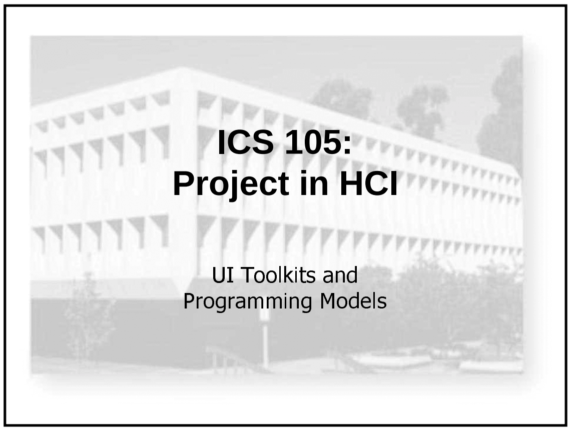
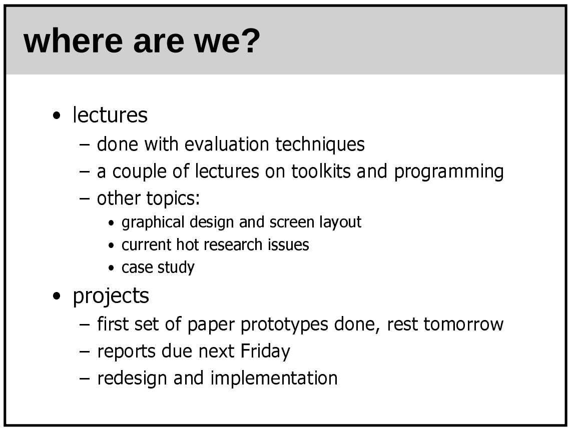


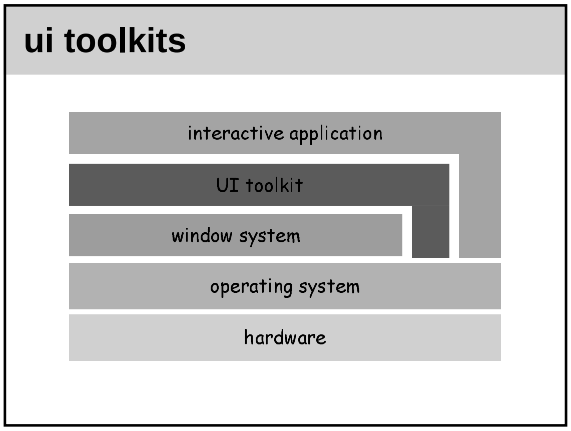
ICS 105: Project in HCI

UI Toolkits and
Programming Models



where are we?

- lectures
  - done with evaluation techniques
  - a couple of lectures on toolkits and programming
  - other topics:
    - graphical design and screen layout
    - current hot research issues
    - case study
- projects
  - first set of paper prototypes done, rest tomorrow
  - reports due next Friday
  - redesign and implementation



