《Java 程序设计》实验报告四

学生姓名：李季鸿　　班级：2021 级计科本（3）班　　学号：20213002624

实验地点：线上线下结合　　指导教师：张春元

实验日期：2023-03-27　　共 2 学时

实验环境：Win10+JDK1.8+IntelliJ IDEA 2022.1.1

1. 实验目的

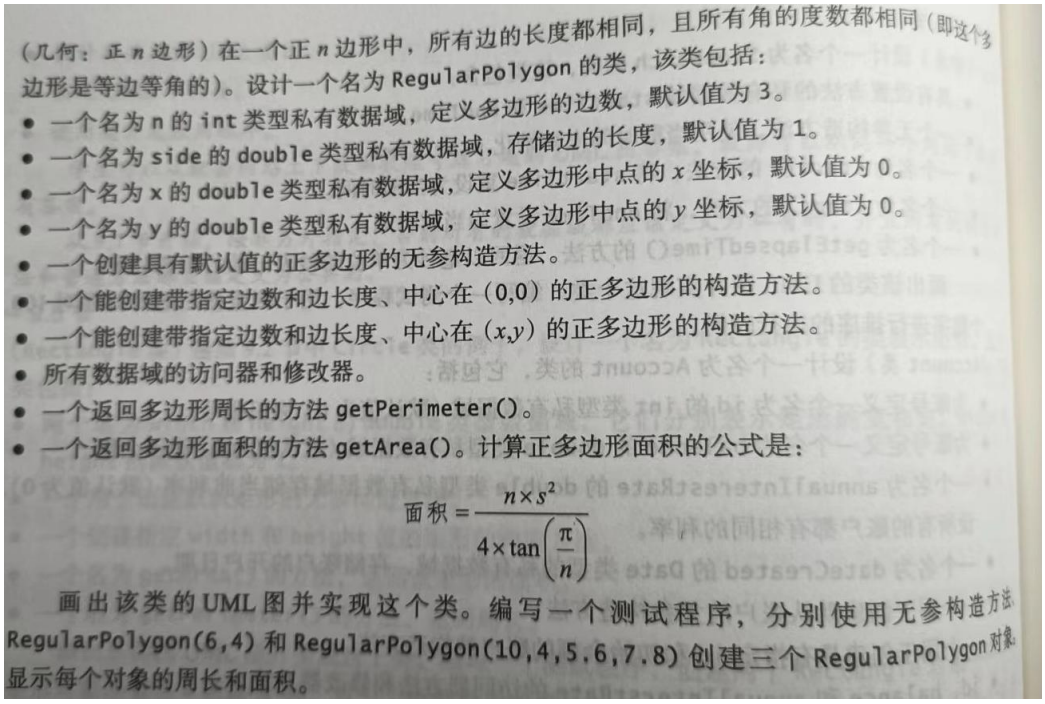
学习类与对象、子类与继承，掌握集成化开发工具工程的创建。

2. 实验内容

实验（1）用集成化开发工具完成实验教材 P28 实验 3 内容。

实验（2）用集成化开发工具完成实验教材 P31 实验 4 内容。

实验（3）：



（几何：正 n 边形）在一个正 n 边形中，所有边的长度都相同，且所有角的度数都相同（即这个多边形是等边等角的）。设计一个名为 RegularPolygon 的类，该类包括：

