



Core Java Course
Graphics Programming
Abstract Window Toolkit (AWT)
(Chapter 7)
Prof. Jeff Blessing
MSOE



Agenda

- ❖ Homework
- ❖ AWT classes
- ❖ Text and Fonts
- ❖ Color
- ❖ Simple Shapes
- ❖ Images
- ❖ Printing
- ❖ Code Examples

2



Abstract Window Toolkit (AWT)

- ❖ An abstraction to capture basic GUI functionality
- ❖ Supposed to be platform independent
- ❖ Should work the same way on any OS
 - Windows 95/98/NT, Solaris, Mac, OS2, SGI, Linux
- ❖ Common denominator approach: only concepts common across platforms are supported
- ❖ The reality:
 - Every platform has its own implementation deviations
 - Every platform may have different bugs/problems
 - Every platform has to be tested

3

AWT Extensions

- ❖ AWT itself provides only very primitive functionality
- ❖ Next step:
 - JavaSoft, Netscape, IBM - Java Foundation Classes (JFC) which includes Swing components
 - Microsoft - Application Foundation Classes (AFC)
- ❖ JFC and AFC provide a much better feature set for respective platforms than AWT
- ❖ AWT and Swing components will be covered

4

AWT: Frames and Windows

- ❖ AWT class hierarchy:
 - Object
 - Component
 - Container
 - Window
 - Frame
- ❖ **Window** - a place to display graphical output
- ❖ **Frame** - the topmost window not contained inside another window
- ❖ **Container** - a place for collection of components
- ❖ **Component** - ancestor for the most GUI classes

5

Swing: Frames and Windows

- ❖ AWT class hierarchy:
 - Object
 - Component
 - Container
 - JComponent
 - JPanel
 - Window
 - Frame
 - JFrame
- ❖ **Jwindow** - a place to display graphical output
- ❖ **JFrame** - the topmost window not contained inside another window

6

Component Class

- ❖ java.awt.Component
 - boolean isVisible();
 - boolean isShowing();
 - boolean isEnabled();
 - void setEnabled(boolean b);
 - Point getLocation();
 - void setLocation(Point p);
 - Dimension getSize();
 - void setSize(Dimension d);
 - void setSize(int width, int height);

7

Window and Frame Classes

- ❖ java.awt.Window
 - void toFront();
 - void toBack();
- ❖ java.awt.Frame
 - void setResizable(boolean b);
 - void setTitle(String s);

8

Graphics in a Window (Example)

- ❖ Inherit from basic AWT components
 - CloseableFrame is inherited from Frame
- ❖ Override the paint() method

```

public static void main(String[] args)
{
    Frame f = new ClosableFrame();
    f.show();
}
public void paint(Graphics g)
{
    // code that implements drawing
}

```

9

Graphics Object

- ❖ **Graphics** object contains a collection of settings for drawing images and text (the graphics *context*)
- ❖ Contains device context (Windows) or graphics context (X11) info for the drawing environment
- ❖ Any drawing (as well as printing output) in Java must use a `Graphics` object
- ❖ All measurements on the screen are done in pixels
- ❖ The (0,0) is the upper left corner of the screen
- ❖ Rendering text
`drawString(String s, int xCord, yCord) i0`

“Event-Driven” Paradigm

- ❖ Identify everything that could happen and associate every entry type with an event
- ❖ Define event handlers - what action to take if a particular event occurs
- ❖ Events themselves are beyond our control and depend on actors external to the system

11

How is `paint()` being called?

- ❖ Graphics programming in Java is event driven
- ❖ Events depend on what the user is doing
 - Maximize/minimize a window
 - Move a window
 - Open another window and cover the original one
- ❖ All these events require redrawing of the window
- ❖ The `update()` method is called by framework
 - The default implementation for `update()` (in class `Component`): erase the background and then call `paint()`

12

Swing Components

- ❖ Came along late in the JDK 1.1 lifecycle
- ❖ Contain many of the sophisticated GUI (Graphical User Interface) elements that programmers have come to expect
- ❖ Swing classes parallel (or extend) existing AWT classes:
 - Window (JWindow), Frame (JFrame), Button (JButton), Applet (JApplet), etc.