ui toolkits

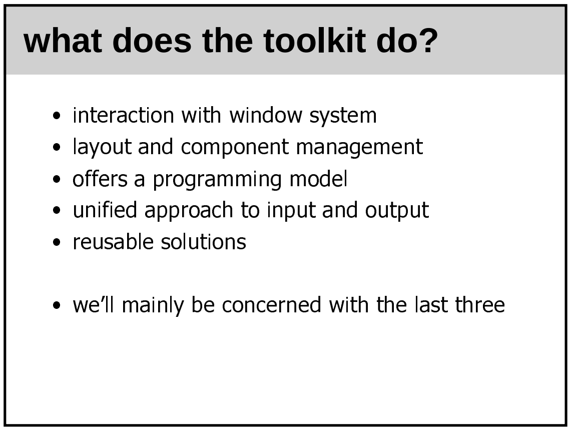
interactive application

UI toolkit

window system

operating system

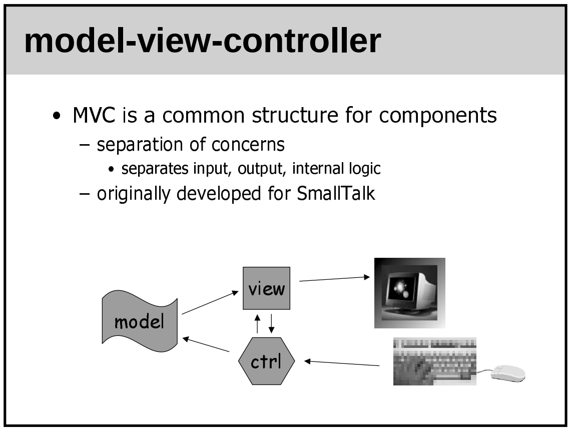
hardware



what does the toolkit do?

- interaction with window system
- layout and component management
- offers a programming model
- unified approach to input and output
- reusable solutions

- we'll mainly be concerned with the last three



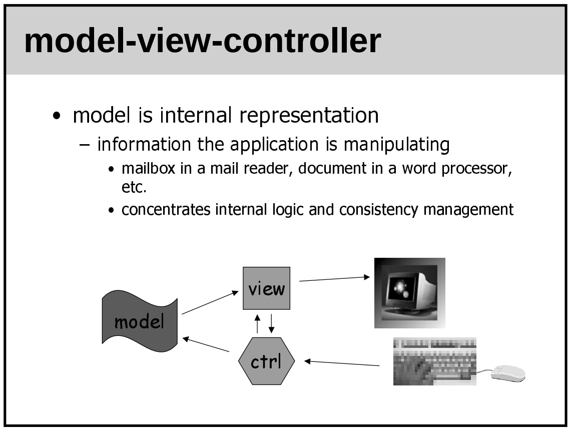
model-view-controller

- MVC is a common structure for components
  - separation of concerns
    - separates input, output, internal logic
  - originally developed for SmallTalk

model

view

ctrl



model-view-controller

- model is internal representation
  - information the application is manipulating
    - mailbox in a mail reader, document in a word processor, etc.
    - concentrates internal logic and consistency management

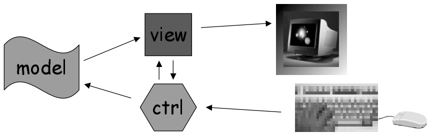
model

view

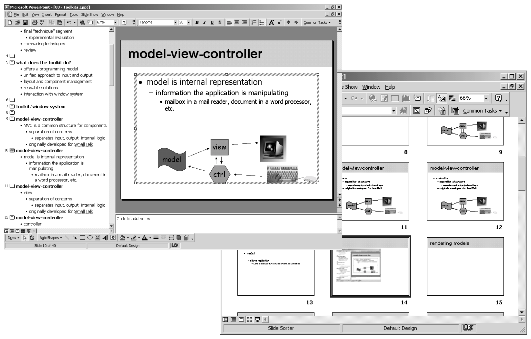
ctrl

model-view-controller

- view is the visual representation
  - may have multiple views
    - e.g. graphical and textual depictions
  - notifications from model when it changes
    - maintains consistency



multiple views

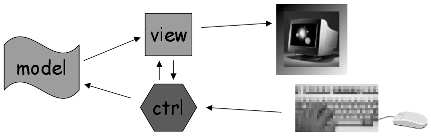


model-view-controller

- model is internal representation
  - information the application is manipulating
    - mailbox in a mail reader, document in a word processor, etc.

model-view-controller

- controller
  - handles all interaction with the user
    - receives input events, decides what they mean
  - makes changes to view and to model
    - e.g. edits vs scrolling



model-view-controller

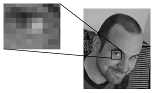
- advantages
  - separation of concerns supports better software engineering
    - easy to modify and maintain
  - allows replication
    - makes it easier to add new views and controls later
- variations
  - many systems combine view and controller
    - in *direct manipulation*, view *is* controller

rendering models

- three components to ui toolkits
  - architecture (e.g. MVC)
  - input (to come)
  - output (focus for now)
- output
  - primary distinction is the *rendering model*
    - how images are described and constructed

raster models

- fundamental structure is the raster image
  - array of color values
  - array of pixel coordinates from (0,0) to size of screen
    - typically top left to bottom right
  - great for images, less good for structured graphics
    - toolkit maintains minimal information about structure
      - e.g. the lines and objects that gave rise to pixel image

