实验内容	第四周实验 继承与多态			成 绩	
姓　名	王秋锋	学号	2015111948	班 级	计算机 2015-03 班
专　业	计算机科学与技术			日 期	2017 年　10 月　7 日

【实验内容】

1、编辑、编译、运行下面 java 程序（分析程序给你，给出运行结果）

要求：

1、运行上面的程序 ，对子类隐藏超类的成员变量进行分析理解

2、将上例子“int x = 2;” 改为“static int x=2”，运行程序，并理解静态成员在子类中的使用

【实验结果与分析】

```
  class A1{
   int x = 2;
   public void setX(int i){
      x =i;
   }
   void printa(){
      System.out.println(x);
   }
}

class B1 extends A1{
   int x = 100;
   void printb(){
      super.x = super.x+10;
      System.out.println("super.x=" + super.x + " x=" + x);
   }
}

public class Exam4_4Test {
   public static void main(String[] args){
      A1 a1 = new A1();
      a1.setX(4);
      a1.printa();

      B1 b1 = new B1();
      b1.printb();
      b1.printa();

      b1.setX(6);
      b1.printb();
      b1.printa();
```

```
        a1.printa();
    }
}
```

修改前实验结果：

```
4
super.x=12 x=100
12
super.x=16 x=100
16
4
```

修改后实验结果

```
4
super.x=14 x=100
14
super.x=16 x=100
16
16
```

实验分析：

修改后将 x 设置为 static 型后，变为所有 A 类公有，因此会产生如上的变化

2、子类构造方法的定义

【实验结果与分析】

要求：

1、运行上面程序，并分析程序运行结果

2、采用修改超类与子类构造方法的定义的不同方式，重新定义构造方法，并对其进行分析。

程序运行结果报错，子类的构造器要显示调用超类构造器，不然会默认调用超类无参数的构造器,但是超类无无参数构造器，所以会报错。

修改：给超类添加一个无参数构造器，修改后的代码如下：

```
class Animal{   //子类构造方法的定义
    protected String name;
    Animal(){}
    public Animal(String n){  //超类定义带参构造方法
        name= n;
    }
    public void Showlnfo(){
        System.out.print("the animal\t");
        System.out.print(name+"\t");
    }
}
public class Fish extends Animal{
    int SwimSpeed;
    Fish(String n,int s){  //子类定义带参构造方法
```

```
        name=n;
        SwimSpeed=s;
    }
    public void ShowInfo(){
        System.out.print("the Fish\t"+name +"\t");
        System.out.print("Swimmingspeed:\t"+ SwimSpeed);
    }
    public static void main(String args[]){
        Fish a = new Fish("Nemo",60);
        a.ShowInfo();
    }
}
```

修改后运行结果：

```
the Fish        Nemo    Swimmingspeed:  60
```

3、类的静态成员与非静态成员

运行下面的程序，并对其进行分析。

```
class TestStatic{
    public int i= 3;//声明成员变量
    public static int s=5;//声明静态变量
    public void counta(){ //非静态方法
        System.out.println("执行成员方法 counta()");
        System.out.println ("i=" + i);  //直接调用成员变量
        System.out.println ("s=" + s);  //直接调用静态变量
    }
    public static void sTest(){  //静态方法
        System.out.println ("执行静态方法 sTest()");
        TestStatic test=new TestStatic();
        test.i=test.i + 3; //静态方法中闲用非静态变量,:要创建类的对象
        System.out.println("i="+test.i);
        s=s + 5;  //直接调用静态变量
        System.out.println("s="+s);
    }
}
public class Example5_2{
    public static void main (String [] args){//主方法
        System.out.println("执行主方法");
        TestStatic t=new TestStatic();  //创建类的对象
        System.out.println("t.s="+t.i);
        System.out.println("t.s="+t.s); //输出实例变量和类变量的值
        System.out.println("TestStatic.s="+TestStatic.s);
        t.counta();      //使用对象调用成员方法
        t.sTest();       //使用对象闲用类方法
        TestStatic.sTest ();  //使用类名直接调用类方法
    }
```

```
}
```

说明：本程序供计算机专业班级使用。

【实验结果与分析】

```
执行主方法
t.s=3
t.s=5
TestStatic.s=5
执行成员方法 counta()
i=3
s=5
执行静态方法 sTest()
i=6
s=10
执行静态方法 sTest()
i=6
s=15
```

