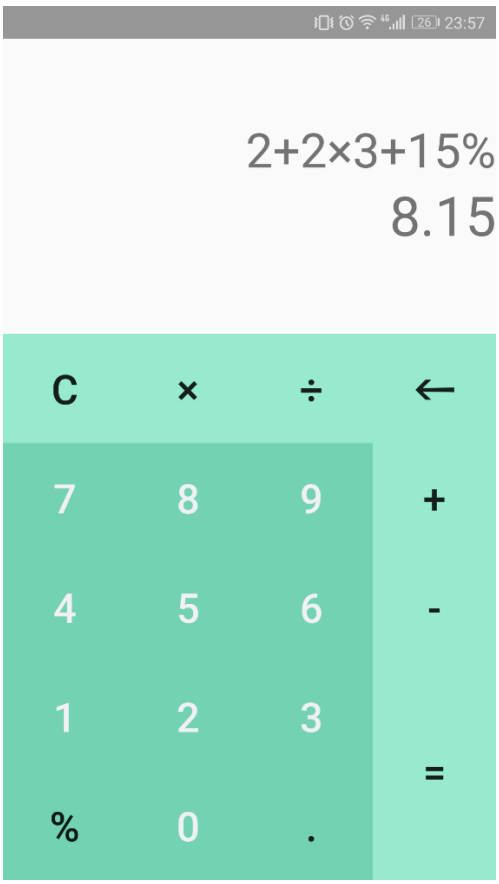
Calculator

软件内容简介：

Calculator 是一个简易计算器，能实现简单数学表达式的四则运算，支持小数运算，百分号运算，加减乘除实现了优先级判断。能进行结果的动态展现，用户体验效果好。

界面设计：



软件操作流程：

Yous Calculator 计算器使用起来跟实际的计算器使用流程相同，用户操作友好，输入正确的运算表达式都可以得到正确的结果。

难点：

在编写计算器的过程中遇到了很多难点，首先是第一次用 Android Studio 制作安卓 App 所以界面控件布局设计刚开始一直拍不好，其次计算器在对输入的表达式的处理计算遇到了小数计算的难点，解决方案为采用 java 的 split()函数进行运算符和数的切割，但是这样就完成不了输入负数的运算。

不足之处：

Calculator 暂时无实现运算的负数运算以及括号优先级运算，对于小数运算有时小数位不精确。

今后设想：

实现括号运算、负数运算等科学计算器应有的运算，使小数计算结果更加精确，设计更好的用户交互界面以及反馈信息，为用户提供更好的服务。

代码设计：

以下使核心代码设计：

```
class Calculator
{
    private String strExpression;
    private Hashtable<Character, Integer> ht = new Hashtable<>();
    private char Key[] = {'#', '+', '-', '×', '÷', '%'};
    private int Priority[] = {0, 1, 1, 2, 2, 3};
    //操作运算符栈
    private Stack<Character> CharStack = new Stack<>();
    //操作数栈
    private Stack<Double> NumStack = new Stack<>();
    //运算结果
    private double dResult = 0;
    private String strResult = "";
    //构造函数
    public Calculator()
    {
        for(int i =0; i<Key.length; i++)
        {
            ht.put(Key[i], Priority[i]);
        }
        this.strExpression = "";
    }
    //带参构造函数
    public Calculator(String strExpression)
    {
        for(int i =0; i<Key.length; i++)
        {
            ht.put(Key[i], Priority[i]);
        }
        strExpression = strExpression.concat("#");
        this.strExpression = strExpression;
```

```
            ht.put(Key[i], Priority[i]);
        }
        strExpression = strExpression.concat("#");
        this.strExpression = strExpression;
    }
    public void setStrExpression(String strExpression)
    {
        strExpression = strExpression.concat("#");
        this.strExpression = strExpression;
    }
    public String getStrExpression() { return this.strExpression; }
    public double getdResult() { return this.dResult; }
    public String getStrResult() { return this.strResult; }
    public double[] getNum(String string)
    {
        String strNumArray[] = string.split("#|\\+|-|×|÷|%");
        double dNumArray[] = new double[strNumArray.length];
        for(int i = 0,j = 0; i < strNumArray.length; i++)
        {
            if(!strNumArray[i].isEmpty())
            {
                dNumArray[j++] = Double.parseDouble(strNumArray[i]);
            }
        }
        return dNumArray;
    }
    public double Operate()
    {
        //操作数数组
        int j = 0;
        double dNumArray[] = getNum(strExpression);
```

```
public double Operate()
{
    //操作数数组
    int j = 0;
    double dNumArray[] = getNum(strExpression);
    CharStack.push('#');
    NumStack.push(dNumArray[j++]);
    for (int i = 0; i < strExpression.length();)
    {
        char charTemp = strExpression.charAt(i);
        if(charTemp == '+' || charTemp == '-' || charTemp == '×' || charTemp == '÷')
        {
            if(ht.get(charTemp) > ht.get(CharStack.peek()))
            {
                CharStack.push(charTemp);
                NumStack.push(dNumArray[j++]);
                i++;
            }
            else
            {
                char charOperator = CharStack.pop();
                calculate(charOperator);
            }
        }
        else if(charTemp == '%')
        {
            NumStack.push(NumStack.pop() *0.01);
            i++;
        }
        else if(charTemp == '#')
        {
            if(CharStack.peek() == '#')
```

```
            else
            {
                char charOperator = CharStack.pop();
                calculate(charOperator);
            }
        }
        else if(charTemp == '%')
        {
            NumStack.push(NumStack.pop() *0.01);
            i++;
        }
        else if(charTemp == '#')
        {
            if(CharStack.peek() == '#')
            {
                dResult = NumStack.peek();
                return  dResult;
            }
            else
            {
                calculate(CharStack.pop());
            }
            /**
             * while
             */
        }
        else
        {
            i++;
        }
    }
    return 0;
```

```
    return 0;
}
public int calculate(char charOperator)
{
    double dNum1 = 0;
    double dNum2 = 0;
    switch(charOperator)
    {
        case '+':
            dNum1 = NumStack.pop();
            dNum2 = NumStack.pop();
            NumStack.push(dNum1 + dNum2);
            break;
        case '-':
            dNum1 = NumStack.pop();
            dNum2 = NumStack.pop();
            NumStack.push(dNum2 - dNum1);
            break;
        case '×':
            dNum1 = NumStack.pop();
            dNum2 = NumStack.pop();
            NumStack.push(dNum1 * dNum2);
            break;
        case '÷':
            dNum1 = NumStack.pop();
            dNum2 = NumStack.pop();
            if(dNum1 != 0)
            {
                NumStack.push(dNum2 / dNum1);
            }
            else
```

```
            break;
        case '×':
            dNum1 = NumStack.pop();
            dNum2 = NumStack.pop();
            NumStack.push(dNum1 * dNum2);
            break;
        case '÷':
            dNum1 = NumStack.pop();
            dNum2 = NumStack.pop();
            if(dNum1 != 0)
            {
                NumStack.push(dNum2 / dNum1);
            }
            else
            {
                return 0;
            }
            break;
...
    }
    return 1;
}
```