Overview

- Introduction to Java Windows & Graphical Output
- Introduction to event-driven programs & user interaction
- Specifics:
  - a little bit of history – AWT & Swing
  - some basic Swing components
  - Java graphics
- Reading:
  - Textbook: Appendix C
  - Eckel Text: Chapter 14
  - online: Sun Java Swing tutorial; Swing API javadoc web pages
  http://java.sun.com/docs/books/tutorial/uiswing/

Graphical User Interfaces

- GUIs are the hallmark of modern software
- User sees and interacts with "controls" or "components"
  - menus
  - scrollbars
  - text boxes
  - check boxes
  - buttons
  - radio button groups
  - graphics panels
  - etc.

Opposing Styles of Interactions

- "Algorithm Driven"
  - When program needs information from the user, it asks for it.
  - Program is in control.
  - Typically in a non-GUI environment
- "Event Driven"
  - When user wants to do something, he/she signals the program
  - These signals come to the program as events.
  - The program is interrupted by these events.
  - User has more control.
  - Typical in a GUI environment.

A Bit of Java History

- Java 1.0: AWT (Abstract Windowing Toolkit)
- Java 1.1: AWT with new Event Handling Model
- Java 1.2 (called Java 2): Swing
  - Greatly enhanced user interface toolkit built on top of AWT.
  - Same basic event handling model as in Java 1.1 AWT.
- Java 1.3, 1.4, 1.5:
  - Bug fixes and significant performance improvements; no major revolution.
- Naming conventions:
  - Most Swing components start with J.
  - No such standard for AWT components.
- We’ll focus on using the Swing classes.
  - You’ll still need to import the awt libraries since swing builds upon the awt infrastructure.

Components and Containers

- Every GUI-related component descends from Component, which contains dozens of basic methods and fields common to all AWT/Swing components.
- "Atomic" components: labels, text, fields, buttons, check boxes, icons, menu items, ...
- Some components are Containers
  - these components can contain other sub-components
Types of Containers

- Top level containers: JFrame, JDialog, JApplet, and others
  - Every program with a Swing GUI must have at least one top-level container.
  - The standard top-level container for an application user interface is JFrame.
  - A JFrame is a window with a title and a border. It can be moved, resized, etc.
  - Containers can hold other components – things like buttons, scroll bars, text areas, etc.
- Mid-Level containers: panels, scroll panes, tool bars, …
  - Can contain certain other components
  - JPanel is best for general use
- Specialized containers: menus, list boxes, combo boxes, …
- Technically, all JComponents are containers

JFrame – a top level component

- Top level application window
  - JFrame win = new JFrame("Optional Window Title");
- Some useful methods:
  - setSize(int width, int height); // frame width & height
  - setBackground(Color c); // background color
  - setVisible( boolean b) // show or hide
  - repaint( ); // Request repaint after content change
  - setPreferredSize(Dimension d) // default size for window; also
    // can set min and max sizes
  - dispose( ); // get rid of the window when done
  - pack( ); // size the window to fit all its sub-components
  - setDefaultCloseOperation(int );  // define what should be done
    // when window is closed

The Simplest of Programs

```java
import java.awt.*;
import javax.swing.*;
public class SimpleSwingExample{
public static void main(String[] args){
    JFrame win = new JFrame("Simple Swing Example");
    win.setSize( new Dimension(350, 200) );
    // Default close operation of a JFrame is to dispose the window, but not to exit the
    // program. We'd like to exit on the close operation.
    win.setDefaultCloseOperation( JFrame.EXIT_ON_CLOSE );
    // Creating a window and making it visible are two separate operations.
    // If we want to make this window visible, we need to call the
    // setVisible method.
    win.setVisible( true );
}
}
```

JPanel – A General Purpose Container

- Commonly added to a window to provide a space for graphics, or collections of buttons, labels, etc.
- JPanels can be nested to any depth (i.e. a JPanel may contain a JPanel)
- Many methods common with JFrame (since both are ultimately instances of Component)
- A bit of advice: if you can't find the method you're looking for, look in the superclass.

Adding Components to Containers

- Swing containers have a "content pane" that manages the components in that container (this differs from the original AWT, which managed components directly)
- To add a component to a container, get the content pane, and use its add method:
  ```java
  JFrame window = new JFrame();
  JPanel panel = new JPanel();
  window.getContentPane().add(panel);
  or
  Container cp = window.getContentPane();
  cp.add(panel);
  ```

New to Java 1.5

- Simply call add on a top-level container to add components to the content pane.
  - In earlier versions of Java, you had to get the content pane.
  - You might still need to access the content pane directly (for example, to set the layout differently)
- See java tutorial for more information on using top level containers.
  - [http://java.sun.com/docs/books/tutorial/uiswing/components/toplevel.html](http://java.sun.com/docs/books/tutorial/uiswing/components/toplevel.html)
Non-Component Classes

- Not all classes are GUI components
- AWT
  - Color, Dimension, Font, layout managers
  - Shape and subclasses like Rectangle, Point, etc.
  - Graphics
- Swing
  - Borders
  - Geometric classes
  - Graphics 2D
- Neither AWT nor Swing
  - Images, Icons

Layout Managers

- What happens if we add several components to a container?
  - What are their relative positions?
- Answer: each component has a layout manager.
  - Several different kinds.
    - FlowLayout – left to right, top to bottom
    - BorderLayout – CENTER, NORTH, SOUTH, EAST, WEST
    - GridLayout – 2-D grid. Size of grid is specified.
    - GridBagLayout – Allows great flexibility. It is also very complicated.
- A container’s state is “valid” or “invalid” depending on whether the layout manager has arranged the components since the last change.
- Default LayoutManager for JFrame is BorderLayout.
- Default LayoutManager for JPanel is FlowLayout.