实验分析：
在 sTest 中 test.i 每次使用 sTest 这个方法的时候都会初始化为 3+3=6，所以没有变化，但是 s 是属于类的静态变量，所以没使用一次 sTest 方法都会增加 5。所以才产生了上述结果。

4、Object 类的学习与使用

```
 class Book
{
   private String name;
   private float price;
   public Book(String name,float price)
   {  this.name = name;
      this.price = price;
   }
   public void print(Book b)
   {  System.out.println("直接输出对象: "+b);
      System.out.println("toString: "+b.toString());
      System.out.println("hashCode: "+Integer.toHexString(b.hashCode()));
   }
}
public class ObjectDemo
{
   public static void main(String[] args)
   {  Book b1,b2;
      b1 = new Book("Java",32.5f);
      b1.print(b1);
      b2 = new Book("C++",36.5f);
```

```
        b2.print(b2);
    }
}
```

要求：

1、运行上面程序给出程序的运行结果

2、编写程序，分别调用 getClass() 、toString()、boolean equals( )、hashCode()的等方法，熟悉 Object 类。

【实验结果与分析】

1. 运行结果：

其中 week_4 是工程名

```
直接输出对象: week_4.Book@2f57d162
toString: week_4.Book@2f57d162
hashCode: 2f57d162
直接输出对象: week_4.Book@2e739136
toString: week_4.Book@2e739136
hashCode: 2e739136
```

实验分析：

直接输出对象和 Object 类自带的 toString 方法是等价的，输出的是类名和散列码

2.

```
class Book{
    private String name;
    private float price;
    public Book(String name,float price){
        this.name=name;
        this.price=price;
    }
    public void print(Book b){
        System.out.println("直接输出对象: "+b);
        System.out.println("toString: "+b.toString());
        System.out.println("hashCode: "+Integer.toHexString(b.hashCode()));
        System.out.println("getClass "+b.getClass());
    }
}

public class ObjectDemo {
    public static void main(String []args){
        Book b1,b2;
        b1=new Book("Java",32.5f);
        b1.print(b1);
        b2=new Book("C++",36.5f);
        b2.print(b2);
        System.out.println("is equal: "+b1.equals(b2));
    }
}
```

运行结果：

```
直接输出对象: week_4.Book@2542880d
toString: week_4.Book@2542880d
hashCode: 2542880d
getClass class week_4.Book
直接输出对象: week_4.Book@32f22097
toString: week_4.Book@32f22097
hashCode: 32f22097
getClass class week_4.Book
is equal: false
```

实验分析：

Book1 和 Book2 不一样，因而调用 boolean equals( )会输出 false

5、编写程序

定义一个描述复数的例子 ComplexNumber：

成员变量：

复数的实部 ： private double realPart;

复数的虚部： private double imaginaryPart;

计数静态变量：count

成员方法

无参与带参的构造方法：ComplexNumber

加运算：add( ); 减运算：decrease( );

乘运算：multiply( ); 除运算：divide( )

字符串输出：toString( ); 对象比较：equals( )

获取 hash 码：hashCode( );

析构方法：finalize; //一旦销毁对象，要 count 减一

获取实数的实部与虚部运算与设置实数的实部与虚部的 set 与 get 方法

要求：1、定义测试类，完成 ComplexNumber 类的测试

2、给出程序代码与运行结果

程序运行结果：

```
count = 2
cyber1 =5.0+4.0i
cyber2 =2.0+3.0i
cyber1.hashCode() = 1465636487
cyber2.hashCode() = 532142987
cyber1+cyber2 =7.0+7.0i
cyber1-cyber2 =3.0+1.0i
cyber1*cyber2 =-2.0+23.0i
cyber1/cyber2 =1.6923076923076923+-0.5384615384615384i
Is cyber1 equal to cyber2? false
5
```

