Overview of Part 1 of the Course

- Demystifying Java: Simple Code
- Introduction to Java
- An Example of OOP in practice
- OOP Concepts -- Advanced
- Hints and for Java
- I/O (Streams) in Java
- Graphical User Interface Coding in Java
- Exceptions and Exception handling

Overview of Java GUI Programming

- Introduction and Observations
- Events Models
- IDEs versus hand-rolled code
- Interfaces and Adapters
- Layout methods
- Interface Objects
  - Heavyweight Containers
  - Containers
  - Components
Introduction to Java

Interface Programming

- Historically, user interfaces were developed after the difficult coding was completed.
- Over the years, the amount of effort dedicated to the interface has increased dramatically.
- Understanding how the interface works makes it easier to debug problems or learn new approaches.
- While we encourage the use of Integrated Development Environments (IDEs), this first course focuses on hand toolled code.

Integrated Development Environments

- Integrated Development Environments (IDEs) are used to build complex and standard interfaces.
- Many of the IDE efforts to simplify Java interface coding have resulted in non-standard Java.
- "Hand tooling" gives you a better sense of what's going on underneath.
- This is important for helping you better understand building quality code.

GUIs are event driven systems

- The interface consists of a hierarchy of objects -- sometimes referred to as window objects or widgets.
- These objects fall into two broad categories:
  - Components are the basic building blocks -- buttons, lists, menus, etc.
  - Containers are the basic objects that hold the objects -- panels, dialogs, etc.
- Components generate events as the result of user action.
- These events are passed to "event handlers" -- methods that wait "listening" for an event to occur.
- In Java these methods are implemented as interface methods from java.awt.event, etc.
Evolution of Java HCI Components

- Java has developed rapidly over the last few years
- The original set of Java widgets was weak
- The Java 1.0 Abstract Windowing Toolkit (AWT) had few components and a weak event model
- In Java 1.4, AWT improved event handling
- In Java 1.2, new window objects were added
- These window objects allow for a richer user interface
- They are part of a broader set known as the Java Foundation (JFC)
- The Swing widgets more than double the number of interface objects and very substantially increase functionality
- The JFC contains a number of features including drag and drop, keyboard accelerators, tabs, and feel and look

Swing

- The Swing component and container classes provide a rich basis for interface construction
- The Java 1.2, the 2D Graphics API provides additional drawing capabilities
- In Java 1.1, the graphics drawing capabilities were primitive—2D lines only, for example
- Keep in mind that although Swing is built upon the AWT, the use of the AWT is deprecated
- AWT continues to be imported, primarily because of its event classes

Developing an Interface

- There are four basic steps
- Create the interface objects
- Create methods to handle events generated by the objects
- Register the event generators and event handlers with the program
- These are the tools we need to build an interface
- How do we create the objects
- What classes are available for constructing objects
- How do we position the objects
- How do we define methods to handle events
- How do we link the objects, detecting the events with the listener methods
Swing Interface Objects

- The Swing package contains a large number of parts.
- We examine only a few of them in three categories:
  - heavyweight containers
  - lightweight containers
  - components
- The various classes are in the javax.swing package.
- The classes are named JXxxx:
  - each class is prefixed by the letter J and starts with a capital letter e.g., JFrame
- Most Swing classes are lightweight.
  - Lightweight components are not mapped to their own window -- they share the window of a parent.

Heavyweight Containers

- Heavyweight containers are used for base windows.
- Heavyweight containers are JWindow, JFrame, and JDialog.
- JFrame
  - is the base class for heavyweight containers.
  - it is balanced in every way compared to a frame.
  - for example, it knows how to close itself.
  - still need to override the close method to exit.
  - it has a default container for components (a JPanel).
- JDialog
  - an important aspect of a dialog is its ability to be modal.
  - unlike a frame, it does not have a menu bar.

