# 面向对象程序设计
# 作业报告

第二次


姓名

班级

学号

电话

Email

日期

# 目录

# 题目 1：创建一个表示复数的类

创建一个用表示复数的 Complex 类类型，当创建完这个类以后，就可以直接使用这个数据类型完成需要复数参与的各种算法中，图 1 是对创建的 Complex 类型的具体说明。

该题需要完成的任务如下：

任务 1：编写 Complex 类类型。

任务 2：编写一个测试程序（TestComplex），要求对 Complex 类型中的每一个公共实例方法进行测试。

任务 3：编写一个使用 Complex 的简易客户端程序（ComplexApp），该程序完成提示用户输入两个复数，之后按照如下输出样式完成对这两个复数完成操作的相关调用。

```
Enter complex number 1 (real and imaginary part): 1.1 2.2
Enter complex number 2 (real and imaginary part): 3.3 4.4
Number 1 is: (1.1 + 2.2i)
(1.1 + 2.2i) is NOT a pure real number
(1.1 + 2.2i) is NOT a pure imaginary number
Number 2 is: (3.3 + 4.4i)
(3.3 + 4.4i) is NOT a pure real number
(3.3 + 4.4i) is NOT a pure imaginary number
(1.1 + 2.2i) is NOT equal to (3.3 + 4.4i)
(1.1 + 2.2i) + (3.3 + 4.4i) = (4.4 + 6.6000000000000005i)
```

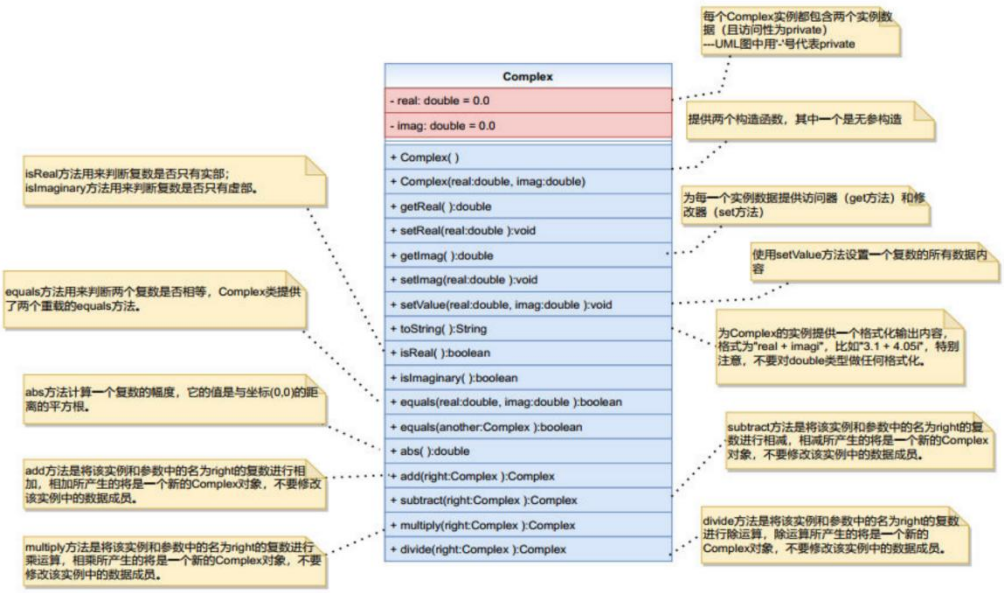



图 1

备注：

1. Complex 类型中的实部和虚部都是用 double 类型表示，对 double 类型数据的处理要注意如下几方面：
   ① double 中有一种特别的数据 Double.NaN，在实现中要对这种数据进行判断。
   ② 在实现 equals 方法时，不能简单使用==运算符进行判断，否则就会出现问题。常见的解决方案是设置一个叫做 EPSILON 的阈值，该值通常设置为 $10^{-8}$，如果两个浮点类型之间相差值小于等于 EPSILON，则认为两个值相等。
2. Complex 类型中的某些成员函数（比如 add、substract 等）的返回类型定义为 Complex，这样做的好处是可以将函数串起来调用，比如有三个 Complex 类型的对象 c1，c2 和 c3，想要完成 c1+c2+c3 这样的运算，对应的 Java 中的调用形式可以是：c1.add(c2).add(c3)。

# 主程序设计：

## 一．编译 Complex 类

1. 编写两个成员变量 real 和 imag，相当于是复数的实数部分和虚数部分；

2. 编写访问器 getReal（）、getImag（）和修改器 setReal（）、setImag（）；

3. 编写 Complex 类的无参构造器和有参构造器；

4. 编写 toString（）方法，订制输出内容；

5. 编写 isReal（）方法，通过调用 imag（虚数部分的值）是否为零以判断函数是否为纯实数；

6. 编写 isImag（）方法，通过调用 real、imag，若 real（实数部分的值）为零且 imag（虚数部分的值）不为零，则判定为纯虚数；

7. 编写 equals（double real，double imag）方法，编写 EPSILON=0.00000001 阈值，若调用 equals 方法的复数对象

的实数部分及虚数部分与 real、imag 的值相差不到 EPSILON，则判断这两个复数对象相等；

8. 编写 abs（）方法，以求解调用此方法的复数对象其与坐标原点位置距离的平方根；

9. 分别编写 add（Complex right）、subtrack（Complex right）、multiply（Complex right）、divide（Complex right）方法，以计算调用该方法的复数对象与方法内引入的复数对象参数的加法运算、减法运算、乘法运算、除法运算（特别声明：除法运算中若除数为 0，则计算的值为 NaN，因此调用 if 循环判断除数是否为 0，若为 0 则提醒用户不能用 0 做除数，否则正常运行 divide（）方法）。

二. 编写 ComplexApp（）类

1. 调用 Scanner（）类，并输出内容提醒用户按照要求分别输入两个复数的实数部分和虚数部分；

2. 创建两个复数对象，并将用户输入的值代入；

3. 调用 toString（）方法，按照（X，Xi）的格式输出当前复数对象；

4. 调用（一）中所写的 isReal（）方法和 isImag（）方法，判断用户给定的两个复数是否为纯实数或者纯虚数，并将判断结果输出；

5. 调用（一）中所写的 equals（）方法，判断用户输入的两个复数是否相等，并将判断结果输出；

6. 使用用户给的其中一个复数对象运用 add（Complex right）方法，参数是用户给的另一个复数对象，以此计算两者相加的结果。

三．编写 TestComplex（）类（仅做测试使用）

1.编写四个复数对象，其值依次为（1.1,+2.2i）、（1.1+0i）、（0+2.2i）、（0+0i），分别模拟复数、纯实数、纯虚数、0以检验代码能否正常运行；

2.依次对四个复数对象调用 getReal（）和 getImag（）方法调出对象的实数部分的值及虚数部分的值并检验输出的值是否与输入的值相等；

3.依次对四个复数对象调用 isReal（）和 isImag（）方法判断其是否为纯实数、纯虚数，并于真实情况相比较以验证输出结果是否正确；

4.调用 getReal（）和 getImag（）方法检验编译器是否出错，后通过 setReal（）和 setImag（）方法修改第一个复数对象的值，再用 getReal（）和 getImag（）方法调出修改后的值，以此检验修改器是否正常运行；

5.调用 equals（Complex right）方法判断已写的几个复数对象中任意两个是否相等（都不相等），再创建一个复数对象，要求其与已有的复数对象中的某一个完全相等，后用 equals（）方法判断这两个复数对象是否相等，根据输出结果检验该方法的正确性；

