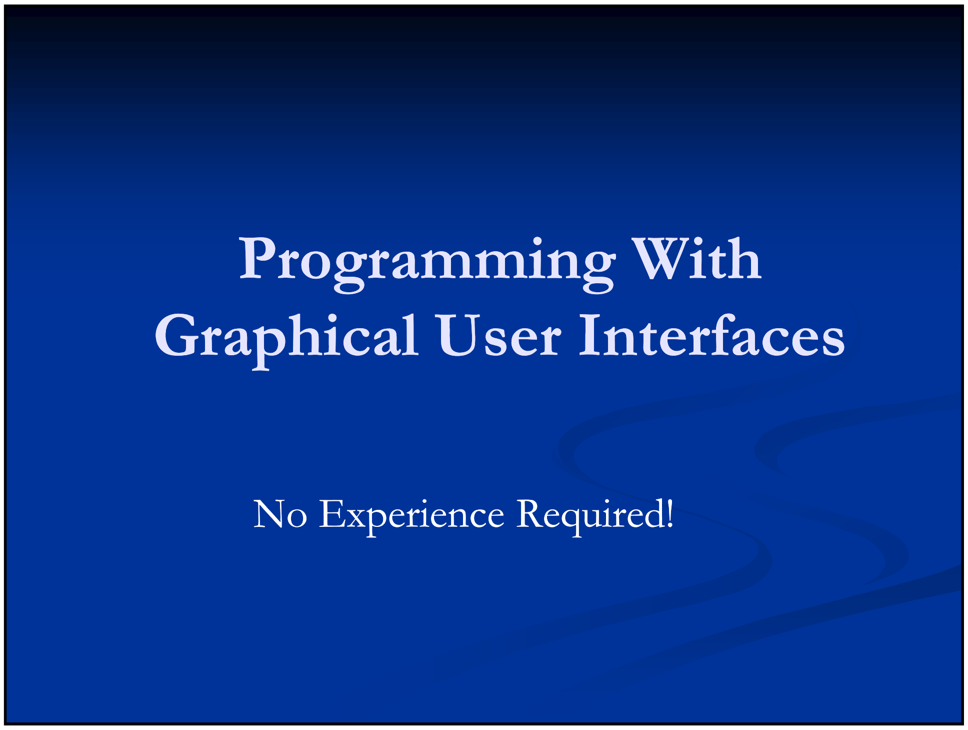
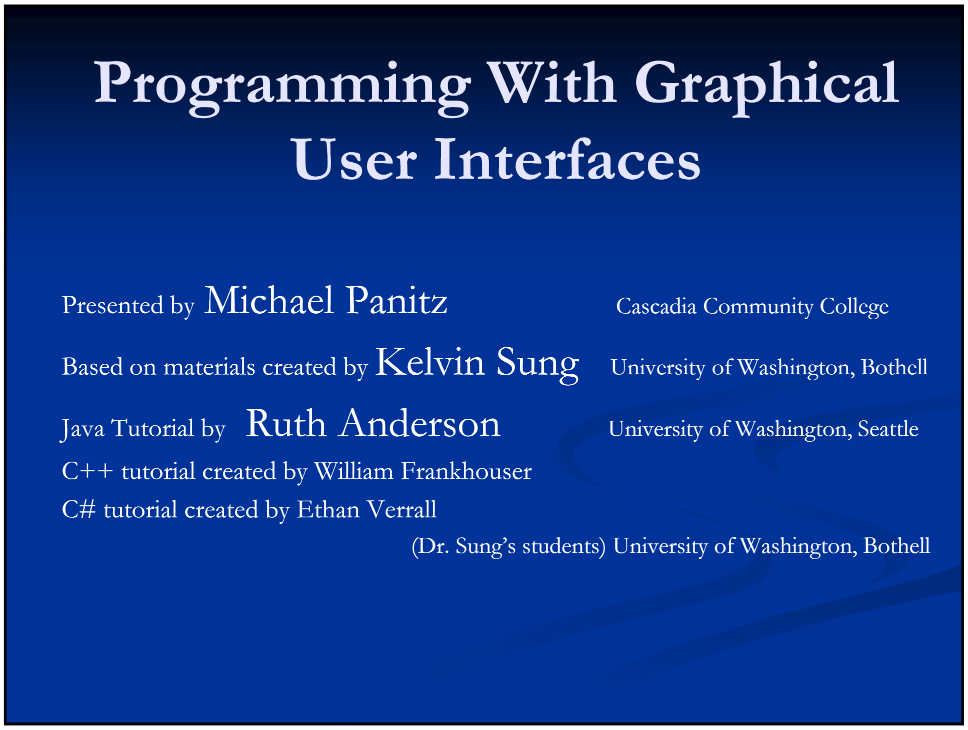


Programming With Graphical User Interfaces

No Experience Required!

