Intro to MFC

- What is MFC?
- Event-driven programming
- Well-mannered GUI programs
- Application frameworks
- MVC
- Document-view architecture

Note: some of this presentation is adapted from lecture notes from an MFC course given at Stanford by Patrick Young.

What is MFC?

- Microsoft Foundation Classes (MFC) is a C++ class library for doing Windows programming
- Other option is using Win32 API
  - API = application programmer interface
  - also known as programming with SDK
    - SDK = standard development kit
- Win32 API is a C-based library of functions / types
- MFC is C++ based: uses classes, inheritance
- More on structure of MFC later ...

Console vs. event-driven programming

- GUI programs have a fundamentally different structure than console-based programs
  - GUI = graphical user interface
- console-based program:

```
ask user for some input;
do some processing;
print some output;
ask user for some more input;
etc.
```

  - application programmer controls when input and output can happen
- GUI program model: the user is in control

Well-mannered GUI programs

- GUI model is: user should be able to give any input at any time. E.g.,
  - click on a button
  - close a window
  - drag to draw
- So, don't hog the processor.
- Means your program can't go off and do some 20 minute or 20 second operation.
  - for computation intensive operations, use time-slicing or threads
- Means your program cannot block to wait for input (a la **`scanf`** or **`cin >>`**)
  - only get input or send output via events

Event-driven programming

- structure GUI programs to respond to user *events*
- events are: mouse clicks, mouse moves, keystrokes, etc.
  - in MFC parlance, usually called *messages*
- Main control structure is an event loop:

```
while (1) {
 wait for next event
 dispatch event to correct GUI component
}
```

  - this code is always the same, so it's handled by MFC
- You just write the code to respond to the events.
  - functions to do this are called *message handlers* in MFC

More on well-mannered GUI programs

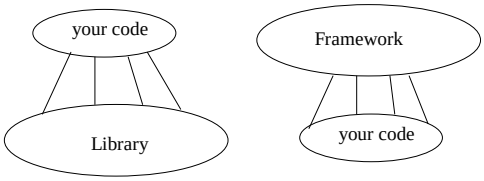
- Application is responsible for making sure display is up-to-date.
- For example:
  - user moves another window in front of yours
  - then moves it away again (yours is exposed)
  - unless you now draw something in your window, formerly exposed part is blank.
- Event-driven:
  - event: view became visible
  - application's response: redraw stuff in the window

GUI Libraries

- GUI programs involve a lot of code.
- But for many different applications much of the code is the same.
- The common code is part of the library. E.g.:
  - getting and dispatching events
  - telling you when user has resized windows, redisplayed windows, etc.
  - code for GUI components (e.g. look and feel of buttons, menus, dialog boxes, etc.)
- But the library is upside-down: library code calls *your* code.
  - this is called an *application framework*

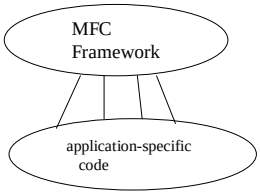
Application Frameworks

- Sometimes called *software architectures*
- Reusable software for a particular domain of applications
- Contrast with class library:



Application Frameworks (cont.)

- In this case:



Application Frameworks: details

- From Horstmann POD text, Ch 19
- general purpose set of classes with pure virtual functions as hooks for more specific versions
- plus, protocol for using the virtual functions:
  - overall control structure given by framework code
  - application writer supplies exact functionality of the methods
- organization of the objects given (data members of framework classes or params to methods)

Application Frameworks: examples

- GUI programming
  - frameworks: MacApp, MFC, Java AWT
- Graphical editors
  - specific applications made from the framework: Drawing, musical composition, CAD
- Compilers (various languages, machines)
- Financial modeling applications
- Simulations

  (we'll discuss later in the semester)

MFC vs. other libraries

- All GUI libraries are top-down like this.
- Using an OO language means we can employ class reuse, templates, and polymorphism.
- MFC provides more in the framework than some other smaller GUI libraries.
  - e.g. "empty" application, get a bunch of menus, and a toolbar (example next page).
  - richer set of components: color-chooser dialog, file browser, and much more.