6.调用 abs（）方法分别计算已有的四个复数对象与坐标原点距离的平方根，经计算器验证后得以验证该方法的正确性；

7.调用 add（Complex right）、subtract（Complex right）、multiply（Complex right）、divide（Complex right）方法输出两个复数对象的和、差、积、除运算，并可使用计算器来验证答案的正确性（通过验证，本人书写的方法输出结果与真实结果相等）；

8.使用任意一个复数对象调用 divide（0），若输出“you can not use 0.0 as a divisor！”则判定为正确；若仍然输出如（7）除运算的结构并输出 NaN，则判定为失败；

代码测试：

一． ComplexApp 类代码运行结果

```
请按照提示输入两个复数:
Enter complex number 1 (real and imaginary part):1.1 2.2
Enter complex number 2 (real and imaginary part):3.3 4.4
Number 1 is: (1.1 + 2.2i)
(1.1 + 2.2i) is a Not pure real number
(1.1 + 2.2i) is a Not pure imaginary number
Number 2 is: (3.3 + 4.4i)
(3.3 + 4.4i) is a Not pure real number
(3.3 + 4.4i) is a Not pure imaginary number
(1.1 + 2.2i) is Not equals to (3.3 + 4.4i)
(1.1 + 2.2i) + (3.3 + 4.4i) = (4.4 + 6.60000000000000005i)

Process finished with exit code 0
```

二． TestComplex 类代码运行结果

```
输出第一个对象:
Number 1 is: (1.1 + 2.2i)
判断第一个对象是否只有实数部分:
(1.1 + 2.2i) is a Not pure real number
判断第一个对象是否只有虚数部分:
(1.1 + 2.2i) is a Not pure imaginary number
输出第二个对象:
Number 2 is: (1.1 + 0.0i)
判断第二个对象是否只有实数部分:
(1.1 + 0.0i) is a pure real number
判断第二个对象是否只有虚数部分:
(1.1 + 0.0i) is a Not pure imaginary number
输出第三个对象:
Number 3 is: (0.0 + 2.2i)
判断第三个对象是否只有实数部分:
(0.0 + 2.2i) is a Not pure real number
判断第三个对象是否只有虚数部分:
(0.0 + 2.2i) is a pure imaginary number
输出第四个对象:
Number 4 is: (0.0 + 0.0i)
判断第四个对象是否只有实数部分:
(0.0 + 0.0i) is a pure real number
判断第四个对象是否只有虚数部分:
(0.0 + 0.0i) is a Not pure imaginary number
```

```
检查修改器能否正常运行:
修改前:
Number 1 is:(1.1 + 2.2i)
修改后:
Number 1 is:(6.6 + 8.8i)

检查两个对象是否相等:
(6.6 + 8.8i) is Not equals to (1.1 + 0.0i)
(6.6 + 8.8i) is equals to (6.6 + 8.8i)

输出(6.6 + 8.8i)与坐标原点距离的平方根:
11.0
输出(1.1 + 0.0i)与坐标原点距离的平方根:
1.1
输出(0.0 + 2.2i)与坐标原点距离的平方根:
2.2
输出(0.0 + 0.0i)与坐标原点距离的平方根:
0.0
```

```
加法运算:
(6.6 + 8.8i) + (1.1 + 0.0i) = (7.699999999999999 + 8.8i)
(0.0 + 2.2i) + (0.0 + 0.0i) = (0.0 + 2.2i)
(0.0 + 2.2i) + (6.6 + 8.8i) = (6.6 + 11.0i)
减法运算:
(6.6 + 8.8i) - (1.1 + 0.0i) = (5.5 + 8.8i)
(1.1 + 0.0i) - (0.0 + 2.2i) = (1.1 + -2.2i)
(1.1 + -2.2i) - (6.6 + 8.8i) = (-5.5 + -11.0i)
乘法运算:
(6.6 + 8.8i) * (1.1 + 0.0i) = (7.26 + 9.6800000000000001i)
(6.6 + 8.8i) * (1.1 + 0.0i) = (7.26 + 9.6800000000000001i)
(7.26 + 9.6800000000000001i) * (0.0 + 2.2i) = (-21.296000000000006 + 15.972000000000001i)
除法运算:
(6.6 + 8.8i) / (1.1 + 0.0i) = (5.999999999999999 + 8.0i)
(6.6 + 8.8i) / (1.1 + 0.0i) = (5.999999999999999 + 8.0i)
(5.999999999999999 + 8.0i) / (0.0 + 2.2i) = (3.6363636363636362 + -2.7272727272727266i)

修改本应输出NaN的数据，提醒输出结果错误:
you can not use 0.0 as a divisor!

Process finished with exit code 0
```

总结:

1. 合理地构造有参构造器能让自己编写的类类型能够被更为灵活的调用;
2. 将会多次使用的代码重写为一种方法，能极大减少书写代码的行数即代码的重复率，达到提高代码效率的效果;
3. 出现合法但不合理的例外时（如将 0 作为除数）应编写合理的代码避免这种出错。

# 题目 2：使用 Complex 类型绘制分形图

1973 年，曼德布罗特（B.B.Mandelbrot）在法兰西学院讲课时，首次提出了分维和分形几何的设想。分形（Fractal）一词，是曼德布罗特创造出来的，其原意具有不规则、支离破碎等意义，分形几何学是一门以非规则几何形态为研究对象的几何学。由于不规则现象在自然界是普遍存在的，因此分形几何又称为描述大自然的几何学。分形几何建立以后，很快就引起了许多学科的关注，这是由于它不仅在理论上，而且在实用上都具有重要价值。

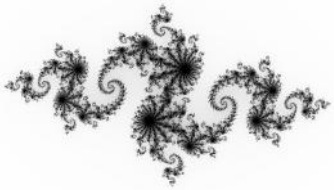
现在，使用刚刚创建的 Complex 类型和课堂上的 Picture 类型，完成一个分形图的制作。

此次制作分形图片的主角是 JuliaSet（朱利亚集合），该集合是由复平面上的若干个点构成，以法国数学家加斯顿·朱利亚（Gaston Julia）的名字命名。

设定一个常复数 c，对于复平面上的某个点 $z_0$ 完成如下公式计算：$z_{n+1} = (z_n)^2 + c$，反复迭代之后，得到一个序列：$z_1$，$z_2$，…，如果这个序列发散于无穷大，那么 $z_0$ 这个数就不在朱利亚集合

中，如果这个序列最终在某一范围内收敛于某一值，那么$z_0$这个数就在朱利亚集合中。将属于朱利亚集合中的点着黑色，不在朱利亚集合中的点着为不同等级的灰色（最终是白色），那么一副分形图就制作出来了。下面是 Complex 类型和 Picture 类型通力合作下生成的几副 Julia 分形图。

①当取 c=-0.285+0.156i 时：



②当取 c=0.285-0.01i 时：

为了得到上面的结果，需要完成如下工作：

工作 1：坐标转换。Picture 处理的图片是由像素点构成的，每个像素的取值是整数（示例中的图片大小是 1024*1024），而复平面中的点是小数（示例中的坐标 x 和 y 都取[-2,2]）。另外，还需要注意图片的坐标原点在左上角，而复平面的坐标点在正中央。转换工作自行完成。

工作 2：如何判定复平面的 $z_0$ 值在经过迭代运算后是发散，即趋于无穷。建议：迭代最大次数为 255 即可（选取 255 为最大迭代次数，主要是为了和颜色的分量最大取值建立关联）。另外，当迭代中如发现 $z_i$ 的幅值 > 2，那么就不用再去迭代了，直接可以判定该值会发散。该工作的示例代码如下，仅供同学们参考：

```
1.  Complex z = z0;
2.  int max = 255;
3.  for (int t = 0; t < max; t++){
4.      if (z.abs() > 2) return t;
5.      z = z.multiply(z).add(z);
6.  }
7.  return max;
```