Online at:

http://www.alternitz.com/CCSC2007/



Programming With Graphical User Interfaces

Presented by Michael Panitz Cascadia Community College

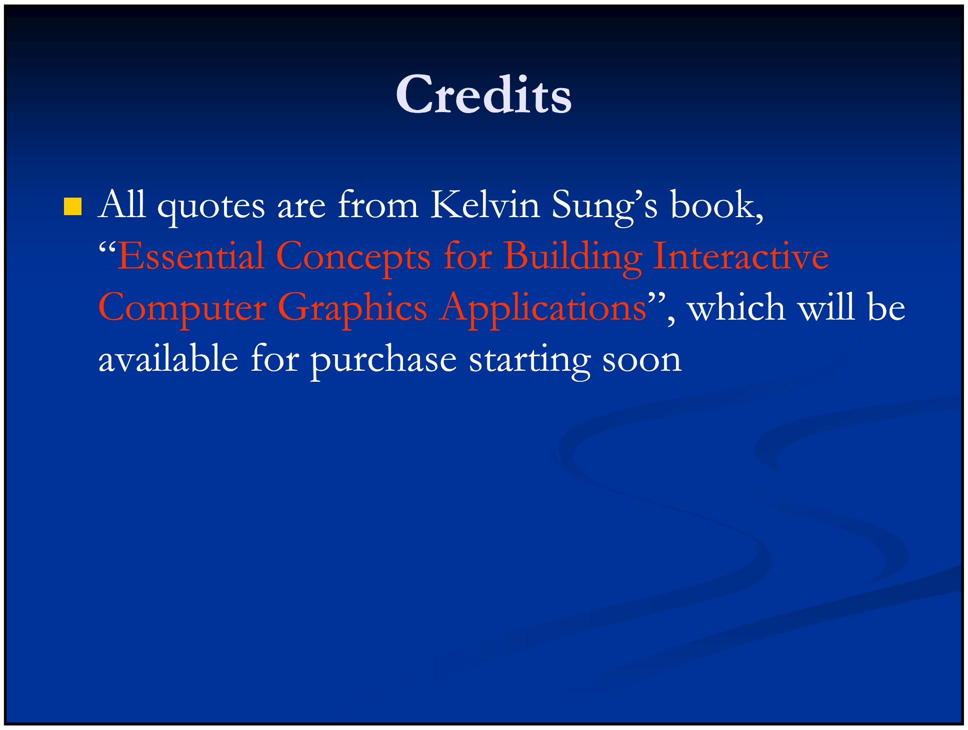
Based on materials created by Kelvin Sung University of Washington, Bothell

