

The University of Sydney

GUIs with the Java AWT

Joe Thurbon and Alan Fekete
University of Sydney, 2004

GUI

- Pronounced “gooey”
- Graphical User Interface
- A type of User Interface
 - for the user to influence the system state
 - to present information about the system state to the user
 - to guide the user about how to control the system
- Graphical
 - Visual communication

GUIs with Java AWT, University of Sydney 2004

Modern GUI style

- Screen with multiple windows
 - “desktop” metaphor for folders
 - Each window has menu bar, scrollbars
- Mouse
 - move cursor to point at windows, icons, widgets
 - click to select or invoke
 - “drag” to move
- Keyboard
 - For text entry

GUIs with Java AWT, University of Sydney 2004

Early History

- 3 Turing Award winners!
- Ivan Sutherland (MIT Lincoln Labs)
 - “Sketchpad” (1963 PhD thesis)
 - Real-time 2D graphics display
- Doug Engelbart (SRI)
 - Mouse as input device (1963)
 - NLS (1968): Multiple windows, hyperlinks, mouse, chord keyset
- Butler Lampson (Xerox PARC)
 - Alto (1973)
 - First “personal computer”
 - WIMP (windows, icons, mouse, pointer) style

Source: www.wikipedia.org
GUIs with Java AWT, University of Sydney 2004

Commercial History

- Xerox Star (1981)
 - Based on Alto, commercial failure
- Apple Lisa (1983)
 - ?Based on Alto?, commercial failure
- Apple Macintosh (1984)
 - Based on Lisa, quite successful
- Microsoft Windows (1985)
 - ?Based on Macintosh?, hugely successful
- CDE, KDE, Gnome (1990s)
 - Desktop front-ends for Unix

Source: www.wikipedia.org

GUIs with Java AWT, University of Sydney 2004

Comparison

- Compared to text-based console style
 - Much easier to learn
 - Uniform idiom
 - Easy to remember or guess
 - But: Much slower for highly expert users
 - large hand movements
 - But: much harder to automate tasks
 - Unless automation has access to programmatic or textual interface

GUIs with Java AWT, University of Sydney 2004

Separation of concerns

- To give high coherence and low coupling, the code should be divided
 - the classes that perform the actual tasks
 - Maintain system state
 - Perform calculations
 - Eg Bank: Account, Customer, InvestmentFund
 - the classes that provide the UI
- This allows multiple UIs on same system

GUIs with Java AWT, University of Sydney 2004

What is AWT?

- Abstract Windowing Toolkit: provides various classes that are important in the GUI
 - Easily write code for a GUI using instances of these classes
 - Calling their API
 - Extend the classes for fancier GUIs
 - Standard Package `java.awt`
 - Introduced in Java 1.0
 - Major change to event model in Java 1.1
 - Since Java 1.2, Swing is more widespread

GUIs with Java AWT, University of Sydney 2004

Basic AWT classes

• Three Basic categories:

- Related to Basic Graphics
- Components
 - Related to the Widgets in a window
 - Related to arranging the widgets
 - Supporting classes
- Related to the events that occur

GUIs with Java AWT, University of Sydney 2004

Graphics-related classes

• Primitive shapes : defining a shape

- java.awt.Point,
- java.awt.Rectangle,
- java.awt.Polygon
- etc.

GUIs with Java AWT, University of Sydney 2004

Graphics-related classes II

• Paint related: carry out actual drawing/painting

- java.awt.Graphics/Graphics2D
- java.awt.Color
- java.awt.Font
- java.awt.Image

You normally deal with these classes in the paint() method.

GUIs with Java AWT, University of Sydney 2004

Widget classes

- A typical UI has many controls placed in various positions within a window
- Classes for different sorts of widget
 - Button
 - Label
 - CheckBox
 - TextField
 - Scrollbar
 - MenuBar

GUIs with Java AWT, University of Sydney 2004

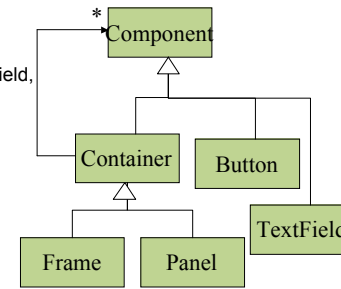
Containers

- Classes which can have multiple controls within them
- Frame: top-level window with title and border
- Window: top-level window but no border or menubar
- Panel: very simple container

GUIs with Java AWT, University of Sydney 2004

Composite design pattern

- A top-level window could have several controls
 - Eg 2 buttons, a textfield, and a subpanel
- Define superclass Component
- Define Container to represent any component that can contain other components
 - add(Component c)



