山东大学

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：实验五

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 11 月 25 日

实验报告

【实验目的】

1.熟练掌握 Java 程序的编写，掌握各个概念。

2.了解 Java 的实际应用。

3.在编写过程中学习了解相关内容。

【实验要求】

学习使用图形用户界面(GUI）设计实现一个简单的计算器，能进行加减乘除四则运算（注意数字大小的越界考虑，一般计算器输入的数字个数不超过 16 个)。

感兴趣的同学可以自由发挥，可以让你的计算器更加复杂，功能更加强大。

【实验源代码】

1.WesleyCalculator 类

```
import javax.swing.*;
import javax.swing.plaf.BorderUIResource;
import java.awt.*;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;

public class WesleyCalculator extends JFrame {
    private final Container container = this.getContentPane();
    private final JTextField textField = new JTextField();
    private final JMenu chooseMode = new JMenu("Standard");
    private final JMenuItem standard = new JMenuItem("Standard");
    private final JMenuItem alphaMode = new JMenuItem("Alpha Mode");
    private final JPanel cards = new JPanel();
    private final CardLayout cardLayout = new CardLayout();
    private final JPanel standardPanel = new JPanel();
    private final JPanel alphaPanel = new JPanel();
    private final AlphaAPI alphaAPI = new AlphaAPI();

    private WesleyCalculator() {
        //设置窗口基本属性和文本输入框
        setBasic();
        cards.setLayout(cardLayout);
        container.add(cards, BorderLayout.CENTER);
        //设置StandardPanel
        setStandardPanel();
        setAlphaPanel();
```

```
        cards.add("Standard", standardPanel);
        cards.add("Alpha", alphaPanel);
        cards.updateUI();
    }

    private void setBasic() {
        //设置窗体的标题
        setTitle("Wesley's Calculator");
        //设置窗体的显示位置及大小
        setBounds(200, 100, 340, 545);
        //设置窗体关闭按钮的动作为退出
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        BorderLayout calculatorLayout = new BorderLayout();
        calculatorLayout.setHgap(5);
        calculatorLayout.setVgap(5);
        container.setLayout(calculatorLayout);
        //加入菜单
        JMenuBar myMenuBar = new JMenuBar();
        myMenuBar.setBackground(Color.white);
        setJMenuBar(myMenuBar);
        chooseMode.setFont(new Font("Lao UI", Font.BOLD, 14));
        myMenuBar.add(chooseMode);
        standard.setBackground(Color.white);
        standard.setFont(new Font("Lao UI", Font.BOLD, 13));
        chooseMode.add(standard);
        alphaMode.setBackground(Color.white);
        alphaMode.setFont(new Font("Lao UI", Font.BOLD, 13));
        chooseMode.add(alphaMode);
        //加入一个输入框
        textField.setHorizontalAlignment(SwingConstants.TRAILING);
        textField.setBorder(new
BorderUIResource.LineBorderUIResource(Color.LIGHT_GRAY));
        textField.setFont(new Font("Lao UI", Font.BOLD, 28));
        textField.setPreferredSize(new Dimension(12, 115));
        container.add(textField, BorderLayout.NORTH);
        //菜单栏事件
        standard.addActionListener(actionEvent -> {
            textField.setText("");
            chooseMode.setText("Standard");
            cardLayout.show(cards, "Standard");
        });
        alphaMode.addActionListener(actionEvent -> {
            textField.setText("");
            chooseMode.setText("Alpha Mode");
```

```
            cardLayout.show(cards, "Alpha");
        });
    }

    private void setStandardPanel() {
        //标准模式下的Panel
        //按键的布局格式为网状
        GridLayout buttonLayout = new GridLayout(7, 5);
        buttonLayout.setHgap(4);
        buttonLayout.setVgap(3);
        standardPanel.setLayout(buttonLayout);
        //设置按键
        JButton[][] buttons = new JButton[7][5];
        String[][] buttonNames = {
                {"sin", "cos", "Del", "CE", "AC"},
                {"<html>&int", "mod", "|x|", "(", ")"},
                {"Dif", "<html>x<sup>y</sup>", "ln", "π", "e"},
                {"lim", "1", "2", "3", "+"},
                {"x", "4", "5", "6", "-"},
                {"y", "7", "8", "9", "×"},
                {"n!", ".", "0", "/", "="}
        };
        for (int row = 0; row < buttonNames.length; row++) {
            for (int column = 0; column < buttonNames[row].length; column++)
{
                buttons[row][column] = new JButton(buttonNames[row][column]);
                buttons[row][column].setFont(new Font("Lao UI", Font.BOLD,
15));
                buttons[row][column].setBackground(Color.white);
                buttons[row][column].setBorder(null);
                switch (buttonNames[row][column]) {
                    case "int":
                    case "sin":
                    case "cos":
                        setButton(buttons[row][column],
buttonNames[row][column], "()", 4);
                        break;
                    case "|x|":
                        setButton(buttons[row][column], "abs", "()", 4);
                        break;
                    case "<html>x<sup>y</sup>":
                        setButton(buttons[row][column], "^", "", 1);
                        break;
                    case "<html>&int":
```

```
                    setButton(buttons[row][column], "int", "()", 4);
                    break;
                case "Dif":
                    setButton(buttons[row][column], "d()/dx", "", 2);
                    break;
                case "lim":
                    setButton(buttons[row][column], "limit(,x->)", "", 6);
                    break;
                case "ln":
                    setButton(buttons[row][column],
buttonNames[row][column], "()", 3);
                    break;
                case "mod":
                    setButton(buttons[row][column], "%", "", 1);
                    break;
                case "n!":
                    setButton(buttons[row][column], "!", "", 1);
                    break;
                case "Del":
                    buttons[row][column].addActionListener(actionEvent -> {
                        textField.setText(textField.getText().substring(0,
textField.getText().length() - 1));

textField.setCaretPosition(textField.getText().length());
                        textField.requestFocus();
                    });
                    break;
                case "CE":
                    buttons[row][column].addActionListener(actionEvent -> {
                        if (textField.getText().contains("=")) {

textField.setText(textField.getText().substring(0,
textField.getText().indexOf("")));
                        }

textField.setCaretPosition(textField.getText().length());
                        textField.requestFocus();
                    });
                    break;
                case "AC":
                    buttons[row][column].addActionListener(actionEvent ->
textField.setText(""));

buttons[row][column].setBackground(Color.decode("#DB5845"));
```

```
                    buttons[row][column].setForeground(Color.white);
                    break;
                case "=":
                    buttons[row][column].addActionListener(actionEvent ->
getStandardResult());

buttons[row][column].setBackground(Color.decode("#4998D3"));
                    buttons[row][column].setForeground(Color.white);
                    break;
                default:
                    setButton(buttons[row][column],
buttonNames[row][column], "", 1);
                    break;
            }
            //将按钮添加到面板中
            standardPanel.add(buttons[row][column]);
        }
    }
    textField.addKeyListener(new KeyAdapter() {
        @Override
        public void keyReleased(KeyEvent e) {
            super.keyReleased(e);
            if (e.getKeyCode() == 10 &
chooseMode.getText().equals("Standard")) {
                getStandardResult();
            }
        }
    });
    standardPanel.updateUI();
}

private void setAlphaPanel() {
    //Alpha 模式下的Panel
    //布局格式为网状
    BorderLayout buttonLayout = new BorderLayout(5, 5);
    alphaPanel.setLayout(buttonLayout);
    alphaPanel.setBackground(Color.white);
    //设置提醒标签
    JLabel remind = new JLabel("       Get Whatever You Want");
    remind.setFont(new Font("Lao UI", Font.BOLD, 22));
    alphaPanel.add(remind, BorderLayout.NORTH);
    //设置结果显示区域
    JTextArea result = new JTextArea();
    //设置自动换行
```

```
        result.setLineWrap(true);
        JScrollPane scroll = new JScrollPane(result);

scroll.setHorizontalScrollBarPolicy(JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEE
DED);

scroll.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED)
;
        result.setBackground(Color.white);
        result.setEditable(false);
        result.setFont(new Font("Lao UI", Font.BOLD, 14));
        alphaPanel.add(scroll, BorderLayout.CENTER);
        //添加事件
        //文本框的回车监听
        textField.addKeyListener(new KeyAdapter() {
            @Override
            public void keyReleased(KeyEvent e) {
                super.keyReleased(e);
                if (e.getKeyCode() == 10 & chooseMode.getText().equals("Alpha
Mode")) {
                    result.setText("Please wait a minute...");
                    Thread queryThread = new Thread(() ->
result.setText(alphaAPI.fullResultQuery(textField.getText())));
                    queryThread.start();
                }
            }
        });
        alphaPanel.updateUI();
    }

    private void setButton(JButton button, String buttonName, String
displayForm, int moveCursorPosition) {
        //这里用于控制输入一个运算符或者数字后的展示形式及光标位置
        button.addActionListener(actionEvent -> {
            StringBuffer showExpression = new
StringBuffer(textField.getText());
            int cursorPosition = textField.getCaretPosition();
            showExpression.insert(cursorPosition, buttonName + displayForm);
            textField.setText(String.valueOf(showExpression));
            textField.setCaretPosition(cursorPosition + moveCursorPosition);
            textField.requestFocus();
        });
    }
```

```java
    private void getStandardResult() {
        Thread resultThread = new Thread(() -> {
            try {
                getLocalResult();
            } catch (Exception e) {
                try {
                    e.printStackTrace();
                    textField.setText(textField.getText() + "=" +
alphaAPI.fastQuery(textField.getText()));
                } catch (Exception exception) {
                    exception.printStackTrace();
                    textField.setText("Modify the expression or try Alpha
Mode");
                }
            }
            textField.setCaretPosition(textField.getText().length());
            textField.requestFocus();
        });
        resultThread.start();
    }

    private void getLocalResult() throws Exception {
        String strExpression = textField.getText().replace("×", "*");
        ReversePolishNotation notation = new
ReversePolishNotation(strExpression);
        String result = textField.getText() + "=" + notation.getResult();
        if (result.endsWith(".0")) {
            result = result.substring(0, result.length() - 2);
        }
        textField.setText(result);
        textField.requestFocus();
    }

    public static void main(String[] args) throws Exception {
        WesleyCalculator frame = new WesleyCalculator();
        frame.setResizable(false);
        frame.setVisible(true);
        UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
    }
}
```

