Announcements

- A3 is up, due Friday, Oct 9
- Prelim 1 scheduled for Thursday October 15
  - We do NOT have any scheduled makeup exam
  - If you have a conflict, let us know now; you can take
    the exam a little earlier on the same day

Interactive Programs

- “Classic” view of computer programs: transform inputs to outputs, stop
- Event-driven programs: interactive, long-running
  - Serves interest with clients
  - Applications interact with user(s)

GUI Motivation

- 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

Swing versus SWT

- Swing builds on AWT
  - Strives for total portability
  - Secretly seems to have a grudge against Windows
  - Basic architecture is pretty standard

- SWT is “new”
  - Goal is best performance
  - Great fit with Windows system
  - Basic architecture is pretty standard

Lonnie opted for SWT in A3 but could just as easily have used Swing. He feels that use of SWT simplified his code.
Java Foundation Classes

- **Pluggable Look-and-Feel Support**
  - Controls look-and-feel for particular windowing environment
  - E.g., Java, Windows, Motif
- **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

<table>
<thead>
<tr>
<th>Statics: what's drawn on the screen</th>
</tr>
</thead>
<tbody>
<tr>
<td>Components</td>
</tr>
<tr>
<td>- button, labels, lists, sliders, menus, ...</td>
</tr>
<tr>
<td>Containers: components that contain other components</td>
</tr>
<tr>
<td>- frames, panels, dialog boxes, ...</td>
</tr>
<tr>
<td>Layout managers: control placement and sizing of components</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dynamics: user interactions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Events</td>
</tr>
<tr>
<td>- button-press, mouse-click, key-press, ...</td>
</tr>
<tr>
<td>Listeners: an object that responds to an event</td>
</tr>
<tr>
<td>Helper classes</td>
</tr>
<tr>
<td>- Graphics, Color, Font, FontMetrics, Dimension, ...</td>
</tr>
</tbody>
</table>

Creating a Window in SWT

```java
import org.eclipse.swt;
import org.eclipse.swt.widgets.*;

public class HelloWorld {
    public static void main(String[] args) {
        // create the window
        Display display = new Display();
        Shell shell = new Shell(display);
        Label label = new Label(shell, SWT.NONE);
        label.setText("Basic Test!");
        label.pack();
        shell.pack();
        shell.open();
        // quit Java after closing the window
        while (!shell.isDisposed()) {
            if (!display.readAndDispatch())
                display.sleep();
        }
        display.dispose();
    }
}
```

Creating a Window in Swing

```java
import javax.swing.*;

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

Creating a Window Using a Constructor

```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() {
        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);
    }
}
```

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() {
        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, ...
- Some windows have hidden components that become visible only when the user takes some action

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"})
            .addItem("item1")
            .addItem("item2")
            .addItem("item3"));
        add(new JComboBox(new String[] {"A", "B", "C"}));
        add(new JCheckBox("JCheckBox"));
        add(new JSlider(0, 100));
        add(new JColorChooser());
        setDefaultCloseOperation(EXIT_ON_CLOSE);
    } 
    public static void main(String[] args) {
        try {
            UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
        } catch (Exception exc) {} 
        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
  - heavy-weight vs. lightweight
  - A heavy-weight component interacts directly with the host system
  - JWindow, JFrame, and JDialog are heavy-weight
  - Except for these top-level containers, Swing components are almost all lightweight
  - JPanel is lightweight

Another important container
- JPanel: used mostly to organize objects within other containers

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

A Component Tree

```
JFrame
   
 JPanel
    
  JComboBox
   
  JPanel
    
  JTextField
  
  JPanel
    
  JTextField
  
  JPanel
    
  JComboBox
      
  JTextField
  
  JPanel
    
  JTextField
  
  JPanel
    
  JComboBox
```

```java
import javax.swing.*;

public class JFrame {
    
    public static void main(String[] args) {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.add(new JButton("Button"));
        frame.add(new JComboBox(new String[] {"A", "B", "C"}));
        frame.add(new JLabel("Label"));
        frame.add(new JTextField("TextField"));
        frame.add(new JColorChooser());
        frame.setVisible(true);
    }
}
```
## 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.
- Five common layout managers: BorderLayout, BoxLayout, FlowLayout, GridLayout, GridBagLayout

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

### Examples
```
FlowLayout
```
```
import javax.swing.
import java.awt.
public class Statics1 {
    public static void main(String[] args) {
        JFrame f = new JFrame("Statics1");
        f.setSize(500, 200);
        f.setVisible(true);
        for (int b = 1; b < 9; b++)
            f.add(new JButton("Button " + b));
    }
}
```

### BorderLayout
```
BorderLayout

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

### Adding components
```
FlowLayout and GridLayoutManager use container.add(component)
BorderLayout uses container.add(component, index) where index is one of
- BorderLayout.NORTH
- BorderLayout.EAST
- BorderLayout.WEST
- BorderLayout.CENTER
```

###FlowLayout Example
```
import java.swing.*;
import java.awt.*;
public class Statics1 {
    public static void main(String[] args) {
        JFrame f = new JFrame("Statics1");
        f.setSize(500, 200);
        f.setVisible(true);
        for (int b = 1; b < 9; b++)
            f.add(new JButton("Button " + b));
    }
}
```

### BorderLayout Example
```
import java.swing.*;
import java.awt.*;
public class Statics1 {
    public static void main(String[] args) {
        JFrame f = new JFrame("Statics1");
        f.setSize(500, 200);
        f.setVisible(true);
        for (int b = 1; b < 9; b++)
            f.add(new JButton("Button " + b));
    }
}
```

### More Layout Managers

- **CardLayout**
  - Tabbed index card look from Windows
  - 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: `javax.swing.*`

**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