CNT 4714: Enterprise Computing Spring 2010

Introduction To GUIs and Event-Driven Programming In Java – Part 5

Instructor :   Dr. Mark Llewellyn
                    markl@cs.ucf.edu
                    HEC 236, 407-823-2790
        http://www.cs.ucf.edu/courses/cnt471/spr2010

School of Electrical Engineering and Computer Science
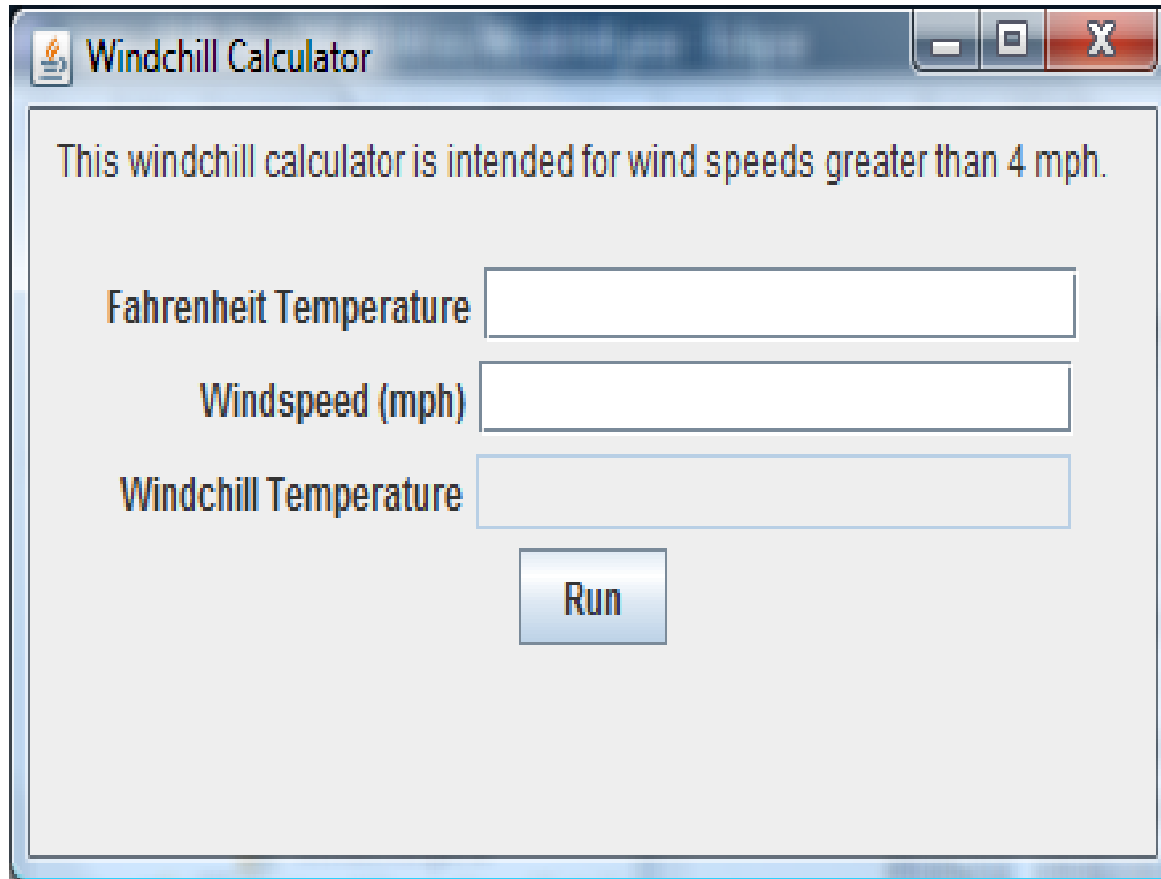University of Central Florida

A Sample GUI

- This final set of notes on GUIs and event-driven programming is devoted exclusively to developing a GUI-based event-driven program that calculates the windchill temperature for a user-specified temperature and windspeed.

  – Windchill is the temperature perceived by a person when taking into account the actual air temperature and the speed of the win. It is similar to a more popular term in Florida which is the heatindex that considers the actual air temperature and the humidity. You can use the GUI we develop this winter when you go skiing.

  – There are several different formulas available for calculating windchill. The one in our program is used by the U.S. National Weather Service and is only valid for windspeeds in excess of 4 mph.

What the GUI Should Look Like

Components of the GUI

- Compared to a console based application program, a GUI has many more objects to consider. A GUI program also has to deal with the interactions of its graphical components.

  - For example, *whenever* a user clicks the windchill calculator run button, the button *dispatches* a signal. The GUI must have a *listener* for that signal that causes the current temperature and windspeed data entry values to be obtained, the windchill to be calculated, and the result of that computation to be assigned to the windchill temperature entry area.

Swing API Classes in the WindChill Window

Title bar of JFrame instance. The JFrame contains 8 GUI elements

This line is a single JTextArea instance
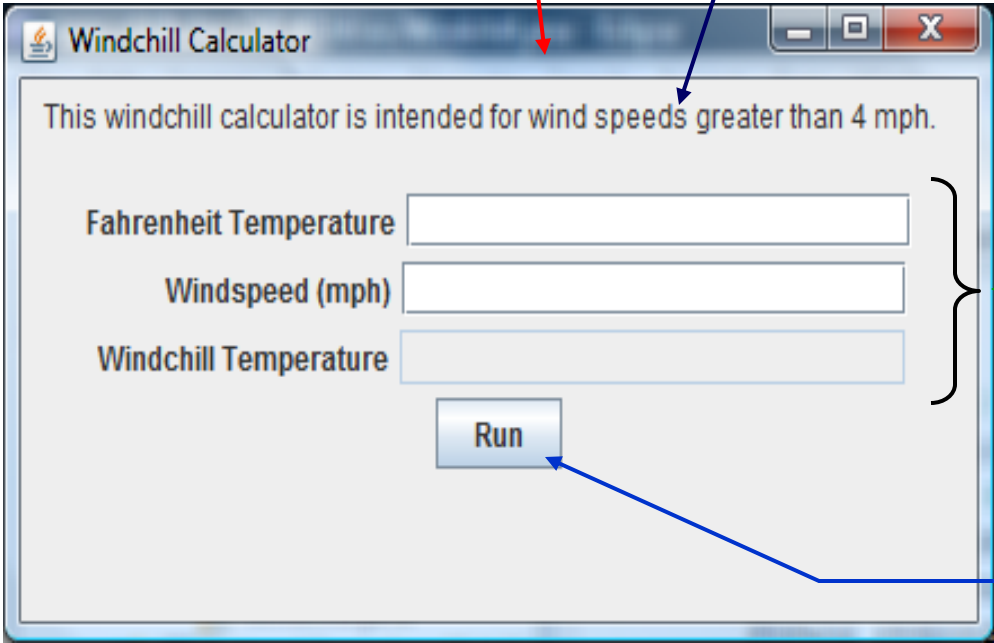
These three lines all contain a JLabel and a JTextField instance

JButton instance

Windchill Calculator

This windchill calculator is intended for wind speeds greater than 4 mph.

