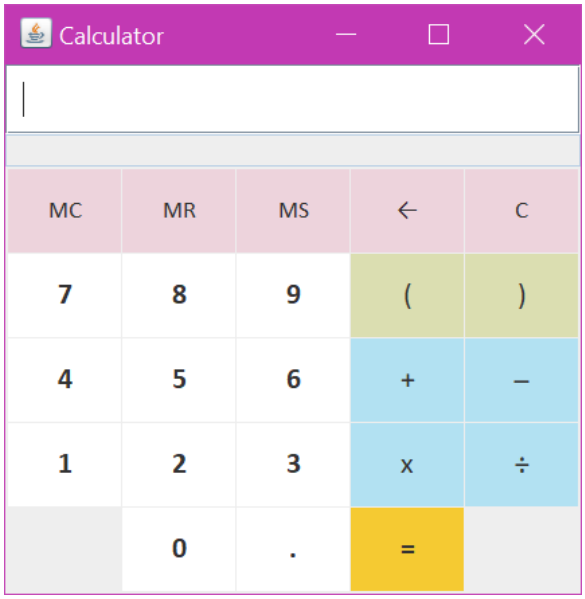
CALCULATOR PROJECT



Authors: Martin O'Sullivan & Robert Kamenicky

C00227188 & C00231987

Lecturers: Dr.Jason Barron and Aíne Byrne

GitLab Link: www.gitlab.com/osullivanMartin/calculatory2

Table of Contents

Introduction

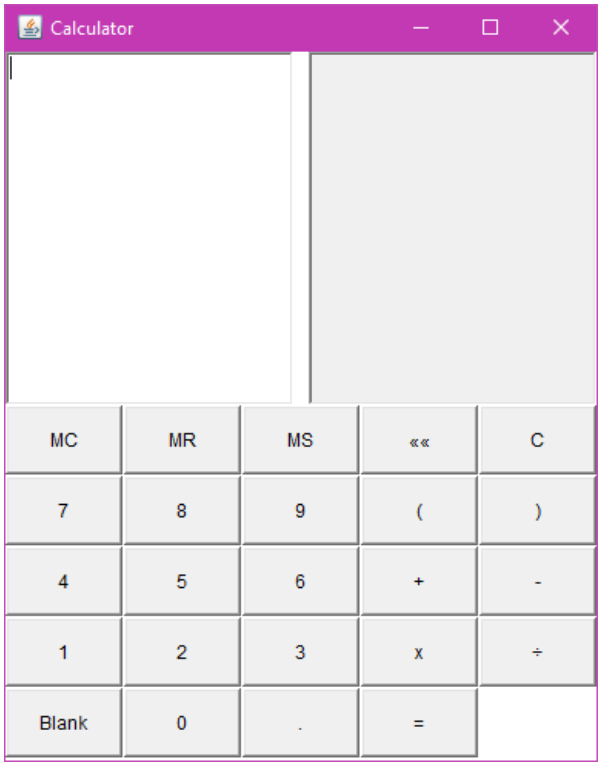
This projects' task is to create a calculator with basic functionality including the backend algorithms and front-end user interface using Java Abstract Windows Toolkit (AWT) or Java Swing. The calculator should be user friendly with a simplistic interface. It is mandatory to follow the correct Java conventions such as indentation or relevant method naming.

Requirements

To develop a basic functioning calculator with a graphical user interface to allow user input and interaction. The following functions are to be included within the calculator:

