Interactive Programs

• “Classic” view of computer programs: transform inputs to outputs, stop

• Event-driven programs: interactive, long-running
  – Servers interact with clients
  – Applications interact with users

GUI Motivation

• Interacting with a Program
  – Program-Directed = Proactive
    • Statements execute in sequential, predetermined order
    • Typically use keyboard or file I/O, but program determines when that happens
    • Usually single-threaded
  – Event-Directed = Reactive
    • Program waits for user input to activate certain statements
    • Typically uses a GUI (Graphical User Interface)
    • Often multi-threaded

• Design...Which to pick?
  – Program called by another program?
  – Program used at command line?
  – Program interacts often with user?
  – Program used in window environment?

Java Support for Building GUIs

• Java Foundation Classes
  – Classes for building GUIs
  – Major components
    • awt and swing
    • Pluggable look-and-feel support
    • Accessibility API
    • Java 2D API
    • Drag-and-drop Support
    • Internationalization

• Our main focus: Swing
  – Building blocks of GUIs
  – Windows & components
  – User interactions

• Built upon the AWT (Abstract Window Toolkit)
  – Java event model

Java Foundation Classes

• Pluggable Look-and-Feel Support
  – Controls look-and-feel for particular windowing environment
  – E.g., Java, Windows, Mac

• Accessibility API
  – Supports assistive technologies such as screen readers and Braille

• Java 2D
  – Drawing
  – Includes rectangles, lines, circles, images, ...

• Drag-and-drop
  – Support for drag and drop between Java application and a native application

• Internationalization
  – Support for other languages

GUI Statics and GUI Dynamics

• Statics: what’s drawn on the screen
  – Components
    • E.g. buttons, labels, lists, sliders, menus, ...
  – Containers
    • components that contain other components
    • E.g. frames, panels, dialog boxes, ...
  – Layout managers
    • control placement and sizing of components

• Dynamics: user interactions
  – Events
    • E.g. button-press, mouse-click, key-press, ...
  – Listeners
    • an object that responds to an event
  – Helper classes
    • E.g. Graphics, Color, Font, FontMetrics, Dimension, ...
Creating a Window

```java
import javax.swing.*;

public class Basic1 {
    public static void main(String[] args) {
        //create the window
        JFrame f = new JFrame("Basic Test!");
        //set size in pixels
        f.setSize(200, 200);
        //show the window
    }
}
```

Creating a Window Using a Constructor

```java
import javax.swing.*;

public class Basic2 extends JFrame {
    public static void main(String[] args) {
        new Basic2();
    }
    public Basic2() {
        setTitle("Basic Test2!");
        //set size in pixels
        setSize(200, 200);
        //show the window
    }
}
```

A More Extensive Example

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

public class Intro extends JFrame {
    private int count = 0;
    private JButton myButton = new JButton("Push Me!");
    private JLabel label = new JLabel("Count: " + count);
    public Intro() {
        //set title
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        //set layout manager
        setLayout(new FlowLayout(FlowLayout.LEFT));
        //add components
        add(myButton);
        add(label);
        myButton.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                count++;
                label.setText("Count: " + count);
            }
        });
        pack();
        setVisible(true);
    }
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception exc) {} 
        new Intro();
    }
}
```

GUI Statics

- Determine which components you want
- Choose a top-level container in which to put the components (JFrame is often a good choice)
- Choose a layout manager to determine how components are arranged
- Place the components

Components = What You See

- Visual part of an interface
- Represents something with position and size
- Can be painted on screen and can receive events
- Buttons, labels, lists, sliders, menus, ...

Component Examples

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

public class ComponentExamples extends JFrame {
    public ComponentExamples() {
        //set layout manager
        setLayout(new FlowLayout(FlowLayout.LEFT));
        //add components
        add(new JComboBox(new String[] { "A", "B", "C" }));
        add(new JCheckBox("JCheckBox"));
        add(new JSlider(0, 100));
        add(new JColorChooser());
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        pack();
        setVisible(true);
    }
    public static void main(String[] args) {
        new ComponentExamples();
    }
}
```
More Components

• **JFileChooser**: allows choosing a file
• **JLabel**: a simple text label
• **JTextArea**: editable text
• **JTextField**: editable text (one line)
• **JScrollBar**: a scrollbar
• **JPopupMenu**: a pop-up menu
• **JProgressBar**: a progress bar
• Lots more!