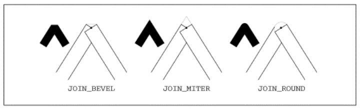


stroked models

- fundamental structures are paths and strokes
  - higher level than individual pixels
    - resolution independence
  - originated in printer Page Description Languages
    - Press, InterPress, PostScript
      - Display Postscript used in NeWS and NeXT
      - PDF-based rendering model in Apple's MacOS X

stroked models

- joins



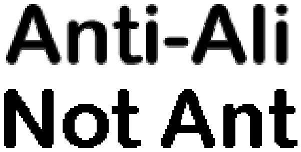
- complex paths

stroked text

other advanced features

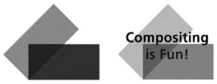
- font support and antialiasing
  - font support can be minimal in raster models
  - need to get from "letter+size" to raster image
    - originally, stored fonts simply as bitmaps
    - these days, use programmatic font support (TrueType)
  - antialiasing makes fonts easier to read

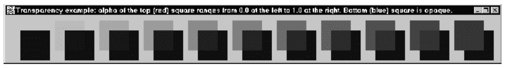
other advanced features



other advanced features

- alpha channel





Transparency example: alpha of the top (red) square ranges from 0.0 at the left to 1.0 at the right. Bottom (blue) square is opaque.

Java 2D

- Java graphics originally based on AWT
  - minimal
    - clearly just enough to ship…
- Java now supports two-level design
  - JFC is the user interface component
  - Java2D is the underlying graphics component
    - much richer rendering model

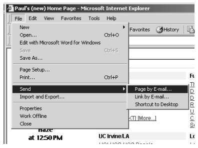
Java2D demo

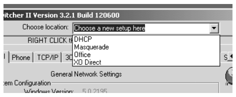
widgets

- Macintosh (1984) first commercial GUI system
  - two aspects
    - user interface to the system itself
    - Mac Toolbox made components available to others
  - seven basic widgets
    - buttons
    - sliders (also implement scrollbars)
    - pull-down menus
    - checkboxes
    - radio buttons
    - text fields
    - file open/save dialog
      - other widgets (e.g. window decorations) not in toolbox

widgets

- second Mac release added more
  - hierarchical (pull-right) menus
  - in-place menus (drop-down selection boxes)
  - lists (single and multiple selections)





widgets

- more recent additions (Macs and others)
  - tabbed dialogs
  - hierarchical lists (trees)
  - “combo boxes” (combination menu, list, text)
- this set pretty much covers conventional UI
  - not all that’s there – e.g. pie menus
  - different models for different
    - interfaces for PDAs?
    - interfaces for interaction on TV?

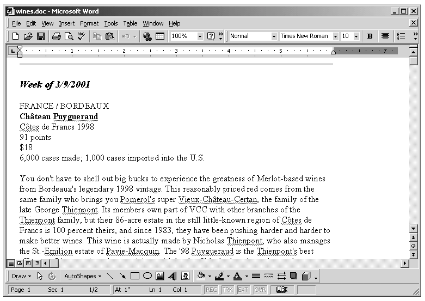
widget model

- convenience for both users and developers
  - users get familiar interaction styles
    - established “genres” of user interface design
    - eases transfer of skills from one application to another
  - programmers get predefined units
    - eases conformance to UI guidelines
    - saves repetition of effort
- only part of the story, though
  - widgets are components
  - how do components fit together?
  - how are behaviors defined?

event-based programming

- basic program structures
  - non-interactive applications
    - start, do something, stop
  - simple interactive applications
    - main loop – await instructions, carry them out, repeat
- most interactive applications more complex
  - lots of state
  - many operations
    - operations of many different sorts
    - how many different operations can you carry out?

event-based programming


wines.doc - Microsoft Word
File Edit View Insert Format Tools Table Window Help
100%
Normal
Times New Roman
10
Draw
AutoShapes
Page 1 Sec 1 1/2 At 1" Ln 1 Col 1 REC TRK EXT OVR

Week of 3/9/2001

FRANCE / BORDEAUX
Château Puygueraud
Côtes de Francs 1998
91 points
$18
6,000 cases made; 1,000 cases imported into the U.S.