- 一个名为 n 的 int 类型私有数据域，定义多边形的边数，默认值为 3。
- 一个名为 side 的 double 类型私有数据域，存储边的长度，默认值为 1。
- 一个名为 x 的 double 类型私有数据域，定义多边形中点的 x 坐标，默认值为 0。
- 一个名为 y 的 double 类型私有数据域，定义多边形中点的 y 坐标，默认值为 0。
- 一个创建具有默认值的正多边形的无参构造方法。
- 一个能创建带指定边数和边长度、中心在（0,0）的正多边形的构造方法。
- 一个能创建带指定边数和边长度、中心在（x,y）的正多边形的构造方法。
- 所有数据域的访问器和修改器。
- 一个返回多边形周长的方法 getPerimeter()。
- 一个返回多边形面积的方法 getArea()。计算正多边形面积的公式是：

$$面积 = \frac{n \times s^2}{4 \times \tan\left(\frac{\pi}{n}\right)}$$

画出该类的 UML 图并实现这个类。编写一个测试程序，分别使用无参构造方法 RegularPolygon(6,4) 和 RegularPolygon(10,4,5.6,7.8) 创建三个 RegularPolygon 对象，显示每个对象的周长和面积。

3. 实验过程

报告撰写具体要求：截屏显示或直接写出实验 1 至实验 3 的源码和运行结果，并完成实验 1 和实验 2 各实验后的练习（结果要写到实验报告中）。

实验内容（1）：

```
（1） /**
（2）  * \* Created with IntelliJ IDEA.
（3）  * \* User: lijihong
（4）  * \* Date: 2023-03-27
（5）  * \* Time: 14:06
```

```
（6） * \
（7） */
（8） public class _4_Village {
（9）     static int waterAmount; //模拟水井的水量
（10）     int peopleNumber; //村庄的人数
（11）     String name; //村庄的名字
（12）     _4_Village(String s) {
（13）         name = s;
（14）     }
（15）     static void setWaterAmount(int m) {
（16）         if(m>0)
（17）             waterAmount = m;
（18）     }
（19）     void drinkWater(int n){
（20）         if( waterAmount-n>=0) {
（21）             waterAmount = waterAmount-n;
（22）             System.out.println(name+"喝了"+n+"升水");
（23）         }
（24）         else
（25）             waterAmount = 0;
（26）     }
（27）     static int lookWaterAmount() {
（28）         return waterAmount;
（29）     }
（30）     void setPeopleNumber(int n) {
（31）         peopleNumber = n;
（32）     }
（33）     int getPeopleNumber() {
（34）         return peopleNumber;
（35）     }
（36） }
（37） /**
（38）  * \* Created with IntelliJ IDEA.
（39）  * \* User: lijihong
（40）  * \* Date: 2023-03-27
（41）  * \* Time: 14:07
（42）  * \
（43）  */
（44）
（45） public class _4_Land {
（46）     public static void main(String args[]) {
（47）         _4_Village.setWaterAmount(200); // 用类名调用
    _4_Village.setWaterAmount(int m),并向参数传值 200
```

```
（48）          int leftWater = _4_Village.waterAmount; //用 _4_Village 类的类名访
        问 waterAmount
（49）          System.out.println("水井中有 "+leftWater+" 升水");
（50）          _4_Village zhaoZhuang,maJiaHeZi;
（51）          zhaoZhuang = new _4_Village("赵庄");
（52）          maJiaHeZi = new _4_Village("马家河子");
（53）          zhaoZhuang.setPeopleNumber(80);
（54）          maJiaHeZi.setPeopleNumber(120);
（55）          zhaoZhuang.drinkWater(50);//zhaoZhuang 调 用 drinkWater(int n),并向参数传
        值 50
（56）          leftWater = maJiaHeZi.lookWaterAmount();//maJiaHeZi 调用 lookWaterAmount()
        方法
（57）          String name=maJiaHeZi.name;
（58）          System.out.println(name+"发现水井中有 "+leftWater+" 升水");
（59）          maJiaHeZi.drinkWater(100);
（60）          leftWater = zhaoZhuang.lookWaterAmount();//zhaoZhuang 调
        用 lookWaterAmount() 方法
（61）          name=zhaoZhuang.name;
（62）          System.out.println(name+"发现水井中有 "+leftWater+" 升水");
（63）          int peopleNumber = zhaoZhuang.getPeopleNumber();
（64）          System.out.println("赵庄的人口:"+peopleNumber);
（65）          peopleNumber = maJiaHeZi.getPeopleNumber();
（66）          System.out.println("马家河子的人口:"+peopleNumber);
（67）      }
（68）  }
```

```
水井中有 200 升水
赵庄喝了50升水
马家河子发现水井中有 150 升水
马家河子喝了100升水
赵庄发现水井中有 50 升水
赵庄的人口:80
马家河子的人口:120
```


