**信息科学与工程学院**

**2020－2021学年第一学期**

实 验 报 告

课程名称： Java编程技术

实验名称： 实验二

专 业 班 级 通信工程 二班

学 生 学 号 201800121050

学 生 姓 名 孟麟芝

实 验 时 间 2020年10月7日

实验报告

【实验目的】

1.掌握简单的Applet程序的编写方法，为GUI的编写奠定基础。

2.掌握理解递归的思想。

3.继续熟练使用java语句进行编程。

【实验要求】

1.尝试变写一个简单的Applet程序。

2.编写控制台程序输出1--200之间的所有素数。

3.嵌套循环（for + while）实现1！+2！+3！+ ... 9!即阶乘之和。(如果能采用递归算法实现阶乘，再调用此函数实现求和，更妙)

4.尝试用applet编写程序实现摄氏温度和华氏温度的转换。

5.利用if语句实现阶梯电费的问题：用电量240度以下（包含240度）每度电0.55元，用电量在240-540度之间，超出240度电的部分按0.70元计算，用电量超过540度电，超出540度电的部分按照0.95元计算。（仅保留小数点后两位）

## 【实验具体内容】

## 【第一个实验】

1.Applet源代码

package applet;

import java.applet.\*;

import java.awt.\*;

public class MyFirstApplet extends Applet {

    public void paint(Graphics g) {

        String s1, s2;

        g.setColor(Color.blue);

        s1 = "这是一个Java Applet程序";

        s2 = "我改变了字体";

        g.drawString(s1, 20, 20);

        g.setColor(Color.red);

        g.setFont(new Font("宋体", Font.BOLD, 36));

        g.drawString(s2, 20, 100);