Java Tutorial by Ruth Anderson University of Washington, Seattle

C++ tutorial created by William Frankhouser

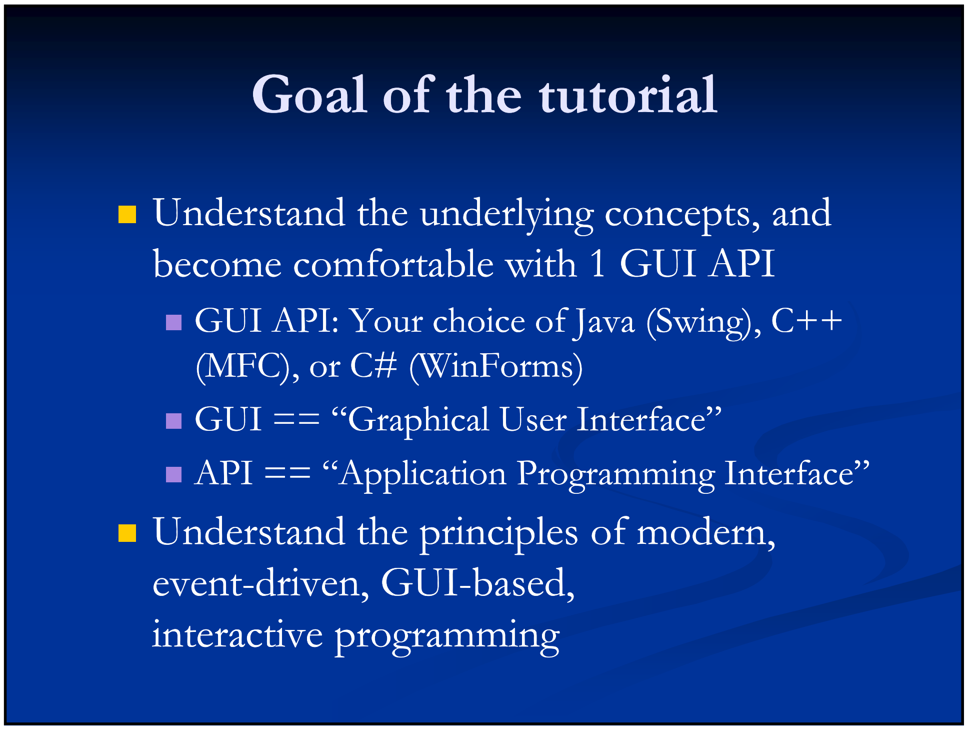
C# tutorial created by Ethan Verrall

(Dr. Sung's students) University of Washington, Bothell



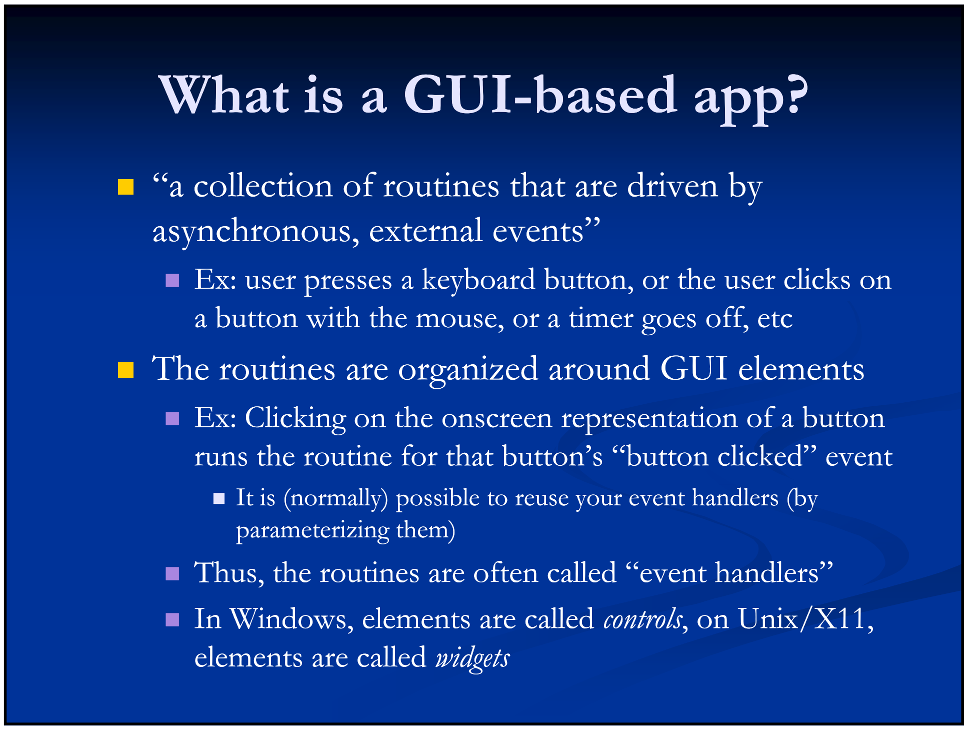
Credits

- All quotes are from Kelvin Sung's book, "Essential Concepts for Building Interactive Computer Graphics Applications", which will be available for purchase starting soon



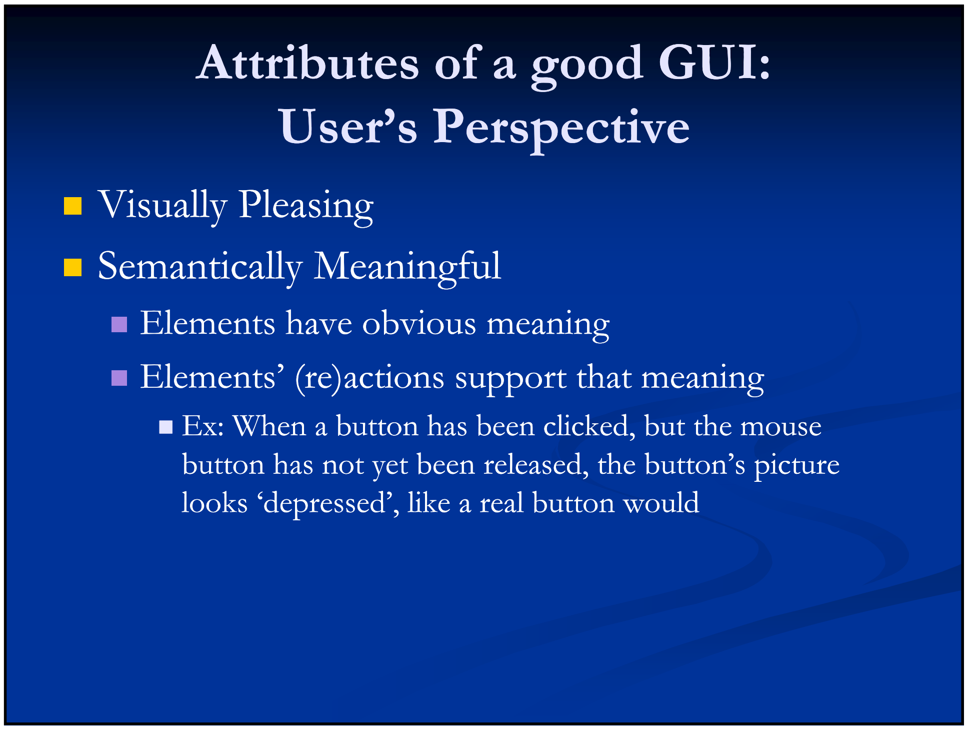
Goal of the tutorial

- Understand the underlying concepts, and become comfortable with 1 GUI API
  - GUI API: Your choice of Java (Swing), C++ (MFC), or C# (WinForms)
  - GUI == “Graphical User Interface”
  - API == “Application Programming Interface”
- Understand the principles of modern, event-driven, GUI-based, interactive programming