Fahrenheit Temperature

Windspeed (mph)

Windchill Temperature

Run

The Windchill GUI Program

```java
//windchill GUI
//Developed: January 2009
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class Windchill implements ActionListener{
    //class constants
    private static final int WINDOW_WIDTH = 410; //pixels
    private static final int WINDOW_HEIGHT = 235; //pixels
    private static final int FIELD_WIDTH = 20;   //characters
    private static final int AREA_WIDTH = 40;    //characters

    private   static   final   FlowLayout   LAYOUT_STYLE   =   new
    FlowLayout();

    private static final String LEGEND = "     This windchill
    "
        + "calculator is intended for wind speeds greater than
    4 mph.";
```

```
//instance variables
   //window for GUI
   private JFrame window = new JFrame("Windchill Calculator");


   //legend
   private JTextArea legendArea = new JTextArea(LEGEND, 2,
   AREA_WIDTH);


   //user entry area for temperature
   private JLabel fahrTag = new JLabel("Fahrenheit
   Temperature");
   private JTextField fahrText = new JTextField(FIELD_WIDTH);


   //user entry area for windspeed
   private JLabel windTag = new JLabel("         Windspeed
   (mph)");
   private JTextField windText = new JTextField(FIELD_WIDTH);


   //entry area for windchill result
   private JLabel chillTag = new JLabel(" Windchill
   Temperature");
   private JTextField chillText = new
   JTextField(FIELD_WIDTH);
```

```java
   //run button
 private JButton runButton = new JButton("Run");

  //Windchill():  constructor
  public Windchill() {
       //configure GUI
       window.setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
       window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

      legendArea.setEditable(false);
      legendArea.setLineWrap(true);
      legendArea.setWrapStyleWord(true);
      legendArea.setBackground(window.getBackground());

      chillText.setEditable(false);
      //chillText.setBackground(Color.WHITE);

      //register event listener
      runButton.addActionListener(this);
```

```java
//add components to the container
Container c = window.getContentPane();
c.setLayout(LAYOUT_STYLE);

c.add(legendArea);
c.add(fahrTag);
c.add(fahrText);
c.add(windTag);
c.add(windText);
c.add(chillTag);
c.add(chillText);
c.add(runButton);

//display GUI
window.setVisible(true);
}
```

```java
//actionPerformed(): run button action event handler
   public void actionPerformed(ActionEvent e) {
      //get user's responses
      String response1 = fahrText.getText();
      double t = Double.parseDouble(response1);
      String response2 = windText.getText();
      double v = Double.parseDouble(response2);
      //compute windchill
      double windchillTemperature = 0.081 * (t - 91.4) * (3.71 *
   Math.sqrt(v) + 5.81 - 0.25*v) + 91.4;
          int perceivedTemperature =
   (int)Math.round(windchillTemperature);

      //display windchill
      String output = String.valueOf(perceivedTemperature);
      chillText.setText(output);
   }

//main():  application entry point
   public static void main(String[] args) {
      Windchill gui = new Windchill();
   }
}
```

Windchill GUI Object Attributes & Instance Variables

- A Windchill GUI object has nine attributes and therefore nine instance variables. These variables are:

  - **window:** references a JFrame representing the window containing the other components of the GUI;

  - **legendArea:** references a JTextArea representing the multiline program legend. In this case it is a single line legend.

  - **fahrTag:** references a JLabel representing the label for the data entry area supplying the temperature.

  - **fahrText:** references a JTextField representing the data entry area supplying the temperature.

  - **windTag:** references a JLabel representing the label for the data entry area supplying the windspeed.

  - **windText:** references a JTextField representing the data entry area supplying the windspeed.

  - **chillTag:** references a JLabel representing the label for the data entry area giving the windspeed.

  - **chillText:** references a JTextField representing the data entry area giving the windspeed.

  - **runButton:** references a JButton representing the button that singals a windchill calculation request.

Windchill GUI Object Class Constants

- In addition to the instance variables, the Windchill class also defines the following class constants. Class constants are common values to which all objects of the class share access.

    - **WINDOW_WIDTH:** an int value giving the initial width of a GUI;

    - **WINDOW-HEIGHT:** an int value giving the initial height of a GUI;

    - **AREA_WIDTH:** an int value giving the width of a program legend;

    - **FIELD_WIDTH:** an int value giving the width of a data entry area;

    - **LEGEND:** reference to the String representation of a program legend;

    - **LAYOUT_STYLE:** reference to a FlowLayout that manages the layout of the GUI components within the window. In particular, a FlowLayout manager arranges the GUI components in a top-to-bottom, left-to-right manner in the order that they are added to the window;

Windchill GUI

- The definition of program `WindChill.java` is markedly different from most of the application programs that we have seen thus far in the course. The differences are apparent from the start of the Windchill class definition.

```
public class Windchill implements ActionListener{
```