GUIs with Java AWT, University of Sydney 2004

Methods of Component

- void setVisible(boolean)
- void setLocation(int, int)
- void setSize(int, int)
- void setBackground(Color)
- void update(Graphics)
- Similar getYY methods
- Many other methods; see API

GUIs with Java AWT, University of Sydney 2004

Layout-related classes

- Layout managers:
 - FlowLayout, BorderLayout, GridBagLayout,
 - Each defines a way to arrange the subcomponents within the space in the window where the container is displayed
 - Layout depends on manager and on order components are added to container

GUIs with Java AWT, University of Sydney 2004

UI related classes

- Other supporting classes:
 - Menu, MenuItem, PrintJob, PopupMenu, Dialog, Cursor

GUIs with Java AWT, University of Sydney 2004

Event related classes

- `java.awt.Event`
 - This class exists only for backward compatibility.
 - From obsolete event model (JDK1.0)
 - You are not supposed to use this class to handle events!

GUIs with Java AWT, University of Sydney 2004

Event related classes

- `AWTEvent`, `AWTEventMulticaster`
 - Used with new event model (JDK1.1 +)
- `java.awt.AWTEvent`
 - Implementation: `java.awt.event.*`,
- `java.awt.AWTEventMulticaster`
 - Provides thread-safe multicasting mechanism for `AWTEvent`.

GUIs with Java AWT, University of Sydney 2004

Example: Graphics/Paint process

```
/** A subclass of java.awt.Canvas*/
public class HelloCanvas extends Canvas
{
    /** paint method */
    public void paint( Graphics g ) {
        g.drawString( "Hello!", 50, 30 );
    }
}
```

GUIs with Java AWT, University of Sydney 2004

Example: - cont'd

- `java.awt.Canvas` extends `java.awt.Component`
- All visual components (subclass of `java.awt.Component`) receive an event indicating that it needs to paint itself.
- When the event is received, “`paint()`” method will be automatically called.

GUIs with Java AWT, University of Sydney 2004

Example: - cont'd

- The paint process of `java.awt.Component` is the one of events automatically generated and handled by AWT components.
- Other types of events are (usually) automatically generated but need to be handled explicitly by some other objects.
 - keyboard, mouse, window, etc.

GUIs with Java AWT, University of Sydney 2004

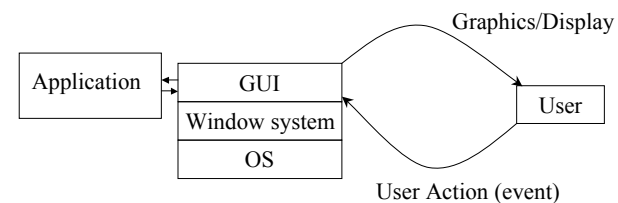
Execution pattern for a GUI

- The execution of a GUI for an application requires complex interaction between window graphics and user input activities.
- Recurring process of :
 - GUI presents some information
 - User does something
 - GUI presents revised information

GUIs with Java AWT, University of Sydney 2004

Execution pattern for a GUI - cntd

- A GUI has a quite different execution pattern compared to traditional computation
 - Endless closed loop of interaction, vs transforming input data to output data (function evaluation).



GUIs with Java AWT, University of Sydney 2004

Variety of actions

- Mouse moves
- Mouse is clicked
- Component is moved
- Window is closed (from menu bar or corner)
- Window is resized
- Etc etc

GUIs with Java AWT, University of Sydney 2004

Event driven programming

- How can programmer code what should happen in the GUI?
 - The user can perform so many sequences of actions
 - Control flow is impractical
- Solution: programmer writes a method for each different action
 - Eg `void mouseEntered() { // do something....`
 - System takes care of invoking the correct method

GUIs with Java AWT, University of Sydney 2004

Event driven programming – cntd

- Even the event-driven system needs some sequential processes at the start.
 - Create/initialize data
 - Construct GUIs
 - etc.
- After the pre-work, the event-driven system goes into an infinite loop
 - In the infinite loop, the system detects events and invokes appropriate developer-written methods.

GUIs with Java AWT, University of Sydney 2004

Event loop in AWT

- JVM assigns a thread, which is separate from the main thread, to handle `java.awt.*` related events
 - One after another
 - For each event, the appropriate handler is called
- No need for application developer to explicitly write an infinite loop.

