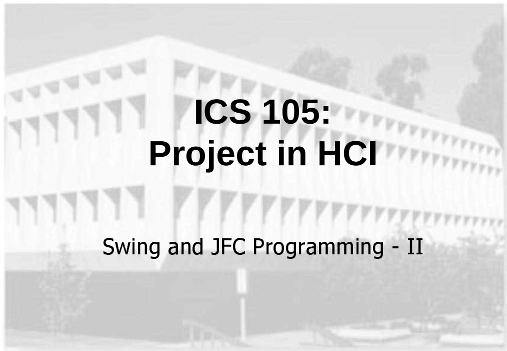


ICS 105: Project in HCI

Swing and JFC Programming - II

projects

- timetable
  - paper prototyping reports are due Friday
    - my office, by 4pm
  - implementation phase
  - the goal is to be able to *evaluate* week of May 28
    - lab-based evaluation
    - focus on the tasks
      - tasks have to be complex enough to test the interface
  - meet with Doshi in next week

today

- brief review on Swing
- painting model
- explore a range of examples
  - none of these is very complex…
  - code on the class web page

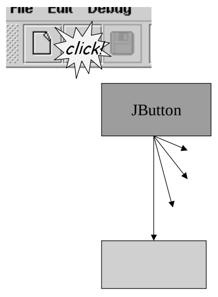
frames, panels and panes

- panels
  - most user interface objects live in panels (JPanel)
  - panels are the basic unit of layout
- layout
  - FlowLayout, BorderLayout, GridLayout,…
  - all express ongoing constraints
- panels are generally nested

swing event model

- event-based programming model
  - user interface actions generate events
  - events delivered to objects that express interest
- the swing approach
  - every object has a set of listeners
    - different listeners for different sorts of events
  - listeners are objects interested in events
    - note – this is OBJECT BASED
      - listeners are objects
      - listeners are associated with particular objects

swing event model



swing event model

- listeners
  - three features
    - the listener need to declare the right interfaces
      - each type of listener has an associated interface
        - » e.g. ActionListener, MouseMotionListener
      - java.awt.event.*;
    - the listener needs to be attached
      - via addXXXListener()
    - the listener needs to handle the event
      - implement the methods specified in the interface
        - » public void actionPerformed(ActionEvent)

swing idioms

- listeners often use anonymous inner classes
  - inner classes are defined inside other classes
  - anonymous inner classes are
    - unnamed
    - defined in-line

```
foo.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    System.out.println(“The action was performed!”);
  }
});
```

drawing example

- simple drawing application
  - open a window
  - listen for mouse events
    - mouse down – start of drawing
    - mouse dragged – draw lines

the painting model

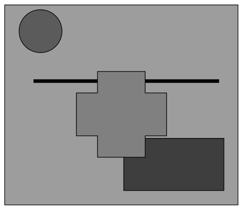
- painting is distributed
  - objects must paint themselves
    - each object knows how it should be painted
  - in AWT, single paint() method
  - in Swing, paint() delegates to:
    - paintComponent()
    - paintBorder()
    - paintChildren()
- the trick about painting is knowing *when*

painting and repainting

- interactive displays are 2.5-dimensional
  - x, y, plus depth
    - depth is just an ordering (hence 2.5 rather than 3)
  - depth adds complexity
    - objects move around, causing changes in what they obscure and reveal
    - complicates painting procedures

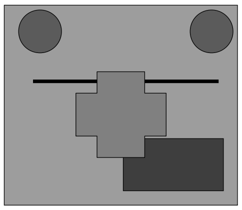
painting and repainting

- move the red triangle to the left
  - paint new circle
  - paint old space in background color



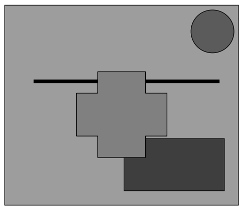
painting and repainting

- move the red triangle to the left
  - paint new circle
  - paint old space in background color



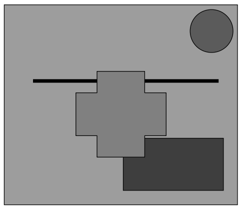
painting and repainting

- move the red triangle to the left
  - paint new circle
  - paint old space in background color



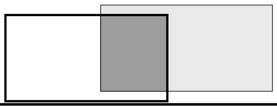
painting and repainting

- move the grey star down and right
  - simple solution clearly doesn’t work



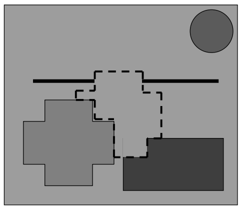
painting and repainting

- demage/redraw solution
  - view informs renderer what areas need to be redrawn
  - window system batches them
  - window system issue redraw instructions
- how is this better?
  - window system uses *clipping*
    - clipping restricts drawing operations to a limited area
    - redraws are clipped to the damaged regions



