Java 程序读与写

（2011 年高级程序设计语言实验教程草稿）

第 一 章. 环境安装与基本 I/O

1. 实验目的

（1） 下载和安装免费软件 JDK 和 Eclipse。

（2） 使用 Scanner 和 System.out 实现控制台的输入与输出。

2. 读程序

2.1 “Hello World”

```
package basic;

import java.util.Scanner;

/**
 * Hello World, 第一个 Java 程序
 * @author Dahogn
 * @version 1.01
 * @since 2009.10.5
 *
 */
public class HelloWorld {
	/**
	 * @param args 此程序中没有使用
	 */
	public static void main(String[] args) {
	/*调用 System.out 实现控制台输出*/
		System.out.println("Hello World");//打印"Hello World"
	}
}
```

2.2 Scanner

```
package basic;

import java.util.Scanner;

public class HelloWorld {

	public static void main(String[] args) {
		/*通过控制台输入姓名，输出到控制台*/
		System.out.println("What's your name:");
		Scanner scan = new Scanner(System.in);
		String name = scan.nextLine();  //存入字符串 name
		System.out.println("Hello, " + name);
```

```
    }
}
```

3. 实验过程与写程序

3.1 JDK 和 Eclipse 的安装

J2SE 安装：下载 J2SE JDK (http://java.sun.com/javase/downloads/index.jsp)，双击自动安装；

Eclipse 安装：下载 Eclipse (http://www.eclipse.org/downloads/index.php) ，解压缩，不需要安装。

熟悉 Eclipse 的使用，包括新建项目和编译运行程序。

3.2 读入一个符号，打印出此符号组成的菱形图

实验描述：输入一个符号，比如字符“A”，打印出一个菱形图，如下图所示：

```
   A
  AA
 AAA
AAAA
 AAA
  AA
   A
```

其中，菱形的行数不限，至少为 4 行。

4. 思考题

（1） 不使用任何 IDE，使用命令行完成 Java 程序的编译与运行。

（2） 尝试使用其他的 IDE，比如 Netbeans 和 JDeveloper。

（3） 控制 3.2 中所打印菱形图的行数。

第 二 章. 基本数据类型

1. 实验目的

（1） 学会使用基本的数据类型。

（2） 能够对数据进行基本的算术运算。

（3） 了解强制类型转化。

2. 读程序

```
package basic;

public class Primitive {

	public static void main(String[] args) {

		int iNumber1=0,iNumber2;
		float fNumber1,fNumber2;
		double dNumber1,dNumber2;
		char cChar;

		//iNumber2 的值还是为 0，iNumber1 的值为 1
		iNumber2=(iNumber1++);
		System.out.println("1.The Value of iNumber2 is "+iNumber2+";The Value of iNumber1
is "+iNumber1);
		//iNumber2 的值还是为 2，iNumber1 的值为 2
		iNumber2=(++iNumber1);
		System.out.println("2.The Value of iNumber2 is "+iNumber2+";The Value of iNumber1
is "+iNumber1);
		//iNumber2 的值为 9
		iNumber2+=17;
		System.out.println("3.The Value of iNumber2 is "+iNumber2);

		//整数的除法
		iNumber2=iNumber2/iNumber1;
		System.out.println("4.The Value of iNumber2 is "+iNumber2);
		dNumber1=iNumber2/2.0;
		System.out.println("5.The Value of dNumber1 is "+dNumber1);
		dNumber1=iNumber2/(double)iNumber1;
		System.out.println("6.The Value of dNumber1 is "+dNumber1);

		//浮点数精度问题
		fNumber1=(float)2.0;
		fNumber2=(float)1.2;
		fNumber1=fNumber1-fNumber2;
		System.out.println("7.The Value of fNumber1 is "+fNumber1);

		dNumber1=2.0;
```

```
        dNumber2=1.1;
        dNumber1=dNumber1-dNumber2;
        System.out.println("8.The Value of dNumber1 is "+dNumber1);

        //char  的 Unicode 问题
        cChar='a';
        iNumber2=(int)cChar;
        System.out.println("9.The Value of iNumber2 is "+iNumber2+";The cChar is "+cChar);

        //转义字符（牢记）
        cChar='\';
        System.out.println("10.The cChar is "+cChar);
    }
}
```