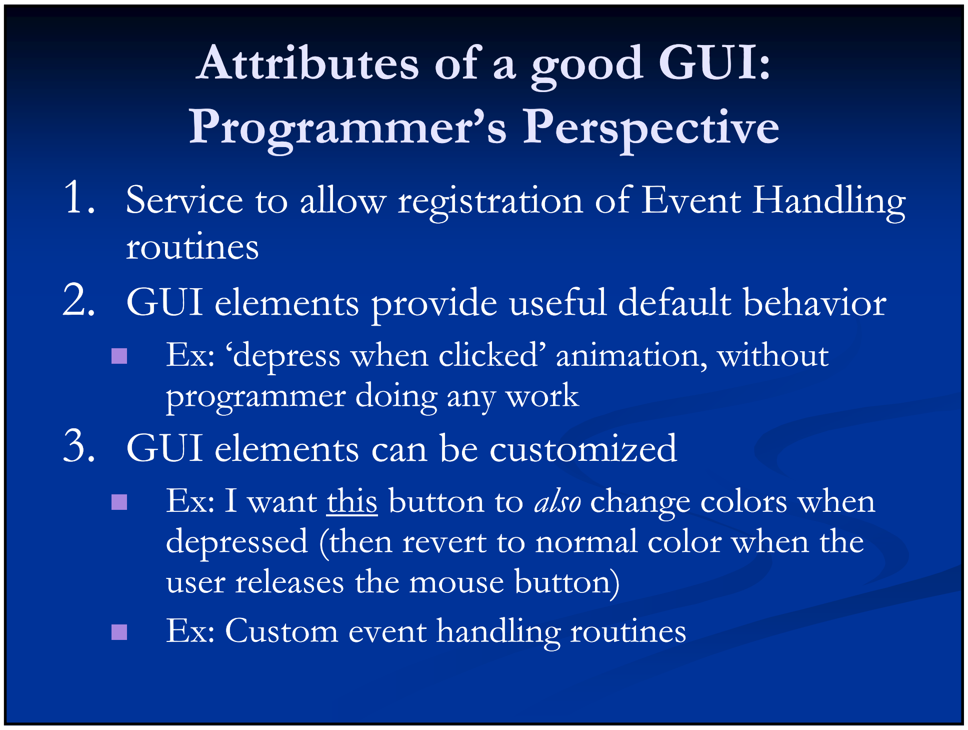
What is a GUI-based app?

- “a collection of routines that are driven by asynchronous, external events”
  - Ex: user presses a keyboard button, or the user clicks on a button with the mouse, or a timer goes off, etc
- The routines are organized around GUI elements
  - Ex: Clicking on the onscreen representation of a button runs the routine for that button’s “button clicked” event
    - It is (normally) possible to reuse your event handlers (by parameterizing them)
  - Thus, the routines are often called “event handlers”
  - In Windows, elements are called *controls*, on Unix/X11, elements are called *widgets*



Attributes of a good GUI: User's Perspective

- Visually Pleasing
- Semantically Meaningful
  - Elements have obvious meaning
  - Elements' (re)actions support that meaning
    - Ex: When a button has been clicked, but the mouse button has not yet been released, the button's picture looks 'depressed', like a real button would



Attributes of a good GUI: Programmer's Perspective

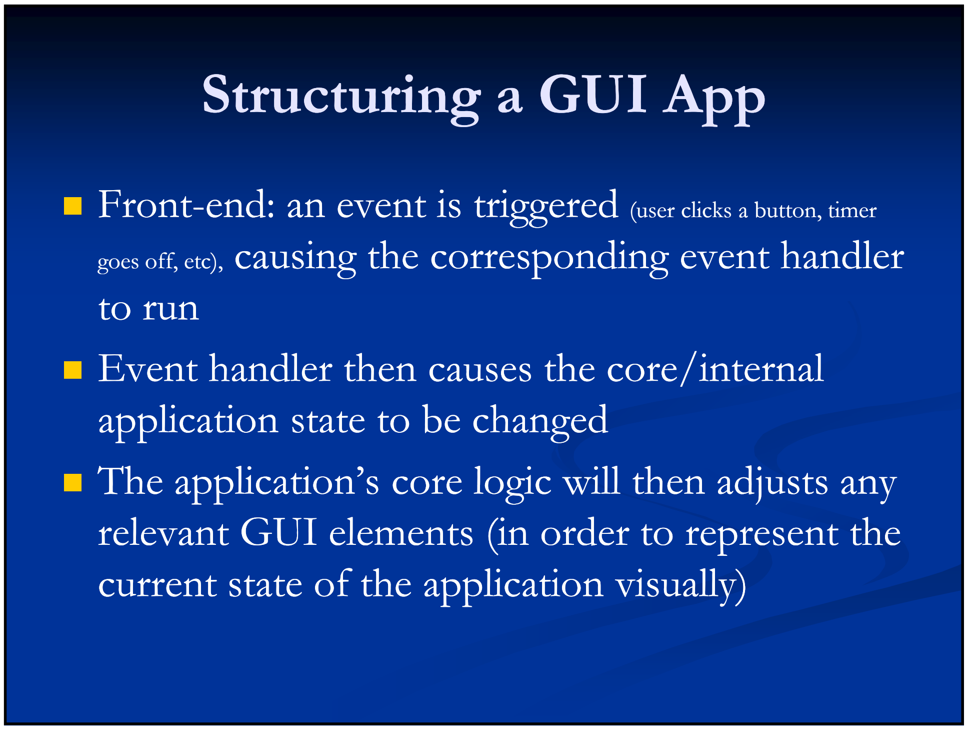
1. Service to allow registration of Event Handling routines
2. GUI elements provide useful default behavior
   - Ex: 'depress when clicked' animation, without programmer doing any work
3. GUI elements can be customized
   - Ex: I want <u>this</u> button to *also* change colors when depressed (then revert to normal color when the user releases the mouse button)
   - Ex: Custom event handling routines