2.ReversePolishNotation 类

```
import java.util.*;

class ReversePolishNotation {
    String[] splitString;
    private final Stack<Operator> operatorStack = new Stack<>();
    private final Queue<String> resultQueue = new LinkedList<>();

    ReversePolishNotation(String expression) throws Exception {
        preprocessing(expression);
        getReversePolish();
    }

    private void preprocessing(String expression) {
        //用于分割字符串的分隔符，加减乘除号及括号等符号都要视作分隔符
        String SEPARATOR = Symbol.OPERATOR + "|" + "[()]";
        //"#"为了防止最后有运算符在栈中未弹出
        String infixNotation = expression + "#";
        //将"-"前全部加上"0"，统一负号与减号
        infixNotation = infixNotation.replace("(-", "(0-");
        if (expression.charAt(0) == '-') {
            infixNotation = "0" + infixNotation;
        }
        splitString = infixNotation.split("(?<=" + SEPARATOR + ")|(?=" +
SEPARATOR + ")");
    }

    private void getReversePolish() throws Exception {
        //level 在后面用于控制^的结合性
        int level = 4;
        for (String element : splitString) {
            //将读到的字符串封装为类，并获取其类型
            Symbol symbol = new Symbol(element);
            String type = symbol.getType();
            //进行逆波兰转换
            switch (type) {
                //若为数，直接输出
                case "number":
                    resultQueue.offer(symbol.getContent());
                    break;
                //若为左括号，直接入栈，遇到右括号之前绝不出栈
                case "leftBracket":
                    Operator operator = new Operator(symbol.getContent());
                    operatorStack.push(operator);
```

```
                break;
            //遇到右括号时，弹出到左括号（左括号也要弹出）
            case "rightBracket":
                while (!operatorStack.peek().getName().equals("(")) {
                    resultQueue.offer(operatorStack.pop().getName());
                }
                operatorStack.pop();
                break;
            //当遇到运算符时，要分一些情况讨论
            case "operator":
                operator = new Operator(symbol.getContent());
                //这里用于控制"^"的结合性，靠右的"^"优先级更高
                if (operator.getName().equals("^")) {
                    operator.setPriority(level);
                    level++;
                }
                if (!operatorStack.empty()) {
                    //算符栈中要保证从下到上优先级递减（且不可有相等）
                    if (operatorStack.peek().getPriority() >=
operator.getPriority()) {
                        //这里使用了短路与，只通过左边即可得出结论时不再判断右边，避免
了空栈异常
                        while (!operatorStack.empty() &&
operatorStack.peek().getPriority() >= operator.getPriority()) {

resultQueue.offer(operatorStack.pop().getName());
                        }
                    }
                    operatorStack.push(operator);
                } else {
                    //栈为空时直接入栈
                    operatorStack.push(operator);
                }
                break;
            default:
                throw new Exception();
            }
        }
        System.out.println(resultQueue);
    }

    public double getResult() throws Exception {
        Stack<Double> numberStack = new Stack<>();
        while (!resultQueue.isEmpty()) {
```

```
            Symbol element = new Symbol(resultQueue.poll());
            switch (element.getType()) {
                case "number":
                    switch (element.getContent()) {
                        case "π":
                            numberStack.push(Math.PI);
                            break;
                        case "e":
                            numberStack.push(Math.E);
                            break;
                        default:

numberStack.push(Double.valueOf(element.getContent()));
                            break;
                    }
                    break;
                case "operator":
                    double firstNum = numberStack.pop();
                    switch (element.getContent()) {
                        case "+":
                            numberStack.push(firstNum + numberStack.pop());
                            break;
                        case "-":
                            numberStack.push(-firstNum + numberStack.pop());
                            break;
                        case "*":
                            numberStack.push(firstNum * numberStack.pop());
                            break;
                        case "/":
                            numberStack.push(numberStack.pop() / firstNum);
                            break;
                        case "%":
                            numberStack.push(numberStack.pop() % firstNum);
                            break;
                        case "^":
                            numberStack.push(Math.pow(numberStack.pop(),
firstNum));
                            break;
                        case "abs":
                            numberStack.push(Math.abs(firstNum));
                            break;
                        case "sin":
                            numberStack.push(Math.sin(firstNum));
                            break;
```

```
                    case "cos":
                        numberStack.push(Math.cos(firstNum));
                        break;
                    case "ln":
                        numberStack.push(Math.log(firstNum));
                        break;
                    case "!":
                        numberStack.push(calculateFactorial(firstNum));
                        break;
                    default:
                        break;
                }
                break;
            default:
                throw new Exception();
        }
    }
    //最后仍在栈中的即为结果
    return numberStack.peek();
}

private double calculateFactorial(double number) {
    double result = 1;
    for (int i = 1; i <= number; i++) {
        result *= i;
    }
    return result;
}
}
```

