Human-Computer Interaction
IS4300

I3: Ethnography

T2: Requirements Analysis Review...

User Analysis

Task Analysis

Problem Scenarios

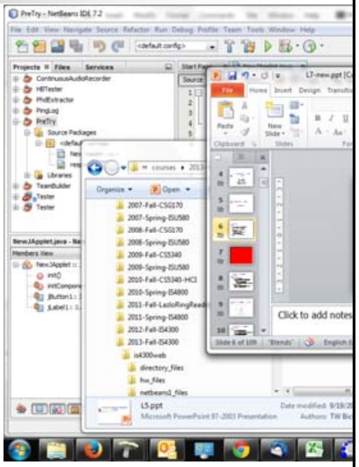
Usability Criteria

Implementation Support

Dix Chapter 8

Exercise

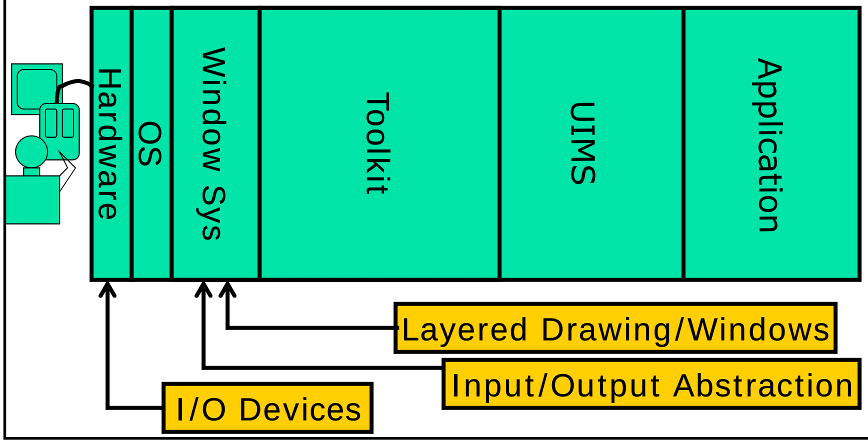
- Your engineers just developed a new desktop computer.
- They give you the following primitives:
  - drawPixel(x,y,color)
  - readMouseX(), readMouseY(), readMouseButton(), readKey()
- They ask you to implement this:



Levels of Abstraction in UI Software

- Windowing systems
  - central environment for both the programmer and user of an interactive system, allowing a single workstation to support separate user-system threads of action simultaneously.
- Interaction toolkits
  - abstract away from the physical separation of input and output devices, allowing programmer to describe behaviors of objects at a level similar to how the user perceives them.
- User interface management systems
  - Allows designer and programmer to control the relationship between the presentation objects of a toolkit with their functional semantics in the actual application.
- Application

Levels of Abstraction



Elements of windowing systems

Device independence

programming the abstract terminal device drivers

image models for output and (partially) input

- pixels
- PostScript (MacOS X, NextStep)
- Graphical Kernel System (GKS)
- Programmers' Hierarchical Interface to Graphics (PHIGS)

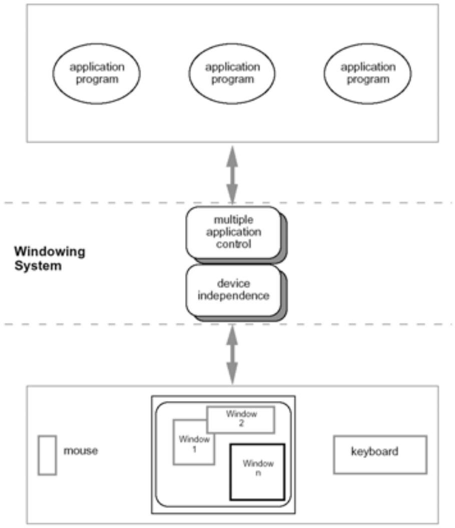
Resource sharing

achieving simultaneity of user tasks

window system supports independent processes

isolation of individual applications

roles of a windowing system



Architectures of windowing systems

three possible software architectures

- all assume device driver is separate
- differ in how multiple application management is implemented

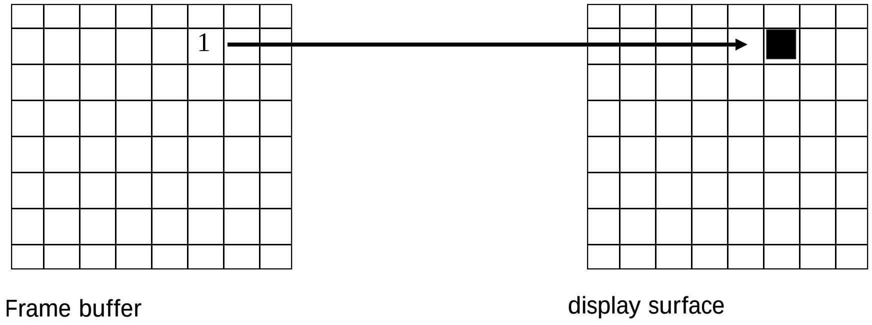
1. each application manages all processes
   - everyone worries about synchronization
   - reduces portability of applications
2. management role within kernel of operating system
   - applications tied to operating system
3. management role as separate application
   maximum portability

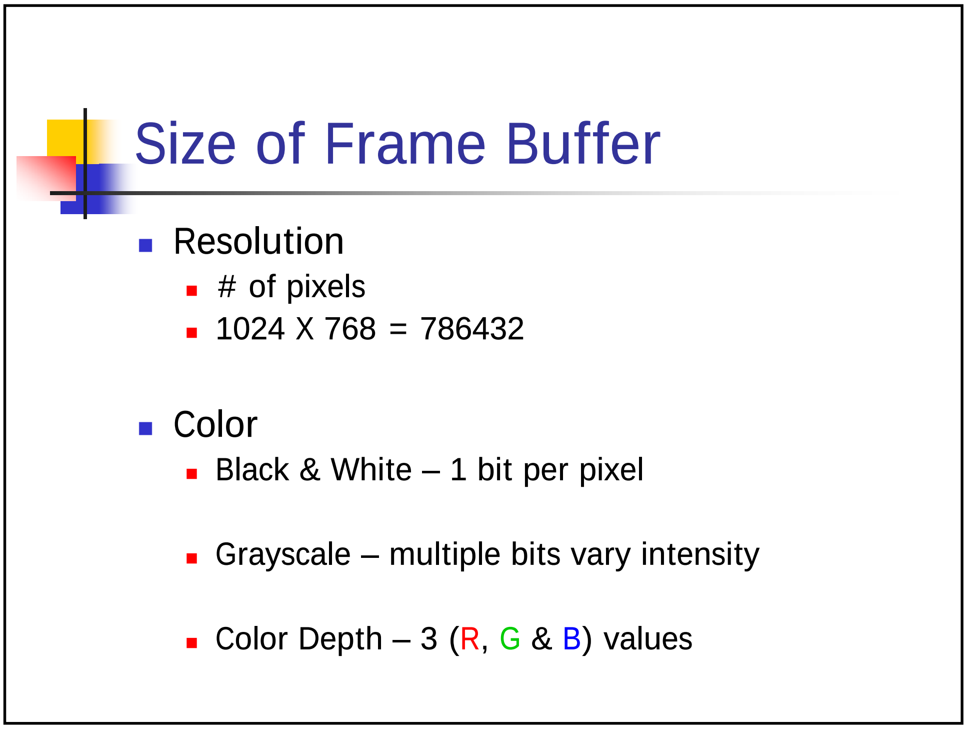
Human Perception and Displays

- Split a picture into a collection of small dots and we can reconstruct it.
  - pixels and resolution
- Present consecutive “frames” of a dynamic scene and we can smooth it.
  - > 15 frames per second refresh rate

Painting a picture

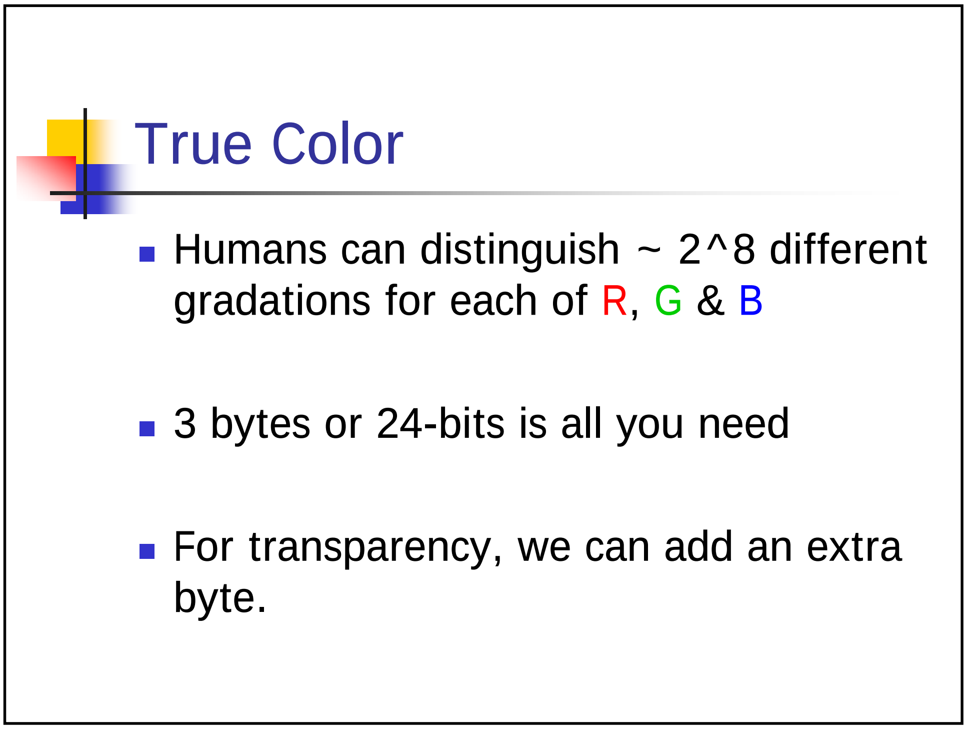
- Each memory cell controls 1 pixel





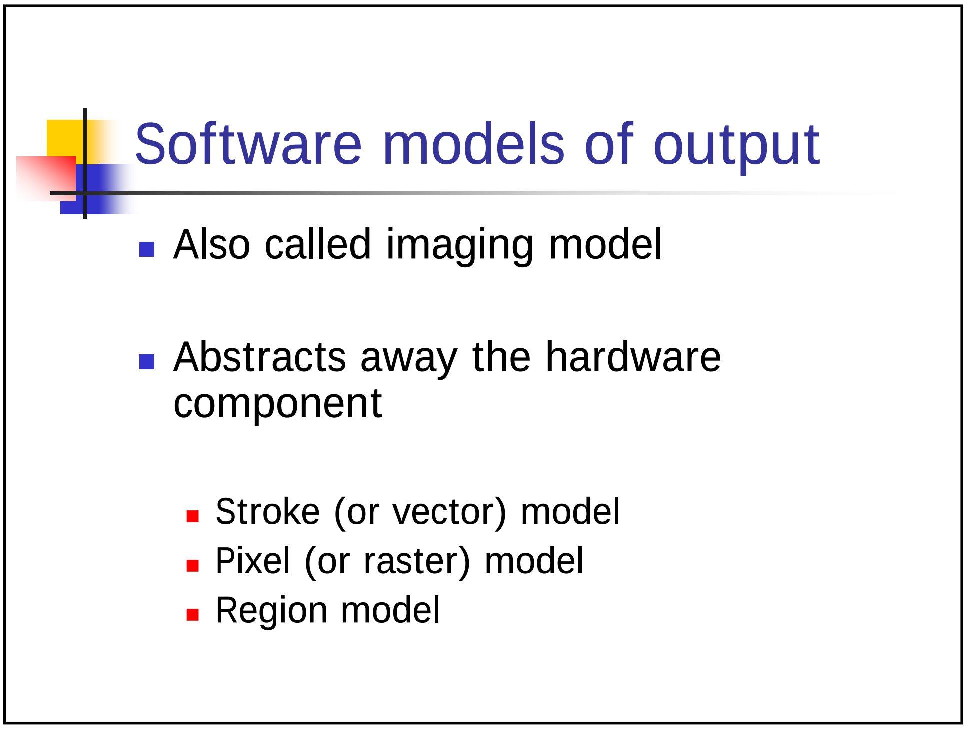
Size of Frame Buffer

- Resolution
  - # of pixels
  - 1024 X 768 = 786432
- Color
  - Black & White – 1 bit per pixel
  - Grayscale – multiple bits vary intensity
  - Color Depth – 3 (R, G & B) values



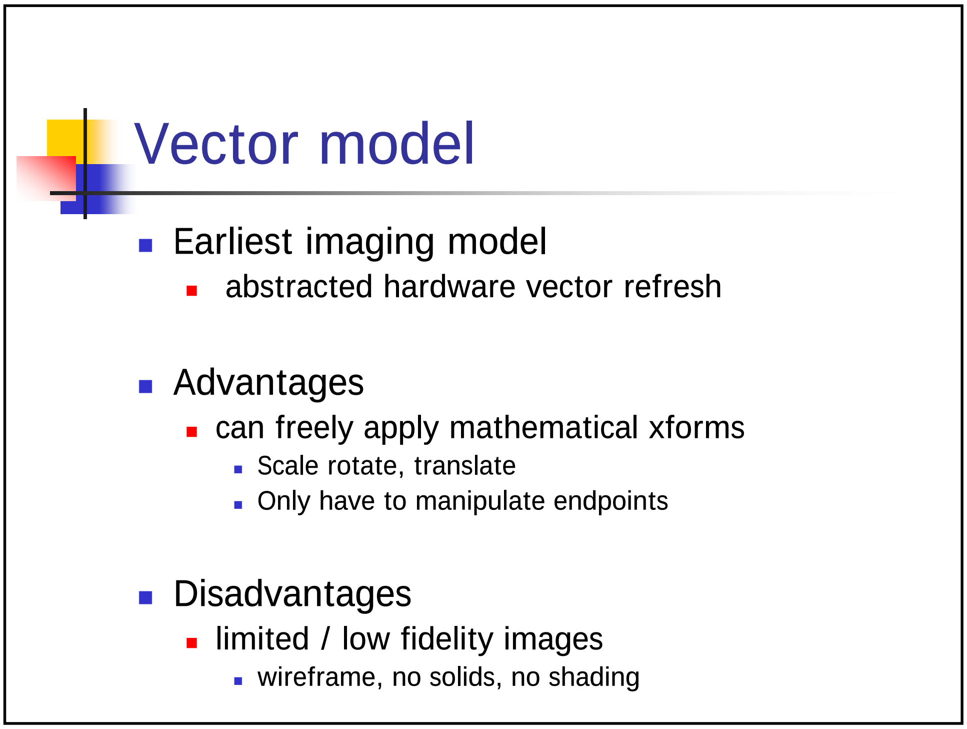
True Color

- Humans can distinguish ~ 2^8 different gradations for each of R, G & B
- 3 bytes or 24-bits is all you need
- For transparency, we can add an extra byte.