工作 3：着色。图片共有 1024*1024 个点，取每一个点完成着色。假设取点(col, row)，首先将该坐标通过工作 1 进行坐标映射，将映射得到的复数代入工作 2 完成判定，用工作 2 的返回值生成一个色度的灰色（Color 中 red、green 和 blue 分量都设置为返回值大小即可）。将这个颜色设置到

设计：

一. 编写 Complex 类（大体和一题的 Complex 类相同，没有直接引用的原因是一题中的 Complex 类的加法运算及乘法运算会输出每一次结果，导致程序运行时间大大增高，效率太过低下，于是我重新编写了 Complex 类在 MakePicture 包内，可直接使用，并改写加法运算和乘法运算的代码，使其在原有功能不变的条件下不进行输出，仅做计算用）

二. 编写 Picture 类

1. 保留老师发布于钉钉的 Picture 类，在下述步骤中添加几个新的功能；

2. 编写 calculates（double real，double imag，Complex c）方法，三个参数依次为复数对象的实数部分、复数对象的虚数部分、给定的复数 c。创建新的复数对象 z，其实数部分和虚数部分和上述 real、imag 相等，再设一整数类型 max，其初始值为 255，调用 for 循环，次数为 255 次，通过 if 语句判断 z 与坐标原点距离是否大于 2（用 Complex 类中所写的 abs（）方法判断），若大于二则返回当前的循环次数的值，否则继续循环，直至结束时返回 255；

3. 编写 calculateColor（int k）方法，k 即为上述 calculates 方法中返回的值，调用 if 语句，若 k==255，

则将颜色变为黑色，否则设一新的整数类型 gray，其值为 255-k，返回一新创建的 Color 对象，其 rgb 均为 gray；编写 changeColor（Picture p，Complex c）方法。第一次调用 for 循环，循环次数 i 为 p.getWidth（）次，即为图片像素宽度大小，在循环内部调用第二次循环，循环次数 j 为 p.getHeight（）次，即为图片像素高度大小，在第二次循环内部创建新的 Complex 对象 z,其实部和虚部分别为(i-512)/256、(j-512)/256（转换坐标），后设整数类型数据 k，其值为 z 匹配给定的复数 c 调用 calculates 方法后所得的值，再创建 Color 对象 m，计算 calculateColor（k）并赋给 m，再用 setColor 方法将第 i 行 j 列的像素的颜色显示出来；

三. 编写 MakePicture（）类

1. 编写 main 函数

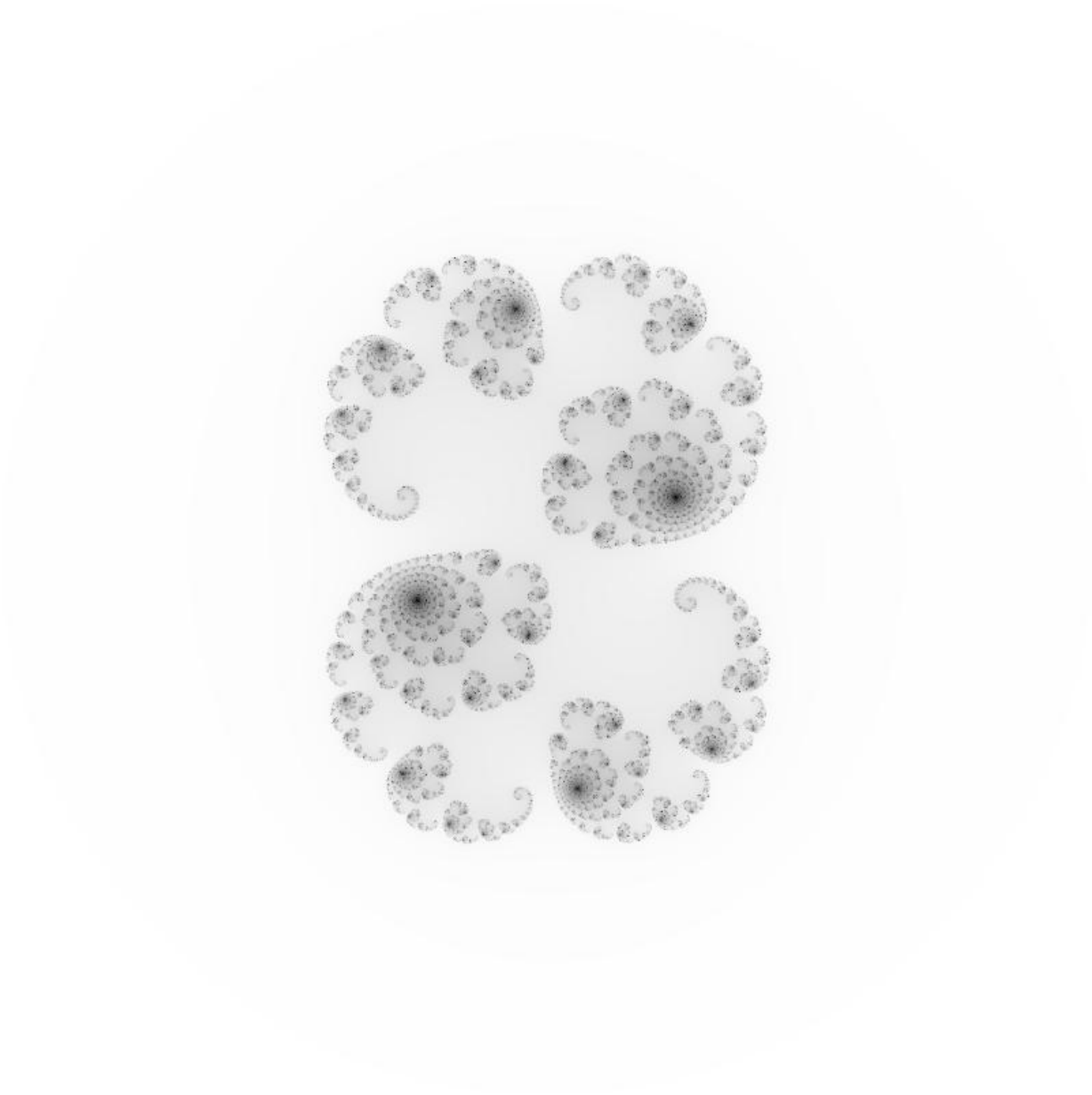
2. 使用 Complex 复数类的有参构造器存入五个给定的复数的值（分别为 c1=-0.8,0.156;c2=0.285,-0.01;c3=0.38,-0.18;c4=-0.835,-0.2321;c5=-0.7269,-0.1889）

3. 使用 Picture 类有参构造器创建 Picture 类型对象 JuliaSet（参数为 String filename，即“Julia.png”），对其调用 changeColor（JuliaSet，c1）方法，以得出其与 c1 匹配所得的图形；如上述步骤可依次调用 c2、c3、c4、c5 的图形，图形代码如下图展示。

