The AWT, Applets, and Swing

Mark Allen Weiss Copyright 1996, 1999, 2000

1

Outline of Topics

- The Abstract Window Toolkit
 - Basic ideas
 - User interfaces
 - Output items: canvases and graphics
 - Events
 - Fancy layouts
- Applets
 - HTML files
 - Converting an application to an applet
 - Restrictions

Basic Ideas

- The Abstract Window Toolkit (AWT) is a GUI toolkit designed to work across multiple platforms.
- Not nearly as fancy as MFC.
- Event-driven: the window is displayed, and when things happen, an event handler is called. Generally, the default event handler is to do nothing.
- Must import java.awt.* and java.awt.event.*

3

Evolution of GUIs

- Java 1.0
 - basic AWT components
 - terrible event model
- Java 1.1
 - same AWT components
 - new event model
- Java 1.2
 - new fancy Swing components
 - same event model as 1.1
 - not supported in all browsers, but plug-in available

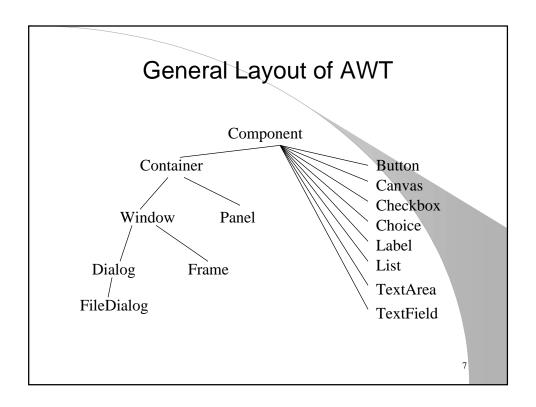
To Swing or Not?

- If you are writing applications, Swing is by far the preferable option
 - faster
 - prettier
 - more flexible
- If you are writing applets, decision is harder
 - consumers not likely to have Java 1.2; can give it to them, but download will be time-consuming. Most consumers won't bother and will go elsewhere
 - corporate clients can be forced to go to Java 1.2

5

AWT vs Swing

- Concepts are all the same.
- We will discuss AWT, so applets will work unobtrusively.
- Swing talk to follow separately. In this class:
 - Use Swing for applications.
 - For applets, consider using HTML forms and server-side servlets. If not, include Swing library in jar file for distribution and hope that user has a fast connection.



Component

- The parent class of many of the AWT classes.
- Represents something that has a position and a size and can be painted on the screen as well as receive input events.
- This is an abstract class and may not be instantiated.
- Some important methods:

```
public void paint( Graphics g );
public void show( );
public void addComponentListener( ComponentListener 1 )
Various routines to set fonts, handle mouse events, etc.
```

Container

- The parent class of all components and one that can contain other classes.
- Has a useful helper object called a LayoutManager, which is a class that positions components inside the container.
- Some methods:

```
void setLayout( LayoutManager mgr );
void add( Component comp );
void add( Component comp, String name );
```

9

Top Level Windows

- Window: A top-level window that has no border.
- Frame: A top-level window that has a border and can also have an associated MenuBar.
- Dialog: A top-level window used to create dialogs. One subclass of this is the FileDialog.

Panel

- Subclass of Container used inside other containers.
- Used to store collections of objects.
- Does not create a separate window of its own.

1.

Important I/O Components

- Button: A push button.
- Canvas: General-purpose component that lets you paint or trap input events from the user. It can be used to create graphics. Typically, is subclassed to create a custom component.
- Checkbox: A component that has an "on" or "off" state. You can place Checkboxes in a group that allows at most 1 box to be checked.
- Choice: A component that allows the selection of one of a group of choices. Takes up little screen space.

More I/O Components

- Label: A component that displays a String at a certain location.
- List: A scrolling list of strings. Allows one or several items to be selected.
- TextArea: Multiple line text components that allows viewing and editing.
- TextField: A single-line text component that can be used to build forms.

13

Events (Java 1.1 World)

- Each component may receive events.
- Only objects that implement an
 EventListener interface (e.g.
 ActionListener, MouseListener) may
 handle the event. The event is handled by a
 performed method (e.g. actionPerformed)
- Such objects must register, via an addListener, that they will handle the particular event.
- It makes more sense when you see the example.
- Java 1.0 had a different, and very poor alternative.