The keyword indicates that the class will implement some interface specfications

ActionListener requires a method actionPerformed() be implemented for handling GUI action events
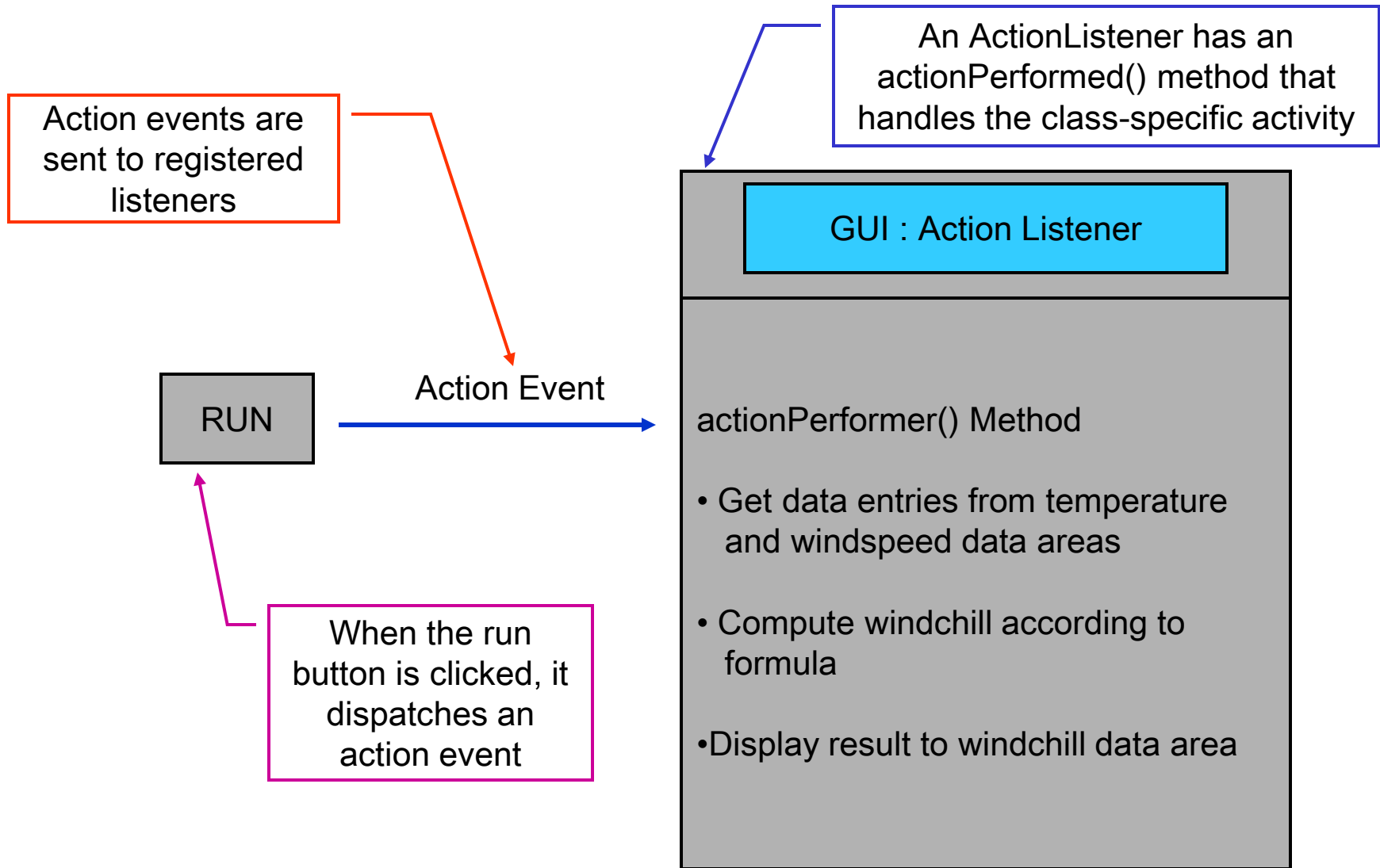
Windchill GUI

- The keyword `implements` indicates that the class definition satisfies the specification of the *interfaces* that follow the keyword.   Informally, an interface is a template describing the features of a class. Java requires that action performers for GUI events implement the `ActionListener` interface.

- Most event are handled directly by the GUI component with which the user interacted (e.g., a JTextField object handles the entering and editing of data in its textbox).

- However, an application-specific response is needed for a run-button event – the event must initiate the computing and displaying of the windchill (in our case). To define its response for that event, Windchill implements the ActionListener interface.

Run Button Action-Event Processing

Action events are sent to registered listeners

An ActionListener has an actionPerformed() method that handles the class-specific activity

GUI : Action Listener

RUN

Action Event

When the run button is clicked, it dispatches an action event

actionPerformer() Method

• Get data entries from temperature and windspeed data areas

• Compute windchill according to formula

•Display result to windchill data area

Windchill Class

- The Windchill class definition has four sections.

  - The first section specifies a collection of private class constants and instance variables that are used elsewhere in the definition.

  - The second section defines the Windchill default constructor. The constructor configures the instance variable GUI components so that they are ready to perform the windchill computation upon the request of the user.

  - The third section defines the event handler method `actionPerformed()`. Implementing this event handling method is the only requirement of the `ActionListener` interface. The interface requires the method have the form:

    `public void actionPerformned(ActionEvent e)`

    where class `ActionEvent` is `part of the standard java.awt.event`. The ActionEvent class is the basis for representing all `swing` windowing events.

Windchill Class

– The fourth section defines the main method, the application's entry point. With GUI-based programs, the main method is often trivial to implement. For example, in this program it defines only a new instance of the class's GUI.
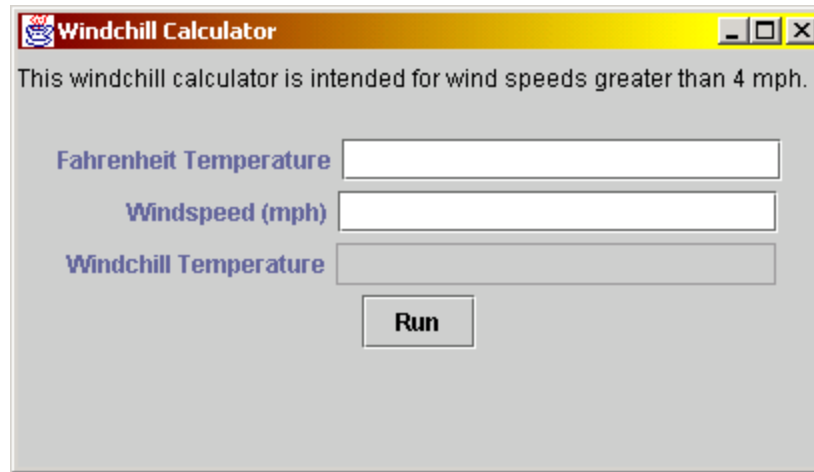
```
Windchill gui = new Windchill();
```

Class Constants and Instance Variables

- The class constants and instance variables section of the Windchill class begins with the definitions of 6 constants. These constants are used in configuring the various components of the Windchill GUI.