    }

}

2.HTML源代码

<title>Test</title>

<hr>

<applet code="applet.MyFirstApplet.class" width=300 height=150>

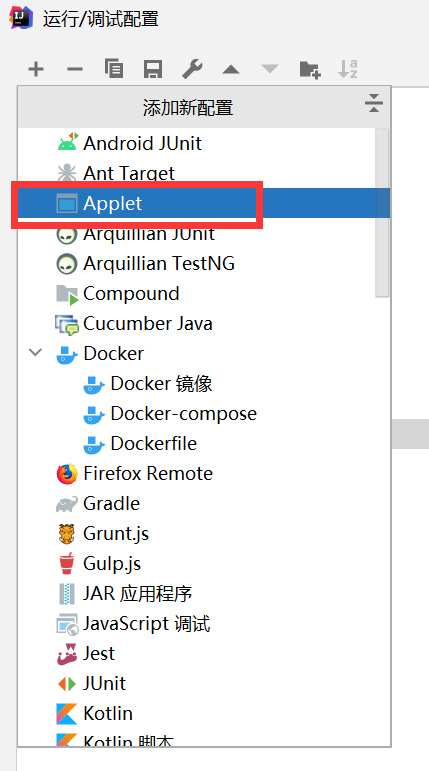
</applet>

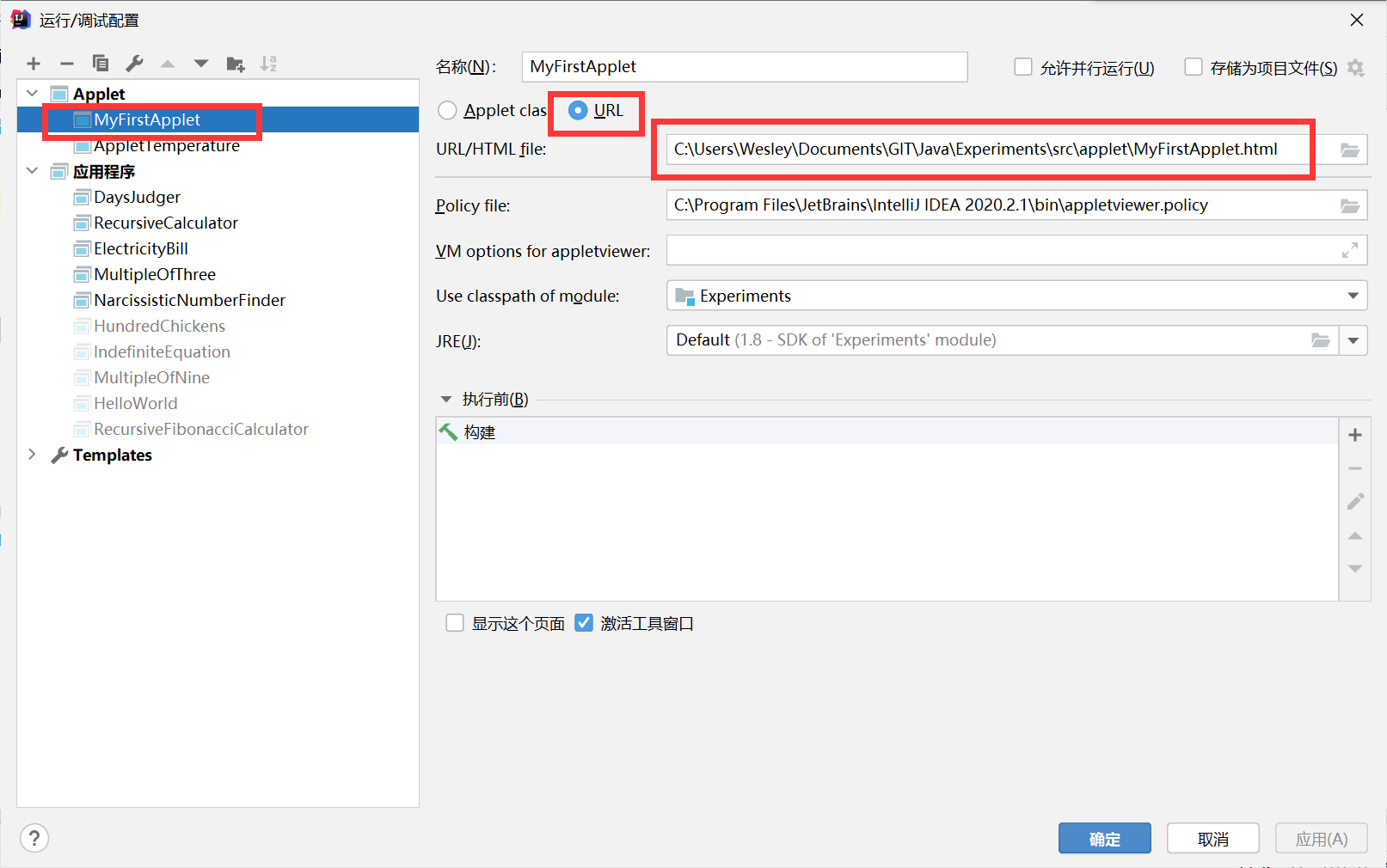
<hr>

3.实验记录与说明分析

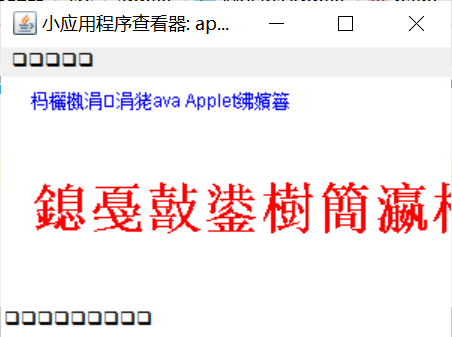
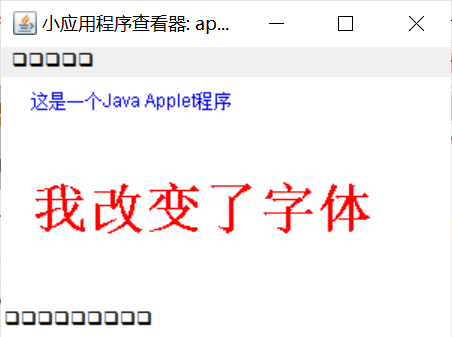
（1）先使用集成开发环境实现。在IDEA中新建一个项目，然后编写好Applet和HTML文件，将其放入项目，需要注意的是由于本实验中Applet带包名，在HTML中引用时也要带包名。