Most Common Listeners

- ActionListener: button pushes, etc.
- KeyListener: keystroke events (pressing, releasing, typing)
- MouseListener: mouse events (pressing, releasing, clicking, enter/exit)
- MouseMotionListener: mouse moving events (dragging, moving)
- TextListener: text in a component changes
- WindowListener: window events (closing, iconifying, etc)

15

Event Handler Classes

- Event handler classes need access to components whose state information might need to be queried or changed
- Common to use inner classes
- Often anonymous inner classes used
 - syntax can look ugly
 - however, can see what actions will occur more easily when event handler functionality is immediately next to component

Adapter Classes

- Some listener interfaces require you to implement several methods
 - WindowListener has 7!
- You must implement all methods of the interface, not just the ones of interest.
- All listener interfaces with multiple methods have a corresponding Adapter class that implements all methods with empty bodies
 - so, you just extend the adapter class with methods you are interested in; others get default

17

Making A Frame Close

- When frame is closed, WindowEvent is generated; someone should listen and handle windowClosing method.
 - Java 1.1: otherwise, frame stays open
 - Java 1.2: otherwise, frame closes, but event thread continues, even if no other visible components
 - Java 1.3: frame closes, and can make arrangements for event thread to stop if no other visible components

Implementing CloseableFrame

```
import java.awt.*;
import java.awt.event.*;
public class CloseableFrame extends Frame implements WindowListener
 public CloseableFrame( )
   { this(" " ); }
 public CloseableFrame( String title )
    { super( title ); addWindowListener( this ); }
 public void windowClosing( WindowEvent event )
    { System.exit( 0 ); }
 public void windowClosed
                              ( WindowEvent event ) { }
 public void windowDeiconified( WindowEvent event ) { }
 public void windowIconified ( WindowEvent event ) { }
 public void windowActivated ( WindowEvent event ) { }
 public void windowDeactivated( WindowEvent event ) \{\ \}
 public void windowOpened ( WindowEvent event ) { }
```

Slicker CloseableFrame

The Event Dispatch Thread

- When event occurs, it is placed in an event queue
 - the event dispatch thread sequentially empties queue and sequentially calls appropriate methods for registered listeners
 - important that your event handlers finish fast;
 otherwise, program will appear unresponsive
 - spawn off a new thread if you can't handle event quickly
 - new thread should not touch user interface: Swing is not thread safe! Use invokeLater and invokeWait methods in the EventQueue class.

21

Graphics

- Generally need to override the function public void paint(Graphics g)
- Rarely call paint directly.
 - Note: repaint schedules update to be run
 - update redraws the background and calls paint
- Use statements such as

```
g.setColor( Color.red );
g.drawLine( 0, 0, 5, 5 ); // Draw from 0,0 to 5,5
```

How Does Everything Fit Together

- What you have to do:
 - Decide on your basic input elements and text output elements. If the same elements are used twice, make an extra class to store the common functionality.
 - If you are going to use graphics, make a new class that extends Canvas.
 - Pick a layout.
 - Add your input components, text output components, and extended canvas to the layout.
 - Handle events; simplest way is to use a button.

23

Example Time

- A simple example that displays a shape in a small canvas.
- Has several selection items.
- Example shows the same interface in two places.

The Example

- Class GUI defines the basic GUI. It provides a constructor, implements ActionListener, and registers itself as the listener. So the GUI instance catches the "Draw" button push.
- Class GUICanvas extends Canvas and draws the appropriate shape. Provides a constructor, a paint routine, and a public method that can be called from GUI's event listener.
- Class BasicGUI provides a constructor, and a main routine. It builds a top-level frame that contains a GUI.

25

Getting Info From The Input

- Choice or List: use getSelectedItem.
 Also available is getSelectedIndex. For lists with multiple selections allowed, use getSelectedItems or getSelectedIndexes, which return arrays.
- TextField: use getText (returns a String, which may need conversion to an int).
- Checkbox: use getState
- Info from the canvas is obtained by catching the event, such as mouse click, mouse drag, etc.

Simple Layout Managers

- Make sure you use a layout; otherwise, nothing will display.
- null: No layout: you specify the position. This is lots of work and is not platform independent.
- FlowLayout: components are inserted horizontally until no more room; then a new row is started.
- BorderLayout: components are placed in 1
 of 5 places: "North", "South", "East", "West",
 or "Center". You may need to create panels,
 which themselves have BorderLayout.