You don't have to shell out big bucks to experience the greatness of Merlot-based wines from Bordeaux's legendary 1998 vintage. This reasonably priced red comes from the same family who brings you Pomerol's super Vieux-Château-Certan, the family of the late George Thienpont. Its members own part of VCC with other branches of the Thienpont family, but their 86-acre estate in the still little-known region of Côtes de Francs is 100 percent theirs, and since 1983, they have been pushing harder and harder to make better wines. This wine is actually made by Nicholas Thienpont, who also manages the St.-Emilion estate of Pavie-Macquin. The '98 Puygueraud is the Thienpont's best

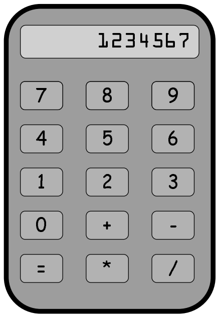
event-based programming

- modal solutions
  - restrict operations that can take place at any time
  - places the burden on the user
    - which mode are you in now?
    - how do you get from mode to mode?
    - easier to make errors
    - barriers in the way of operations
- complexity grows
  - effective design requires more sophisticated model

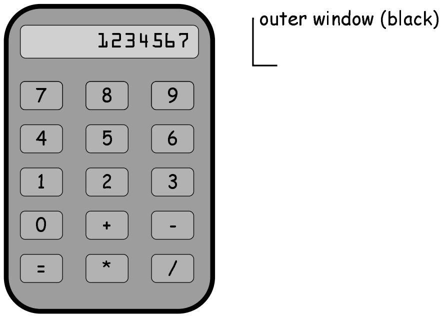
event-based programming

- turn things around
  - instead of user waiting on system, have system wait on user
  - this is the *event based approach*
    - declarative approach to programming
    - user actions generate *events*
      - e.g. mouse clicked, button pressed, scroll bar moved
    - set up object structure
      - describe structure of solution
      - describe how objects will respond to events
    - implicit main loop
      - collects events, determines targets, sends events

interactor tree

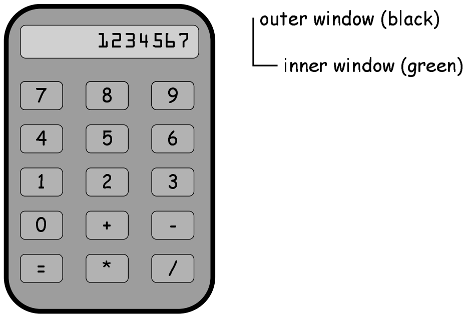


interactor tree

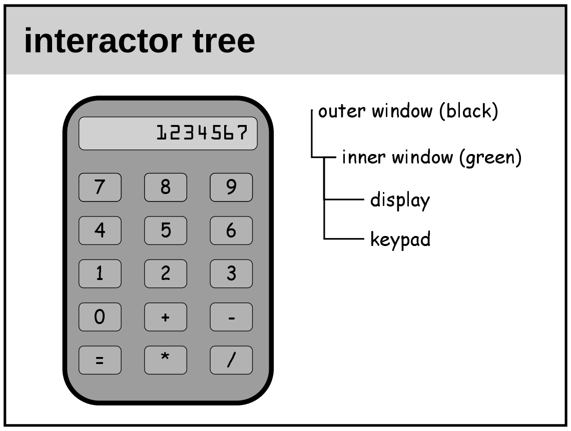


outer window (black)