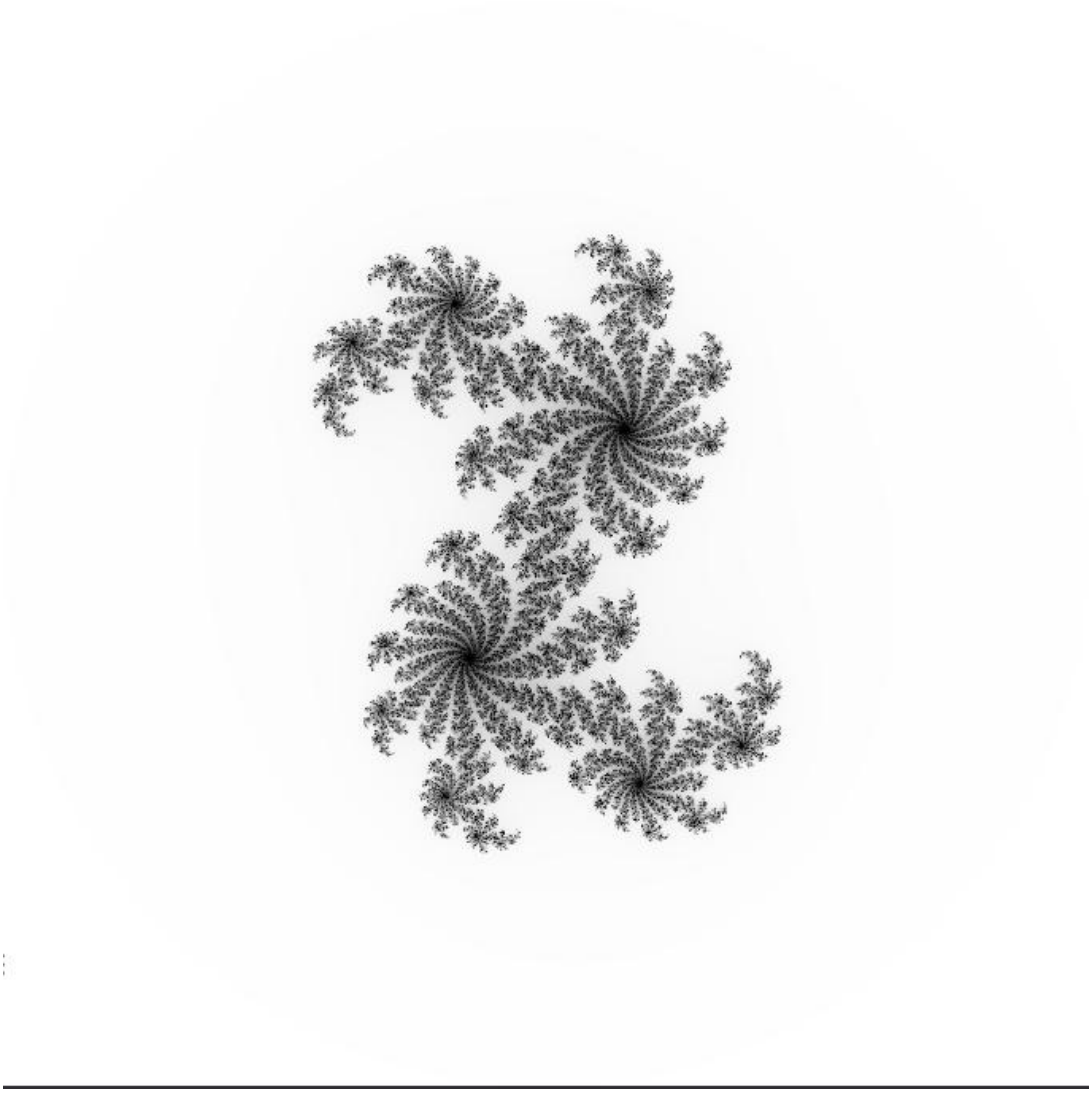
# 运行结果展示:

1.c1=-0.8,0.156;

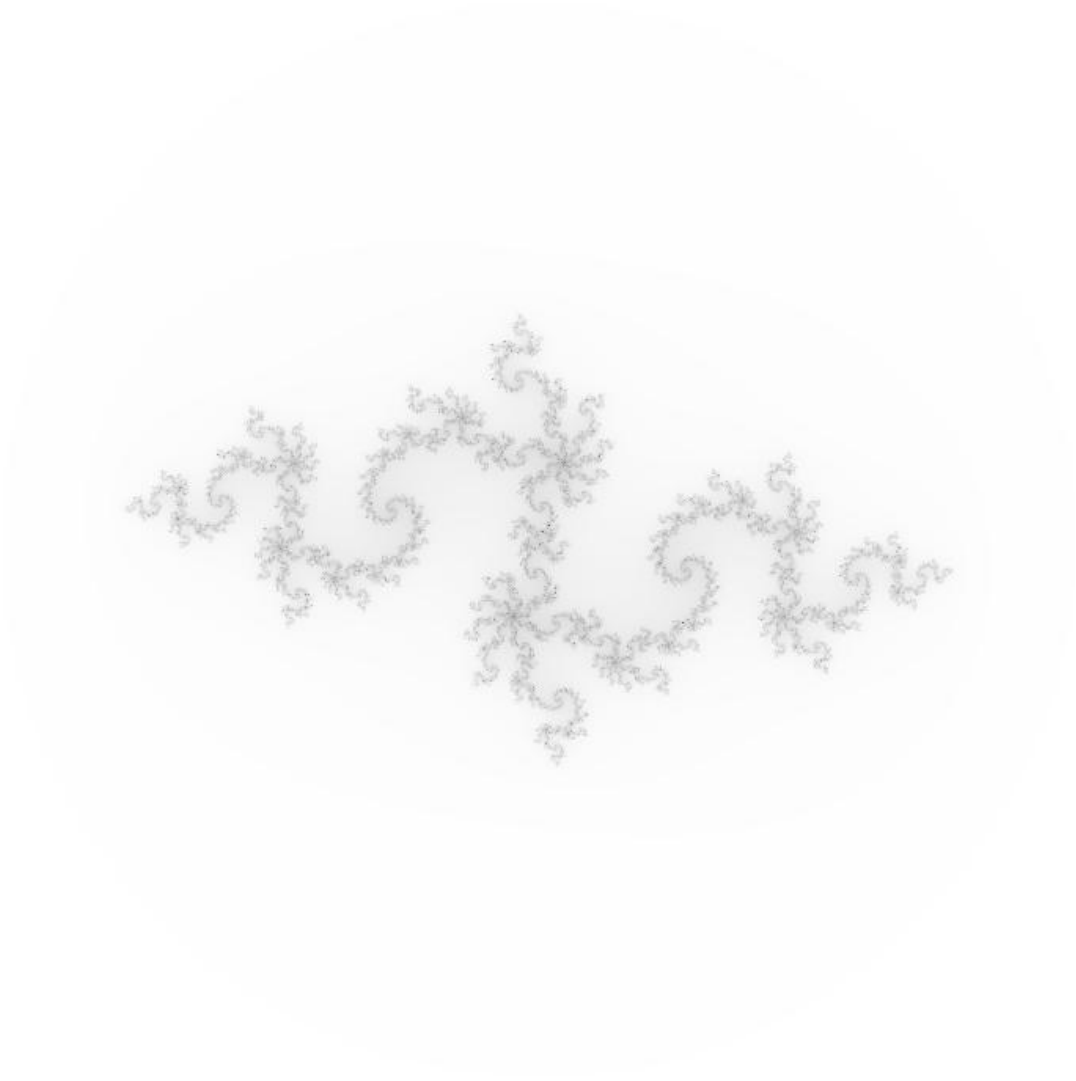
2.c2=0.285,-0.01;



