

CMSC 433 – Programming Language
Technologies and Paradigms
Spring 2005

Graphical User Interfaces
April 19, 2005

(Incomplete) History of GUIs

- 1973: Xerox Alto
 - 3-button mouse, bit-mapped display, windows
- 1981: Xerox Star
 - Double-clicking, overlapping windows, dialogs
- 1983: Apple Lisa
 - Pull-down menus and menu bars
- 1984: Apple Macintosh, X Windows
- 1985: Microsoft Windows

(from <http://toastytech.com/guis/guitimeline.html>)

2

Why a GUI Toolkit/API?

- Easier to build GUIs
- Consistency of interface
 - Among applications
 - Between platforms (in Java)

3

GUI Basics

- Interface consists of many *components*
 - Windows, menus, buttons
- User can perform variety of operations
 - Move window
 - Click button
 - Press key
- These are *events*

4

A Note on AWT versus Swing

- Abstract Window Toolkit (AWT)
 - Original Java GUI
 - Still supported, but being replaced
- Swing -- souped-up AWT
 - Pluggable look-and-feel
 - More, better widgets
- AWT class X has corresp. Swing class JX
 - E.g. Button vs. JButton

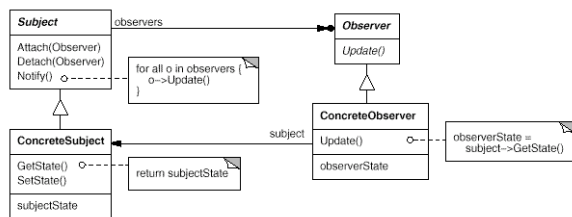
5

Creating Components

- Components are the usual widgets
 - JButton, JPopupMenu, JScrollBar
 - Components know how to draw themselves
 - But they (generally) don't say where they are
- In Swing, components live in containers
 - JFrame, JDialog, JApplet
 - Containers decide how to layout components
 - Various layout algorithms

6

Observer pattern



7

Programming with Events

- Initialize objects
 - Create everything on the screen
 - Register your event handlers (observers)
- Then toolkit takes over with event loop

```
while (true) {
    e = getEvent();
    findEventHandler(e).actionPerformed(e);
}
```

8

Adding Observers (Listeners)

- Components have a set of listeners
 - void addActionListener(ActionListener l)
 - void addKeyListener(KeyListener l)
 - void addMouseListener(MouseListener l)
- Possible listeners vary with component

9

Listener Interfaces

```
public interface ActionListener {
    void actionPerformed(ActionEvent e);
}

public interface MouseInputListener {
    void mouseClicked(MouseEvent e);
    void mouseEntered(MouseEvent e);
    void mousePressed(MouseEvent e);
    ...
}
```

10

Events

- Event objects contain the detailed info

```
public class MouseEvent {
    ...
    int getClickCount();
    Point getPoint();
}
```
- Tradeoff between number of event classes and amount of info in each event

11

SwingApplication.java

- (Optional) Step 1: Install look and feel

```
public static void main(String[] args) {
    try {
        UIManager.setLookAndFeel(
            UIManager.getCrossPlatformLookAndFeelClassName()
        );
    } catch (Exception e) { }
    ...
}
```

- Default is to use “native” look and feel

12

Step 2: Make the Components

```
JFrame frame = new JFrame("SwingApplication");
SwingApplication app = new SwingApplication();
Component contents = app.createComponents();
frame.getContentPane().add(contents,
    BorderLayout.CENTER);
```

- Frames are windows
 - `ContentPane` is the display area (w/o menu bar)
- contents added to the frame
 - Layed out as `BorderLayout.CENTER`

13

Step 3: Install any Listeners

```
frame.addWindowListener(new WindowAdapter() {
    public void windowClosing(WindowEvent e) {
        System.exit(0);
    }
});
```

- `WindowAdapter` implements `WindowListener`
 - Contains default do-nothing methods
- Notice we ignore the value of `e`

