CS211
GUI Statics

Announcements

- No KP office hours today—9-10am on Fri just this week
- Final Exam info?
- Prelim 2
- Course grade cutoffs...

Overview

- Motivation
- JFC
- AWT and Swing
- Creating and Using A GUI
- Containers
- Layout Managers
- Where to go for more info?
  Java Tutorial! (where's that...?)

Motivation

- Program Driven:
  - statements execute in sequential, pre-determined order
  - typically use keyboard/file I/O from console
- Event Driven:
  - program waits for user input to activate certain statements
  - typically use graphical I/O
  - ***GUI***: graphical user interface
- Which to pick? Questions to ask:
  - program called by another program?
  - program used at command line?
  - program interacts often with user?
  - program used in window environment?
  - “old school” vs “new school”?
- Up next...How does Java do GUIs?

Java Foundation Classes

- ***JFC***: Java Foundation Classes
  - API classes for building GUIs
  - five major components:
    - Swing
    - Pluggable Look and Feel Support
    - Accessibility API
    - Java 2D API
    - Drag and Drop Support
- Our focus: ***Swing***
  - the visual components of the GUI
    - building windows
    - user interactions
  - built upon something called ***AWT*** (Abstract Window Toolkit)
- What are the four other things....?

More Aspects of JFC

- Pluggable Look and Feel Support
  - define look for particular windowing environment
  - ex) Windows, Motif
- Accessibility API
  - assistive technologies such as screen readers and Braille
  - displays for non-standard I/O
- Java 2D
  - draw images
  - rectangles, lines, circles, images, ...
- Drag and Drop
  - drag and drop between Java application and a native application
- Want more?
  - http://java.sun.com/docs/books/tutorial/uiswing/

Brief Example

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

public class Statics0 extends JFrame {

  private int count;
  private JButton b = new JButton("Push Me!");
  private JLabel label = new JLabel(generateLabel());
  private Container c = getContentPane();


  public static void main(String[] args) {

    Statics0 f = new Statics0();

    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.setSize(200,100);
    f.setVisible(true);

  }
```


Example (continued)

```
  public Statics0() {

    c.setLayout(new FlowLayout(FlowLayout.LEFT) );
    c.add(b);
    c.add(label);

    b.addActionListener( new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        count++;
        label.setText(generateLabel());
      }
    } );

  }

  private String generateLabel() {
    return "Count: "+Integer.toString(count);
  }

}
```

So...how did I do that...?

Creating a GUI

- We'll split the your GUI learning into 2 parts:
  - ***Statics***: what you draw on the screen
    - ***Components***: what you see on the screen
    - ***Containers***: special kind of components that contain other components
    - ***Layout managers***: objects that control placement and sizing of components
  - ***Dynamics***: how user interacts with elements on screen
    - ***Events***: an object that represents an occurrence
    - ***Listeners***: an object that listens for an event
    - ***Helper classes***: AWT classes `Graphics`, `Color`, `Font`, `FontMetrics`, `Dimension`
- Start with statics:
  - figure out which components you want
  - pick a ***top-level*** container in which to put the components
  - pick layout manager to arrange components
  - place components
- Swing or AWT?

AWT and Swing

- ***AWT***:
  - use code for windowing system from your computer
  - called ***heavyweight***
  - disadvantage: not being able to port to other OS
  - basic API package: **`java.awt.*`**
- ***Swing***:
  - Swing classes have no native code
  - more portable, added functionality
  - called ***lightweight*** because **mostly** written in Java
  - essentially supersedes many AWT components
  - basic API package: **`javax.swing.*`**
- Swing replaced AWT? not quite:
  - Swing uses AWT event model (see dynamics)
  - still need AWT for each OS
- So...where are all these classes?

Hierarchy for Statics Classes

- Java API!
- Basic statics hierarchy (AWT, Swing):

`Object`
Helper Classes
Layout Managers
`Component`
AWT components, like `Button`, `Canvas`, etc.
`Container` (in AWT)
`Panel`
`Applet`
`JApplet` (heavyweight)
`Window`
`Frame`
`JFrame` (heavyweight)
`Dialog`
`JDialog` (heavyweight)
`JWindow`
`JComponent` (lightweight)
many subclasses that start with `J`

Now, more detail on Components, Containers, Layout Managers...

Components

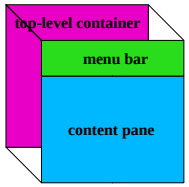
- ***Components***...what you ***paint***:
  - visual part of interface
  - represents something with position and size
  - can be painted on screen and receive events
  - buttons, labels, etc.
- see http://java.sun.com/docs/books/tutorial/uiswing/components/index.html
- Some examples in `ComponentExamples.java` (next slide)

Component Examples

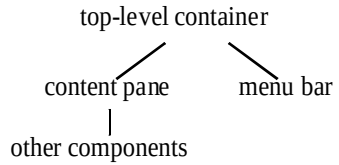
```
import javax.swing.*;
import java.awt.*;

