Graphics And Events

Chapters 7 and 8

Graphics and Graphical Interfaces

Java™ graphics API

- **Java™ 1.0 AWT** (abstract windows toolkit)
  - AWT graphics are based on native peers
  - Sun trademarked the slogan “write once, run everywhere”
  - Platform look and feel
  - Lowest common denominator capability (equally mediocre everywhere)
  - Subtle differences between platforms lead to inconsistent results
  - Different bugs in the AWT on different platforms lead to the slogan “write once, debug everywhere”

- **Java™ 1.1 AWT**
  - Still peer based (i.e., still lowest common denominator)
  - More consistent naming and features
  - Completely redesigned event model
  - More object-oriented (modular) design
Graphics and Graphical Interfaces

Continued

- Java™ 1.2 & 1.3: AWT (peer based) and Swing (non-peer based)
- In 1996 Netscape creates the IFC (Internet Foundation Classes)
  < GUI components are painted on a blank screen
  < The only peer functionality used is creating and painting a window
  < GUI components look and feel the same on all platforms
  < Sun & Netscape perfected the project code named Swing
- Swing is the official name for the non-peer-based GUI portion of the JFC (Java™ Foundation Classes)
  < Swing uses the Java™ 1.1 event delegation model
  < Swing GUI components appear more slowly
  < Platform independent
    – Less sensitive to platform-specific bugs
    – Provides components not available on all platforms
    – Consistent look and feel across all platforms

Programming Graphics with awt

Abstract Windows Toolkit

- import java.awt.*;
- User-class extends Frame
  < Independent (top-level) window
    – System pull-down menu on left
    – Minimize, maximize, and close buttons on right
  < Create object from user’s class
  < Call setSize() (the default size is 0 × 0)
  < Call setTitle() (optional – can also set title in Frame constructor)
  < Call show() (or setVisible(true) )
    – show() makes a frame visible and moves it to the front
    – setVisible() can make a frame visible or invisible but does not move it to the front
    – Component.show() is deprecated but Window.show() is not deprecated
First Graphical Java™ Program
The quintessential “Hello World” (see awt/FrameDemo.java)

```java
import java.awt.*; // Graphics capability
class FrameDemo extends Frame {
    public static void main(String[] args) {
        Frame f = new FrameDemo(); // create Frame
        f.setTitle("Frame Demo"); // title at top of window
        f.setSize(600, 400); // default size is 0 x 0
        f.show(); // display window
    }

    public void paint(Graphics g) // awt drawing methods
    {
        g.drawString("Hello Window World", 20, 40);
    }
}
} // class FrameDemo
```

First Graphical Java™ Program
Alternate organization

```java
import java.awt.*;

class FrameDemo extends Frame {
    public static void main(String[] args) {
        new FrameDemo();
    }

    public FrameDemo() {
        setTitle("Frame Demo");
        setSize(600, 400);
        show();
    }

    public void paint(Graphics g) {
        g.drawString("Hello Window World", 20, 40);
    }
}
} // class FrameDemo
```
Frame Format

Java™ frame coordinate system

- Measured in pixels
- Everything in a Frame is drawn on a Graphics context

Closing The Window

Dealing with “zombies”

- This window will not close!!
- Windows
  - Give the focus to the DOS window and press control-C
  - “Three finger salute” (control-alt-delete)
    - Select the task from the task manager
    - Press “End Task”
- Unix
  - Give the focus to the terminal window that started the Java™ program press control-C (or other key if the intr key was reset)
  - Kill the process
    - ps to get the PID
    - kill -9 pid
Programming Graphics With Swing

Part of the JFC

- Steps to programming with Swing
  <import java.awt.*;
  <import javax.swing.*;
  <extend JFrame

- javax indicates an extension package
  <Browsers' security managers disallow loading “java” packages
  <Started as com.sun.java.swing
  <Moved to java.awt.swing (early 1.2 beta)
  <Back to com.sun.java.swing (late 1.2 beta)
  <Ended as javax.swing
  <It is an extension in 1.1 but is part of the 1.2 core

- NOTE: pressing the X button only hides the frame but does not stop the program (see slide 21)

Swing Example

"Hello World" in Swing (see swing/FrameDemo.java)

```java
import java.awt.*; // Graphics capability
import javax.swing.*; // Swing is in an extension

class FrameDemo extends JFrame
{
    public static void main(String[] args)
    {
        JFrame f = new FrameDemo(); // create Frame
        f.setTitle("Frame Demo"); // title at top
        f.setSize(600, 400); // default is 0 x 0
        f.show(); // display window

        public void paint(Graphics g) // not correct for Swing
        {
            g.drawString("Hello Window World", 20, 40);
        }
    } // class FrameDemo
```
**JFrame Structure**

A JFrame is a container

- JFrame is created with peer API
- menu bar is optional

**Adding Components to a JFrame**

Rendering text (see swing/HelloWorld.java)

- To draw (including rendering text)
  - Define a new class that extends JPanel
  - Override the `paintComponent` method in the new class
  - Instantiate an object from the new class and add to the content pane of a JFrame object

```java
class HelloPanel extends JPanel {
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        g.drawString("Hello World", 300, 200);
    }
} // HelloPanel
```
Adding Components to a JFrame