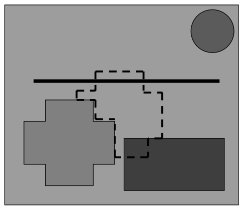
painting and repainting

- two features of this strategy
  - clipping is handled by the graphics system
    - no need for UI system to be able to redraw *parts*
    - redraw whole object and let the graphics system clip
  - minimise drawing
    - only update damaged regions



painting and repainting

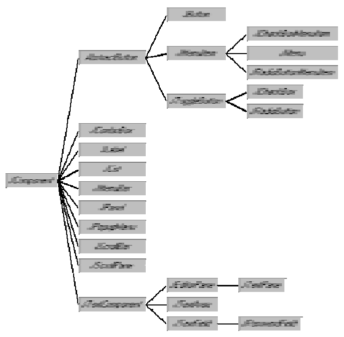
- two features of this strategy
  - simplifies painting needs
    - clipping is handled by graphics system directly
    - no need for UI system to be able to redraw *parts*
    - redraw whole object and let the graphics system clip
  - minimise drawing
    - only update damaged regions



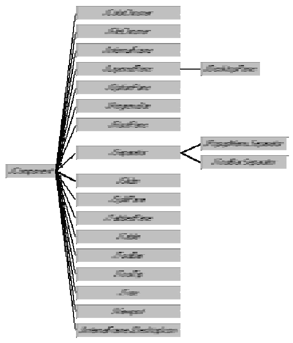
swing widgets

- swing widgets use JavaBean conventions
  - properties controlled through getFoo() and setFoo()
  - widget tree rooted in JComponent
- many of the widgets are containers
  - e.g. panels, menus,
  - call add() to add subcomponents

basic widgets



expanded widgets



buttons

- (almost) simplest widget – JButton
  - button has:
    - graphical properties
    - label
    - ActionListeners
      - JButton has basically only one action – being pressed
  - buttons can use text or icon (or both)

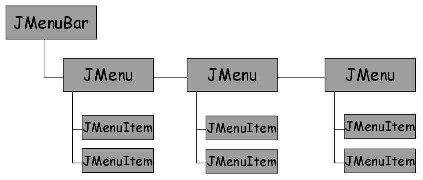
```
JButton b1 = new JButton(“Open”);
b1.addActionListener(new ActionListener() {
  public void actionPerformed(ActionEvent ae) {
    …
```

okay, so I lied…

- there’s a lot more I won’t go into here
  - even buttons turn out to be more complex
  - AbstractButton class
    - alignment properties
    - rollover behaviour
    - keyboard accelerators…

menus and menuitems

- various menu-related classes
  - JPopupMenu
  - JMenuBar

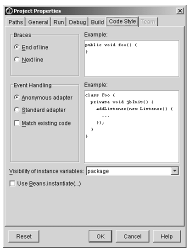


menus

- JMenuItems have individual action listeners
  - menu objects play more structural role
- menubar
  - class is JMenuBar (contains JMenus)
  - only JFrame has a menubar
    - explicitly set with JFrame.setJMenuBar()
- popup menus
  - class is JPopupMenu
  - no automatic support, need to pop it up by hand

radio buttons

- radio buttons require two levels of grouping
  - need to be graphically laid out on the screen
  - need to be grouped into "sets"
  - so, two collections
    - panel, etc, for layout
    - ButtonGroup for grouping
      - determines exclusion criteria
      - holds final value



Braces: [✓ End of line] Next line

Example:

```
public void foo() {
}
```

Event Handling: [✓ Anonymous adapter] Standard adapter

☐ Match existing code

Example:

```
class Foo {
  private void jbInit() {
    addListener(new Listener() {
      ...
    });
  }
}
```

Visibility of instance variables: package

☐ Use Beans.instantiate(...)

lists

- basic multiple choice selection
  - select single or multiple elements
  - elements can be pretty much anything
    - in AWT, only strings…

lists follow MVC

- lists are actually more complex
- list uses:
  - ListModel
    - describes the content of the list
  - ListSelectionModel
    - describes which elements of the list are
  - ListCellRenderer (view)
    - renders the contents of a given item
      - if item is a File, then ListCellRenderer could give filename, pathname, icon
      - need to render both selected and unselected items

trees

- JTree is a more complex MVC-based class
  - trees are a more complex data structure
    - so the model becomes more complex
    - and so does the selection model
  - Swing provides some basic implementations
    - DefaultTreeModel
      - tree of DefaultMutableTreeNodes
      - notification model

next time

- looking at graphical design issues
  - screen design
  - visual design features
- reading
  - Preece ch 4.2, ch 5