3.Symbol 类

```java
class Symbol {
    private String content;
    private String type;
    public static final String FUNCTION = "^sin$|^cos$|^abs$|^ln$";
    public static final String NUMBER = "\\d\\d*\\.?\\d*|[e]|[π]";
    public static final String OPERATOR = "[+\\-*/%^!#]";

    Symbol(String content) throws Exception {
        this.content = content;
        setType();
    }

    public void setContent(String content) throws Exception {
        this.content = content;
        setType();
    }

    public String getContent() {
        return this.content;
    }

    private void setType() throws Exception {
        if (content.matches(NUMBER)) {
            this.type = "number";
            return;
        }
        if (content.matches(OPERATOR)) {
            this.type = "operator";
            return;
        }
        if (content.matches(FUNCTION)) {
            this.type = "operator";
            return;
        }
        String LEFT = "\\(";
        if (content.matches(LEFT)) {
            this.type = "leftBracket";
            return;
        }
        String RIGHT = "\\)";
        if (content.matches(RIGHT)) {
            this.type = "rightBracket";
            return;
```

```
        }
        //均不匹配则应当抛出异常
        throw new Exception();
    }

    public String getType() {
        return this.type;
    }
}
```

4.Operator 类

```java
class Operator {
    private int priority = 0;
    private final String name;

    public int getPriority() {
        return priority;
    }

    public String getName() {
        return name;
    }

    Operator(String operator) throws Exception {
        this.name = operator;
        setPriority();
    }

    public void setPriority(int priority) {
        this.priority = priority;
    }

    private void setPriority() throws Exception {
        switch (this.name) {
            //左括号遇到右括号前要一直留在栈里，故优先级设为最低
            case "(":
            case ")":
                this.priority = 0;
                break;
            //'#'用于标记字符串结尾，防止最后有运算符在栈内未弹出
            case "#":
                this.priority = 1;
                break;
            case "+":
            case "-":
                this.priority = 2;
                break;
            case "*":
            case "/":
            case "%":
                this.priority = 3;
                break;
            case "^":
                this.priority = 4;
```

```
                break;
            //函数可以被看作单目运算符
            case "abs":
            case "sin":
            case "ln":
            case "cos":
            case "!":
                this.priority = 2147483646;
                break;
            default:
                throw new Exception();
        }
    }
}
```

