Introduction to Graphical User Interfaces (GUIs)

Lecture 13
CS2110 – Fall 2011

Announcements

- A4 is up, due Sunday, Nov 6, 11:59pm
  - Use the time to do something fun
- Prelim 1 will be graded tonight
  - hopefully
- Prelim 2 scheduled for 11/15, 7:30-9pm
  - If you have a conflict, let us know now

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
  - OS responds to kernel calls

GUI Motivation

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

How does Java do GUIs?

Java Support for Building GUIs

- Java Foundation Classes
  - Classes for building GUIs
  - Major components
    - java.awt and javax.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
    - Buttons, labels, lists, sliders, menus, ...
  - Containers: components that contain other components
    - Frames, panels,对话 boxes, ...
  - Layout managers: control placement and sizing of components
- **Dynamics**: user interactions
  - Events
    - Button-press, mouse-click, key-press, ...
  - Listeners: an object that responds to an event
  - Helper classes
    - 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!");
        //quit Java after closing the window
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setSize(200, 200);
        //set size in pixels
        f.setVisible(true); //show the window
    }
}
```

Creating a Window Using a Constructor

```java
import javax.swing.*;
import java.awt.*;
public class Basic2 extends JFrame {
    public static void main(String[] args) {
        new Basic2();
    }
    public Basic2() {
        setTitle("Basic Test2!"); //set the title
        //quit Java after closing the window
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setSize(200, 200); //set size in pixels
        setVisible(true); //show the window
    }
}
```

A More Extensive Example

```java
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() {
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        setLayout(new FlowLayout(FlowLayout.LEFT));
        //set layout manager
        add(myButton); //add components
        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() {
        setLayout(new FlowLayout(FlowLayout.LEFT));
        add(new JButton("Button"));
        add(new JLabel("Label"));
        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);
}
```

More Components

- JFileChooser: allows choosing a file
- JLabel: a simple text label
- JTextArea: editable text
- JTextField: editable text (one line)
- JScrollPane: 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
  - JFrame, JWindow, and JDialog are heavyweight
  - Except for these top-level containers, Swing components are almost all lightweight
  - JPanel is lightweight

- There are 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

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:
    - JPanel <init> FlowLayout
    - JFrame <init> BorderLayout

- Five common layout managers:
  - BorderLayout, BoxLayout, FlowLayout, GridBagLayout, GridLayout

- General syntax
  `container.setLayout(new LayoutManager());`

- Examples:
  ```java
  JPanel p1 = new JPanel(new BorderLayout());
  p1.setLayout(new GridLayout());
  ```

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

- BorderLayout
  - Divides window into five areas: North, South, East, West, Center

- GridBagLayout
  - Components placed in grid pattern
  - number of rows & columns specified in constructor
  - Grid is filled left-to-right, then top-to-bottom
FlowLayout Example

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

public class Static1 {
    public static void main(String[] args) {
        new S1GUI();
    }
}

class S1GUI {
    private JFrame f;
}

public S1GUI() {
    f = new JFrame("Static1");
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setSize(500, 200);
    f.setLocation(new FlowLayout(FlowLayout.LEFT));
    for (int b = 1; b < 9; b++)
        f.add(new JButton("Button " + b));
    f.setVisible(true);
}
```

BorderLayout Example

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

public class Static1 {
    public static void main(String[] args) {
        new S1GUI();
    }
}

class S1GUI extends JFrame {
    public S1GUI() {
        super("Static1");
        add(new Component1());
        add(new Component2());
    }
}
```

GridLayout Example

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

public class Static1 {
    public static void main(String[] args) {
        new S1GUI();
    }
}

class S1GUI extends JFrame {
    public S1GUI() {
        add(new Component1());
        add(new Component2());
    }
}
```

More Layout Managers

- **CardLayout**
  - Take layout card look from Windows

- **GridLayout**
  - Most versatile, but complicated

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

- **GridBagLayout**
  - Most versatile, but complicated

- **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 replace 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

- **Static1.java**

- **FlowLayout example**

- **BorderLayout example**

- **Static3.java**

- **GridLayout example**

- **LayoutDemo.java**

- **Multiple layouts**