Attributes of a good GUI: Programmer's Perspective

4. GUI elements maintain useful state information
   - Ex: slider bar remembers where the 'thumb' is
   - The application's core logic can poll the element for this info
5. GUI Elements each have a unique ID
   - So that our application's core logic can access the GUI elements
6. GUI framework provides a easy to use, but sophisticated, set of types for control variables
   - In Object Oriented programming, this would be a set of classes, so that one can make use of the Button class's full-featured, yet easy to use, functionality

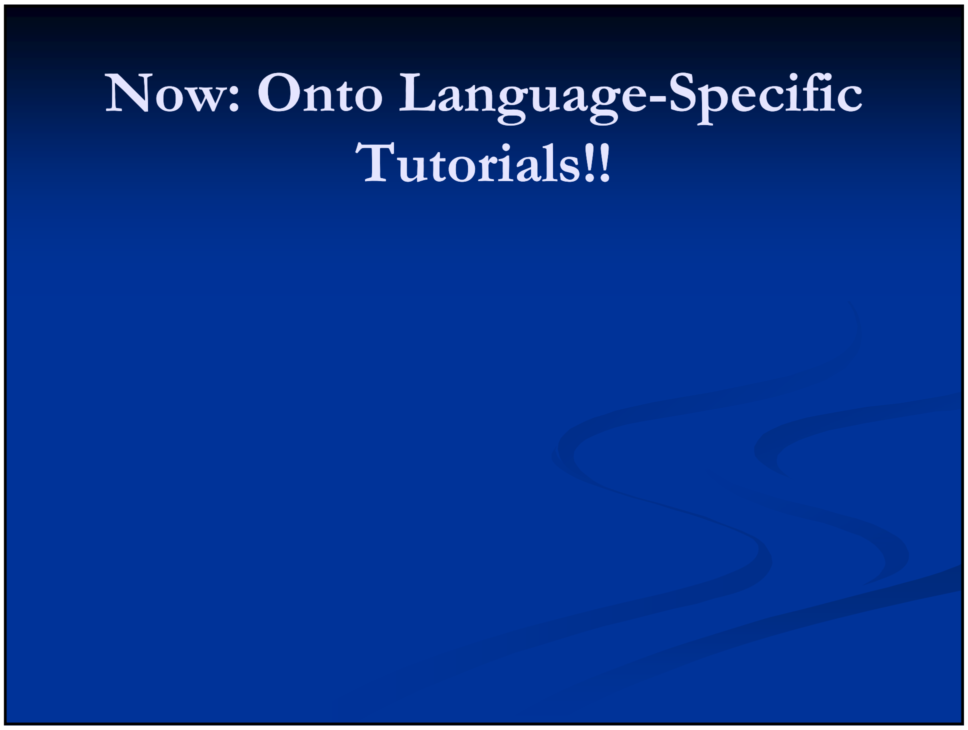
How to create a GUI app

1. Layout the front-end
   - Old School: map stuff out on graph paper, then hand-code all the GUI elements
   - Current: Use a GUI builder program that presents you with a 'view of your program', that allows you to 'draw' the buttons, check boxes, etc, etc directly onto your window(s)
2. Connect the front-end to the back-end
   - Linkage: external modules compiled in at link-time, vs. the IDE modifying the same files as the programmer

