2、具体内容

下面通过几个简短的程序要求对于继承的概念进行巩固，对于程序开发而言，简单Java类是其基础的组成，也是所有概念最好的融合。

■案例分析一：

1、建立一个人类（Person）和学生类（Student），功能要求如下：

（1）Person中包含4个私有型的数据成员name、addr、sex、age，分别为字符串型、字符串型、字符型及整型，表示姓名、地址、性别和年龄。一个4参构造方法、一个两参构造方法、一个无参构造方法、一个输出方法显示4种属性。

（2）Student类继承Person类，并增加成员math、english存放数学和英语成绩。一个6参构造方法、一个两参构造方法、一个无参构造方法和重写输出方法用于显示6种属性。

```
class Person {
	private String name ;
	private String addr ;
	private char sex ;
	private int age ;
	public Person() {}
	public Person(String name,String addr) {
		this(name,addr,'男',0) ;
	}
	public Person(String name,String addr,char sex,int age) {
		this.name = name ;
		this.addr = addr ;
		this.sex = sex ;
		this.age = age ;
	}
	public String getInfo() {
		return "姓名：" + this.name + "、地址：" + this.addr + "、性别：" + this.sex + "、
年龄：" + this.age ;
	}
```

```
}
class Student extends Person {
	private double math ;
	private double english ;
	public Student() {}
	public Student(String name,String addr) {
		super(name,addr) ;
	}
	public Student(String name,String addr,char sex,int age,double math,double english) {
		super(name,addr,sex,age) ;
		this.math = math ;
		this.english = english ;
	}
	public String getInfo() {
		return super.getInfo() + "、数学成绩：" + this.math + "、英语成绩：" + this.english
;
	}
}
public class JavaDemo {
	public static void main(String args[]) {
		Student stu = new Student("张三","天安门",'男',12,78.99,89.98) ;
		System.out.println(stu.getInfo()) ;
	}
}
```

```
class Person
{
	private String name;
	private String addr;
	private char sex;
	private int age;
	public Person() {
	}
	public Person(String name, String addr) {
		this(name,addr,'男',0);
	}
	public Person(String name, String addr, char sex, int age) {
		this.name = name;
		this.addr = addr;
		this.sex = sex;
		this.age = age;
	}
	public String getName() {
		return name;
	}
	public void setName(String name) {
		this.name = name;
	}
	public String getAddr() {
		return addr;
	}
	public void setAddr(String addr) {
```

```
		this.addr = addr;
	}
	public char getSex() {
		return sex;
	}
	public void setSex(char sex) {
		this.sex = sex;
	}
	public int getAge() {
		return age;
	}
	public void setAge(int age) {
		this.age = age;
	}

	public String getInfo() {
		return "Person [name=" + name + ", addr=" + addr + ", sex=" + sex + ", age=" +
age + "]";
	}
}

class Student extends Person{
	private double math;
	private double english;
	public Student() {

	}

	public double getMath() {
		return math;
	}

	public void setMath(double math) {
		this.math = math;
	}

	public double getEnglish() {
		return english;
	}

	public void setEnglish(double english) {
		this.english = english;
	}

	public Student(String name, String addr, double math, double english) {
		super(name, addr);
		this.math = math;
		this.english = english;
	}

	public Student(String name, String addr, char sex, int age, double math, double
english) {
```

```
        super(name, addr, sex, age);
        this.math = math;
        this.english = english;
    }
    public String getInfo() {
        return super.getInfo()+"Student [math=" + math + ", english=" + english + "]";
    }

}


public class JavaDemo {
    public static void main(String args[]) {
        Student stu = new Student("sanzhang","shanghai",'男',18,79.8,98.78);
        System.out.println(stu.getInfo());
    }
}
```