3. 写程序

3.1 简单的数学运算程序

实验描述：输入两个操作数（操作数 1 和操作数 2），输入需要进行的计算的符号，得到数学运算的结果。

比如，通过控制台输入操作数 1，值为“100”；输入操作数 2，值为“200”；输入操作符号“+”；得到运算结果 300。

3.2 简单的 Unicode 查找程序

实验描述：输入十进制数字能够得到对应的 Unicode 字符，输入 Unicode 字符能够得到对应的是十进制数字编码。

比如，通过控制台输入数字 97，输出字符 a；输入字符 a，输出数字 97。

4. 思考题

（1） 在程序 3.1 中，如果输入为字符或者字符串，程序如何处理。

（2） 在程序 3.1 中，如果输入的整数长度超过 long 的最大极限，程序如何处理。

（3） 在程序 3.2 中，如果输入的十进制数字为负数，会得到怎样的结果。

第 三 章. 循环与分支语句

1. 实验目的

（1） 了解和熟悉布尔运算。

（2） 熟练使用分支语句和循环语句控制程序执行流程。

2. 读程序

2.1 “BoolResult”

```
package basic;

public class BoolResult {
	public static void main(String[] args) {
		int a = 5;
		int b = -5;
		int c = 20;
		int d = -20;

		/*
		 * && 优先级高于 ||，即 && 先计算
		 */
		if ((b-- < -5) && (a++ < 5) || (c < 30)) {
			System.out.println("HERE FIRST");
		}
		System.out.println("FIRST a is " + a);
		System.out.println("FIRST b is " + b);

		/*
		 * 只要 || 左边的表达式值为 true，则不计算右边的表达式的值，整个表达式值为 true。
		 */
		a = 5;
		b = -5;
		if ((c < 30) || (b-- < -5) && (a++ < 5)) {
			System.out.println("HERE Second");
		}
		System.out.println("Second a is " + a);
		System.out.println("Second b is " + b);

		a = 5;
		b = -5;
		if ((c < 20) || (a++ < 5) && (b-- < -5) || (d++ < -10)) {
			System.out.println("HERE Third");
		}
		System.out.println("Third a is " + a);
		System.out.println("Third b is " + b);
```

```
        System.out.println("Third d is " + d);

        /*
         * 运算符 && 左边的表达式值若为 false，则不用计算 && 右边的表达式的值，整个 && 表达式值为 false。
         */
        a = 5;
        b = -5;
        if ((c < 20) && (b-- < -5) || (a++ < 15)) {
            System.out.println("HERE Fourth");
        }
        System.out.println("Fourth a is " + a);
        System.out.println("Fourth b is " + b);

        a = 5;
        b = -5;
        if ((c < 20) && (b-- < -5) && (a++ < 15)) {
            System.out.println("HERE Fifth");
        }
        System.out.println("Fifth a is " + a);
        System.out.println("Fifth b is " + b);

        a = 5;
        b = -5;
        d = -20;
        if ((c < 30) && (b-- < -5) && (a++ < 15) || (d++ < -10)) {
            System.out.println("HERE Sixth");
        }
        System.out.println("Sixth a is " + a);
        System.out.println("Sixth b is " + b);
        System.out.println("Sixth d is " + d);
    }
}
```

2.2 “Branch”

```
package basic;

public class Branch {

    public static void main(String[] args) {

        int iNumber1=5,iNumber2;
        final int iNumber3=20;
        iNumber2=10;
        //iNumber1++进行了计算，进行判断时 iNumber1 值为 5
        if(iNumber1++>5)
            System.out.println("1.The Value of iNumber2 is "+iNumber2+";The Value of iNumber1 is "+iNumber1);
        //写和不写 else 大不一样
        System.out.println("2.The Value of iNumber2 is "+iNumber2+";The Value of iNumber1 is "+iNumber1);

        //iNumber1++没有计算
        if(iNumber2<20||iNumber1++>5)
        {
```

```
                System.out.println("3.The Value of iNumber2 is "+iNumber2+";The Value of iNumber1 is "+iNumber1);

            }

            if(iNumber2>8)

                if(iNumber1>6)
                {
                System.out.println("4.The Value of iNumber2 is "+iNumber2+";The Value of iNumber1 is "+iNumber1);
            }
            else//匹配"if(iNumber1>4)"
            System.out.println("5.The Value of iNumber2 is "+iNumber2+";The Value of iNumber1 is "+iNumber1);

            iNumber2=5;
            switch(iNumber2)
            {
            case 5:
                System.out.println("6.The Value of iNumber2 is "+iNumber2);
            case 10:
                System.out.println("7.The Value of iNumber2 is "+iNumber2);
            case 15:
                System.out.println("8.The Value of iNumber2 is "+iNumber2);
                break;
            case iNumber3:
                System.out.println("9.The Value of iNumber2 is "+iNumber2);
                break;
            default:
                System.out.println("10.The Value of iNumber2 is "+iNumber2);
            }

            //使用 ?:
            iNumber2=iNumber1>10?20+iNumber1:30+iNumber1;
            System.out.println("11.The Value of iNumber2 is "+iNumber2);
        }
}
```

2.3 “Loop”

```
package basic;

public class Loop {

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        int iNumber1=10,count=0;
        //计算 10 加到 19
        while(iNumber1<20)
        {
            count+=iNumber1;
            iNumber1++;
        }
        System.out.println("1.The Value of count is "+count+"; The Value of iNumber1 is "+iNumber1);
```

```
        iNumber1=10;
        count=0;
        do
        {
            iNumber1++;
            count+=iNumber1;
        }while(iNumber1<20);
        System.out.println("2.The Value of count is "+count+"; The Value of iNumber1 is "+iNumber1);

        iNumber1=10;
        count=0;
        do
        {
            count+=iNumber1;
            iNumber1++;
        }while(iNumber1<10);
        System.out.println("3.The Value of count is "+count+"; The Value of iNumber1 is "+iNumber1);

        iNumber1=10;
        count=0;
        while(iNumber1<10)
        {
            count+=iNumber1;
            iNumber1++;
        }
        System.out.println("4.The Value of count is "+count+"; The Value of iNumber1 is "+iNumber1);

        iNumber1=10;
        count=0;
        for(;iNumber1<20;)
        {
            count+=iNumber1;
            iNumber1++;
        }
        System.out.println("5.The Value of count is "+count+"; The Value of iNumber1 is "+iNumber1);

        for(iNumber1=10,count=0;iNumber1<20;count+=iNumber1,iNumber1++);
        System.out.println("6.The Value of count is "+count+"; The Value of iNumber1 is "+iNumber1);

        /*注意作用域
        for(int iNumber2=10,count2=0;iNumber1<20;count2+=iNumber2,iNumber2++);
        System.out.println("7.The Value of count2 is "+count2+"; The Value of iNumber2 is "+iNumber2);
        */

        iNumber1=10;
        count=0;
        do
        {
            count+=iNumber1;
            iNumber1++;
```

```
                if(iNumber1==20)
                    break;//只能终止一层循环
            }while(true);
            System.out.println("7.The Value of count is "+count+"; The Value of iNumber1 is "+iNumber1);


            iNumber1=10;
            count=0;
            for(;;)
            {

                count+=iNumber1;
                iNumber1++;
                if(iNumber1==20)
                    break;
            }
            System.out.println("8.The Value of count is "+count+"; The Value of iNumber1 is "+iNumber1);

            //continue
            iNumber1=10;
            do
            {
                iNumber1++;
                if(iNumber1%2==0)
                {
                    System.out.println("9.The Value of iNumber1 is "+iNumber1);
                    continue;
                }
            }while(iNumber1<20);


            for(iNumber1=2,count=0;iNumber1<10;iNumber1++)
            {
                System.out.println("10.The Value of iNumber1 is "+iNumber1);
                count+=iNumber1;
                System.out.println("11.The Value of count is "+count+"; The Value of iNumber1 is "+iNumber1);
            }
            System.out.println("12.The Value of count is "+count+"; The Value of iNumber1 is "+iNumber1);
        }
}
```