Containers

• A container is a component that
  – Can hold other components
  – Has a layout manager
• Heavyweight vs. lightweight
  – A heavyweight component interacts directly with the host system
  – JWindow, JFrame, and JDialog are heavyweight
  – Except for these top-level containers, Swing components are almost all lightweight
  – JPanel is lightweight

Three basic top-level containers:

• **JWindow**: top-level window with no border
• **JFrame**: top-level window with border and (optional) menu bar
• **JDialog**: used for dialog windows
• Another important container
  – JPanel: used mostly to organize objects within other containers

A Component Tree

![Component Tree Diagram]

![FlowLayout Example]

Layout Managers

• A layout manager controls placement and sizing of components in a container
  – If you do not specify a layout manager, the container will use a default:
    – JFrame default = BorderLayout
  – Five common layout managers:
    – BorderLayout, BoxLayout, FlowLayout, GridBagLayout, GridLayout
  – General syntax
    – container.setLayout(new LayoutManager());
  – Examples:
    – JFrame p1 = new JPanel(new BorderLayout());
    – JFrame p2 = new JPanel();
    – p2.setLayout(new BorderLayout());

Some Example Layout Managers

• **FlowLayout**
  – Components placed from left to right in order added
  – When a row is filled, a new row is started
  – Lines can be centered, left-justified or right-justified (see FlowLayout constructor)
  – See also BoxLayout

• **GridLayout**
  – Components are placed in grid pattern
  – number of rows & columns specified in constructor
  – Grids filled left-to-right, then top-to-bottom

• **BorderLayout**
  – Divides window into five areas: North, South, East, West, Center
  – Adding components
    – FlowLayout and GridLayout use container.add(component)
    – BorderLayout uses container.add(component, index) where index is one of:
      – BorderLayout.NORTH
      – BorderLayout.SOUTH
      – BorderLayout.EAST
      – BorderLayout.WEST
      – BorderLayout.CENTER

FlowLayout Example

```java
import javax.swing.*;
import java.awt.*;
public class Statics1 {
    public static void main(String[] args) {
        new S1GUI();
    }
}
class S1GUI {
    private JFrame f;
    public S1GUI() {
        f = new JFrame("Statics1");
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(300, 200);
        f.setLayout(new FlowLayout());
        for (int i = 1; i < 9; i++)
            f.add(new JButton("Button "+ i));
        f.setVisible(true);
    }
}
```
More Layout Managers

- CardLayout
  - Tabbed index card look from Windows

- GridBagLayout
  - Most versatile, but complicated

- Custom
  - Can define your own layout manager
  - But best to try Java's layout managers first...

- null
  - No layout manager
  - Programmer must specify absolute locations
  - Provides great control, but can be dangerous because of platform dependency

AWT and Swing

- AWT
  - Initial GUI toolkit for Java
  - Provided a "Java" look and feel
  - Basic API: java.awt.*

- Swing
  - More recent (since Java 1.2) GUI toolkit
  - Added functionality (new components)
  - Supports look and feel for various platforms (Windows, Mac)
  - Basic API: java.swing.*

- Did Swing replaced AWT?
  - Not quite: both use the AWT event model

Code Examples

- Intro.java
  - Button & counter

- Basic1.java
  - Create a window

- Basic2.java
  - Create a window using a constructor

- Calculator.java
  - Shows use of JOptionPane to produce standard dialogs

- ComponentExamples.java
  - Sample components

- Statics1.java
  - FlowLayout example

- Statics2.java
  - BorderLayout example

- Statics3.java
  - GridLayout example

- LayoutDemo.java
  - Multiple layouts