极拳！'

class Chinese extends Person {

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

public Chinese (String name,String sex,int age){

this.name=name;

this.sex=sex;

this.age=age;

}

void eat(){

System.out.println("姓名："+name+"，性别："+sex+"，年龄："+age+"，我是中国人，我喜欢吃饭！");

}

public void shadowBoxing(){

System.out.println(name+"在练习太极拳！");

}

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

}

// English类继承自Person类

// 构造函数初始化name、sex和age

// 重写父类方法eat() 输出'姓名：name，性别：sex，年龄：age，我是英国人，我喜欢吃三明治!'

// 定义子类特有方法horseRiding(),当父类引用指向子类对象时无法调用该方法 输出'name在练习骑马！'

class English extends Person {

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

public English(String name,String sex,int age){

this.name=name;

this.sex=sex;

this.age=age;

}

void eat(){

System.out.println("姓名："+name+"，性别："+sex+"，年龄："+age+"，我是英国人，我喜欢吃三明治！");

}

public void horseRiding(){

System.out.println(name+"在练习骑马！");

}

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

}

3．

   // 测试运动员(乒乓球运动员和篮球运动员)

        // 乒乓球运动员

        // 通过带参构造函数实例化PingPangPlayer对象ppp

        // 输出'name---age'

        // 分别调用sleep()、eat()、study()、speak()方法

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

        PingPangPlayer ppp=new PingPangPlayer(pppName,pppAge);

        System.out.println(ppp.name+"---"+ppp.age);

        ppp.sleep();

        ppp.eat();

        ppp.study();

        ppp.speak();

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

        System.out.println("----------------");

        // 篮球运动员

        // 通过带参构造函数实例化BasketballPlayer对象bp

        // 输出'name---age'

        // 分别调用sleep()、eat()、study()方法

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

        BasketballPlayer bp=new BasketballPlayer(bpName,bpAge);

        System.out.println(bp.name+"---"+bp.age);

        bp.sleep();

        bp.eat();

        bp.study();

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

        System.out.println("----------------");

        // 测试教练(乒乓球教练和篮球教练)

        // 乒乓球教练

        // 通过带参构造函数实例化PingPangCoach对象ppc

        // 输出'name---age'

        // 分别调用sleep()、eat()、teach()、speak()方法

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

        PingPangCoach ppc = new PingPangCoach(ppcName,ppcAge);

        System.out.println(ppc.name+"---"+ppc.age);

        ppc.sleep();

        ppc.eat();

        ppc.teach();

        ppc.speak();

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

        System.out.println("----------------");

        // 篮球教练

        // 通过带参构造函数实例化BasketballCoach对象bc

        // 输出'name---age'

        // 分别调用sleep()、eat()、teach()方法

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

        BasketballCoach bc =new BasketballCoach(bcName,bcAge);

        System.out.println(bc.name+"---"+bc.age);

        bc.sleep();

        bc.eat();

        bc.teach();

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

        System.out.println("----------------");

    }

}

// 说英语接口 声明抽象方法speak()

interface SpeakEnglish {

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

    abstract void speak();

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

}

// 定义人的抽象类Person 封装name和age

// 无参构造函数

// 有参构造函数初始化name和age

// 定义具体方法sleep() 输出'人都是要睡觉的'

// 抽象方法eat()（吃的不一样）

abstract class Person {

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

    String name;int age;

    public Person(){

    }

    public Person(String name,int age){

        this.name=name;

        this.age=age;

    }

    void sleep(){

        System.out.println("人都是要睡觉的");

    }

    abstract void eat();

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

}

// 定义运动员Player（抽象类）继承自Person类

// 无参构造函数

// 有参构造函数初始化name和age

// 运动员学习内容不一样，抽取为抽象 定义抽象方法study()

abstract class Player extends Person {

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

    public Player(){

    }

    public Player(String name,int age){

        this.name=name;

        this.age=age;

    }

     abstract void study();

