Objectives:

1. To introduce the notion of a “component” and some basic Swing components (JLabel, JTextField, JButton)
2. To introduce the rudiments of using containers, with absolute positioning.
3. To introduce event-driven programming using an action listener

Materials:

1. Dr. Java to demonstrate individual operations
2. Demo program: SimpleGUIDemo.java, .class, .html
3. Demo program: GUIDemo2.java, .class, .html

I. Introduction

A. Today we will begin looking at creating and using graphical user interfaces (GUI’s) in a program. This is a large subject, which we will continue to develop later in this course and in future courses. But after this series of lectures you will be able to create and use simple GUI’s.

B. A GUI performs two major tasks:

   1. It displays information (graphics, text, and controls) on the screen.
   2. It responds to user actions such as clicking the mouse button.

C. The fundamental concept in a GUI is the window:

   1. All of a program’s output is displayed in one or more windows that the program creates.
   2. A program responds to input when the user takes some action with the cursor positioned over a window belonging to that program. (Thus, when two or more programs are running, it is possible to change from program to program simply by clicking on different windows).
   3. Many times, a window has a menu bar associated with it. On Wintel platforms, this is displayed as part of the window; on the Mac it is displayed at the top of the screen but is still associated with the currently active window as you can see by watching what happens to the menu bar as you click on windows belonging to different applications.
4. The critical role of windows in GUI applications is highlighted by the fact that Microsoft has chosen the name Windows for its family of operating systems that succeeded its command-line based DOS systems.

II. Introduction to the Java awt

A. One of the distinctive features of Java is its built-in support for implementing graphical user interfaces. This is done through a portion of the standard Java library called the abstract windowing toolkit (often referred to as awt) and another portion - which builds on the awt - called Swing. It is possible to build significant GUI's using just the awt, but Swing provides much richer facilities. We will focus on using Swing in this course, though we will have to make some reference to portions of the awt that Swing uses.

1. The classes comprising the awt reside in the package java.awt, and those comprising Swing reside in the package javax.swing. To use Swing in a program, one normally includes the statements

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

2. In addition, it may be necessary to import one or more subpackages - e.g.

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

   (This package contains classes that are used for responding to user input. GUI-related actions performed by the user - e.g. clicking the mouse button - result in the creation of special objects called events that the program can respond to. Both awt and Swing make use of these classes.)

3. The awt and Swing are quite large - consisting of over 90 classes in the awt package and 120 in the Swing package in JDK 1.4, plus several subpackages with additional classes. We will only give a first introduction to them now.

B. One of the key classes in awt is the class Window, which represents a displayable window on the screen. In order to do any GUI activities, a program must utilize one or more objects belonging to this class or - more typically - a subclass derived from it. Everything that a program displays resides in some window.

1. In labs 3 and 4, we have made use of a class called JFrame. This class is a Swing extension of the Window class, and is Swing's top-level window.
2. When an applet is run, the host application creates a window for it, and places the applet in the window - so the window actually belongs to the “host” application in this case.

3. This is perhaps the fundamental distinction between the structure of an application and that of an applet - an application has to create its own window(s); an applet relies on its host application to do that for it, and is, in fact, already a component of that window when it is “born”.

C. The awt and Swing classes fall into several broad categories:

1. Windows and components that can be displayed in windows.
2. Classes used to manage the layout of windows.
3. Classes supporting adding menus to a window.
4. Classes used to support graphics (e.g. the Graphics and Color classes we have used for drawing.)

D. In addition, there are several subpackages. If classes from any of these is needed, the subpackage must be explicitly imported. The only one we will use is the java.awt.event subpackage with classes that support responding to user activities such as clicking a button.

E. Today, we will cover a few of the displayable components (JButton, JLabel, and JTextField), say a little bit about containers, and give a brief introduction to the event handling facilities. Using these together, it is possible to build simple GUI’s.

III. Introduction to Swing Components

A. One of the fundamental classes in the awt package is the class Component. This class is the root of a hierarchy of classes that represent windows and things that users can see in windows. It has two kinds of subclasses:

1. Subclasses representing individual GUI components - including the following Swing components, all descended from Component by way of JComponent, plus others:
   a) JButton
   b) JCheckBox
   c) JComboBox
d) JLabel
e) JList
f) JProgressbar
g) JRadioButton
h) JScrollPane
i) JSlider
j) JSpinner
k) JTable
l) JTextArea
m) JTextField
n) JToggleButton
o) JTree

2. Containers - components that can themselves hold other components. All containers are subclasses of the awt class Container.

   a) A JFrame or JApplet has a container called its content pane which actually holds its contents.
   b) A JPanel is a generic container for grouping related items within a window.
   c) A JScrollPane is a container for holding a component or container that is too large to all fit on the screen at once, so scroll bars are used to control what part is actually seen.

3. Today, we are going to learn about three kinds of ordinary components: JLabels, JTextFields, and JButtons plus a little bit about containers.