Lightweight Containers

- There are many different kinds of panels all of which have interesting and useful functions.
- System designers should have some sense of what can and can’t be done with each.
- JPanel is the most basic container.
  - this is the container, primarily responsible for layout objects in frames or dialogs.
  - it is the basic panel for components that both text and graphics.
- In this section we:
  - briefly examine the JRootPane
  - take a little longer look at the JSplitPane
  - define the functionality of a JTabbedPane
JRootPane

- A basic panel used for the content of frames, windows, and dialogs.
- It is generally not created, but simply used.
- It is constructed of:
  - contentPane (JPanel) which contains the components
  - glassPane (JPanel) which traps mouse events
  - layeredPane (JLayeredPane) which holds contentPane and menubar
- It has several important methods
  - getContentPane()
  - getGlassPane()
  - setContentPane()

JSplitPane

- The basic focus of this object is to allow a sliding window on two components.
- In an editor, this might be two separate text windows on the same document.
- It could also be a graphical and textual view of the same object.
- It provides utilities for updating the components in the split pane.

JSplitPane Constructors

- The constructors (selected) include:
  - JSplitPane(): Returns a new JSplitPane configured to arrange the child components side-by-side horizontally with no continuous layout. (Two buttons are used as default components.)
  - JSplitPane(int newOrientation, boolean newContinuousLayout): Returns a new JSplitPane with the specified orientation and redrawing style.
  - JSplitPane(int newOrientation, boolean newContinuousLayout, Component newLeftComponent, Component newRightComponent): Returns a new JSplitPane with the specified orientation and redrawing style, and with the specified components.
Selected JSplitPane Fields

- A JSplitPane has several properties, including:
  - `bottomComponent` the component to the bottom or right
  - `rightComponent` the component to the bottom or right
  - `topComponent` the component to the top or left
  - `leftComponent` the component to the top or left
  - `dividerLocation` either a real which gives a proportion of the window, or:
    - an integer which gives the pixel location
  - `orientation` an int specifying the orientation
    - the default is horizontal

Selected JSplitPane Methods

- JSplitPane informational methods include:
  - `Component getBottomComponent()` Returns the component below, or to the right of the divider.
  - `int getDividerLocation()` Returns the location of the divider from the look and feel implementation.
  - `int getDividerSize()` Returns the size of the divider.
  - `int getMaximumDividerLocation()` Returns the maximum location of the divider from the look and feel implementation.
  - `int getOrientation()` Returns the orientation.
  - `boolean isContinuousLayout()` Returns true if the child components are continuously redisplayed and reflowed during user intervention.
  - `boolean isOneTouchExpandable()` Returns true if the pane provides a UI widget to collapse/expand the divider.

- JSplitPane action methods include:
  - `void remove(Component component)` Removes the child component, component from the pane.
  - `void resetToPreferredSizes()` Messaged to relayout the JSplitPane based on the preferred size of the children components.
  - `void setBottomComponent(Component comp)` Sets the component below, or to the right of the divider.
  - `void setDividerLocation(double proportionalLocation)` Sets the divider location as a percentage of the JSplitPane's size.
  - `void setOrientation(int orientation)` Sets the orientation, or how the splitter is divided.
JTabbedPane
• provides an environment for organizing large amounts of information
• allows a progressive disclosure of information
• utility functions allow positioning and tabbing among other things
• tabs can set top, bottom, left or right of window
• whole tabs can be enabled or disabled

Exercise
• Look up the Fields, Constructors, and Methods for a JTabbedPane and summarize them, suggesting several possible uses for this container.
• Consider other containers as well including:
  • JScrollPane
  • JEditorPane
  • JTextPane
  • JLayeredPane

Components
• Components are the action producers and display objects that make up the user interface
• They can be very simple -- a JLabel or JButton
• They can be very complex -- a JTable, JComboBox, JTextArea, or JMenuBar
• The Swing components have many new features and utilities that make them powerful and flexible
• This section begins with a look at JComponent
• We then look at four components in some detail -- JLabel, JButton, JTextField, and JTextArea
• Finally, we give a functional overview of JMenu
JComponents

- The class from which all components are derived
- It provides incredible common functionality
  - A "pluggable look and feel" option
  - Easy extension to create custom components
  - Keystroke-handling that works with nested components
  - Adjustable property
  - The ability to set the component minimum and maximum size for a component
  - TextTips – short descriptions that pop up on mouse hover
  - Autoscrolling – automatic scrolling in a list, table, or tree during mouse drag
  - Support for Accessibility and international Localization

JLabel

- Provides graphic, text, and combination labels
- It also now allows the use of HTML text
- To use HTML text, begin with "<html>…"
- Use the use of lower case
- The most general constructors are:
  - JLabel(String text) Creates a JLabel with the specified text.
  - JLabel(Icon image) Creates a JLabel with the specified icon.
  - JLabel(String text, Icon icon, int horizontalAlignment) Creates a JLabel instance with the specified text, image, and horizontal alignment
- Methods are generally not used.

