```
/*******************************************************************************
 * This class implements un UpperAirData class that stores temperature data for pressure
 * levels and processes them to calculate index values to determine the possibility
 * of severe weather
 *
 * The class has some error checking features to check if the data is valid and give some
 * useful error message is it is not
 *
 * The class has a toString() method to generate a report that can be used to display
 * the weather risk assessment
 *
 * CST 183 Programming Assignment 6
 * @author Michael Clinesmith
 *
 ******************************************************************************/

public class UpperAirData
{
    private int T850, T700, T500, Td850, Td700;         // temperature fields at various pressures

    /**
     * Constructor with no parameters
     */
    public UpperAirData()
    {
        T850 = 0;
        T700 = 0;
        T500 = 0;
        Td850 = 0;
        Td700 = 0;
    }

    /**
     * Constructor when given integer parameters
     * temperatures are to be in Celsius
     * @param v850 int containing temperature at 850 mb
     * @param v700 int containing temperature at 700 mb
     * @param v500 int containing temperature at 500 mb
     * @param vd850 int containing dew point at 850 mb
     * @param vd700 int containing dew point at 700 mb
     */
    public UpperAirData(int v850, int v700, int v500, int vd850, int vd700)
    {
        T850 = v850;
        T700 = v700;
        T500 = v500;
        Td850 = vd850;
        Td700 = vd700;
    }

    /**
     * Constructor when given integer parameters
     * temperatures are to be in Celsius
     * @param s850 String containing temperature at 850 mb
     * @param s700 String containing temperature at 700 mb
     * @param s500 String containing temperature at 500 mb
     * @param sd850 String containing dew point at 850 mb
     * @param sd700 String containing dew point at 700 mb
     */
    public UpperAirData(String s850, String s700, String s500, String sd850, String sd700)
```