在这样的操作案例里面发现子类对象实例化、构造方法调用、方法覆写。

■案例分析二

2、定义员工类，具有姓名、年龄、性别属性，并具有构造方法和显示数据方法。定义管理层类，继承员工类，并有自己的属性职务和年薪。定义职员类，继承员工类，并有自己的属性所属部门和月薪。

```
class Employee {
    private String name ;
    private int age ;
    private String sex ;
    public Employee() {}
    public Employee(String name,int age,String sex) {
        this.name = name ;
        this.age = age ;
        this.sex = sex ;
    }
    public String getInfo() {
        return "姓名：" + this.name + "、年龄：" + this.age + "、性别：" + this.sex ;
    }
}
class Manager extends Employee {        // 管理层
    private String job ;
    private double income ;
    public Manager() {}
    public Manager(String name,int age,String sex,String job,double income) {
        super(name,age,sex) ;
        this.job = job ;
        this.income = income ;
    }
    public String getInfo() {
```

```
            return "【管理层】" + super.getInfo() + "、职务：" + this.job + "、年薪：" +
this.income ;
        }
}
class Staff extends Employee {
        private String dept ;
        private double salary ;
        public Staff() {}
        public Staff(String name,int age,String sex,String dept,double salary) {
                super(name,age,sex) ;
                this.dept = dept ;
                this.salary = salary ;
        }
        public String getInfo() {
                return "【职员】" + super.getInfo() + "、部门：" + this.dept + "、月薪：" +
this.salary ;
        }
}
public class JavaDemo {
        public static void main(String args[]) {
                Manager man = new Manager("张三",38,"女","主管",150000.00) ;
                Staff sta = new Staff("李四",18,"男","出纳",3000.00) ;
                System.out.println(man.getInfo()) ;
                System.out.println(sta.getInfo()) ;
        }
}
```

■案例分析三

编写程序，统计出字符串“want you to know one thing”中字母n和字母o的出现次数。

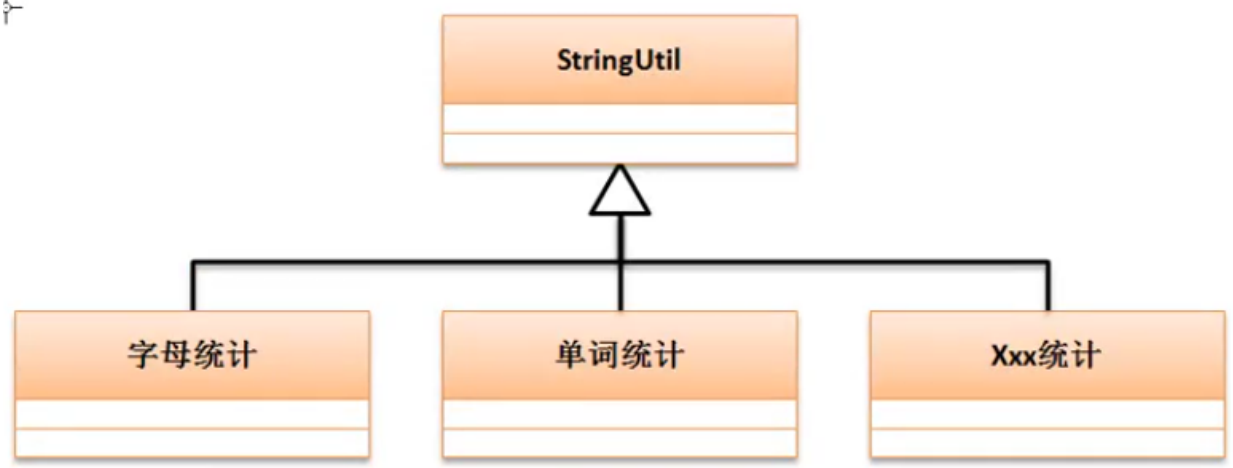
对于本程序而言，最简单操作方式就是直接在主方法里面定义一个操作，或者直接定义一个新的类进行处理。

范例：定义一个单独的处理类

```
class StringUtil {
        // 返回的第一个内容为字母n的个数，第二个内容为字母o的个火速
        public static int [] count(String str) {
                int countData [] = new int [2] ;
                char [] data = str.toCharArray() ;    // 将字符串变为字符数组
                for (int x = 0 ; x < data.length ; x ++) {
                        if (data[x] == 'n' || data[x] == 'N') {
                                countData[0] ++ ;
                        }
                        if (data[x] == 'o' || data[x] == 'O') {
                                countData[1] ++ ;
                        }
```

```
            }
            return countData ;
        }
}
public class JavaDemo {
        public static void main(String args[]) {
            String str = "want you to know one thing" ;
            int result [] = StringUtil.count(str) ;
            System.out.println("字母n的个数：" + result[0]) ;
            System.out.println("字母o的个数：" + result[1]) ;
        }
}
```