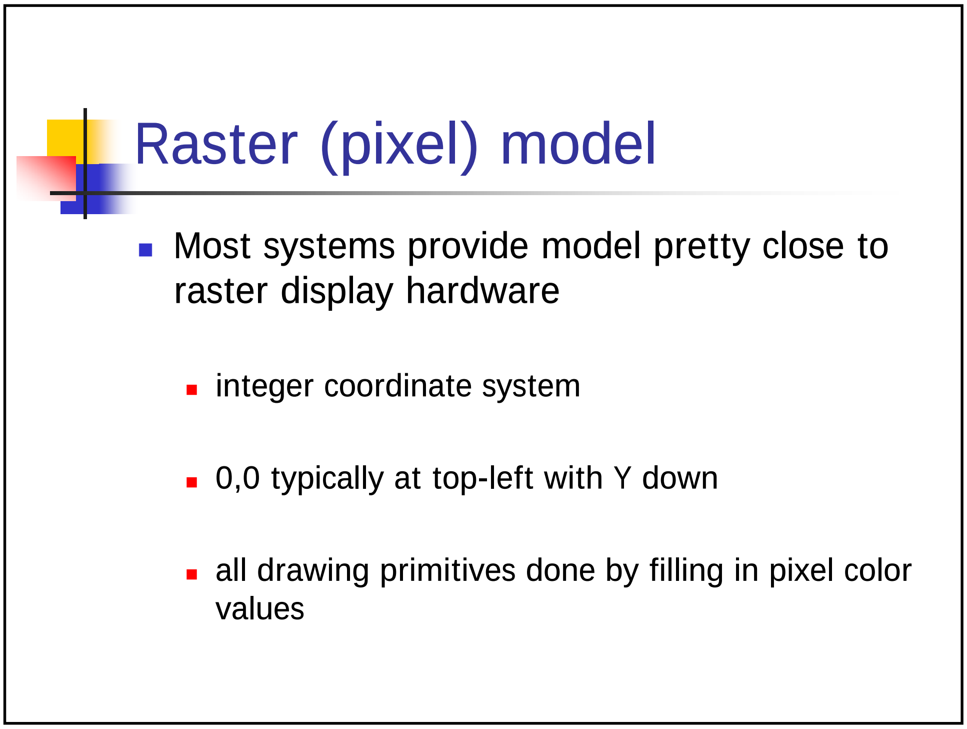
Software models of output

- Also called imaging model
- Abstracts away the hardware component
  - Stroke (or vector) model
  - Pixel (or raster) model
  - Region model



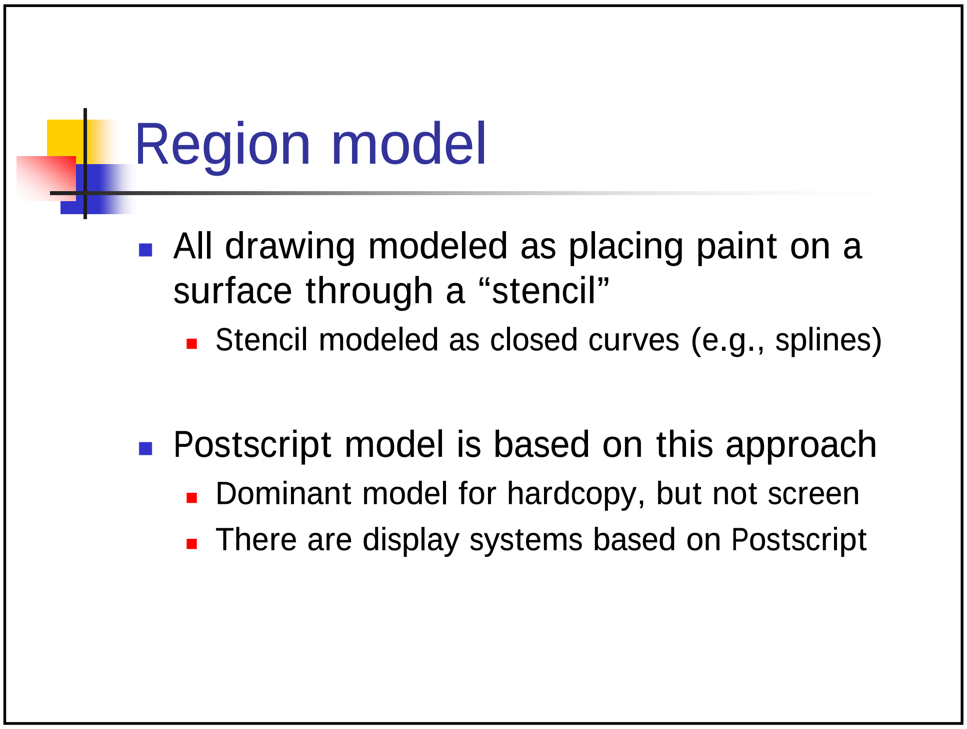
Vector model

- Earliest imaging model
  - abstracted hardware vector refresh
- Advantages
  - can freely apply mathematical xforms
    - Scale rotate, translate
    - Only have to manipulate endpoints
- Disadvantages
  - limited / low fidelity images
    - wireframe, no solids, no shading



Raster (pixel) model

- Most systems provide model pretty close to raster display hardware
  - integer coordinate system
  - 0,0 typically at top-left with Y down
  - all drawing primitives done by filling in pixel color values



Region model

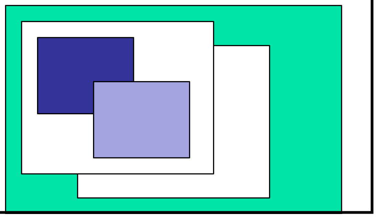
- All drawing modeled as placing paint on a surface through a “stencil”
  - Stencil modeled as closed curves (e.g., splines)
- Postscript model is based on this approach
  - Dominant model for hardcopy, but not screen
  - There are display systems based on Postscript

Region model

- Advantages
  - Resolution & device independent
    - does best job possible on avail HW
    - Don’t need to know size of pixels
  - Can support full transformations
    - rotate & scale
- Disadvantages
  - Slower
    - Less and less of an issue
    - But interactive response tends to be dominated by redraw time
  - Much harder to implement

A Hierarchy of Windows

- Most UIs are described as a collection of hierarchically ordered windows or elements (called interactors).
  - Top of “tree” or root is whole display
- Geometric relationships (containment, overlap) are important.

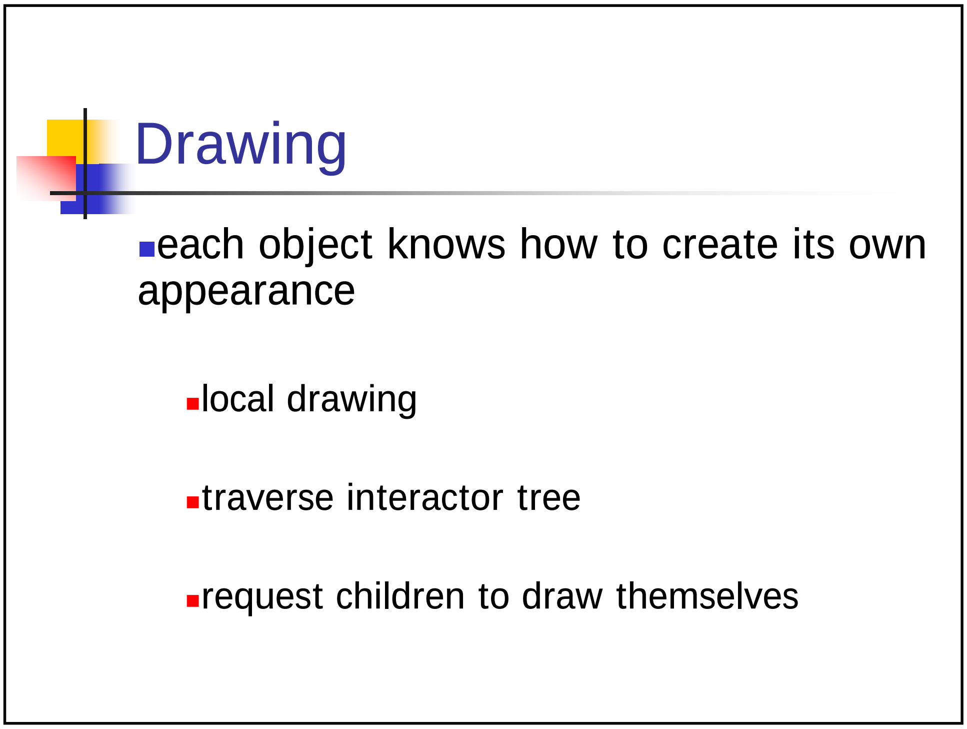


Output and the Interactor Tree

- output is organized around the tree structure
  - each object has own behaviors & states
    - can draw itself
    - can do other tasks
    - knows own capabilities and those of children
  - generic tasks are specialized to specific subclasses

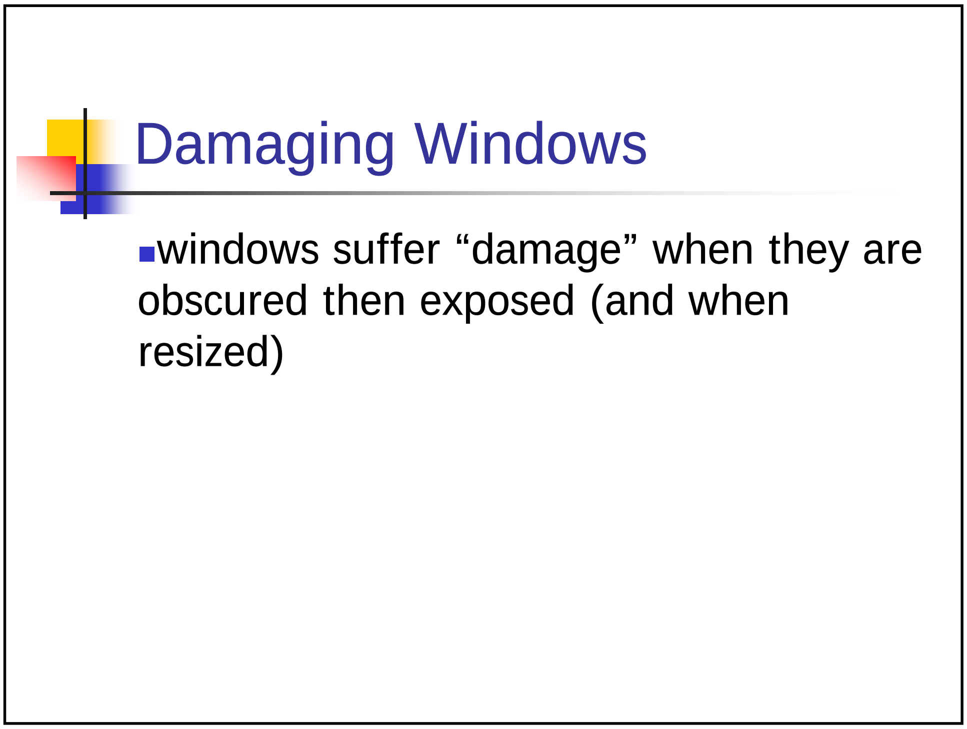
Output Tasks in Windowing Systems

- 3 main tasks
  - draw / redraw
  - damage management
  - layout



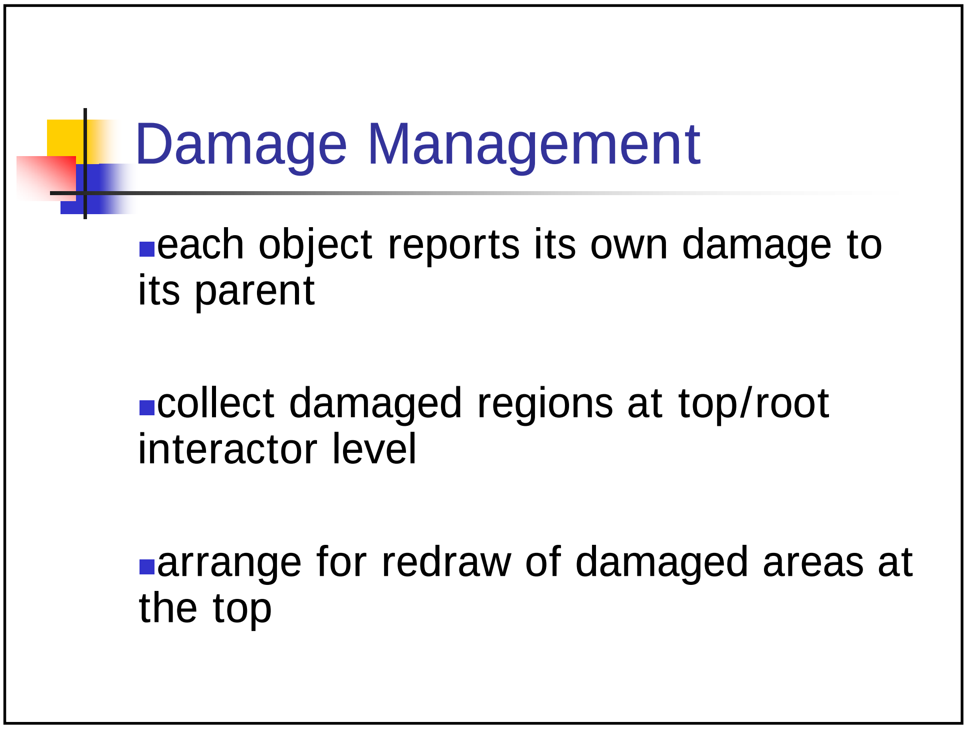
Drawing

- each object knows how to create its own appearance
  - local drawing
  - traverse interactor tree
  - request children to draw themselves