程序代码：

```
public class ComplexNumber {
   private double realPart;
```

```
    private double imaginaryPart;
    public static int count = 0;
    ComplexNumber(){
        realPart = 0;
        imaginaryPart = 0;
        count++;
    }
    ComplexNumber(double r, double i){
        realPart = r;
        imaginaryPart = i;
        count++;
    }

    public ComplexNumber add(ComplexNumber x){
        return new ComplexNumber(this.realPart+x.realPart,
this.imaginaryPart+x.imaginaryPart);
    }

    public ComplexNumber decrease(ComplexNumber x){
        return new ComplexNumber(this.realPart-x.realPart,
this.imaginaryPart-x.imaginaryPart);
    }

    public ComplexNumber multiply(ComplexNumber x){
        return new
ComplexNumber(this.realPart*x.realPart-this.imaginaryPart*x.imaginaryPart,
this.realPart*x.imaginaryPart+this.imaginaryPart*x.realPart);
    }

    public ComplexNumber divide(ComplexNumber x){
        double div = Math.pow(x.realPart,2)+Math.pow(x.imaginaryPart,2);
        double newReal  =
this.realPart*x.realPart+this.imaginaryPart*x.imaginaryPart;
        double newImage  =
this.imaginaryPart*x.realPart-this.realPart*x.imaginaryPart;
        return new ComplexNumber(newReal/div, newImage/div);
    }

    public String toString(){
        return realPart+"+"+imaginaryPart+"i";
    }

    public boolean equals(ComplexNumber x){
        if(this.realPart == x.realPart && this.imaginaryPart == x.imaginaryPart)
            return true;
        else
```

```
            return false;
    }

    public int hashCode(){
        return this.toString().hashCode();
    }

    void FinalizeComplex(){
        count--;
    }
    //获得实部
    public double getreal(){
        return this.realPart;
    }
    //获得虚部
    public double getimag(){
        return this.imaginaryPart;
    }
    //修改实部
    public void setreal(double r){
        this.realPart=r;
    }
    //修改虚部
    public void setimag(double i){
        this.imaginaryPart=i;
    }

    public static void main(String[] args){
        ComplexNumber cyber1 = new ComplexNumber();
        ComplexNumber cyber2 = new ComplexNumber(2, 3);
        cyber1.setreal(5); cyber1.setimag(4);
        System.out.println("count = "+ComplexNumber.count);
        System.out.println("cyber1 ="+cyber1.toString());
        System.out.println("cyber2 ="+cyber2.toString());
        System.out.println("cyber1.hashCode() = "+cyber1.hashCode());
        System.out.println("cyber2.hashCode() = "+cyber2.hashCode());
        System.out.println("cyber1+cyber2 ="+cyber1.add(cyber2));
        System.out.println("cyber1-cyber2 ="+cyber1.decrease(cyber2));
        System.out.println("cyber1*cyber2 ="+cyber1.multiply(cyber2));
        System.out.println("cyber1/cyber2 ="+cyber1.divide(cyber2));
        System.out.println("Is cyber1 equal to cyber2? "+cyber1.equals(cyber2));
        cyber1.FinalizeComplex();
        System.out.println(ComplexNumber.count);
    }
}
```

6. 编写 Java 程序

（1）Employee 类：Employee 类包含 no(工号：编号从 10000 开始)、name（String 类型）、age、daylyRate（每天酬金）、workDays（当月工作天数）、dept(所属部门：研发部、销售部、售后部)、Tel（电话，String 类型）、count（用来记录 Employee 对象个数，或职工人数）、finalize（销毁对象，或离职，count 减 1）等个成员变量；成员方法：Employee 类中有多个构造方法，分别为无参的、带一个或多个参数的构造方法， set 与 get 方法原来设置或返回属性值，show 方法（用来显示所有属性信息），pay()（返回每月薪水）。

（2）Manager 类（部门经理，负责本部门员工的管理，Employee 的子类）

增加成员变量：postPay（管理岗位补贴）

成员方法：showAll（）（显示所负责部门的所有员工）

（定义重载方法，显示员工的基本信息；显示员工的薪水等）

show（）（重新定义：多态的覆盖）

pay()（重新定义：返回每月薪水）

定义多个构造方法。

.....

要求：

（1） 定义上述两个类，并定义一个测试类进行运行测试

（2）可自行扩展，做一个小型的公式员工管理系统（该要求是可选项）

程序运行结果：

```
now count is 3
no:10001        name:Alice      age:25          daylyRate: 200
workDays:30     dept:研发部       Tel:15528330350
no:66666        name:Bob        age:23          daylyRate: 300
workDays:30     dept:售后部       Tel:15528331237
no:55555        name:jyly       age:27          daylyRate: 400
workDays:30     dept:售后部       Tel:15523530350
Alice的月薪是:6000
now count is 4
kaggle负责部门是:售后部        该部门员工有:
no:66666        name:Bob        age:23          daylyRate: 300
workDays:30     dept:售后部       Tel:15528331237
Bob的薪水是:9000
no:55555        name:jyly       age:27          daylyRate: 400
workDays:30     dept:售后部       Tel:15523530350
jyly的薪水是:12000
no:10034        name:kaggle     age:30          daylyRate: 500
workDays:28     dept:售后部       Tel:15524138947
postPay:3000
kaggle的月薪是:17000
Alice is dismissed,now count is 3
```