30/30 points for Program 6

Awesome solution! Graphics were way-cool

Interface was above-and-beyond..

```
{
    T850 = Integer.parseInt(s850);
    T700 = Integer.parseInt(s700);
    T500 = Integer.parseInt(s500);
    Td850 = Integer.parseInt(sd850);
    Td700 = Integer.parseInt(sd700);
}

/**
 * Mutator method to temperature value at 850 mb
 * @param t850 int containing temperature at 850 mb
 */
public void setT850(int t850)
{
    T850 = t850;
}

/**
 * Mutator method to temperature value at 700 mb
 * @param t700 int containing temperature at 700 mb
 */
public void setT700(int t700)
{
    T700 = t700;
}

/**
 * Mutator method to temperature value at 500 mb
 * @param t500 int containing temperature at 500 mb
 */
public void setT500(int t500)
{
    T500 = t500;
}

/**
 * Mutator method to dew point value at 850 mb
 * @param td850 int containing dew point at 850 mb
 */
public void setTd850(int td850)
{
    Td850 = td850;
}

/**
 * Mutator method to dew point value at 700 mb
 * @param td700 int containing dew point at 700 mb
 */
public void setTd700(int td700)
{
    Td700 = td700;
}

/**
 * Accessor method to get temperature at 850 mb
 * @return int containing temperature at 850 mb
 */
public int getT850()
{
    return T850;
```

```
}

/**
 * Accessor method to get temperature at 700 mb
 * @return int containing temperature at 700 mb
 */
public int getT700()
{
    return T700;
}

/**
 * Accessor method to get temperature at 500 mb
 * @return int containing temperature at 500 mb
 */
public int getT500()
{
    return T500;
}

/**
 * Accessor method to get dew point at 850 mb
 * @return int containing dew point at 850 mb
 */
public int getTd850()
{
    return Td850;
}

/**
 * Accessor method to get dew point at 700 mb
 * @return int containing dew point at 700 mb
 */
public int getTd700()
{
    return Td700;
}

/**
 * This method takes the values stored in the fields and generates a report in string form
 * including index values and risk assessments
 *
 * @return String containing the report
 */
@Override
public String toString()
{
    String outString;

    // store field values
    outString = "T850: " + T850 +
            " C\nT700: " + T700 +
            " C\nT500: " + T500 +
            " C\nTd850: " + Td850 +
            " C\nTd700: " + Td700 + " C\n";

    if(!isValid())  // if not valid store error message
    {
        outString += "\n\n" + invalidMessage();
    }
```

```
    else            // if valid store index report summary
    {
        outString += "\nTTIndex: " + calculateTTIndex() +
                "\nTTIndex Report: " + TTIndexMessage() +
                "\n\nKIndex: " + calculateKIndex() +
                "\nKIndex Report: " + KIndexMessage();
    }

return outString;
}


/**
 * Method to calculate TTIndex
 * @return int containing the TTIndex value
 */
public int calculateTTIndex()
{
    return T850 + Td850 - 2 * T500;
}

/**
 * Method to calculate KIndex
 * @return int containing the KIndex value
 */
public int calculateKIndex()
{
    return T850 + Td850 - T500 - (T700 - Td700);
}

/**
 * The method checks the field data and returns a code value based on what part of the
 * data is invalid
 * Codes:
 * 0                data is valid
 * 1                T850 is outside of the valid range of -40 to 40
 * 2                T700 is outside of the valid range of -60 to 10
 * 3                T500 is outside of the valid range of -50 to 0
 * 4                Td850 is outside of the valid range of -40 to 40
 * 5                Td700 is outside of the valid range of -60 to 10
 * 6                Td850 is greater than T850
 * 7                Td700 is greater than T700
 *
 *  If there are multiple errors, it returns a code representing the first error found
 * @return  int a code representing if the object has valid data or invalid data
 */
public int invalidCode()
{
    int code=0;

    if ( T850 > 40 || T850 < -40)
    {
        code = 1;
    }
    else if (T700 > 10 || T700 < -60)
    {
        code = 2;
    }
    else if (T500 > 0 || T500 < -50)
    {
```

```
        code = 3;
    }
    else if (Td850 > 40 || Td850 < -40)
    {
        code = 4;
    }
    else if (Td700 > 10 || Td700 < -60)
    {
        code = 5;
    }
    else if (Td850 > T850)
    {
        code = 6;
    }
    else if (Td700 > T700)
    {
        code = 7;
    }

    return code;
}

/**
 * The method returns a value indicating if the data stored is valid or invalid
 * It calls the invalidCode method to do the check
