Windows and Events

created originally by Brian Bailey
Announcements

• Review next time
• Midterm next Friday
UI Architecture

- Applications
- UI Builders and Runtimes
- Frameworks
- Toolkits
- Windowing System
- Operating System
Windowing System

- Manages windows and their relationships
  - window hierarchy

- Manages events and dispatch of events to individual windows
  - an event is a notification of an occurrence such as mouse click, key press, or timer pop
Windows

• A window is a rectangular area on screen
  – enables a user to view output
  – enables app to solicit input events from area
  – inexpensive to create and manage

• A window has properties
  – visibility, size, border, color, and more

• Almost every widget maps to a window

• Windowing system manages the windows
Window Hierarchy

- Windows arranged in a tree (root desktop)
  - defines a *stacking order* of the windows
Window Hierarchy

• Windows arranged in a tree
  – defines a *stacking order* of the windows
Window Hierarchy

- Windows arranged in a tree
  - defines a *stacking order* of the windows
  - Point P (black circle) is *in* a window if it is contained within its visible area
Window Exposure

- User brings B forward
  - parts of B and E have become exposed
- Should
  - B receive one expose event and E just one?
  - B receive one expose event and redraw E?
- Managing is difficult!
Draw/Redraw Windows

• When window is first displayed
  – after being created
  – re-displayed after having been minimized

• When window content is updated
  – e.g., when the display is manipulated

• When obscuring windows are moved
When is Drawing Actually Done

- Right away?
- After next input event?
- When idle?
- When idle or within time limit?
Window Coordinates

• Use pixels as the coordinates
• Origin is at the upper-left hand corner
  – X increases toward the right
  – Y increases toward the bottom
• Coordinates always relative to a specific window
UI Events

• An event is an *asynchronous* notification of user action, timer pop, status change

• UI applications are *event-driven*
  – loop of waiting for an event and responding
  – contrast to top-down sequential flow
Event Sources

• Physical objects
  – Keyboard: key pressed, released, typed
  – Mouse: button pressed/released/clicked, mouse motion, mouse drag …

• Virtual objects
  – Window: expose, enter, leave, resize, focus…
  – Widget: child added/removed
  – Selection: item selected in tree, list control…
Event Structure

- Event type (e.g., mouse press, key press, exposure, focus change, etc.)
- Window identifier
- (X, Y) position of mouse
- Time of event
- Modifiers (shift, caps lock, etc)
- Much more
**Windows and Events**

- Windowing system associates events with a specific window (usually focus window)
  - mouse, keyboard, and window events

- Windowing system continuously tracks mouse and which window contains it
  - containing window $\neq$ the focus window

- Windowing system appends an event onto an application’s event queue
  - uses the application that owns the window
  - application retrieves event and handles it
Example Event Flow

- Mouse click generates hardware interrupt
- OS maps interrupt to system handler (a routine in the windowing system)
- Windowing system
  - identifies window associated with event and application that owns the window
  - constructs the event structure
  - appends event onto the application’s event queue
- Application
  - removes event from event queue
  - maps window and event to a registered handler
  - invokes that handler
Event-Driven Programming

• Applications respond to events
  – no central flow of control
  – setup interface, handle events, and clean-up

• Core of the application is the event loop
  – wait for event, handle event, repeat
  – input handlers must be fast - [50, 500ms]
The Event Loop

While (true) {
    Event event = get_next_event();
    Handler handler = lookup_handler(event);
    handle(event);
}

Note: handlers are indexed in a table by event type, window, and other detail
Program Structure

• **Substantial initialization code**
  – construct data objects and user interface
  – register event handlers
  – do any setup processing

• **Event loop core**
  – provided by most toolkits

• **Special cases**
  – Modal dialogs
Java History

• Gosling et al. envisioned merger of consumer and computing devices
  – developed a language to enable development and be portable across devices
  – ahead of its time in a niche market

• Language found a new home on the Web where it could bring static pages to life
Java Language

• Object-oriented language
  – classes, objects, inheritance, polymorphism, exceptions, interfaces, etc.
  – very clean and pure language model

• Applications are cross-platform
  – compile code into a “byte code”
  – develop a virtual machine for each platform that can interpret the byte code

• Well-documented and supported
  – lots of java books available
AWT and Swing

- AWT is the windowing system
  - support basic building blocks from which to construct higher-level controls for toolkits
- Swing is higher-level toolkit built on AWT
  - buttons, sliders, edit boxes, menus, …
AWT and Swing (cont.)

• Java 1 (or most of it) used a peer model
  – each widget created in the AWT maps to a widget in the platform-specific toolkit
  – difficult to maintain cross-platform feel because a widget may behave differently on different platforms
  – must write AWT part of the JVM for each platform

• Java 2 uses a pluggable look and feel model
  – use a single window and drawing commands
  – do everything else in Java
  – pluggable look and feel, but performance deficient
import java.awt.*;
import java.awt.event.*;

public class SimpleWindow extends Frame {

    public static void main(String args[]) {
        SimpleWindow sw = new SimpleWindow();
        sw.pack();
        sw.show();
        sw.setBounds(225, 250, 640, 480);  // place the window
    }

    public SimpleWindow() {
        setTitle("Simple Window");
        …
    }

    …

    Where is the Event Loop?
public class SimpleWindow extends Frame ... {

    public void paint(Graphics g) {
        super.paint(g);
        g.setColor(new Color(235, 235, 235));
        g.fillRect(0, 380, 150, 100);
        g.setStroke(new BasicStroke(4.0f));
        g.setColor(Color.blue);
        g.drawRect(0, 380, 150, 100);
        g.setColor(Color.black);
        g.drawString("X: " + mousex + " Y: " + mousey, 20, 400);
        g.drawString(eventString, 20, 430);
    }
}
public void paint(Graphics g) {
    ... 
    Rectangle area = g.getClipBounds();
    System.out.println(area);
    ...
    
    g.drawString(area, 20, 430);
}

The clipping rectangle in the graphics object identifies the exposed region
public class SimpleWindow extends Frame implements MouseListener {
    public SimpleWindow() {
        ...
        addMouseListener(this);
    }

    public void mouseEntered(MouseEvent e) {
        eventString = "mouse entered";
        repaint();
    }

    public void mouseExited(MouseEvent e) {
        eventString = "mouse exited";
        repaint();
    }

    Java uses a publisher/subscribe event model
public class SimpleWindow extends Frame implements MouseMotionListener {
    private int mousex, mousey;
    ...
    public SimpleWindow() {
        ...
        addMouseMotionListener(this);
        mousex = mousey = 0;
    }
    ...
    public void mouseMoved(MouseEvent e) {
        mousex = e.getX();
        mousey = e.getY();
        repaint(0, 380, 150, 100);
    }
    ...

Retrieve context from the event object
Higher-level Components

• Construct a simple push button from a window and low-level events
  – demo constructed push button

• Required about 100 lines of code and many desired features were not implemented
  – changing font size, centering text, support for icons, registering callbacks, etc.

• Lesson: Construct higher-level interaction components and place them in a toolkit!
  – Swing, MFC, Motif, Cocoa, etc.
UI Toolkits

• **Programming at the low level is absurd**
  – hundreds of lines of code to manage a single button on the screen
  – handle expose, enter, leave, click events, position text for different font metrics, etc.

• **Large applications are almost impossible when programming at this level**

• **Need higher-level programming abstractions**
Toolkits Provide

• **Widgets**
  – interaction vocabulary

• **Geometry management**
  – widget layout

• **Resource management**
  – defaults, user overrides, internationalization