实验三课后练习：

（1）答：不行，要设置成静态成员函数才行。

（2）答：可以，因为是静态的，所以谁调用都一样。

```
zhaoZhuang.drinkWater( n: 50);//zhaoZhuang 调 用drinkWater(int n),并向参数传值 50
leftWater = _4_Village.lookWaterAmount();//maJiaHeZi 调用 lookWaterAmount()方法
```

实验内容（2）：

```
（69）  /**
（70）   * \* Created with IntelliJ IDEA.
（71）   * \* User: lijihong
（72）   * \* Date: 2023-03-27
（73）   * \* Time: 14:38
（74）   * \
（75）   */
（76）  package tom.jiafei;
（77）  public class _4_SquareEquation {
（78）     double a,b,c;
（79）     double root1,root2;
（80）     boolean boo;
（81）     public _4_SquareEquation(double a,double b,double c) {
（82）        this.a=a;
（83）        this.b=b;
（84）        this.c=c;
（85）        if(a!=0)
（86）           boo=true;
（87）        else
（88）           boo=false;
（89）     }
（90）     public void getRoots() {
（91）        if(boo) {
（92）           System.out.println("是一元 2 次方程");
（93）           double disk=b*b-4*a*c;
（94）           if(disk>=0) {
（95）              root1=(-b+Math.sqrt(disk))/(2*a);
（96）              root2=(-b-Math.sqrt(disk))/(2*a);
（97）              System.out.printf("方程的根:%f,%f\n",root1,root2);
（98）           }
（99）           else {
（100）              System.out.printf("方程没有实根\n");
（101）           }
（102）        }
（103）        else {
（104）           System.out.println("不是一元 2 次方程");
（105）        }
（106）     }
（107）     public void setCoefficient(double a,double b,double c) {
（108）        this.a=a;
（109）        this.b=b;
```

```
（110）         this.c=c;
（111）         if(a!=0)
（112）             boo=true;
（113）         else
（114）             boo=false;
（115）     }
（116）  }
（117）  package tom;
（118）
（119）  import tom.jiafei.*;
（120）
（121）  /**
（122）   * \* Created with IntelliJ IDEA.
（123）   * \* User: lijihong
（124）   * \* Date: 2023-03-27
（125）   * \* Time: 14:39
（126）   * \
（127）   */
（128）  public class _4_SunRise {
（129）     public static void main(String args[]) {
（130）         _4_SquareEquation equation = new _4_SquareEquation(4,5,1);
（131）         equation.getRoots();
（132）         equation.setCoefficient(-3,4,5);
（133）         equation.getRoots();
（134）     }
（135）  }
```

```
"D:\IDEA\IntelliJ IDEA 2022.1.1\jbr\bin\java.exe" "-javaagent:D:\IDEA\IntelliJ IDEA 2022.1.1\lib\idea_rt.jar=60768:D:\IDEA\IntelliJ IDEA 2022.1.1\bin"
 -Dfile.encoding=UTF-8 -classpath D:\java_study_codes\out\production\java_study_codes tom._4_SunRise
是一元 2 次方程
方程的根:-0.250000,-1.000000
是一元 2 次方程
方程的根:-0.786300,2.119633

进程已结束,退出代码0
```

实验四的练习：

（1）运行编译结果是找不到或无法加载主类 SuqareEquation，因为 SuqareEquation.java 有包名，所以不能将它以及它的字节码文件存放在 D：\5000 中