 * @return boolean - true if data is valid, false if not
 */
public boolean isValid()
{
    boolean valid = true;

    if( invalidCode() !=0)
    {
        valid = false;
    }

    return valid;
}

/**
 * This method returns an error message based on the error code
 * @return String, a informative message indicating a problem with field data
 */
public String invalidMessage()
{
    int code = invalidCode();  // call to invalidCode method to find location of invalid data
    String errorMessage;

    switch (code)
    {
        case 0:
            errorMessage = "No error with weather data.";
            break;
        case 1:
            errorMessage = "T850 is outside the valid range of -40 to 40.";
            break;
        case 2:
            errorMessage = "T700 is outside the valid range of -60 to 10.";
            break;
        case 3:
```

```
            errorMessage = "T500 is outside the valid range of -50 to 0.";
            break;
        case 4:
            errorMessage = "Td850 is outside the valid range of -40 to 40.";
            break;
        case 5:
            errorMessage = "Td700 is outside the valid range of -60 to 10.";
            break;
        case 6:
            errorMessage = "The dew point Td850 exceeds the temperature T850.";
            break;
        case 7:
            errorMessage = "The dew point Td700 exceeds the temperature T700.";
            break;
        default:
            errorMessage = "Unknown error with data.";

    }
    return errorMessage;

}

/**
 * This method displays a risk assessment message based on the TTIndex
 * If the data is valid, it returns the risk assessment based on the TTIndex
 * If the data is invalid, it gives a message to correct the data
 *
 * @return String  - a risk assessment or message to correct data
 */
public String TTIndexMessage()
{
    String Message;
    int TTIndex = calculateTTIndex();

    if (invalidCode() != 0)     // if invalid data, give informative message to correct data
    {
        Message = invalidMessage() +
                "\nNo Index has been calculated." +
                "\nPlease correct the input data.";

    } else                      // if valid data, get risk statement
    {
        if (TTIndex < 44)
        {
            Message = "Thunderstroms Unlikely";
        } else if (TTIndex < 46)
        {
            Message = "Isolated Moderate Thunderstorms";
        } else if (TTIndex < 48)
        {
            Message = "Scattered Moderate, Few Heavy Thunderstorms";
        } else if (TTIndex < 50)
        {
            Message = "Scattered Moderate, Few Heavy, Isolated Severe Thunderstorms";
        } else if (TTIndex < 52)
        {
            Message = "Scattered Heavy, Few Severe Thunderstorms, Isolated Tornadoes";
        } else if (TTIndex < 56)
        {
            Message = "Scattered to Numerous Heavy, Few to Scattered Severe Thunderstorms, Isolated Tornadoes";
```

```
            } else
            {
                Message = "Numerous Heavy, Scattered Severe Thunderstorms, Few to Scattered Tornadoes";
            }

        }
        return Message;
    }

    /**
     * This method displays a risk assessment message based on the KIndex
     * If the data is valid, it returns the risk assessment based on the KIndex
     * If the data is invalid, it gives a message to correct the data
     *
     * @return String  - a risk assessment or message to correct data
     */
    public String KIndexMessage()
    {
        String Message;
        int KIndex = calculateKIndex();

        if (invalidCode() != 0)
        {
            Message = invalidMessage() +
                    "\nNo Index has been calculated." +
                    "\nPlease correct the input data.";

        } else
        {
            if (KIndex < 20)
            {
                Message = "Thunderstroms Unlikely";
            } else if (KIndex < 26)
            {
                Message = "Isolated Thunderstorms";
            } else if (KIndex < 31)
            {
                Message = "40% - 60% chance of thunderstorms";
            } else if (KIndex < 36)
            {
                Message = "60% - 80% chance of thunderstorms, some severe";
            } else if (KIndex < 41)
            {
                Message = "80% - 90% chance of heavy thunderstorms, some severe";
            } else
            {
                Message = "Almost 100% chance of thunderstorms, some severe";
            }
        }
    return Message;
    }

}
```