（2）运行之前需要进行运行配置，安装好Applet插件，作如下配置即可：

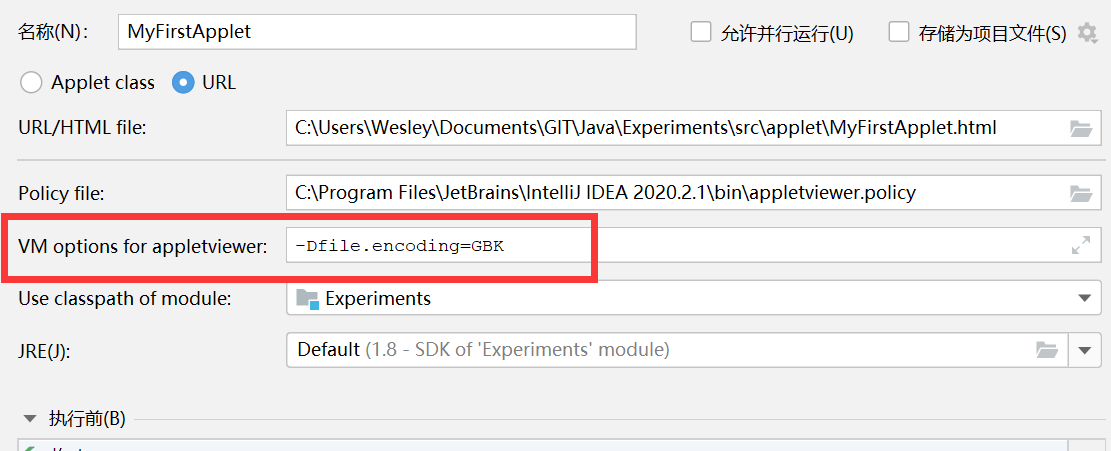


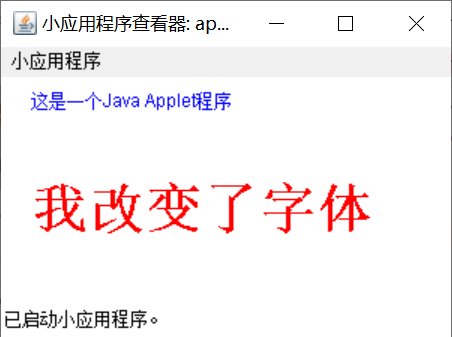


（3）直接点击运行，即可运行编写的Applet程序了，要注意的是，由于内置浏览器和小程序运行器都是GBK编码，如果使用UTF8编码将会乱码，修改编码即可解决：

这里若出现方框，增加以下配置即可解决问题：



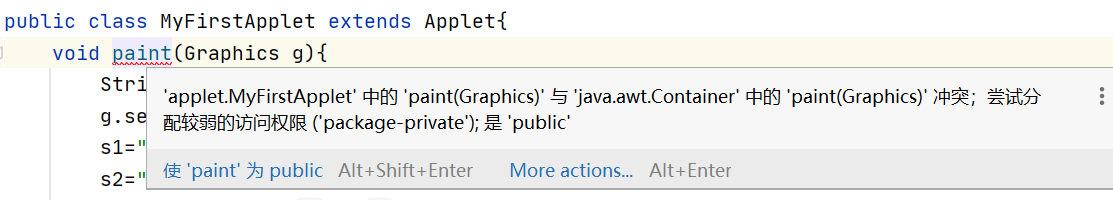


另外，现在大多浏览器已经不支持Applet，故不可在浏览器中直接运行Applet。且Java较新的版本也已经无法正常编译Applet程序。

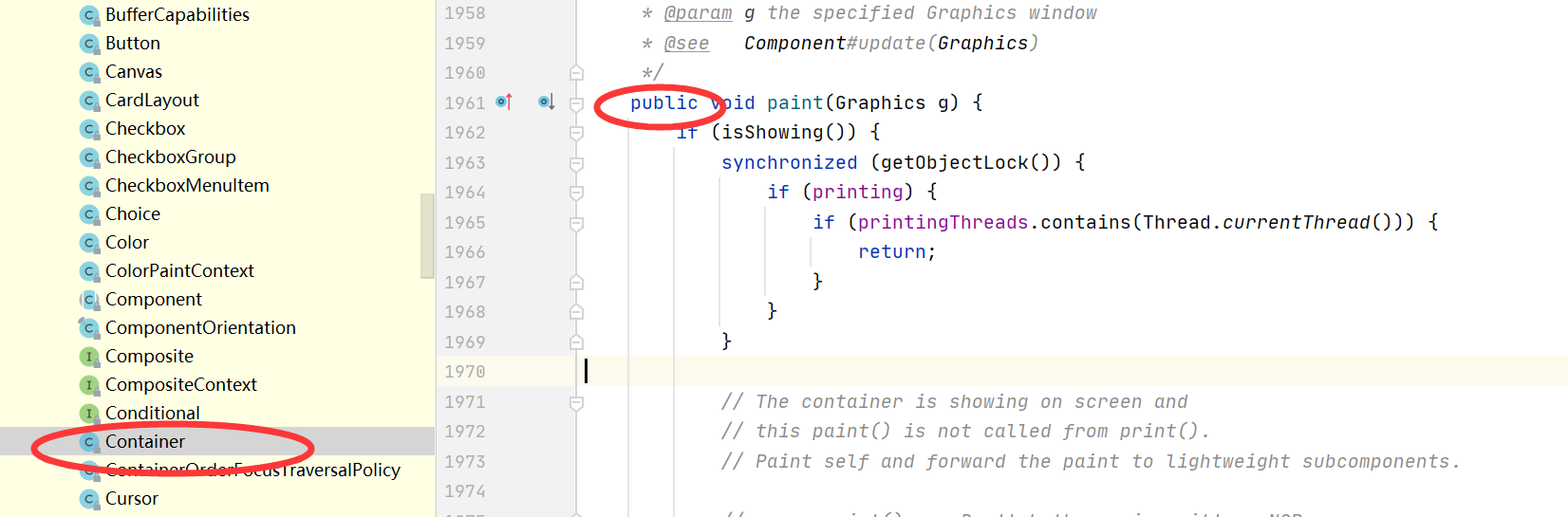
思考：

1) 程序中的主类如果不用public修饰，编译能通过吗？

答：这时，编译器将会报错，提示冲突，并无法覆盖原方法。从报错的描述可见这里的paint是对父类中方法的重写。