（2）（3）都可以正常的编译运行。

实验内容（3）：

```
1.  /**
2.   * \* Created with IntelliJ IDEA.
3.   * \* User: lijihong
```

```
4.    * \* Date: 2023-03-27
5.    * \* Time: 14:01
6.    * \
7.    */
8.   public class _4_RegularPolygon {
9.       private int n;
10.      private double side;
11.      private double x;
12.      private double y;
13.
14.      public _4_RegularPolygon() {
15.          n = 3;
16.          side = 1;
17.          x = 0;
18.          y = 0;
19.      }
20.
21.      public _4_RegularPolygon(int n, double side) {
22.          this.n = n;
23.          this.side = side;
24.          x = 0;
25.          y = 0;
26.      }
27.
28.      public _4_RegularPolygon(int n, double side, double x, double y) {
29.          this.n = n;
30.          this.side = side;
31.          this.x = x;
32.          this.y = y;
33.      }
34.
35.      public int getN() {
36.          return n;
37.      }
38.
39.      public void setN(int n) {
40.          this.n = n;
41.      }
42.
43.      public double getSide() {
44.          return side;
45.      }
46.
```

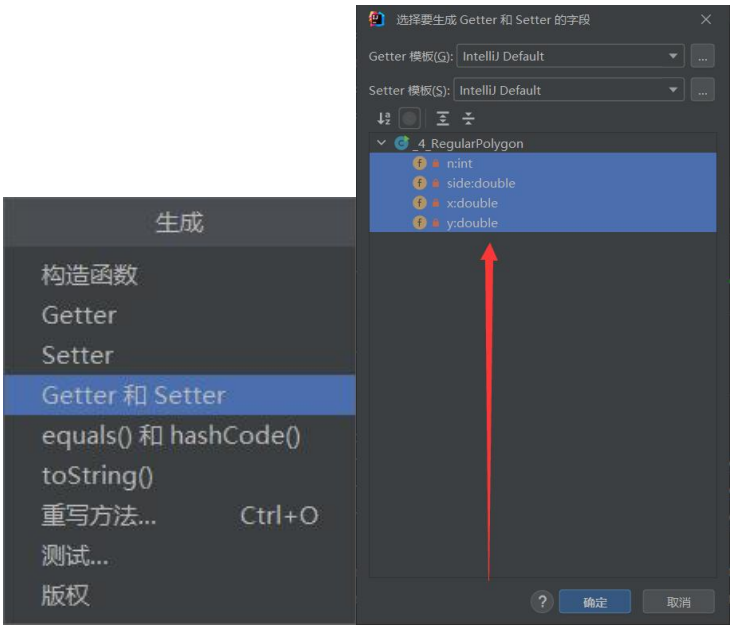
```
47.     public void setSide(double side) {
48.         this.side = side;
49.     }
50.
51.     public double getX() {
52.         return x;
53.     }
54.
55.     public void setX(double x) {
56.         this.x = x;
57.     }
58.
59.     public double getY() {
60.         return y;
61.     }
62.
63.     public void setY(double y) {
64.         this.y = y;
65.     }
66.
67.     public double getPerimeter() {
68.         return n * side;
69.     }
70.
71.     public double getArea() {
72.         return (n * side * side) / (4 * Math.tan(Math.PI / n));
73.     }
74.
75.     public static void main(String[] args) {
76.         _4_RegularPolygon polygon1 = new _4_RegularPolygon();
77.         _4_RegularPolygon polygon2 = new _4_RegularPolygon(6, 4);
78.         _4_RegularPolygon polygon3 = new _4_RegularPolygon(10, 4, 5.6, 7.8);
79.
80.         System.out.println("Polygon 1 perimeter: " + polygon1.getPerimeter());
81.         System.out.println("Polygon 1 area: " + polygon1.getArea());
82.
83.         System.out.println("Polygon 2 perimeter: " + polygon2.getPerimeter());
84.         System.out.println("Polygon 2 area: " + polygon2.getArea());
85.
86.         System.out.println("Polygon 3 perimeter: " + polygon3.getPerimeter());
87.         System.out.println("Polygon 3 area: " + polygon3.getArea());
88.     }
89. }
```

```
Polygon 1 perimeter: 3.0
Polygon 1 area: 0.43301270189221946
Polygon 2 perimeter: 24.0
Polygon 2 area: 41.569219381653056
Polygon 3 perimeter: 40.0
Polygon 3 area: 123.10734148701015
```