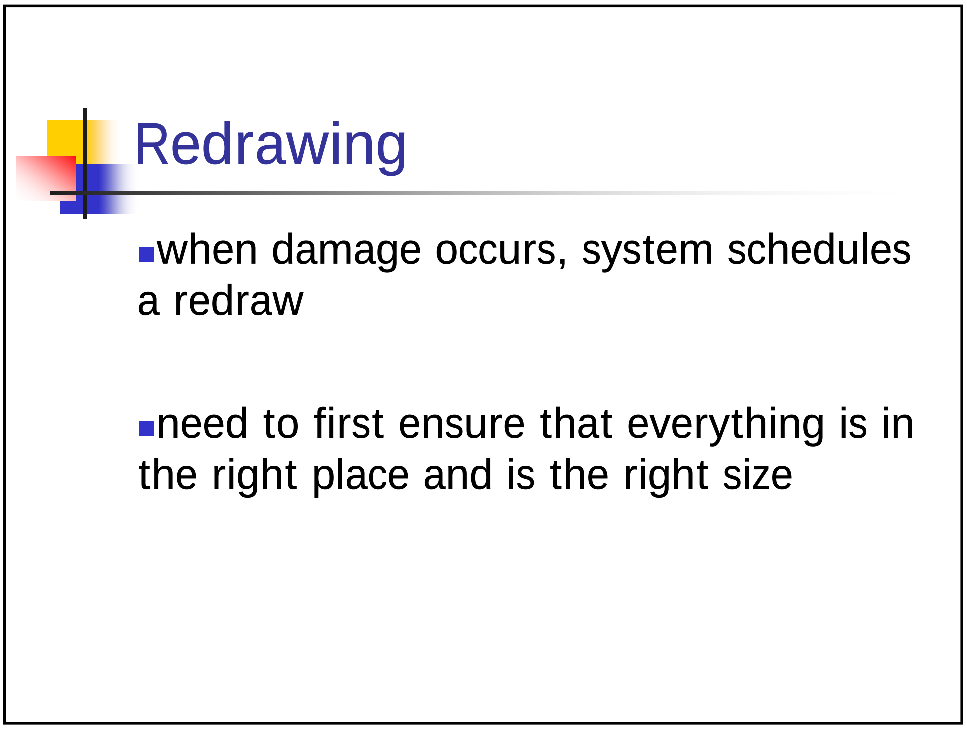
Damaging Windows

- windows suffer “damage” when they are obscured then exposed (and when resized)



Damage Management

- each object reports its own damage to its parent
- collect damaged regions at top/root interactor level
- arrange for redraw of damaged areas at the top



Redrawing

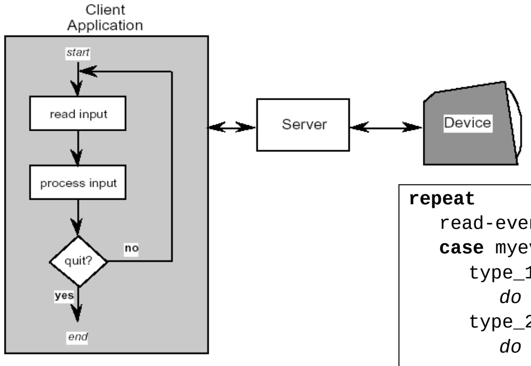
- when damage occurs, system schedules a redraw
- need to first ensure that everything is in the right place and is the right size

Drawing issue

- cannot size and position as we draw
- look of first child might depend on last child’s size
  - arbitrary dependencies
  - may not follow redraw order
- need to compute layout prior to starting to draw

Programming the application

read-evaluation loop



```
repeat
   read-event(myevent)
   case myevent.type
      type_1:
         do type_1 processing
      type_2:
         do type_2 processing
      ...
      type_n:
         do type_n processing
   end case
end repeat
```

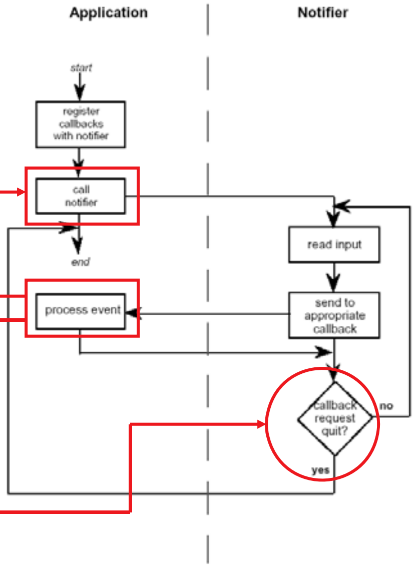
Programming the application

notification-based

```
void main(String[] args) {
   Menu menu = new Menu();
   menu.setOption(“Save”);
   menu.setOption(“Quit”);
   menu.setAction(“Save”,mySave)
   menu.setAction(“Quit”,myQuit)
      ...
}

int mySave(Event e) {
   // save the current file
}

int myQuit(Event e) {
   // close down
}
```



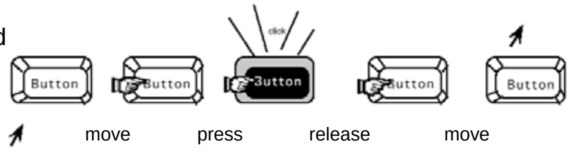
Read-eval loop vs. Notifications

- Pros & Cons of each?

Using interaction toolkits

Interaction objects

- input and output intrinsically linked



Toolkits provide this level of abstraction

- programming with interaction objects (or widgets, gadgets)
- promote consistency and generalizability through similar look and feel
- amenable to object-oriented programming

Objects and the UI

- Why are they so well suited?
  - Natural metaphor (direct manipulation)
  - Encapsulation (info hiding)
  - Class-instance
  - Subclassing
  - Prototype instances
  - Message passing

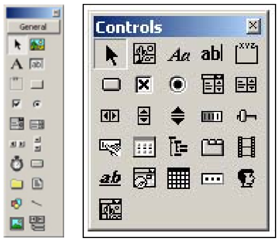
Standard UI Widgets
The “Macintosh 7”

1984

- Button
- Slider
- Pulldown menu
- Check box
- Radio buttons
- Text entry fields
- File pick/save

Influence on today’s GUIs

- The Macintosh 7 have become *standard* (common) interaction techniques
- MFC as an example



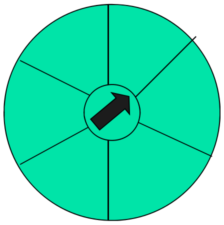
- Sure enough, inside the Swing toolkit as well

The good & the bad

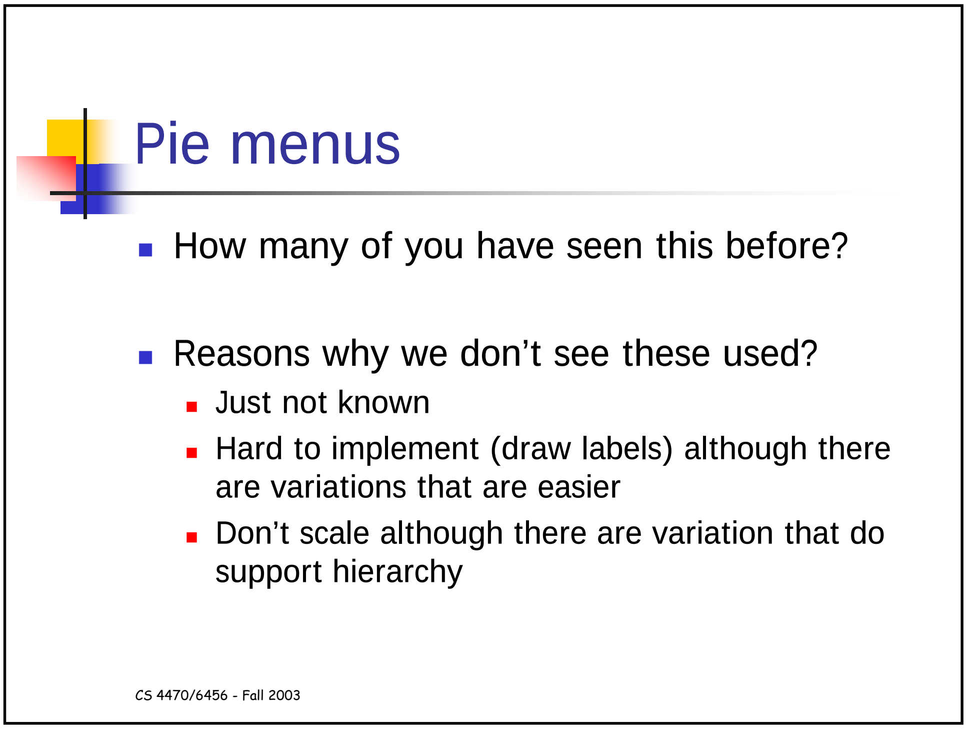
- Collection of good interaction techniques that work well
  - uniformity is good for usability
- Significant stagnation
  - Failing to customize interaction techniques to tasks

Example of non-standard widget: Pie menus

- A circular pop-up menu with “dead area” at center
  - basically only angle counts



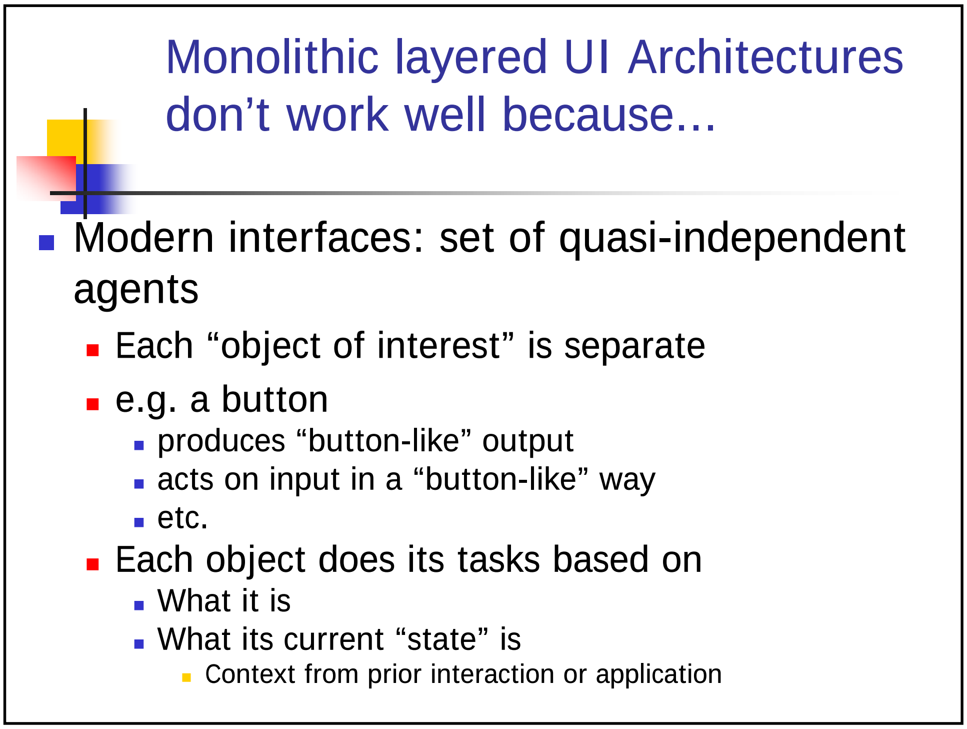
- What are Fitts’ law properties?
  - minimum distance to travel
  - minimum required accuracy (dependent on # of options)
  - very fast (dependent on # of options)



Pie menus

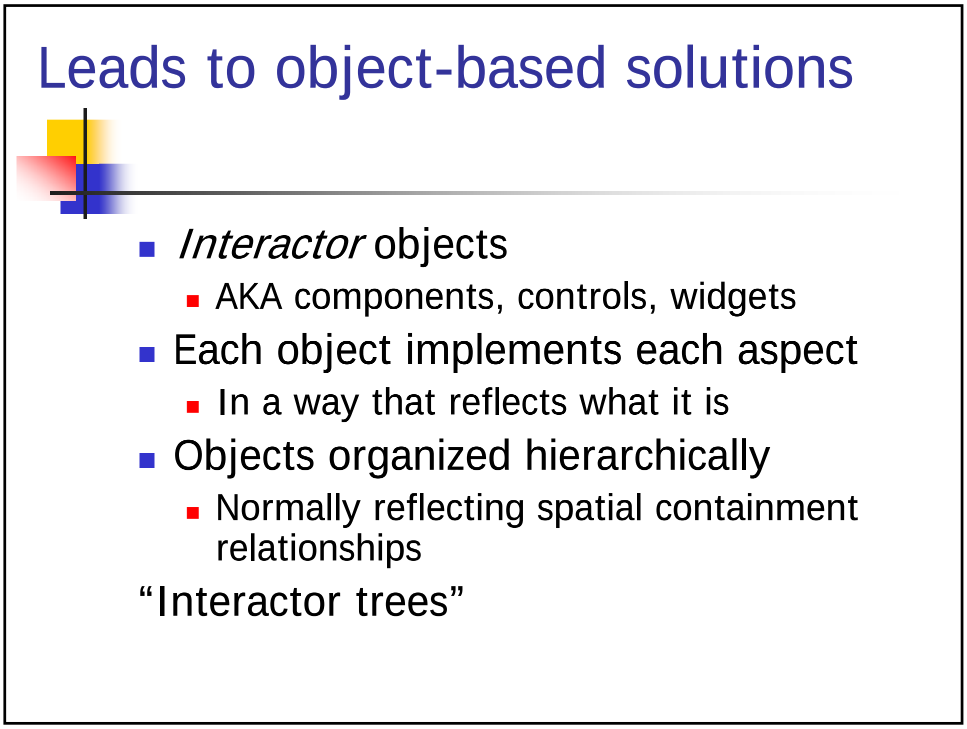
- How many of you have seen this before?
- Reasons why we don’t see these used?
  - Just not known
  - Hard to implement (draw labels) although there are variations that are easier
  - Don’t scale although there are variation that do support hierarchy