Continued

```java
public class HelloWorld extends JFrame {
    public static void main(String[] args) {
        JFrame F = new HelloWorld();
        F.setTitle("HelloWorld");
        F.setSize(600, 400);
        F.show();
    }

    public HelloWorld() {
        Container contentPain = getContentPane();
        contentPain.add(new HelloPanel());
    } // HelloWorld
} // class HelloWorld
```

The Event Delegation Model

Connecting event sources to event listeners

- **Introduced at Java™ 1.1**
  - Continues with Java™ 1.2 and newer
  - Also used in Swing
- **Event actions are delegated to listeners**
  - Event listeners are objects interested in particular events
  - Event sources are objects that may generate those events
  - Listeners must register with the sources that may generate events in which the listener is interested
  - When events occur, source objects send notification to all registered listener objects (order is not specified)

```java
eventSourceObject.addEventListener(eventListenerObject);
```
- eventSourceObject and eventListenerObject may be the same
- source may have many listeners
- listener may listen to many sources
Java™ Event Handlers

Event listener interfaces

- AWT provides 11 listener interfaces (for different kinds of events)
  - ActionListener
  - AdjustmentListener
  - ComponentListener
  - ContainerListener
  - ItemListener
  - KeyListener
  - MouseListener
  - FocusListener
  - ItemListener
  - TextListener
  - WindowListener

- The user must implement one or more interface
- The user must provide all methods specified in each implemented interface
- There are three, essentially equivalent, techniques for handling events

Chapter 7 Slide 15 of 30

First Technique

Verbose but straightforward (see event1.java)

- import java.awt.event.*;
- User class must implement WindowListener interface
- Register listener with source addWindowListener(this);
- User class must override seven WindowListener methods
  - public void windowClosing(WindowEvent e) { System.exit(0); } *
  - public void windowClosed(WindowEvent e) {} 
  - public void windowIconified(WindowEvent e) {} 
  - public void windowOpened(WindowEvent e) {} 
  - public void windowDeiconified(WindowEvent e) {} 
  - public void windowActivated(WindowEvent e) {} 
  - public void windowDeactivated(WindowEvent e) {} 

- * If windowClosing does not exit the program, the default action takes place following the event handler (see Slide 21)
Java™ Graphics Program Example

First technique (from event1.java)

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

public class event1 extends Frame implements WindowListener {
    // main and paint are unchanged FrameDemo.java

    public event1() // constructor
    {
        addWindowListener(this); // register listener
    }