3. 写程序

3.1 “WSAD”方向控制

实验描述：编写一个程序，当键盘输入为“WSAD”这四个按键中的其中一个（大小写皆可），输出相对应的方向。比如用户输入“W”，输出“左”。

3.2 Hi-Lo 猜猜游戏程序

实验描述：从 1-1000 中随机选择一个数，反复让用户猜该数字是什么，直到用户猜对或用户退出为止。每猜一次告诉用户猜测的结果是对还是过大或是过小。使用一个标识值确定用户是否想退出。当用户猜对时报告其猜测的次数。每次游戏结束时询问用户是否想继续玩，直到用户选择结束。

3.3 日历输出程序

实验描述：输入一个月份，给出 2011 年这个月的日历，日历要求每个月显示 4~5 行，每行显示 7 列，对应星期一到星期日；输入一个年份，系统自动输出一整年的详细的日历，要求每行同时显示三个月，月日历显示方式同上。

4. 思考题

（1） 改写第二章中的写程序题“3.1 简单的数学运算程序”，能够对“除以零”等情况进行排除，并且能够实现连续的数学表达式输入，根据括号等运算法则计算结果。比如输入“（1+2）*3+4*5”，能够得到 29。

（2） 对于 3.3 中的程序，能够输出中国农历。

第 四 章. 设计与使用类

1. 实验目的

（1） 了解和熟悉面向对象的基本思想和类的设计与实现。

（2） 熟悉类的构造方法、类的封装性特征。

2. 读程序

2.1 "Encapsulation"

```
public class Encapsulation    {

    private int number1;
    private int number2;//默认是 public

    //构造方法的使用
    Encapsulation() {
        number1 = 5;
        number2 = 6;
    }

    Encapsulation(int num1, int num2) {
        number1 = num1;
        number2 = num2;
    }

    public int getNumber1() {
        return number1;
    }

    public void setNumber1(int number1) {
        this.number1 = number1;
    }

    public int getNumber2() {
        return number2;
    }

    public void setNumber2(int number2) {
        this.number2 = number2;
    }

    public void SetOperator(int num1, int num2) {
        number1 = num1;
        number2 = num2;
    }
```

```
	public int Add() {
		return number1 + number2;
	}

	public int Sub() {
		return number1 - number2;
	}

	public static void main(String[] args) {
		Encapsulation optFourth1 = new Encapsulation();
		optFourth1.setNumber1(20);
		System.out.println("1. First Step of optFourth1.Add is "
				+ optFourth1.Add());

		Encapsulation optFourth2 = new Encapsulation(34, 35);
		System.out.println("2. Second Step of optFourth2.Add is "
				+ optFourth2.Add());

		optFourth2.SetOperator(45,46);
		System.out.println("3. Third Step of optFourth2.Add is "
				+ optFourth2.Add());
	}
}
```

2.2 “Overwriting”

```
/**
	说明函数 Overwriting 的使用
*/
public class Overwriting {
	public int Add(final int number1,final int number2)
	{
		System.out.println("Use int");
		return (number1+number2);
	}
	public int Add(int number1,int number2,int number3)
	{
		System.out.println("Use int 3");
		return this.Add(this.Add(number1,number2),number3);
	}
	public double Add(double number1,double number2)
	{
		System.out.println("Use double");
		return (number1+number2);
	}
	public float Add(float number1,float number2)
	{
		System.out.println("Use float");
		return (number1+number2);
	}
/*	形参列表相同而返回值不同，不行
	public double Add(float number1,float number2)
	{
		System.out.println("Use float");
		return (double)(number1+number2);
	}
```

```
    */
    public short Add(short number1,short number2)
    {
        System.out.println("Use short");
        return (short)(number1+number2);//默认返回 int，必须强制类型转化
    }
    public int Add(int number1,short number2)
    {
        System.out.println("Use int and short");
        return (number1+number2);//默认返回 int，必须强制类型转化
    }
    public int Add(short number1,int number2)
    {
        System.out.println("Use short and int");
        return (number1+number2);//默认返回 int，必须强制类型转化
    }
    public static void main(String[] args) {
        Overwriting add=new Overwriting ();
        System.out.println(add.Add(Integer.MAX_VALUE, Integer.MAX_VALUE));
        add.Add(4, 5, 6);
        add.Add(4.5, 5.3);
    }
}
```

3. 写程序

3.1 复数类

复数是指具有实部和虚部的数。请定义一个复数类，实现复数的加法、减法、乘法运算。

3.2 背着书包买书

一位同学背着一个书包去书店买书，请用面向对象的思想构建类“书（Book）”，“书包（Bag）”，完成买书的过程。

4. 思考题

第 五 章. 类的继承与高级特性

1. 实验目的

（1） 熟悉类的继承关系及类的多态性。

（2） 熟悉类的接口的概念。

2. 读程序

请把以下四个程序放在一起阅读。

2.1 “Person”

```
package inheritance;
/**
 * 简单的 Person 类，说明继承关系。具有一个字符串类型的变量 name
 * @author Dahogn
 * @version 1.01
 * @since 2009.11.15
 *
 * 。
 */
public class Person implements Serializable{// 继承自 Object 类可以不写，Object 类是所有 Java 类的父类
	private String name; // 人名，字符串形式
	/**
	 * 默认构造方法。如果一个类没有构造方法，存在一个形参为空的默认构造方法； 只要声明了任何一个构造方法，那么默认构造方法就无法调用；
	 * 如果还想使用形参为空构造对象，就需要创建默认构造方法。
	 */
	public Person() {
		System.out.println("Person()");
		name = "No";
	}

	/**
	 * 初始化类的实例变量 name，这是具有一个形参的构造方法
	 * @param initialName
	 *
	 */
	public Person(String initialName) {
		name = initialName;
	}

	/**
```

```
     * set 方法，一般情况下，类的属性声明为 private，通过 set 和 get 来设置和调用
     * @param newName
     *
     */
    public void setName(String newName) {
        name = newName;
    }

    /**
     * get 方法，一般情况下，类的属性声明为 private，通过 set 和 get 来设置和调用
     * @return
     *
     */
    public String getName() {
        return name;
    }

    public void writeOutput() {
        outputClassName();
        System.out.println("Name: " + name);
    }

    /**
     *
     * @param otherPerson
     * @return  如果忽略大小写，名字匹配，则返回真
     */
    public boolean sameName(Person otherPerson) {
        return (this.name.equalsIgnoreCase(otherPerson.name));
    }

    public void outputClassName() {
        System.out.println("Person");
    }
}
```