于是我们便可以到父类中去寻找一下，的确找到了这一方法，其修饰符为public

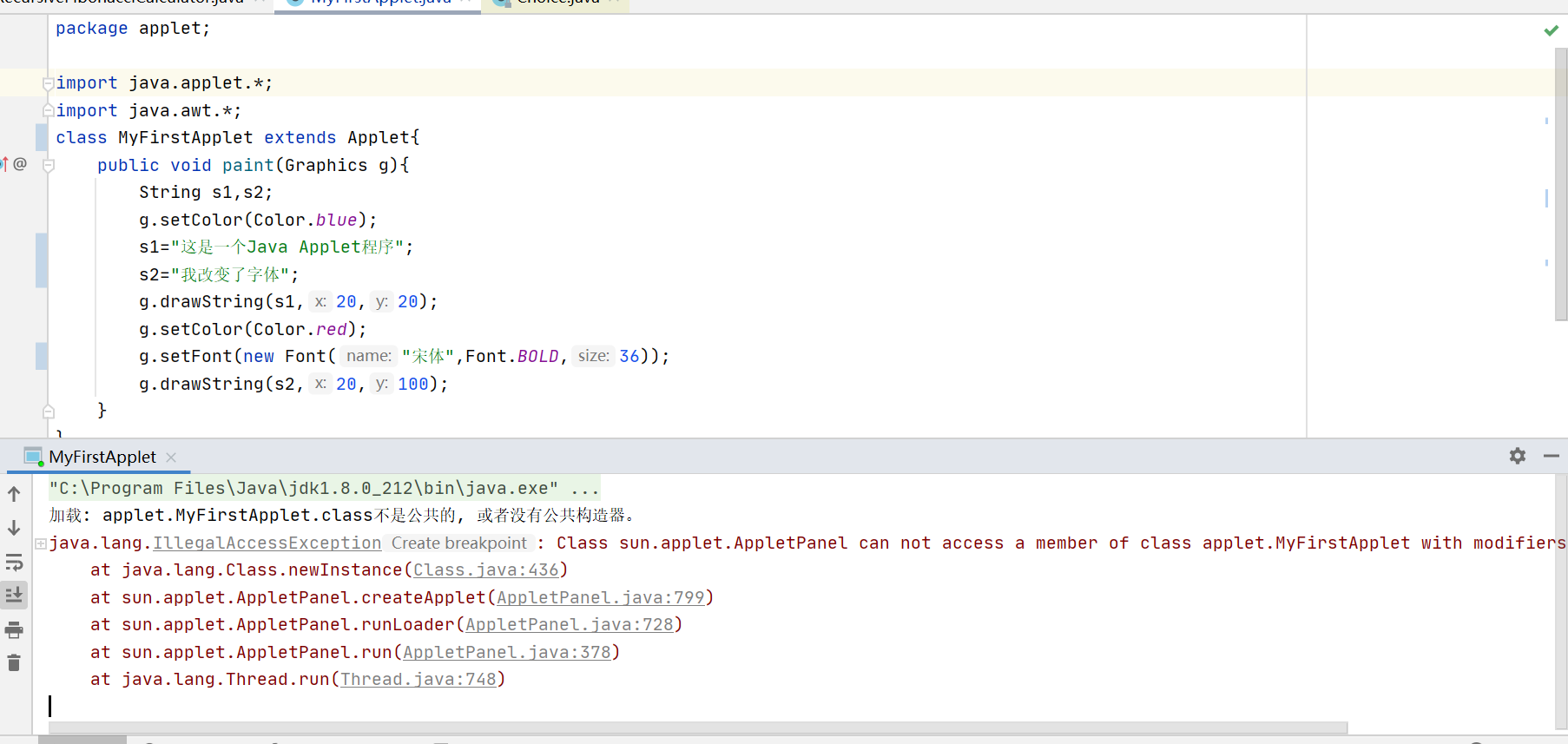


而若重写父类中的方法，其权限不可以比原方法更窄，故这里不加修饰符即为默认权限，从下表可见其宽松度是介于protected和private之间的，必然比public更窄，故会报错。这时程序也不能正常的运行。

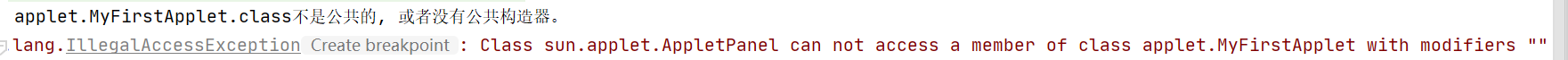
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **访问级别** | **访问控制修饰符** | **同类** | **同包** | **子类(不同包)** | **不同包(其他类)** |
| 公共 | public | 允许 | 允许 | 允许 | 允许 |
| 受保护 | protected | 允许 | 允许 | 允许 | 不允许 |
| 默认 | 没有修饰符 | 允许 | 允许 | 不允许 | 不允许 |
| 私有 | private | 允许 | 不允许 | 不允许 | 不允许 |