GUIs with Java AWT, University of Sydney 2004

History of Java Event Model

- **Java 1.0**
 - More straightforward approach
 - Eg Button has mouseDown() method
 - Classes and methods remain in SDK for backward compatibility, but they are deprecated
- **Java 1.1**
 - Many new and changed classes
 - New style of placing handlers in “Listener” classes
 - Concepts (and many classes) have remained in Swing

GUIs with Java AWT, University of Sydney 2004

Delegation model

- Eg if user moves mouse so cursor enters the region occupied by a drawing canvas
 - Source is the canvas
 - MouseEvent m is constructed by AWT
 - The mouseEntered(m) method is called in each MouseListener object
 - Referenced by the source

GUIs with Java AWT, University of Sydney 2004

Event classes

```

java.lang.Object
|-java.awt.Event (obsolete)
|-java.util.EventObject
|-java.awt.AWTEvent
  |-java.awt.event.ActionEvent
  |-java.awt.event.AdjustmentEvent
  |-java.awt.event.ItemEvent
  |-java.awt.event.TextEvent
  |-java.awt.event.ComponentEvent
    |-java.awt.event.ContainerEvent
    |-java.awt.event.FocusEvent
    |-java.awt.event.PaintEvent
    |-java.awt.event.WindowEvent
    |-java.awt.event.InputEvent |
      |-java.awt.event.KeyEvent
      |-java.awt.event.MouseEvent
    
```

GUIs with Java AWT, University of Sydney 2004

Generic Concept of an Event

- An event is considered as a more generic entity for the communication between components.
- It is quite ok to create an event which is nothing to do with user inputs nor a window system.
 - By extending java.util.EventObject

GUIs with Java AWT, University of Sydney 2004

Varieties of Listener

- User Input
 - MouseListener
 - MouseMotionListener
 - KeyListener
- Component
 - ComponentListener
 - ContainerLiseter
 - FocusListener
- Window
 - WindowListener
- GUI
 - ActionListener
 - ItemListener
 - TextListener
 - AdjustmentListener

GUIs with Java AWT, University of Sydney 2004

Listener interfaces

- Eg WindowListener must have methods
 - windowActivated(WindowEvent e)
 - windowClosed(WindowEvent e)
 - windowClosing(WindowEvent e)
 - windowDeactivated(WindowEvent e)
 - windowDeiconified(WindowEvent e)
 - windowIconified(WindowEvent e)
 - windowOpened(WindowEvent e)

GUIs with Java AWT, University of Sydney 2004

Adding/Removing Event Listeners

- All event listeners need to be added to the event source.
 - General form: addXListener()
 - Eg: addComponentListener(), addMouseListener(), addMouseMotionListener(), etc.
- Source may be predefined component from AWT, or it may be a subclass written by programmer
- Event listeners which no longer need to receive events should be released from the event source:
 - General form: removeXListener().
 - Eg removeComponentListener(), removeMouseListener(), removeMouseMotionListener(), etc.

GUIs with Java AWT, University of Sydney 2004

Example: Event Processing

```

/** Changes text color according to a mouse event */
import java.awt.*;
import java.awt.event.*;
public class Flicker extends Canvas {
    /** mouse listener */
    private FlickHandler handler;
    /** text */
    private String message = "Hello!";
    /** a flag to indicate the mouse position */
    boolean stateFlag = false;
    /** constructor */
    public Flicker() {
        super();
        setSize( 200, 40 );
        handler = new FlickHandler();
        addMouseListener( handler); // add a listener
    }
  
```

GUIs with Java AWT, University of Sydney 2004

Example - cntd

```

/** drawing method */
public void paint( Graphics g ) {
    if( stateFlag == false ) { // mouse out
        g.setColor( Color.yellow );
        g.fill3DRect( 1, 1, 198, 38, true );
        g.setColor( Color.black );
        g.drawString( message, 5, 22 );
    } else { // mouse in
        g.setColor( Color.black );
        g.fill3DRect( 1, 1, 198, 38, false );
        g.setColor( Color.yellow );
        g.drawString( message, 5, 22 );
    }
    :
    :
}

```

GUIs with Java AWT, University of Sydney 2004

```

/** FlickHandler processes mouse events */
import java.awt.*;
import java.awt.event.*;
public class FlickHandler
    implements MouseListener {
    /** called when the mouse enters */
    public void mouseEntered(MouseEvent evt) {
        Flicker flick = (Flicker) evt.getSource();
        flick.stateFlag = true;
        flick.repaint();
    }

    /** called when the mouse exits */
    public void mouseExited(MouseEvent evt) {
        Flicker flick = (Flicker) evt.getSource();
        flick.stateFlag = false;
        flick.repaint();
    }

    /** called when the mouse is pressed */
    public void mousePressed(MouseEvent evt) {
    }

    /** called when the mouse is released */
    public void mouseReleased(MouseEvent evt) {
    }

    /** called when the mouse is clicked */
    public void mouseClicked(MouseEvent evt) {
    }
}

