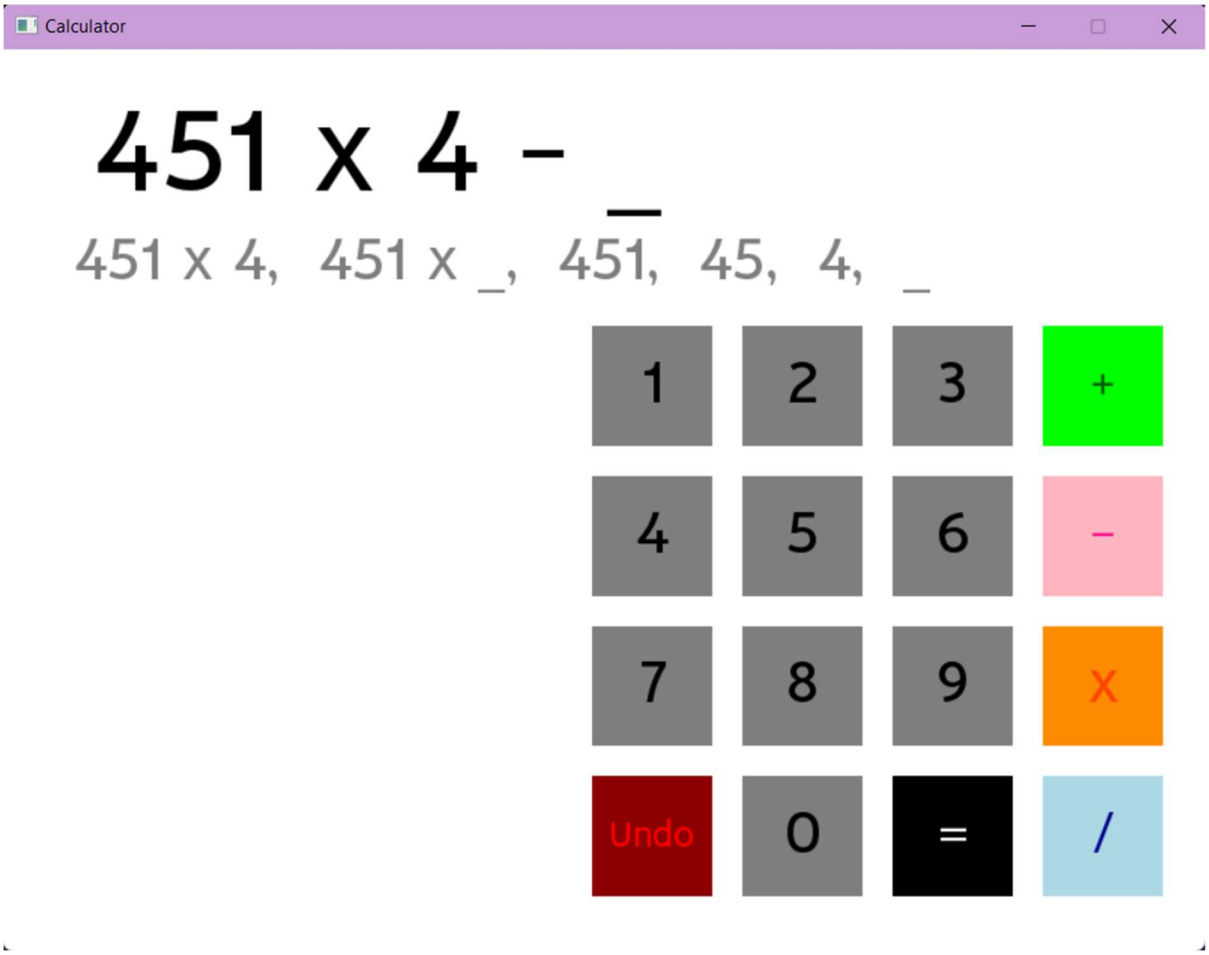
Updated Design Report for Visual Calculator

Name: Aidan Davies
Student ID: 103986200

Summary of Program

This program simulates a basic arithmetic calculator. The user is shown their current list of pending operations and prompted to input numbers and operations to conduct, which they do by clicking buttons on a visual interface. Once the user has provided all the steps they want to calculate, they can tell the program to perform all the calculations in order and output the result. The user may then continue to input operations to perform on the new result.

This calculator is capable of adding, subtracting, multiplying, and dividing 32-bit integers together, and will simply execute them in the inputted order, ignoring mathematical order of operations.



Extended Functionality

The HD version of this program includes two additional features. The first is the undo button: when clicked, this will return to the previous step of the calculation. The second is the update history on the left, showing previous calculation steps, with the left-most change being the most recent.

This program demonstrates an improvement over the Distinction-level design due to its use of design patterns to optimise the code. These optimisations reduce the amount of memory used by the program during runtime and make the program more easily extensible, as the program has been designed to reduce dependencies.

Created Classes

The following classes were created for this program:

- Program
- Calculator
- Calculator.Memento
- GUIHandler
- Caretaker
- TextDisplay
    - Button
        - CalculateButton
        - ModeButton
        - NumberButton
        - UndoButton
- Operation
- CalculationStrategy
    - AdditionStrategy
    - SubtractionStrategy
    - MultiplicationStrategy
    - DivisionStrategy

Design Features/Methods

Calculator.Calculate()

This method runs through all operations inside the Calculator's _operationList, calling the Calculate() method for each, and returning the final result. If any operations return *null*, then the calculation is cancelled. If all calculations succeed, then the Calculator calls Save() to store a memento of its previous operation list inside of the Caretaker.

Operation.Calculate(num: int)

When an Operation's Calculate() method is called, it simply passes its current number and *num* as parameters to the Calculate() method defined by its CalculationStrategy (in this case, it either adds, subtracts, multiplies or divides the two numbers), and returns the result. This method returns *null* if the calculation fails for any reason.

TextDisplay.Draw()

The object draws its label to the screen at its set position using SplashKit.DrawText(). The method contains an overload to offset the text's coordinates: this is used for aligning the text to the centre of buttons.

GUIHandler.Draw()

This method draws all required GUI elements to the screen. This includes the TextDisplays for the current equation and edit history, and the Buttons the user can clicked on.

Caretaker.Undo()

This method calls the Restore() method of the most recent Memento (and thus the user's most recent change) and resets the calculator's state back to it. Then, it removes that memento from its list.

Button.IsAt(Point2D mousePos)

This method checks whether the coordinates of *mousePos* fall within the button's boundaries. While there is a set SplashKit function for checking whether a point falls within a rectangle (PointInRectangle), this method requires the definition of a SplashKit.Rectangle object, which was considered too clunky to incorporate into this implementation of the program.

GUIHandler.CheckButtons(Point2D mousePos)

The GUIHandler calls IsAt() for all buttons on screen and determines if any were clicked. Since no buttons overlap, if *mousePos* falls within the bounds of any button, that button is immediately returned.

Program Input Handling

All mouse clicks from the user are read in the Main() method of Program: when the user left clicks, GUIHandler.CheckButtons is called. If a button was clicked, it can be immediately activated, since all buttons (ModeButton, NumberButton, etc.) are children of the Button class and thus share the Activate() method. If the Activate() method was successful (returns true), then the labels of the display are updated; otherwise, a basic error message is printed to the console.

Final UML Class Diagram