2.2 “Student”

```
package inheritance;
/**
 * 说明类的继承关系，Student 类继承自 Person
 * @author Dahogn
 * @version 1.01
 * @since 2008.11.15
 *
 */
public class Student extends Person {
    private int studentNumber;// 学生还有学号

    public Student() {
        super();// 必须是第一句
        System.out.println("Student()");
        // 在形参列表为空的构造方法里面，出现 super（），写和不写一样
        studentNumber = 0;// Indicating no number yet
```

```
    }

    public Student(String initialName, int initialStudentNumber) {
        super(initialName);// 必须是第一句

        studentNumber = initialStudentNumber;
    }

    public Student(String initialName) {

        this(initialName, 0);// 可以调用同一个类里面的其他构造方法
    }

    public void reset(String newName, int newStudentNumber) {
        setName(newName);
        studentNumber = newStudentNumber;
    }

    public int getStudentNumber() {
        return studentNumber;
    }

    public void setStudentNumber(int newStudentNumber) {
        studentNumber = newStudentNumber;
    }

    /**
     * 覆盖 Person 类中定义的 writeOutput()方法， 因为两个方法名称一样，形参列表一样，此时返回值类型必定一样，否则会出错
     * 所以将使父类方法无效。
     */
    public void writeOutput()
    // 如果此处定义为 Final，Undergraduate 类中的同名函数（writeOutput）将定义错误
    {

        // System.out.println("Name: " + getName( ));
        super.writeOutput();// 和上一句等同
        System.out.println("Student Number: " + studentNumber);
    }

    /**
     * 一段说明 重载（overloading）writeOutput 方法， 因为两个方法名称一样，形参列表不一样
     * @param notes
     *
     */
    public void writeOutput(String notes) {
        this.writeOutput();
        System.out.println("notes:" + notes);
    }

    /**
     * Object 类固有的方法 这是一个较好的 equals 方法，形参类型为所有类的父类 Object,这样允许 equals 方法更灵活适用于存在继承的情况
     */
```

```
    public boolean equals(Object otherObject) {
        if (otherObject == null)
            return false;
        else if (!(otherObject instanceof Student))
            return false;
        else {
            Student otherStudent = (Student) otherObject;
            return (this.sameName(otherStudent) && (this.studentNumber ==
otherStudent.studentNumber));
        }
    }

    /**
     * Object 类固有的方法 能够将类的信息打印输出，使类可以在 Java 中正确打印
     */
    public String toString() {
        return ("Name=" + getName() + "\r\nStudent number=" + studentNumber);
    }

    public String getName(String personName) {
        return personName;
    }

    public void outputClassName() {
        System.out.println("Student");
    }

    public static void main(String[] args) {

        Student student1 = new Student("Tom", 12345);
        student1.writeOutput();
        System.out.println(student1);//由于定义了 tostring，可以正确输出
        Person person1 = new Student("Tom", 12345);

        // 显示 equals（）定义中形参为 Object 类型的重要性，
        // 如果定义为 equals(Student otherstudent),此处将无法进行比较
        if (student1.equals(person1))
            System.out.println("person1 equals student1");
        else
            System.out.println("person1 not equals student1");

        int studNum = student1.getStudentNumber();
        System.out.println("studNum:" + studNum);
        // int studNum2=person1.getStudentNumber();将不能使用
        Student student2=(Student)person1;
        int studNum2 = student2.getStudentNumber();
        System.out.println("studNum2:" + studNum2);
        student2.getName();
    }
}
```

2.3 “Undergraduate”

```
package inheritance;
/**
```

```
 * 说明类的继承关系，Undergraduate 类继承自 Student，Java 里面每个类只能继承自一个父类
 * @author Dahogn
 * @version 1.01
 * @since 2009.11.15
 *
 */
public class Undergraduate extends Student
{
	private int level; //1 for freshman, 2 for sophomore,
                                //3 for junior, or 4 for senior.

	public Undergraduate( )
	{
		super( );
		System.out.println("Undergraduate( )");
		level = 1;
	}

	public Undergraduate(String initialName,
                            int initialStudentNumber, int initialLevel)
	{
		super(initialName, initialStudentNumber);
		setLevel(initialLevel); //Checks 1 <= initialLevel <= 4
	}

	public void reset(String newName,
                                    int newStudentNumber, int newLevel)
	{
		setName(newName);
		reset(newName, newStudentNumber);//调用 Student 类的方法
		setLevel(newLevel); //Checks 1 <= newLevel <= 4

	}

	public int getLevel( )
	{
		return level;
	}

	public void setLevel(int newLevel)
	{
		if ((1 <= newLevel) && (newLevel <= 4))
			level = newLevel;
		else
		{
			System.out.println("Illegal level!");
			System.exit(0);
		}
	}

	public void writeOutput( )
	//覆盖父类(Student)中定义的 writeOutput
	{

		super.writeOutput( );
		//super.super.writeOutput( );  是不合法的
```

```
            System.out.println("Student Level: " + level);
      }

      public boolean equals(Undergraduate otherUndergraduate)
      //一个不太好的 equals 方法，应该定义为 Object 形参
      {
            return (super.equals(otherUndergraduate)
                  && (this.level == otherUndergraduate.level));
      }
      public void outputClassName() {
            System.out.println("Undergraduate");
      }
      public static void main(String[] args) {

            Undergraduate ug=new Undergraduate();
            System.out.println(ug);//显示构造方法的调用次序

            Undergraduate undergraduate1 = new Undergraduate("Tom", 12345,1);
            undergraduate1.writeOutput();//根据 Person 类中的 writeOutput()定义，
                  //将调用 outputClassName()方法，在运行时调用到 Undergraduate 中的方法

      }

}
```