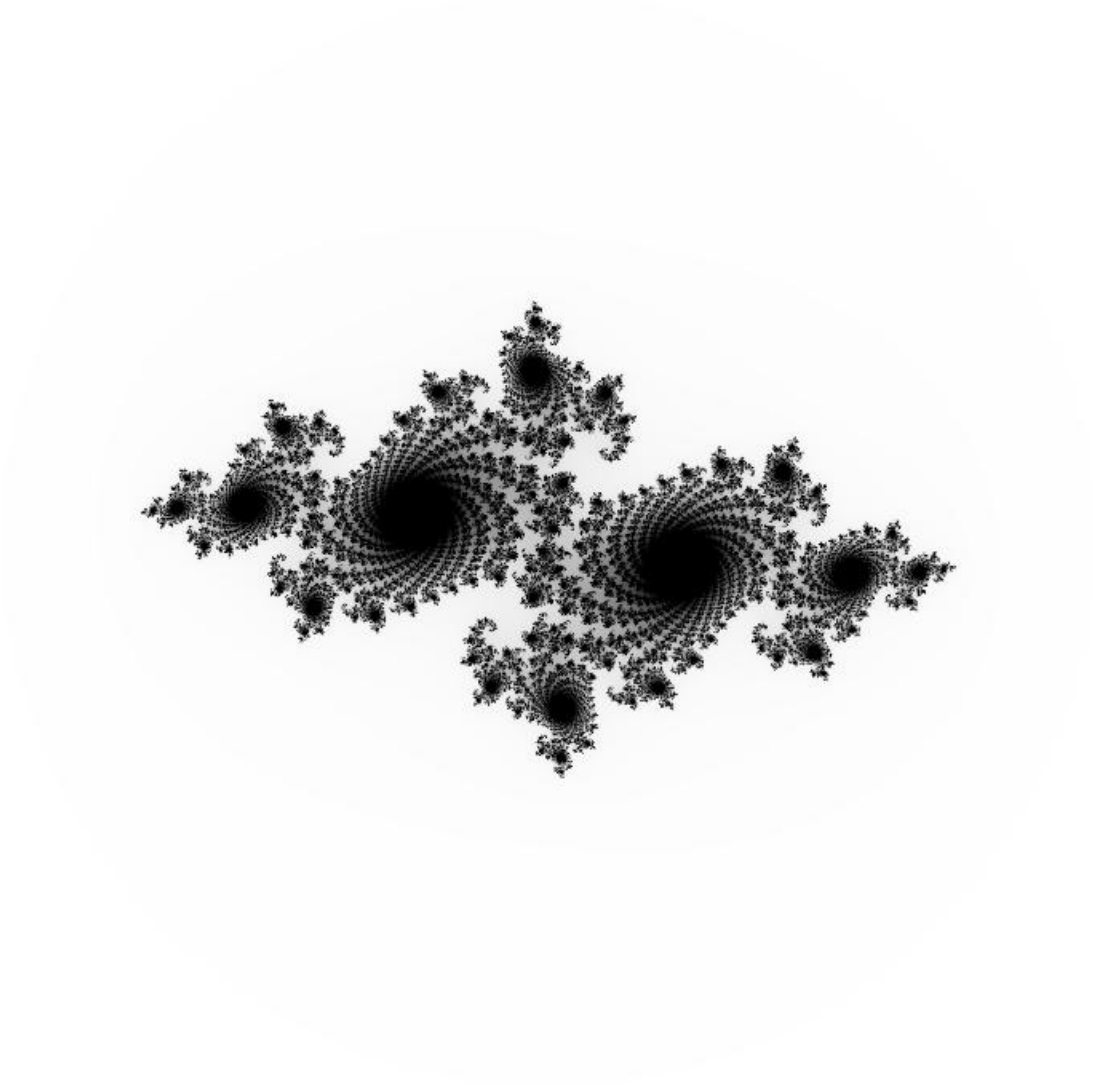
3.c3=0.38,-0.18;



4.c4=-0.835,-0.2321;



5.c5=-0.7269,-0.1889。



总结:

1. 当给定的复数 c 的值不同时，所得的图形形状、颜色相差较大;

2. 遇到诸如此题的难题时（对目前的我而言比较难），合理编写并运用更多的类类型和类方法往往能达到事半功倍的效果。

# 题目 3：模拟一个购物车

双 11 即将到来，购物车满起来!!!

创建一个购物车类类型（ShoppingCart），每一个 ShoppingCart 对象应该包含若干个商品（Item 类型）对象，购物车类型应该至少提供添加商品、删除商品以及汇总购物车中商品价格的能力。这是一个开放题目，发挥的空间很大。下面是对该题完成所需满足的基本要求：

要求 1：商品类型至少要有三种以上（不同类型的商品有不同的属性），为了复用和多态，请使用继承完成商品这一组类型的构建；

要求 2：不论是商品类型，还是购物车类型都需要实现 toString 方法，针对不同类型中包含的不同数据成员订制输出内容；

要求 3：购物车对象应该不限制包含商品的数量，这时需要使用具有可动态扩容能力的数组数据类型（ArrayList），这个类型是 Java API 中已经定义好的数据类型，具体使用方法参见本文的附录。