CS 4470/6456 - Fall 2003



Monolithic layered UI Architectures don’t work well because…

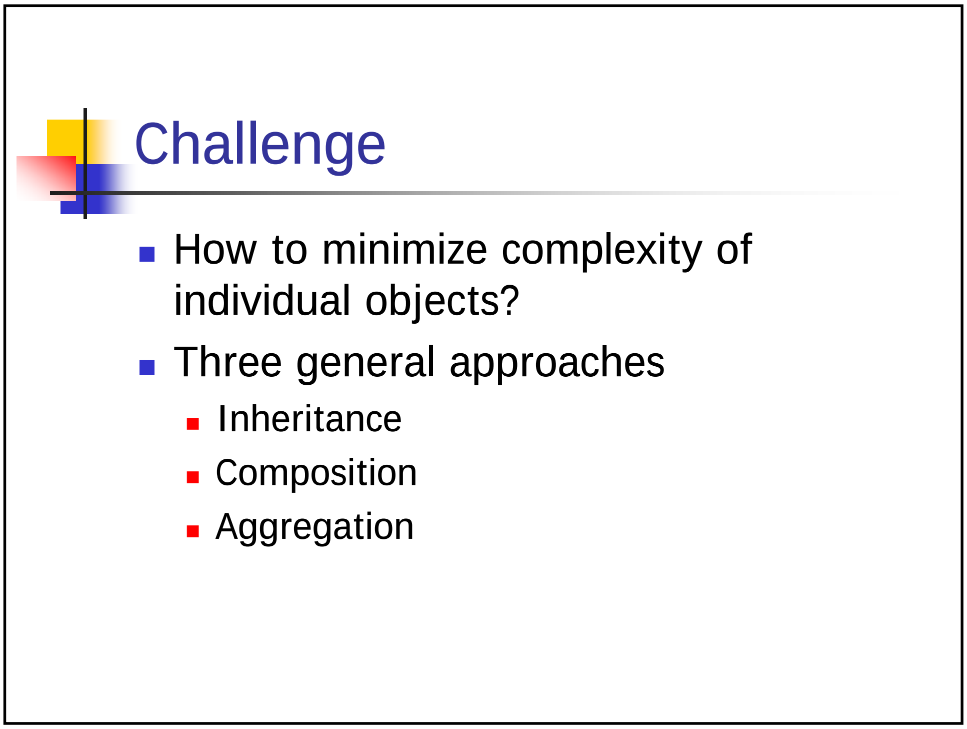
- Modern interfaces: set of quasi-independent agents
  - Each “object of interest” is separate
  - e.g. a button
    - produces “button-like” output
    - acts on input in a “button-like” way
    - etc.
  - Each object does its tasks based on
    - What it is
    - What its current “state” is
      - Context from prior interaction or application



Leads to object-based solutions

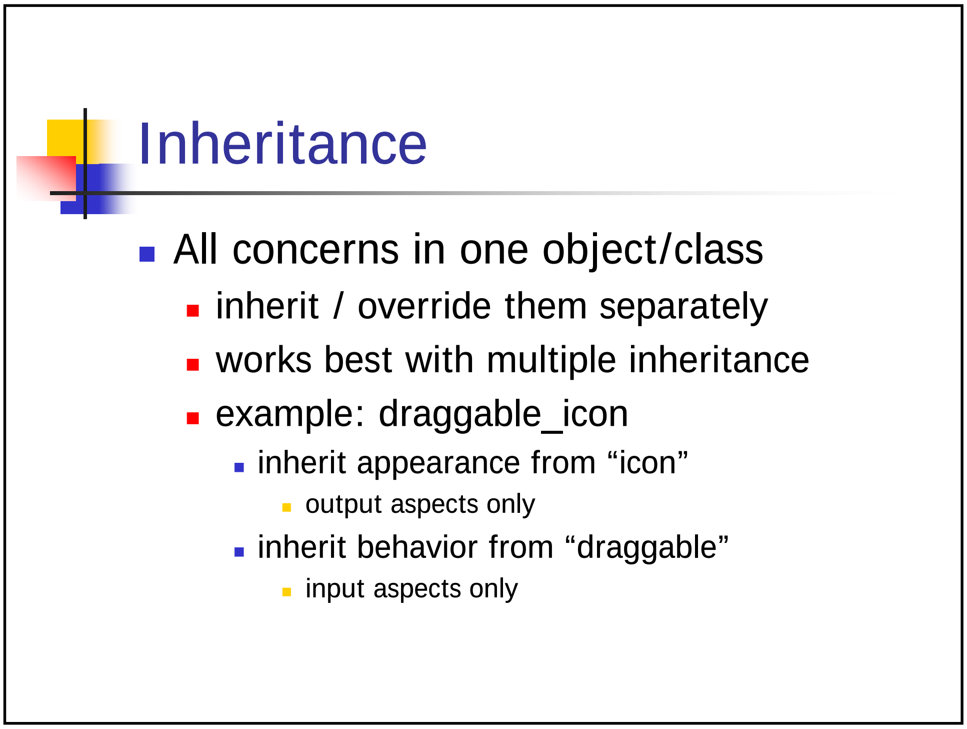
- *Interactor* objects
  - AKA components, controls, widgets
- Each object implements each aspect
  - In a way that reflects what it is
- Objects organized hierarchically
  - Normally reflecting spatial containment relationships

“Interactor trees”



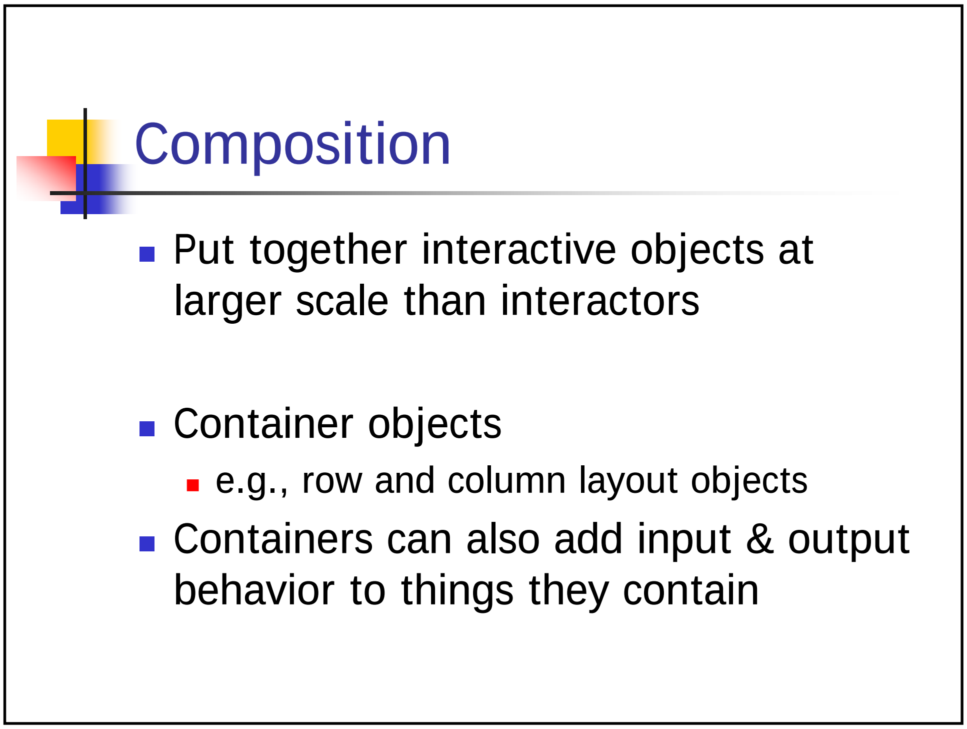
Challenge

- How to minimize complexity of individual objects?
- Three general approaches
  - Inheritance
  - Composition
  - Aggregation



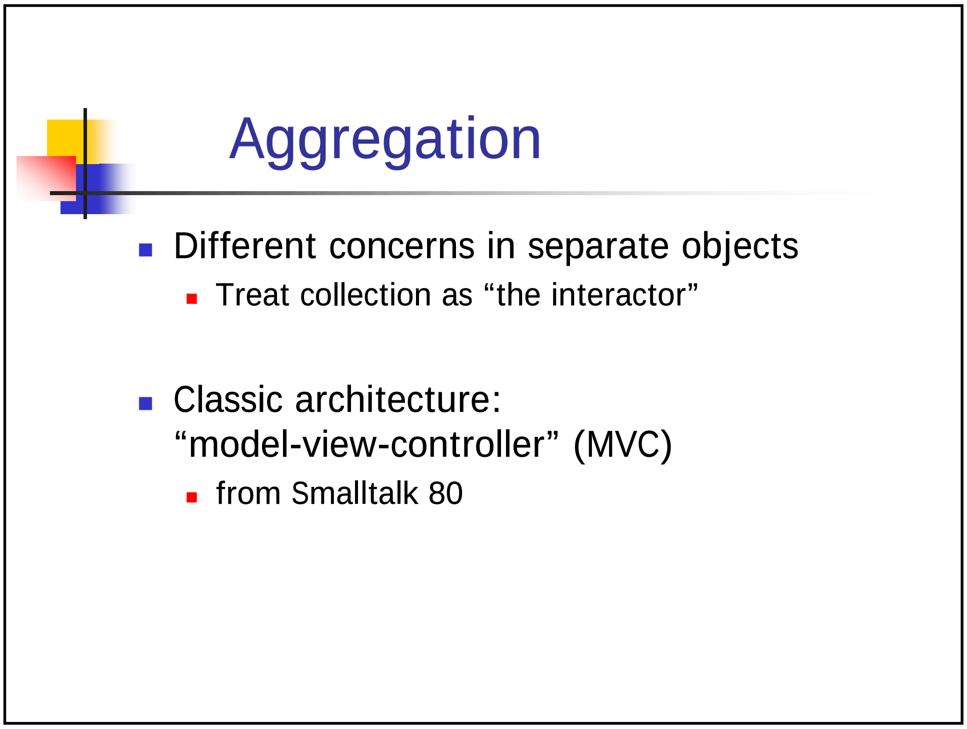
Inheritance

- All concerns in one object/class
  - inherit / override them separately
  - works best with multiple inheritance
  - example: draggable_icon
    - inherit appearance from “icon”
      - output aspects only
    - inherit behavior from “draggable”
      - input aspects only



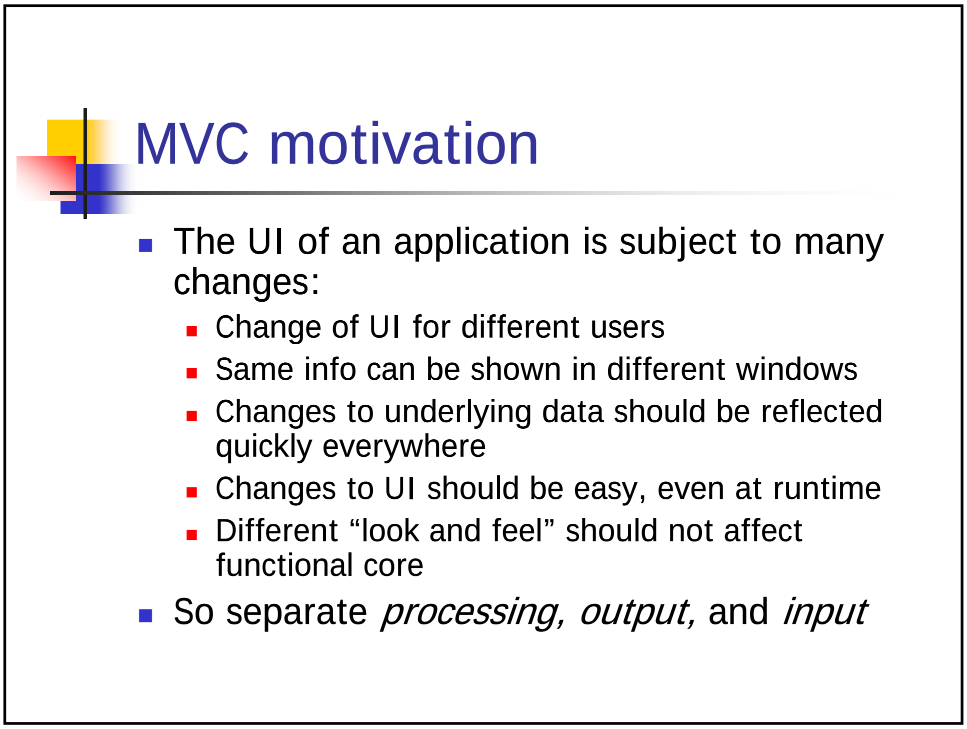
Composition

- Put together interactive objects at larger scale than interactors
- Container objects
  - e.g., row and column layout objects
- Containers can also add input & output behavior to things they contain



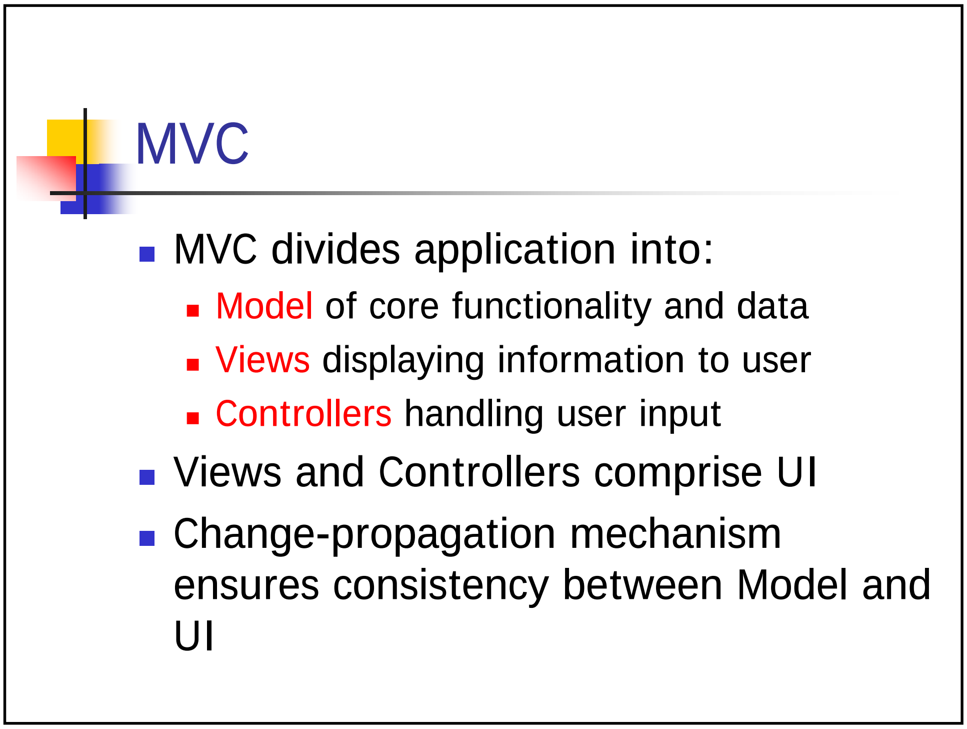
Aggregation

- Different concerns in separate objects
  - Treat collection as “the interactor”
- Classic architecture: “model-view-controller” (MVC)
  - from Smalltalk 80



