设计实现：带函数图形绘制的高级计算器

本设计实现基于面向对象的程序设计课程教授的内容，分功能分析与建模、核心流程设计分析、高级设计意图三个部分，在本学期通过三个阶段的实践，利用 Java 语言进行面向对象的程序设计，以练习和运用学习的面向对象的程序设计思想，加深对相关软件工程思想的理解，并实现一个通过面向对象编程的程序。

一、功能分析与建模

（1）需求模型

场景：用户使用计算器进行计算，按按键输入算式计算，或按按键输入表达式，绘制函数图像。

按照用例法，构建需求模型。

（1） 正常处理

【用例名称】

计算器计算和函数绘制

【场景】

任意场景

【用例描述】

1、 用户打开软件；

2、 用户选择计算或绘图模式；

3、 用户输入算式或含自变量的表达式；

4、 用户输入完成，程序显示计算结果或函数图像；

5、 计算完成，用户点击关闭退出程序。

【用例价值】

用户使用后得到了需要的计算结果或函数图像。

【约束和限制】

1、 输入数字的长度不超过 16 位；

2、 绘制的函数是初等函数。

（2） 异常处理

【用例描述】

1、 用户打开软件；

2、 用户选择计算或绘图模式；

3、 用户输入算式或含自变量的表达式；

3.1 用户输入错误，要删除一个字符；

3.2 用户想重新输入，要清空已经输入的字符。

4、 用户输入完成，程序显示计算结果或函数图像；

4.1 用户输入的算式或表达式不满足运算规则，要提示用户修改表达式。

5、 计算完成，用户点击关闭退出程序。

（3） 替代处理

【用例描述】
1、 用户打开软件；
2、 用户选择计算或绘图模式；
3、 用户输入算式或含自变量的表达式；
3-A 用户点击按钮输入；
3-B 用户通过键盘输入。
4、 用户输入完成，程序显示计算结果或函数图像；
4-A 用户点击“=”或“绘制”按钮完成；
4-B 用户通过键盘回车完成。
5、 计算完成，用户点击关闭退出程序。

综合上述分析，得到完整的用例

【用例名称】
计算器计算和函数绘制
【场景】
任意场景
【用例描述】
1、 用户打开软件；
2、 用户选择计算或绘图模式；
3、 用户输入算式或含自变量的表达式；
3.1 用户输入错误，要删除一个字符；
3.2 用户想重新输入，要清空已经输入的字符；
3-A 用户点击按钮输入；
3-B 用户通过键盘输入。
4、 用户输入完成，程序显示计算结果或函数图像；
4.1 用户输入的算式或表达式不满足运算规则，要提示用户修改表达式；
4-A 用户点击“=”或“绘制”按钮完成；
4-B 用户通过键盘回车完成。
5、 计算完成，用户点击关闭退出程序。
【用例价值】
用户使用后得到了需要的计算结果或函数图像。
【约束和限制】
1、 输入数字的长度不超过 16 位；
2、 绘制的函数是初等函数。

功能提取，得到功能矩阵

功能编号	功能描述	备注
001	选择模式	用户完成
002	按按钮输入	用户完成
003	键盘输入	用户完成
004	删除字符	用户完成
005	清空字符	用户完成