LayoutManager Test

```java
public class LayoutManagerTest extends JFrame {
    public LayoutManagerTest(LayoutManager layout, String windowTitle) {
        super(windowTitle);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        Container cp = getContentPane();
        cp.setLayout(layout);
    }
}
```

LayoutManagerTestDriver

```java
public class LayoutManagerTestDriver {
    public static void main(String[] args) {
        LayoutManager manager = null;
        if (args.length == 0) {
            manager = new FlowLayout();
        } else {
            manager = new GridLayout(8, 6);
        }
        LayoutManagerTest win = new LayoutManagerTest(manager, "Layout Test");
        win.setPreferredSize(new Dimension(500, 400));
        for (int i = 1; i <= 41; i++) {
            win.add(new Button("" + i));
        }
        win.pack();
        win.setVisible(true);
    }
}
```

pack and validate

- When a container is altered, either by adding components or changes to components (resized, contents change, etc.), the layout needs to be updated.
  - Swing does this automatically more often than AWT does, but not always.
- Common methods after changing a layout
  - validate() – redo the layout to take into account new or changed components
  - pack() – redo the layout using the preferred size of each component

Graphics and Drawing

- The windows, panes, and other components supplied with Swing are sufficient for predefined GUI components.
- For more complex graphics, extend a suitable class and override the empty (inherited) method `paintComponent` that draws its contents.
Method paintComponent is called by the underlying system whenever it needs the window to be repainted.
- Triggered by the window being moved, resized, uncovered, etc.
- Can happen at anytime – you don't control it!!!
- If your code does something that requires repainting, call method repaint()
  Requests that paintComponent be called sometime in the future, when convenient for the underlying system window manager.

Painting Rules
- Always override paintComponent() of any component you will be drawing on.
  - Not necessary for things that don't require Graphics
- Never call paint() or paintComponent() directly.
- Always paint the entire picture from scratch.
- Always call the super.paintComponent(…) method before doing your own painting.
- Don't create a Graphics or Graphics 2D object to draw with
  - Only use the one given to you as a parameter of paintComponent()

Classes Graphics and Graphics 2D
- The parameter to paintComponent and paint is a graphics context where the drawing should be done
  - Class Graphics 2D is a subclass of Graphics with better features
  - In Swing components, the parameter has static type Graphics, but dynamic type Graphics2D
  - So, cast it to a 2D and use that
- A fairly procedural-like interface
  - Call graphics methods to draw on the graphics object

Drawing Graphical Objects
- Many of the methods in the Graphics class take an argument of type Shape. Shape is an interface.
- Lots of methods available to draw various kinds of outline and solid shapes and control colors and fonts
  - setColor, setFont, drawArc, drawLine, fillPolygon, drawOval, fillRect, and many others

Another Example
```java
public class RectangleGraphic extends JPanel {
    private Rectangle r;
    private Color color;

    public RectangleGraphic() {
        super();
        r = new Rectangle(120, 80);
        this.color = Color.green;
        this.setBackground(color);
    }

    public void enlarge() {
        r.width = 2 * r.width;
        r.height = 2 * r.height;
    }

    public Dimension getPreferredSize() {
        return new Dimension(r.width, r.height);
    }

    public Dimension getMinimumSize() {
        return new Dimension(r.width, r.height);
    }

    public void setColor(java.awt.Color c) {
        this.color = c;
    }

    public void paintComponent(Graphics g) {
        Graphics2D g2 = (Graphics2D)g;
        g2.setColor(color);
        g2.fill(r);
    }
}
```

Using The RectangleGraphic
```java
public class SwingExample {
    public static void main(String[] args) {
        JFrame window = new JFrame("Swing Example");
        Container contents = window.getContentPane();
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        contents.setLayout(new GridLayout(3, 1));
        JPanel panel = new JPanel();
        panel.setBackground(Color.red);
        contents.add(panel);
        JPanel panelTwo = new JPanel();
        panelTwo.setBackground(Color.black);
        contents.add(panelTwo);
        JButton uselessButton = new JButton("Useless Button");
        panelTwo.add(uselessButton);
        JTextArea textArea = new JTextArea("Add Text Here: ", 10, 60);
        textArea.setCaretPosition(textArea.getText().length());
        panel.add(textArea);
        textArea.setBackground(panel.getBackground());
        contents.add(new RectangleGraphic());
        window.pack();
        window.setVisible(true);
    }
}
```
Lots of Info Thus Far

- What to do
  - Start reading your text (Appendix C)
  - Browse the Swing tutorial and Java Swing/AWT documentation from Sun to start to feel your way around
  - Practice! Try writing a simple program that draws a window on the screen.