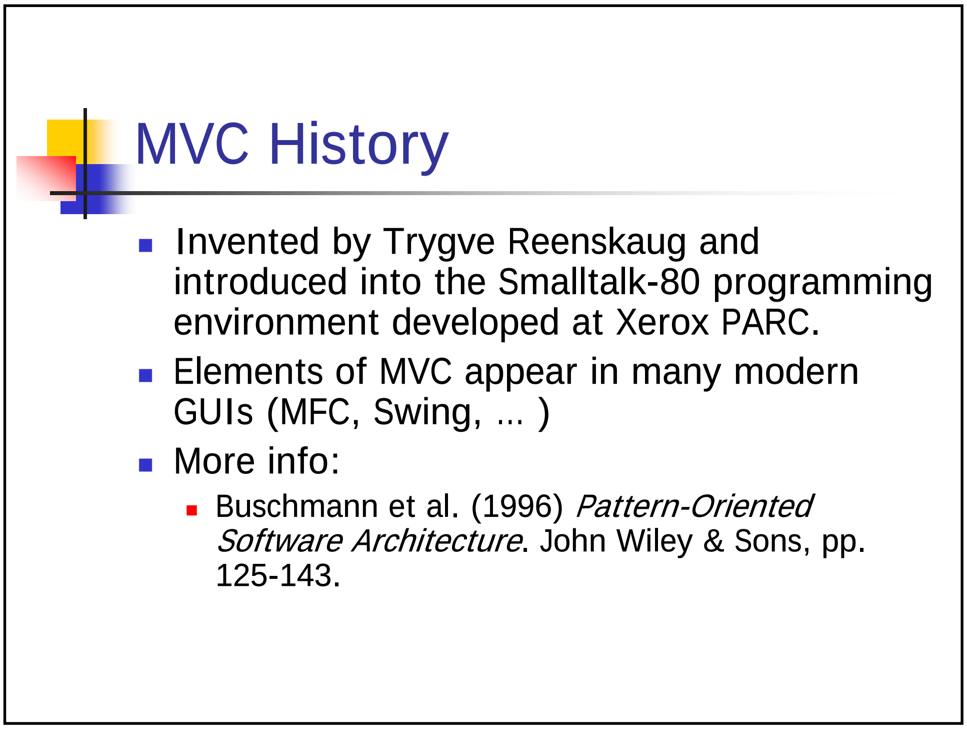
MVC motivation

- The UI of an application is subject to many changes:
  - Change of UI for different users
  - Same info can be shown in different windows
  - Changes to underlying data should be reflected quickly everywhere
  - Changes to UI should be easy, even at runtime
  - Different “look and feel” should not affect functional core
- So separate *processing, output,* and *input*



MVC

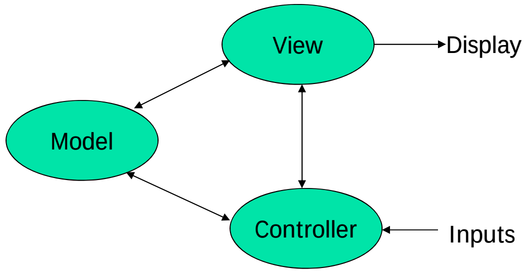
- MVC divides application into:
  - Model of core functionality and data
  - Views displaying information to user
  - Controllers handling user input
- Views and Controllers comprise UI
- Change-propagation mechanism ensures consistency between Model and UI



MVC History

- Invented by Trygve Reenskaug and introduced into the Smalltalk-80 programming environment developed at Xerox PARC.
- Elements of MVC appear in many modern GUIs (MFC, Swing, …)
- More info:
  - Buschmann et al. (1996) *Pattern-Oriented Software Architecture*. John Wiley & Sons, pp. 125-143.

Model-View-Controller Architecture



What are the advantages to separating these?

Model

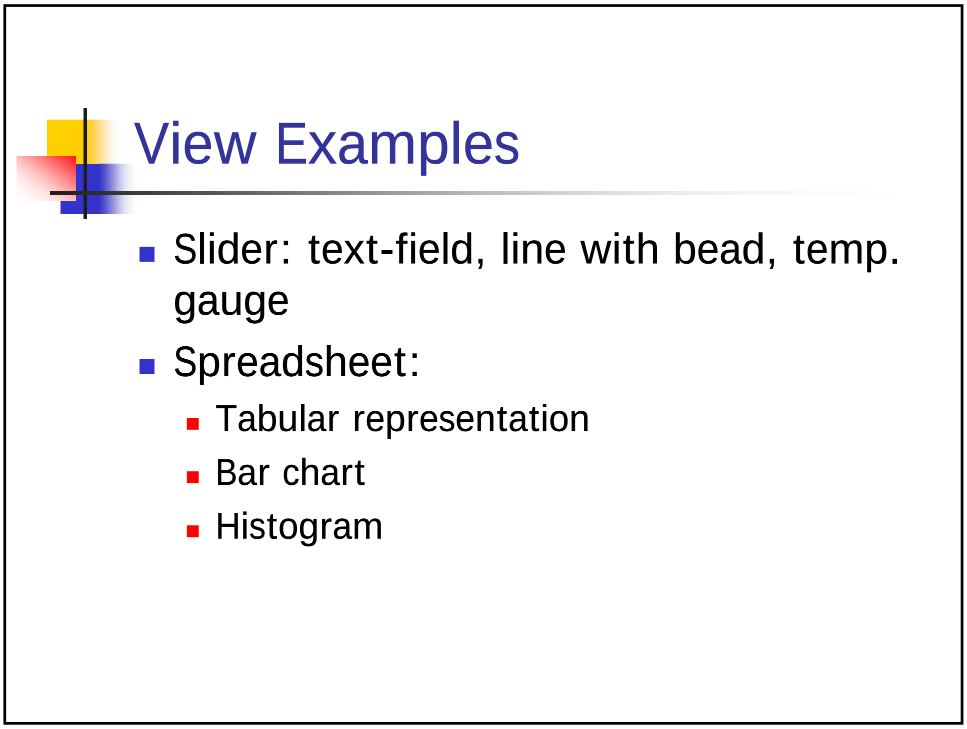
- Encapsulates application-specific data and functionality, providing:
  - methods to edit data, which Controller can call
  - methods to access state, which View and Controller can request
- Maintains registry of dependent Views and Controllers to be notified about data changes

Model Examples

- text editor: model is text string
- slider: model is an integer
- spreadsheet: collection of values related by functional constraints

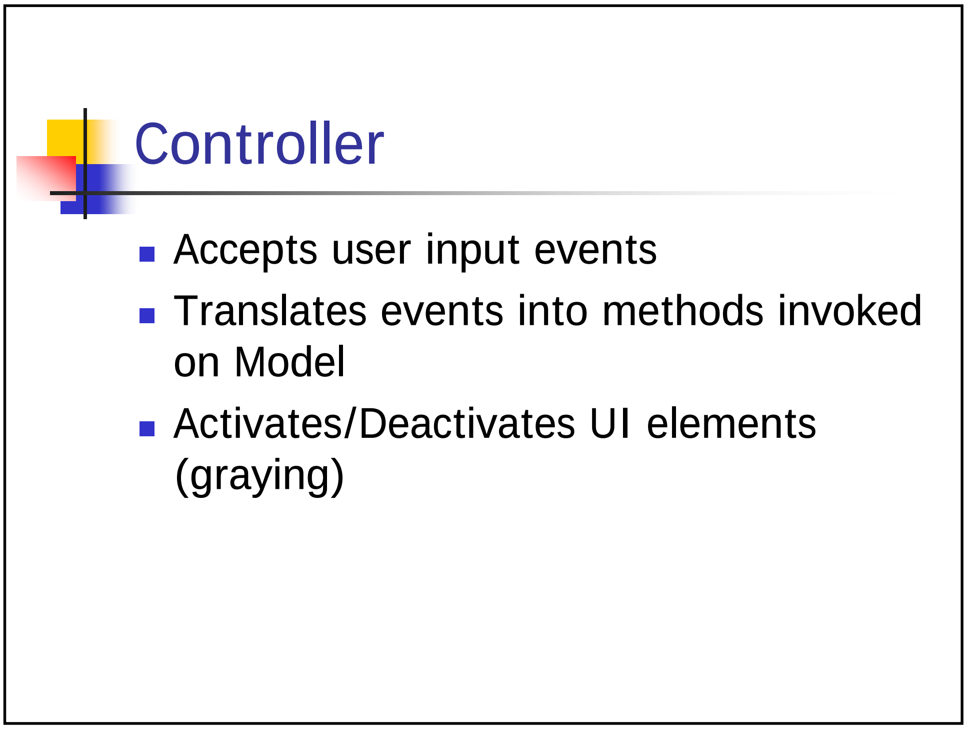
View

- Mechanism needed to map model data to rendition (view / display)
- When Model changes, View is informed
  - View requests relevant model information
  - View arranges to update screen
    - Declare damaged areas
    - Redraw when requested



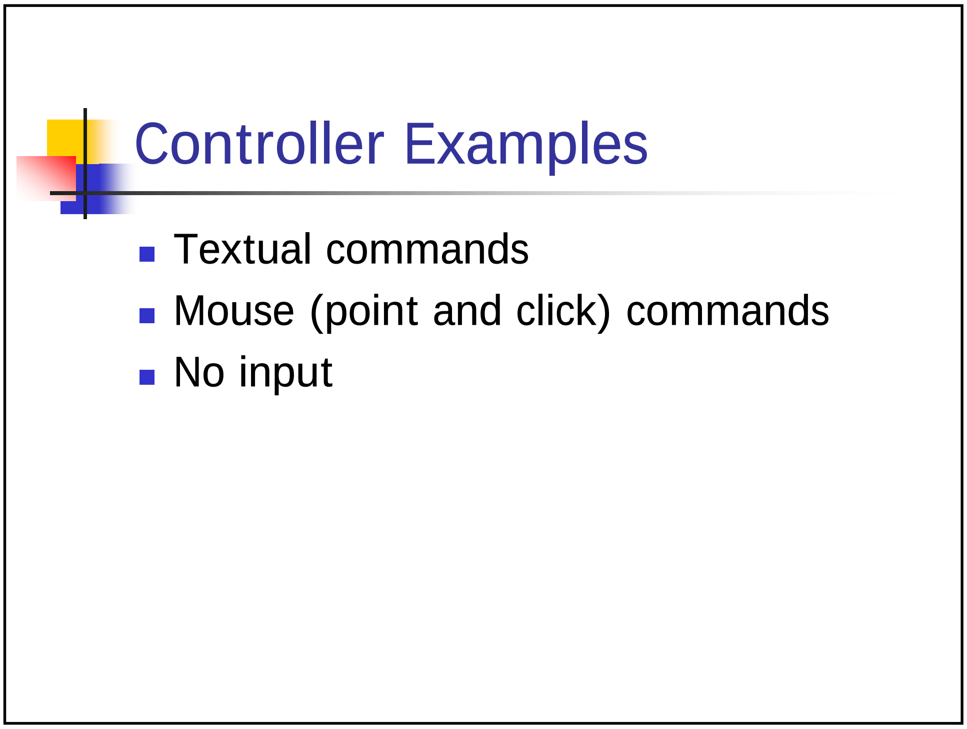
View Examples

- Slider: text-field, line with bead, temp. gauge
- Spreadsheet:
  - Tabular representation
  - Bar chart
  - Histogram



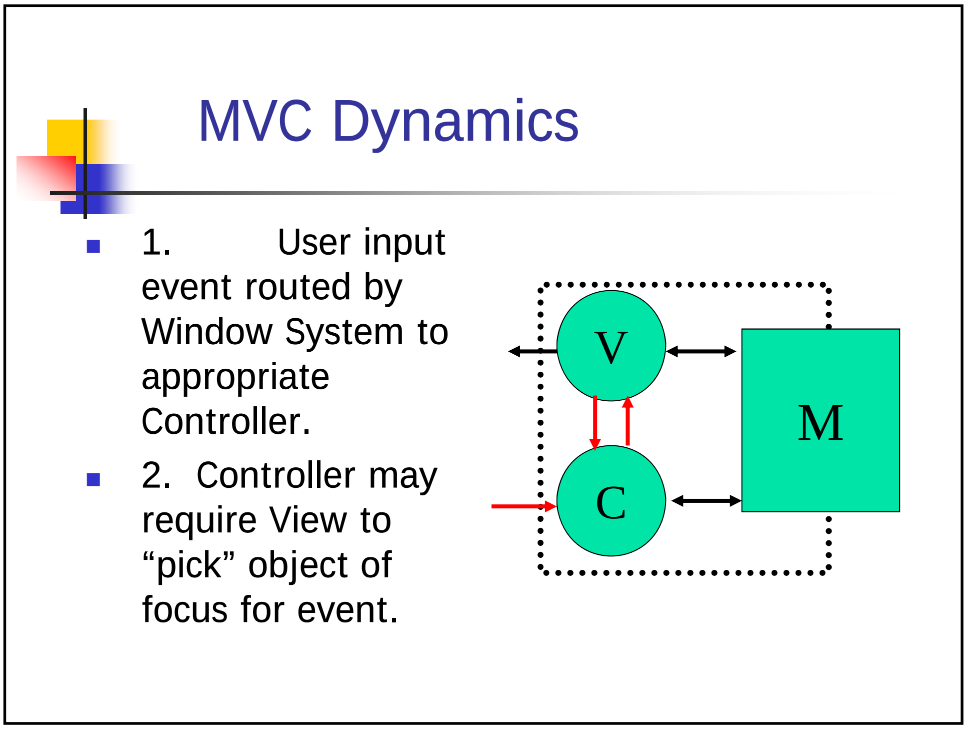
Controller

- Accepts user input events
- Translates events into methods invoked on Model
- Activates/Deactivates UI elements (graying)