14

Step 4: Make Window Visible

```
frame.pack();
frame.setVisible(true);
}
```

- `pack()` sizes window to fit subcomponents
- `setVisible(true)` shows the window
 - Not drawn until this is set
- `main(String[] args)` exits
 - Swing event thread handles events

15

Layout and Painting

- Swing places components according to layout manager
 - Don't need to worry about screen size etc.
 - (Not really true)
- Window painted from back to front
 - Component paints itself before subcomponents
 - Double-buffered so painting looks smooth

16

The Event-Dispatching Thread

- All events are dispatched by a single thread
 - Implies events dispatched in order
 - Also implies next event not processed until current event dispatch returns
 - Need to make event dispatch fast
 - (Re)-painting also done in event-dispatching thread
- What if another thread needs to modify GUI?
 - Swing library mostly *not* thread safe

17

Single-Thread Rule

“Once a Swing component has been realized, all code that might affect or depend on the state of that component should be executed in the event dispatching thread.”

java.sun.com tutorial

- A component is realized after
 - setVisible(true) (== deprecated show())
 - pack()

18

Dealing with Threads

- Wait a second, what about the example?

```
f.pack();           // f is realized here
f.setVisible(true); // so isn't this unsafe?
```

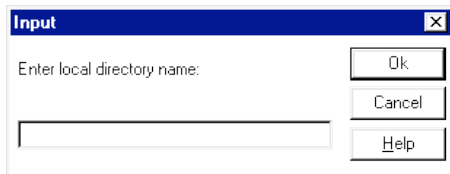
 - Apparently this “usually” works, so it’s OK (?)
- A few methods can be used from any thread
- SwingUtilities.invokeLater(Runnable r)
 - Event-dispatching thread will invoke r
 - (after all pending events dealt with)

19

Making Event Dispatch Fast

- Standard problem in event or interrupt-driven systems
 - Short tasks in response to events are fine
- Divide long tasks into top half, bottom half
 - Top half is quick; typically, buffers
 - Bottom half runs in separate thread
 - Consumes data from buffer
 - Can use invokeLater() if it eventually updates GUI₂₀

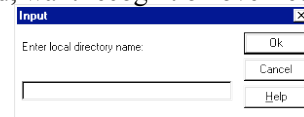
UI Hall of Fame or Shame?



21

UI Hall of Shame

- Application directory dialog (not system)
 - Inconsistent
- Requires typing a path name
 - No browse option
 - What if you have many directories?
- Instead, want recognition over recall



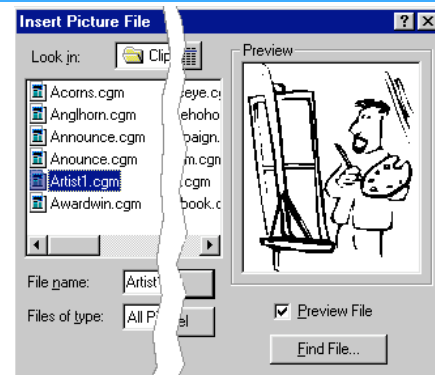
22

Tips

- Don't make the user look stupid
- The goal of all software users is to be more effective

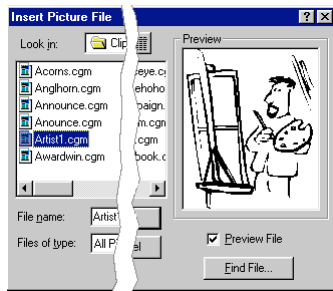
23

UI Hall of Fame or Shame?



24

UI Hall of Fame



- MS Publisher
- Modified file dialog
- Recognition based
 - Browsable names
 - Browsable content
 - Preview can be turned off
 - Optional search
- Design Suggestion:
 - Add entry box for directly typing in name with auto-completion of filenames

25

Tips

- User interfaces that directly wrap underlying systems are often bad

26