Component with a Border

- A new feature all components inherit from JComponent is the ability to have various borders.
- The method is setBorder()
- A border argument is most easily created using the javax.swing class BorderFactory
- Here is an HTML label with a text labeled border.

```java
//Panel LP = new JPanel(new GridLayout(3,1));
String LT2="<html><FONT COLOR=RED>Red text, line break</FONT><FONT COLOR=WHITE> and white text</FONT></html>";
JLabel l3 = new JLabel(LT2, JLabel.CENTER);
l3.setBorder(BorderFactory.createTitledBorder("HTML Label"));
LP.add(l3);
```
**JButtons**

- Provides both text and graphic buttons
- Most methods come from the AbstractButton class
- AbstractButton also supports JMenuItem and JToggleButton
- The constructors are:
  - Button() Creates a button with no set text or icon.
  - JButton(Icon icon) Creates a button with an icon.
  - JButton(String text) Creates a button with text.
  - JButton(String text, Icon icon) Creates a button with initial text and an icon.
- An ActionListener is the appropriate event handler for a JButton.

**JTextField**

- Provides the basic capability to exchange textual information across the interface
- Inherits most of its methods from JTextComponent, along with JTextArea.
- Constructors include:
  - TextField() Constructs a new text field.
  - TextField(int columns) Constructs a new empty text field with the specified number of columns.
  - TextField(String text) Constructs a new text field initialized with the specified text.
  - TextField(String text, int columns) Constructs a new text field initialized with the specified text to be displayed, and wide enough to hold the specified number of columns.

**Selected JTextField Methods**

- Some of the more interesting methods include
  - void setText(String t) Sets the text that is presented by this text component to be the specified text.
  - void setHorizontalAlignment(int Alignment) Sets the alignment of the text in the text component.
  - String getSelectedText() Gets the selected text from the text that is presented by this text component.
  - void setEditable(boolean b) Sets the flag that determines whether or not this text component is editable.
**JTextArea**

- Provides a two dimensional text area.
- Inherits most methods from the JTextComponent.
- The constructors include:
  - TextArea() Constructs a new text area.
  - TextArea(int rows, int columns) Constructs a new empty text area with the specified number of rows and columns.
  - TextArea(String text, int rows, int columns) Constructs a new text area with the specified text and with the specified number of rows and columns.

**Selected JTextArea Methods**

- Methods include those available to JTextField.
- Added methods related to the multi-line nature include:
  - void append(String str) Appends the given text to the text area.
  - Dimension getPreferredSize() Determines the preferred size of this text area.
  - int getScrollbarVisibility() Gets an enumerated value that indicates which scrollbars the text area uses.
  - void replaceRange(String str, int start, int end) Replaces text between the indicated start and end positions with the specified replacement text.
  - void insert(String str, int pos) Inserts the specified text at the specified position in this text area.
  - void setColumns(int columns) Sets the number of columns for this text area.
  - void setRows(int rows) Sets the number of rows for the text area.

**JMenu**

- Menus can be placed in any container, including applets.
- Icons can be associated with any menu item.
- A menu is part of a menubar.
- A menu is made up of menu items.
- To create a JMenu, you must first create a JMenuBar.
  - You can then add JMenuItem's to the JMenu.
  - For JMenuItem, it will be necessary to add an action listener.
Layout Methods

- How objects are laid out is controlled by layout methods.
- Every container has an approach to how the components in the container should be laid out.
- The object that will be used most to control layout is the panel.
- A panel using one layout for its components can be placed in another panel that uses a different layout.
- By nesting objects in panels, a parent panel becomes possible to move and arrange sets of objects.
- The basic layout methods are flow layout and border layout.

FlowLayout

- The most basic -- simply fills the container with the added objects.
- Adds object left to right and top to bottom.
- Objects are center aligned in each row.
- When a row is filled, a new row is begun.
- The alignment can be changed to left or right.