13

Creating a Closeable Frame

```
import javax.swing.*;

class FirstFrame extends JFrame
{
    public FirstFrame()
    {
        setTitle("FirstFrame");
        setSize(300, 200);
    }
}

public class FirstTest
{
    public static void main(String[] args)
    {
        JFrame frame = new FirstFrame();
        frame.show();
    }
}
```

Problem: Upon execution the close button only minimizes the window, it does not close it!

Solution: User's code must handle an event

14

Java's AWT Event Model

- ❖ User's code must register a *listener* for the *event* which signifies "window closing".
- ❖ Java provides an *interface* for this, named `WindowListener`
- ❖ `WindowListener` lists 7 methods to be provided by an implementor (lots of work!)
- ❖ Java provides a `WindowAdapter` class, which the User can subclass, for easier coding

15

WindowAdapter Use

```

import java.awt.event.*;
import javax.swing.*;

class CloseableFrame extends JFrame
{
    public CloseableFrame()
    {
        setTitle("CloseableFrame");
        setSize(300, 200);
        addWindowListener(new Terminator());
    }
}

class Terminator extends WindowAdapter
{
    public void windowClosing(WindowEvent e)
    {
        System.exit(0);
    }
}
    
```

16

The Rest of the Example:

```

public class CloseableTest
{
    public static void main(String[] args)
    {
        JFrame frame = new CloseableFrame();
        frame.show();
    }
}
    
```

Now, upon clicking "close", the application handles the event by terminating program execution

17

Inner Class Opportunity

❖ The Terminator class could be eliminated!

- replace it with an *anonymous inner class*:

```

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

18

Centering a Frame on the Screen

- ❖ Frames are a fixed size
- ❖ They appear large on a laptop and small on a high resolution monitor
- ❖ We'd like to make the frame a %'age of the screen size, but this requires OpSys help
- ❖ Toolkits provide system dependent info.
- ❖ *Icons* represent the window when it is minimized (*iconified*)

19

CenteredTest.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class CenteredFrame extends JFrame
{
    public CenteredFrame()
    {
        setTitle("CenteredFrame");
        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });
    }
};
```

20

```
Toolkit tk = Toolkit.getDefaultToolkit();
Dimension d = tk.getScreenSize();
int screenHeight = d.height;
int screenWidth = d.width;
setSize(screenWidth / 2, screenHeight / 2);
setLocation(screenWidth / 4, screenHeight / 4);
Image img = tk.getImage("icon.gif");
setIconImage(img);
}

public class CenteredTest
{
    public static void main(String[] args)
    {
        JFrame frame = new CenteredFrame();
        frame.show();
    }
}
```

21

Windows are a Pane!

- ❖ A JFrame contains many panes layered inside the frame
 - The *root*, *layered*, and *glass* pane are used with menu bars and “look & feel” operations
 - The *content* pane is used to draw on in Swing
 - To draw anything on the frame, first get the content pane and then add the object to it
 - Panels are good components to draw on and add to the content pane

22

Drawing on a JPanel

- ❖ Create a subclass of JPanel and override the `paintComponent()` method to draw on an instance of your panel
- ❖ Be sure to call the superclass's `paintComponent()` method to draw the background color before invoking your drawing code

23


NotHelloWorld.java

```

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class NotHelloWorldPanel extends JPanel
{
    public void paintComponent(Graphics g)
    {
        super.paintComponent(g);
        g.drawString("Not a Hello, World program",
                    75, 100);
    }
}
    
```

24




```

class NotHelloWorldFrame extends JFrame
{
    public NotHelloWorldFrame()
    {
        setTitle("NotHelloWorld");
        setSize(300, 200);
        addWindowListener(new WindowAdapter()
        {
            public void windowClosing(WindowEvent e)
            {
                System.exit(0);
            }
        });

        Container contentPane = getContentPane();
        contentPane.add(new NotHelloWorldPanel());
    }
}
    
```

25




Driver (main) for the Example:

```

public class NotHelloWorld
{
    public static void main(String[] args)
    {
        JFrame frame = new NotHelloWorldFrame();
        frame.show();
    }
}
    
```