5.AlphaAPI 类

```
import com.wolfram.alpha.*;

import java.io.BufferedReader;
import java.io.InputStreamReader;
import java.net.URLEncoder;

class AlphaAPI {
    // PUT YOUR APPID HERE:
    private static final String APPID = "YUVLLV-32YX2JJVVQ";

    public String fastQuery(String input) throws Exception {
        String query;
        StringBuffer result = new StringBuffer();
        query = "http://api.wolframalpha.com/v1/result?appid=" + APPID +
"&i=" + URLEncoder.encode(input, "UTF-8");
        System.out.println("Fast Query URL is: " + query);
        java.net.URL url = new java.net.URL(query);
        BufferedReader read = new BufferedReader(new
InputStreamReader(url.openStream()));
        result.append(read.readLine());
        System.out.println("Get it! Result is: " + result.toString());
        read.close();
        return String.valueOf(result);
    }

    public String fullResultQuery(String input) {
        StringBuffer result = new StringBuffer();
        // Use "pi" as the default query, or caller can supply it as the
lone command-line argument.
        // String input = "ln_2(3)";
        // The WAEngine is a factory for creating WAQuery objects,
        // and it also used to perform those queries. You can set properties
of
        // the WAEngine (such as the desired API output format types) that
will
        // be inherited by all WAQuery objects created from it. Most
applications
        // will only need to crete one WAEngine object, which is used
throughout
        // the life of the application.
        WAEngine engine = new WAEngine();
        // These properties will be set in all the WAQuery objects created
from this WAEngine.
```

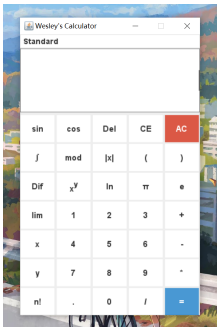
```
        engine.setAppID(APPID);
        engine.addFormat("plaintext");
        // Create the query.
        WAQuery query = engine.createQuery();
        // Set properties of the query.
        query.setInput(input);
        try {
            // For educational purposes, print out the URL we are about to
send:
            System.out.println("Query URL:");
            System.out.println(engine.toURL(query));
            System.out.println();
            // This sends the URL to the Wolfram|Alpha server, gets the XML
result
            // and parses it into an object hierarchy held by the
WAQueryResult object.
            WAQueryResult queryResult = engine.performQuery(query);
            if (queryResult.isError()) {
                result.append("Query error" + "\n");
                result.append("  error code: " + queryResult.getErrorCode() +
"\n");
                result.append("  error message: " +
queryResult.getErrorMessage() + "\n");
            } else if (!queryResult.isSuccess()) {
                result.append("Query was not understood; no results
available.");
            } else {
                // Got a result.
                result.append("Successful query. Pods follow:\n");
                for (WAPod pod : queryResult.getPods()) {
                    if (!pod.isError()) {
                        result.append(pod.getTitle() + "\n");
                        result.append("------------" + "\n");
                        for (WASubpod subpod : pod.getSubpods()) {
                            for (Object element : subpod.getContents()) {
                                if (element instanceof WAPlainText) {
                                    result.append(((WAPlainText)
element).getText() + "\n");
                                }
                            }
                        }
                        result.append("\n");
                    }
                }
```

```
                // We ignored many other types of Wolfram|Alpha output, such
as warnings, assumptions, etc.
                // These can be obtained by methods of WAQueryResult or
objects deeper in the hierarchy.
            }
        } catch (WAException e) {
            e.printStackTrace();
        }
        return String.valueOf(result);
    }
}
```

【概述】

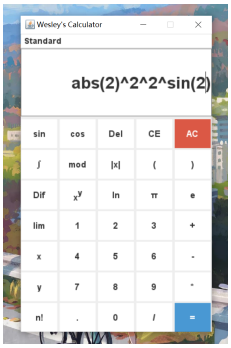
一、标准模式的演示说明

如图所示，打开计算器后将进入一个这样的界面，左上角提示当前的模式为”Standard”，表示可以进行科学计算及微积分运算（还包含一些彩蛋），点击它还可切换模式。

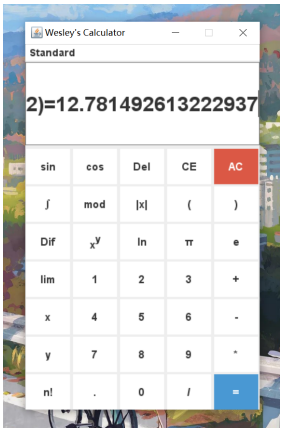


输入时可以使用键盘输入，也可使用按钮输入，按钮输入时考虑了光标位置（如插入，带括号运算符自动移动光标至括号内）。

下面演示基本的科学运算，如计算$|x|^{2^{2^{sin\,(2)}}}$

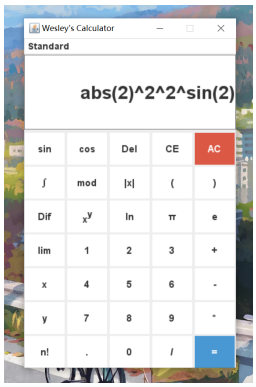


输入后，可以按回车或者等于号得到结果：



可以验证，该结果是正确的。

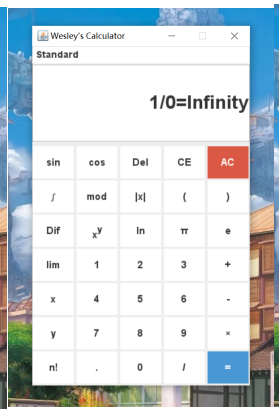
这时如果想要保留算式，稍作修改继续运算，则可以按下 CE 键，只会清除等号后的内容，如下所示，又得到了原来的算式，可继续运算：