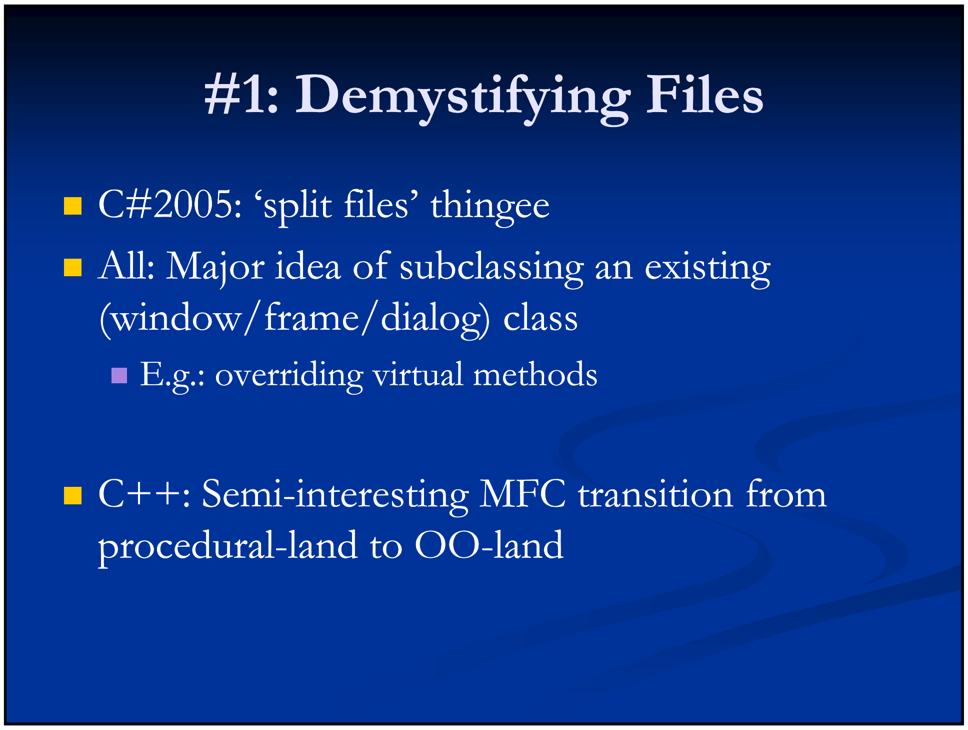


Structuring a GUI App

- Front-end: an event is triggered (user clicks a button, timer goes off, etc), causing the corresponding event handler to run
- Event handler then causes the core/internal application state to be changed
- The application's core logic will then adjusts any relevant GUI elements (in order to represent the current state of the application visually)

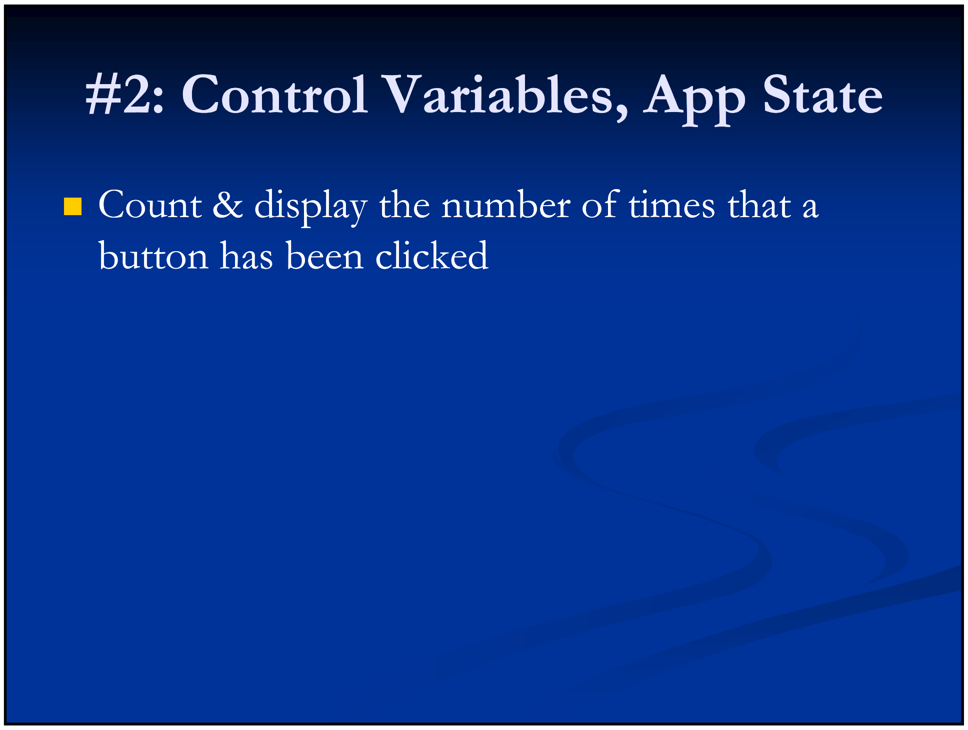


Now: Onto Language-Specific Tutorials!!