006	显示输入和结果	计算机完成
007	计算结果和函数图像的点	计算机完成
008	提示修改表达式	计算机完成

（2）抽象

①、抽取关键的类

使用用例法进行筛选，根据用例得到初选名词，进行审查和筛选

【初选名词列表】

计算器、函数、用户、软件（程序）、模式、算式（表达式）、字符、按钮、键盘、图像

【删除无用名词】

1）计算器：是需要实现的整体系统，不是具体的部分

2）函数：表达式的其中一种形式

3）用户：只是软件的使用者

4）软件（程序）：指设计本身

5）模式：是表达式的一个属性

5）字符：表达式和按钮的组成部分

6）键盘：物理设备，用于输入

【最终名词列表】

表达式、按钮、图像

②、添加类的属性

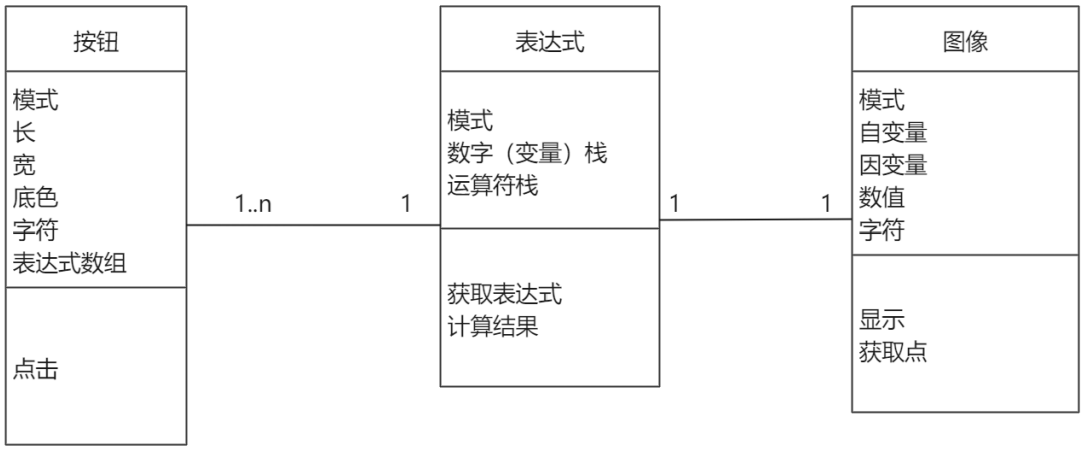
名词	**属性**	**备注**
表达式	模式、数字（变量）栈、运算符栈	根据计算机运算的方式，设置栈，利用逆波兰表达式实现计算器的功能
按钮	模式、长、宽、底色、字符、表达式数组	按钮形状设置为矩形，根据模式不同部分按钮的字符不同，表达式数组作为记录输入字符串的缓冲区。
图像	模式、自变量、因变量、数值、字符	该类综合对输入和算式结果以及函数图像的显示功能

③、添加类的方法

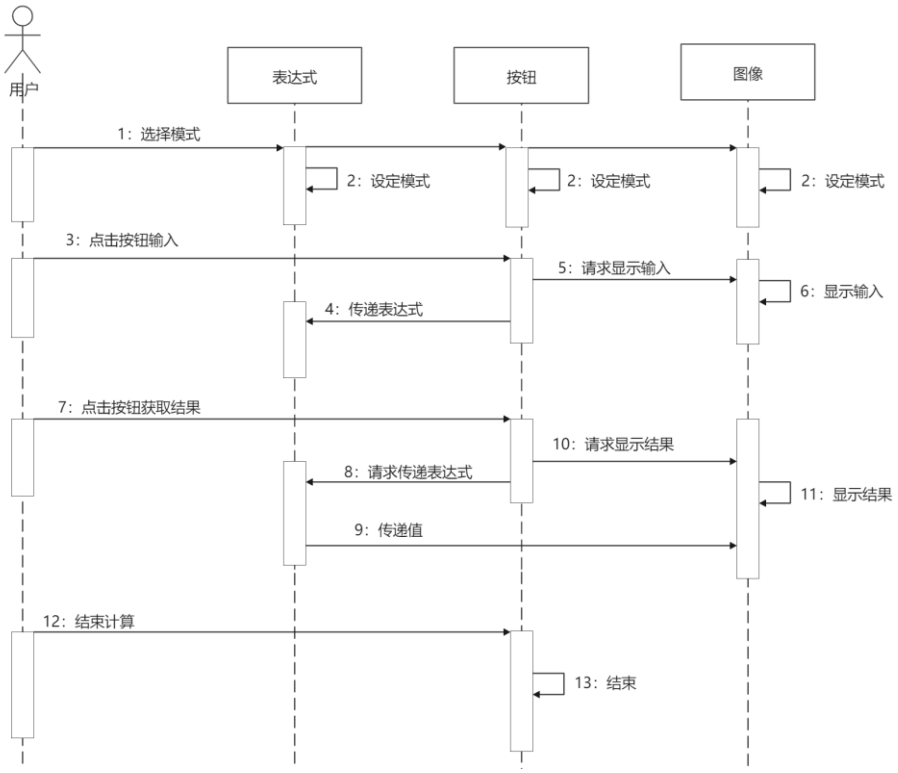
名词	**属性**	**方法**
表达式	模式、数字（变量）栈、运算	获取表达式、计算结果

