Java Chapter 12 Part 1

* GUI Programming: Windows, Components, and Events
* CIS 255 • Shelby-Hoover Campus

GUI Programming

* The **Java Foundation Classes (JFC)** are used for creating GUI programs
  + The **Abstract Windowing Toolkit (AWT)** classes use peer classes in the operating system to draw elements on the screen; thus a program is limited to what is available on a specific OS, and behavior may vary
  + The **Swing** classes introduced in Java 2 draw most of their own elements on the screen; these offer consistent appearance and behavior across multiple operating systems, and they are customizable
* GUI programs are **event-driven**
  + An **event object** is generated when the user interacts with a GUI element in a particular manner
  + A program must have an **event listener** ready to respond to the creation of the event object
* Use wildcard import statements for the AWT and Swing packages
  + import javax.swing.\*;
  + import java.awt.\*;

Creating Windows

* Create a Java window using a JFrame object
  + A JFrame object is an example of a **container**, a GUI component that contains other GUI components
  + Containers that are displayable as windows are known as **frames** (they generally include borders, a title bar, and window controls)
* The window associated with a JFrame object is not visible by default
  + Set the title of the window using setTitle(titleString)
  + Set the size (width and height in pixels) of the window using setSize(width, height)
  + The method setDefaultCloseOperation(constant) accepts a value defining what happens when the window is closed
    - JFrame.EXIT\_ON\_CLOSE – exit the program
    - JFrame.HIDE\_ON\_CLOSE – hide the window, but do not exit
  + Visibility is set by calling setVisible(booleanValue)
* Example: ShowWindow.java (Code Listing 12-1)

Inheritance

* Cluttering the main method of a program with window setup methods can be annoying, especially if a window contains many components and complex event handling
* A program can package the method calls in another class by using **inheritance**
  + The header of the class should have an extends clause with the name of the class being extended (here, JFrame)
  + The new (derived) class inherits fields and operations from the original (base) class
  + The constructor for the derived class invokes the window setup methods by name only (the object being constructed is the implied “calling” object)
* When main instantiates the derived class, the constructor executes; thus, the program displays the window automatically
* Example: SimpleWindow.java (Code Listing 12-2) and SimpleWindowDemo.java   
  (Code Listing 12-3)

Embedded main Methods

* The object-oriented class definitions in chapter 6 separate the definition of each class from the program that instantiates it
* A main method can be included in an object-oriented class definition to allow the programmer to “test” the program without an external program
* If another program instantiating this class has its own main method, Java ignores the embedded main method (the external main has priority)
* Example: EmbeddedMain.java (Code Listing 12-4)

Adding GUI Components

* The textbook begins with three basic Swing components:
  + JLabel: displays a String of text (the argument to its constructor)
  + JTextField: accepts a single line of text input (the argument to its constructor is the field width, an int indicating the number of characters)
  + JButton: causes an action to be performed (the argument to its constructor is a String containing the button text)
  + These classes offer other constructors with various options
* After construction, the program must add these objects to the window object’s **content pane**
  + The content pane is a container within the window object
  + The add method adds a GUI component to a container
  + A program may organize components within a **panel** container (a JPanel object) that it then adds to the content pane
  + A class may set up various parts of a window using helper methods (usually private)

GUI Components Example

* The first phase of the KiloConverter class (Code Listing 12-5) contains private members for a panel, a label, a text field, and a button
* The constructor calls a helper method, buildPanel(), that constructs the components and adds them to the panel
* The constructor then adds the panel to the window
* The embedded main method instantiates the window class

Responding to Events

* When an event occurs (e.g., the user clicks a button), the **event source** object (e.g., the button) generates an **event object**
* One or more **event listener** objects may respond to this event
  + The event object serves as the argument to an event object parameter in one of the listener object’s methods
  + That method executes in response to the event object’s creation (**event firing**)

Event Listener Classes

* A programmer usually defines an event listener class as a private **inner class** nested within the class that requires the listener (available only to the outer class)
* Each listener class must implement an **interface** (a list of method headers required for a particular set of classes)
  + A class that implements an interface has an implements clause in its header
  + Interfaces for event listeners are provided in the package java.awt.event.\* (must be imported separately from the package java.awt.\*)
  + Example: To respond to a JButton click, the listener class should implement the ActionListener interface, which requires the method actionPerformed
* To associate a listener class with a GUI component, a program must **register** a new listener object with that GUI component; the name of the method to do this is usually “add” + the interface name)
* When the component generates an event object, the event listener’s method receives the event object and executes its code
* A program may contain multiple private inner classes

Event Listener Example

* The second phase of the KiloConverter class (Code Listing 12-6) adds another wildcard import statement for the event-related classes in the package java.awt.event
* The private inner listener class CalcButtonListener implements the interface ActionListener
  + Its definition is nested inside the KiloConverterWindow class definition, and it has access to all of the members of KiloConverterWindow (including kiloTextField)
  + As mentioned earlier, the method required by the ActionListener interface is actionPerformed, a public void method with a parameter of type ActionEvent
  + The program displays the result in a separate message dialog box
* Within the buildPanel method, the class registers a new listener object with the button:  
    
  calcButton.addActionListener(new CalcButtonListener());

Color

* Many GUI components have two colors that can be set: foreground (usually text) and background
* There are several predefined color constants that are static members of the class Color (included in java.awt.\*; see Table 12-3 or the Java APIs for a list of them)
* A program uses the methods setForeground and setBackground to set the initial color of an object and / or to change its color when an event occurs
* The background of the overall window can be set by invoking setBackground on the content pane using getContentPane():  
    
  getContentPane().setBackground(Color.BLUE);
* Example: ColorWindow.java (Code Listing 12-7)

Event Object Methods

* Event objects support different methods to determine information about the event
* The class ActionEvent includes the method getActionCommand() that returns a String representation of the **action command** (often, the text appearing in the GUI component)
* Many event object classes provide the method getSource() that returns a reference to the GUI component object that fired the event
* Listeners of a single class may be associated with several different components; thus, these methods isolate which GUI component initiated the event

getActionCommand() Example

* A single listener class could handle all three color buttons; the action command String from a JButton object is the button’s caption (the text on the button):  
    
  private class ButtonListener implements ActionListener  
  {  
   public void actionPerformed(ActionEvent e)  
   {  
   String ac = e.getActionCommand();  
    
   if (ac.equals("Red"))   
   {  
   panel.setBackground(Color.RED);  
   messageLabel.setForeground(Color.BLUE);  
   }  
   else if (ac.equals("Blue"))  
   {  
   panel.setBackground(Color.BLUE);  
   messageLabel.setForeground(Color.YELLOW);  
   }  
   else if (ac.equals("Yellow"))  
   {  
   panel.setBackground(Color.YELLOW);  
   messageLabel.setForeground(Color.BLACK);  
   }  
   }  
  }

getSource() Example

* This version of the listener class compares the value returned by getSource() to each of the three button objects:  
    
  private class ButtonListener implements ActionListener  
  {  
   public void actionPerformed(ActionEvent e)  
   {  
   if (e.getSource() == redButton)  
   {  
   panel.setBackground(Color.RED);  
   messageLabel.setForeground(Color.BLUE);  
   }  
   else if (e.getSource() == blueButton)  
   {  
   panel.setBackground(Color.BLUE);  
   messageLabel.setForeground(Color.YELLOW);  
   }  
   else if (e.getSource() == yellowButton)  
   {  
   panel.setBackground(Color.YELLOW);  
   messageLabel.setForeground(Color.BLACK);  
   }  
   }  
  }