Coding excellent, as always.

```
/*********************************************************************************************
 *  This program simulates a weather risk assessment program and demonstrates using a
 *  graphical interface to test the functionality of the UpperAirObject class
 *
 *  The user is to enter data representing temperature and dew pressure values at multiple
 *  air pressures and after clicking the calculate button, a report is generated and some
 *  weather icons displayed
 *
 *
 *
 *  CST 183 Programming Assignment 5
 *  @author Michael Clinesmith
 *********************************************************************************************/

import javafx.application.Application;
import javafx.scene.control.*;
import javafx.scene.image.Image;
import javafx.scene.image.ImageView;
import javafx.scene.layout.HBox;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.layout.VBox;
import javafx.geometry.Pos;
import javafx.event.EventHandler;
import javafx.event.ActionEvent;
import javafx.scene.text.Font;
import javafx.scene.text.FontPosture;
import javafx.scene.text.FontWeight;

public class WeatherInterface extends Application
{
   /*-------------------------------------------------------------------------------
   * Object declarations
   *
   * Note: I am unsure if putting all of these objects as private global variables is
   * good or standard programming practice, however I do not know how to better organize
   * the program to set up all the objects otherwise
   -------------------------------------------------------------------------------*/
   // graphical interface objects
   private Label T850, T700, T500, Td850, Td700, Instructions, TempLabel, DewLabel, ForcastLabel;
   private Label TTIndexLabel, TTIndex, KIndexLabel, KIndex, TTWarning, KWarning, RiskLabel;
   private Label ImageCredits, ProgramTitle, DesignerCredits;
   private TextField T850TF, T700TF, T500TF, Td850TF, Td700TF;
   private Button CalculateButton, ClearButton, QuitButton;
   private TextArea ForcastArea;

   private Image  sunnyImage, sunny1Image, partlySunnyImage, partlySunny1Image, mostlyCloudyImage,
                  mostlyCloudy1Image, drizzleImage, drizzle1Image, rainImage,
                  rain1Image, lightningImage, lightning1Image, tornadoImage, tornado1Image;
   private ImageView sunnyIView, sunny1IView, partlySunnyIView, partlySunny1IView, mostlyCloudyIView,
                     mostlyCloudy1IView, drizzleIView, drizzle1IView, rainIView,
                     rain1IView, lightningIView, lightning1IView, tornadoIView, tornado1IView;

   // storage containers to set up graphical interface
   private HBox CreditsBox, PartialForcastBox;
   private HBox op0Box, op1Box, op2Box, op3Box, op4Box, op5Box, op6Box;
   private VBox LeftForcastBox, RightForcastBox, ForcastBox, op1VBox;

   private UpperAirData airData;                              // object to hold weather data
```

```
final private String INTRO_MESSAGE = "Enter temperature values into fields then click the " +
        "calculate button to generate a report.";

private FontWeight   FontWt = FontWeight.BOLD;           // used to adjust text settings
private FontPosture  FontPo = FontPosture.REGULAR;

/**
 * main method of program, used to launch graphical interface
 * @param args String array - arguments are not used besides being passed to launch method
 */
public static void main(String[] args)
{
   // Launch the application.
   launch(args);
}

 /**
  * Method to start the graphical interface
  * @param primaryStage Stage object used to show graphical interface
  */
@Override
public void start(Stage primaryStage)
{
   initializeScene();

   // Set up overall scene
   Scene scene = new Scene(op1VBox, 1000, 700);
   primaryStage.setScene(scene);
   primaryStage.setTitle("Weather Forcaster");
   primaryStage.show();

 }

 /**
  * Method that calls other methods to initialize objects for display on the scene
  */
 public void initializeScene()
 {
     // call methods to create scene
     loadImages();
     setLabels();
     createTextAreas();
     createButtons();
     createForcastBox();
     createSceneContainers();
 }

 /**
  * Method to create objects for images
  */
public void loadImages()
{
   // load images
   sunnyImage = new Image("file:sunny.png");
   sunny1Image = new Image("file:sunny.png");
   partlySunnyImage = new Image("file:partlySunny.png");
   partlySunny1Image = new Image("file:partlySunny.png");
   mostlyCloudyImage = new Image("file:mostlyCloudy.png");
   mostlyCloudy1Image = new Image("file:mostlyCloudy.png");
   drizzleImage = new Image("file:drizzle.png");
```

```
    drizzle1Image = new Image("file:drizzle.png");
    rainImage = new Image("file:rain.png");
    rain1Image = new Image("file:rain.png");
    lightningImage = new Image("file:lightning.png");
    lightning1Image = new Image("file:lightning.png");
    tornadoImage = new Image("file:tornado.png");
    tornado1Image = new Image("file:tornado.png");

    // create ImageView objects for display
    sunnyIView = new ImageView(sunnyImage);
    sunny1IView = new ImageView(sunny1Image);
    partlySunnyIView = new ImageView(partlySunnyImage);
    partlySunny1IView = new ImageView(partlySunny1Image);
    mostlyCloudyIView = new ImageView(mostlyCloudyImage);
    mostlyCloudy1IView = new ImageView(mostlyCloudy1Image);
    drizzleIView = new ImageView(drizzleImage);
    drizzle1IView = new ImageView(drizzle1Image);
    rainIView = new ImageView(rainImage);
    rain1IView = new ImageView(rain1Image);
    lightningIView = new ImageView(lightningImage);
    lightning1IView = new ImageView(lightning1Image);
    tornadoIView = new ImageView(tornadoImage);
    tornado1IView = new ImageView(tornado1Image);
}

 /**
  * Method to create and initialize label objects
  */
public void setLabels()
{
    T850 = new Label("T850:");
    T700 = new Label("T700:");
    T500 = new Label("T500:");
    Td850 = new Label("Td850:");
    Td700 = new Label("Td700:");
    Instructions = new Label("Enter values for Temperatures in degrees C");
    TempLabel = new Label( "Pressure Level Temperatures: ");
    DewLabel = new Label( "Pressure Level Dew Points: ");
    ForcastLabel =  new Label("Projected Weather Forcast Below:");
    TTIndexLabel = new Label( "Total Totals Index: ");
    KIndexLabel = new Label("K-Index: ");
    TTIndex = new Label("42.0");
    KIndex = new Label( "19.0");
    TTWarning = new Label( "Thunderstorms Unlikely");
    KWarning = new Label("Thunderstorms Unlikely");
    RiskLabel = new Label("Severe Weather Risk: ");
    ImageCredits = new Label("HTC Sense5 Icons credited to Jesse Penico");
    DesignerCredits = new Label( "designed by Michael Clinesmith");
    ProgramTitle = new Label("Weather Risk Assessment Program");

    // adjust label size and style
    ForcastLabel.setFont(Font.font("Arial", FontWt, FontPo, 20));
    ProgramTitle.setFont(Font.font("Arial", FontWt, FontPo, 20));

}

 /**
  * Method to create and initialize text fields and areas
  */
public void createTextAreas()
```