	符栈	
按钮	模式、长、宽、底色、字符、表达式数组	点击
图像	模式、自变量、因变量、数值、字符	显示、获取点

④、得到初步的类模型



⑤、辅助模型——顺序图



二、核心流程设计分析

（1）用户界面

要实现用户与软件的交互，首先要有一个良好的交互界面。对于计算器，用命令行达不到理想的交互效果，一个可以点击的图形界面是必须的，这也是我们平时使用计算器的熟悉的形式，依靠图形界面的布局，逐步可以建立起整个功能的实现。

Java 为图形界面的设计提供了 JFrame 类，利用其提供的类的方法，可以实现对界面的设计布局，依据网络的资料[1]，首先构建起一个不具有具体计算功能的界面，代码如下：

```
package calculator;

import javax.swing.*;

public class Window extends JFrame {
    private String[] contents =
{"sin","cos","tan","(",")","ln","AC","←","%","÷","x^y","7","8","9","×","x!
","4","5","6","-","π","1","2","3","+","模式","e","0",".","="};
    private JButton buttons[] = new JButton[contents.length];
    private JTextField result = new JTextField("");
    private JLabel modeTxt = new JLabel("计算模式",JLabel.LEFT);
    public Window() {
        super("计算器");
        this.setLayout(null);
        result.setBounds(20,5,320,40);
        result.setHorizontalAlignment(JTextField.RIGHT);
        result.setEditable(false);
        modeTxt.setBounds(20,325,100,40);
        this.add(modeTxt);
        this.add(result);
        this.setResizable(false);
        this.setBounds(500,200,375,400);
        this.setDefaultCloseOperation(EXIT_ON_CLOSE);
        this.setVisible(true);
        int x = 20,y = 55;
        for(int i = 0;i < buttons.length;i++) {
            buttons[i] = new JButton();
            buttons[i].setText(contents[i]);
            buttons[i].setBounds(x,y,60,40);
            if(x < 280) {
                x += 65;
            }else {
                x = 20;
                y += 45;
```

```
            }
            this.add(buttons[i]);
            this.repaint();
        }
    }

}
```

这样一个与用户交互的界面就完成了，效果如下：

（2）事件响应

在初步建立起界面后，需要使按钮发挥作用，这样才能达到交互的效果。Java 提供了用于事件处理的接口 ActionListener。实现其中的 ActionPerformed 方法就可以在按钮被按下的时候做出规定的响应。由于界面中只有一类按钮，对于事件响应的处理也比较简单。在实现了对于事件的响应后，我们对于用户的输入也有了接收的途径，同时，切换模式的功能可以通过在响应事件后修改标签来完成了。代码如下：

```
public class Window extends JFrame implements ActionListener{
    public void actionPerformed(ActionEvent e) {
        String label = e.getActionCommand();
        if(label == "模式") {
            if(modeTxt.getText() == "计算模式") {
                modeTxt.setText("函数绘制模式");
                buttons[contents.length-1].setText("绘制");
                buttons[2].setText("x");
            }else {
```

```
                modeTxt.setText("计算模式");
                buttons[contents.length-1].setText("=");
                buttons[2].setText("tan");
            }
            result.setMode();
        }else {
            result.react(label);
        }
        this.repaint();
    }
}
```

此时点击模式按钮，就可以看到计算器的模式切换了：

（3）数据处理

①、设置缓冲区

用户输入信息后，需要在显示框中显示出来，并且能够满足用户对于输入的修改，这样一来，我们需要一个缓冲区来记录用户输入的信息。由于显示框的内容与缓冲区相关，我们将缓冲区设为继承自 JTextField 的子类，这样便于同步更新显示框的内容。