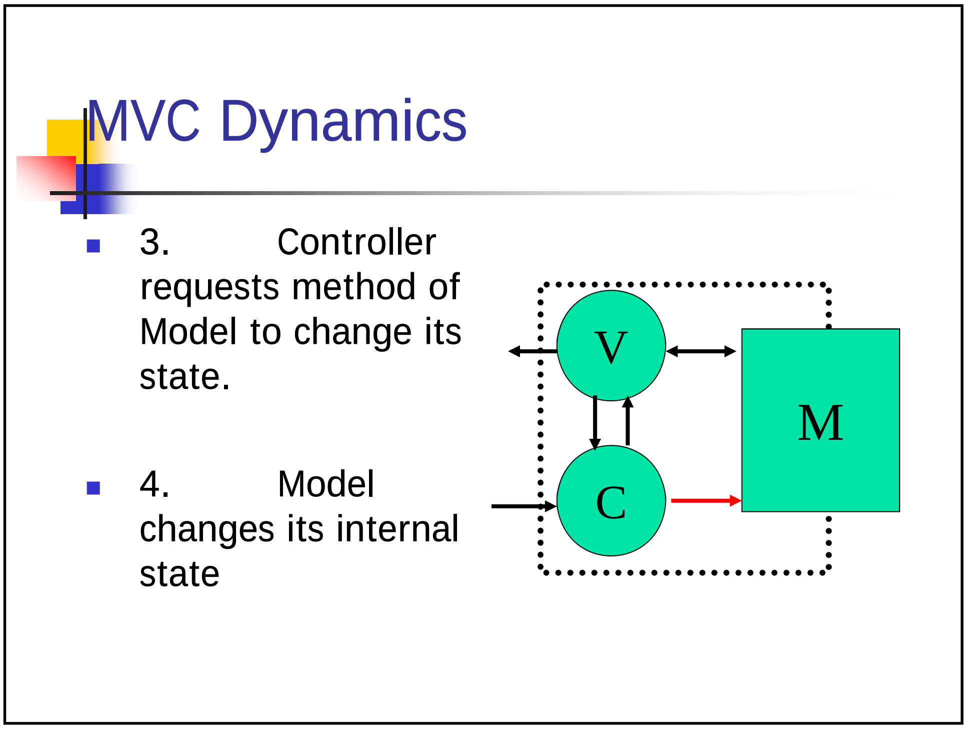
Controller Examples

- Textual commands
- Mouse (point and click) commands
- No input



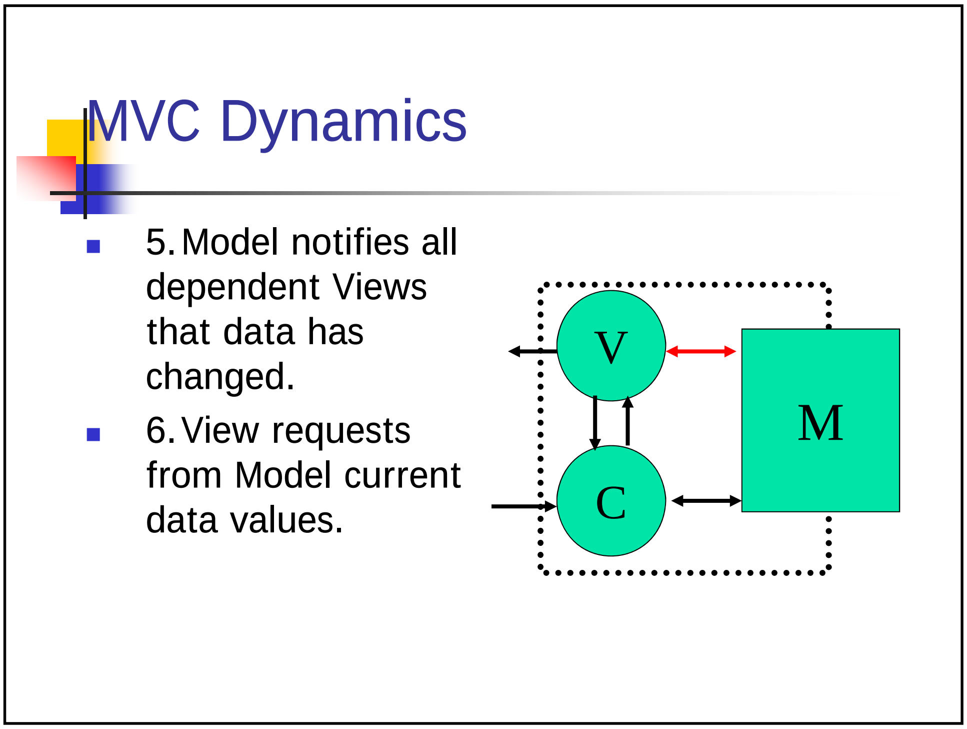
MVC Dynamics

- 1. User input event routed by Window System to appropriate Controller.
- 2. Controller may require View to “pick” object of focus for event.



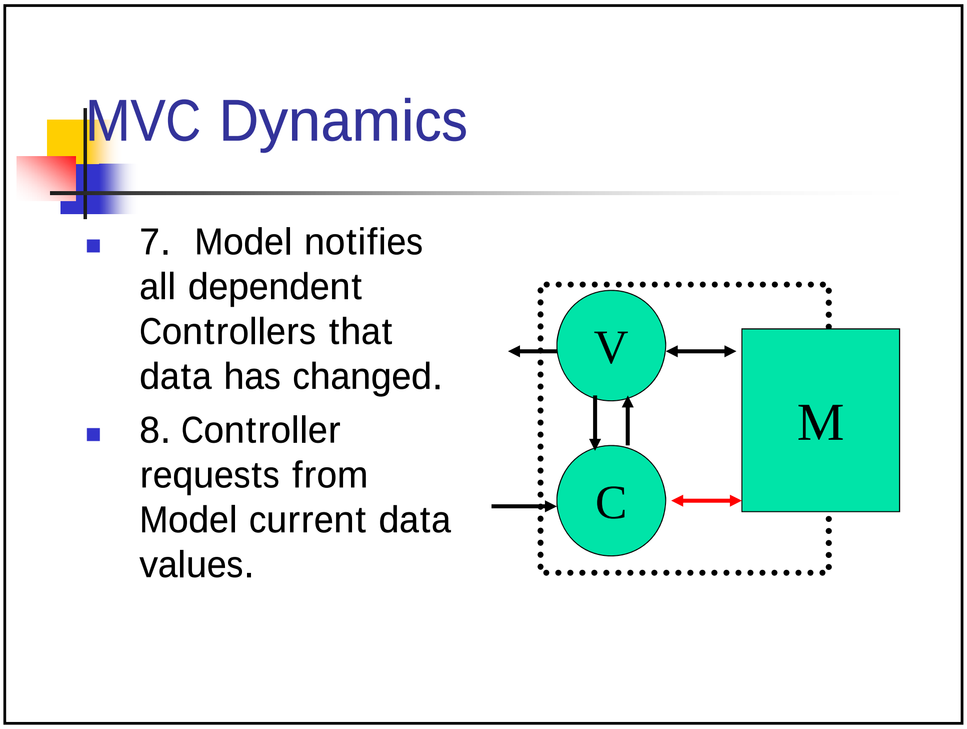
MVC Dynamics

- 3. Controller requests method of Model to change its state.
- 4. Model changes its internal state



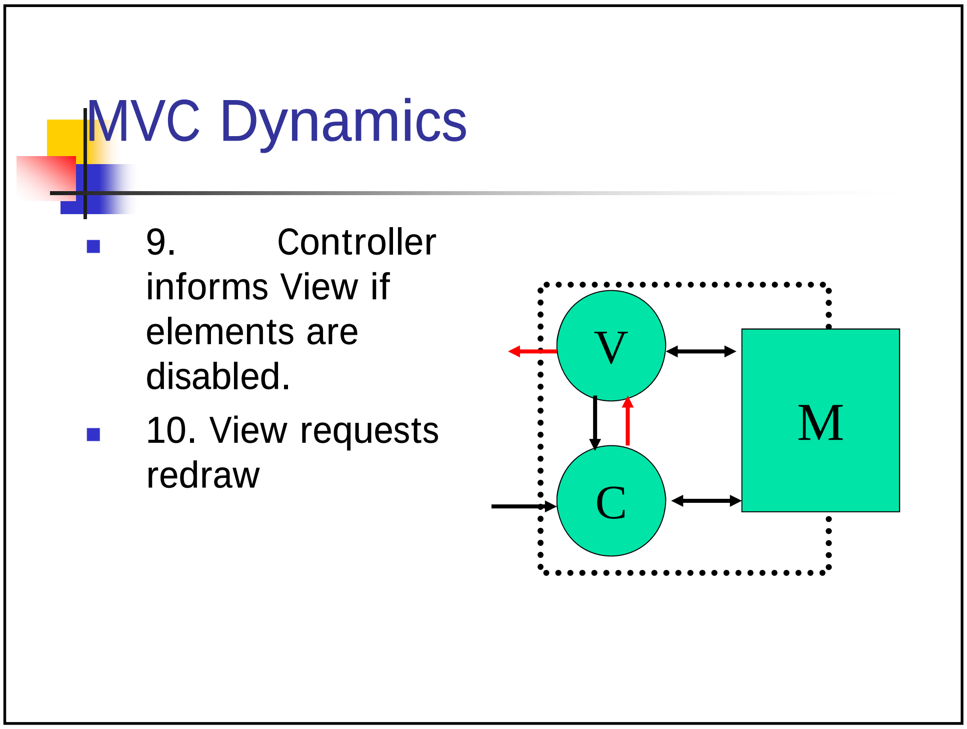
MVC Dynamics

- 5. Model notifies all dependent Views that data has changed.
- 6. View requests from Model current data values.



MVC Dynamics

- 7. Model notifies all dependent Controllers that data has changed.
- 8. Controller requests from Model current data values.



MVC Dynamics

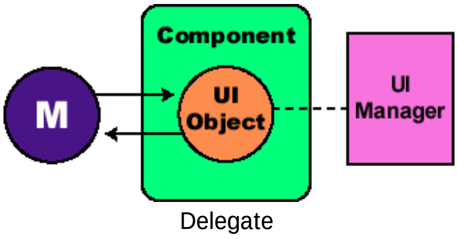
- 9. Controller informs View if elements are disabled.
- 10. View requests redraw

MVC: Pros and Cons

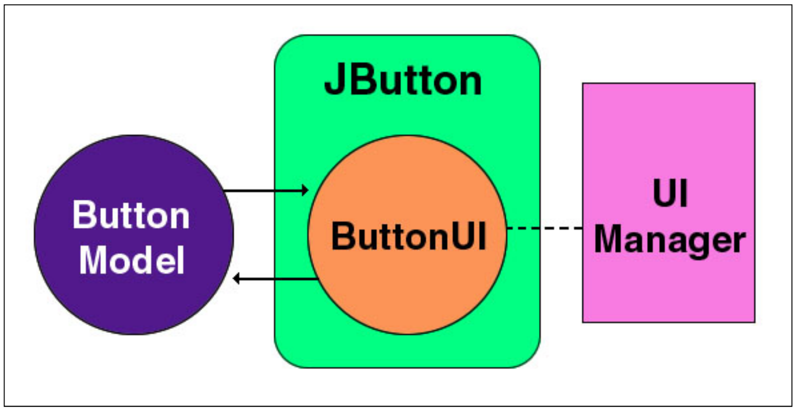
- Pros:
  - Multiple views of same model
  - Synchronized views
  - Pluggable V & C and “look and feel”
- Cons:
  - Complexity for simple interactors
  - Potentially excessive updates/messages
  - Tight coupling, in practice (V-C, VC-M)
  - Lack of portability
  - Some toolkits make MVC framework hard

Swing’s Modified MVC Architecture (“Model-Delegate”)

- Collapse View & Controller
  - Hard to write these independently
  - Allows pluggable look and feel



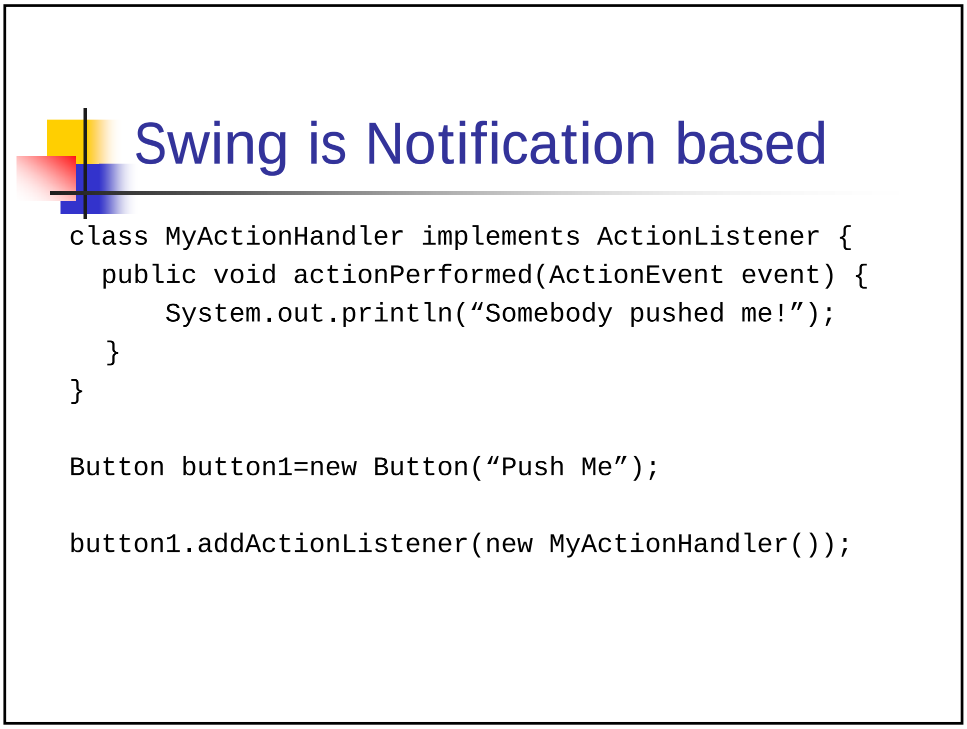
Example: pieces in a JButton



CS 4470/6456 - Fall 2003

Interaction toolkit example

- Java SWING!
  - Hold that thought…

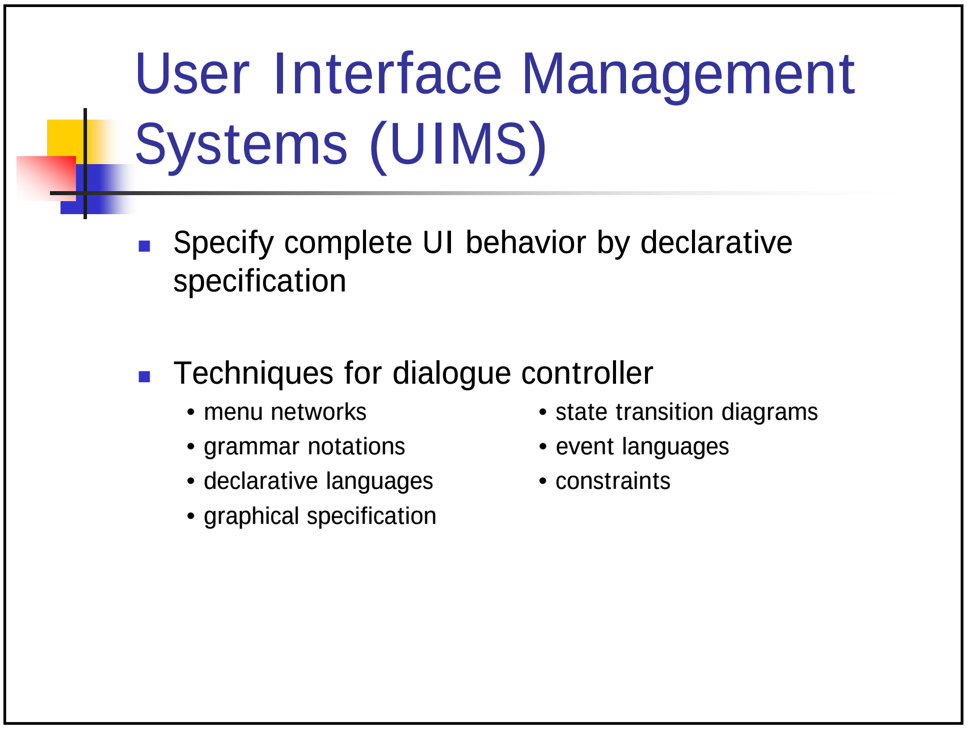


Swing is Notification based

```
class MyActionHandler implements ActionListener {
  public void actionPerformed(ActionEvent event) {
      System.out.println(“Somebody pushed me!”);
   }
}

Button button1=new Button(“Push Me”);

button1.addActionListener(new MyActionHandler());
```



