OO Design for Graphics: A Simple RPN Calculator

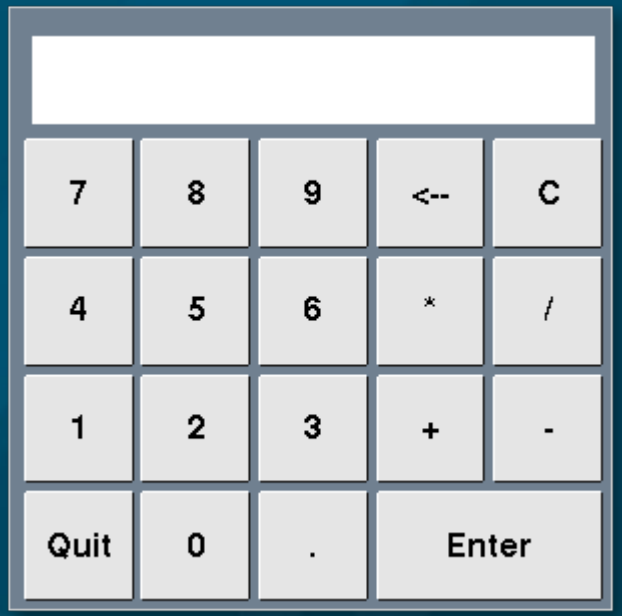
- button.py
- calculator.py

2 Dec 2009
CMPT140
Dr. Sean Ho
Trinity Western University

TRINITY WESTERN UNIVERSITY

Top-down GUI design

- Task: make a simple RPN calculator
  - Reverse Polish Notation: use a stack
- GUI design:
  non-functional mock-up
  - Analogous to Sample I/O
    in lab write-ups
- Buttons:
  - Location, size, text label
  - Need to detect mouse clicks
- Write and test a class just for a Button



Button class: see button.py

- Attributes:
  - A Rectangle for the body of the button
  - A Text label that goes on top
- Constructor:
  - Get position (centre), size (height/width), and string label
  - Create and draw a Rectangle box
    - Compute top-left and bottom-right corners
  - Create and draw the Text label
- Also lots of accessor (get) functions

Button: activate() and clicked()

- Activation: a bool flag enabling/disabling button
  - Public methods: activate(), deactivate()
  - Set flag and change appearance (dim)
- Check for button click: clicked(pt)
  - Parameter: a Point, from getMouse()
  - Check if button is active
  - Check if coordinates of pt are within the button's box
  - Return a bool

Unit testing

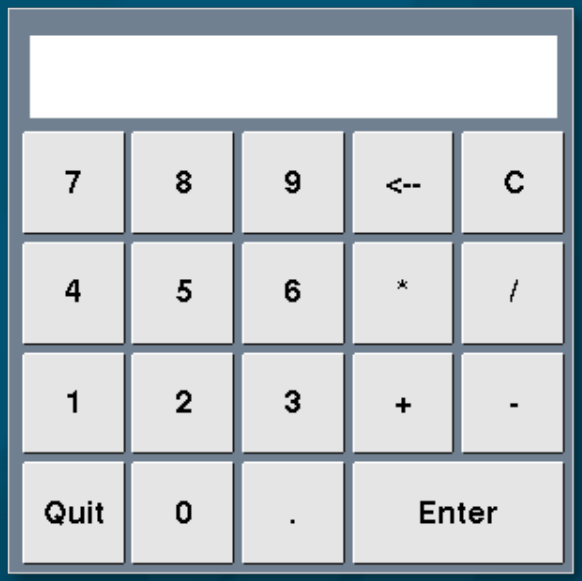
- Unit testing is testing a component in isolation
  - Need to know the public interface
    - **Constructor, (de)activate(), clicked()**
  - Any invariants (conditions guaranteed by the component to be true)? Check them!
  - Pre/post-conditions of all public methods
  - Write test cases: then unit testing can even be automated! (see: pyunit)
- Once all components have been unit tested, integration testing ensures they work together

Unit testing: Button

- In IDLE, create a window:
  - **from graphics import ***
  - **win = GraphWin()**
- Create a Button or two:
  - **from button import Button**
  - **b1 = Button(win, 50, 50, 60, 20, “hello!”)**
- Activate/deactivate them: **b1.activate()**
- Try testing for clicks:
  - **b1.clicked(win.getMouse())**
    - Will it work when button is deactivated?

Calculator: attributes

- Now on to the main program (see calculator.py)
- GUI widgets (attributes):
  - Main window
  - Display/output:
    text box and rectangle
  - Function buttons
- Attributes for internal computation:
  - Stack for RPN calculations
  - Current number being entered by user
    - **Allow backspace button to edit this**



Calculator: constructor

- We split up the constructor into several parts:
  - Setup the window and coordinate system
  - Setup the output display area
    - **Subroutine: __createDisplay()**
  - Setup all the buttons
    - **Subroutine: __createButtons()**
  - Initialize the stack/internals
- Coordinate system: centre the buttons on a 5x5 grid from (0,0) to (4,4)
  - Margin: setCoords(-0.5, 4.5, 4.5, -0.5)

Calculator: button config

- We have a lot of buttons to create: for each, set
  - Location (x, y coords of centre)
  - Label (text string)
- Store the configuration in a dictionary:
  - **buttonConfig = { “*”:(3,2), “/”:(4,2) … }**
- Loop over the dictionary to create the buttons:
  - **for (label, (x, y)) in buttonConfig: ….**
- Easy to reconfigure the buttons!

Calculator: main event loop

- The overall flow for the main program is an event loop (run() method):
- Infinite loop waits for an event from user
  - Events: mouse click, motion, keypress, ...
  - In this case, just mouse clicks: getMouse()
- When user clicks, find which button was clicked
  - Iterate over all buttons, testing if .clicked()
- Then do the appropriate action
  - Big if/elif on all possible actions: 0-9, +-*/, Enter, backspace, quit, ...