Lecture 8: Graphical User Interfaces

Time: Tuesday, March 24
Location: Lab III (BIT Lab)

Object-Oriented Programming in Java

Melissa R. Ho, PhD Candidate
University of California, Berkeley
School of Information
Office Hours: Tuesdays 3-5, or by appt.

Announcements

- Office Hours, Tuesdays 3-5
- Midterm, April 7
  - Classes, Objects, Methods
  - Inheritance, Abstract Classes, Interfaces
  - Recursion
  - Data Structures: Linked Lists, Vectors, Hashtables
  - List Manipulalation (creating, adding items, removing items)
- Class Projects
  - Meetings on April 14-15 in Project Groups to select topics
    - Appointment sign-up sheet will be on my door
  - A Working Java Application – to be demo'd on May 5-6
  - Groups of 5-6 Students

List Review

What does this code do?

```
ListCell list;
list = new ListCell("embuzi", null);
list = new ListCell("ente", list);
list = new ListCell(new Integer(5), null);
list = new ListCell(new Integer(-6), list);
list = new ListCell(new Integer(4), list);
list = new ListCell(new Integer(3), list);
list = new ListCell("nkoko", list);
```

Note that each time you call "new" you are creating a new ListCell object! That means you draw a double-box…

Overview: Graphical User Interfaces

- AWT and Swing
- Class Hierarchy
- Components and Containers
- Layout Managers

+

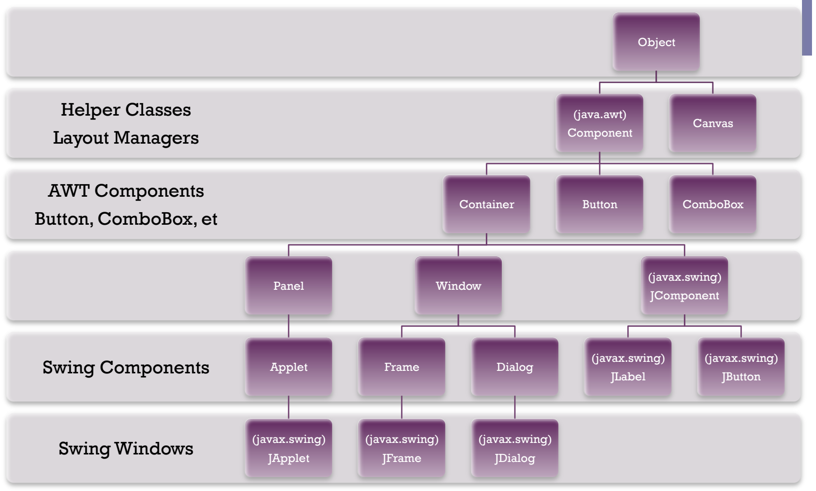
AWT and Swing

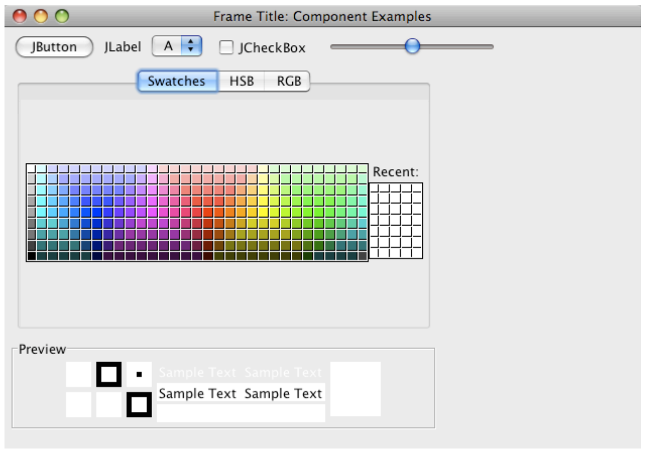
The Abstract Windowing Toolkit (AWT) and Swing are Java's built in packages for building GUIs

- AWT: a **heavyweight** windowing toolkit
  - uses code for the windowing system from your computer
  - doesn't port well to other operating systems (OSes)
  - basic API package: java.awt.*
- Swing: a *lightweight* windowing package
  - written mostly in Java
  - more portable, added functionality
  - similar names to AWT classes, but starts with "J"
  - basic API package: javax.swing.*
- Does Swing replace AWT?
  - AWT still used for the event model (to be covered next week)
  - AWT still needed for each OS
  - See http://java.sun.com/products/jfc/tsc/articles/mixing

+

Class Hierarchy (Java API)





- visual part of the interface
- represents something with a **position** and **size**
- buttons, labels

- can be painted on screen and receive events in reponse to user actions or program responses

+ Components

In Java components are the individual Objects that you put together to compose a graphical user interface. Essentially you are **painting them on the screen**.