2) 程序中的主类如果不用public修饰，程序能正确运行吗？

答：若不加public修饰，发现运行出现报错。



这几句重点在于这一句，比较能体现问题：



这意味着另一个类AppletPanel无法访问到程序的主类。我们该如何找到这个类呢？在程序中写出这一行，按住ctrl点击AppletPanel即可转到这个类



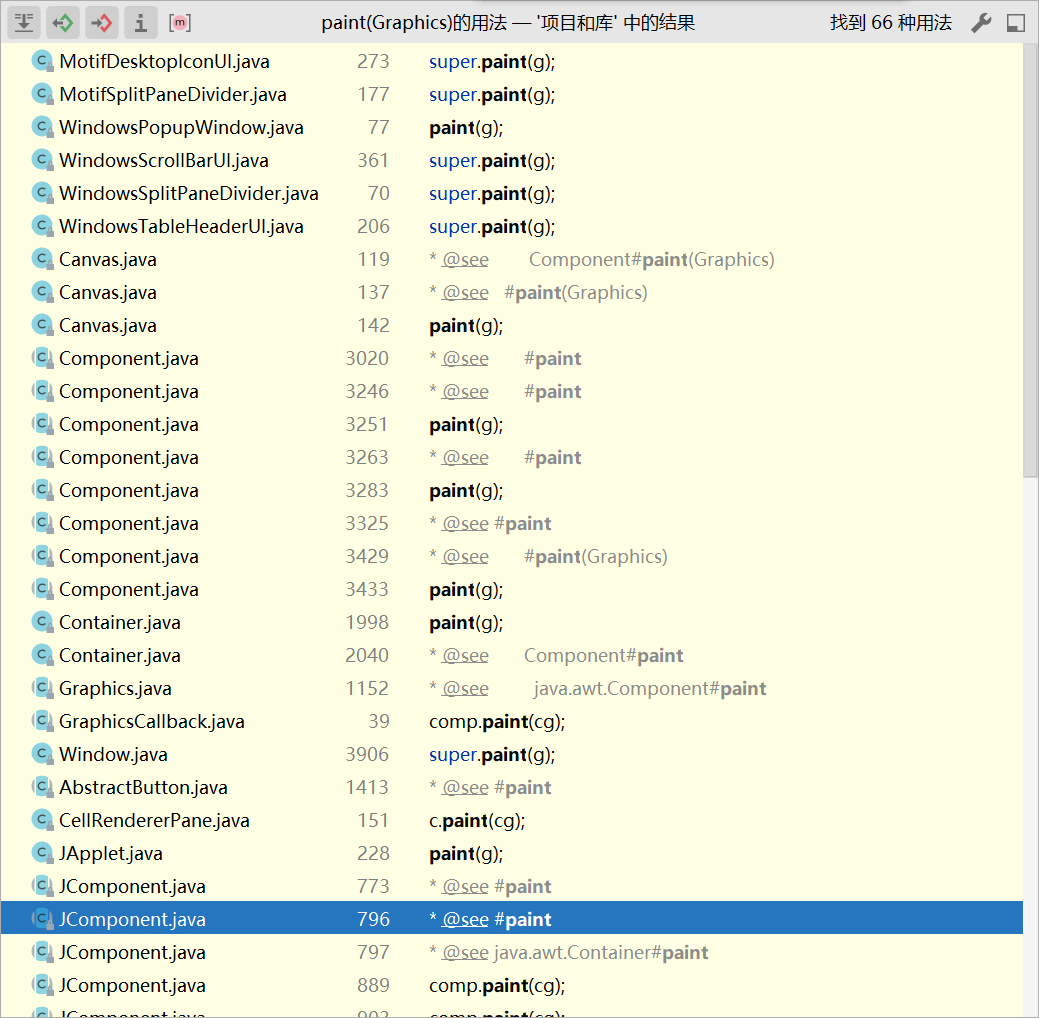


经过查看，可以猜测这个类的作用是对图形界面进行绘制，进一步可以猜测需要访问我们的主类中的成员，故去掉public会使之无法访问，出现问题。

3) 程序将paint方法误写成Paint，编译能通过么？

答：从原理上将，这里只修改了方法的名字，编译器应该不会报错，但是这样会导致这一paint方法不再是原方法的重写，可能会因其他类调用不到而出错。

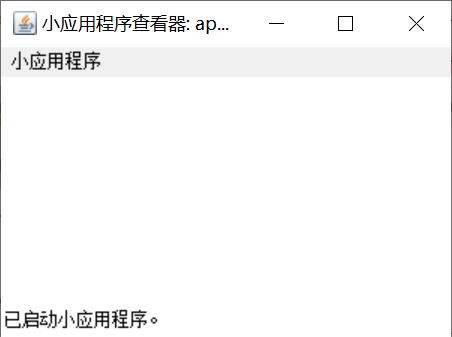
查找以下paint被调用的情况，发现它被在许多地方调用过，且各自的父类可能相同可能不同。。



至于具体的交叉调用关系，则较为复杂，不再深究。

4) 程序将paint方法误写成Paint，运行时能看到有关的输出信息吗？

答：如上所述，这时很可能得不到输出，事实也确实如此。



## 【第二个实验】

1.实验源代码

package arithmetic.prime;

public class PrimeNumberFinder {

    public static void main(String[] args) {

        int count = 1;

        System.out.println(2 + " 是素数");

        // 单独对2打印后，只需筛查剩下的奇数

        for (int i = 3; i <= 200; i += 2) {

            boolean flag = true;