以上的解决方案严格来讲只是一种顺序式的思维模式解决的，假设说下现在统计的是字母o或n的个数，那么还有可能进行其它统计的设计。



改良后的程序

```
class StringUtil {
        private String content ;  // 需要保存字符串
        public StringUtil(String content) {
            this.content = content ;
        }
        public String getContent() {
            return this.content ;
        }
        public String getInfo() {  // 默认的信息返回
            return this.getContent() ;
        }
}
class StringCount extends StringUtil {
        private int nCount ;
        private int oCount ;
        public StringCount(String content) {
            super(content) ;
            this.countChar() ;    // 构造方法统计
        }
        public void countChar() {
            char [] data = super.getContent().toCharArray() ;  // 将字符串变为字符数组
            for (int x = 0 ; x < data.length ; x ++) {
```

```
                if (data[x] == 'n' || data[x] == 'N') {
                    this.nCount ++ ;
                }
                if (data[x] == 'o' || data[x] == 'O') {
                    this.oCount ++ ;
                }
            }
        }
        public int getNCount() {
            return this.nCount ;
        }
        public int getOCount() {
            return this.oCount ;
        }
        public String getInfo() {
            return "字母n的个数：" + this.nCount + "、字母o的个数：" + this.oCount ;
        }
}
public class JavaDemo {
        public static void main(String args[]) {
            StringCount sc = new StringCount("want you to know one thing") ;
            System.out.println(sc.getInfo()) ;
        }
}
```

任何方案都可以，如果采用第一种方案比较直观，但是第二种方案更加适合于结构化的设计。

■案例分析四

4、 建立一个可以实现整型数组的操作类（Array），而后在里面可以操作的数组的大小由外部来决定，而后在Array类里面需要提供有数组的如下处理：进行数据的增加（如果数据满了则无法增加）、可以实现数组的容量扩充、取得数组全部内容。

完成之后在此基础上再派生出两个子类：

- 数组排序类：返回的数据必须是排序后的结果；
- 数组反转类：可以实现内容的首尾交换。

对于本程序而言首先要考虑的一定都是父类如何定义完善。

第一步：实现基本的数组操作类定义

```
class Array {    // 数组的操作类
        private int [] data ; // 整型数组
        private int foot ; // 进行数组索引控制
        public Array(int len) {
            if (len > 0) {
                this.data = new int [len] ;        // 开辟数组
            } else {
                this.data = new int [1] ; // 开辟一个空间
```

```
            }
        }
        // 实现数组的容量扩充，给出的是扩充大小，实际大小：已有大小 + 扩充大小
        public void increment(int num) {
            int newData [] = new int [this.data.length + num] ;
            System.arraycopy(this.data,0,newData,0,this.data.length) ;
            this.data = newData ; // 改变数组引用
        }
        public boolean add(int num) {       // 数据增加
            if (this.foot < this.data.length) {     // 有位置
                this.data[this.foot ++] = num ;
                return true ;
            }
            return false ;
        }
        public int[] getData() {
            return this.data ;
        }
}
public class JavaDemo {
        public static void main(String args[]) {
            Array arr = new Array(5) ;
            System.out.println(arr.add(10)) ;
            System.out.println(arr.add(5)) ;
            System.out.println(arr.add(20)) ;
            System.out.println(arr.add(3)) ;
            System.out.println(arr.add(6)) ;
            arr.increment(3) ;
            System.out.println(arr.add(1)) ;
            System.out.println(arr.add(7)) ;
            System.out.println(arr.add(0)) ;
        }
}
```