B. We will demonstrate the various components using Dr. Java.

   Setup:

   ```java
   import java.awt.*;
   import javax.swing.*;
   JFrame f = new JFrame();
   Container p = f.getContentPane();
   p.setLayout(new GridLayout(0,1));
   f.show();
   ```

C. A JLabel is a component that displays text on the screen.

   1. The text in a label cannot be edited by the user.
2. A label is created by code like the following:

   JLabel prompt = new JLabel("Please type your name");

   Note that the constructor takes a parameter that specifies what the JLabel is to display.

   DEMO: the above code, then
   p.add(prompt);
   f.pack();

3. It is also possible to create a JLabel without specifying any text, and then specify the text later - e.g.

   JLabel result = new JLabel();
   ...
   result.setText("The answer is 42");

   DEMO: the above constructor code, then
   p.add(result);
   f.pack();
   then the setText() code.

4. Finally, the JLabel constructor takes an optional parameter to specify how the label is positioned within the area reserved for it on the screen - e.g.

   JLabel message = new JLabel("Hello", JLabel.CENTER);

   Possible values of the centering parameter are:

   a) JLabel.LEFT (default) - left justified
   b) JLabel.CENTER - centered
   c) JLabel.RIGHT - right justified

   DEMO: the above code, then
   p.add(message);
   f.pack();
D. A JTextField is a component that displays editable text on the screen.

1. In contrast to a JLabel, the text that is displayed in a JTextField can be edited by the user. The library class provides support for normal editing operations like placing the cursor, inserting and deleting characters, and cut and paste. (The program can also disable user editing and re-enable it later if desired.)

2. A JTextField is normally created by code like the following

   JTextField nameIn = new JTextField(20);

   where the integer specifies the number of characters to be displayed. (This is not an upper limit on the number of characters that can be typed, since the field will scroll if necessary.)

   DEMO: the above code, then
   p.add(nameIn);
   f.pack();

3. It is also possible to specify initial contents for the field

   a) e.g.

   JTextField huh = new JTextField("unknown");

   In this case, the width can be omitted, since it can be inferred from the initial content (e.g. in the above example, 7) - or a different width can be specified to allow the input to be wider than the initial value.

   DEMO: the above code, then
   p.add(huh);
   f.pack();

   b) Note well the difference between the following two cases:

   JTextField age = new JTextField(18);
   JTextField age = new JTextField("18");

   The former creates a text field wide enough for 18 characters - initially empty; the latter creates a text field wide enough for 2 characters - initially containing "18".
E. A JButton is a component that a user can click to request that some particular action occur.

1. A JButton is typically constructed as follows:

```java
JButton ok = new JButton("OK");
```

where the string specified is the name that appears inside the button

DEMO: the above code, then

```java
p.add(ok);
f.pack();
```

2. When we talk about GUI events later in the lecture, we will learn how the program can respond to events generated when the button is clicked.

IV. Introduction to Containers

A. A container is a special kind of component whose basic task is to hold and position other components.

1. Top level windows (JFrames) and applets (JApplets) have a container called the content pane that holds their actual contents.

2. Another kind of container is a JPanel, which can be used to group components in another container - a sort of “window within a window”.

B. Container objects have a method called add, which can be used to add any component (including another container) to the container. Once a component has been added to a container, it will be displayed inside that container. (It is an error, and can cause weird run time problems, to add the same component to more than one container, or to add it twice to the same container.)

Example: If we wanted to create an applet that displays the message Hello, one way to do it would be as follows:

SHOW SimpleGUIDemo.java

DEMO (drag the .html file to browser)
C. Layout of Components

1. One of the tasks of a container is to manage the layout of its components - i.e. where each component appears on the screen, and how much screen space is allocated to it.

2. The standard way to do this is through a special object associated with the container called a Layout Manager. The java.awt package includes a number of different kinds of layout managers that implement different policies. Layout managers can do some very sophisticated layout work, but they are complicated to use. For this reason, Wu does not discuss them at all. (We will say something about them when we get to chapter 14, and we will make use of some in “cookbook” fashion in lab.). (Note: the example we just did made use of a default layout manager which worked fine for the case where we only had one component.)

3. The approach to layout that Wu uses is called absolute positioning - in which we explicitly specify the position and size for each component.

   a) This is a much simpler approach in simple cases like the examples in the book.

   b) It is not, however, the recommended approach for most programs for two reasons:

      (1) Absolute positioning is somewhat dependent on details about the display device used to show the GUI, whereas the standard layout manager objects handle this automatically.

      (2) Absolute positioning is not responsive to changes in the size of a window resulting from resizing by the user, but the standard layout managers handle this as well.

   c) To use absolute positioning, you must do two things:

      (1) Set the layout manager for the container to null.