缓冲区我们使用一个 String 数组来记录输入，因为使用字符数组来操作便于更改缓冲区的内容，但对于“sin”这样的整体的运算符的操作（如增加或删除）会变得复杂，而把所有的输入连成一个 String，对于单个运算符或数字的操作又会复杂，利用 String 数组既能像字

符数组一样灵活处理每一个单元，又减少了处理像“sin”这样整体的运算符的难度。基于String 数组，我们在接收输入时把按钮的标签作为一个字符串加入数组，删除时减少一个数组元素即可。另外，缓冲区长度一定，但输入并不一定填满缓冲区，还要用一个指针指示当前最后一个输入元素的下标。代码如下：

```
public class Buffer extends JTextField implements Include{
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    public String[] inputBuffer = new String[maxSize];
    public int ptr;
    Buffer(String text){
        super(text);
        ptr = 0;
    }
    public void setBuffer(String input) {
        switch(input) {
            case "AC" :
                ptr = 0;
                break;
            case "←" :
                ptr--;
                break;
            case "x^y" :
                inputBuffer[ptr++] = "^";
                break;
            case "x!" :
                inputBuffer[ptr++] = "!";
                break;
            default:
                inputBuffer[ptr++] = input;
                if(input == "sin" || input == "cos" || input == "tan" ||
input == "ln") {
                    inputBuffer[ptr++] = "(";
                }
        }
    }
    public String[] getBuffer() {
        return inputBuffer;
    }
    public int getPtr() {
        return ptr;
    }
}
```

② 、显示缓冲区数据

每次输入或删除数据后，不仅要更新缓冲区的内容，同时要把更改后的缓冲区的内容显示在显示框中，为此，设置 Screen 类继承自 Buffer 类，每当用户点击按钮后，把信息传入 Screen，Screen 处理输入的数据，设置缓冲区，并及时设置显示框的标签更新显示框内容，代码如下：

```
public class Screen extends Buffer {
    /**
     *
     */
    private static final long serialVersionUID = 1L;
    Screen(String text){
        super(text);
    }
    public void setMode() {
        ptr = 0;
        refreshScreen();
    }
    private void refreshScreen() {
        String display = "";
        for(int i = 0;i < ptr;i++) {
            display = display + inputBuffer[i];
        }
        this.setText(display);
    }
    public void react(String input) {
        if(input == "=") {
            Calculator cal = new Calculator();
            double res = cal.calculate(inputBuffer, ptr);
            inputBuffer[0] = Double.toString(res);
            ptr = 1;
        }else if(input == "绘制") {
            DrawWindow draw = new DrawWindow(inputBuffer,ptr);
            draw.setVisible(true);
        }else {
            setBuffer(input);
        }
        refreshScreen();
    }
}
```

以上功能实现后，程序界面就能显示出用户输入的表达式了：

③ 、结果计算

整个计算器处理数据中最核心的部分即是结果计算，将输入的字符串变成计算机可以操作的数据处理，然后输出出来。我们设置一个 Calculator 类来计算结果。

```
public class Calculator implements Include{
    private double[] numStack = new double[maxSize];
    private String[] opStack = new String[maxSize];
    private String[] chaStack = new String[maxSize];
    private int numPtr,opPtr,chaPtr;
    Calculator(){
        numPtr = 0;
        opPtr = 0;
        chaPtr = 0;
    }
    public double calculate(String[] buffer,int ptr) {
    }
    private void refresh() {
        numPtr = 0;
        opPtr = 0;
        chaPtr = 0;
    }
}
```