Variance of the images was a very nice touch.

```
{
   T850TF = new TextField();
   T700TF = new TextField();
   T500TF = new TextField();
   Td850TF = new TextField();
   Td700TF = new TextField();
   ForcastArea = new TextArea(INTRO_MESSAGE);
   ForcastArea.setWrapText(true);                // wraps text instead of scrolling

   // adjust size of text areas
   T850TF.setPrefColumnCount(4);
   T700TF.setPrefColumnCount(4);
   T500TF.setPrefColumnCount(4);
   Td850TF.setPrefColumnCount(4);
   Td700TF.setPrefColumnCount(4);
}

 /**
  * Method to create buttons
  */
public void createButtons()
{
   CalculateButton = new Button( "Calculate");
   ClearButton = new Button( "Clear");
   QuitButton = new Button( "Quit");

   // set buttons to use same click handler
   CalculateButton.setOnAction(new ButtonClickHandler());
   ClearButton.setOnAction(new ButtonClickHandler());
   QuitButton.setOnAction(new ButtonClickHandler());

}

 /**
  * Method that creates and organizes the ForcastBox that displays the data report
  */
public void createForcastBox()
{
   LeftForcastBox = new VBox();
   RightForcastBox = new VBox();
   CreditsBox = new HBox(ImageCredits);
   CreditsBox.setAlignment(Pos.CENTER);
   PartialForcastBox = new HBox(10, LeftForcastBox, ForcastArea, RightForcastBox);
   PartialForcastBox.setAlignment(Pos.CENTER);
   ForcastBox = new VBox(10, PartialForcastBox, CreditsBox);
}

 /**
  * Method to create the containers to attach to the main node, op1VBox
  */
public void createSceneContainers()
{
   //  put together horizontal containers for main node
   op0Box = new HBox(10, ProgramTitle);
   op1Box = new HBox(10, DesignerCredits);
   op2Box = new HBox(10, TempLabel, T850, T850TF, T700, T700TF, T500, T500TF);
   op3Box = new HBox(10, DewLabel, Td850, Td850TF, Td700, Td700TF);
   op4Box = new HBox(10, Instructions );
   op5Box = new HBox(10, CalculateButton, ClearButton, QuitButton);
   op6Box = new HBox(10, ForcastLabel);
```

```
   // center containers
   op0Box.setAlignment(Pos.CENTER);
   op1Box.setAlignment(Pos.CENTER);
   op2Box.setAlignment(Pos.CENTER);
   op3Box.setAlignment(Pos.CENTER);
   op4Box.setAlignment(Pos.CENTER);
   op5Box.setAlignment(Pos.CENTER);
   op6Box.setAlignment(Pos.CENTER);

   // put together main node
   op1VBox = new VBox(10, op0Box, op1Box, op2Box, op3Box, op4Box, op5Box, op6Box, ForcastBox);
}

 /**
  * class created to handle button clicks
  */
 class ButtonClickHandler implements EventHandler<ActionEvent>
 {
     /**
      * This method handles button click events
      *
      * If the calculate button is clicked, the data in the textfields is stored into a
      * UpperAirData object and a report is generated which includes updating the side boxes
      * to contain weather icons
      *
      * If the clear button is clicked, the field data is cleared and forcast box is reset
      *
      * If the quit button is clicked, the program ends
      *
      * @param event ActionEvent object that contains data about a button click event
      */
     @Override
     public void handle(ActionEvent event)
     {

         if (event.getSource() == CalculateButton)             // calculate based on data
         {
             int int850, int700, int500, int850d, int700d;
             int intTTIndex, intKIndex;

             // get textfield values for weather object

             int850 = Integer.parseInt(T850TF.getText());
             int700 = Integer.parseInt(T700TF.getText());
             int500 = Integer.parseInt(T500TF.getText());
             int850d = Integer.parseInt(Td850TF.getText());