  - **Remember:** You can tell these definitions are specifying class constants since they use the **final** and **static** modifiers.

- Constants `WINDOW_WIDTH` and `WINDOW_HEIGHT` are the initial dimensions of the GUI.

- Constant `FIELD_WIDTH` is the width of the text boxes used for the inputs and outputs of the computation performed by the GUI.

- Constant `AREA_WIDTH` is the width of the text box for displaying the Windchill GUI legend at the top of the GUI.
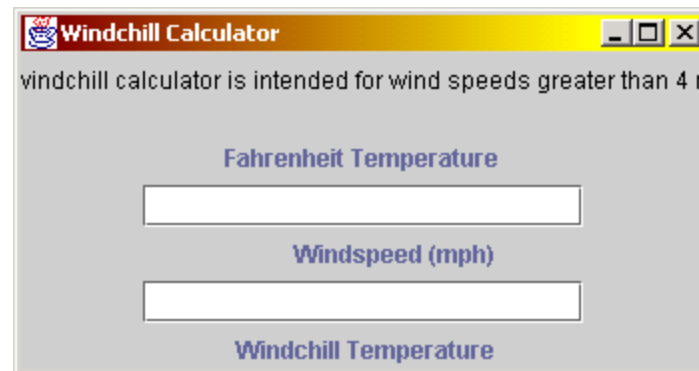
```
private static final int WINDOW_WIDTH = 410; //pixels

private static final int WINDOW_HEIGHT = 235; //pixels
private static final int FIELD_WIDTH = 20;   //characters
private static final int AREA_WIDTH = 40;    //characters
```

Illustration of Changing Window Parameters
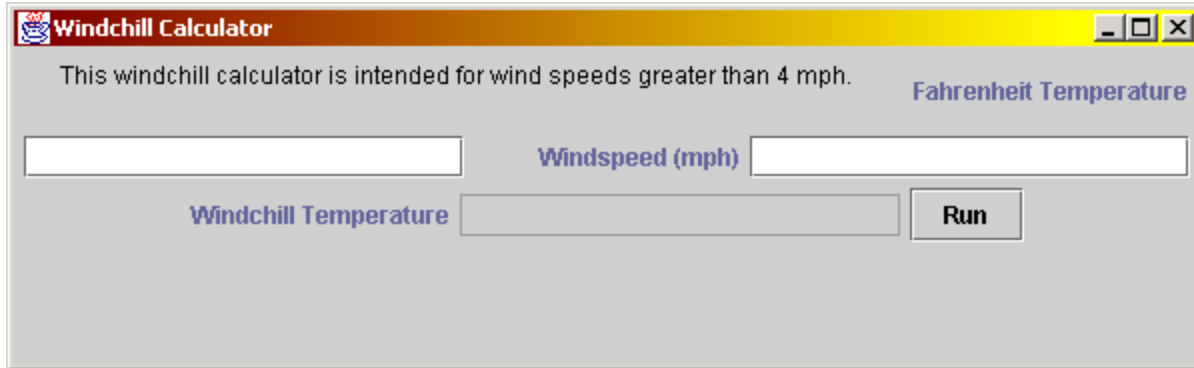
WINDOW_WIDTH = 410, WINDOW_HEIGHT = 235

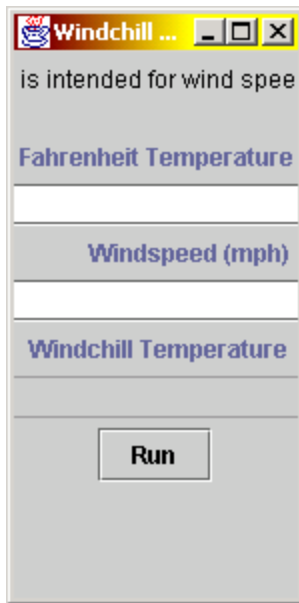Notice that the window is not large enough now to fit the legend in one line nor to see the output or the run button!

WINDOW_WIDTH = 350, WINDOW_HEIGHT = 185

Illustration of Changing Window Parameters

WINDOW_WIDTH = 600, WINDOW_HEIGHT = 185

Notice that increasing the width of the JFrame causes the GUI components to flow upward to fill the available space

Notice that the window is not wide enough to even completely display the input text boxes nor the entire legend.

WINDOW_WIDTH = 150
WINDOW_HEIGHT = 300
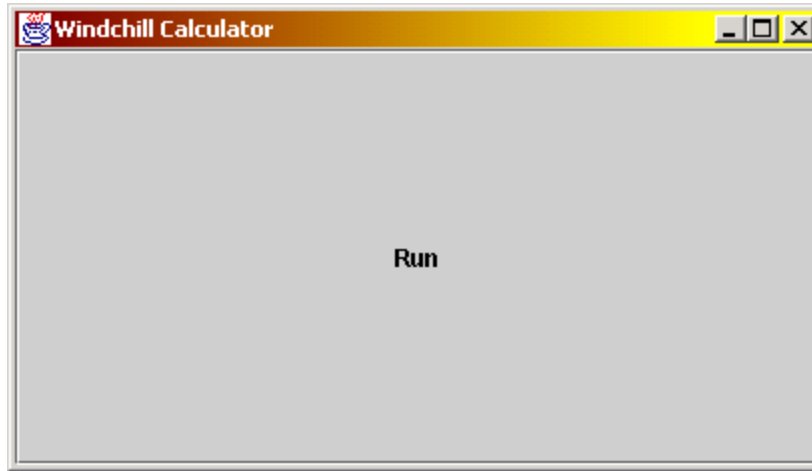
Class Constants and Instance Variables (cont.)