2.4 “UndergraduateDemo”

```
package inheritance;
/**
 * 说明使用对象作为方法形参和作为方法返回值的例子
 *
 * @author Dahogn
 * @version 1.01
 * @since 2009.11.15
 *
 */
public class UndergraduateDemo {
      private Undergraduate undergraduate;

      public Undergraduate getUndergraduate() {
            return undergraduate;
      }

      public void setUndergraduate(Undergraduate undergraduate) {
            this.undergraduate = undergraduate;
      }


      public UndergraduateDemo(String initialName, int initialStudentNumber,
                  int initialLevel) {
            undergraduate = new Undergraduate(initialName, initialStudentNumber,
                        initialLevel);
      }

      public void reset(String initialName, int initialStudentNumber,
                  int initialLevel) {
```

```
        undergraduate = new Undergraduate(initialName, initialStudentNumber,
                initialLevel);
    }

    public void upgrade() {
        if (undergraduate == null)
            System.out.println("undergraduate is null");
        else {
            int level = undergraduate.getLevel();
            if (level < 4)
                undergraduate.setLevel(level + 1);
            else
                undergraduate.setLevel(4);
        }
    }

    public void showUpgradeAddOne(int level) {
        level = level + 1;
        System.out.println("undergraduate    AddOne is:" + level);
    }


    // 一般声明为 static
    public static void ShowUndergraduate(Object object) {
        if (object == null)
            System.out.println("Object is null");
        else if (object instanceof Undergraduate) {
            Undergraduate ug = (Undergraduate) object;
            ug.writeOutput();
        } else
            System.out.println("Object is wrong");
    }
    public static void ShowUndergraduate(Undergraduate ug) {

            ug.writeOutput();

    }
    // 一般声明为 static
    public void UpgradeUndergraduate(Undergraduate ug) {
        int level = ug.getLevel();
        if (level < 4)
            ug.setLevel(level + 1);
        else
            ug.setLevel(4);
    }

    // 一般声明为 static
    public static void UpgradeUndergraduate(Object object) {
        if (object == null)
            System.out.println("Object is null");
        else if (object instanceof Undergraduate) {
            Undergraduate ug = (Undergraduate) object;
            int level = ug.getLevel();
            if (level < 4)
                ug.setLevel(level + 1);
            else
                ug.setLevel(4);
```

```
            } else
                System.out.println("Object is wrong");

        }

        public Undergraduate InitUndergraduate(String initialName,
                int initialStudentNumber, int initialLevel) {
            Undergraduate temp = new Undergraduate(initialName,
                    initialStudentNumber, initialLevel);
            return temp;
            // 或者直接写  return new
            // Undergraduate(initialName,initialStudentNumber,initialLevel);
        }

        public static void main(String[] args) {


            Undergraduate undergraduate1 = new Undergraduate("Tom", 12345, 1);
            UndergraduateDemo demo = new UndergraduateDemo("Tom", 12345, 1);

            System.out.println("====Step1====");
            demo.ShowUndergraduate(undergraduate1);

            System.out.println("====Step2====");

            demo.UpgradeUndergraduate(undergraduate1);

            demo.showUpgradeAddOne(undergraduate1.getLevel());

            undergraduate1.writeOutput();// 注意此时 level 的值将变了

            System.out.println("====Step3====");
            Undergraduate undergraduate2 = demo
                    .InitUndergraduate("Jerry", 12346, 2);
            demo.ShowUndergraduate(undergraduate2);

            System.out.println("由于定义了以 Object 为形参的方法，以下程序才能正确执行");
            Person person = new Undergraduate("Tom", 12345, 1);

            System.out.println("====Step4====");
            UndergraduateDemo.ShowUndergraduate(person);

            System.out.println("====Step5====");
            UndergraduateDemo.UpgradeUndergraduate(person);
            person.writeOutput();// 注意此时 level 的值将变了

            System.out.println("====Step6====");
            Student student = demo.InitUndergraduate("Jerry", 12346, 2);
            UndergraduateDemo.ShowUndergraduate(student);
        }
}
```

3. 写程序

3.1 完整的形状类

矩形、正方形、椭圆、形圆形、六边形、正六边形都是形状，请以形状（Shape）为最顶层的类，设计出一个层次化的类结构，至少能够对每个形状命名，并求面积、周长、重心。

3.2 植物与僵尸

“植物大战僵尸”是近几年比较风靡的一款益智策略类塔防御战游戏，其中的角色主要分成植物和僵尸两类，其植物部分角色如下：

名称	英文名称	类别	攻击力	防御力	价格	特点
豌豆射手	Peashooter	攻击类	20	300	100	攻击僵尸
向日葵	SunFlower	生产阳光	0	300	50	产生 25 点阳光
樱桃炸弹	Cherry Bomb	攻击类	1800	300	150	攻击周围僵尸
坚果	Wall Nut	抵御攻击	0	4000	50	阻止僵尸前进
阳光菇	Sun-shroom	生产阳光	0	300	25	生产 15 点阳光
大喷菇	Fume shroom	攻击类	20	300	75	攻击僵尸