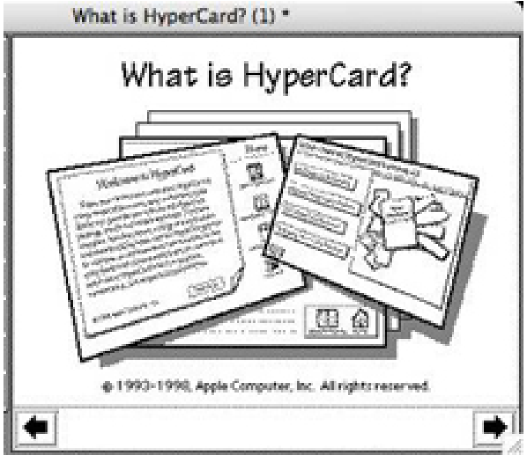
User Interface Management Systems (UIMS)

- Specify complete UI behavior by declarative specification
- Techniques for dialogue controller
  - menu networks
  - grammar notations
  - declarative languages
  - graphical specification
  - state transition diagrams
  - event languages
  - constraints

Graphical specification

- what it is
  - draw components on screen
  - set actions with script or links to program
- in use
  - with raw programming most popular technique
  - e.g. Visual Basic, Dreamweaver, Flash
- local vs. global
  - hard to 'see' the paths through system
  - focus on what can be seen on one screen

HyperCard

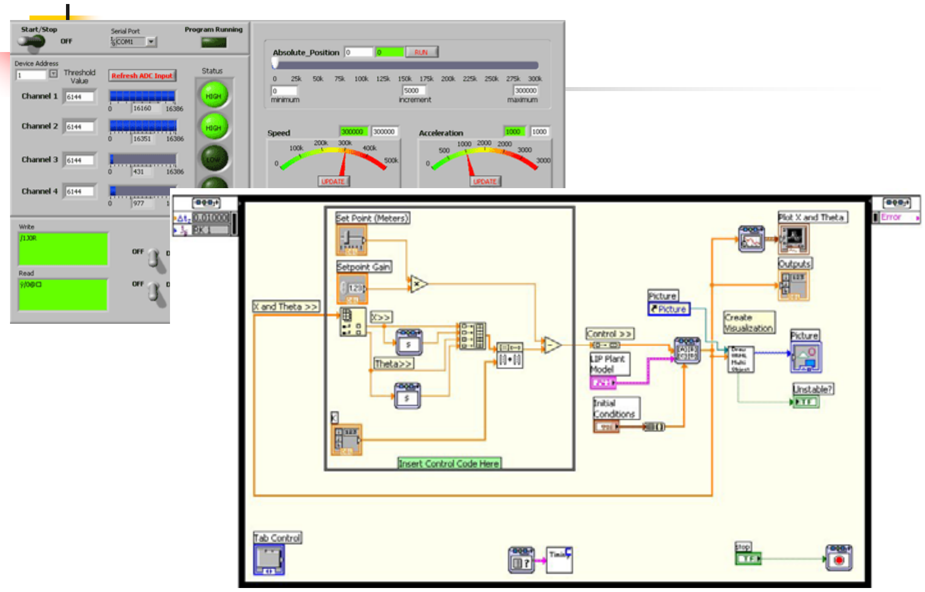


What is HyperCard? (1) *

What is HyperCard?

© 1993-1998, Apple Computer, Inc. All rights reserved.

LabView



Research Example: SILK

Swing & Netbeans

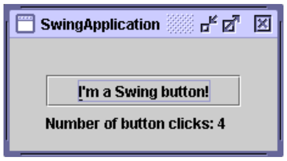
Java UI APIs

- AWT
  - The original – now mostly obsolete.
- Swing
  - The current standard.
- SWT (Standard Widget Toolkit)
  - Open source widget toolkit.
- JavaFX
  - Becoming new standard UI toolkit, but not as many components available yet, can’t customize look-and-feel (yet). Oracle plans to open source.

AWT vs. Swing

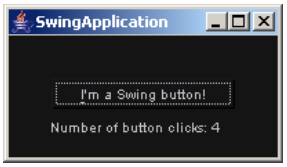
- AWT used “heavy weight” components
  - Uses native widget & processes
- Swing uses “light weight” components
  - 1997, 1.1.5
  - Uses native window for top-level frame, but Swing provides its own windowing system within the frame
    - Even draws its own menus
  - Thus,
    - Can have “pluggable look-and-feel”
    - Can be deployed on any device (with req’d libs)
    - Many more (non-native) widgets

Pluggable Look-and-Feel



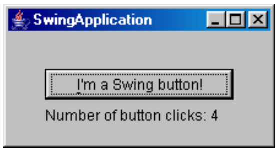
Number of button clicks: 4

Java



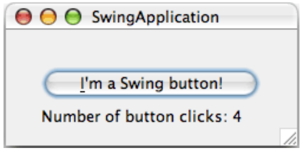
Number of button clicks: 4

GTK+



Number of button clicks: 4

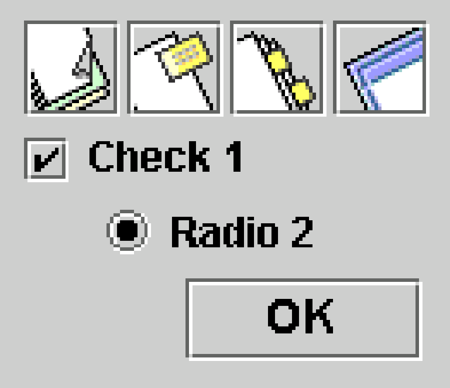
Windows



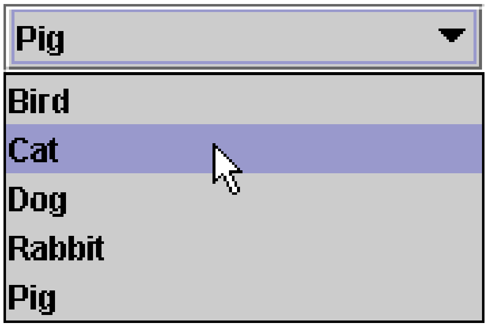
Number of button clicks: 4

Mac

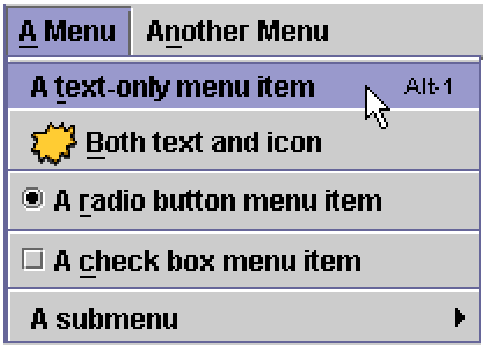
Buttons



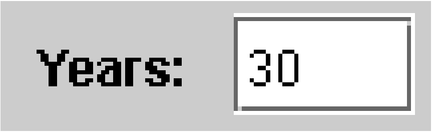
Combo Box



Menus

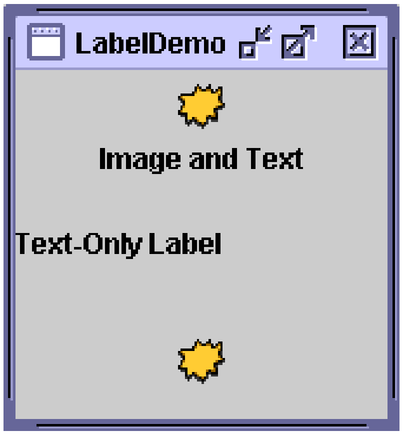


Text Field

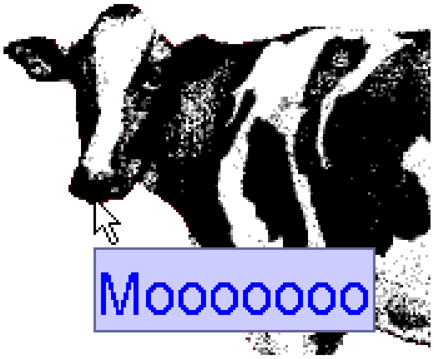


Years: 30

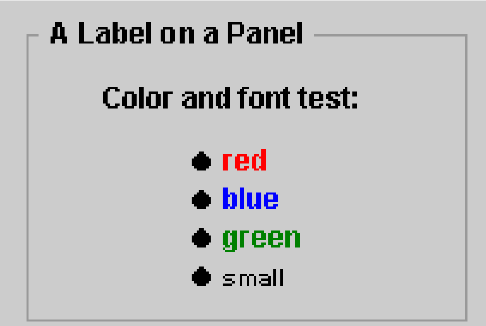
Labels



Tool tips



Embedded Panels

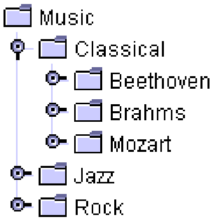


A Label on a Panel

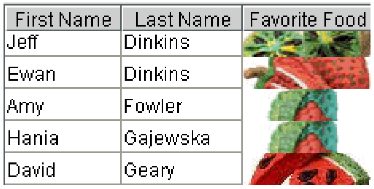
Color and font test:

- red
- blue
- green
- small

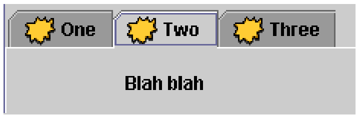
Advanced (not this homework)



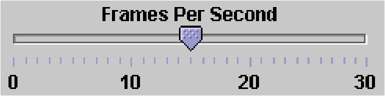
- Music
  - Classical
    - Beethoven
    - Brahms
    - Mozart
  - Jazz
  - Rock



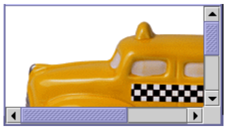
First Name	Last Name	Favorite Food
Jeff	Dinkins	
Ewan	Dinkins	
Amy	Fowler	
Hania	Gajewska	
David	Geary	