第二步：进行排序子类定义

```
class Array {    // 数组的操作类
        private int [] data ; // 整型数组
        private int foot ; // 进行数组索引控制
        public Array(int len) {
            if (len > 0) {
                this.data = new int [len] ;       // 开辟数组
            } else {
                this.data = new int [1] ; // 开辟一个空间
            }
        }
        // 实现数组的容量扩充，给出的是扩充大小，实际大小：已有大小 + 扩充大小
        public void increment(int num) {
            int newData [] = new int [this.data.length + num] ;
            System.arraycopy(this.data,0,newData,0,this.data.length) ;
            this.data = newData ; // 改变数组引用
        }
        public boolean add(int num) {       // 数据增加
```

```
            if (this.foot < this.data.length) {     // 有位置
                this.data[this.foot ++] = num ;
                return true ;
            }
            return false ;
        }
        public int[] getData() {
            return this.data ;
        }
}
class SortArray extends Array {       // 定义排序子类
        public SortArray(int len) {
            super(len) ;
        }
        public int[] getData() {    // 获得排序结果
            java.util.Arrays.sort(super.getData()) ;    // 排序
            return super.getData() ;
        }
}
public class JavaDemo {
        public static void main(String args[]) {
            SortArray arr = new SortArray(5) ;
            System.out.println(arr.add(10)) ;
            System.out.println(arr.add(5)) ;
            System.out.println(arr.add(20)) ;
            System.out.println(arr.add(3)) ;
            System.out.println(arr.add(6)) ;
            arr.increment(3) ;
            System.out.println(arr.add(1)) ;
            System.out.println(arr.add(7)) ;
            System.out.println(arr.add(0)) ;
            int result [] = arr.getData() ;
            for (int temp : result) {
                System.out.print(temp + "、") ;
            }
        }
}
```

第三步：定义反转子类

```
class Array {    // 数组的操作类
        private int [] data ; // 整型数组
        private int foot ; // 进行数组索引控制
        public Array(int len) {
            if (len > 0) {
                this.data = new int [len] ;       // 开辟数组
            } else {
                this.data = new int [1] ; // 开辟一个空间
            }
        }
        // 实现数组的容量扩充，给出的是扩充大小，实际大小：已有大小 + 扩充大小
        public void increment(int num) {
            int newData [] = new int [this.data.length + num] ;
```

```
            System.arraycopy(this.data,0,newData,0,this.data.length) ;
            this.data = newData ; // 改变数组引用
        }
        public boolean add(int num) {       // 数据增加
            if (this.foot < this.data.length) {     // 有位置
                this.data[this.foot ++] = num ;
                return true ;
            }
            return false ;
        }
        public int[] getData() {
            return this.data ;
        }
}
class SortArray extends Array {      // 定义排序子类
        public SortArray(int len) {
            super(len) ;
        }
        public int[] getData() {    // 获得排序结果
            java.util.Arrays.sort(super.getData()) ;    // 排序
            return super.getData() ;
        }
}
class ReverseArray extends Array { // 定义反转子类
        public ReverseArray(int len) {
            super(len) ;
        }
        public int[] getData() {    // 获得排序结果
            int center = super.getData().length / 2 ;
            int head = 0 ;
            int tail = super.getData().length - 1 ;
            for (int x = 0 ; x < center ; x ++) {
                int temp = super.getData()[head] ;
                super.getData()[head] = super.getData()[tail] ;
                super.getData()[tail] = temp ;
                head ++ ;
                tail -- ;
            }
            return super.getData() ;
        }
}
public class JavaDemo {
        public static void main(String args[]) {
            ReverseArray arr = new ReverseArray(5) ;
            System.out.println(arr.add(10)) ;
            System.out.println(arr.add(5)) ;
            System.out.println(arr.add(20)) ;
            System.out.println(arr.add(3)) ;
            System.out.println(arr.add(6)) ;
            arr.increment(3) ;
            System.out.println(arr.add(1)) ;
            System.out.println(arr.add(7)) ;
```

```
            System.out.println(arr.add(0)) ;
            int result [] = arr.getData() ;
            for (int temp : result) {
                System.out.print(temp + "、") ;
            }
        }
}
```

父类之中定义的方法名称往往都很重要，如果功能相同的时候子类应该以覆写父类的方法为优先考虑。