**// 根据整数的质因数分解理论，其实只需筛查3~根号n之间的质数是否为其因子，下面的筛选范围广一些，为范围内的奇数**

**for (int j = 3; j <= java.lang.Math.sqrt(i); j += 2) {**

**if (i % j == 0) {**

**flag = false;**

**break;**

**}**

**}**

**if (flag) {**

**count++;**

**System.out.println(i + " 是素数");**

            }

        }

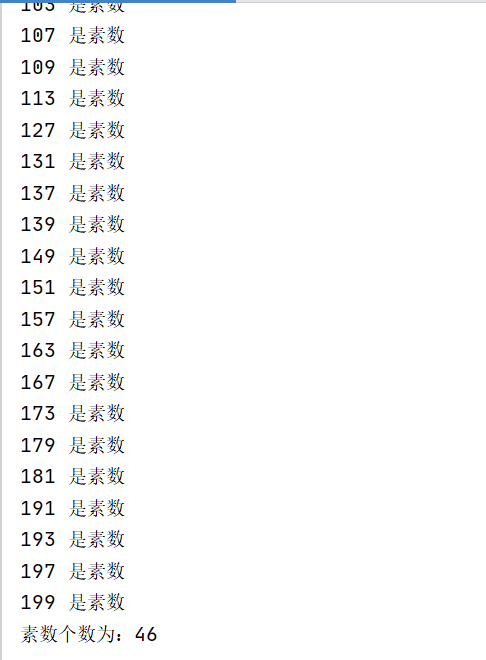
        System.out.println("素数个数为：" + count);

    }

}

2.实验记录与说明分析

算法的原理是这样的，要想筛查一个范围内的素数，首先由于偶数全部含有因子2，则应全部排除，下面只需筛选奇数，而要验证一个奇数是不是素数。只需验证之间的奇数是否有其因子，若有则不是素数，若没有则是素数。



可见程序输出了正确的结果。

对素数的筛法其实还可以利用埃氏筛法优化，对于任意一个数字，均可以分解成，其中互质，即一个数可以分解为多个质数的幂的乘积，那么当筛查范围很大时，可以建立一个长度为要筛查范围长度的数组，用下标代表数，用true和false代表是否为素数。从2开始剔除全部倍数，再将3的倍数剔除，5的倍数，7的倍数，依次类推。