#1: Demystifying Files

- C#2005: ‘split files’ thingee
- All: Major idea of subclassing an existing (window/frame/dialog) class
  - E.g.: overriding virtual methods
- C++: Semi-interesting MFC transition from procedural-land to OO-land



#2: Control Variables, App State

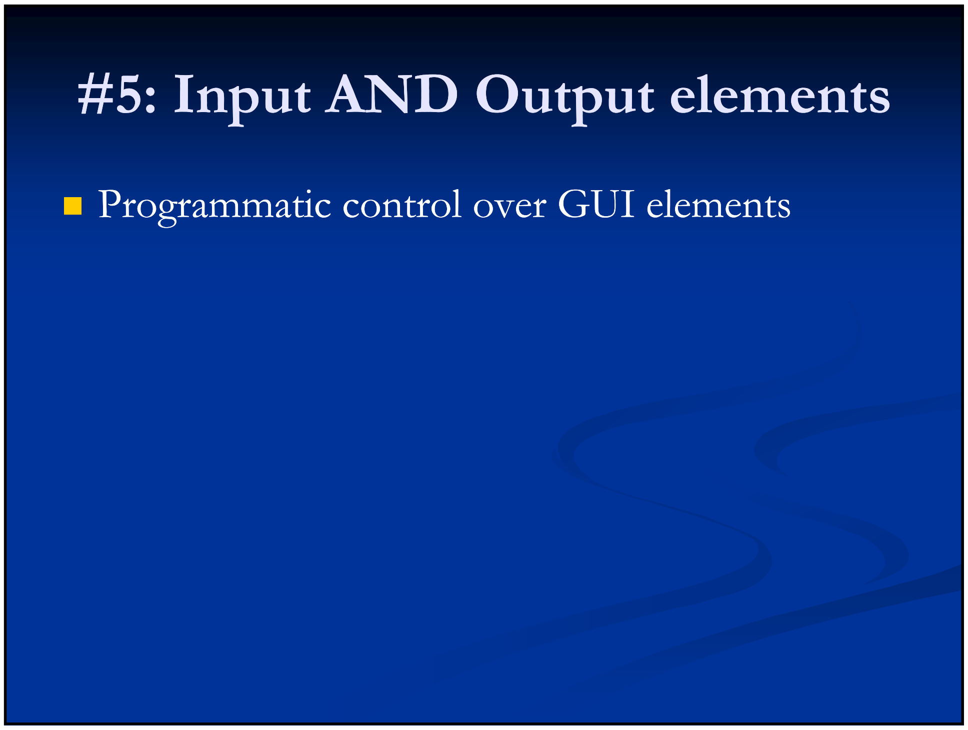
- Count & display the number of times that a button has been clicked

#3: More Complex Elements: Slider Bars

- C++: 'binding' between raw variables, and the GUI elements

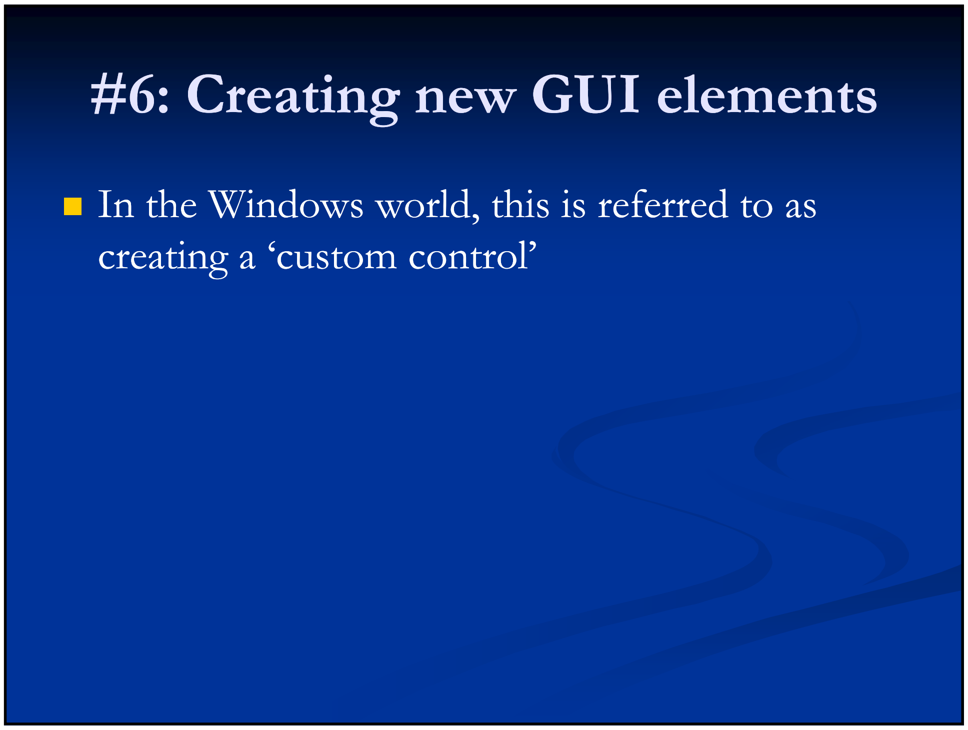
#4: Events not caused by the user

- Timer event: every X units of time, this event will be triggered (by the OS)
  - There is a limit to how often this can be run