Empty MFC application

(only part of the window shown)

MFC vs. other libraries (cont.)

- Advantages to application framework:
  - less code for you to write:
    - application gets built faster
    - including less low-level tedious code
  - more reuse of code between applications:
    - we can focus on what's different about our application
  - uniform look and feel to applications produced
    - less frustration for users/customers
- Disadvantages to application framework
  - larger learning curve
  - may produce a slower application
  - may be harder to do exactly what you want
    - e.g., you want a different look-and feel, or you want a new component

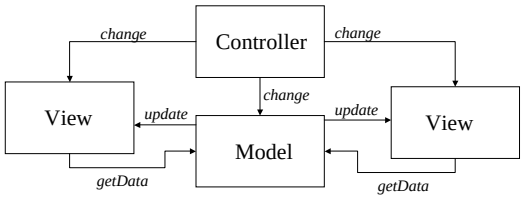
Model-View-Controller Architecture

- Model-View-Controller (MVC)
  - example of an OO design pattern
  - started with Smalltalk
  - a way to organize GUI programs
- Main idea: separate the GUI code from the rest of the application.
- Why?
  - more readable code
  - more maintainable code (more details later)

MVC (cont.)

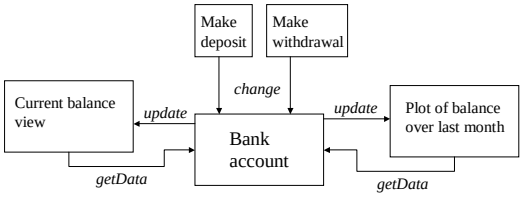
- *Model* classes maintain the data of the application
- *View* classes display the data to the user
- *Controller* classes allow user to
  - manipulate data in the model
  - or to change how a view is displayed
- Modified version: controllers and views combined (MFC does this)

MVC structure



model	view	controller
maintains data	displays current state to user	user manipulates data in a model or how view displayed

MVC example: Bank account



Document-view architecture

- In MFC version of Model-View-Controller:
  - Models are called Document objects
  - Views and Controllers are called View objects
- Example: in Microsoft Word
  - Views:
    - multiple windows open displaying same document
    - different types of views (normal, page layout, outline views)
  - Document:
    - same data regardless of the view above
    - contains text/formatting of Word document

SDI and MDI

- MFC has two flavors of applications
  - SDI = Single document interface
  - MDI = Multiple document interface
- Examples
  - Word uses MDI
    - can have multiple documents open simultaneously
    - may see multiple smaller windows in the larger window
  - Notepad uses SDI
    - only can have one document open at a time
    - view fills up frame of window
- We'll focus on SDI

SDI classes

- Every SDI application has the following four classes:
  - CApp
  - CDoc
  - CView
  - CMainFrame
- Our application will have classes derived from these classes
  - AppWizard will create them automatically when we ask for an SDI MFC application
- The relationship between these classes is defined by the framework.

Four classes of SDI Application

- Instances:
  - Always one App
  - Always one MainFrame
  - Always one Document
  - May have multiple views on Same Document
- Key part of learning MFC:
  - familiarize yourself with these four classes
  - learn what each one does
  - learn when and where to customize each of them

Examples of Customization

- Views
  - OnDraw handles most output (you write; MFC calls)
  - respond to input (write message handlers; MFC calls them)
- Document
  - stores data
  - most of the (non-GUI) meat of the application will be in this object or objects accessible from here
- CMainFrame
  - OnCreate is used to set up control bars
  - (rarely need to customize in practice)
- CWinApp
  - can use to store application-wide data
  - (rarely need to customize in practice)

Benefits of Document/View

- Recall organization:
  - GUI stuff is in View classes
  - non-GUI stuff is in Document (and related) classes
- Benefits: modifiability and readability
  - Can add new Views fairly easily
    - would be difficult if data were closely coupled with its view
    - Examples:
      - spreadsheet: have a grid of cells view; add a bar graph view
      - target a different platform (with different GUI primitives)
  - Can develop each part independently
    - clear interface between the two parts