Calculator 类里设置三个栈数组，分别用来记录数字和操作符以处理成后缀表达式，以及计算后缀表达式的数字栈。

当数据传入 Calculator 后，首先进行预处理，先把相邻的单个的数字连接成一个数据：

```
for(int i = 0;i < ptr;i++) {
            if(checkNum(buffer[i])) {
                if(i+1 < ptr && (checkNum(buffer[i+1]) || buffer[i+1] ==
".")) {
                    String num = buffer[i];
                    int end;
                    for(int j = i;;) {
                        if(j+1 < ptr && (checkNum(buffer[j+1]) ||
buffer[j+1] == ".")) {
                            num = num + buffer[j+1];
                            j++;
                        }else {
                            end = j;
                            break;
                        }
                    }
                    buffer[i] = num;
                    for(int j = i + 1;(j+end-i) < ptr;j++) {
                        buffer[j] = buffer[j+end-i];
                    }
                    ptr = ptr-(end-i);
                }
            }
}
```

然后给常量 e 和 π 赋值：

```
for(int i = 0;i < ptr;i++) {
            if(buffer[i] == "π") {
                buffer[i] = Double.toString(pi);
            }
            if(buffer[i] == "e") {
                buffer[i] = Double.toString(e);
            }
}
```

接下来处理单目运算符，利用递推计算出其操作数的值，然后计算被其作用后的值：

```
for(int i = 0;i < ptr;i++) {
            if(buffer[i] == "sin" || buffer[i] == "cos" || buffer[i] ==
"tan" || buffer[i] == "ln") {
                int end;
                int total = 1;
                for(int j = i + 2;;j++) {
                    if(buffer[j] == "(") {
```

```
                    total++;
                }else if(buffer[j] == ")") {
                    total--;
                }
                if(total == 0) {
                    end = j;
                    break;
                }
            }
            String[] buffer2 = new String[maxSize];
            for(int j = i + 1;j <= end;j++) {
                buffer2[j-i-1] = buffer[j];
            }
            double mid = calculate(buffer2,end-i);
            switch(buffer[i]) {
                case "sin" :
                    mid = Math.sin(mid);
                    break;
                case "cos" :
                    mid = Math.cos(mid);
                    break;
                case "tan" :
                    mid = Math.tan(mid);
                    break;
                case "ln" :
                    mid = Math.log(mid);
            }
            buffer[i]=Double.toString(mid);
            for(int j = i + 1;(j+end-i) < ptr;j++) {
                buffer[j] = buffer[j+end-i];
            }
            ptr = ptr-(end-i);
        }
    }
    for(int i = 0;i < ptr;i++) {
        if(buffer[i] == "!") {
            if(buffer[i-1] != ")") {
                double mid = Double.parseDouble(buffer[i-1]);
                mid = factorial(mid);
                buffer[i-1] = Double.toString(mid);
                for(int j = i;j < ptr-1;j++) {
                    buffer[j] = buffer[j+1];
                }
                ptr--;
```

```
                    i--;
                }else {
                    int begin;
                    int total = 1;
                    for(int j = i-2;;j--) {
                        if(buffer[j] == ")") {
                            total--;
                        }else if(buffer[j] == "(") {
                            total++;
                        }
                        if(total == 0) {
                            begin = j;
                            break;
                        }
                    }
                    String[] buffer2 = new String[maxSize];
                    for(int j = begin;j < i;j++) {
                        buffer2[j-begin] = buffer[j];
                    }
                    double mid = calculate(buffer2,i-begin);
                    mid = factorial(mid);
                    buffer[begin]=Double.toString(mid);
                    for(int j = begin + 1;(i+j-begin) < ptr;j++) {
                        buffer[j] = buffer[i+j-begin];
                    }
                    ptr = ptr-(i-begin+1);
                    i = begin;
                }
            }
}
```

然后对于可能出现在表达式开始或紧跟括号的负号进行处理，将其与紧跟的数值合并，以免在处理双目运算符时出现错误：

```
for(int i = 0;i < ptr;i++) {
            if(buffer[i] == "-") {
                if(i == 0 || buffer[i-1] == "(") {
                    if(Double.parseDouble(buffer[i+1]) >= 0) {
                        buffer[i] = buffer[i] + buffer[i+1];
                    }else {
                        buffer[i] = Double.toString(-
Double.parseDouble(buffer[i+1]));
                    }
                    for(int j = i + 1;j + 1 < ptr;j++) {
                        buffer[j] = buffer[j+1];
                    }
```

```
                    ptr--;
                }
            }
}
```