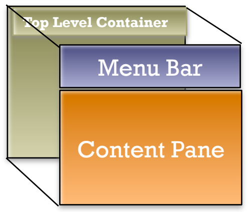
+ An Example

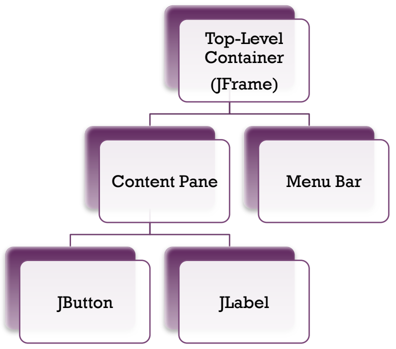
This Java Program creates and displays the window that I showed you on the previous slide.

```
package must.ics.oop.lec7;

import java.awt.FlowLayout;

import javax.swing.*;

public class ComponentExamples extends JFrame {

    /**
     * Constructor for the ComponentExamples class
     */
    public ComponentExamples()
    {
        // call JFrame's constructor, which takes a String argument as a title
        super("Frame Title: Component Examples");

        // Tell the frame how you want your components to be distributed
        // within the frame.  In this case, you want the components (i.e. parts,
        // to be laid out from left to right, left aligned
        setLayout(new FlowLayout(FlowLayout.LEFT));

        // for each, the string argument is the title that will
        // be displayed on the component
        add(new JButton("JButton"));
        add(new JLabel("JLabel"));
        add(new JComboBox(new String[] {"A", "B", "C"}));
        add(new JCheckBox("JCheckBox"));
        add(new JSlider(0,100));
        add(new JColorChooser());

    }

    /**
     * @param args
     */
    public static void main(String[] args) {
        ComponentExamples f = new ComponentExamples();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.pack();
        f.setVisible(true);
    }
}
```

Containers

- A Container is a special Component that can hold other Components
- A **top-level** container (e.g. JFrame) is a special container that holds all the components that will appear on the screen





javax.swing.JFrame

- Commonly used top-level container
- Example

```
JFrame frame = new JFrame("Title of frame!");
```

- Using default layout manager to place components
  - You can use the default layout manager to place components
  - Or you can set different layout managers!
- Content Pane
  - Old Java: add to content pane

    ```
    frame.getContentPane().add(new JButton("OK"));
    ```
  - Java 1.5: add to frame directly puts them in content pain

    ```
    frame.add(new JButton("OK"));
    ```
  - See JRootPane in Java API for more Info

javax.swing.JPanel

- Simplest Container
  - opaque container
  - handy for drawing graphics
  - stores components but no borders
  - cannot stand alone
- Example

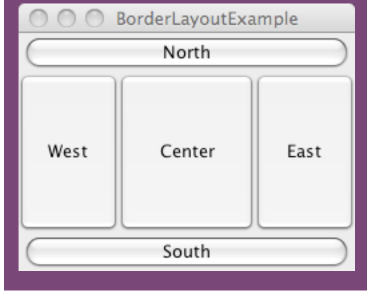
```
JFrame frame = new JFrame("Title");
JPanel panel = new JPanel();
panel.add(new JButton("OK"));
frame.add(panel);
```

Layout Managers

- BorderLayout
- BoxLayout
- FlowLayout
- GridBagLayout
- GridLayout
- container.setLayout(new LayoutManager());

BorderLayout

This layout to positions components or containers in five locations, depending on what you specify when you add the component. The size of the component expands to fill the available space, with the "Center" component occupying the most space.

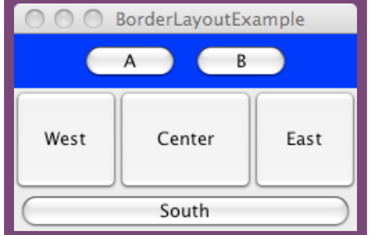


```
package must.ics.oop.lec7;
import java.awt.BorderLayout;

import javax.swing.JButton;
import javax.swing.JFrame;

public class BorderLayoutExample extends JFrame {

    public BorderLayoutExample()
    {
        super("BorderLayoutExample");

        setLayout(new BorderLayout());


        add(new JButton("North"),"North");
        add(new JButton("Center"), "Center");
        add(new JButton("East"), "East");
        add(new JButton("West"),"West");
        add(new JButton("South"),"South");
    }
    /**
     * @param args
     */
    public static void main(String[] args) {
        BorderLayoutExample f = new BorderLayoutExample();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.pack();
        f.setVisible(true);
    }
}
```

Lecture 7: Graphical User Interface Statics 5/5/09

BorderLayout

You can add more than one component to a frame using a BorderLayout by layering components within containers. Here JButtons "A" and "B" are within a JPanel (using default FlowLayout), which is then placed within the "North" slot of the BorderLayout of the BorderLayoutExample JFrame.



```
import java.awt.Color;

import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;

public class BorderLayoutExample extends JFrame {

    public BorderLayoutExample()
    {
        super("BorderLayoutExample");

        setLayout(new BorderLayout());

        JPanel panel = new JPanel();
        panel.add(new JButton("A"));
        panel.add(new JButton("B"));
        panel.setBackground(Color.BLUE);

        add(panel,"North");
        add(new JButton("Center"), "Center");
        add(new JButton("East"), "East");
        add(new JButton("West"),"West");
        add(new JButton("South"),"South");
    }
    /**
     * @param args
     */
    public static void main(String[] args) {
        BorderLayoutExample f = new BorderLayoutExample();
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.pack();
        f.setVisible(true);
    }
}
```

+

Questions?

Contact
Melissa R. Ho
mho@ischool.berkeley.edu
Office Hours: Tuesdays 3-5, or by appt
ICS, Office 1

Note: These lecture notes are based on lectures from the CS211 course taught at Cornell University during Spring 2008, co-taught by Dexter Kozen, Rich Caruana, and David Schwarz.