## 【第三个实验】

1.实验源代码（循环求解）

**package arithmetic.factorial;**

**public class CyclicCalculator {**

**public static void main(String[] args) {**

**int max = 9;**

**int totalSum = 0;**

**while (max != 0) {**

**int factorial = 1;**

**for (int i = 1; i <= max; i++) {**

**factorial \*= i;**

**}**

**totalSum += factorial;**

**max--;**

**}**

**System.out.println("所求和为：" + totalSum);**

**}**

**}**

2.实验源代码（递归求解）

**package arithmetic.factorial;**

**public class RecursiveCalculator {**

**public static void main(String[] args) {**

**int totalSum = 0;**

**for (int i = 1; i <= 9; i++)**

**totalSum += getFactorial(i);**

**System.out.println(totalSum);**

**}**

**public static int getFactorial(int factorial) {**

**// 这里给出起始值**

**if (factorial <= 1) {**

**return (1);**

**} else {**

**// 实现F(N)\*F(N-1)\*...\*F(1)\*F(0)**

**return (getFactorial(factorial - 1) \* factorial);**

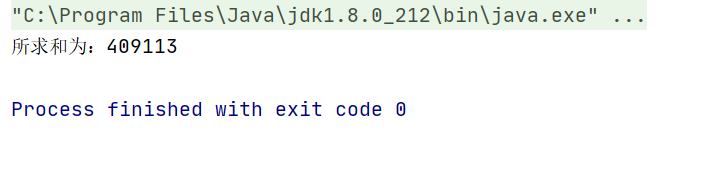
**}**

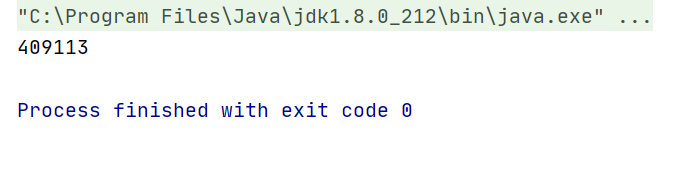
**}**

**}**

3.实验记录与说明分析

将程序运行，两种方法得到的结果是一模一样的，但在实际应用中最好不使用递归的方法：





## 【第四个实验】

1.Applet源代码

**package applet;**

**import javax.swing.\*;**

**import java.applet.Applet;**

**import java.awt.\*;**

**import java.awt.event.ActionEvent;**

**import java.awt.event.ActionListener;**

**public class AppletTemperature extends Applet implements ActionListener {**

**// 这一序列是IDE随机生成的**

**private static final long serialVersionUID = 1830598611881336880L;**

**TextField temperature1, temperature2;**

**Label lab1, lab2;**

**JButton start = new JButton("转换");**

**public void init() {**

**lab1 = new Label("摄氏温度：");**

**lab2 = new Label("华氏温度：");**

**temperature1 = new TextField(10);**

**temperature2 = new TextField(10);**

**add(lab1);**

**add(temperature1);**

**add(lab2);**

**add(temperature2);**

**temperature1.addActionListener(this);**

**temperature2.addActionListener(this);**

**// 给按钮加入监听后，点击按钮即会调用actionPerformed方法**

**start.addActionListener(this);**

**add(start);**

**}**

**public void actionPerformed(ActionEvent evt) {**

**// 这一方法时对ActionListener接口中同名方法的实现**

**if (temperature1.getText().equals("")) {**

**String fah = temperature2.getText();**

**Double fah1 = Double.valueOf(fah);**

**Double cel2 = (fah1 - 32) / 1.8;**

**float cel3 = (float) (Math.round(cel2 \* 100)) / 100;**

**temperature1.setText(cel3 + "°C");**

**} else {**

**String cel = temperature1.getText();**

**Double cel1 = Double.valueOf(cel);**

**Double fah2 = 1.8 \* cel1 + 32;**

**float fah3 = (float) (Math.round(fah2 \* 100)) / 100;**

**temperature2.setText(fah3 + "F");**

**}**

**}**

**}**

2.HTML源代码

**<title>Test</title>**

**<hr>**

**<applet code="applet.AppletTemperature.class" width=500 height=300>**

**</applet>**

**<hr>**

3.实验记录与说明分析

将程序运行，可得到如下结果，摄氏温度和华氏温度可以相互转换：



另外还可以发现，Applet程序十分的框架化，许多部分已经在接口中定义，只需将其实现即可完成相应功能。

## 【第五个实验】

1.实验源代码

**package electricitybill;**

**import javax.swing.\*;**

**public class ElectricityBill {**

**public static void main(String[] args) {**

**String strElectricity;**

**double bill = 0;**

**strElectricity = JOptionPane.showInputDialog("请输入您实际的用电量");**

**double electricity = Double.parseDouble(strElectricity);**

**if (electricity >= 0 && electricity <= 240) {**

**// 这里的bill若使用float类型，需要强制类型转换，因0.55是double类型**

**bill = electricity \* 0.55;**

**} else if (electricity > 240 && electricity <= 540) {**

**bill = 240 \* 0.55 + (electricity - 240) \* 0.7;**

**} else if (electricity > 540) {**

**bill = 240 \* 0.55 + (540 - 240) \* 0.7 + (electricity - 540) \* 0.9;**

**}**

**JOptionPane.showMessageDialog(null, "您需要缴纳电费为： " + String.format("%.2f", bill) + "元", "电费",**

**JOptionPane.PLAIN\_MESSAGE);**

**}**

**}**

2.实验记录与说明分析

JOptionPane类中有许多产生消息提示框的方法，每个方法可能会有重载，这里我们用到了showInputDialog和showMessageDialog，分别用来接收输入和显示输出。得到的程序运行结果如下：



可见程序能得到正常的结果

## 【思考题】

1.break label的用法？

答：break label是跳出当前的代码块，只能在循环中使用，要注意，跳出一层循环以后，其外层循环将进入下一次循环，将内层循环初始化后开始内层循环。

2.Integer与int有何区别？

答：Integer是int这一基本数据类型对应的引用类型，所有的基本数据类型都有它对应的引用类型，如下表：

|  |  |
| --- | --- |
| int（4字节） | Integer |
| byte（1字节） | Byte |
| short（2字节） | Short |
| long（8字节） | Long |
| float（4字节） | Float |
| double（8字节） | Double |
| char（2字节） | Character |
| boolean（未定） | Boolean |

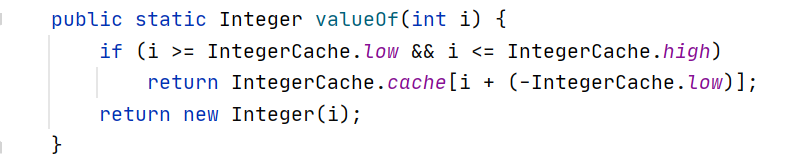
JavaSE5之前，使用引用时都需要进行手动的实例化操作，即：

Integer i = new Integer(10);

在之后的版本中，加入了自动装箱的功能，只需要这样写即可：

Integer i = 10;

自动装箱时，程序会调用Integer的valueOf(int)方法，在通过valueOf方法创建Integer对象的时候，如果数值在[-128,127]之间，便返回指向IntegerCache.cache中已经存在的对象的引用；否则创建一个新的Integer对象。查找一下，即可发现端倪：



下面是一段代码：

1. **public** **class** Main {
2. **public** **static** **void** main(String[] args) {
4. Integer i1 = 100;
5. Integer i2 = 100;
6. Integer i3 = 200;
7. Integer i4 = 200;
9. System.out.println(i1==i2);
10. System.out.println(i3==i4);
11. }
12. }