进行预处理后，表达式中只含完整相连的数据，六种双目运算符“+”“-”“*”“/”“%”“^”和“（”“）”了，此时把表达式转换为后缀表达式：

```
for(int i=0;i < ptr;i++){
            if(buffer[i] == "("){
                opStack[opPtr++] = buffer[i];
            }else if(buffer[i] == ")"){
                while (opStack[--opPtr] != "(")
                {
                    chaStack[chaPtr++] = opStack[opPtr];
                }
            }else if(buffer[i] == "^"){
                while (opPtr != 0 && opStack[opPtr-1] == "^")
                {
                 chaStack[chaPtr++] = opStack[--opPtr];
                }
                opStack[opPtr++] = buffer[i];
            }else if(buffer[i] == "*" || buffer[i] == "/" || buffer[i] ==
"%"){
                while (opPtr != 0 && (opStack[opPtr-1] == "*" ||
opStack[opPtr-1] == "/" || opStack[opPtr-1] == "%"))
                {
                 chaStack[chaPtr++] = opStack[--opPtr];
                }
                opStack[opPtr++] = buffer[i];
            }else if(buffer[i] == "+" || buffer[i] == "-"){
                while (opPtr != 0 && opStack[opPtr-1] != "(" &&
opStack[opPtr-1] != ")")
                {
                 chaStack[chaPtr++] = opStack[--opPtr];
                }
                opStack[opPtr++] = buffer[i];
            }else{
             chaStack[chaPtr++] = buffer[i];
            }
        }
        while (opPtr != 0){
        chaStack[chaPtr++] = opStack[--opPtr];
}
```

然后对后缀表达式进行计算，得到结果：

```
for(int i = 0;i < chaPtr;i++) {
```

```
        if(checkSym(chaStack[i])) {
            double op2 = numStack[--numPtr];
            double op1 = numStack[--numPtr];
            numStack[numPtr++] = operate(op1,op2,chaStack[i]);
        }else {
            numStack[numPtr++] = Double.parseDouble(chaStack[i]);
        }
}
```

（4）绘制图像

Java 为图形界面和绘图提供了充足的模板以供调用，利用 JFrame 提供的图形界面制作工具和诸如 Graphics，geom 等类来绘制图像。这里我们分别创建两个类 DrawGraph 和 DrawWindow 来处理图像绘制和显示窗口。在用户输入函数表达式后，通过点击绘制按钮把表达式传给 DrawWindow，DrawWindow 弹出绘制函数图像的窗口，并创建一个 DrawGraph 对象绘制函数图像。

