Widget Toolkits

Lecture 7
CIS 410/510 UI Programming
Winter 2007

User Interface Objects

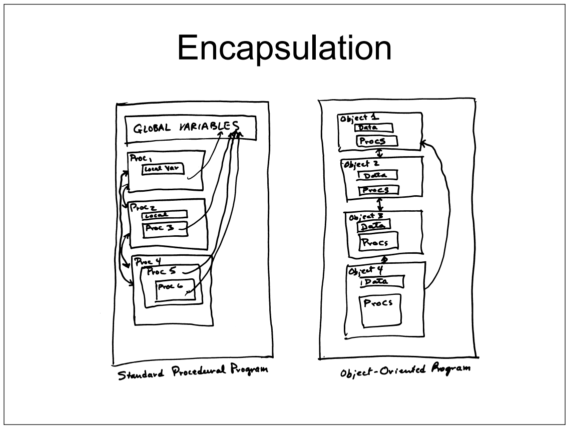
Topics

- Object-Oriented Approach
  - Programming languages
  - UIMS Toolkits
- Resource Definition Files
- Composite Objects
- Multi-Media
- Geometry Management
- Cross-Platform Implementation
- Limitations and Benefits

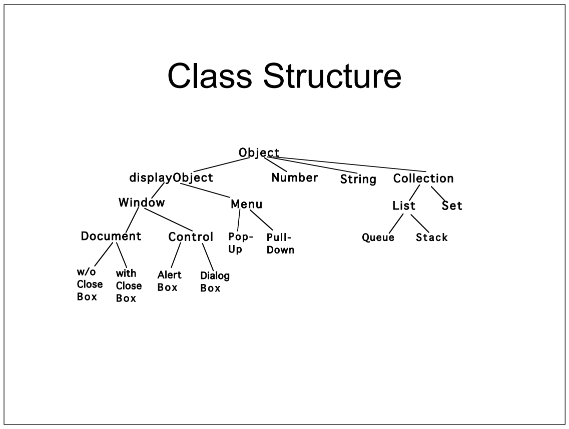
Object-oriented Programming:

Why is it so useful for GUI programming?

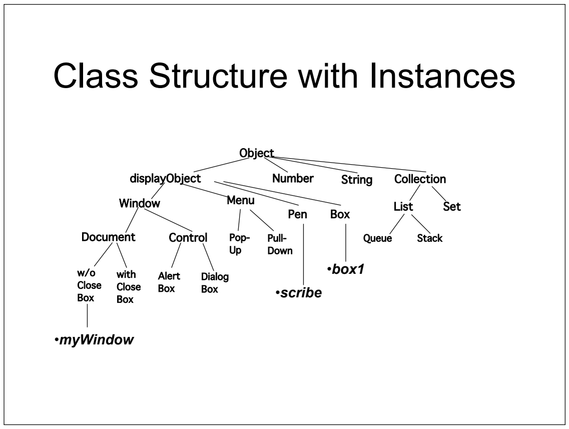
- Encapsulation
  - Each object has its local data and local procedures
  - Creates modularity
  - Programming objects map directly onto graphical, manipulable interface objects
  - Message passing control is event-based paradigm
- Class inheritance
  - Similar objects grouped together at levels of abstraction (Class/subclass relations)
  - Share code through inheritance of similarity, promote reuse of commonly used objects



Encapsulation



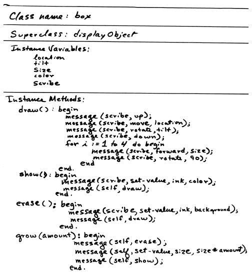
Class Structure



Class Structure with Instances

O-O Programming
Pen Example

- Create Instance of Class
  - message (object, method, name, Xlocation, Ylocation, tilt, size, color)
  - message (box, make-instance, box1, 200, 200, 0, 20, black)
- Display
  - message (box1, show)
- Increase Size
  - message (box1. grow, 30)
- Move
  - message (box1, move, 400, 200)

