

Swing: Menus, Scrollbars, Window Events, Drawing

17 Feb 2010

CMPT166

Dr. Sean Ho

Trinity Western University

Menus: JMenuBar

- A **JMenuBar** is a top-level **container** for menus:
 - **JMenuBar bar = new JMenuBar();**
- You can either **add()** it to the window and use the panel's regular **layout** manager:
 - **add(bar); // in constructor of window**
- Or use the **JFrame**'s **setJMenuBar()** method to lay it out at the **top** of the window:
 - **setJMenuBar(bar);**
- You may have **multiple** menubars per window
- Each menubar may contain **menus** and **items**

Menus: JMenu and JMenuItem

- A **JMenu** represents one **menu** (e.g., “File”)
 - ◆ **JMenu fileMenu = new JMenu();**
 - ◆ **bar.add(fileMenu);**
- Contains menu items: **JMenuItem**
 - ◆ **JMenuItem saveItem = new JMenuItem(“Save”);**
 - ◆ **fileMenu.add(saveItem);**
- Attach a **handler** to the menu item:
 - ◆ **saveItem.addActionListener(handler);**
- **JMenu** is itself a subclass of **JMenuItem**:
allows **nested submenus**

Scroll bars

- Widgets can be contained inside **scroll panes**: show only a **viewport** of the whole widget
- e.g., a **text area**:
 - ◆ `JTextArea blogEntry = new JTextArea(10, 40)`
 - Only shows **10** lines, **40** characters of text
 - ◆ `JScrollPane scrBlog = new JScrollPane(blogEntry);`
 - **Wrap** in a scroll pane
 - ◆ `add(scrBlog);`
 - **Add** to a panel or window
- Scroll bar **policy**: whether to show a scrollbar
 - ◆ `.setVerticalScrollBarPolicy(JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED);`

Window events

- We have seen: **ActionEvent** (button, menu)
 - also **InputEvent** (KeyEvent, MouseEvent)
- A **WindowEvent** is sent when the window interacts with the OS windowing system:
 - **opening, closing, iconifying, activating**
- A JFrame can register a **window listener** to handle these events:
 - ◆ `myJFrame.setWindowListener(winevents);`
- This handler must implement the **WindowListener** interface

Window listeners

- Implementing **WindowListener** means providing:
 - ◆ `class WinEvents implements WindowListener {
 public void windowOpened(WindowEvent e);`
 - ◆ Also `windowClosing`, `windowClosed`,
`windowIconified`, `windowDeiconified`,
`windowActivated`, `windowDeactivated`
- **Closing**: once the close button is clicked
- **Closed**: after the window is done
- **Activated**: usually when a window is clicked in
 - Only one window may be active at a time

WindowAdapter class

- Implementing the **WindowListener** interface means needing to implement **all** its methods, even if you don't need them
- **WindowAdapter** is an **abstract superclass** that implements **WindowListener** and provides **default** blank bodies for the methods
- **Subclass** **WindowAdapter** and **override** just the ones you need:
 - ◆ `class WinEvents extends WindowAdapter {
 public void windowClosed(WindowEvent e) {`

Swing graphics: .paint()


- **JFrames** have a **.paint()** method, which **draws** the window on the screen
 - To do our own drawing, **override paint()**
 - Make sure to call **super.paint() first** to draw the JFrame, then do our own drawing on top
- **paint()** takes a **Graphics** context as its **argument**
 - Drawing routines are **methods** of **Graphics**

```
public class SmileyFace extends JFrame {  
    public void paint( Graphics g ) {  
        super.paint( g );  
        g.drawOval( .... );  
    }  
}
```


paint() vs. paintComponent()

- JFrames: use `paint()` method
- JPanels and other JComponents: use `paintComponent()`
- `paint()` and `paintComponent()` are only called when a **redraw** is necessary
 - e.g., **expose** after being covered
- If you make a change and want to **request** a redraw, call
 - `repaint()` (method of `JFrame` or `JComponents`)

Lines and rectangles

- ◆ `import java.awt.Graphics;`
 - `g.drawLine(int x1, int y1, int x2, int y2);`
 - Coordinates in pixels from **top-left** of component
 - `drawRect(x, y, w, h), fillRect`
 - `(x,y)` is **top-left** corner of rectangle
 - `draw3DRect(x, y, w, h, boolean raised)`
 - Border-shading so it looks **raised** or **sunken**
 - `drawRoundRect(x, y, w, h, arcW, arcH)`
 - Specify diameter of rounded **corners**
- 

Ovals and arcs

- `g.drawOval(x, y, w, h), fillOval`
 - **Circles** are ovals with equal width and height
- `drawArc(x, y, w, h, angle, sweep), fillArc`
 - Specify **starting** angle (0 points to right)
 - Specify how **far** the arc should go (sweep)
 - Angle and sweep are both in **integer degrees**

Colours (colors)

- ◆ `import java.awt.Color;`
- Set the current **drawing colour** before drawing the object:
 - ◆ `g.setColor(Color.BLUE);`
 - ◆ `g.drawArc(50, 50, 100, 100, 200, 140);`
 - ◆ `g.setColor(Color(0.7, 0.9, 0.1));`
 - ◆ `g.drawOval(80, 80, 40, 40);`
- A few **named** colours, or use an **RGB** triple
- `JColorChooser`: **dialog** to select a Color
 - ◆ `JColorChooser.showDialog(this, "title", defaultColor);`