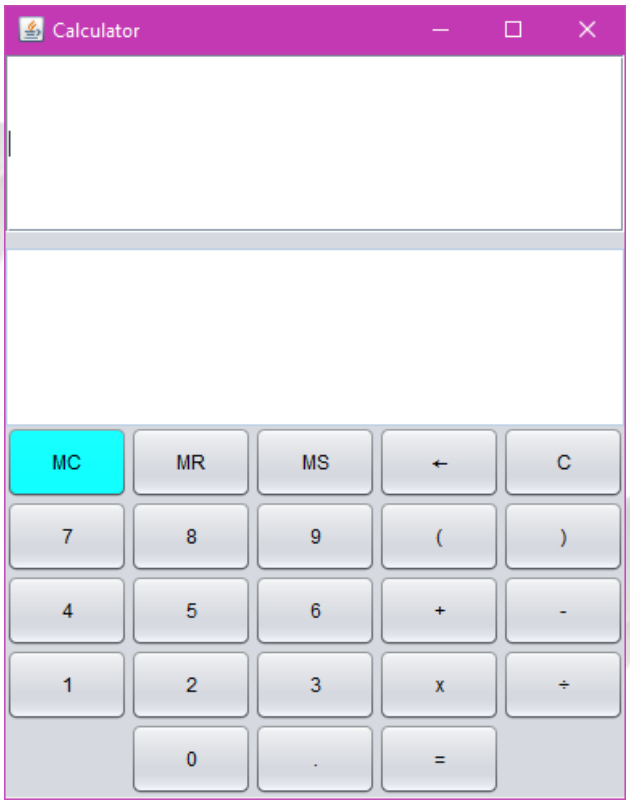
- Addition
- Subtraction
- Multiplication
- Division
- Clear Screen capability
- Memory capabilities
- The use of at least one user created Exception class to be used in Error Handling

Graphical User Interface



Our approach to the Graphical User Interface (GUI) was to first develop a prototype and planned to refine it at a later date. This approach enabled us to focus on the algorithm and bug-fixing involved. Our prototype was built using AWT.

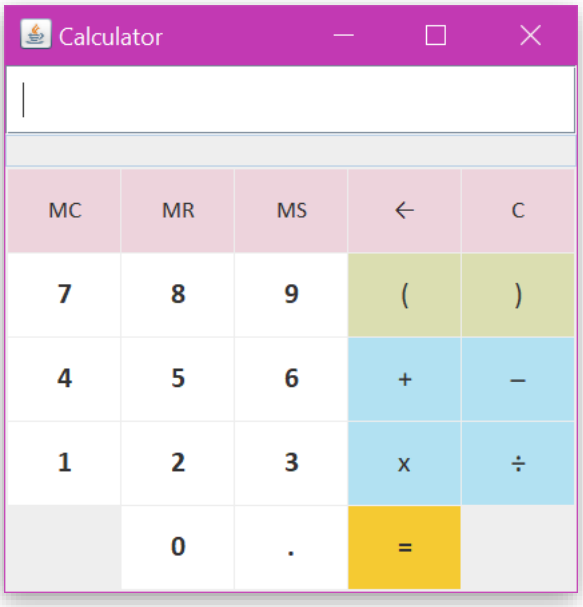
Our Prototype needed to encapsulate the design we envisioned, which can be seen on the right, without spending the time to refining the design. In our prototype, the final design vision can be seen with the “blank” button being used at bottom left corner and the blank space at the bottom right. At this time of development, we were unable to use certain characters such as the backspace button which can be seen in the finished design. This is due to the limitations of base Java AWT being unable to support Extended Unicode characters.