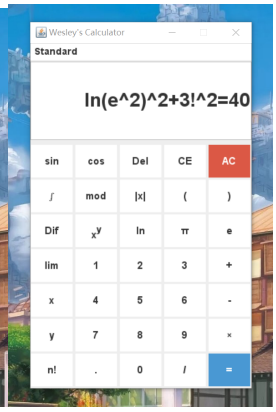


若按下 AC 键，将会把输入显示框内的内容全部清除，用户可以重新输入。

按下 Del 键，则会清除输入显示框最右边的内容，若想清除指定位置的字符，可以使用键盘上的 Backspace 或 Delete。

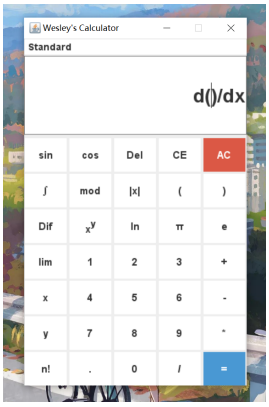
下面演示一些其他计算，可以验证这些结果都是正确的。

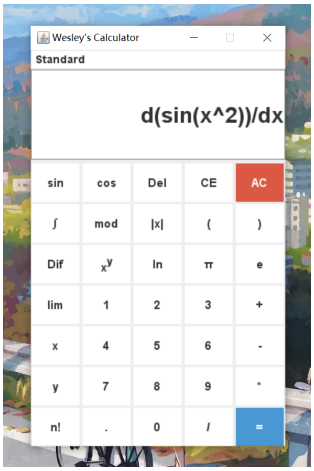




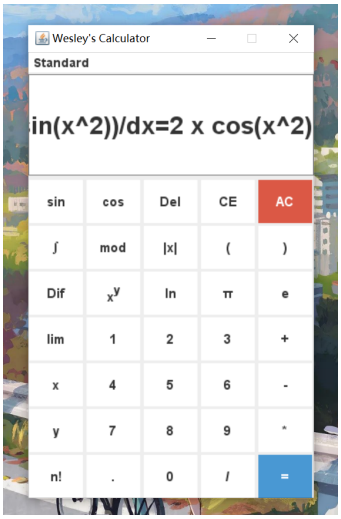
可见，能想像到的各种类型的科学运算，该计算器都是可以处理的。

下面演示微积分功能，如图所示，若要计算微分，则可使用 Dif 键，按下 Dif 有如下效果，可见，插入了模板，光标移动到了括号内。

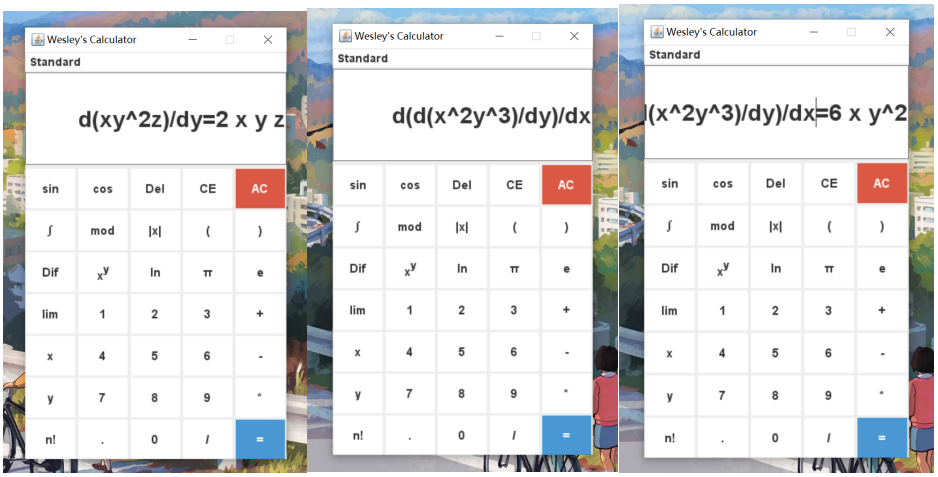




计算一个式子，如$\frac{dsin(x^2)}{dx}$，可以验证，该结果是正确的。



我们还可以计算偏微分或高阶微分，如下所示

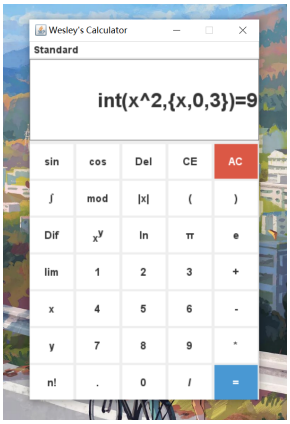


这时分母或表达式需要用户自己通过键盘来改动，功能十分丰富，不再过多演示。

该计算器也可定积分或不定积分，按下“∫”键，得到如下式子，可以进行填补

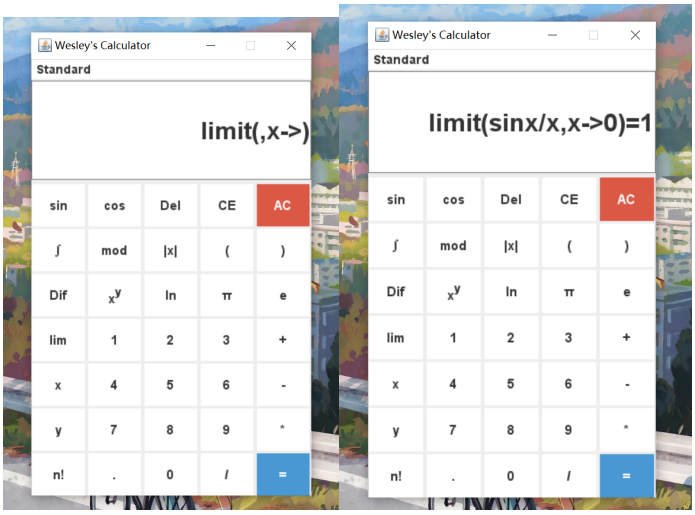
如计算x^2的不定积分，可得到如下结果

计算定积分$\int_0^3 x^2 dx$，得到如下结果：



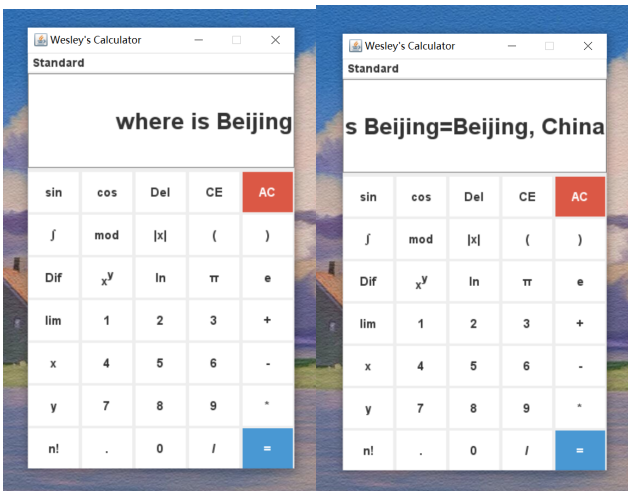
此外，该计算器还可以实现取极限，按下 lim 得到如下表达式，可进行计算，如计算：