由上面的分析可知，这段代码的结果将会是true，false。

其他基本数据类型进行装箱时，由于其调用的函数不同，故类似操作的结果也会不同，只要在使用时注意其源码是如何实现的，即可不出差错。

另外，使用new建立基本类包是不需要也不建议的，从JavaSE9开始，基本类包的建构式都被标示为弃用（Deprecated）了。

3.字符串String在内存中的分配问题？

答：java中String有两种赋值方式：

1、String s = “I love Java";

2、String s = new String("I love Java");

对于第一种，java虚拟机为它分配的内存在常量池，即：会先在常量池中找是否已经有一块内存存的是“I love Java”，如果常量池中有就让栈中的 s 指向它；如果没有，则就在常量池里为它分配一块地方来存储“I love Java”。

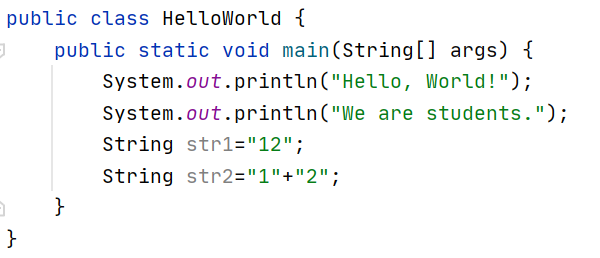
对于第二种，是在堆内存中分配了一块内存，然后用栈中的 s 指向它。（在堆中创建一个字符串对象，并把引用 s 放在栈中指向他的首地址）

要注意的是，下面这两种写法，其右边的值对编译器而言是等价的：

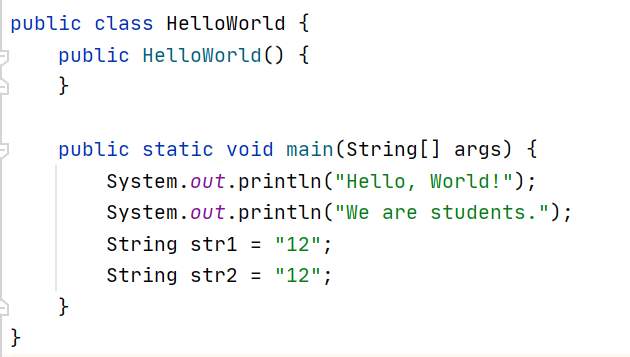
String str1=”a”+”b”;

String str2=”ab”;

这一点通过反编译.class文件即可窥见端倪，下面是源文件：



下面是反编译得到的代码：



4.为什么System.in可以作为函数的参数？

答：直接转到其声明，发现它是System下的一个静态变量，它的类型是抽象类InputStream类型，显然是可以调用作为参数的。但是这个成员被final修饰，意味着不可更改，经过查阅，其有一重定向的过程，会指向用户输入的字符串。



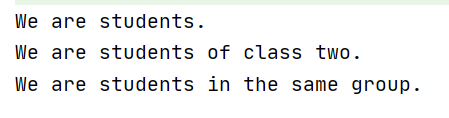
5.调用子类的构造函数的时候，父类的构造函数会发生什么？

答：派生类在实例化的过程中，要先调用其父类的构造函数，若父类还有父类，则要调用父类的父类的构造函数。再层层向下调用，这样说来还比较模糊，可以由下面这个程序一探究竟，其中的类名只是为了方便演示：

我们是学生->我们是二班的学生->我们是同一组的学生。

1. **public** **class** HelloWorld {
2. **public** **static** **void** main(String[] args) {
3. WeAreStudentsInTheSameGroup obj = **new** WeAreStudentsInTheSameGroup();
4. }
5. }
7. **class** WeAreStudents {
8. **public** WeAreStudents() {
9. System.out.println("We are students.");
10. }
11. }
13. **class** WeAreStudentsOfClassTwo **extends** WeAreStudents {
14. **public** WeAreStudentsOfClassTwo() {
15. System.out.println("We are students of class two.");
16. }
17. }
19. **class** WeAreStudentsInTheSameGroup **extends** WeAreStudentsOfClassTwo {
20. **public** WeAreStudentsInTheSameGroup() {
21. System.out.println("We are students in the same group.");
22. }
23. }

它的运行结果如下：



可见，先调用了最基本的类的构造函数，再逐级向下调用。

【实验心得】

经过本次实验，对java的类、方法、对象有了更深的认识，逐渐意识到了面向对象的含义，还直观的感受了了重写、继承等操作的过程与要求，如实验中对Applet类中paint方法的重写。

在实验过程中，还学习了一些IDE中快捷方便的操作，最重要的操作就是如何快速查看声明与引用，这一点对库函数的理解帮助巨大。虽然库函数完全可以被当做一个“黑匣子”去调用，不过这样永远触及不到它的本质，ctrl+鼠标左键，直接跳转到其声明与实现，则可以直观的看出它的来龙去脉，对Java的学习帮助巨大，JDK API也可以产生相似的作用。

另外，对class文件进行反编译，也是理解编译器在做什么事情的很好的方法，可以将一些“语法糖”背后的东西显化出来，理解其本质。

这次实验中还遇到了接口、抽象类、下转型、上转型等概念，比较难以理解，网络上的资料也参差不齐，希望在日后的学习中可以逐渐解决。