public class ComponentExamples extends JFrame {
  public static void main(String[] args) {
    ComponentExamples f = new ComponentExamples();
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    f.pack();
    f.setVisible(true);
  }
  public ComponentExamples() {
    Container c = getContentPane();
    c.setLayout( new FlowLayout(FlowLayout.LEFT) );
    c.add(new JButton("Button"));
    c.add(new JLabel("Label"));
    c.add(new JComboBox(new String[] { "A", "B","C" } ) );
    c.add(new JCheckBox("JCheckBox"));
    c.add(new JSlider(0,100));
    c.add(new JColorChooser());
  }
}
```


Containers

- *Container*:
  - Component that holds other components
  - eg, see **J**... components
- *Top-level container*:
  - special kind of container (eg, **JFrame**)
  - holds all components that will appear on screen



JFrame



Next, a bit more about JFrame....

JFrame

- commonly-used top-level container
- example)

```
JFrame f = new JFrame("Title!");
f.getContentPane().add(new JButton("OK"));
```

- using default layout manager to place components
- another very useful container...

JPanel

- Simplest container:
  - opaque container
  - handy for place to draw graphics
  - store components but no borders
- Not top-level:
  - cannot be “stand-alone”
  - must put in other container
- example)

```
JFrame frame = new JFrame("Title!");
JPanel panel = new JPanel();
p.add(new JButton("OK!"));
frame.getContentPane().add(panel);
```

- Some complete examples...

Example 1

```
import javax.swing.*;

public class Basic1 {

  public static void main(String[] args) {

    // Create window:
    JFrame f = new JFrame("Basic Test!");

    // Set 500x500 pixels^2:
    f.setSize(500,500);

    // Show the window:
    f.setVisible(true);

    // Quit Java after closing the window:
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

  }

}
```


Example 2

```
import javax.swing.*;

public class Basic2 {
  public static void main(String[] args) {
    new MyGUI();
  }
}

class MyGUI {
    {
      JFrame f = new JFrame("Basic Test2!");
      f.setSize(500,500);
      f.setVisible(true);
      f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```


Layout Managers

- What is a ***layout manager***?
  - object that controls placement and sizing of components in container
  - if you do not specify a layout manager, the container will use a default:
    - `JPanel`—`FlowLayout`
    - `JFrame`—`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());
```


Some Layout Managers

- **`FlowLayout`**:
  - components arranged in container from left to right in order added
  - new row started each time row ends
  - simple alignment with **`RIGHT`**, **`LEFT`**, **`CENTER`** fields
  - see also **`BoxLayout`**
- **`GridLayout`**:
  - arranges components in rectangular grid (think array)
  - rows, columns defined by constructor
  - components go into grid left-to-right, then top-to-down
- **`BorderLayout`**:
  - divides window into 5 areas: East, South, West, North, Center
  - add components with **`add(`*`Component,index`*`)`**
  - indices are **`BorderLayout.EAST`**, …

More Layout Managers

- **`CardLayout`**:
  - tabbed index card look from Windows
- **`GridBagLayout`**:
  - most versatile, but most complicated
- Custom:
  - define your own layout manager
  - best to try Java's supplied version first...
- Null Layout
  - don't use a layout manager
  - programmer has to give absolute locations
  - can be dangerous to application because of platform dependency

Complete Statics Example

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

public class Statics1 {
  public static void main(String[] args) {
    new MyGUI();
  }
}

class MyGUI {
  private JFrame f;
  private Container c;

  public MyGUI() {
    f = new JFrame("Statics1");
    f.setSize(500,500);

    c = f.getContentPane();
    c.setLayout(new FlowLayout(FlowLayout.LEFT) );

    for (int b = 1; b < 9; b++)
      c.add(new JButton("Button "+b));

    f.setVisible(true);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}
```


Second Statics Example

```
import javax.swing.*;
import java.awt.*;
public class Statics2 {
  public static void main(String[] args) { new MyGUI(); }
}

class MyGUI {
  private JFrame f;
  private Container c;
  private LayoutManager l;
  private MyPanel[] p;
  private int dim;

  public MyGUI() {
    makeWindow();
    showWindow();
  }
```


Example Continued

```
  private void makeWindow() {
    dim = 4;
    f = new JFrame("Statics2");
    p = new MyPanel[dim*dim];
    c = f.getContentPane();
    l = new GridLayout(dim,dim,2,2);
    c.setLayout(l);
    for (int i=0;i<p.length;i++) {
      p[i]=new MyPanel();
      c.add(p[i]);
    }
  }

  private void showWindow() {
    f.setSize(500,500);
    f.setVisible(true);
    f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
  }
}

class MyPanel extends JPanel {
  public void paintComponent (Graphics g) {
    super.paintComponent(g); // clear drawing area
    g.setColor(Color.white);
    g.fill3DRect(0,0,getWidth(),getHeight(),true);
  }
}
```