26



Text and Fonts

- ❖ `drawString(String t, int cx, int cy);`
- ❖ Fonts are used to display text. A Font is an object that has name, style, and size attributes
 - `Font(String name, int style, int size);`
 - SansSerif, Serif, Monospaced, Dialog, Dialoginput
 - Example:

```

public void paint(Graphics g)
{
    Font f = new Font("Serif", Font.ITALIC, 12);
    g.setFont(f);
    g.drawString("This is Serif", 50, 50);
}
    
```

27

Positioning Text

- ❖ Class `FontMetrics` contains global information about the font size properties
- ❖ `stringWidth(String s)` - returns width of a string in pixels
- ❖ Dimension `getSize()` - returns size of a component
- ❖ **Insets** - dimension of the container borders
- ❖ Usable window width calculation:

```
int width = getSize().width - getInsets().left -
getInsets().right;
```

28

Color

- ❖ Color is an object in Java
- ❖ `Color(byte r, byte g, byte b);`
- ❖ Class `Graphics` has `setColor(Color c)` method to set or change the color
- ❖ Standard colors (13):
- black, green, red, blue, lightGray, white, cyan, magenta, yellow, darkGray, orange, gray, pink

29

More Colors...

- ❖ `SystemColor` class encapsulates colors used for various elements of the user interface
- ❖ Examples of system colors:
- desktop, window, windowBorder, windowText, text, control, scrollbar
- ❖ `void setColor(Color c);` // Graphics
// Component
- ❖ `void setBackgroundColor(Color c);`
- ❖ `void setForegroundColor(Color c);`

30

Drawing Shapes

- ❖ Class `Graphics` has the following methods to draw shapes (all of them are `void`):
 - `drawLine(int x1, int y1, int x2, int y2);`
 - `drawArc(int x, int y, int w, int h, int startAngle, int arcAngle);`
 - `drawPolyLine(int[] xPts, int[] yPts, int n);`
 - `drawPolygon(Polygon p);`
 - `drawRectangle(int x, int y, int w, int h);`
 - `drawRoundRectangle(..., int arcW, int arcH);`
 - `draw3DRect(..., boolean raised);`
 - `drawOval(int x, int y, int w, int h);`
 - `fillRect(...); // same set as draw`

31

Paint Modes

- ❖ **Override** paint mode - the mode when every new drawing on the screen irreversibly covers/overrides the previous ones
- ❖ **XOR** paint mode - converting existing colors to complimentary colors. Widely used for highlighting. Toggles back if applied once more.
- ❖ Class `Graphics` has `setColor(Color c)` and `setXORMode(Color c)` methods in order to switch between the two modes

32

Displaying Images

- ❖ **Image** - an array of graphics data pixel-by-pixel stored in a graphics file or in the memory
- ❖ Toolkit class has `Image getImage(String filename)` method that allows to open an image file
- ❖ `Graphics` class has two `drawImage` methods:
 - `boolean drawImage(Image img, int x, int y, ImageObserver observer); //non-scalable`
 - `boolean drawImage(Image img, int x, int y, int w, int h, ImageObserver observer); // scalable`

33

Printing (Java 1.1)

- ❖ Code that is written to display an image can be reused for printing the same image
- ❖ An instance of a subclass of class `Graphics` supporting `PrintGraphics` interface is used as printing context variable instead of `Graphics g`
- ❖ To determine whether we are printing or drawing we can use the the following test

```
if (g instanceof PrintGraphics)
// printing
else // drawing
```

34

Printing Steps

1. Get `PrintJob` object from default `Toolkit`
2. From the `PrintJob` object get `Graphics` object `g` that implements `PrintGraphics` interface
3. Use graphics context `g` to render the page
4. Call `dispose` method on graphics context `g` to eject the page
5. Repeat steps 2 - 4 until done with all pages
6. Call `end()` on the `PrintJob` object to terminate printing

35

Homework Assignment

- ❖ Enhance the `ShapeBoxTest` application for drawing real shapes
- ❖ Use `drawPolygon`, `drawRectangle`, `drawOval` methods of class `Graphics` to draw triangle, rectangle and circle
- ❖ Implement real `Scalable` and `Movable` interfaces

36