- The `FlowLayout` constant `LAYOUT_STYLE` describes how the components of the GUI are to be arranged in the window. In particular, a `FlowLayout` manager arranges GUI components in a top-to-bottom, left-to-right arrangement in the order in which they are added to the window.

```
private static final FlowLayout LAYOUT_STYLE =
    new FlowLayout();
```

- If a window does not specify a layout manager, then the last component added to the window occupies the entire window. The next slide illustrates what our Windchill GUI would look like without the specification of a layout manager.

Illustration of Changing Window Parameters

WINDOW_WIDTH = 410, WINDOW_HEIGHT = 235
No Layout Manager Specified

Class Constants and Instance Variables (cont.)

- The last constant definition is for String constant LEGEND representing the text of the program legend.

```
private static final String LEGEND = "     This windchill "
        + "calculator is intended for wind speeds greater "
        + "than 4 mph.";
```

- Following the class constants come the instance variable definitions. These definitions initialize the instance variables for each new Windchill GUI object. Each Windchill GUI object has its own copy of the instance variables.

Class Constants and Instance Variables (cont.)

- The first instance variable is the `JFrame` variable `window`. A `JFrame` acts as a **container** that holds the components of the GUI. A `JFrame` is similar in form to the other windows on your desktop (e.g., it has a frame and a title bar) and can be manipulated like other windows (e.g., minimized, maximized, and moved).

- Variable `window` references a new `JFrame` window object. The `JFrame` constructor titles the new window using its `String` parameter "Windchill Calculator".

```
//window for GUI
   private JFrame window = new JFrame("Windchill Calculator");
```

Class Constants and Instance Variables (cont.)

- The second instance variable is the `JTextArea` variable `legendArea`. It references a new `JTextArea` object that acts as a multiline text box (in our case only a single line). The `JTextArea` constructor creating the object takes three parameters.

```
//legend
    private JTextArea legendArea =
new JTextArea(LEGEND, 2, AREA_WIDTH);
```

- The first parameter is the string to be displayed in its text box, which in this case is the String referenced by LEGEND.

- The second and third parameters are the dimensions of the new text box – the number of lines and the number of characters per line. In this case two lines can be displayed each with AREA_WIDTH number of characters per line

Class Constants and Instance Variables (cont.)

- There is a pair of instance variables associated with each of the following: the input temperature, the input windspeed, and the windchill output. Each pair defines two new objects – a `JLabel` and a `JTextField`. The label clues the user as to what information is needed or supplied by the GUI, and the text entry area serves as the conduit between the user and the program.

- For example,

```
//user entry area for temperature
    private JLabel fahrTag = new JLabel("Fahrenheit Temperature");
    private JTextField fahrText = new JTextField(FIELD_WIDTH);
```

Fahrenheit Temperature

A JLabel is noneditable by the user

By default the text field of a JTextField is editable by the user.

Alignment of the Label and Text Entry Areas

- In order to align the three label and text entry pairs one after the other in the window, blanks are used to make the windspeed and windchill labels the same length as the temperature label.

  - On systems where the default label and entry area font is a monospaced font (i.e., all characters have the same width like Courier) this is fairly easy to do by simply counting characters.

  - On systems where the default label and entry area font is nonmonospaced (the more common situation), determining the number of padding blanks necessary to align labels can be a trial and error process.

The `runButton` Instance Variable

- The last of the instance variables in our Windchill GUI is the `runButton`.

- The `JButton` constructor creating the object expects a single parameter that specifies the button's label. In this case, the new button has the label "RUN".

- With these class constants and instance variables, the Windchill default constructor configures and displays the GUI so that whenever its run button is clicked, the `actionPerformed()` method first accesses the data entry areas that are referenced by `fahrText` and `windText` so that it can compute the associated windchill. Method `actionPerformed()` then displays the windchill in the entry area referenced by `chillText`.

Construction of the GUI

- When a constructor begins execution, it configures, as necessary, the newly initialized copies of the instance variables for the object under construction. For the Windchill GUI, all nine instance variables require manipulation by the constructor.

- The Windchill constructor begins by sizing the window that will hold the GUI. For this purpose, the constructor signals the window through `JFrame` instance method `setSize()`.   Method `setSize()` expects two parameters giving the width and height of the new window in pixels.

```
window.setSize(WINDOW_WIDTH, WINDOW_HEIGHT);
```

Construction of the GUI (cont.)

- Next, the constructor configures the program to terminate when the window closes. This is done by the constructor invoking the `JFrame` instance method `setDefaultCloseOperation()`.

```
window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

- The parameter `JFrame.EXIT_ON_CLOSE` is a `JFrame` class constant whose value indicates that the program is to be terminated when this `JFrame` is closed. Note: `JFrame.EXIT_ON_CLOSE` should only be used in applications not applets.

  – The default case is to `HIDE_ON_CLOSE`. Also available are `DO_NOTHING_ON_CLOSE` and `DISPOSE_ON_CLOSE` all of which are inherited through the interface `WindowConstants`.

Construction of the GUI (cont.)

- Next, the constructor configures the legend for the GUI. Variable `legendArea` is associated with the `JTextArea` object that holds the program's legend.

```
legendArea.setEditable(false);
```
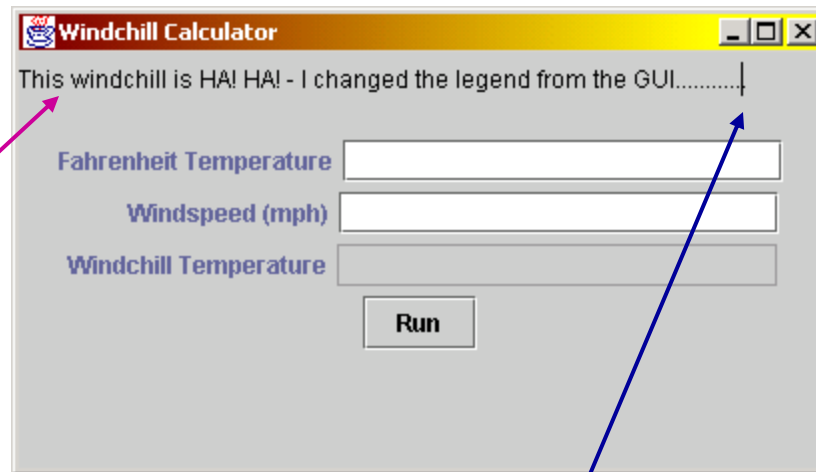
- Since the user should not be able to modify the legend, the associated JTextArea object is signaled through its instance method setEditable() to make its text field *noneditable*.

- Method `setEditable()` expects a single boolean value as its parameter. If the parameter value is false, then the associated `JTextArea` is noneditable. If the parameter value is true, then the associated `JTextArea` is editable.

Construction of the GUI (cont.)

- The reason the legend area should be uneditable is shown in the figure below.

The user can edit the legend area just like any other data area if setEditable(true)

Windchill Calculator

This windchill is HA! HA! - I changed the legend from the GUI..........

Fahrenheit Temperature

Windspeed (mph)

Windchill Temperature

Run

Notice the cursor bar since the field is editable

Construction of the GUI (cont.)

- The additional configuring done with the `legendArea` ensures that the legend is displayed properly within its entry area. By default, a `JTextArea` object does not wrap its text. To signal that a wrapping is desired, its method `setLineWrap()` is invoked.