interactor tree

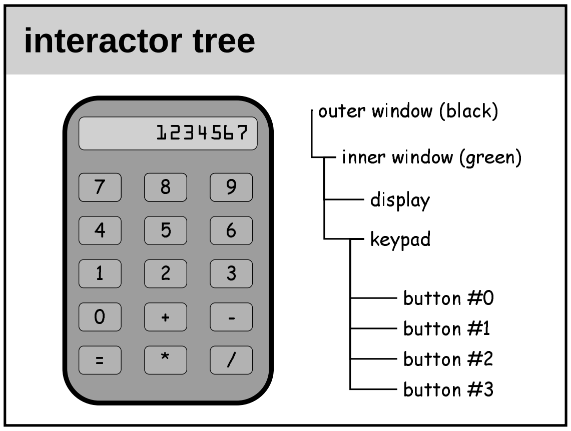


outer window (black)
inner window (green)



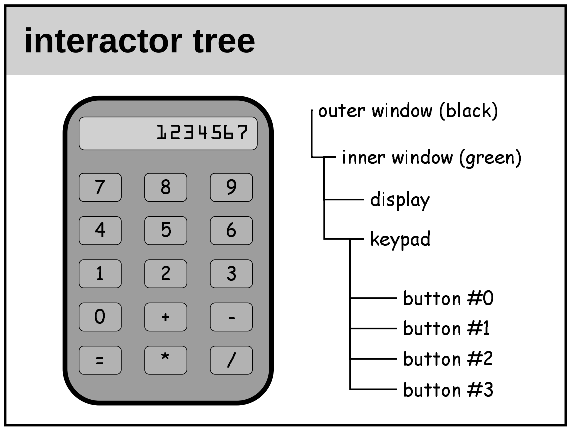
interactor tree

- outer window (black)
  - inner window (green)
    - display
    - keypad



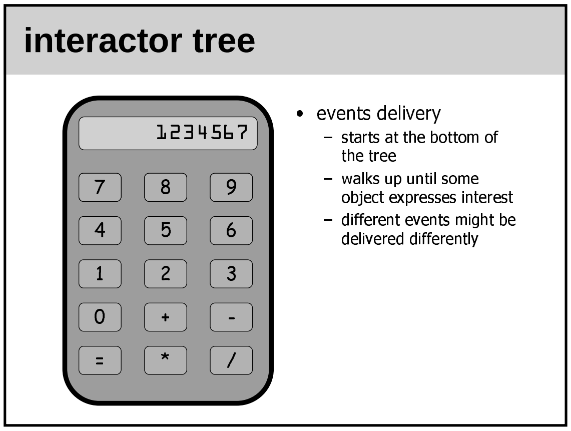
interactor tree

- outer window (black)
  - inner window (green)
    - display
    - keypad
      - button #0
      - button #1
      - button #2
      - button #3



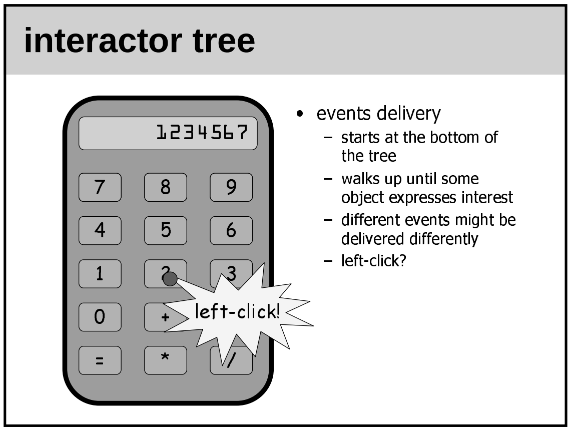
interactor tree

- outer window (black)
  - inner window (green)
    - display
    - keypad
      - button #0
      - button #1
      - button #2
      - button #3



