# **1. Objective**

**Purpose**

The purpose of this app (basic calculator) is to create an interactive GUI that will calculate any given number accurately and interactively.

## **Features and Functionality**

The calculator includes:

* A user interface with buttons for digits (0-9), operations (+, -, \*, /), and special functions (clear, evaluate, and decimal point).
* A display for showing current input and calculation results.
* Support for keyboard inputs for digits, operations, and evaluating expressions.
* Basic error handling (e.g., if there's an invalid calculation, it displays "Error").

# **2. Demonstration of Object-Oriented Tools and Techniques**

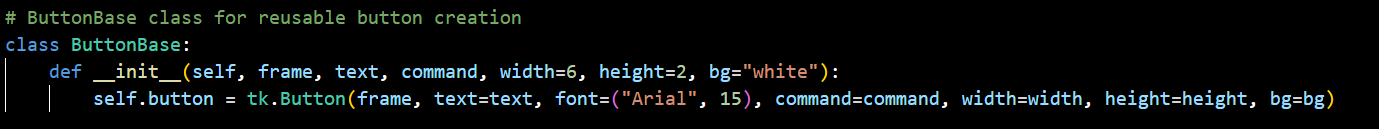
## **Object-Oriented Concepts Implemented**

### **Classes and Objects**

The calculator application is built around two main classes:

1. **ButtonBase**: Responsible for creating reusable button objects. Each button has properties like text, command, size, and colour.
2. **Calculator**: Manages the main application window, display, and button layout. This class handles user interactions and calculations.

**Example Code for ButtonBase and Object Creation:**



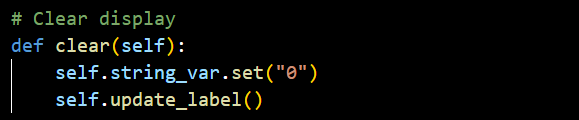
**Explanation**: Here, ButtonBase creates a button object that is then used throughout the application. Each button is an instance of this class.

### **Encapsulation**

Data in the calculator is controlled through class methods:

* The display value (string\_var) is not accessible directly; it is manipulated through methods like append(), evaluate(), and clear().
* This prevents accidental changes to the data and ensures the display behaves correctly.

**Example of Encapsulation:**

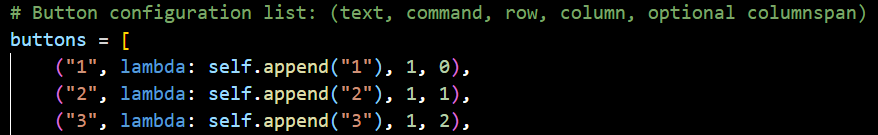
**Explanation**: The clear method modifies the display value through a controlled method call.

### **Polymorphism**

The application exhibits polymorphism in its handling of button actions. All buttons, despite being instances of ButtonBase, behave differently depending on the assigned command. For example:

* Number buttons add digits.
* The "=" button evaluates the expression.
* The "C" button clears the display.

**Example of Polymorphism:**



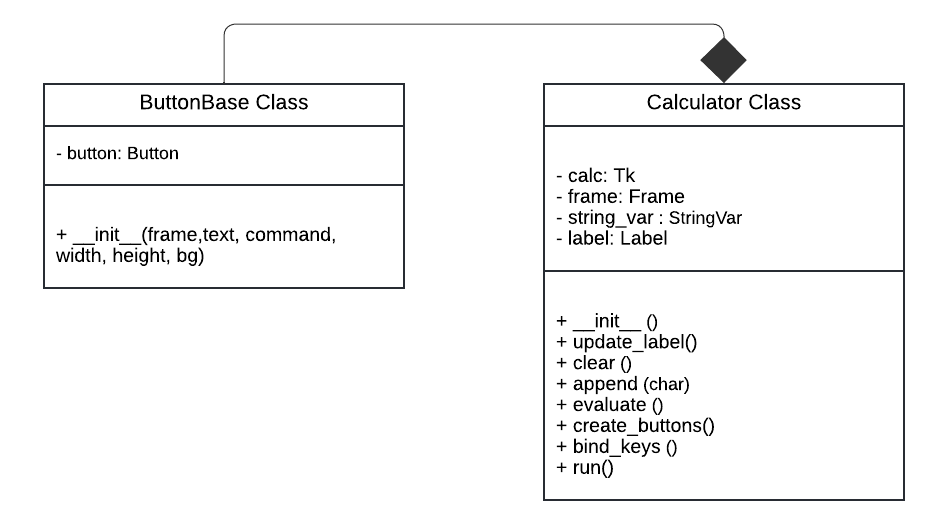
**Explanation**: Buttons are created using the same ButtonBase class but are assigned different behaviours via the command parameter.