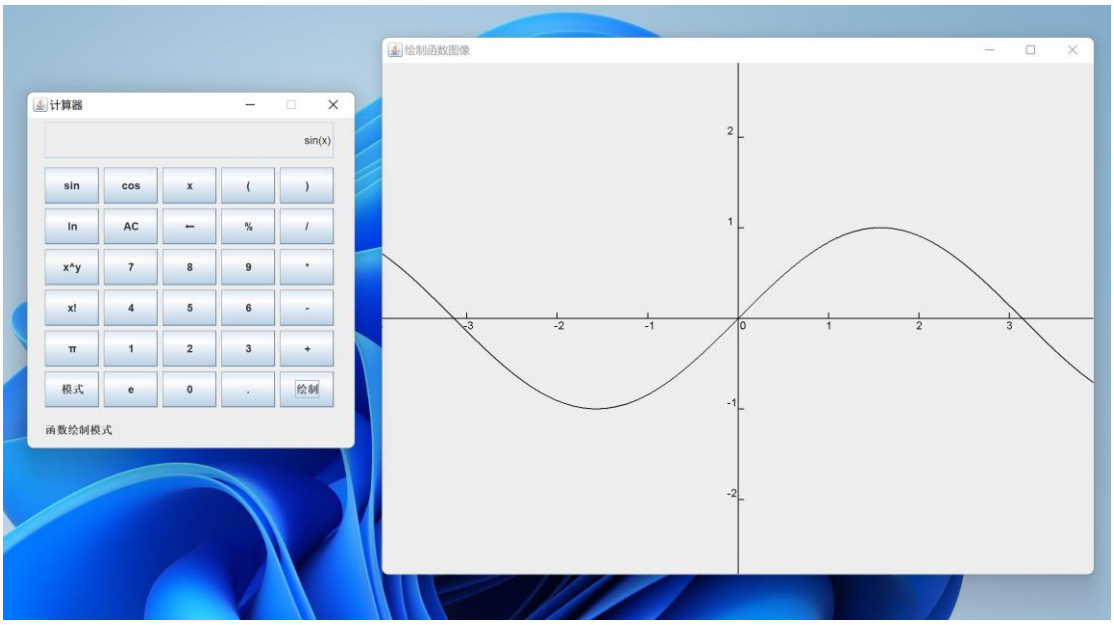
DrawGraph 在横坐标上每隔固定距离取一个 x 的值，将它替换表达式中的“x”，然后传给 Calculator 计算，得到纵坐标的值后在相应位置上画一个点，参考网络的资料[2]，我们利用相关工具类完成图像绘制，代码如下：

```
public class DrawWindow extends JFrame implements Include{
    private DrawGraph dg;
    private static final long serialVersionUID = 1L;
    DrawWindow(String[] buffer,int ptr){
        super("绘制函数图像");
        this.setDefaultCloseOperation(DISPOSE_ON_CLOSE);
        this.setSize(800,600);
        this.setLocationRelativeTo(null);
        dg = new DrawGraph(buffer,ptr);
        getContentPane().add(dg);
    }
}
public class DrawGraph extends JPanel implements Include{
    private int width,height,X,Y;
    private final int UnitLength = 100;
    private String[] expression;
    private int ptr;
    private static final long serialVersionUID = 1L;
    public DrawGraph(String[] expression,int ptr) {
        this.expression = expression;
        this.ptr = ptr;
    }
    public void paintComponent(Graphics g) {
        g.setColor(Color.WHITE);
        width = this.getWidth();
```

```
        height = this.getHeight();
        X = width/2;
        Y = height/2;
        this.drawAxes(g);
        this.drawFunc(g);
    }
    private void drawAxes(Graphics g) {
        g.setColor(Color.BLACK);
        g.drawLine(0,Y,width,Y);
        g.drawLine(X,0,X,height);
        g.drawString("0",X + 2,Y + 12);
        for(int i = 1;i*UnitLength < width;i++) {
            g.drawLine(X + i*UnitLength,Y - 1,X + i*UnitLength,Y - 6);
            g.drawLine(X - i*UnitLength,Y - 1,X - i*UnitLength,Y - 6);
            g.drawString(String.valueOf(i),X + i*UnitLength - 3,Y + 12);
            g.drawString(String.valueOf(i*(-1)),X - i*UnitLength - 3,Y +
12);
            g.drawLine(X + 1,Y + i*UnitLength,X + 6,Y + i*UnitLength);
            g.drawLine(X + 1,Y - i*UnitLength,X + 6,Y - i*UnitLength);
            g.drawString(String.valueOf(i),X - 12,Y - i*UnitLength - 3);
            g.drawString(String.valueOf(i*(-1)),X - 12,Y + i*UnitLength -
3);
        }
    }
    private void drawFunc(Graphics g1) {
        Calculator cal = new Calculator();
        Point2D temp1,temp2;
        double x,y;
        Graphics2D g = (Graphics2D)g1;
        g.setColor(Color.BLACK);
        x = (-1.0) * X / UnitLength;
        y = cal.calculate(replace(expression,ptr,Double.toString(x)), ptr);
        temp1 = new Point2D.Double(alterX(x * UnitLength),alterY(y *
UnitLength));
        for(int i = 0;i < width;i++) {
            x = x + 1.0/UnitLength;
            y = cal.calculate(replace(expression,ptr,Double.toString(x)),
ptr);
            if(Math.abs(y) < Y) {
                temp2 = new Point2D.Double(alterX(x * UnitLength),alterY(y *
UnitLength));
                g.draw(new Line2D.Double(temp1,temp2));
                temp1 = temp2;
            }
```

```
        }
    }
    private double alterX(double x) {
        return x + X;
    }
    private double alterY(double y) {
        return (y - Y) * (-1);
    }
}
```