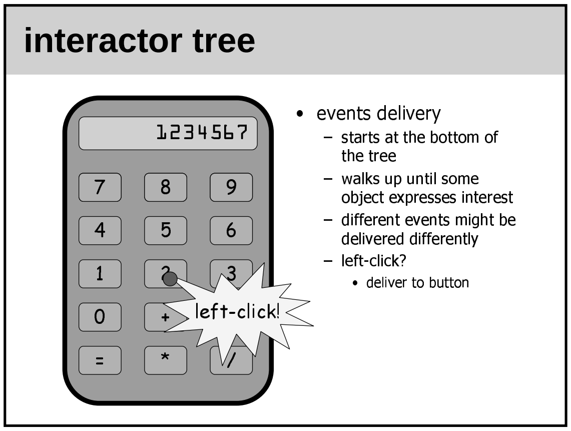
interactor tree

- events delivery
  - starts at the bottom of the tree
  - walks up until some object expresses interest
  - different events might be delivered differently



interactor tree

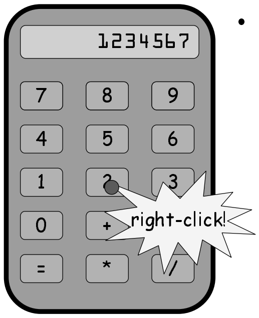
- events delivery
  - starts at the bottom of the tree
  - walks up until some object expresses interest
  - different events might be delivered differently
  - left-click?



interactor tree

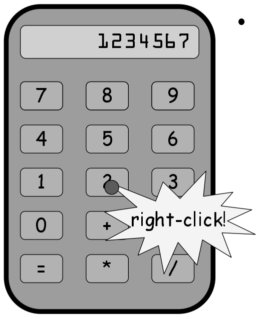
- events delivery
  - starts at the bottom of the tree
  - walks up until some object expresses interest
  - different events might be delivered differently
  - left-click?
    - deliver to button

interactor tree



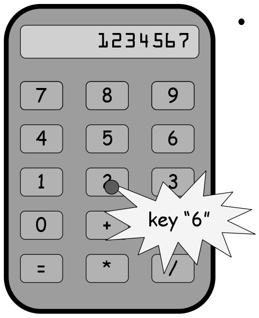
- events delivery
  - starts at the bottom of the tree
  - walks up until some object expresses interest
  - different events might be delivered differently
  - right-click?

interactor tree



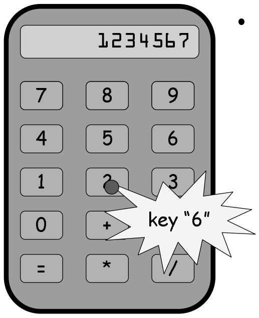
- events delivery
  - starts at the bottom of the tree
  - walks up until some object expresses interest
  - different events might be delivered differently
  - right-click
    - button not interested
    - deliver to keypad
      - keypad menu

interactor tree



- events delivery
  - starts at the bottom of the tree
  - walks up until some object expresses interest
  - different events might be delivered differently
  - keypress?

interactor tree



- events delivery
  - starts at the bottom of the tree
  - walks up until some object expresses interest
  - different events might be delivered differently
  - keypress?
    - button not interested
    - keypad not interested
    - deliver to window
      - global input handling

ui and oop

- event-based model meshes naturally with OOP
  - objects and containment structures
  - keep “behavior” close to “data”
  - delegate event processing between objects

constraints

- event model is the conventional approach
  - another common approach is to use constraints
- constraint-based programming
  - declarative approach to programming
  - constraint is a desired invariant
    - a := b * 2
    - a <-> b * 2
  - complexity
    - satisfaction engine ensures all constraints maintained
    - single and multi-way constraints

constraints

- constraints apply naturally to UI
  - think of MVC
    - view must track model
    - controller must keep view in sync
    - examples
      - manage a scrollbar by expressing a constraint between the location of the scroll box and the current view port
      - keep item centered in window as it resizes by expressing constraint about the size of padding on either side

constraints

- advantages of constraint approach?
  - declarative programming style
    - express what you want to happen once and for all
    - event-based programming distributes activity
      - hard to find the one place where things happen
  - express natural regularities
    - people understand causation naturally
    - constraint-based designs can be very intuititive
- disadvantages?
  - computationally expensive
  - not yet mainstream (but we’re working on it)

next week

- more in-depth on Swing/JFC