# **3. Application Design**

## **Overview**

The application consists of a simple, user-friendly interface for basic calculations. It includes a top display area and a grid of buttons for digits and operations. Users can interact via button clicks or keyboard inputs.

## **UML Class Diagram**



## **Key Features and Interactions**

### **User Interface Design**

* **Display**: A label at the top shows the current input and results. It updates whenever a button is pressed or a calculation is performed.
* **Buttons**: There are buttons for digits, operators, and special functions:
  + Number buttons: Append digits to the display.
  + Operator buttons: Append operators.
  + "C" button: Clears the display.
  + "=" button: Evaluates the expression.

### **Methods and Functions**

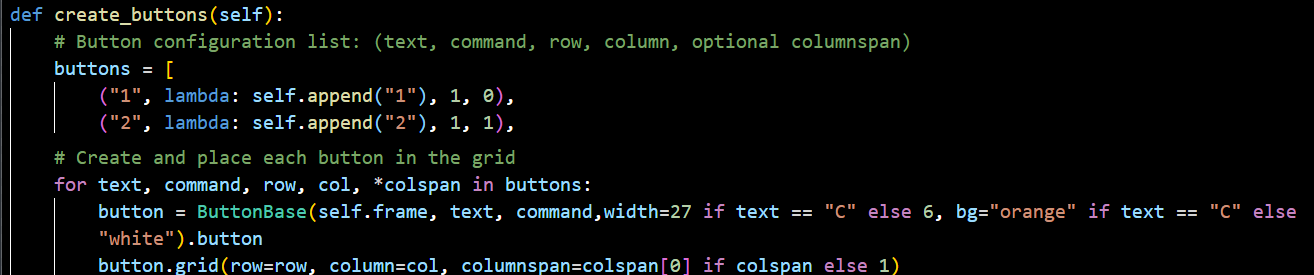
* **create\_buttons()**: Generates the calculator buttons using a loop, creating instances of ButtonBase.
* **append(char)**: Adds a number or operator to the current input.
* **clear()**: Resets the input display to "0".
* **evaluate()**: Evaluates the expression in the display.
* **bind\_keys()**: Sets up keyboard shortcuts.

# **4. Development Process**

## **Implementation**

Step-by-step development:

1. **Create Main Window**: Initialized with TKinter, setting up the primary window.
2. **Setup Display**: Added a display label to show input and results.
3. **Create Buttons**: Developed ButtonBase for creating buttons, and set up the grid layout.
4. **Add Functionality**: Implemented methods like append, clear, and evaluate.
5. **Keyboard Shortcuts**: Configured bind\_keys() for easy input using the keyboard.

**Key Code Snippet: Setting up Buttons**

## **Challenges and Solutions**

* **Challenge**: Managing input errors (e.g., invalid expressions).
  + **Solution**: Implement error handling in the evaluate method to display "Error" when necessary.
* **Challenge:** Button Layout Issue for 'C' (Clear Button).
  + **Solution:** Adjusted the button's properties manually during button creation.

# **5. Testing**

## **Error Documentation and Correction**

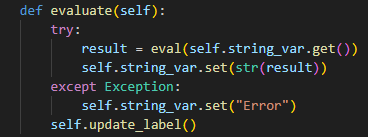
### **Identified Bugs/Errors**

* **Invalid Expression Handling**: Entering an incomplete expression caused a crash.
* **Division by Zero**: Needed to manage division by zero.
* **Button Layout Issue for 'C' (Clear Button):** The "C" button was intended to span 4 columns to make it visually distinct and maintain a structured look. The code "("C", self.clear, 5, 0, 4)" was an attempt to span 4 columns, but it did not work as expected.