请根据以上特点，以植物类（Plants）为最顶层的类，设计以上的植物角色。

4. 思考题

请基于 3.2，设计一个简单的“植物与僵尸”游戏。不需要图形界面，命令行游戏即可。

第六章. 数组与异常

1. 实验目的

（1） 熟悉数组的使用和原理。

（2） 熟悉异常的使用和原理。

2. 读程序

2.1 “ArrayTest”

```
public class ArrayTest {

	private Student[] studentList;
	private int studentNumber;

	public ArrayTest() {
		studentNumber = 5;
		studentList = new Student[studentNumber];
		for (int i = 0; i < studentNumber; i++)
			studentList[i] = new Student("Student" + i * i, i * i);
	}

	public void initArray() {
		System.out.println("Enter number of the Students:");
		Scanner scan = new Scanner(System.in);
		studentNumber = scan.nextInt();
		studentList = new Student[studentNumber];
		System.out.println("Enter the Start number of the StudentNumber:");
		int tempInt = scan.nextInt();
		for (int i = 0; i < studentNumber; i++)
			studentList[i] = new Student("Student" + tempInt, tempInt);
	}

	public void OutputStudent(Student stud) {
		stud.writeOutput();
	}

	public void OutputStudentList() {
		for (int i = 0; i < studentNumber; i++)
			studentList[i].writeOutput();
		// 等同于 OutputStudent(studlist[i]);
	}

	// 输入的 studlist 长度可变
	public static void OutputStudentList(Student[] studlist) {
		for (int i = 0; i < studlist.length; i++)
			studlist[i].writeOutput();
```

```
        // 等同于 OutputStudent(studlist[i]);
    }

    public static void ChangeStudentList(Student[] studlist) {
        for (int i = 0; i < studlist.length; i++) {
            studlist[i].setName("Student" + i * i * i);
            studlist[i].setStudentNumber(i * i * i);
        }
    }

    // 判断两个 Student 的数组是否每个元素相等
    public boolean equals(Student[] studlist, Student[] studlist2) {
        boolean match;
        if (studlist.length != studlist.length)
            match = false;
        else {
            match = true; // tentatively
            int i = 0;
            while (match && (i < studlist.length)) {
                if (!studlist[i].equals(studlist2[i]))
                    match = false;
                i++;
            }
        }
        return match;
    }

    public Student[] SetOneStudentList(int length) {
        Student[] temp = new Student[length];
        for (int i = 0; i < temp.length; i++)
            temp[i] = new Student("Student" + i * i, i * i);
        return temp;
    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("====Step1====");
        // 基本类型的数据初始化
        int[] studentNumber = new int[5];

        for (int i = 0; i < 5; i++) {
            studentNumber[i] = i * i;
        }
        // 数组大小为初始化元素的个数
        int[] studentNumber2 = { 0, 1, 4, 9, 16 };
        if (studentNumber == studentNumber2)
            System.out.println("studentNumber == studentNumber2");
        else
            System.out.println("studentNumber != studentNumber2");
        for (int i = 0; i < studentNumber2.length; i++)
            System.out.println(studentNumber2[i]);

        System.out.println("====Step2====");
        // 这是一次首地址拷贝
```

```
        studentNumber = studentNumber2;

        for (int i = 0; i < studentNumber.length; i++)
            studentNumber[i] = studentNumber2[i];

        for (int i = 0; i < studentNumber2.length; i++) {
            System.out.println("studentNumber is:");
            System.out.println(studentNumber[i]);
            System.out.println("studentNumber2 is:");
            System.out.println(studentNumber2[i]);
        }

        System.out.println("====Step3====");
        ArrayTest arrayTest = new ArrayTest();
        arrayTest.OutputStudentList();

        // 初始化一个 StudentList
        Student[] studlist;
        studlist = arrayTest.SetOneStudentList(5);
        ArrayTest.OutputStudentList(studlist);
        ArrayTest.ChangeStudentList(studlist);

    }
}
```

2.2 “TwoDimensionalArray”

```
public class TwoDimensionalArray {

    /**
     * @param args
     */
    public static void showTable(int[][] table, int rowlength, int columnlength) {
        for (int row = 0; row < rowlength; row++) {
            System.out.print(row + 1 + ":\t");
            for (int column = 0; column < columnlength; column++)
                System.out.print(table[row][column] + "\t");
            System.out.print("\n");
        }
    }

    public static void showTable(int[][] table) {
        for (int row = 0; row < table.length; row++) {
            System.out.print(row + 1 + ":\t");
            for (int column = 0; column < table[row].length; column++)
                System.out.print(table[row][column] + "\t");
            System.out.print("\n");
        }
    }

    public static int[][] getBigTable(int rowlength, int columnlength) {
        int[][] temp = new int[rowlength][columnlength];
        for (int row = 0; row < rowlength; row++)
            for (int column = 0; column < columnlength; column++)
                temp[row][column] = row * 100 + column;
```

```
        return temp;
    }

    public static void main(String[] args) {
        int[][] table = new int[10][6];

        for (int row = 0; row < 10; row++)
            for (int column = 0; column < 6; column++)
                table[row][column] = row * 10 + column;

        System.out.println("显示 table");
        TwoDimensionalArray.showTable(table, 10, 6);

        int[][] table2 = TwoDimensionalArray.getBigTable(11, 7);
        System.out.println("显示 table2");
        TwoDimensionalArray.showTable(table2);

    }

}
```

2.3 “ExceptionDemo”

```
public class ExceptionDemo {
    public static int div(int number, int div) throws DivideByZeroException
    // throws，有可能抛出异常，而在方法内没有被处理
    // 使用这样的方法，应该放在 try{}里面
    {
        if (div == 0)
            throw new DivideByZeroException("除数不能为 0");
        // 抛出异常后，将立刻中止此方法的执行
        else
            return number / div;
    }

    public static int sqrt(int number) throws SqrtExceedException
    // throws，有可能抛出异常，而在方法内没有被处理
    // 使用这样的方法，应该放在 try{}里面
    {
        if (number > 46340)
            throw new SqrtExceedException("求平方数越界");
        // 抛出异常后，将立刻中止此方法的执行
        else
            return number * number;
    }

    public static int div() {
        return (10 / 0);
    }

    public static int div(int div) throws DivideByZeroRuntimeException
    // throws，有可能抛出异常，而在方法内没有被处理
    // 使用这样的方法，应该放在 try{}里面
```

```
{
        if (div == 0)
            throw new DivideByZeroRuntimeException("除数不能为 0");
        // 抛出异常后，将立刻中止此方法的执行
        else
            return 10 / div;
    }
    public static void main(String[] args) {

        int div = 0;
        int number = 10;
        int result = 0;
        Scanner scan = new Scanner(System.in);
        System.out.println("请输入一个整数");
        div = scan.nextInt();
        // result=number/div; //有危险的写法
        // 用 if-else 规避风险
        if (div == 0) {
            System.out.println("除数不能为 0");
            result = number / (div + 1);
            System.out.println("result=" + result);

        } else {
            result = number / div;
            System.out.println("result=" + result);

        }

        // 用 Exception 类，不过一般不要直接用 Exception 类
        try {
            result = 10;
            System.out.println("请再输入一个整数");
            div = scan.nextInt();
            if (div == 0) {
                throw new Exception("除数不能为 0");
                // 将不能再添加语句
                // result=number/(div+1);
                // System.out.println("result="+result);
            } else {
                result = number / div;
                System.out.println("result=" + result);
            }
        } catch (Exception e) {

            System.out.println(e.getMessage());
            System.out.println("result=" + result);
        }

        // 用自定义的 DivideByZeroException 异常，较好的用法
        try {
            result = 10;
            System.out.println("2:请再输入一个整数");
            div = scan.nextInt();
            if (div == 0)
                throw new DivideByZeroException();
            else {
```

```
                    result = number / div;
                    System.out.println("result=" + result);
                }
            } catch (DivideByZeroException e) {
                System.out.println(e.getMessage());
                System.out.println("result=" + result);
            }

            // 使用还有未处理异常可能的 div()函数
            try {
                ExceptionDemo.div(10, 0);
                ExceptionDemo.sqrt(100);
            } catch (DivideByZeroException e) {
                System.out.println(e.getMessage());
            } catch (SqrtExceedException e) {
                System.out.println(e.getMessage());
            } catch (Exception e) {
                System.out.println(e.getMessage());
            }

            //对于 unchecked exception, 可以直接使用，异常最终被抛出到控制台
            ExceptionDemo.div();
            ExceptionDemo.div(10);
            //ExceptionDemo.div(10, 0); //此句话将出错
        }

}
```