要求 4：购物车的商品要能排序，因此要求商品类必须实现 Comparable 接口，排序算法可以使用冒泡排序算法。商品排序的规则自行规定，需要在实验报告中说明。

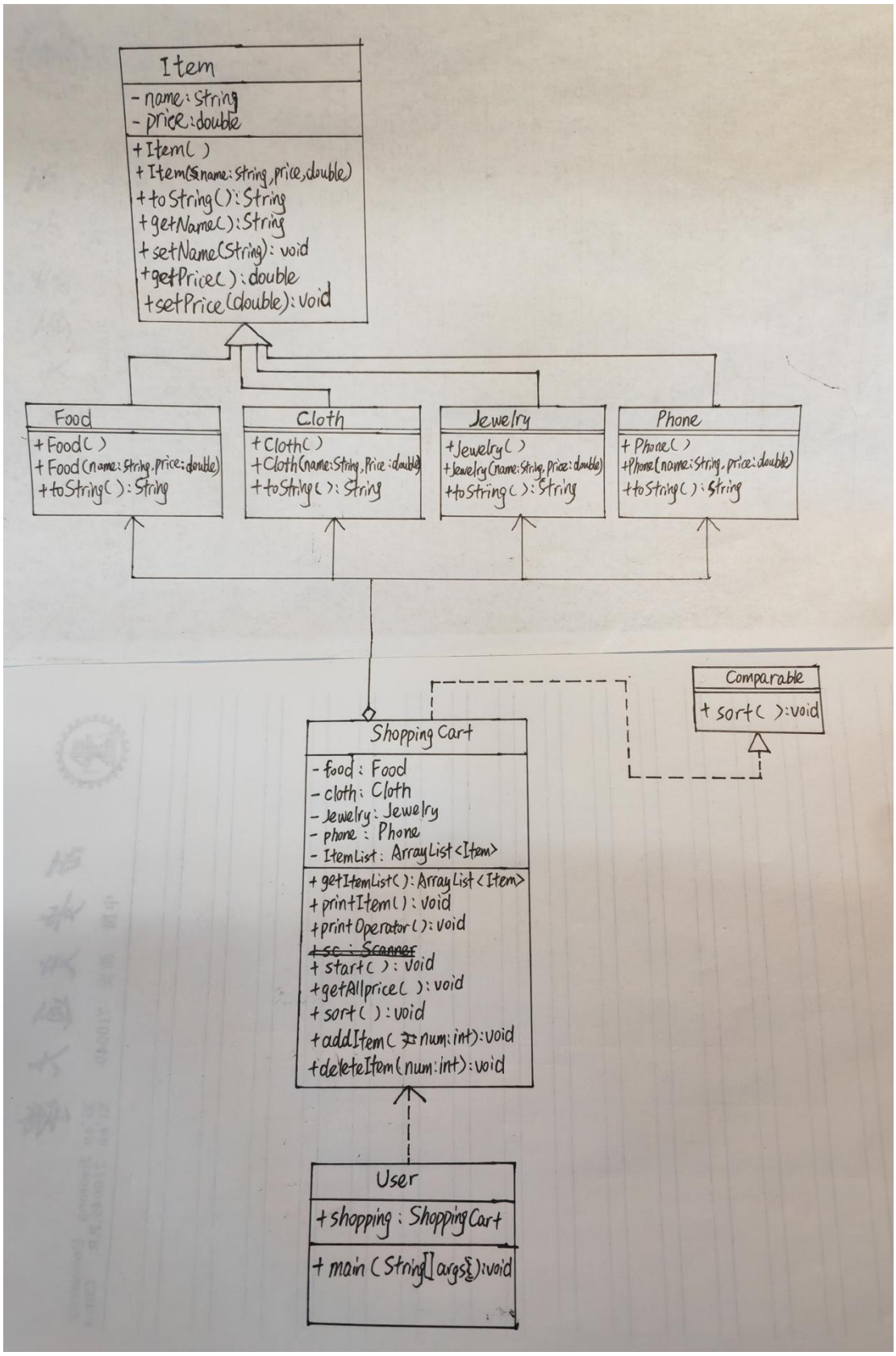
要求 5：该题在实验报告中的设计环节描述请使用 UML 类图完成。

要求 6：编写一个客户端程序，要有商品添加、商品删除、商品价格汇总以及商品排序等环节的调用和结果展示。

备注：ShoppingCart 类型和 Item 类型之间是一种一对多的组合关系，请自行查找并理解类型间的组合关系，以及在 UML 图中如何表示类型间组合关系的合法图元。

# 设计：

## UMI 类图：




具体代码设计:

一. 创建编写 Item 类

1. 创建数据成员 name、price，分别用于存储和输出商品类对象的名字和价格；

2. 编写 toString () 方法订制输出格式为{return this.name + “$”+ this.price;}

3. 构造有参构造器 Item（String name,double price），用于为输入的参数赋值（即将输入的商品的名字和价格存入该对象中）；

4. 为商品类对象的每一个数据成员（name 和 price）编写访问器和修改器，方便子类和其他类调用这些数据成员的值；

5. 编写 Food（）类类型，其继承了 Item（）类，编写 Food（）类的无参构造器，在 Food（）类里创建新的对象并编入食物的名称和其价格，使用 this 指针给外界调用 Food () 类创建的对象赋值（想要让外界创造的对象赋值，只需将 Food（）类里编写好的对象的值赋给外界对象即可，本题每个子类都只编写了一个写好的对象供调用）；

6. 编写 Cloth（）类、Jewelry（）类、Phone（）类三个类型，这三个类型均为 Item（）类的子类，其编写方法和上述 Food（）类的编写方法类似；

## 二．编写 ShoppingCart（）类

1.创建 food、cloth、jewelry、phone 对象，依次拷贝 Food（）类、Cloth（）类、Jewelry（）类、Phone（）类已有

的商品数据赋值在自己身上，方便后续代码调用这些商品对象；

2.创建 ArrayList<Item>类，向其添加、删除、输出对象，用以实现模拟购物车所需的几大功能；

3.编写 printOperate（）方法，仅用于输出一定的内容，且输出内容为编写者自己编写的为了方便用户使用而给出的操作指引；

4.编写 printItem（）方法，用途和上述 printOperate（）方法类似，仅用于向用户展示已有的商品对象的详细信息并对其编号，方便后续让用户选择编号而实现添加、删除该编号对应的商品的功能；

5.编写 Scanner（）类，用以后续让用户输入内容以使用购物车程序；

6.编写 start（）方法，用于模拟购物车的各种行为

（1）.编写 while（true）循环，其范围为整个 start（）方法，用于实现多次使用 start（）方法、让用户能不断使用添加商品、删除商品、汇总商品价格等功能；

（2）.调用 printOperate（）方法，使每次循环后都能再次显示编写者为用户提供的指引操作；

（3）.创建 Scanner 的对象，让用户输入 1~4 的数字以实现对应功能；

（4）.使用 switch（command）语句判断是否与用户输入的值相等，command 的值即为上述用户所输入的数字的值：若值为 1，则实现添加商品的功能；若值为 2，则实现删除商品的功能；若值为 3，则实现汇总购物车里商品的总价格的功能；若值为 4，则实现结账功能，并打印购物车里商品已有的商品及商品的总价格，最后再结束程序；若用户输入的是除上述外的其他内容，则输出内容提醒用户输入正确的数字；

7.编写 addItem（int num）方法，用于实现添加商品对象的功能：

（1）创建新的 Item（）商品类对象，并使用 ItemList.add（item）以存入该对象各个数据成员的信息(item)即为新创建的商品类对象的名称；

（2）调用 switch（num）语句，num 即为标题所写的引入的整数参数（num 的值实际上取决于用户输入的值）。若 num 为 1，则调用对 item 调用 setName（）和 setPrice（）方法（参数分别为 food.getName，food.getPrice）让 item 对象拷贝 food 对象的所有数据，并提醒用户成功添加了 food 对象这一商品；若 num 分别为 2,3,4，则分别让 item 对象拷贝 cloth 对象、jewelry 对象、phone 对象的所有数据（具体操作步骤如上述拷贝 food 对象）；若 num 为其他值，则提醒用户输入正确的值（注意此时已经添加了一个空的 item（）

到 ItemList 里，所以使用了 ItemList.remove（item）来删除这一空的对象）；

8.编写 deleteItem（int num）方法，用于实现删除商品的功能，其中 num 的值为用户自己输入的数字的值：

（1）调用 switch（num）方法，并为其贴上标签“OUT”方便循环内能直接终止 switch 循环；

（2）若 num=1，则使用 for 循环（循环次数为 ItemList 的长度），并用 if 语句判断该 ItemList 里是否有对象的 name 的值与已有的 food 对象的 name 的值相等，若 ItemList 中有对象与 food 对象 name 值相等则使用 remove 方法删除该对象，若没有则提醒用户其购物车里没有该商品；num=2，3,4,时分别为执行删除 Itemlist 中与 cloth 对象、jewelry 对象、phone 对象相等的对象，具体删除功能的实现参照上述对删除与 food 对象相等的对象的描写；若 num 的值为其他数字，则提醒用户输入正确的数字以删除对应商品；

9.编写 sort（）方法，用于对 ItemList 里的对象排序（具体排序方法为根据对象 price 的值从小到大排序）。第一次调用 for 循环，循环次数为 ItemList 的长度，创建 Item 类 item1 对象并将 ItemList 中第 i（i 为正在执行的循环的次数）个对象的值赋给 item1；第二次调用 for 循环，次数与第一次相同，再次创建 Item 类对象命名为 item2，并将 ItemList 中第 j（j 为正在执行的内循环的次数）个对象的值赋给 item2；调用

if 语句判断 item1 和 item2 的数据成员 price 的大小，若 price 的值 item1<item2，则将 item1 和 item2 交换位置（冒泡排序法）；

10.编写 getAllprice（）方法，用于计算 ItemList 中所有对象的 price 的总和。先创建一个 double 类型变量 sum，设定其初始值为 0.0，后用 for 循环将 ItemList 中每一个对象的 price 的值调出并累计在 sum 中，最后输出 sum；