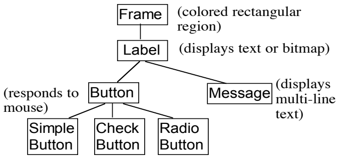


```
Class name : box
Superclass : displayObject
Instance Variables:
      location
      tilt
      size
      color
      scribe
Instance Methods:
  draw() : begin
             message (scribe, up);
             message (scribe, move, location);
             message (scribe, rotate, tilt);
             message (scribe, down);
             for i := 1 to 4 do begin
                  message (scribe, forward, size);
                  message (scribe, rotate, 90);
                end
           end.
  show(): begin
            message(scribe, set-value, ink, color);
            message (self, draw);
          end.
  erase(): begin
             message(scribe, set-value, ink, background);
             message (self, draw);
           end.
  grow (amount): begin
             message (self, erase);
             message (self, set-value, size, size*amount);
             message (self, show);
           end.
```

UI Object-Oriented Programming

- UI objects in O-O language
  - Smalltalk, C++, Java
  - Mapped onto Windowing System
- UI objects in UIMS Toolkits (Widgets)
  - Xtk, Motif, Tcl/Tk
  - Mapped onto Windowing System
  - O-O inheritance often not accessible to programmer
  - Often not extensible
  - Can't interact with each other through programmer

Widget Class Hierarchy
Tcl/Tk Example


Frame
(colored rectangular region)
Label
(displays text or bitmap)
(responds to mouse)
Button
Message
(displays multi-line text)
Simple Button
Check Button
Radio Button

Where do these go?
- Listbox
- Scrollbar
- Scale
- Menu
- Canvas
- Text

Widget Resource Files

- Variable data of a widget stored in a file
- Can be edited by the user & read at run-time by the UIMS when client requests creation of widget
- Independent from application code
- Macintosh model
  - stored in “resource” fork of the program
  - edited by a program called ResEdit
- Client-server model
  - stored by UIMS
  - edited by text editor

Resource File

Xtrinsics Example

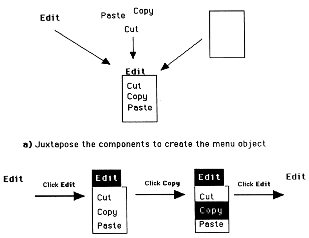
```
#######################################################
# Draw: Class resource file the simple draw program
#######################################################

Draw*commands.columns:                               1
Draw*quit.label:                                     Quit
Draw*drawline.label:                                 Draw Line
Draw*drawrect.label:                                 Draw Rectangle
Draw*movelineright.label:                            Move Line Right
Draw*movelineleft.label:                             Move Line Left
Draw*canvas.xRefName:                                commands
Draw*canvas.xAddWidth:                               True
Draw*canvas.xAttachRight:                            True
Draw*canvas.xAttachLeft:                             True
Draw*canvas.xAttachBottom:                           True
Draw*canvas.xAttachTop:                              True
Draw*canvas.xAttachRight:                            True
```

Widget Composite Objects

- Composite Object can have children
  - not a subclass-class relation, i.e. not specializations
  - instead, part-whole relation

Menus as Composite Objects



a) Juxtapose the components to create the menu object

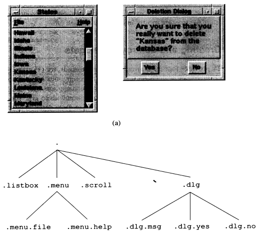
b) The new Menu object in action

More on Composite Objects

- Composite object allows run-time hierarchy in which position of child is specified relative to parent, therefore movement occurs automatically
- “Container” object has size, position, children, but no interaction of its own
  - Example: “Frame” in Tcl/Tk
- Containers can be children of other containers
- Event propagation by parent notification
  - If user generates move event that is not of interest ot a particular object, it gets passed up the hierarchy
  - Example: move to dialog box passed to container which is parent

Composite Object

Tcl/Tk Dialog Box



Are you sure that you really want to delete "Kansas" from the database?