```
legendArea.setLineWrap(true);
```

- Method `setLineWrap()` expects a single boolean value as its parameter. A value of false indicates that its text is not to be wrapped; a value of true indicates that its text is to be wrapped.

Construction of the GUI (cont.)

- Simply requesting line wrapping is insufficient. The default line wrapping style is to break up text only when a text-field line is completely full. This style can cause a word to split over two lines.

- To ensure line wrapping occurs only a word boundaries, a `JTextArea` object is signaled through its method `setWrapStyleWord()`. This is illustrated in the next slide.

```
legendArea.setWrapStyleWord(true);
```

- Method `setWrapStyleWord()` expects a single boolean value as its parameter. A value of true indicates that line wrapping is to occur only a word boundaries; false will wrap the text whenever a text-field line is full.
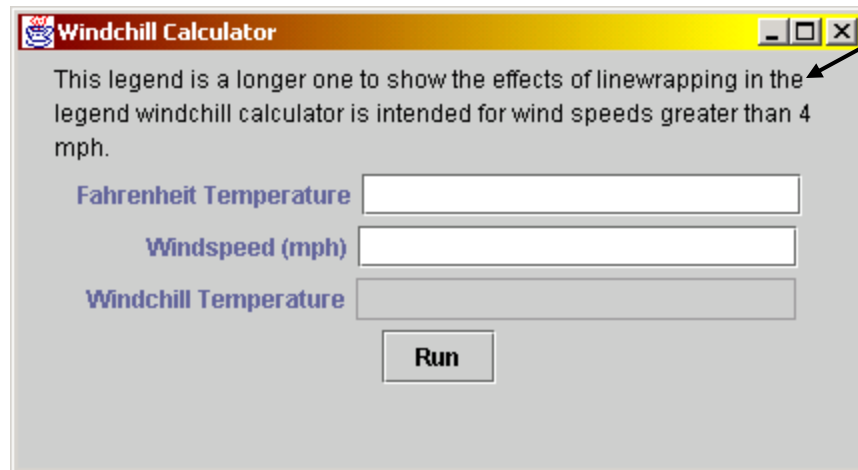
Construction of the GUI (cont.)
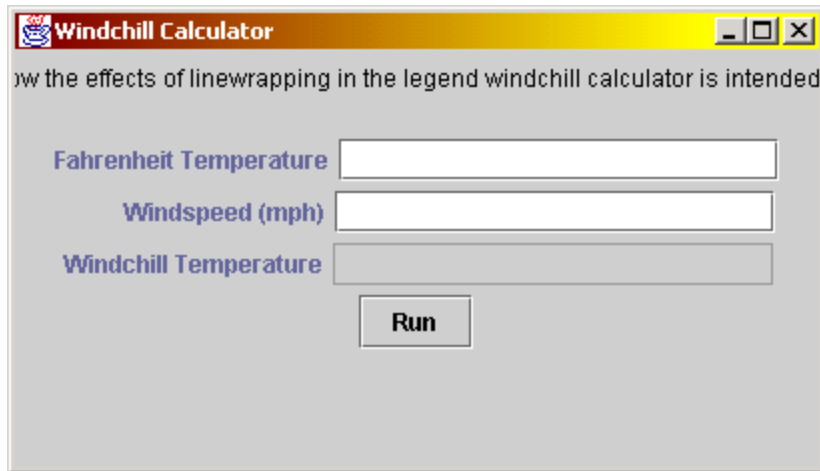
In this case setLineWrap(false) causes the too long legend to disappear off both ends of the window.

In this case setLineWrap(true) allows the legend to wrap into a second line, but setWrapStyle(false) allows the wrapping to occur at whatever point the textfield becomes full and not necessarily at a word boundary.
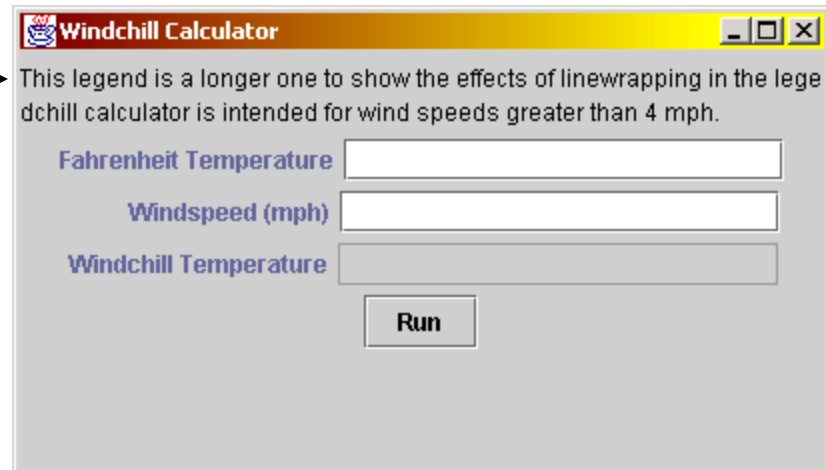
Construction of the GUI (cont.)

In this case setLineWrap(true) and setWrapStyleWord(true) causes the too long legend to wrap into 3 separate lines of legend with breaks only a word boundaries.

Construction of the GUI (cont.)

- The background color of the legend area can be altered to match the background color of the window (the default background color of a `JTextArea` is white). To signal a `JTextArea` background color change, its method `setBackground()` is invoked with a parameter specifying the desired color.

- To match the two background colors, the color of the `JFrame` is obtained using its instance method `getBackground()`.