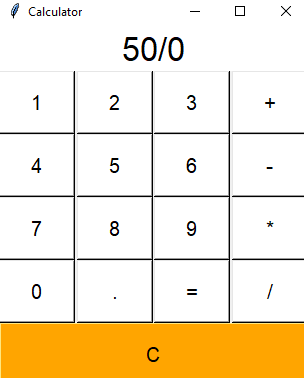
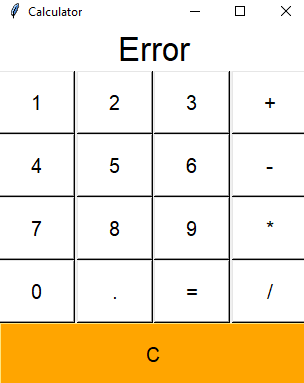
Additional Requirement: The "C" button needed to have a different colour (orange) to distinguish its purpose as a clearing action.

### **Proposed Solutions**

* **Error Handling**: Wrapped the eval() call in a try-except block to catch exceptions.



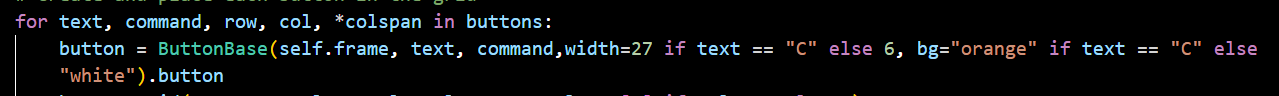
* **Division by Zero**: Displayed "Error" if such a situation occurred.

* **Solution for Column Span**: Initially, the code tried to use columnspan directly within the button configuration, but it did not apply the column span correctly.



Adjusted the button's properties manually during button creation to set the width to a larger value (width=27) specifically when the button's label is "C". This workaround achieved the desired look.



## **Test Data Design**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Test Case ID | Objective | Steps | Expected Outcome | Actual Outcome |
| 1 | Verify addition works correctly | Enter 2 + 2 and press "=" | Result is 4 | Works as expected |
| 2 | Test clear button functionality | Press "C" after any input | Display resets to "0" | Works as expected |
| 3 | Handle division by zero | Enter 5 / 0 and press "=" | Displays "Error" | Works as expected |
| 4 | Check keyboard input | Use keyboard for "7 + 8 =" | Result is 15 | Works as expected |
| 5 | Ensure 'C' button spans 4 columns | Observe layout after initialization | Button spans the width of 4 columns | Works as expected (using width adjustment) |

### **Test Execution and Results**

All test cases passed successfully, confirming that the calculator handles basic operations, input errors, and keyboard inputs correctly.

### **Test Report Summary**

* **Overall Findings**: The calculator performs accurately for valid inputs and gracefully handles errors.
* **Recommendations**: Consider extending the calculator to support more complex expressions (like parentheses and modulus).

# **6. Conclusion**

The development of the calculator application provided a practical exercise in applying OOP principles:

* **Reusability**: The ButtonBase class made it simple to generate buttons with consistent behaviour.
* **Encapsulation**: Controlling data through methods ensured data integrity.
* **Polymorphism**: Simplified input handling by assigning different commands to buttons.

Overall, the project demonstrated how OOP makes code easier to manage, extend, and debug.

# **7. Appendices**

## **Source Code**

The full source code can be found below:



## **References**

Python Software Foundation. (2024) Retrieved November 21, 2024, from. tkinter documentation for Python GUI development.

Python Documentation. Retrieved November 21, 2024, from <https://docs.python.org/3/library/tk.html>

Starcher, L. (2024). Python eval() for expression evaluation. Real Python. Retrieved November 21, 2024, from <https://realpython.com/python-eval-function/>

W3Resource. (2024). Python tkinter basic exercise 13. Retrieved November 21, 2024, from <https://www.w3resource.com/python-exercises/tkinter/python-tkinter-basic-exercise-13.php>

Wells, R. (2024). Building a calculator with Python tkinter. wellsr.com. Retrieved November 21, 2024, from <https://wellsr.com/python/building-a-calculator-with-python-tkinter/>

Spiff, D. (2024). Calculator with GUI using Python tkinter. dev.to. Retrieved November 21, 2024, from <https://dev.to/spiff/calculator-with-gui-using-python-tkinter-jpn>

Visual Paradigm. (2024). *UML class diagram tutorial*. Visual Paradigm. Retrieved November 25, 2024, from <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/uml-class-diagram-tutorial/>

OpenAI. (2024). ChatGPT [Online tool]. Retrieved November 21, 2024, from <https://chatgpt.com>