$$\lim_{x \to 0} \frac{sin(x)}{x}$$

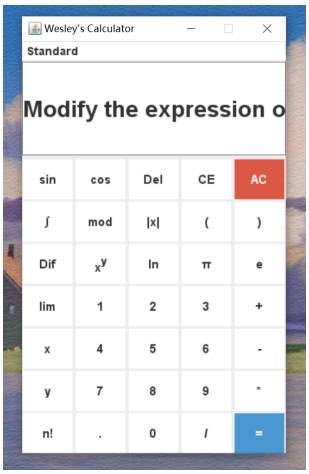


可见得到了正确结果。

此外，程序内还有一些小彩蛋，如进行一些简单的问答：



最后，当计算或者得不到结果时，将会提示用户更改表达式或者使用 Alpha Mode：

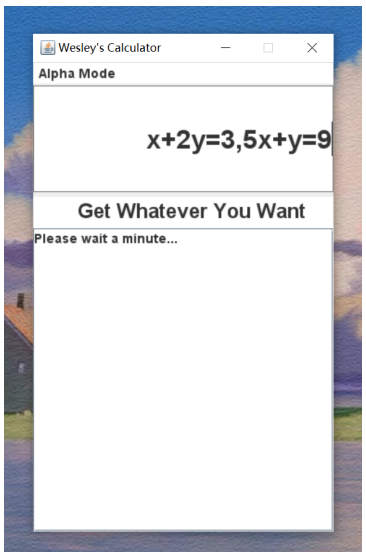


另外，该计算器还有许多未知功能有待探索。

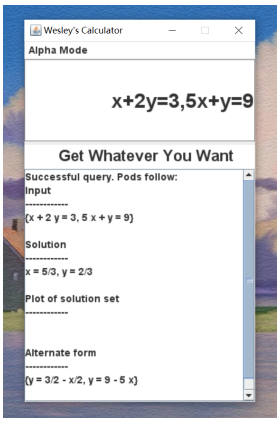
二、Alpha 模式的演示

Alpha Mode 通过调用 WolframAlpha 的 API，从网页中爬取所需结果，展示给用户。

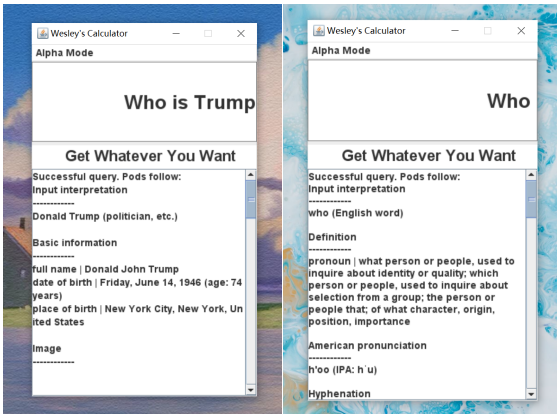
输入一个表达式按下回车后，程序会创建一个线程去抓取结果，这时会显示



得到的结果如下：



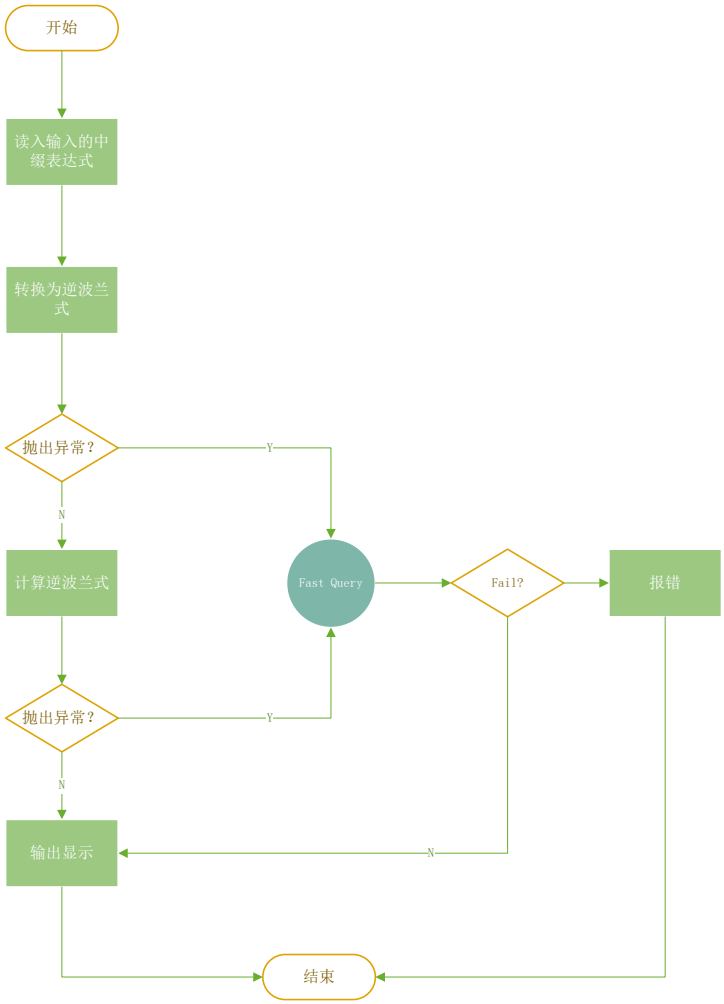
WolframAlpha 不仅仅是一个数学引擎，它还是一个知识引擎，在这里你可以获取任何你想知道的事情，用户可以尽情探索：



【原理介绍与代码实现】

【Standard Mode】

Standard 模式的功能其实已经十分强大了，具体的实现是途径可以由下面的流程图解释：



用代码来表示是这样的：

```
private void getStandardResult() {
    Thread resultThread = new Thread(() -> {
        try {
            getLocalResult();
        } catch (Exception e) {
            try {
                e.printStackTrace();
                textField.setText(textField.getText() + "=" + alphaAPI.fastQuery(textField.getText()));
            } catch (Exception exception) {
```

```
                exception.printStackTrace();
                textField.setText("Modify the expression or try Alpha Mode");
            }
        }
        textField.setCaretPosition(textField.getText().length());
        textField.requestFocus();
    });
    resultThread.start();
}
```