             int700d = Integer.parseInt(Td700TF.getText());

             // create object
             airData = new UpperAirData(int850, int700, int500, int850d, int700d);

             // perform actions based on data

             ForcastArea.setText(airData.toString());
             if (airData.isValid())        // if valid data
             {
                 intTTIndex = airData.calculateTTIndex();
                 intKIndex = airData.calculateKIndex();
```

```
            LeftForcastBox.getChildren().clear();
            RightForcastBox.getChildren().clear();

            // left box icons for TT index value

            if (intTTIndex < 44)
            {
                LeftForcastBox.getChildren().addAll(sunnyIView);

            } else if (intTTIndex < 46)
            {
                LeftForcastBox.getChildren().addAll(partlySunnyIView, drizzleIView);
            } else if (intTTIndex < 48)
            {
                LeftForcastBox.getChildren().addAll(mostlyCloudyIView, rainIView, lightningIView);
            } else if (intTTIndex < 50)
            {
                LeftForcastBox.getChildren().addAll(rainIView, lightningIView);
            } else
            {
                LeftForcastBox.getChildren().addAll(rainIView, lightningIView, tornadoIView);
            }

            // right box icons for K-index value

            if (intKIndex < 20)
            {
                RightForcastBox.getChildren().addAll(sunny1IView);
            } else if (intKIndex < 26)
            {
                RightForcastBox.getChildren().addAll(partlySunny1IView, drizzle1IView);
            } else if (intKIndex < 31)
            {
                RightForcastBox.getChildren().addAll(mostlyCloudy1IView, rain1IView, lightning1IView);

            } else if (intKIndex < 36)
            {
                RightForcastBox.getChildren().addAll(rain1IView, lightning1IView);
            } else
            {
                RightForcastBox.getChildren().addAll(rain1IView, lightning1IView, tornado1IView);

            }
        }
        else          // clear the side box icons since not valid data
        {
            LeftForcastBox.getChildren().clear();
            RightForcastBox.getChildren().clear();
        }

        LeftForcastBox.setAlignment(Pos.CENTER);
        RightForcastBox.setAlignment(Pos.CENTER);

    } else if (event.getSource() == ClearButton)      // clear the text and icons
    {
        T850TF.setText("");
        T700TF.setText("");
        T500TF.setText("");
        Td850TF.setText("");
        Td700TF.setText("");
```

```
                ForcastArea.setText(INTRO_MESSAGE);

                LeftForcastBox.getChildren().clear();
                RightForcastBox.getChildren().clear();

                LeftForcastBox.setAlignment(Pos.CENTER);
                RightForcastBox.setAlignment(Pos.CENTER);

            } else if (event.getSource() == QuitButton)       // exit program
            {
                System.exit(0);
            }
        }
    }
}
```

Weather Forcaster

Weather Risk Assessment Program

designed by Michael Clinesmith

Pressure Level Temperatures: T850: 14 T700: 4 T500: -14

Pressure Level Dew Points: Td850: 11 Td700: -8

Enter values for Temperatures in degrees C

Calculate Clear Quit

Projected Weather Forcast Below:

T850: 14 C
T700: 4 C
T500: -14 C
Td850: 11 C
Td700: -8 C

TTIndex: 53
TTIndex Report: Scattered to Numerous Heavy, Few to Scattered Severe Thunderstorms, Isolated Tornadoes

KIndex: 27
KIndex Report: 40% - 60% chance of thunderstorms

HTC Sense5 Icons credited to Jesse Penico