3. 写程序

3.1 字符串类

基于 char[]，写一个字符串类，能够实现字符串的大部分功能，而且能够抛出自定义的“容量超出异常”（NegativeArraySizeException 和 ArrayIndexOutOfBoundsException 等）。

3.2 三维整数数组类

基于一维数组 int[]，设计并实现一个三维数组类，能够实现 int[][][]的大部分功能，并且能够抛出大部分的异常。

4. 思考题

第 七 章. 查找与排序算法

1. 实验目的

（1） 熟悉查找和排序的算法。

（2） 熟悉类的接口的概念。

2. 读程序

2.1 SortAlgorithm

```
public class SortAlgorithm {
    // -----------------------------------------------------------------
    // Sorts the specified array of objects using the selection
    // sort algorithm.
    // -----------------------------------------------------------------
    public static void selectionSort(int[] list) {
        int min;
        int temp;

        for (int index = 0; index < list.length - 1; index++) {
            min = index;
            for (int scan = index + 1; scan < list.length; scan++)
                if (list[scan] < list[min])
                    min = scan;

            // Swap the values
            temp = list[min];
            list[min] = list[index];
            list[index] = temp;
        }
    }

    // -----------------------------------------------------------------
    // Sorts the specified array of objects using the insertion
    // sort algorithm.
    // -----------------------------------------------------------------
    public static void insertionSort(int[] list) {
        for (int index = 1; index < list.length; index++) {
            int key = list[index];
            int position = index;

            // Shift larger values to the right
            while (position > 0 && key < list[position - 1]) {
                list[position] = list[position - 1];
                position--;
            }
```

```
                list[position] = key;
            }
        }
}
```

2.2 SearchAlgorithm

```
public class SearchAlgorithm {
    /* 线性查找，从第一个元素开始查找 */
    public static int linearSearch(int[] list, int target) {
        int index = 0;
        boolean found = false;
        while (!found && index < list.length) {
            if (list[index] == target)
                found = true;
            else
                index++;
        }

        if (found)
            return list[index];
        else
            return -1;
    }

    /* 二分查找，非递归实现的算法 */
    public static int binarySearch(int[] list, int target) {
        int min = 0, max = list.length, mid = 0;
        boolean found = false;

        while (!found && min <= max) {
            mid = (min + max) / 2;
            if (list[mid] == target)
                found = true;
            else if (target < list[mid])
                max = mid - 1;
            else
                min = mid + 1;
        }

        if (found)
            return list[mid];
        else
            return -1;
    }
    public static void main(String[] args) {
        int[] a = {0,1,2,3,4,5,6,7,8,9};
        System.out.println("Found:"+SearchAlgorithm.binarySearch(a,3));
    }
}
```

3. 写程序

3.1 三分查找

结合“二分查找”的算法，设计并实现一个“三分查找”的程序。

3.2 归并排序

设计并实现一个“归并（Merge）排序”的程序。

4. 思考题

（1） 以上的排序算法中，对于普通的无序整数序列，哪一个效率最高。

（2） 对于读取磁盘上的文件，在文件中进行随机查找的算法，哪一个效率最高。

第 八 章. 递归程序

1. 实验目的

（1） 能够读懂基本的递归程序。

（2） 能够书写简单的递归程序。

2. 读程序

2.1 NumberZeros

```
/**
 * 计算数字里面零的个数，说明递归的使用方法
 * @author Dahogn
 * @version 1.01
 * @since 2008.11.25
 */
public class NumberZeros
{
	public static void main(String[] args)
	{
		System.out.println("Enter a nonnegative number:");
		Scanner scan=new Scanner(System.in);
		int number = scan.nextInt();
		System.out.println(number + " contains "
					+ numberOfZeros(number) + " zeros.");
	}

	/**
	  Precondition: n >= 0
	  Returns the number of zero digits in n.
	*/

	public static int numberOfZeros(int n)
	{
		int temp=0;
		if (n == 0)
			return 1;
		else if (n < 10)//and not 0
			return 0;//0 for no zeros
		else if (n%10 == 0)
			return(numberOfZeros(n/10) + 1);
		else //n%10 != 0
			return(numberOfZeros(n/10));
	}
}
```

2.2 TowersOfHanoi

```
public class TowersOfHanoi {
	private int totalDisks;

	// -----------------------------------------------------------------
	// Sets up the puzzle with the specified number of disks.
	// -----------------------------------------------------------------
	public TowersOfHanoi(int disks) {
		totalDisks = disks;
	}

	// -----------------------------------------------------------------
	// Performs the initial call to moveTower to solve the puzzle.
	// Moves the disks from tower 1 to tower 3 using tower 2.
	// -----------------------------------------------------------------
	public void solve() {
		moveTower(totalDisks, 1, 3, 2);
	}

	// -----------------------------------------------------------------
	// Moves the specified number of disks from one tower to another
	// by moving a subtower of n-1 disks out of the way, moving one
	// disk, then moving the subtower back. Base case of 1 disk.
	// -----------------------------------------------------------------
	private void moveTower(int numDisks, int start, int end, int temp) {
		if (numDisks == 1)
			moveOneDisk(start, end);
		else {
			moveTower(numDisks - 1, start, temp, end);
			moveOneDisk(start, end);
			moveTower(numDisks - 1, temp, end, start);
		}
	}

	// -----------------------------------------------------------------
	// Prints instructions to move one disk from the specified start
	// tower to the specified end tower.
	// -----------------------------------------------------------------
	private void moveOneDisk(int start, int end) {
		System.out.println("Move one disk from " + start + " to " + end);
	}

	public static void main(String[] args) {
		TowersOfHanoi towers = new TowersOfHanoi(4);

		towers.solve();
	}
}
```