27

Fancy Layouts

- CardLayout: Saves space; looks ugly in AWT.
- GridLayout: Adds components into a grid. Each grid entry is the same size.
- GridBagLayout: Adds components into a grid, but you can specify which grid cells get covered by which component (so components can have different sizes).

Applets

• A piece of code that can be run by a javaenabled browser.

29

Making an Application an Applet

- import java.applet.*
- Have the class extend Applet.
 - Applet already extends Panel
 - Typically just put a Panel inside of the Applet and you are set.
 - No main needed.

```
import java.applet.*;
public class BasicGUIApplet extends Applet {
  public void init( ) {
    add( new GUI( ) );
  }
}
```

HTML Stuff

- Need to reserve browser space by using an <APPLET> tag.
- To run this applet, make an HTML file with name BasicGUIApplet.html and contents:

31

Applet Initialization Methods

- The constructor
 - called once when the applet is constructed by the browser's VM. You do not have any browser context at this point, so you cannot get the size of the applet, or any parameters sent in the web code.
- init
 - called once. Magically, prior to this call, you have context in the browser. Put any initialization code that depends on having context here. Many people prefer to do all construction here.

Other "Applet Lifetime" Methods

• start

called by VM on your behalf when applet "comes into view." The precise meaning of this is browser dependent. Do not put any code that you would be unhappy having run more than once.

• stop

 called by VM on your behalf when applet "leaves view." The precise meaning of this is browser dependent. Called as many times as start.

destroy

 called by VM on your behalf when the applet is being permanently unloaded

33

Polite Applets

- Lifetime methods should manage Applet's background threads.
- Pattern #1: thread that lives as long as Applet
 - init creates thread
 - start should start/resume thread
 - stop should suspend thread
 - destroy should stop thread
- These all use deprecated thread methods.

Polite Pattern #2

- New thread each activation
 - start creates and starts a thread
 - stop stops and destroys a thread
- Use polling instead of deprecated method

35

Example of Polite Pattern #2

```
private Thread animator = null;
public void stop() {
    animator = null;
}
public void start() {
    if( animator == null ) {
        animator = new Thread( this );
        animator.start();
    }
}
public void run() {
    while( animator != null )
    ...
}
```

Applet Limitations

- An applet represents code that is downloaded from the network and run on your computer.
- To be useful, there must be guarantees that the downloaded applet isn't malicious (i.e. doesn't create viruses, alter files, or do anything tricky).

37

Basic Applet Restrictions

- Network-loaded applets, by default, run with serious security restrictions. Some of these are:
 - No files may be opened, even for reading
 - Applets can not run any local executable program.
 - Applets cannot communicate with any host other than the originating host.
- It is possible to grant privileges to applets that are trustworthy.
- Will talk about Java security in a few weeks.

Parameters

• Applets can be invoked with parameters by adding entries on the html page:

<APPLET CODE="program.class" WIDTH="150" HEIGHT="150">
<PARAM NAME="InitialColor" VALUE="Blue">
</APPLET>

• The applet can access the parameters with public String getParameter(String name)

39

Packaging The Applet

- Class file needs to be on the web server
 - same directory as web page assumed
 - can set CODEBASE in applet tag to change
- If other classes are needed, they either have to be on the web page too or available locally.
 - if not found locally, new connection made to get class (if not on web page, ClassNotFoundException)
 - repeated connections expensive
 - if lots of classes, should package in a jar file and use ARCHIVE tag.
 - jar files are compressed and download in one connection

Creating Jar Files

- Can store images, class files, entire directories.
 - Basically a zip file
 - Use command line tool (make sure jdk/bin is in the PATH environment variable)

```
jar cvf GUI.jar *.class
```

• On web page,

41

Why Swing?