可见，当按下回车，计算器创建一个线程进行结果计算，计算器遇到一个式子，先会尝试调用 getLocalResult()方法从本地获取计算结果，如果失败则会调用 WolframAlpha 的 Simple Results API 从云端获取结果，如果两种途径都失败，则会报错。

其中，如流程图所示，本程序的最终版本里 getLocalResult()是通过逆波兰表达式获取结果的，可以计算各种函数，如 abs，sin，cos，ln，阶乘等（其实函数和阶乘可以看作单目运算符），可以识别数学常数，如 e，π，还通过优先级的控制实现了幂运算符^。

其实，在使用这种方式之前走了许多弯路，起初为方便使用了 JS Engine 中的 eval 方法，这样实现起来很简单，但无法计算幂运算符（至少 JAVA 8 SE 中的 JavaScriptEngine 还没有支持幂运算符），后来探索使用正则表达式进行匹配，先寻找幂运算符前面的第一个操作数，再寻找后面的操作数，循环往复的寻找匹配，这样一来实现了较为复杂的计算，甚至可以算 abs(2)^2^(1+2)这样的式子，但许多加括号的情况（比如 2^(1+(2))）仍不能计算，最后便思考，计算机到底是如何做这个工作的呢？便发现原来在数据结构中有对逆波兰式的处理，便可实现任何运算。

★逆波兰表达式简介

下面简要介绍一下中缀表达式转为逆波兰表达式的过程及逆波兰表达式的计算方法。

一、将中缀表达式转换成后缀表达式算法：

1.从左至右扫描一中缀表达式。

2.若读取的是操作数，则判断该操作数的类型，并将该操作数存入输出队列

3.若读取的是运算符（函数认为是单目运算符），分以下几种情况：

（1）若该运算符为左括号"("，则直接存入运算符栈。遇到右括号之前绝不弹出（程序

中可以通过设置其优先级最低实现）。

（2）若该运算符为右括号")"，则输出运算符栈中的运算符到输出队列，直到遇到左括号为止。

（3）该运算符为非括号运算符：

若比运算符栈栈顶的运算符优先级高，则直接存入运算符栈。

若比运算符栈栈顶的运算符优先级低或相等，则不断输出栈顶运算符到输出队列，直到栈顶没有运算符的优先级大于或者等于当前运算符（即栈顶存在运算符的话，优先级一定是小于当前运算符），最后将当前运算符压入运算符栈。

4.当表达式读取完成后运算符栈中尚有运算符时，则依序取出运算符到输出队列，直到运算符栈为空。（这一步可以通过在表达式最后加一优先级最低的特殊字符”#”，实现只把”#”留在栈内，其余输出）。

二、逆波兰表达式求值算法：

1.从左到右依次扫描逆波兰式中语法单元的项目。

2.如果扫描的项目是操作数，则将其压入操作数堆栈，并扫描下一个项目。

3.如果扫描的项目是一个二元运算符，则对栈的顶上两个操作数执行该运算。

4.如果扫描的项目是一个一元运算符，则对栈的最顶上操作数执行该运算。

5.将运算结果重新压入堆栈。

6.重复步骤 2-5，堆栈中即为结果值。

以上是整个转换与求解过程，值得一提的是这种求解没有考虑幂运算符从右向左的结合性，该如何实现呢？只需要在扫描过程中，每遇到一个”^”都将其优先级增 1 即可，这样可以保证靠右的”^”先进行运算，但要注意”^”优先级不能高于本高于幂运算符的操作符。

★逆波兰式转换与计算程序实现简述

整个程序包含五个类：

AlphaAPI 用来接入 WolframAlpha 的 API。

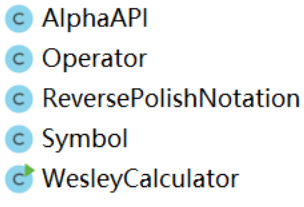
Operator 类用来形成一个个操作符对象，包含 name，priority 两个属性，包含 getter 和 setter，方便波兰表达式转换求解。

ReversePolishNotation 类用于转换、计算中缀表达式。

Symbol 类用来在分割字符串后形成一个个字符对象，包含 content，type 两个属

性，包含 getter 和 setter。举例来说，如”1.2”封装为 Symbol 对象，其 content 为“1.2”，type 为”number”。

最后一个类则是本次实验 GUI 界面实现的类。



首先，要将表达式进行预处理，包括将负号前加”0-”，在表达式末尾添加”#”，对表达式进行分割。分割这一步骤使用 String 类中的 split 方法配合正则表达式的零宽断言，可以轻松实现。

ReversePolishNotation 类中 preprocessing()方法代码如下：

```
private void preprocessing(String expression) {
    //"#"为了防止最后有运算符在栈中未弹出
    String infixNotation = expression + "#";
    //将"-"前全部加上"0"，统一负号与减号
    infixNotation = infixNotation.replace("(-", "(0-");
    if (expression.charAt(0) == '-') {
        infixNotation = "0" + infixNotation;
    }
    splitString = infixNotation.split("(?<=" + SEPARATOR + ")|(?=" + SEPARATOR + ")"
);
}
```

其中 SEPARATOR = "[+\\-*/%^()#]"。

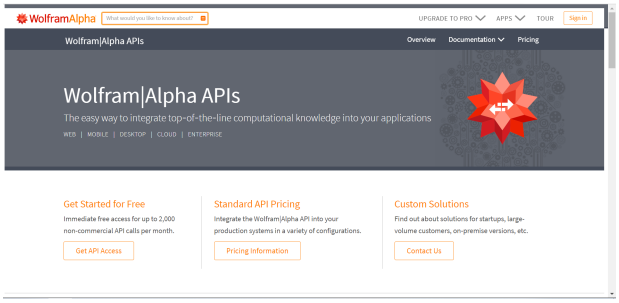
处理和计算逆波兰表达式的程序较长，其处理过程与原理中所述完全相同。这里不再贴出，可直接参见源代码。