UML 图：

RegularPolygon
- n: int - side: double - x: double - y: double
+ RegularPolygon() + RegularPolygon(n: int, side: double) + RegularPolygon(n: int, side: double, x: double, y: double) + getN(): int + setN(n: int): void + getSide(): double + setSide(side: double): void + getX(): double + setX(x: double): void + getY(): double + setY(y: double): void + getPerimeter(): double + getArea(): double

其中所有数据域的访问器和修改器，可以使用集成化开发工具的快捷键生成对应的 getter() 和 setter():




```
0 个用法  新 *
public int getN() {
    return n;
}

0 个用法  新 *
public void setN(int n) {
    this.n = n;
}

0 个用法  新 *
public double getSide() {
    return side;
}

0 个用法  新 *
public void setSide(double side) {
    this.side = side;
}

0 个用法  新 *
public double getX() {
    return x;
}

0 个用法  新 *
public void setX(double x) {
    this.x = x;
}

0 个用法  新 *
public double getY() {
    return y;
}
```

4. 实验总结

写出实验中的心得体会（对第 4 章理论课重点简述）。

通过此次实验了解到了类的两种基本的成员：变量和方法，变量用来刻画对 象的属性，方法用来体现对象的功能，并且成员变量还可以分为实例变量和 静态变量或者类变量，大致掌握了类变量和实例变量，在此次实验中，在 Village 类中有一个静态成员变量 waterAmount。通过实验可以了解到实例 方法可操作实例成员变量和静态成员变量，静态方法只能操作静态成员变 量，通过实验后的练习 1 可以发现把代码 3 换成

Village.drinkWater（50） 程序就不可以运行；练习 2 可以把代码 4 换成 Village.lookWaterAmount（）； 练习 3 如果把 Land 类 main 方法中倒数第二行的代码改了之后会发现出现了 无法从静态上下文中引用非静态的错误。

此次实验使我们更加理解 Java 语言中的包，关键字 package 可以声明一个 包语句，并且 package 语句作为 Java 源文件的第一条语句，指明该源文件 定义的类所在的包；在写 Java 程序语句时，我们会用到很多不同的类，并 且这些类都存在于不同的包中，然后我们可以使用 import 语句来引入包中 类；import 语句必须写在 package 语句和源文件中类的定义之间。

重点总结：

（1）什么是封装？

答：封装的本质就是让类的调用者不必太多的了解类的实现者是如何实现类的, 只要知道如何使用类就行了. 这样就降低了类使用者的学习和使用成本, 从而降低了复杂程度

（2）private 关键字表示“访问权限控制”

被 private 修饰的成员变量或者成员方法，不能被类的调用者使用（其使用范围在类的{}内），外部不知道有其存在

在 JAVA 中，所谓权限修饰符指的是修饰的属性，方法，类，到底可见范围有多大。一共有四大访问修饰符，可见范围由小到大依次为：pivate<(default)<protected<public

注意： private 不能修饰外部类

（3）当 set 方法的形参名字和类中的成员属性的名字一样的时候，如果不使用 this, 根据就近匹配原则，编译器会找最近的名字相同的变量，相当于自赋值 ， this 表示当前实例 的引用.

不是所有的字段都一定要提供 setter / getter 方法, 而是要根据实际情况决定提供哪种方法.

在 IDEA 中可以使用 alt + insert (或者 alt + F12) 快速生成 setter / getter 方法.