程序源代码

```
class Employee{
   private int no; //工号
```

```
    private String name;
    private int age;
    private int daylyRate;//每天酬金
    private int workDays; //当月工作天数
    private String dept; //所属部门：研发部、销售部、售后部
    private String Tel; //电话
    public static int count = 0; //职工人数
    public static int finalize = 0; //销毁数量
    //无参构造
    Employee(){
        no = 10000;
        count++;
    }
    //多个参数构造
    Employee(int _no, String _name, int _age, int _daylyRate, int _workDays,
String _dept, String _Tel){
        this.no = _no;
        this.name = _name;
        this.age = _age;
        this.daylyRate = _daylyRate;
        this.workDays = _workDays;
        this.dept = _dept;
        this.Tel = _Tel;
        count++;
    }

    //设置各个属性值
    public void setno(int _no){
        this.no = _no;
    }
    public void setname(String _name){
        this.name = _name;
    }
    public void setage(int _age){
        this.age = _age;
    }
    public void setday(int _daylyRate){
        this.daylyRate = _daylyRate;
    }
    public void setworkDays(int _workDays){
        this.workDays = _workDays;
    }
    public void setdept(String _dept){
        this.dept = _dept;
    }
    public void setTel(String Tel){
```

```
        this.Tel=Tel;
    }

    //得到各个属性值
    public int getno(){
        return this.no;
    }
    public String getname(){
        return this.name;
    }
    public int getage(){
        return this.age;
    }
    public int getday(){
        return this.daylyRate;
    }
    public int getworkDays(){
        return this.workDays;
    }
    public String getdept(){
        return this.dept;
    }
    public String getTel(){
        return this.Tel;
    }
    public int getcount(){
        return count;
    }
    public int getfinalize(){
        return finalize;
    }
    //用来显示所有属性信息
    public void show(){

    System.out.println("no:"+no+"\t"+"name:"+name+"\t"+"age:"+age+"\t\t"+"day
lyRate: "+daylyRate);

    System.out.println("workDays:"+workDays+"\t"+"dept:"+dept+"\t"+"Tel:"+Tel
);
    }
    //返回每月薪水
    public int pay(){
        return this.daylyRate*this.workDays;
    }
    void finalize_obj(){
        count--;
```

```
        finalize++;
    }
}

class Manager extends Employee{
    private int postPay;//管理岗位补贴

    Manager(){ super();}
    Manager(int _postPay){
        super();
        this.postPay = _postPay;
    }
    Manager(int _no, String _name, int _age, int _daylyRate, int _workDays, String
_dept, String _Tel, int _postPay){
        super(_no, _name, _age, _daylyRate, _workDays, _dept, _Tel);
        this.postPay = _postPay;
    }
    //显示所负责部门的所有员工,显示员工的基本信息；显示员工的薪水等
    public void showAll(String _dept, Employee a[]){
        System.out.println(this.getname()+"负责部门是:"+_dept+"\t该部门员工有:
");
        for(int i=0; i<a.length; i++){
            if(a[i].getdept() == _dept){
                a[i].show();
                System.out.println(a[i].getname()+"的薪水是:"+a[i].pay());
            }
        }
    }
    //重新定义：多态的覆盖
    public void show(){
        super.show();
        System.out.println("postPay:"+postPay);
    }
    //重新定义：返回每月薪水
    public int pay(){
        return getday()*getworkDays()+postPay;
    }
}

public class Demo {
    public static void main(String[] args){
        Employee  a[] = new Employee [3];
        a[0] = new Employee(10001, "Alice", 25, 200, 30, "研发部", "15528330350");
        a[1] = new Employee(66666, "Bob", 23, 300, 30, "售后部", "15528331237");
        a[2] = new Employee(55555, "jyly", 27, 400, 30, "售后部", "15523530350");
        System.out.println("now count is "+Employee.count);
```

```
        a[0].show();
        a[1].show();
        a[2].show();
        System.out.println("Alice的月薪是:"+a[0].pay());

        Manager a4 = new Manager(10034, "kaggle", 30, 500, 28, "售后部",
"15524138947", 3000);
        System.out.println("now count is "+Employee.count);
        a4.showAll("售后部", a);
        a4.show();
        System.out.println(a4.getname()+"的月薪是:"+a4.pay());
        a[0].finalize_obj();
        System.out.println("Alice is dismissed,now count is "+Employee.count);
    }
}
```