(a)

(b)

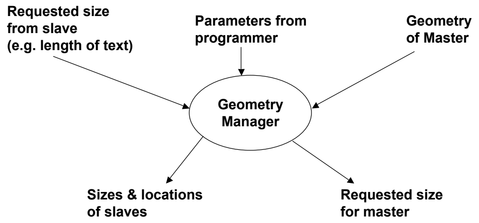
Integrating Multimedia into Toolkit Widgets

- Requires widget to support multiple media technologies such as audio, computer-generated animation, and full-motion video
- Example
  - Window with a set of buttons for controlling a sub-window of full-motion video
  - Functions: Stop, Play, Fast Forward, Reverse, Single Frame
- At the moment, this is very much a research issue!

Geometry Management

- Related to Composite Object
- Some toolkits have automatic geometry management of children by parent
  - Parent determines overall size and position
  - Parent determines size of child within a range
  - If child is parent of embedded objects, it informs them of new size, and so on
  - Sometimes child and parent may negotiate
    - child gives minimum size
    - Example: if text field is too small may change to icon

The Geometry Manager


Requested size from slave (e.g. length of text)
Parameters from programmer
Geometry of Master
Geometry Manager
Sizes & locations of slaves
Requested size for master

Geometry Management cont.

- Form of constraint-based programming
- Frees application from responsibility for placing objects
  - But lose design control for usability
- Example: Tcl/Tk “packer”is row/column manager
- May be difficult to understand and program
  - Example: Java’s GridBagLayoutManager

Tcl/Tk Geometry Managers

- “packer” for layouts with rows and columns
- “placer” for layouts with fixed position slaves relative or absolute to master
- “grid” part of the canvas widget, allows mixing embedded widgets with other elements such as lines and text

Widget Cross-Platform Look & Feel

- Each virtual widget implemented in windowing system widgets of platform
- Uses geometry manager
- May cause inconsistencies in usability
  - Example: multiple mouse buttons
  - Example: layout of icon panel on different sized screen
- Frequently buggy!

Benefits of O-O Approach

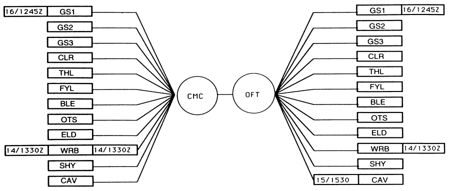
- Reuse improves programming productivity
- Reuse improves standardization of UI look and feel
- Natural cognitive mapping to concrete objects improves programming productivity
- Modularity and inheritance reduce programming errors

Limitations of O-O Approach

- May be difficult or impossible to change UIMS Toolkit widgets
  - Example: Drawing diagonal lines

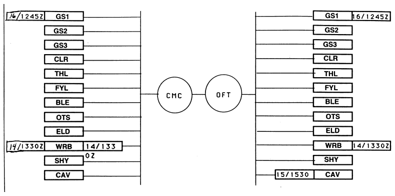
Drawing Diagonal Lines

What you want



Drawing Diagonal Lines

What you get: Athena Widget Toolkit



Widgets based on windows with sides parallel to screen

Does it mean the same?

Limitations of O-O Approach

cont.

- UIMS Toolkits may not be first-class O-O
  - hard to integrate into client application
- Hard to debug
  - May not know the inheritance path
  - Problems of multiple inheritance more confusing
- Learning difficult
  - Often hard to choose widget needed because behavior not obvious from class name
  - Complex: must learn all classes and their methods
    - Smalltalk has 200+ classes each with average of 4 methods

Summary

- O-O Programming is a natural match for UI programming
  - object mapping
  - event-based control through messages
  - reuse improves productivity and reduces bugs
  - Model-View-Controller
- Becomes more limited as gains complexity
- Extensions to O-O paradigm motivated by UI
  - Composite objects
  - Geometry management
  - Constraint-based programming