（4）this 关键字

1.调用当前对象的成员变量

2.调用当前变量的成员方法

调用类中普通成员方法

调用构造方法（若类中不同参数的构造方法之间出现了重复代码，可使用 this.(参数)调用其他构造方法）

3.表示当前对象的引用，当前通过哪个对象调用的属性或方法 this 就代表谁（相当于照镜子）

注意：

this 调用其他构造方法必须放在构造方法首行，否则会报错

this 调用不能成环

this( ) 不能在普通方法中使用，只能写在构造方法中。

（5）构造方法的使用

构造方法是一种特殊方法, 使用关键字 new 实例化新对象时会被自动调用, 用于完成初始化操作.

new 执行过程：

在堆中为对象分配内存空间

调用对象的构造方法，为对象成员变量赋值，成员变量的默认值就是在构造方法中赋值的

语法规则：

方法名称必须与类名称相同

构造方法没有返回值类型声明，默认返回值就是对象类型本身

每一个类中一定至少存在一个构造方法（没有明确定义，则编译器自动生成一个无参构造）
若类中定义了构造方法，则默认的无参构造将不再生成
构造方法支持重载，参数可以是 0 个 1 个…n 个
（6）static 关键字的使用
1.修饰属性(类属性，类变量)
2.修饰方法(类方法，具方法)
3.修饰代码块(静态代码块)
4.修饰类(静态内部类)
a)修饰属性
1.当一个成员变量被 statice 变量修饰，它就表示类的一个属性，该类的所有对象都共享这个属性(与类强相关)。
2.statice 修饰的属性在 JVM 方法区中存储，所以该类对象共享这个属性
方法区储存:
1.所有类中方法
2.常量，静态变量
b）修饰方法
如果在任何方法上应用 static 关键字，此方法称为静态方法。
结论：
静态方法属于类，而不属于类的对象。
可以直接通过类.方法名 调用静态方法，而无需创建类的实例。
静态方法可以访问静态方法和静态属性，并可以更改静态属性的值。
静态方法不可以访问成员方法和属性 （静态方法无须通过对象调用，没有对象无法访问成员方法和属性）
在成员方法中既可以调用静态方法也可以调用成员方法
一般将工具类的方法设计为 static 方法，如 Arrays 提供的操作数组的方法
思考：为什么主方法是静态方法？
答：主方法是程序的入口，程序从主方法开始执行，静态方法无需对象就能执行，若主方法是个成员方法，就得通过对象调用，此时都还没对象如何执行。
c）全局常量
若在类中定义了一个常量，我们通常情况下都会把 static 和 final 共同使用，成为类的常量（全局常量）
注意 ：
静态的常量，属于类本身，只有一份 被 final 修饰，后续不可更改
类中常量必须在定义时赋值
类的常量的优点：共享属性，节省空间
常量命名规则：所有字母大写，多个单词由_隔开，如 static final String STDENT_SCHOOL=“福清一中”
（7）代码块
使用 {} 定义的一段代码成为代码块.
根据代码块定义的位置以及关键字，又可分为以下四种：
普通代码块
构造代码块
静态代码块

同步代码块
普通代码块：定义在方法中的代码块
这种用法较少见
构造代码块
构造块：定义在类中的代码块(不加修饰符)。也叫：实例代码块。构造代码块一般用于初始化实例成员变量。
注意：
构造代码块比构造方法先执行
new 几个对象构造代码块执行几次
静态代码块
使用 static 定义的代码块。一般用于初始化静态成员属性。
结论：

定义在类中，类加载时执行一次与对象无关，无论产生几个对象都只执行一次

静态代码块优先于构造块和构造方法执行

当类中存在静态变量，静态代码块【1…N】，类加载时，这些静态变量的赋值以及静态代码块最终合并为一个大的静态代码块，由 JVM 执行

主类的静态块优先主方法执行，JVM 要执行主方法，首先得加载主类，主类一加载，静态块就执行了