    public void windowClosing(WindowEvent e) { System.exit(0); }
    public void windowClosed(WindowEvent e) { }
    public void windowIconified(WindowEvent e) { }
    public void windowOpened(WindowEvent e) { }
    public void windowDeiconified(WindowEvent e) { }
    public void windowActivated(WindowEvent e) { }
    public void windowDeactivated(WindowEvent e) { }
} // class event1
```

Chapter 7 Slide 17 of 30

Second Technique

Compact but requires a second class (see event2.java)

- Must import java.awt.event.*;
- Each listener interface with more than 1 method has an adapter class
  `<ComponentAdapter` `<KeyAdapter` `<MouseMotionAdapter`
  `<ContainerAdapter` `<MouseAdapter` `<WindowAdapter`
  `<FocusAdapter`
- Adapter classes implement the appropriate interface and have “dummy” methods that satisfy the interface
- Create a class that extends the appropriate adapter class and overrides the needed methods
- Instantiate and register an object from this class
Adapter Class Example

Excerpts from event2.java

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

public class event2 extends Frame
{
    public event2() {
        WindowCloser wc = new WindowCloser();
        addWindowListener(wc);
    }
}

Third Technique

Based on inner classes-- slick but obtuse (see event3.java)

- The listener is instantiated as an anonymous object

    public class event3 extends Frame
    {
        public event3() {
            addWindowListener( new WindowAdapter()
            {
                public void windowClosing(WindowEvent e)
                {
                    System.exit(0);
                }
            });
        }
    }
Swing and the “X” Button

Introduced at Java 1.3

- JFrame method: setDefaultCloseOperation(int closeAction)
  - WindowConstants.DO NOTHING_ON_CLOSE
  - WindowConstants.HIDE_ON_CLOSE
    - the default action for a JFrame
    - hide the frame after invoking any registered WindowListener objects
  - WindowConstants.DISPOSE_ON_CLOSE
  - JFrame.EXIT_ON_CLOSE
    - close with System.exit()
    - use only in applications (i.e., not in applets)

- Overloaded methods in
  - JDialog
  - JInternalFrame

Graphical and Event Methods

A summary

- void paintComponent(Graphics g)
  - Override in Swing (JFrame and JApplet)
- void paint(Graphics g)
  - Override in AWT (Frame and Applet)
- void repaint()
  - causes a call to component's update method as soon as possible
- void validate()
  - Ensures that component has a valid layout
- void setEnabled(boolean b)
  - Enables or disables this component, depending on the value of b
  - Disabled components are “grayed out” and do not generate events
Screen Measurements

Dynamically determine size (see Graph.java)

- **Methods**
  - `int getHeight()`
  - `int getWidth()`
  - `Dimension getSize()`
    - width
    - height
  - `Insets getInsets()`
    - left
    - right
    - top
    - bottom
  - `Insets i = getInsets();`
  - `Dimension screenSize = getSize();`
  - `usableHeight = screenSize.height - i.top - i.bottom;`
  - `usableWidth = screenSize.width - i.left - i.right;`

---

Screen Size

Getting the current screen size in pixels

- **Toolkit class contains information about the local environment**

  public static void main(String args[ ])
  {
    JFrame f = new ScreenSize(); // main is in a class named ScreenSize
    f.setTitle("Full Size Frame");

    Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
    f.setSize(screenSize.width, screenSize.height);

    f.show();
  }
Screen Formatting Methods

java.awt.Component (see Screen.java)

- Locations are measured relative to the upper left hand corner
  
  - Point getLocation() // parent’s coordinates
  - Point getLocationOnScreen() // screen coordinates
  - Rectangle getBounds() // parent’s coordinates
  - void setLocation(int x, int y) // parent’s coordinates
  - void setLocation(Point p) // parent’s coordinates
  - Dimension getSize()

- int getHeight() / int getWidth() // preferred
  
  - int getX() / int getY() // preferred

Lines and Shapes

Graphics methods (call from paint or paintComponent: g.drawLine ...)

- void drawLine(int x0, int y0, int x1, int y1)
- void drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
- void drawPolygon(Polygon p)
- void drawPolygon(int xPoints[], int yPoints[], int numPoints)
- void drawRect(int x, int y, int width, int height)
- void drawRoundRect(int x, int y, int width, int height, int arcWidth, int arcHeight)
- void draw3DRect(int x, int y, int width, int height, boolean raised)
- void drawOval(int x, int y, int width, int height)
More About Shapes

The meaning of parameters

\[(x,y)\]

### drawRect\((x,y,\text{width},\text{height})\)

\[(x,y)\]

### drawOval\((x,y,\text{width},\text{height})\)

\[(x,y)\]

### drawRoundRect\((x,y,\text{width},\text{height},\text{arcWidth},\text{arcHeight})\)

\[(x,y)\]

### drawArc\((x,y,\text{width},\text{height},\text{startAngle},\text{arcAngle})\)

\[(x,y)\]

---

Color

Foreground, background, lines, and fill (see colorDemo.java)

- **Color class**
  - `Color\((\text{int red, int green, int blue})\)` // all values are [0-255]
  - `Color\((\text{int RGB})\)` // RGB: 0xrrrgbb
    - RGB is in hexadecimal: blue bits 0-7; green bits 8-15; red bits 16-23
  - **Predefined colors**
    - `Color\.(\text{black} \ (0,0,0))` `Color\.(\text{green} \ (0,255,0))` `Color\.(\text{red} \ (255,0,0))`
    - `Color\.(\text{blue} \ (0,0,255))` `Color\.(\text{lightGray})` `Color\.(\text{white} \ (255,255,255))`
    - `Color\.(\text{cyan})` `Color\.(\text{magenta})` `Color\.(\text{yellow})`
    - `Color\.(\text{darkGray})` `Color\.(\text{orange})`
    - `Color\.(\text{gray})` `Color\.(\text{pink})`

- **Graphics and Frame methods**
  - `\text{void } g\.(\text{setColor}(\text{Color c}))` // call in paint or paintComponent
  - `\text{void } \text{setForeground}(\text{Color c})` // call in any method
  - `\text{void } \text{setBackground}(\text{Color c})` // call in any method
  - Effects all subsequent drawing operations
Images

GIFs and JPEGs

- Image image = Toolkit.getDefaultToolkit().getImage("file");
  <Images may be loaded from a JAR file (p. 629)
  <URL url = Token.class.getResource("Flag.gif");
  <Image flag = Toolkit.getDefaultToolkit().getImage(url);

- Graphics method: g.drawImage(image, x, y, observer);
  <x,y: coordinates of the upper left hand corner
  <observer: the object on which the image is drawn

- MediaTracker tracker = new MediaTracker(observer);
  <Makes the main program wait until all images are loaded
  <tracker.addImage(image, 0);
  try
  { tracker.waitForID(0); // wait here for images in group 0 to load
    // tracker.waitForAll(); // wait here for all images to load
  } catch (InterruptedException e) { } // ignore exceptions

Text and Fonts

Displaying or rendering text in graphical windows

- Graphics class
  <g.drawString(String s, int x, int y);
  <g.setFont(Font font);
  <FontMetrics f = g.getFontMetrics();

- Font class
  <Font(String name, int style, int size)
    – Helvetica, TimesRoman, Courier, Dialog, and Symbol
    – Font.PLAIN, Font.BOLD, Font.ITALIC, Font.BOLD | Font.ITALIC
  <String getName( ) / String getFontName( ) / String getFamily( )
  <boolean isBold( ) / boolean isItalic( ) / boolean isPlain( )
  <int getSize( )
  <int stringWidth(String s)