- Better, more flexible GUIs.
- Faster. Uses "lightweight components"
- Automatic keyboard navigation
- Easy scrolling
- Easy tooltips
- Mnemonics
- General attempt to compete with WFC
- Can set custom look-and-feel

Swing Components

- Usually start with 'J':
- All components are lightweight (written in Java) except:
 - JApplet
 - JFrame
 - JWindow
 - JDialog
- AWT components have Swing analogs

43

AWT to Swing Mappings

- Almost all map by prepending a 'J'
- Examples:
 - Button -> JButton
 Panel -> JPanel
 List -> JList
- Exceptions:
 - Checkbox -> JCheckBox (note case change)
 - Choice -> JComboBox

Some New Components

• JTree

- Creates a tree, whose nodes can be expanded.
- Vast, complicated class

• JTable

- Used to display tables (for instance, those obtained from databases)
- Another enormous class

45

Big Difference Between Swing & AWT

- Components cannot be added to a heavyweight component directly. Use getContentPane() or setContentPane().
- Example:

```
JFrame f = new JFrame("Swing Frame");
JPanel p = new JPanel();
p.add( new JButton( "Quit" ) );
f.setContentPane( p );
--- OR (NOT PREFERRED)---
Container c = f.getContentPane();
c.add( new JButton( "Quit" ) );
```

Buttons, Checkboxes, etc

- Abstract class AbstractButton covers all buttons, checkboxes, radio groups, etc.
- Concrete classes include JButton, BasicArrowButton, JToggleButton, JCheckBox, JRadioButton.
- Can add images to buttons.

47

Tooltips

- Use setToolTipText to install a tooltip.
- Works for any JComponent.

```
Jbutton b = new Jbutton( "Quit" );
b.setToolTipText("Press to quit");
```

Borders

- Use setBorder to set borders for a JComponent.
- Available borders include
 - TitledBorder
 - EtchedBorder
 - LineBorder
 - MatteBorder
 - BevelBorder
 - SoftBevelBorder
 - CompoundBorder
- In package javax.swing.border

49

Popup Menus and Dialogs

- JPopupMenu
 - Appears when you right-click on a component
- JOptionPane
 - Contains static methods that pop up a modal dialog.
 Commonly used methods are:

```
showMessageDialog( )
showConfirmDialog( )
showInputDialog( )
```

Sliders

- Sliders are implemented with the JSlider class.
- Important methods:

```
JSlider( int orient, int low, int high, int val );
void setValue( int val );
int getValue( );
void setPaintTicks( boolean makeTicks );
void setMajorTickSpacing( int n );
void setMinorTickSpacing( int n );
```

 ChangeListener interface handles slider events. Must implement stateChanged method. Note: this interface is in package javax.swing.event.

51

Progress Bars

- JProgressBar displays data in relative fashion from empty (default value=0) to full (default value=100).
- Interesting methods

```
double getPercentComplete( );
int getValue( );
void setValue( int val );
```

Scrolling

- Use a JScrollPane to wrap any Component.
- Scrolling will be automatically done.
- Example:

```
JPanel p = new JPanel();
JList list = new JList();
for( int i = 0; i < 100; i++ )
    list.addItem( "" + i );
p.add( new JScrollPane( list ) );</pre>
```

53

Look-and-feel

- Used to give your GUIs a custom look. Currently there are three options:
 - Metal (platform independent)
 - Windows
 - Motif (X-windows, for Unix boxes)
- Use static method setLookandFeel in UIManager to set a custom look and feel.

```
static String motifClassName =
   "com.sun.java.swing.plaf.motif.MotifLookAndFeel";
try { UIManager.setLookAndFeel(motifClassName); }
catch( UnsupportedLookAndFeelException exc )
   { System.out.println( "Unsupported Look and Feel" ); }
```

Other Stuff (There's Lots)

- JFileChooser: supports file selection
- JPasswordField: hides input
- Swing 1.1.1 Beta 2 makes many components HTML-aware. Can do simple HTML formatting and display
- JTextPane: text editor that supports formatting, images, word wrap
- Springs and struts
- Automatic double-buffering
- General attempts to be competitive with WFC

55

Summary

- AWT is more portable; Swing is better looking
- Event handling uses delegation pattern
 - listeners register with components
 - event sent to all listeners
 - listeners implement an interface
 - listeners should finish quickly or spawn a thread
- Applets are just components
 - lifetime methods manage threads
 - context not set until init is called
 - applets run with security restrictions

Summary continued

- Most old AWT is easily translated:
 - Add J in front of the class names
 - Remember to have JFrame call setContentPane
- Swing has easy-to-use new stuff including tooltips, mnemonics, borders, JOptionPane.