```

GUIs with Java AWT, University of Sydney 2004

ActionEvent

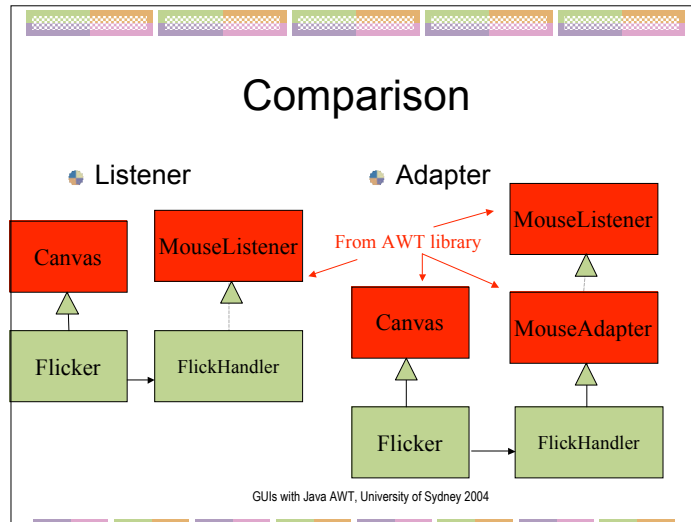
- One of the most common cases is for GUI to respond to events that indicate a selection or data entry
 - Eg mouse clicked when cursor over a button
 - Eg user presses “enter” key in a text field
- These are expressed as ActionEvent
 - Handled by actionPerformed() method in an ActionListener
 - Source is relevant component (button, text field, etc)

GUIs with Java AWT, University of Sydney 2004

Using adapters

- We can write a class to deal with events by implementing the appropriate Listener interface
 - Eg FlickHandler implements MouseListener
- Your code must provide a body for every method of the interface
 - Including those you aren't interested in
- Library contains a MouseAdapter class which has all necessary methods with empty bodies
 - An object of this class would be useless
- A subclass of the adapter can override the method of interest and ignore the others, and so implement the appropriate Listener.

GUIs with Java AWT, University of Sydney 2004



Example, as an adapter

```

import java.awt.*;
import java.awt.event.*;
public class FlickHandler extends MouseAdapter {
    /** called when the mouse enters */
    public void mouseEntered(MouseEvent evt) {
        Flicker flick = (Flicker) evt.getSource();
        flick.stateFlag = true;
        flick.repaint();
    }

    /** called when the mouse exits */
    public void mouseExited(MouseEvent evt) {
        Flicker flick = (Flicker) evt.getSource();
        flick.stateFlag = false;
        flick.repaint();
    }
    // No other methods need be written
}
    
```

GUIs with Java AWT, University of Sydney 2004

Listening to oneself

- Conceptually, source component and listener are separate classes
- Event causes invocation of method in listener
- If you want, you can write the event-handling methods in the component
- Same class takes two roles

Why can't this be done as adapter?

GUIs with Java AWT, University of Sydney 2004

Example: Listening to oneself

```

/** Changes text color according to a mouse event */
import java.awt.*;
import java.awt.event.*;
public class Flicker extends Canvas implements MouseListener{
    /** text */
    private String message = "Hello!";
    /** a flag to indicate the mouse position */
    boolean stateFlag = false;
    /** constructor */
    public Flicker() {
        super();
        setSize( 200, 40 );
        addMouseListener( this ); // add an adapter
    }
    // write code for mouseEntered(), mouseExited() etc
}
    
```

GUIs with Java AWT, University of Sydney 2004

Making responsive GUIs

- If handling the event takes a long time
 - AWT event-handling thread is busy
 - It doesn't process next event till previous one's processing is finished
 - This annoys users, who have short attention spans
 - At minimum user wants display of progress and possibility of cancellation

GUIs with Java AWT, University of Sydney 2004

Multi-threaded GUI handling

- So, event handler can construct and start a new thread to handle a slow operation
- Event-handling thread continues and can handle next event (eg cancel button pushed!)
 - If necessary, it can communicate with previously started threads by standard mechanisms

GUIs with Java AWT, University of Sydney 2004

Acknowledgements

- Funding from Australian Government Science Lectureship "Building the Internet Workforce"

GUIs with Java AWT, University of Sydney 2004