```
legendArea.setBackground(window.getBackground());
```

Construction of the GUI (cont.)

- The `JTextField` associated with `chillText` should be made noneditable – it is the program that supplies the value not the user.

```
chillText.setEditable(false);
```

- The prohibition regarding the editing of the text field applies only to the user and not to the object itself.

- Making it noneditable, causes Java to change its background color from the standard `JTextField` background color (remember the default color is white). To override the color change, `JTextField` method `setBackground()` is invoked.
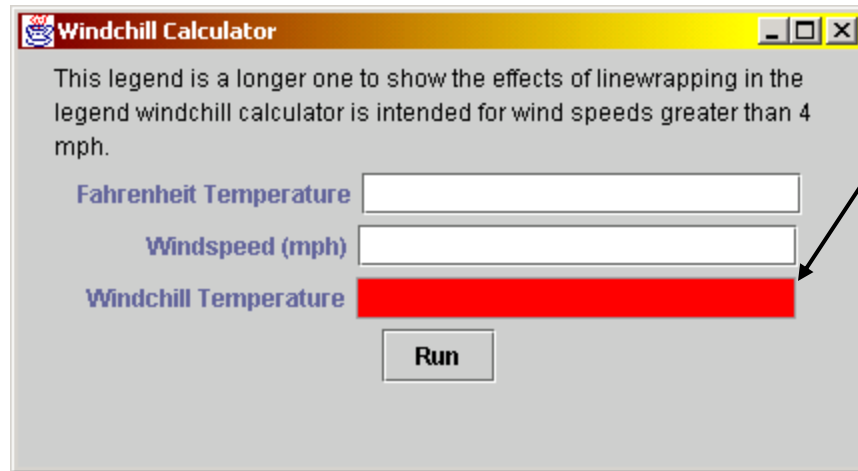
```
//chillText.setBackground(Color.blue);
```

Construction of the GUI (cont.)
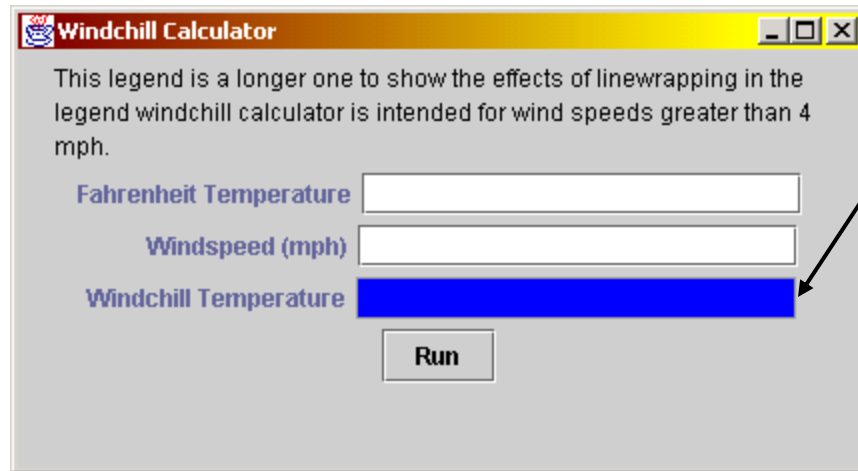
chilltext.setBackground(Color.blue) causes the text box for the result to appear in a nonstandard color.

chilltext.setBackground(Color.red) causes the text box for the result to appear in a nonstandard color.

Construction of the GUI (cont.)

- The standard colors available in Java for painting text boxes and objects can be found in the `java.awt.Color` class.

- The colors are: black, blue, cyan, darkGray, gray, green, lightGray, magenta, orange, pink, red, white, and yellow.

- Opaque and transparent shading is also possible.

- There are also constructors in this class to create any color within the RGB scheme of 0-255.

Construction of the GUI (cont.)

- Next step for the constructor is to set the action performer for a `JButton` using `JButton` instance method `addActionListener()`.

```
runButton.addActionListener(this);
```

- The parameter to `addActionListener()` specifies the object whose method `actionPerformed()` processes the clicking of the run button. The object in question is the GUI under construction, i.e., the object currently being configured by the Windchill default constructor method. The parameter to `addActionListener()` is **this**. (Remember that the keyword this is the Java technique for referencing the object being manipulated by a constructor or an instance method.)

Construction of the GUI (cont.)

- It may seem a bit confusing, but the GUI components are not added directly to the `JFrame`. Instead they are added to a container inside that frame. As its name implies, a `JFrame` is a frame. The frame includes the title bar and border edging that can be manipulated like other windows under the operating system.

- Inside the perimeter of the frame is a content pane. The content pane is the container that directly holds the other components of the GUI. The type of the content pane is `awt` class `Container`.

- To gain access to the content pane, the Windchill constructor uses `JFrame` method `getContentPane()` to initialize `Container` variable `c`.

```
Container c = window.getContentPane();
```

Construction of the GUI (cont.)

- With variable `c`, the layout manager can be set for the content pane using `Container` method `setLayout()`.

```
c.setLayout(LAYOUT_STYLE);
```

- To complete the configuration of the GUI, the eight GUI components are added to the content pane using `Container` method `add()`.

```
c.add(legendArea);
c.add(fahrTag);
c.add(fahrText);
c.add(windTag);
c.add(windText);
c.add(chillTag);
c.add(chillText);
c.add(runButton);
```

Construction of the GUI (cont.)

- Once the configuration is complete, it is appropriate to make the window *visible*.

- Up to this point, the Windchill constructor has set up the graphical components that make up its interface, but it has not displayed them. To do so, the constructor uses `JFrame` method `setVisible()`. (Inherited from the `java.awt.Component` class.)

```
window.setVisible(true);
```

- It is also possible to use the `Window` class instance method `show()`. Which also makes a window visible.