三．编写 User 类，用户在此运行购物车程序

四．编写 Test 类，测试购物车程序的功能是否实现

```
public class Test {
    public static void main(String[] args){
        //新建购物车
        ShoppingCart shoppingCart = new ShoppingCart();
        //模拟添加两个苹果，一件衬衫，一部华为手机
        shoppingCart.addItem( num: 4);shoppingCart.addItem( num: 1);shoppingCart.addItem( num: 1);shoppingCart.addItem( num: 2);
        //试图输入给定商品编号外的数字
        shoppingCart.addItem( num: 8);
        //输出没有排序的购物车
        System.out.println("没有排序的购物车:"+shoppingCart.getItemList());
        //对购物车排序并将其输出
        shoppingCart.sort();
        System.out.println("排序后的购物车:"+shoppingCart.getItemList());
        //删除购物车中的一个苹果，并试图删除一个24K纯金项链（购物车里没有这个商品）
        shoppingCart.deleteItem( num: 1);shoppingCart.deleteItem( num: 3);
        System.out.println("删除了一个苹果的购物车:"+shoppingCart.getItemList());
        //试图删除给定商品编号外的数字的商品
        shoppingCart.deleteItem( num: 6);
        //计算购物车商品总价格
        shoppingCart.getAllprice();
    }
}
```

# 运行结果展示：

## 一．User 类（模拟用户购物过程，步骤随机）

```
请选择以下操作:
1.添加商品
2.删除商品
3.汇总商品价格
4.付账并退出购物车程序
2
请输入您想删除的商品的编号:
1
您的购物车里没有苹果
请选择以下操作:
1.添加商品
2.删除商品
3.汇总商品价格
4.付账并退出购物车程序
1
商店里的商品如下:
1.苹果，4元
2.衬衫，100元
3.24K纯金项链,5000元
4.华为mate60,8899元
请输入您想添加的商品的编号:
1
您已添加商品:苹果,价格为:4.0元
请选择以下操作:
1.添加商品
2.删除商品
3.汇总商品价格
4.付账并退出购物车程序
2
请输入您想删除的商品的编号:
1
您删除了一个苹果
```

```
请选择以下操作:
1.添加商品
2.删除商品
3.汇总商品价格
4.付账并退出购物车程序
1
商店里的商品如下:
1.苹果，4元
2.衬衫，100元
3.24K纯金项链,5000元
4.华为mate60,8899元
请输入您想添加的商品的编号:
4
您已添加商品:华为mate60,价格为:8899.0元
请选择以下操作:
1.添加商品
2.删除商品
3.汇总商品价格
4.付账并退出购物车程序
1
商店里的商品如下:
1.苹果，4元
2.衬衫，100元
3.24K纯金项链,5000元
4.华为mate60,8899元
请输入您想添加的商品的编号:
4
您已添加商品:华为mate60,价格为:8899.0元
```

```
请选择以下操作:
1.添加商品
2.删除商品
3.汇总商品价格
4.付账并退出购物车程序
1
商店里的商品如下:
1.苹果，4元
2.衬衫，100元
3.24K纯金项链,5000元
4.华为mate60,8899元
请输入您想添加的商品的编号:
2
您已添加商品:衬衫,价格为:100.0元
请选择以下操作:
1.添加商品
2.删除商品
3.汇总商品价格
4.付账并退出购物车程序
1
商店里的商品如下:
1.苹果，4元
2.衬衫，100元
3.24K纯金项链,5000元
4.华为mate60,8899元
请输入您想添加的商品的编号:
3
您已添加商品:24k纯金项链,价格为:5000.0元
```

```
请选择以下操作:
1.添加商品
2.删除商品
3.汇总商品价格
4.付账并退出购物车程序
1
商店里的商品如下:
1.苹果，4元
2.衬衫，100元
3.24K纯金项链,5000元
4.华为mate60,8899元
请输入您想添加的商品的编号:
4
您已添加商品:华为mate60,价格为:8899.0元
请选择以下操作:
1.添加商品
2.删除商品
3.汇总商品价格
4.付账并退出购物车程序
2
请输入您想删除的商品的编号:
4
您删除了一部华为mate60
请选择以下操作:
1.添加商品
2.删除商品
3.汇总商品价格
4.付账并退出购物车程序
3
您的购物车里有:
衬衫 24k纯金项链 华为mate60 华为mate60
您购买物品的总价格为:22898.0
```

```
请选择以下操作:
1.添加商品
2.删除商品
3.汇总商品价格
4.付账并退出购物车程序
1
商店里的商品如下:
1.苹果，4元
2.衬衫，100元
3.24K纯金项链,5000元
4.华为mate60,8899元
请输入您想添加的商品的编号:
2
您已添加商品:衬衫,价格为:100.0元
请选择以下操作:
1.添加商品
2.删除商品
3.汇总商品价格
4.付账并退出购物车程序
4
您的购物车里有:
衬衫 衬衫 24k纯金项链 华为mate60 华为mate60
您购买物品的总价格为:22998.0
谢谢惠顾

Process finished with exit code 0
```

二．Test 类（测试购物车程序功能是否实现，测试代码如上）

```
您已添加商品:华为mate60,价格为:8899.0元
您已添加商品:苹果,价格为:4.0元
您已添加商品:苹果,价格为:4.0元
您已添加商品:衬衫,价格为:100.0元
请输入正确的商品编号以添加该商品!
没有排序的购物车:[华为mate60 $8899.0, 苹果 $4.0, 苹果 $4.0, 衬衫 $100.0]
排序后的购物车:[苹果 $4.0, 苹果 $4.0, 衬衫 $100.0, 华为mate60 $8899.0]
您删除了一个苹果
您的购物车里没有24K纯金项链
删除了一个苹果的购物车:[苹果 $4.0, 衬衫 $100.0, 华为mate60 $8899.0]
请输入正确的商品编号以删除该商品!
您的购物车里有:
苹果 衬衫 华为mate60
您购买物品的总价格为:9003.0

Process finished with exit code 0
```

## 总结与收获:

1. 为实现对象的动态扩容功能，通过翻阅教材等方法学习并掌握了 ArrayList 类型的基础功能的使用，并在后续编程购物车程序中再次巩固加深了对其的印象;

2. 在编程过程中逐步巩固加深了对继承、接口的正确使用;

# 题目 4： 撰写继承、多态和接口方面的知识梳理

代码编写是对理论知识的一种实践。能够编写出正确、合理、有效的代码，理论知识必须得扎实。同学们需要沉下心来认真阅读参考资料、课件，再结合课堂所讲和课下所练，总结面向对象中对继承、接口以及多态的理解和认识。

# 总结:

## 一.继承

### 1.概述

(1).继承是使用已存在的类的定义作为基础建立新类的技术，新类的定义可以增加新的数据或新的功能，也可以用父类的功能，但不能选择性地继承父类。通过使用继承能够非常方便地复用以前的代码，能够大大的提高开发的效率。

(2).继承所描述的是“is-a”的关系，如果有两个对象 A 和 B，若可以描述为“A 是 B”，则可以表示 A 继承 B，其中 B 是被继承者称之为父类或者超类，A 是继承者称之为子类或者派生类。

(3).继承者是被继承者的特殊化，它除了拥有被继承者的特性外，还拥有自己独有的特性。例如猫有抓老鼠等其他动物没有的特性。同时在继承关系中，继承者完全可以替换被继

承者，反之则不可以，例如我们可以说猫是动物，但不能说动物是猫就是这个道理，这在继承关系中被称为“向上转型”。

(4).对于若干个相同或者相识的类，我们可以抽象出他们共有的行为或者属相并将其定义成一个父类或者超类，然后用这些类继承该父类，他们不仅可以拥有父类的属性、方法还可以定义自己独有的属性或者方法。

(5).重点：

1).子类拥有父类非 private 的属性和方法；

2).子类可以拥有自己的属性和方法，即可对父类进行拓展；

3).子类可以重写父类的方法。

## 2.构造器

(1).一般而言，子类可以继承父类的属性和方法，除非是声明了 private 的变量或者方法，此外，子类也无法继承父类的构造器。对于构造器而言，它只能够被调用，而不能被继承。 调用父类的构造方法我们使用 super()即可。构建过程是从父类“向外”扩散的，也就是从父类开始向子类一级一级地完成构建。而且我们并没有显示的引用父类的构造器，因为编译器会默认给子类调用父类的构造器，但是这个默认调