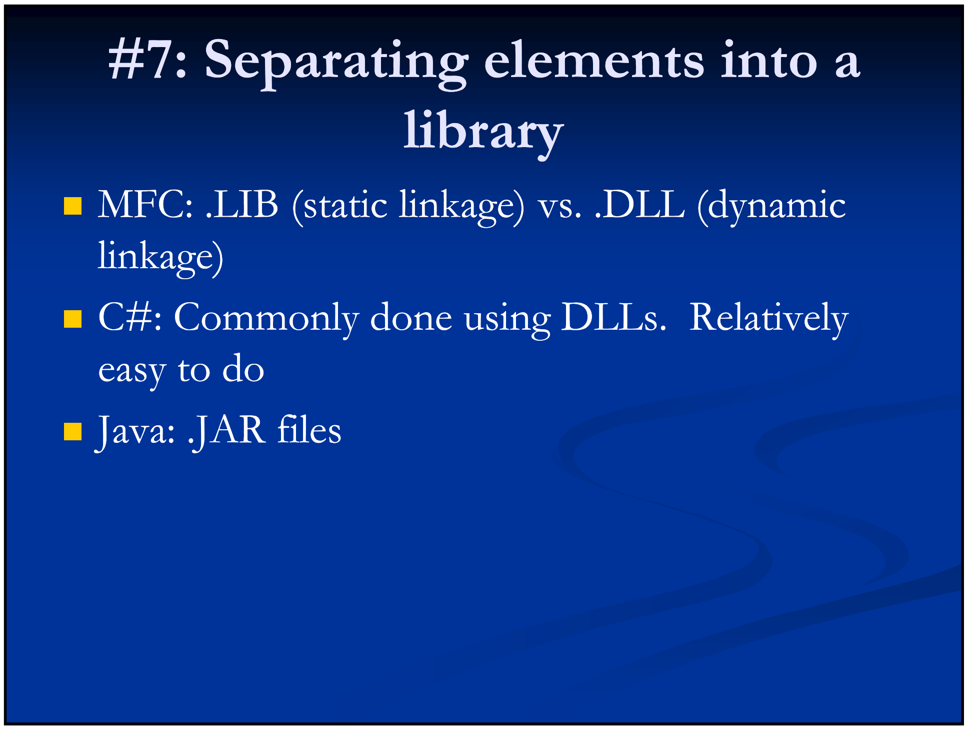
#5: Input AND Output elements

- Programmatic control over GUI elements



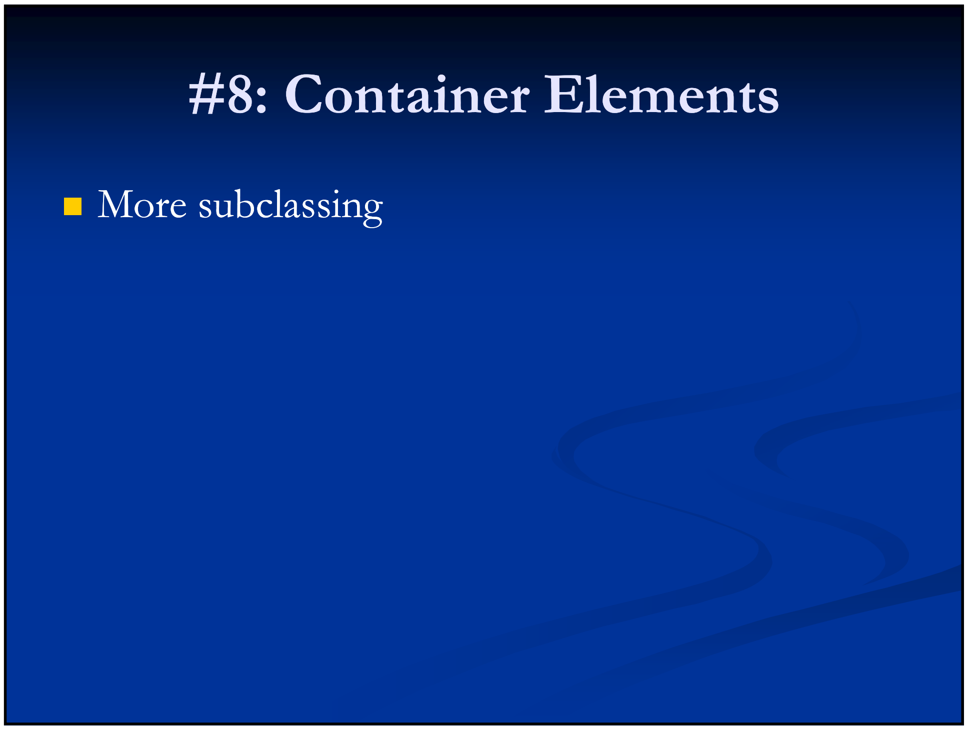
#6: Creating new GUI elements

- In the Windows world, this is referred to as creating a 'custom control'



#7: Separating elements into a library

- MFC: .LIB (static linkage) vs. .DLL (dynamic linkage)
- C#: Commonly done using DLLs. Relatively easy to do
- Java: .JAR files



#8: Container Elements

- More subclassing