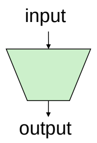
CS/ENGRD 2110 Object-Oriented Programming and Data Structures

Spring 2011
Thorsten Joachims

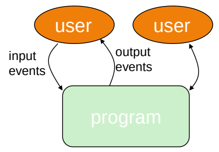
Lecture 14: Graphical User Interfaces (Static)

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 user(s)



2

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

3

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

4

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

5

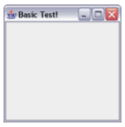
GUI Statics and GUI Dynamics

- 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, ...

6

Creating a Window

```
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
   }
}
```

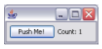


Creating a Window Using a Constructor

```
import javax.swing.*;

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(EXIT_ON_CLOSE);
      setSize(200, 200); //set size in pixels
      setVisible(true);  //show the window
   }
}
```

A More Extensive Example

```
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() {
      setDefaultCloseOperation(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

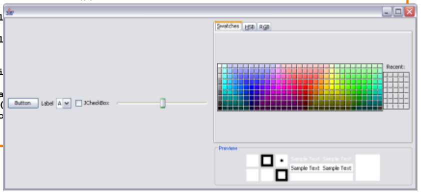
- Determine which *components* you want
- Choose a top-level *container* in which to put the components (**`JFrame`** is often a good choice)
- Choose a *layout manager* to determine how components are arranged
- Place the components

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

```
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());

      setDefaul
      pack();
      setVisibl
   }

   public stati
      try {
         UIMana
      } catch (
      new Compo
   }
}
```



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!

13

Containers

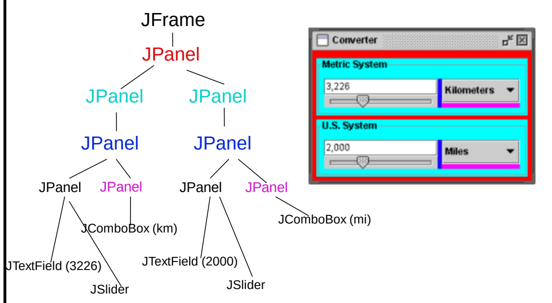
- A container is a component that
  - Can hold other components
  - Has a layout manager
- Heavyweight vs. lightweight
  - A heavyweight component interacts directly with the host system
  - JWindow, JFrame, and JDialog are heavyweight
  - Except for these top-level containers, Swing components are almost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

14

A Component Tree



15

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 default = FlowLayout
    - JFrame default = BorderLayout
- Five common layout managers:
  - BorderLayout, BoxLayout, FlowLayout, GridBagLayout, GridLayout
- General syntax
  - `container.setLayout(new LayoutMan());`
- Examples:
  - `JPanel p1 = new JPanel(new BorderLayout());`
  - `JPanel p2 = new JPanel();`
  - `p2.setLayout(new BorderLayout());`

16

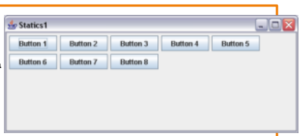
Some Example Layout Managers

- 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
  - Grid is 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

17

`FlowLayout` Example

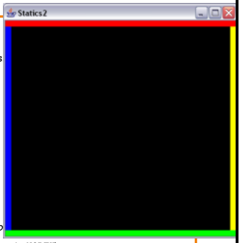
```
import javax.swing.*;
import java.awt.*;

public class Statics1 {
   public static void main
      new S1GUI();
   }
}

class S1GUI {
   private JFrame f;

   public S1GUI() {
      f = new JFrame("Statics1");
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      f.setSize(500, 200);
      f.setLayout(new FlowLayout(FlowLayout.LEFT));
      for (int b = 1; b < 9; b++)
         f.add(new JButton("Button " + b));
      f.setVisible(true);
   }
}
```



18

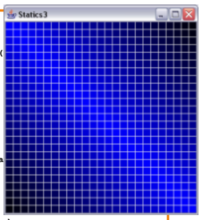
`BorderLayout` Example

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

public class Statics2 {
   public static void main(String[] args) { new S
}

class ColoredJPanel extends JPanel {
   Color color;

   ColoredJPanel(Color color) {
      this.color = color;
   }

   public void paintComponent(Graphics g) {
      g.setColor(color);
      g.fillRect(0, 0, 400, 400);
   }
}

class S2GUI extends JFrame {
   public S2GUI() {
      setTitle("Statics2");
      setDefaultCloseOperation(JFrame.EXIT_ON_CLO
      setSize(400, 400);
      add(new ColoredJPanel(Color.RED), BorderLayout.NORTH);
      add(new ColoredJPanel(Color.GREEN), BorderLayout.SOUTH);
      add(new ColoredJPanel(Color.BLUE), BorderLayout.WEST);
      add(new ColoredJPanel(Color.YELLOW), BorderLayout.EAST);
      add(new ColoredJPanel(Color.BLACK), BorderLayout.CENTER);
      setVisible(true);
   }
}
```



`GridLayout` Example

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

public class Statics3 {
   public static void main(String[] args) { new S3GUI(
}

class S3GUI extends JFrame {
   static final int DIM = 25;
   static final int SIZE = 12;
   static final int GAP = 1;

   public S3GUI() {
      setTitle("Statics3");
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      setLayout(new GridLayout(DIM, DIM, GAP, GAP));
      for (int i = 0; i < DIM * DIM; i++) add(new MyPa
      pack();
      setVisible(true);
   }

   class MyPanel extends JPanel {[
      MyPanel() {
          setPreferredSize(new Dimension(SIZE, SIZE)); }

      public void paintComponent(Graphics g) {
         float gradient =
            1f - ((float)Math.abs(getX() - getY()))/(float)((SIZE + GAP) * DIM);
         g.setColor(new Color(0f, 0f, gradient));
         g.fillRect(0, 0, getWidth(), getHeight());
      }
   }
}
```



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: javax.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