Standard 模式中的微积分和问答功能由 WolframAlpha 的 API 实现，将在下面 Alpha Mode 部分集中介绍。

【Alpha Mode】

下面来介绍一下 WolframAlpha 的 API。

WolframAlpha 官网提供了非常丰富的 API 可供使用，且有许多 API 是免费的。



本次实验主要使用了 Short Answers API 和 Full Result API，本实验中 Standard 模式中微积分功能和简短问答即是由 Short Answers API 得到的，而 Alpha 模式则是完全基于 Full Result API，读取 query 得到的 XML 文件中的内容并分类显示出来，这一过程的实现还是比较复杂的，好在 WolframAlpha 官网给出了 Demo 程序。

除去 Demo 中的内容，为了获取”SimpleQuery”的结果，可以使用 URL 类进行网页访问，使用 BufferedReader 读入 URL 输入流。要注意的是，输入 URL 地址中的待查询字符串需要使用 URLEncoder.encode 这一方法解析，如下所示：

```
URLEncoder.encode(input, "UTF-8")
```

具体代码如下：

```
1. public String fastQuery(String input) throws Exception {
2.     String query;
3.     StringBuffer result = new StringBuffer();
4.     query = "http://api.wolframalpha.com/v1/result?appid=" + APPID + "&i=" + URLEnco
   der.encode(input, "UTF-8");
5.     System.out.println("Fast Query URL is: " + query);
6.     java.net.URL url = new java.net.URL(query);
7.     BufferedReader read = new BufferedReader(new InputStreamReader(url.openStream())
   );
8.     result.append(read.readLine());
9.     System.out.println("Get it! Result is: " + result.toString());
```

```
    read.close();
    return String.valueOf(result);
}
```

WolframAlpha API 的其他相关内容可在 Wolfram|Alpha APIs: Computational Knowledge Integration 这一网页中查看。在此不再赘述。

【GUI 界面搭建】

GUI 的 container 使用的是 BorderLayout 布局，放置一个输入文本框在 NORTH 处，一个卡片布局在 CENTER 处，用户点击左上角的菜单可以实现在两张卡片间切换。GUI 界面的外观设计尽可能地做到了美观、人性化（界面长宽比基本符合黄金比例）。功能实现方面，有几个比较有趣的点值得一说。

1.数组索引控件

对于计算器这样的界面来说，这一点是再方便不过了，可以使用一个 JButton 类型的对象数组，批量设置按键：

```
JButton[][] buttons = new JButton[7][5];
```

2.在 Button 上显示带格式的文字

这一点可以通过 HTML 文本实现，如显示分式可以使用：

```
<html>x<sup>y</sup>
```

其效果如下：

x^y

3.控制光标位置与获取焦点

在实际使用时，人们很难做到一遍输入正确，这时要实现在光标位置插入文本，则要使用 getCaretPosition()方法和 requestFocus()方法，本程序中实现了可使用界面上的按钮在光标位置插入所需式子，函数按钮（如 abs）按下后自动加括号，且光标移动到括号内，输入式子后不丢失光标等，为了方便实现这些，可以写一个函数：

```
private void setButton(JButton button, String buttonName, String displayForm, int moveCursorPosition) {
    //这里用于控制输入一个运算符或者数字后的展示形式及光标位置
    button.addActionListener(actionEvent -> {
        StringBuffer showExpression = new StringBuffer(textField.getText());
        int cursorPosition = textField.getCaretPosition();
        showExpression.insert(cursorPosition, buttonName + bracketForm);
        textField.setText(String.valueOf(showExpression));
        textField.setCaretPosition(cursorPosition + moveCursorPosition);
        textField.requestFocus();
    });
}
```

第一个参数是要设置的按钮，第二个参数是按下按钮后将在文本框内显示的内容，第三个参数则是按下按钮后光标将位移多少，控制这三个参数，可以方便实现上

述功能。

4.键盘监听的加入

本实验通过实现 KeyAdapter 抽象类中的 keyReleased()方法，给 textField 加入了按键监听，使得用户按下回车可以获取结果。

5.多线程的使用

多线程的使用在本实验中是很有必要的，因为在获取结果的过程中，若不另开线程，程序将会“卡死”一段时间。使用 Thread 类实现多线程，可以避免这个问题。

【实验心得】

本次实验过后，可谓大有所获，实验中涉及许多陌生的东西，也加深了对曾经所学知识的理解。多线程、正则表达式、逆波兰式、对 API 的调用、URL 内容的获取，大大开拓了自己的眼界，下面做一些总结。

逆波兰表达式的转换、计算是本实验中最令人感到神奇的部分，实现它其实并不困难，尤其是 JAVA 中已经有了现有的栈、队列等，可以直接使用。但要将幂运算、单目表达式运算加入指令集，且实现后保证可扩展性（即日后可随意加入函数等）是需要一些考虑的。

本实验中将字符串分割后，把所有 symbol 封装为了对象，如 1+2 分割为“1”，“+”，“2”，它们都是 Symbol 类的对象，那么“1”和“2”的 type 即为“数”，“+”的 type 即为“操作符”，传入字符类型的判别只需要在构造函数中调用一个 setType()方法即可，若传入了非法字符，则会抛出异常。

这样一来，转换逆波兰过程中，对操作符、操作数的分类处理就很好实现了，只需要一个 switch 语句即可实现。

那么操作符的优先级如何控制呢？便可使用一个 Operator 类，直接将其属性封装起来，操作符栈存放这个类的对象即可，同样，若使用了未加入指令集中的运算符，也会抛出异常。

如此一来，使程序之间耦合度更低，可扩展性更好，程序代码也更加清晰。由于进行了异常的考虑，程序将会更加健壮。

本次实验中，令我意识到，如果用好工具，选对方向，将会大大简化工作，正则表达式方便了字符串的分割、symbol 类型的判别等，若一一遍历，将会十分复杂。WolframAlpha 的 API 十分强大，调用之便可有效丰富程序的功能。要多多探索，大胆尝试，才能不断发现“新天地”。