用父类的构造器是有前提的：父类有默认构造器。如果父类没有默认构造器，我们就要必须显示的使用 super()来调用父类构造器，否则编译器会报错。

(2).对于子类而言,其构造器的正确初始化是非常重要的,而且只有一个方法可以保证这点：在构造器中调用父类构造器来完成初始化，而父类构造器具有执行父类初始化所需要的所有知识和能力。对于继承而言，子类会默认调用父类的构造器，但是如果没有默认的父类构造器，子类必须要显示的指定父类的构造器。

### 3.注意事项

(1).父类变，子类就必须变。

(2).继承破坏了封装，对于父类而言，它的实现细节对与子类来说都是透明的。

(3).继承是一种强耦合关系。

## 二.接口

### 1.概述

(1).什么是接口：Java 中的接口是一系列方法的声明，是一些方法特征的集合；接口只有方法的特征(只有声明) 没有

方法的实现(没有方法体)，因此这些方法可以在不同的地方被不同的类实现，而这些实现可以具有不同的行为；如果说类的内部封装了成员变量、构造方法和成员方法，那么接口的内部主要就是封装了方法，包含抽象方法、默认方法和静态方法。

(2).为什么需要接口：是多态的基础；可以多实现。

2.接口的定义格式：使用关键字 interface

3.接口的使用方式

(1).类与接口的关系为实现关系，即类实现接口，该类可以称为接口的实现类，也可以称为接口的子类。实现的动作类似继承，格式相仿，只是关键字不同，实现使用 implements 关键字。

(2).非抽象类实现接口注意事项:必须重写接口中所有抽象方法;继承了接口的默认方法，即可以直接调用，也可以重写;对于接口中定义的抽象方法，子类必须全部实现（重写）;对于接口中的默认方法，子类可以继承，也可以重写,但是只能通过实现类的对象来调用;对于接口中的静态方法，静态与.class 文件相关，只能使用接口名调用，不可以通过实现类的类名或者实现类的对象调用；接口中，无法定义成员变量，但是可以定义常量，其值不可以改变，默认使用 public static final 修饰。

4.接口的多实现

(1).在继承体系中，一个类只能继承一个父类（单继承）。而对于接口而言，一个类是可以实现多个接口的，这叫做接口的多实现。并且，一个类能继承一个父类，同时实现多个接口。

(2).接口多实现的抽象方法：接口中，有多个抽象方法时，实现类必须重写所有抽象方法 。如果抽象方法有重名的，只需要重写一次。

(3).接口多实现的默认方法：接口中，有多个默认方法时，实现类都可继承使用。如果默认方法有重名的，必须重写一次。

(4).接口多实现中的静态方法：接口中，存在同名的静态方法并不会冲突，原因是只能通过各自接口名访问静态方法。

5.接口的多继承

(1).一个接口能继承另一个或者多个接口，这和类之间的继承比较相似。

(2).接口的继承使用 extends 关键字，子接口继承父接口的方法。如果父接口中的默认方法有重名的，那么子接口需要重写一次。

6.总结抽象类和接口的区别

相同点：

(1).都位于继承的顶端，用于被其他类实现或继承；

(2).都不能直接实例化对象；

(3).都包含抽象方法，其子类都必须覆写这些抽象方法；

区别：

(1).抽象类为部分方法提供实现，避免子类重复实现这些方法，提高代码重用性；接口只能包含抽象方法；

(2).一个类只能继承一个直接父类(可能是抽象类)，却可以实现多个接口(接口弥补了 Java 的单继承)；

(3).抽象类为继承体系中的共性内容，接口为继承体系中的扩展功能；

# 三.多态

## 1.多态综述

所谓多态就是指程序中定义的引用变量所指向的具体类型和通过该引用变量发出的方法调用在编程时并不确定，而是在程序运行期间才确定，即一个引用变量倒底会指向哪个类的实例对象，该引用变量发出的方法调用到底是哪个类中实现的方法，必须在由程序运行期间才能决定。因为在程序运行时才确定具体的类，这样，不用修改源程序代码，就可以让引用变量绑定到各种不同的类实现上，从而导致该引用调用的具体方法随之改变，即不修改程序代码就可以改变程序运行时所绑定的具体代码，让程序可以选择多个运行状态，这就是多态性。

指向子类的父类引用由于向上转型了，它只能访问父类中拥有的方法和属性，而对于子类中存在而父类中不存在的方法，该引用是不能使用的，尽管是重载该方法。若子类重写了父类中的某些方法，在调用该些方法的时候，必定是使用子类中定义的这些方法（动态连接、动态调用）。

对于面向对象而言，多态分为编译时多态和运行时多态。其中编辑时多态是静态的，主要是指方法的重载，它是根据参数列表的不同来区分不同的函数，通过编辑之后会变成两个不同的函数，在运行时谈不上多态。而运行时多态是动态的，它是通过动态绑定来实现的，也就是我们所说的多态性。

2.多态的实现条件

Java 实现多态有三个必要条件：继承、重写、向上转型。

(1).继承：在多态中必须存在有继承关系的子类和父类。

(2).重写：子类对父类中某些方法进行重新定义，在调用这些方法时就会调用子类的方法。

(3).向上转型：在多态中需要将子类的引用赋给父类对象，只有这样该引用才能够具备技能调用父类的方法和子类的方法。

只有满足了上述三个条件，我们才能够在同一个继承结构中使用统一的逻辑实现代码处理不同的对象，从而达到执行不同的行为。对于 Java 而言，它多态的实现机制遵循一个原则：当超类对象引用变量引用子类对象时，被引用

对象的类型而不是引用变量的类型决定了调用谁的成员方法，但是这个被调用的方法必须是在超类中定义过的，也就是说被子类覆盖的方法。

3.多态的实现形式

在 Java 中有两种形式可以实现多态：继承和接口。

(1).基于继承实现的多态:

基于继承的实现机制主要表现在父类和继承该父类的一个或多个子类对某些方法的重写，多个子类对同一方法的重写可以表现出不同的行为。

基于继承实现的多态可以总结如下：对于引用子类的父类类型，在处理该引用时，它适用于继承该父类的所有子类，子类对象的不同，对方法的实现也就不同，执行相同动作产生的行为也就不同。如果父类是抽象类，那么子类必须要实现父类中所有的抽象方法，这样该父类所有的子类一定存在统一的对外接口，但其内部的具体实现可以各异。这样我们就可以使用顶层类提供的统一接口来处理该层次的方法。

(2).基于接口实现的多态

继承是通过重写父类的同一方法的几个不同子类来体现的，那么接口就是通过实现接口并覆盖接口中同一方法的几不同的类体现的。

在接口的多态中，指向接口的引用必须是指定这实现了该接口的一个类的实例程序，在运行时，根据对象引用的实际类型来执行对应的方法。

继承都是单继承，只能为一组相关的类提供一致的服务接口。但是接口可以是多继承多实现，它能够利用一组相关或者不相关的接口进行组合与扩充，能够对外提供一致的服务接口，所以它相对于继承来说有更好的灵活性。

当超类对象引用变量引用子类对象时，被引用对象的类型而不是引用变量的类型决定了调用谁的成员方法，但是这个被调用的方法必须是在超类中定义过的，也就是说被子类覆盖的方法。（但是如果强制把超类转换成子类的话，就可以调用子类中新添加而超类没有的方法了。）

当超类对象引用变量引用子类对象时，被引用对象的类型而不是引用变量的类型决定了调用谁的成员方法，但是这个被调用的方法必须是在超类中定义过的，也就是说被子类覆盖的方法。