         If you are inside the constructor for the container, this can be done by:

         ```java
         setLayout(null);
         ```

         or, in the case of the constructor for a JFrame or JApplet:

         ```java
         getContentPane().setLayout(null);
         ```

         Otherwise, it is done by

         ```java
         whateverContainer.setLayout(null);
         ```
(2) Invoke the setBounds method on each component after adding it to the container:

```java
whateverComponent.setBounds(x, y, w, h);
```

Where x and y are the coordinates of the upper left-hand corner of the component relative to its container, w is the width, and h is the height.

d) When doing absolute positioning, it should be noted that the coordinates assigned to a component are *relative to its container*. This is, of course, also true when using the paint method:

*Example:* Suppose we have a window whose corner is currently at (100, 50) on the screen. Suppose it contains a component whose bounds were set to (40, 20). Suppose that the draw method for the component contains `graphics.drawRect(70, 30, 200, 100)`.

(window at 100, 50 relative to screen)

(Component at 40, 20 relative to window)

(Rectangle at 70, 30 relative to Component it’s drawn in)
The actual coordinates of the rectangle on the screen are:

Upper left corner - x:
100 (x position of window) + 40 (x position of component in window) + 70
(x position of rectangle in component) = 210

Upper left corner - y:
50 (y position of window) + 20 (y position of component in window) + 30 (y
position of rectangle in component) = 100

(Actually, both x and y would be somewhat bigger than this because of space
taken by the outline of the window on the screen - the coordinates of the
component are relative to the inside of the window.)

D. An Example of Creating a GUI Within a container: GUIDemo2.java

SHOW code - ignore ActionListener stuff and actionPerformed method for now

DEMO (drag .html file to browser)

V. Introduction to Event-Driven Programming

A. We said at the outset that GUI’s perform two basic tasks: displaying
information and responding to user input. We now turn to the handling of
the second task.

B. Any windowing operating system has a software component that responds to
changes in the position of the mouse, pressing or releasing the mouse’s
button(s), and pressing of keys. Each such action by the user constitutes an
event, which this component of the OS delivers to the appropriate application
(namely the application that owns the window that the cursor is over when
the event occurs.) At this point, further processing of the event is up to the
application.

C. In the case of Java, the Java awt provides a standard mechanism for handling
events that any program can build on.

To use the event handling mechanism, a program must import the package
java.awt.event.

D. When a Java program receives an event, the Java library delivers it to the
appropriate GUI component - e.g. to the JButton object if the mouse is over
a button; to the JTextField object if the mouse is over a text field, etc.
1. A given type of component may handle certain types of events on its own - e.g. a key pressed event that is delivered to a text field object causes the character that was typed to be inserted in the text at the appropriate point.

2. User-written software may also express an interest in handling a particular type of event by registering an event listener with the component. When an event listener is registered, and the appropriate type of event occurs, the event listener is activated to respond to it.

3. Events that the component is not interested in and that have no registered listeners are simply ignored. For example, every mouse movement results in an event, but the vast majority of them are ignored. (One could register an interest in mouse movements, however, if one wanted to highlight some component on the screen when the mouse was moved over it.)

4. Java has its system for classifying types of events. The only type of event we will talk about today is an ActionEvent - which is created whenever a user does something that typically calls for an active response from the program - e.g.

   a) Clicks a button
   b) Presses return while typing in a text field
   c) Chooses a menu item
   etc.

E. To register an event listener with a component, one uses a listener object that implements the appropriate interface. In the case of an ActionEvent, a listener object must

1. Be declared as implementing ActionListener. ActionListener is what is called an interface in Java - a specification for behavior that all objects that implement the interface must have. In this case, the necessary behavior is having an actionPerformed() method that handles an action event in an appropriate way.

2. Have a method with the following signature:

   public void actionPerformed(ActionEvent e)

3. Be registered with the component. This is done by some code (typically the constructor for the GUI) sending an addActionListener message to the component, passing as a parameter the listener object.
example: Show each of the above in GUIDemo2.java

F. When an event occurs for which there is an appropriately registered listener, the `actionPerformed()` method of the listener object is called. The actual event is represented as an object that is passed as a parameter to this method. (The event object contains information that varies from type of event to type of event, but typically includes the precise coordinates where the cursor was when the event occurred, details about any modifier keys that were pressed, etc.) The `actionPerformed()` method is responsible for doing what needs to be done to respond to the event.

example: Show code for `actionPerformed()` in GUIDemo2.java

note: If we also wanted the user to be able to initiate computation by clicking return after typing in the text input box, we could add the following code to the constructor, making the applet be a listener for action events emanating from either the text field or the button.

```
numberInput.addActionListener(this);
```

In this case, an action event emanating from either component would activate the applet’s `actionPerformed()` method. If the applet needed to know which component was the source of the event, it could find out by examining the `ActionEvent` object passed as a parameter.

G. Note how the action listener reads the number typed by the user as a string and then uses the “wrapper class” approach to convert it to a number that can be used in computation. It is instructive to see what happens when a “bad” value is typed.

DEMO with -1 as input

DEMO with abc as input - show opening Console application to see what happened