3. 写程序

3.1 斐波纳契数列（Fibonacci 数列）

波纳契数列（Fibonacci Sequence），又称黄金分割数列，指的是这样一个数列：1、1、2、3、5、8、13、21、……在数学上，斐波纳契数列以如下被以递归的方法定义：$F_0=0$，$F_1=1$，$F_n=F_{(n-1)}+F_{(n-2)}$（n>=2，n∈N*）。

请用递归程序编程实现此算法。

3.2 全排列

从 n 个不同元素中任取 m（m≤n）个元素，按照一定的顺序排列起来，叫做从 n 个不同元素中取出 m 个元素的一个排列。当 m＝n 时所有的排列情况叫全排列。如(1,2,3)三个元素的全排列为：(1,2,3), (1,3,2), (2,1,3), (2,3,1), (3,2,1), (3,1,2)。

请用递归程序编程实现此算法。

4. 思考题

（1） 请不用递归程序，用循环解决以上两个问题。

（2） 如果汉诺塔的盘子数量很多，比如大于 1000 个，那么有没有更高效的算法能够解决此问题。

第 九 章. 链表的实现与基本算法

1. 实验目的

（1） 熟悉链表的原理和基本算法。

（2） 加深对引用的理解和使用。

2. 读程序

2.1 ListNode

```
public class ListNode
{
	private String data;
	private ListNode link;

	public ListNode( )
	{
		link = null;
		data = null;
	}
	public ListNode(String newData, ListNode linkValue)
	{
		data = newData;
		link = linkValue;
	}
	public void setData(String newData)
	{
		data = newData;
	}
	public String getData( )
	{
		return data;
	}
	public void setLink(ListNode newLink)
	{
		link = newLink;
	}
	public ListNode getLink( )
	{
		return link;
	}
}
```

2.2 StringLinkedList

```
public class StringLinkedList {
```

```
private ListNode head;

public StringLinkedList() {
	head = null;
}

/**
  * Returns the number of nodes in the list.
  */
public int length() {
	int count = 0;
	ListNode position = head;
	while (position != null) {
		count++;
		position = position.getLink();
	}
	return count;
}

/**
  * Adds a node at the start of the list. The added node has addData as its
  * data. The added node will be the first node in the list.
  */
public void addANodeToStart(String addData) {
	head = new ListNode(addData, head);
}

public void deleteHeadNode() {
	if (head != null) {
		head = head.getLink();
	} else {
		System.out.println("Deleting from an empty list.");
		System.exit(0);
	}
}

public boolean onList(String target) {
	return (Find(target) != null);
}

/**
  * Finds the first node containing the target data, and returns a reference
  * to that node. If target is not in the list, null is returned
  */
private ListNode Find(String target) {
	ListNode position = head;
	String dataAtPosition;
	while (position != null) {
		dataAtPosition = position.getData();
		if (dataAtPosition.equals(target))
			return position;
		position = position.getLink();
	}
	// target was not found,
	return null;
}
```

```
    public void showList() {
        ListNode position = head;
        while (position != null) {
            System.out.println(position.getData());
            position = position.getLink();
        }
    }

    // add a node at the tail of the list
    public void AddNodeAtTail(String newData) {
        if (head == null) {
            addANodeToStart(newData);
        } else {
            ListNode position = head;
            ListNode current = null;

            while (position != null) {
                current = position;
                position = position.getLink();
            }
            current.setLink(new ListNode(newData, null));
        }
    }

    // delete a node at the tail of the list
    public void DeleteNodeAtTail() {
        if (head != null) {
            ListNode position = head;
            ListNode current = position;
            if (position.getLink() == null) {
                head = null;
            } else {
                while (position.getLink() != null) {
                    current = position;
                    position = position.getLink();
                }
                current.setLink(null);
            }
        }
    }

    // delete a node at the tail of the list
    // if no target is found, no data add
    public void FindAndInsertAfter(String target, String newData) {
        ListNode current = Find(target);
        if (current != null)
            current.setLink(new ListNode(newData, current.getLink()));
        else
            System.out.println("No target is found, nothing insert.");
    }
}
```

2.3 LinkedListDemo

```
public class LinkedListDemo
{
```

```
    public static void main(String[] args)
    {
        StringLinkedList list = new StringLinkedList( );
        list.addANodeToStart("One");
        list.addANodeToStart("Two");
        list.addANodeToStart("Three");
        System.out.println("List has " + list.length( )
                                + " entries.");
        list.showList( );

        if (list.onList("Three"))
            System.out.println("Three is on list.");
        else
            System.out.println("Three is NOT on list.");

        list.deleteHeadNode( );

        if (list.onList("Three"))
            System.out.println("Three is on list.");
        else
            System.out.println("Three is NOT on list.");
        list.FindAndInsertAfter("Two", "Five");
        list.deleteHeadNode( );
        list.deleteHeadNode( );
        System.out.println("Start of list:");
        list.showList( );
        System.out.println("End of list.");
    }
}
```

3. 写程序

以下所有程序都参照 2.1 所述的 Node 结构和 2.2 所述的程序框架。

3.1 找出值相等的点

请书写一个程序，能够找出所有 ListNode 中 data 值相等的点，列出查找的结果。

3.2 按序重排列

请书写一个程序，在不构造新链表的情况下，按照 data 的数据大小（字符串类型为字典顺序），对链表进行按序重新排列。

3.3 组成双向链表

请书写一个程序，加入一个引用指向前趋结点，构成双向链表，并重写插入、

删除、排序等主要的链表操作方法。

4. 思考题

（1） 如果最后一个节点的指向不是为 NULL，而是头（Head）结点，组成循环链表，那么链表将会带来哪些便利，所有的操作需要注意哪些方面。

（2） 能否不用引用，实现链表。