```
window.show();
```

Construction of the GUI (cont.)

- By default, JTextArea, JLabel, JTextField, and JButton instances are visible once the window in which they have been placed is made visible.

- Thus, setting their visibility individually is not necessary, although you have the capability of turning them on only after certain conditions have occurred, if so desired.

- The GUI is now displayed as the constructor completes.

- Before we look at event handling and actionPerformed(), we'll see the final configuration of our GUI once the constructor has completed.

The Final GUI
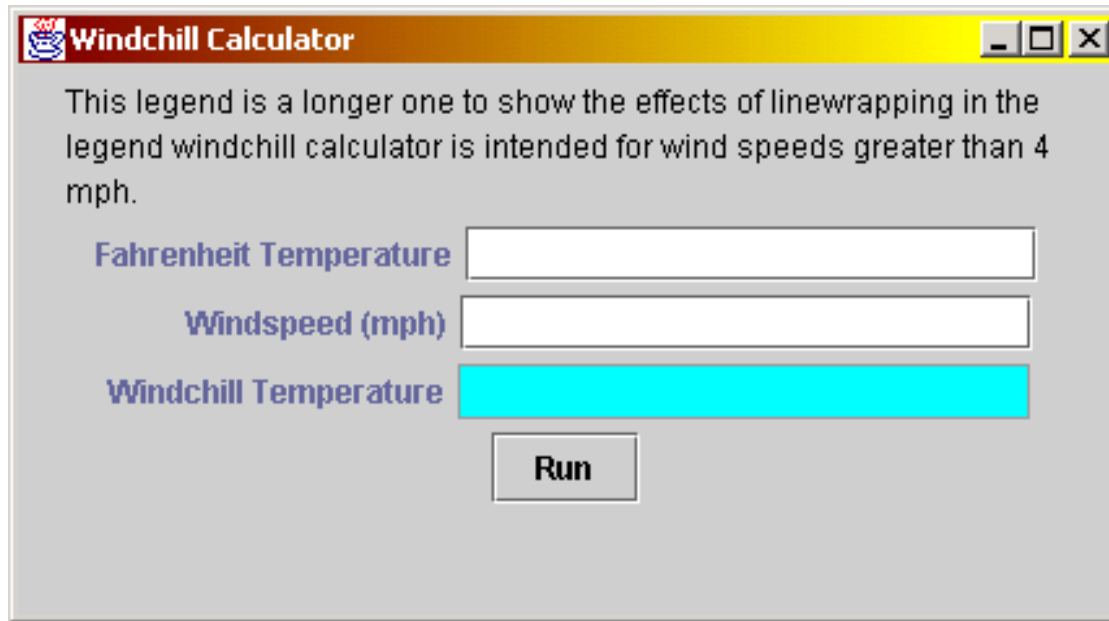
Event Handling and `actionPerformed()`

- Implementing the button action performer method `actionPerformed()`. is relatively straightforward.

  - When invoked in response to the user selecting the run button, the performer first gets the user inputs from the text fields associated with `JTextField` variables `fahrText` and `windText`.

  - Method `actionPerformed()` uses the two values to compute the associated windchill. The windchill value is then used to set the text field associated with the variable `chillText`.

- A `JTextField` has a method `getText()` that returns a copy of the text in its text field in `String` form. The `String` representation can be converted to a numeric representation using `Double` class method `parseDouble()`.

```
String response1 = fahrText.getText();
double t = Double.parseDouble(response1);
```

Event Handling and `actionPerformed()`(cont.)

- The action performer uses a similar code segment to initialize a double variable `v` to represent the windspeed.

```
String response2 = windText.getText();
double v = Double.parseDouble(response2);
```

- Given `v` and `t`, a double variable `windchillTemperature` can be defined and properly initialized. To translate the windchill formula:

$$t_{wc} = 0.081(t - 91.4)(3.71\sqrt{v} + 5.81 - 0.25v) + 91.4$$

into a valid Java expression, the API Math provides a class method `sqrt()` for calculating square root values.

```
double windchillTemperature = 0.081 * (1 - 91.4)
    * (3.71 * Math.sqrt(v) + 5.81 - 0.25*v) + 91.4;
```

Event Handling and `actionPerformed()`(cont.)

- Since the variable `windchillTemperature` is a double, its value can add a significant number of digits after the decimal point. These digits are uninteresting to most users. Therefore, the action performer uses the `Math` class method `round()` to produce the integer values closest to the original value.

  - Method `round()` returns a `long` value, so the cast is necessary to convert that value into a `int` value as Java does not implicitly narrow a `long` value to an `int`.

```
int perceivedTemperature =
                (int)Math.round(windchillTemperature);
```

Event Handling and `actionPerformed()`(cont.)

- To display the windchill, the action performer converts the `int` value to a `String` representation using the `String` class method `valueOf()`. The method expects a single `int` value as its parameter and returns a `String` version of the number.

> **String output = String.valueOf(perceivedTemperature);**

- The action performer uses that string value as a parameter to JTextField method setText(). The method updates the text box associated with variable chillText. In particular, by invoking:

> **chillText.setText(output);**

the correct windchill output is displayed.

- The displaying of the windchill computation finishes the event handling for the button-clicking event. The program then continues with the even dispatching loop until another action event occurs or the program is ended.

Method `Main()`

- Method `main()` of the Windchill program is trivial to implement. The method just defines a new instance of Windchill.

```
public static void main(String[] args) {
        Windchill gui = new Windchill();
    }
```

- No other work is require, because the constructor handles the building and displaying of the GUI and the actionPerformed() method handles the user interaction.

- If desired, method `main()` can be modified to create multiple Windchill GUIs. Each of the windchill calculators would be displayed simultaneously. This is illustrated in the next slide.

```
public static void main(String[] args) {
      Windchill gui1 = new Windchill();
      Windchill gui2 = new Windchill();
   }
```
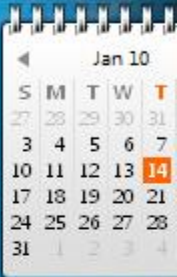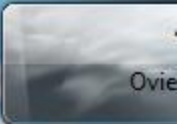
Conclusion

- With this set of notes you now have enough information to construct your own GUIs.

- One of the difficulties of working with a language like Java is knowing what sort of classes and methods are available for you to use in your programs without needing to write them yourself. The simplest way to do this is to become familiar with the language. The following website will help immensely:

  http://java.sun.com