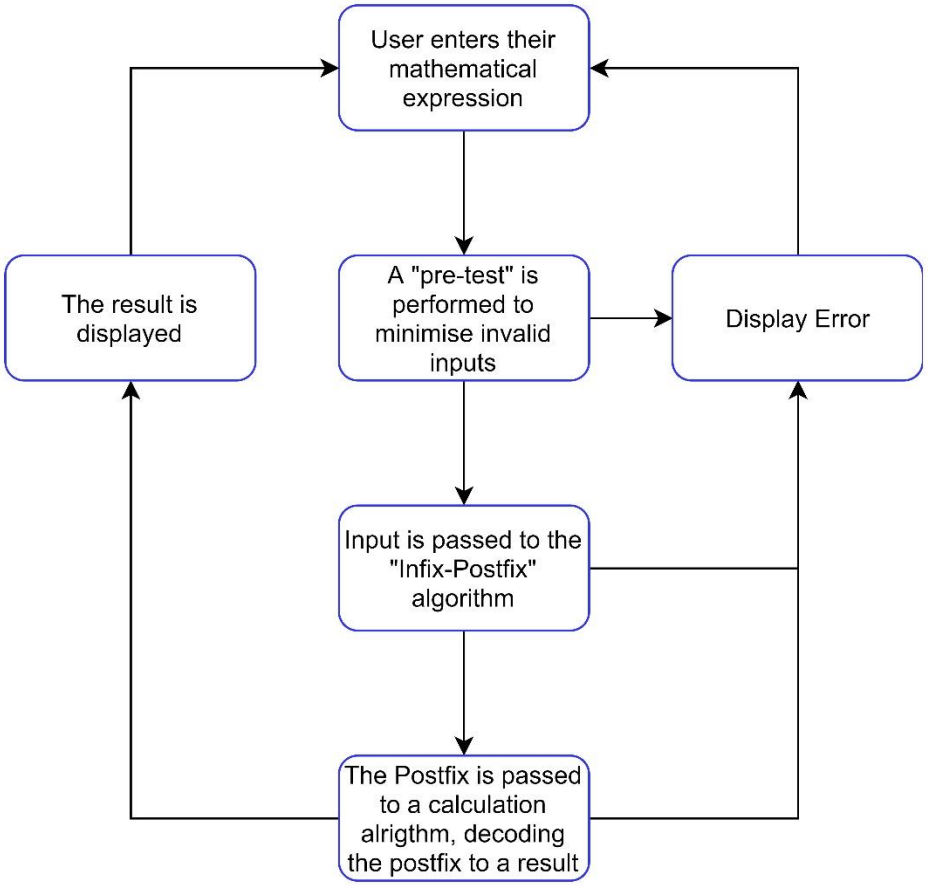
The second iteration can be seen on the left. At this point we transitioned to Java Swing. We also decided to use the “Nimbus” look and feel. We appreciate the styling Nimbus brought, though we weren’t happy with some of the styling, for example upon setting a button colour, the gradient was eliminated from the buttons. We wanted to colour buttons without causing a clear disconnect from the rest of the calculator design.

Upon fixing the input box, and the display screen below it, we made

the discovery that the spacing of “Gridlayout” can be made negative. With this discovery, we were quickly able to develop our envisioned look. This brought the removal of the Nimbus look and feel. The buttons were now able to be coloured whilst keeping the flat colour scheme, the font was changed to Calibri, numbers were made bold, and all button font sizes were changed to ensure readability. Certain buttons were visually grouped by applying matching colours.



Calculation Logic



From a user perspective, we wanted our calculator to be as user intuitive as possible. Our mission was to enable the user to enter an expression and receive a result. An example of this being is upon a user desiring to do a multiply with brackets, such as “9(5+3)”, the calculator should return 72, without the user explicitly stating the multiply operating between the “9” and “(5+3)”.

Calculation doesn’t take place until the user presses the “=” button or enter on their keyboard. After this event, the input expression is first split into an ArrayList. This allows simplified manipulation of the equation. This ArrayList is now sent to an interpreting algorithm that makes appropriate changes, such as concatenating a subtraction to a number that is negative. The interpretations it handles are: multi-digit numbers such as 198, numbers with decimal places, negative numbers, multiplication due to brackets such as “(5)9”

Once the interpreting algorithm has completed, the equation is passed to be converted to Reverse Polish Notation, also known as “Postfix”.

We use this to allow simplistic handling of expressions, while allowing the use of brackets for the user. This conversion requires the use of the “Stack” data structure. The postfix is saved in a new ArrayList.
The algorithm iterates over the input, also known as “Infix”. The simplistic breakdown of this algorithm is:

If a “(“ bracket is found, push it onto the Stack.

If a “)” bracket is found, pop the Stack and save the popped values. Keep popping until a “(“bracket is found.