至此，我们完成了计算器和函数绘制的功能实现，完整代码已上传至仓库，效果如下：

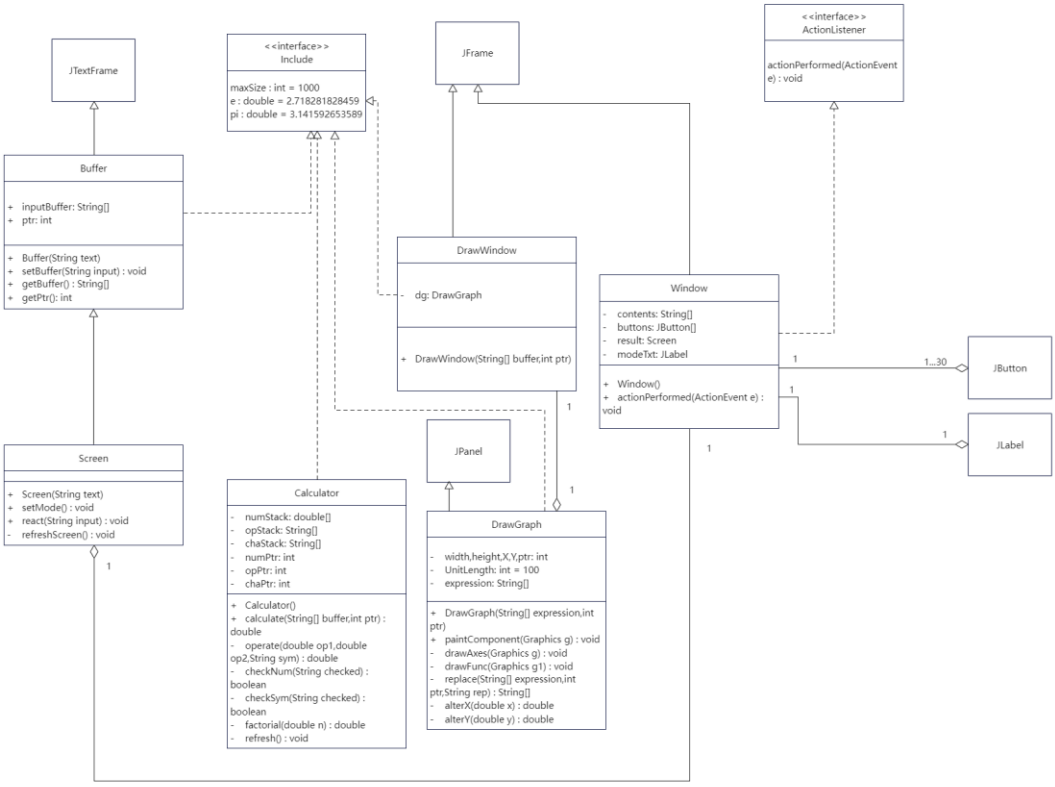


(5) 分析

通过对程序的具体实现，我们对第一部分构建的模型进行了调整，构建了 6 个类，Window 类显示计算器的界面，Buffer 类继承自 JTextField 并记录输入数据，Screen 类继承自 Buffer，真正作为显示框的处理工具，Calculator 类是计算器计算的核心，DrawWindow 和 DrawGraph 类分别用于构建函数窗口和绘制函数图像。

程序中实现了一个重要的接口 ActionListener 来完成图形界面的交互，此外，由于程序功能简单，仅构建了一个 Include 接口，用于实现类似头文件的功能。

完成程序的实现后，类模型如下：



参考资料

[1] 猫猫虫(一一). Java 语言编写计算器[OL]. [2021-11-27]. https://blog.csdn.net/qq_41398808/article/details/79558789

[2] lumaomao_. Java 画函数[OL]. [2021-11-28]. https://blog.csdn.net/lumaomao_/article/details/80112366