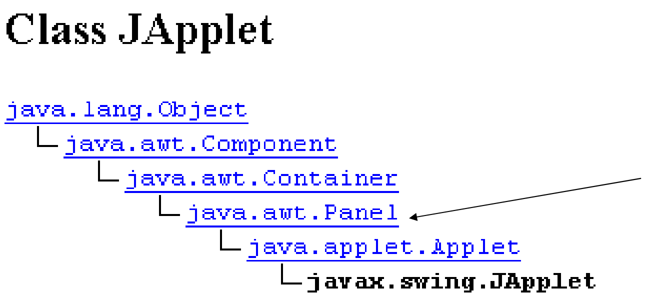
Blah blah



Frames Per Second



JApplet

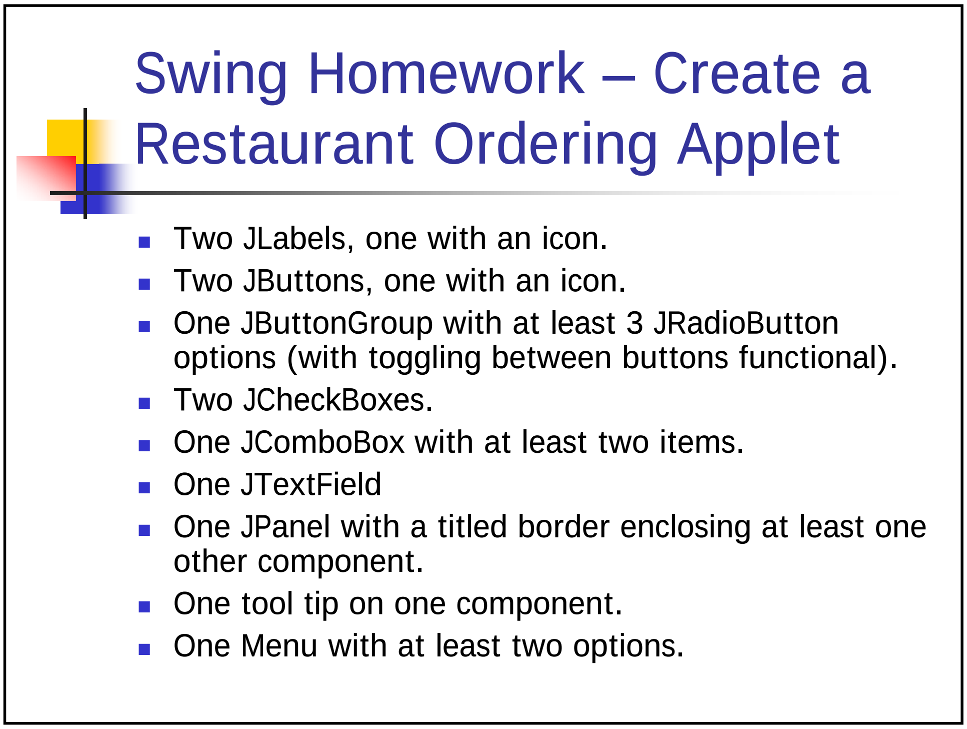


Class JApplet

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Panel
              └java.applet.Applet
                  └javax.swing.JApplet
```

A kind of Panel

Live Demo of NetBeans for Applet Building

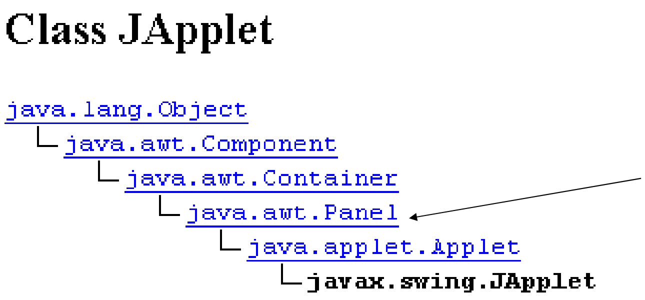


Swing Homework – Create a Restaurant Ordering Applet

- Two JLabels, one with an icon.
- Two JButtons, one with an icon.
- One JButtonGroup with at least 3 JRadioButton options (with toggling between buttons functional).
- Two JCheckBoxes.
- One JComboBox with at least two items.
- One JTextField
- One JPanel with a titled border enclosing at least one other component.
- One tool tip on one component.
- One Menu with at least two options.

Swing Events & Graphics Primitives

JApplet



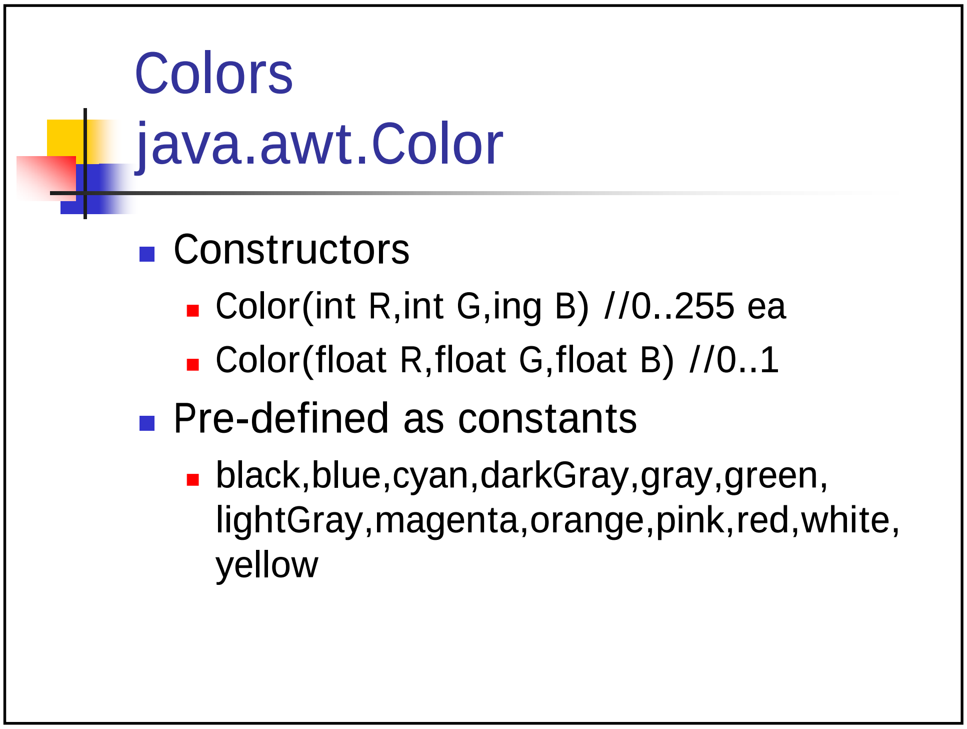
Class JApplet

```
java.lang.Object
  └java.awt.Component
      └java.awt.Container
          └java.awt.Panel
              └java.applet.Applet
                  └javax.swing.JApplet
```

A kind of Panel

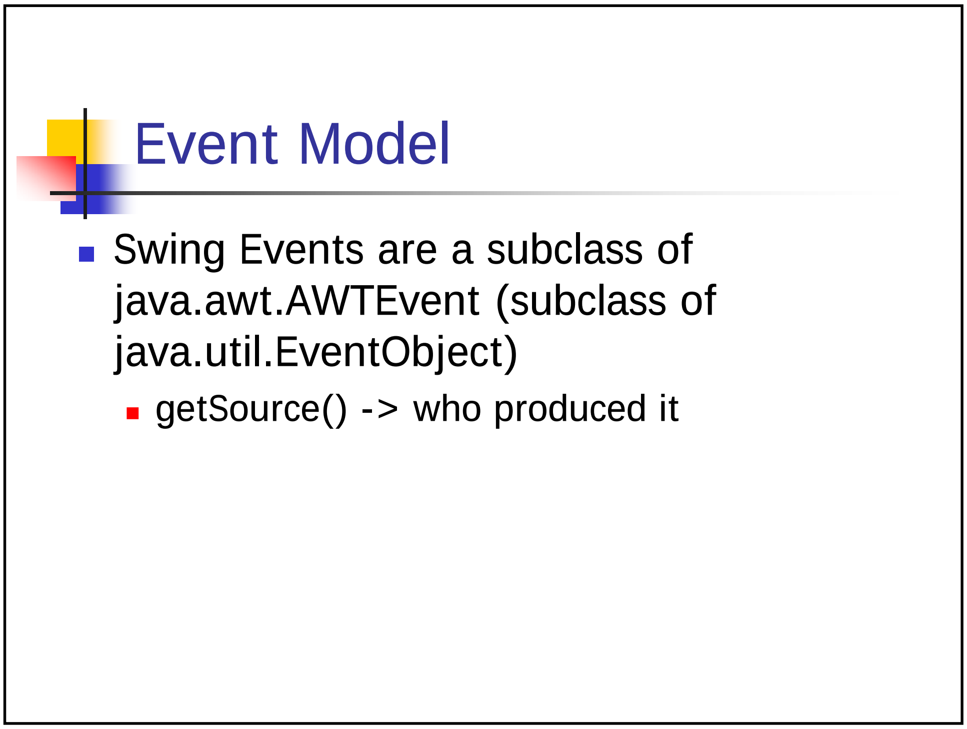
Useful stuff

- Graphics getGraphics() *called within JApplet*
  - Returns a ‘Graphics’ object
  - Device-independent interface to graphics
  - Basics (plus ‘fillX’ for most of these):
    - drawLine(x1,y1,x2,y2);
    - drawRect(x,y,w,h);
    - drawOval(x,y,w,h)
    - drawPolygon(int[] xpts,int[] ypts,numpts)
    - drawString(“a string”,x,y)
    - drawArc(x,y,w,h,startAngle,endAngle)
  - setColor(Color)
- Notes: ‘java.awt’ pkg, coordinate system



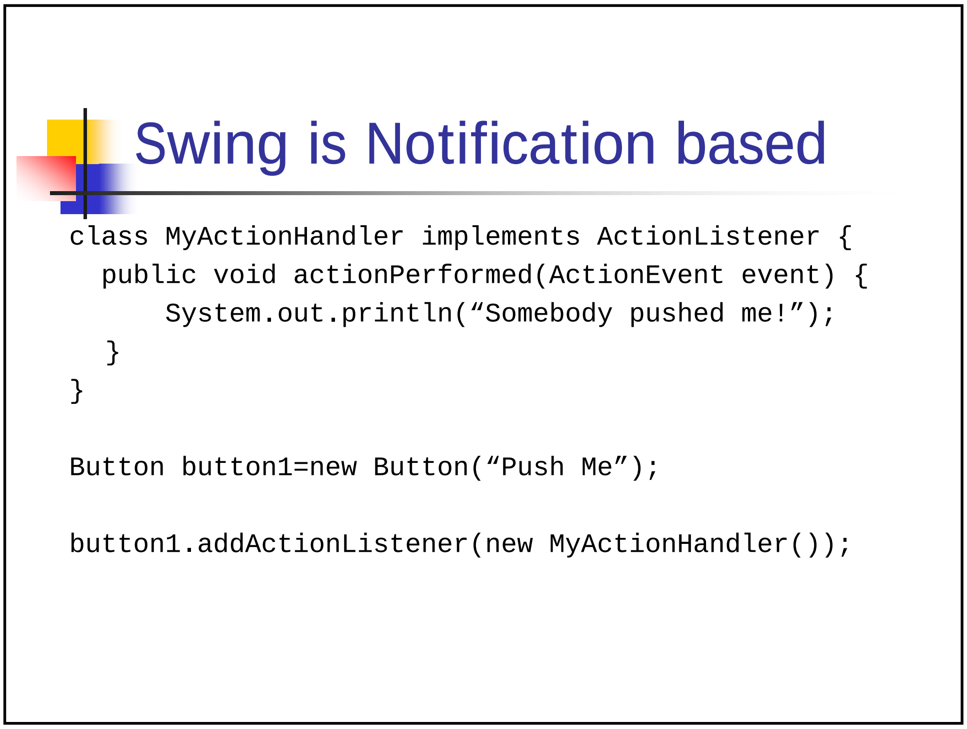
Colors
java.awt.Color

- Constructors
  - Color(int R,int G,ing B) //0..255 ea
  - Color(float R,float G,float B) //0..1
- Pre-defined as constants
  - black,blue,cyan,darkGray,gray,green, lightGray,magenta,orange,pink,red,white, yellow



Event Model

- Swing Events are a subclass of java.awt.AWTEvent (subclass of java.util.EventObject)
  - getSource() -> who produced it

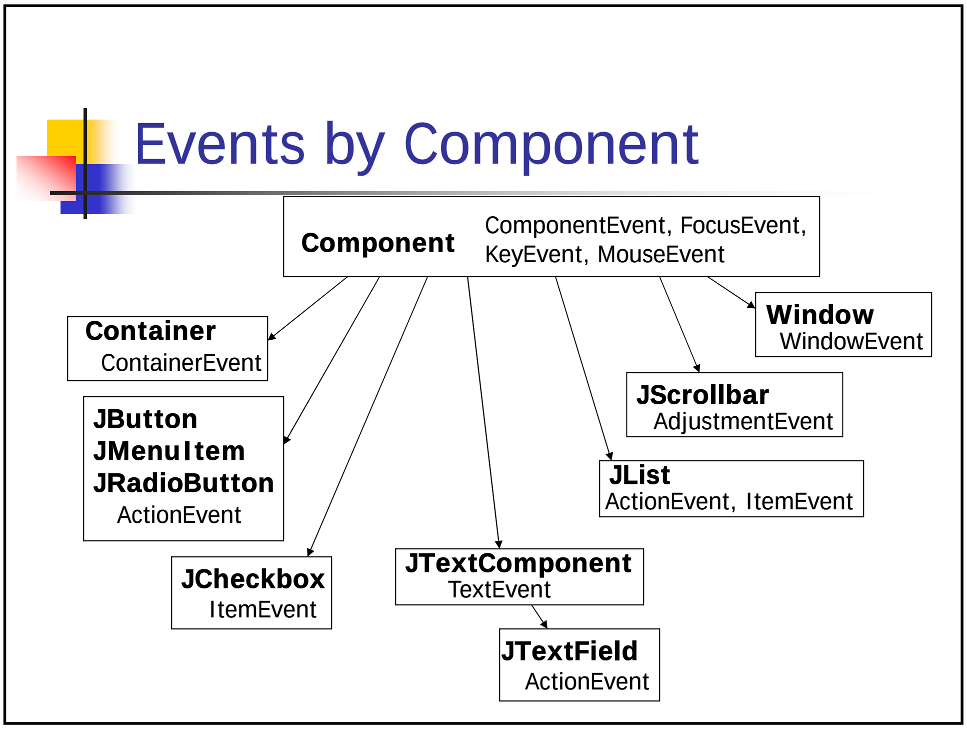


Swing is Notification based

```
class MyActionHandler implements ActionListener {
  public void actionPerformed(ActionEvent event) {
      System.out.println(“Somebody pushed me!”);
   }
}

Button button1=new Button(“Push Me”);

button1.addActionListener(new MyActionHandler());
```



Events by Component

Component ComponentEvent, FocusEvent, KeyEvent, MouseEvent

Container ContainerEvent

JButton
JMenuItem
JRadioButton
ActionEvent

JCheckbox ItemEvent

JTextComponent TextEvent

JTextField ActionEvent

JList ActionEvent, ItemEvent

JScrollbar AdjustmentEvent

Window WindowEvent

Event Types

Event	**Listener Methods**
Action	actionPerformed()
Adjustment	adjustmentValueChanged()
Component	componentHidden(), componentMoved(), componentResized(), componentShown()
Container	componentAdded(), componentRemoved()
Focus	focusGained(), focusLost()
Item	itemStateChanged()
Key	keyPressed(), keyReleased(), keyTyped()

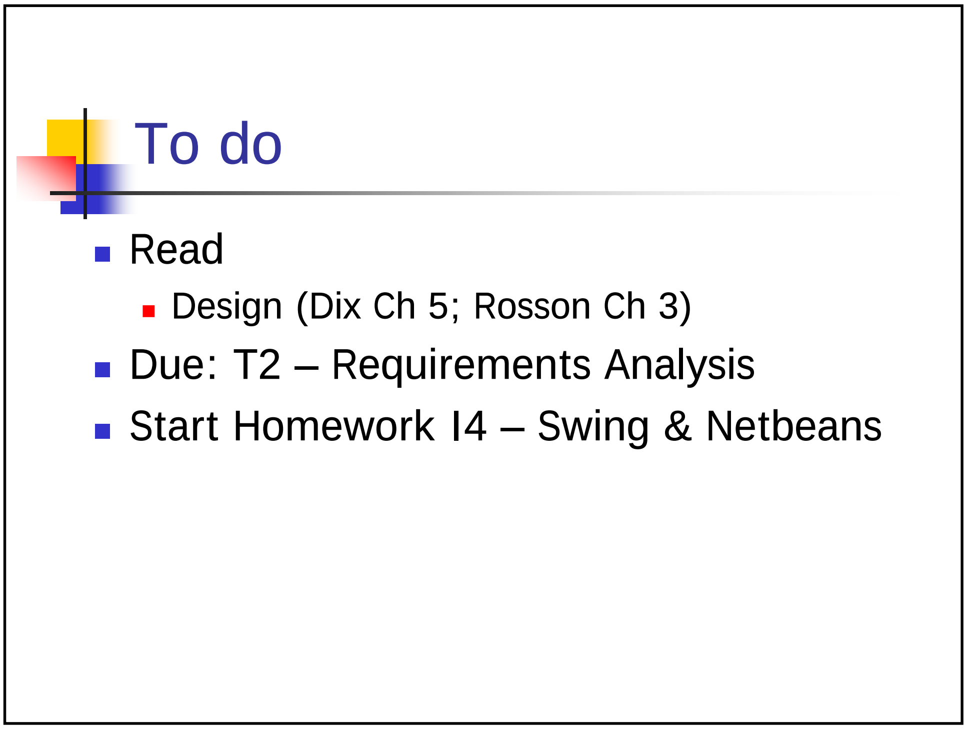
Mouse	**MouseListener / MouseAdapter:**
	mouseClicked(), mouseEntered(), mouseExited(), mousePressed(), mouseReleased()
	MouseMotionListener / MouseMotionAdapter:
	mouseDragged(), mouseMoved()

Text	textValueChanged()

Some Event Methods

ItemEvent	getStateChange() //SELECTED | DESELECTED
KeyEvent	getKeyChar(), getKeyCode()
MouseEvent	getX(), getY(), getClickCount()

NetBeans Example



To do

- Read
  - Design (Dix Ch 5; Rosson Ch 3)
- Due: T2 – Requirements Analysis
- Start Homework I4 – Swing & Netbeans