7、java native 方法及 JNI 实例

(参考： java native 方法及 JNI 实例 - xwdreamer 的专栏 - CSDN 博客
http://blog.csdn.net/xw13106209/article/details/6989415
)

要求：

（1）参考该博客的 native 与 JNI 实例，完成本实例的编写与调试，并给出步骤与运行结果

说明：本程序供计算机专业班级使用

实验结果与分析：

实验步骤：

1 编写带有 native 声明的方法的 java 类： HelloWorld.java

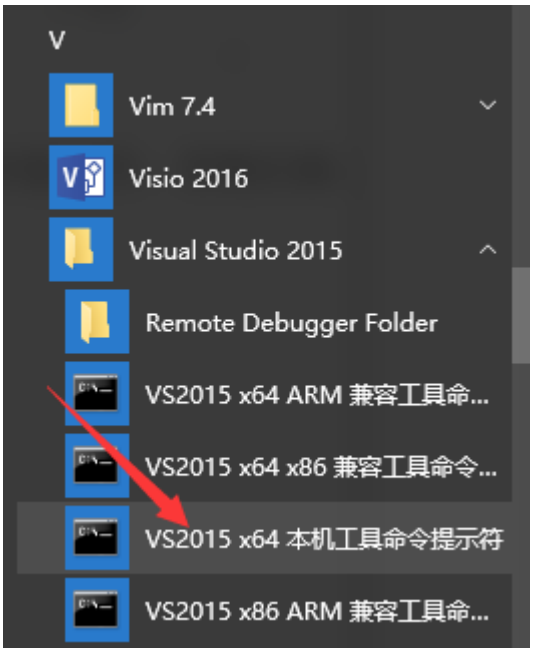
2 使用 javac 命令编译所编写的 java 类

3 使用 javah -jni java 类名生成扩展名为 h 的头文件

4 创建 HelloWorldImpl.cpp

5 将 C/C++编写的文件生成动态连接库， 将 C:\Program Files (x86)\Java\jdk1.8.0_25\include\jni.h 和 C:\Program Files (x86)\Java\jdk1.8.0_25\include\win32\jni_md.h 这两个文件拷贝到 D:\JNI\目录下

6 执行 cl/LD D:\JNI\HelloWorldImpl.cpp 得到 HelloWorldImpl.dll 文件（必须用 64 位的命令提示符，否则后面会出错），使用命令 cl/LD D:\JNI\HelloWorldImpl.cpp



执行完上述命令以后，我们在 D:\VS2015\VC 可以看到生成的四个文件，分别是：

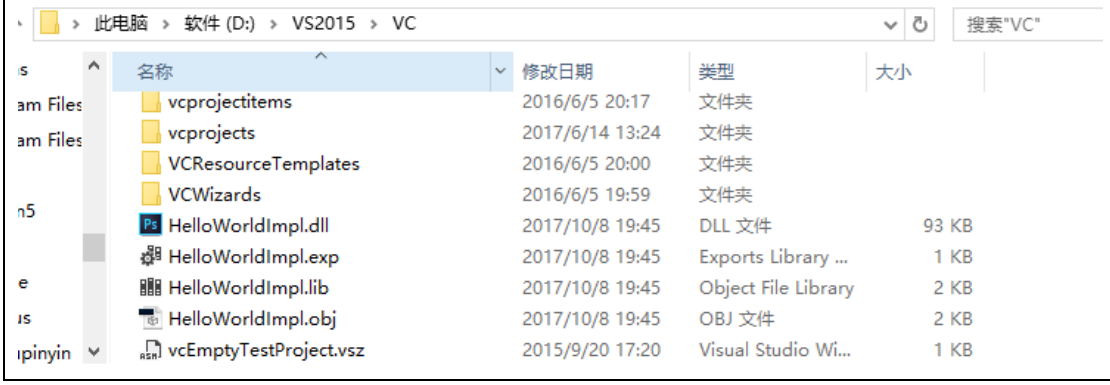
HelloWorldImpl.dll

HelloWorldImpl.exp

HelloWorldImpl.lib

HelloWorldImpl.obj

将其中的 HelloWorldImpl.dll 拷贝到 D:\JNI\目录下。



7 执行 class 得到结果

```
d:\JNI>java HelloWorld
Hello World!
```

实验分析：用 C/C++语言实现的输出 helloworld 程序，并且被编译成了 DLL，由 java 去调用。