BorderLayout

- Allows objects to be placed CENTER, EAST, WEST, NORTH and SOUTH.
- Is restricted to five components.
- You can specify a pixel gap between components.
- If a position for a component is not specified, it is not displayed.
- If more than one component is specified to a position, only the last is seen.
**GridLayout**
- allows a number of rows and columns to be specified
- optionally, the vertical and horizontal gap between components can also be specified
- the objects are laid out in sequence as added

**CardLayout**
- this layout manager is best conceptualized as a deck of cards
- only the top most card is visible
- each card is normally just a pane with components on it
- this would generally be replaced now by a tabbed pane

**GridBagLayout**
- this is the most flexible and complicated of all the layout managers — it builds on the grid layout
- objects can vary in size
- objects can be added in any order
- objects can occupy multiple rows or columns
- using a GridBagLayout requires the specification of GridBagConstraints
  - the x and y weight parameters of the components control growth
  - when the container grows, how much of the growth accrues to the component
  - if it is 0, the component does not participate in growth
New Layout Methods in Swing

- These methods and more on layout will be covered in course 2.
- ScrollPaneLayout -- built into the JScrollPane
- ViewportLayout -- built into the JViewport
- BoxLayout
- OverlayLayout

Events

- There are two event models in Java, one for 1.0, another for 1.1
  - The 1.0 model is sometimes still needed to write applets for 1.0 compliant browsers.
  - The 1.1 model is more mature and more filling with what we might expect.
  - The 1.1 events are contained in the package java.awt.event.
  - There are some additional event classes in java.Swing.

1.0 Event handler model

- Events are represented by the class Event.
- Events are sent first to the handleEvent method of the originating component. Events not handled by a component are passed to its parent.
- The event object has fields specifying the method such as:
  - id specifies event type (defined in the class)
  - target specifies the object that generated the event
  - x, y specify location data
- There are several event processing methods that may be defined for a component:
  - action()
  - keyUp() (keyDown())
  - mouseDown() (mouseUp())
  - mouseDrag()
- The top of the hierarchy handles all events.
The Java 1.1 event model

- The classes of events are made more specialized.
  - ActionEvent
  - AdjustmentEvent
  - ComponentEvent
  - ContainerEvent
  - KeyEvent
  - MouseEvent
  - TextEvent
  - WindowEvent

- A more conventional model of event handling is used.
  - Each time a certain event occurs, an event occurs, an event occurs.
  - When a target object receives an event, it
    - in order to get an event, a target object must implement the
    - appropriate interfaces.
  - ActionListener interface, example.
  - MouseMotionListener is another example.

Event Handlers Classes (Interfaces)

- An event handler is introduced to a class by implementing
  an interface.
  - class BFrame extends JFrame implements WindowListener

- The more common basic event listener interfaces include the
  following:
  - ActionListener
  - AdjustmentListener
  - ComponentListener
  - ContainerListener
  - FocusListener
  - ItemListener
  - KeyListener
  - MouseMotionListener
  - TextListener
  - WindowListener

- In Swing, there are more than 40 interfaces for specific
  types of events and behaviors.

The Methods of an Interface

- You have used two event handler interfaces.
  - WindowListener
  - ActionListener

- WindowListener has seven methods:
  - public void windowClosing(WindowEvent e) {}
  - public void windowClosed(WindowEvent e) {}
  - public void windowIconified(WindowEvent e) {}
  - public void windowOpened(WindowEvent e) {}
  - public void windowDeiconified(WindowEvent e) {}
  - public void windowActivated(WindowEvent e) {}
  - public void windowDeactivated(WindowEvent e) {}

- ActionListener has one method:
  - public void actionPerformed(ActionEvent e) {}
Adapters

- For interfaces with multiple methods, it may be the case that only one of the methods is really needed.
- Adapters provide a way to define a single method.
- Adapters provide null override methods.
- Only the needed method of an adapter is overridden.
- Assume a subclass of JFrame has been instantiated.
- Use the addWindowListener method.
- The argument to the addWindowListener method is a WindowAdapter defining the necessary method.

```java
addWindowListener(new WindowAdapter()
    public void windowClosing(WindowEvent evt)
    System.exit(0));
)
```

Exercise

- Use the base source code provided.
- Expand the code by adding text areas, menus, and other components.