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

}

// 定义教练Coach（抽象类）继承自Person类

// 无参构造函数

// 有参构造函数初始化name和age

// 教练教的不一样 定义抽象方法teach()

abstract class Coach extends Person {

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

    public Coach(){

    }

    public Coach(String name,int age){

        this.name=name;

        this.age=age;

    }

     abstract void teach();

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

}

// 定义乒乓球运动员具体类PingPangPlayer 继承自Player类并实现SpeakEnglish类（兵乓球运动员需要说英语）

// 无参构造函数

// 有参构造函数初始化name和age

// 实现自己的eat()方法 输出'乒乓球运动员吃大白菜，喝小米粥'

// 实现自己的study()方法 输出'乒乓球运动员学习如何发球和接球'

// 实现自己的speak()方法 输出'乒乓球运动员说英语'

class PingPangPlayer extends Player implements SpeakEnglish {

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

    public PingPangPlayer(){

    }

    public PingPangPlayer(String name,int age){

        this.name=name;

        this.age=age;

    }

    void eat(){

        System.out.println("乒乓球运动员吃大白菜，喝小米粥");

    }

    void study(){

        System.out.println("乒乓球运动员学习如何发球和接球");

    }

    public void speak(){

        System.out.println("乒乓球运动员说英语");

    }

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

}

// 定义篮球运动员具体类BasketballPlayer 继承自Player类 不需要继承接口，因为他不需要说英语

// 无参构造函数

// 有参构造函数初始化name和age

// 实现自己的eat()方法 输出'篮球运动员吃牛肉，喝牛奶'

// 实现自己的study()方法 输出'篮球运动员学习如何运球和投篮'

class BasketballPlayer extends Player {

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

    public BasketballPlayer(){

    }

    public BasketballPlayer(String name,int age){

        this.name=name;

        this.age=age;

    }

    void eat(){

        System.out.println("篮球运动员吃牛肉，喝牛奶");

    }

    void study(){

        System.out.println("篮球运动员学习如何运球和投篮");

    }

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

}

// 定义乒乓球教练具体类 PingPangCoach 继承自Coach类并实现SpeakEnglish类（兵乓球教练需要说英语）

// 无参构造函数

// 有参构造函数初始化name和age

// 实现自己的eat()方法 输出'乒乓球教练吃小白菜，喝大米粥'

// 实现自己的teach()方法 输出'乒乓球教练教如何发球和接球'

// 实现自己的speak()方法 输出'乒乓球教练说英语'

class PingPangCoach extends Coach implements SpeakEnglish {

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

    public PingPangCoach(){

    }

    public PingPangCoach(String name,int age){

        this.name=name;

        this.age=age;

    }

    void eat(){

        System.out.println("乒乓球教练吃小白菜，喝大米粥");

    }

    void teach(){

        System.out.println("乒乓球教练教如何发球和接球");

    }

    public void speak(){

        System.out.println("乒乓球教练说英语");

    }

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

}

// 定义篮球教练具体类BasketballCoach 继承自Coach类 不需要继承接口，因为他不需要说英语

// 无参构造函数

// 有参构造函数初始化name和age

// 实现自己的eat()方法 输出'篮球教练吃羊肉，喝羊奶'

// 实现自己的teach()方法 输出'篮球教练教如何运球和投篮'

class BasketballCoach extends Coach {

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

    public BasketballCoach(){

    }

    public BasketballCoach(String name,int age){

        this.name=name;

        this.age=age;

    }

    void eat(){

        System.out.println("篮球教练吃羊肉，喝羊奶");

    }

    void teach(){

        System.out.println("篮球教练教如何运球和投篮");

    }

    public void speak(){

        System.out.println("乒乓球教练说英语");

    }

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

}

1. 实验总结
2. 在继承的时候，父类也有构造方法，如果要创建子类的对象，那么执行的过程首先是调用父类的无参构造方法生成父类的对象，然后再调用子类的无参构造方法来生成子类对象。继承的时候都是先生成父类的对象，然后再生成子类的对象。
3. 在类方法中不能使用super、this关键字。（this表示当前类的对象，由static修饰的方法是类直接调用，不需要创建对象，所以不能用this）；