If an operator is found, compare its precedence priority with a “peak” of the stack. If the operator has higher precedence than the peak element, push it to the stack

If an operand (number) is found, save its value.

Upon reaching the end of the infix expression, pop all the elements in the stack and append them to the end of the postfix ArrayList.

The postfix is now ready to be evaluated. Beginning from the left side, a search for an operator is initiated. Upon a find, two numbers before the found operator are *take* and are evaluated. The result is placed back into the position of the operator and two numbers. This is repeated until a single number remains

Testing

To ensure the calculator is functioning as intended we need to run tests of sample input. For these tests, we will compare the results of our calculator to that of WolframAlpha. It is often highly regarded in Mathematical Computations. A test will be considered a success if it returns the same final result value as WolframAlpha.

Re: WolframAlpha, www.wolframalpha.com

Test No.	Test Scenario	Test Case	Expected Result	Actual Result	Status
TN 1	User types a number	Number typed	A number is displayed	A number is displayed	Success
TN 2	Test Addition with three numbers	1+2+3	“6” should be returned	“6” is the result	Success
TN 2.1	Test Addition with three decimal numbers	1.1+2.2+3.3	“6.6” should be displayed	“6.6” is the result	Success
TN 3	Test Subtraction with three numbers	10-1-1	“8” should be the result	“8” is returned	Success
TN 3.1	Test Subtraction with three decimal numbers	10.1-1.5-2.6	“6” should displayed to the user	“6” is shown	Success
TN 4	Test Multiplication with three numbers	2*3*4	“24” being displayed	“24” is displayed	Success

TN 4.1	Test Multiplication with three decimal numbers	2.2*3.3*4.4	“31.944” is expected	“31.944” is returned	Success
TN 5	Test Division with three numbers	2/3/4	“0.1666” repeating will be shown	“0.16666” repeating is shown	Success
TN 5.1	Test Division with three decimal numbers	4.412/0.136/0.123	“263.74940219” will be returned	“263.74940219” is the result	Success
TN 6	Test Single Bracket	5+(1+2)	“8” should be the result	“8” is the result	Success
TN 6.1	Test nested Brackets	5+(1+2+(5+3))	“16” being returned	“16” was the result	Success
TN 6.2	Test multi bracket with precedence	1+(2/3*(4+2))/23.12*(5.123+2)	“2.2323529” should be the result	“2.2323529” is returned	Success
TN 7	Test Memory Functions	-User types an expression -After this click “MS” to save the expression -Click “MR” to get result back -Click “MC” to clear the memory	The expression should be saved, duplicated in the input box and finally clear the memory	The expression is duplicated	Success
TN 8	Test if user enters input with operand in the last place	2+3+	WolframAlpha can’t calculate and returns an error, however we expect this to be allowed in our calculator	“5” is returned	Success
TN 9	Test Backspace Function	User clicks “←”	A single character is deleted from the input box	A single character is deleted	Success
TN 9.1	Test Clear Function	User Clicks “C”	The expression screen is cleared	The input box is cleared	Success

Improvements and Current Limitations

As we chose to allow the user to type, without the limitation of being forced to use the buttons, certain checks aren't being performed on a typing basis. This can be fixed by running these checks using a KeyListener. An example of what is possible that isn't by using the buttons is typing as many operands as desired, for example 5++++9. If the buttons are used typing this, this duplication of operands is contained.

We wanted to implement the ability to get the last equation entered by using up and down keys on the keyboard. This is a simple feature that would improve the experience of using the calculator in a minor and subtle way. Due to time constraints we were unable to implement this.

The calculator currently has limited functions, it is unable to do complex functions such as modulo, square root or raising numbers to the power of a number. This can be added in future iterations.

Conclusion

All the requirements have been met as stated in the functional specification. While the calculator may have limited functionality, it makes up in its ability to be user-friendly. We aimed to make the user experience as simplified as possible. We were dissatisfied with the user being limited in how they interact with the calculator such as being unable to type. The calculator needed to feel natural to use, you enter a desired expression, the calculator returns a result.