Creating "Nifty" Assignments for Java Programming Courses

and a Discussion of the Necessary Aspects of Java

Prof. David Bernstein
James Madison University

Computer Science Department

bernstdh@jmu.edu

Purpose

- Foundation/Background:

  Some aspects of Java that you may not be familiar with

- Application:

  Using these aspects of Java to create "nifty" (i.e., engaging) programming assignments

A Quick Look Ahead

- Foundation/Background:

  The basics of GUI programming

  The basics of 2-D graphics programming

  The basics of animation

- Application:

  An assignment using arithmetic operators and functions to create an automobile dashboard (GUI)

  An assignment using iteration to calculate wind-chill factors and display them on a map (Graphics)

  An assignment using recursion to explore a building (Animation)

Part 1

Graphical User Interfaces

The Building Blocks of GUIs

- Components/Widgets:

  The parts of a GUI (e.g., buttons, sliders, text fields)

- Containers:

"Screen real estate" that contains components/widgets

Some Top-Level Containers (i.e., Realizations of **`JRootPaneContainer`**)

- **`JFrame`** :

  A main "window" most commonly used by an application

- **`JApplet`** :

  Most commonly a portion of a WWW page (displayed in a browser)

Creating a **`JFrame`**

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


/**
 * An example that uses a JFrame
 *
 * @version 1.0
 * @author  Prof. David Bernstein, James Madison Univeristy
 */
public class JFrameDriver
{

    /**
     * The entry point of the example
     *
     * @param args   The command line arguments
     */
    public static void main(String[] args)
    {
        JFrame   f;


        f = new JFrame();
        f.setSize(400,400);
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        f.setVisible(true);
    }
}
```

Working with Top-Level Containers

- The **`JRootPane`** :

  `JRootPaneContainer` objects have a **`JRootPane`** that manages layers (if any), menus (if any), and content

  Should not be used

- The Content Pane:

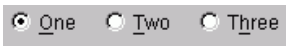
  A **`JRootPane`** has a **`Container`** called the content pane that manages all of the usable "screen real estate"

  Accessed using the **`getContentPane()`** method

Some GUI Components

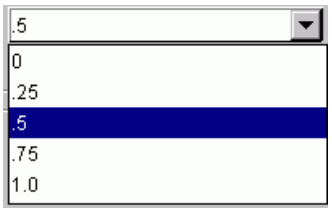
- **`JButton`** :

- **`JCheckBox` /`JRadioButton`** :



- **`JTextField` /`JTextArea`** :

- **`JComboBox`** :



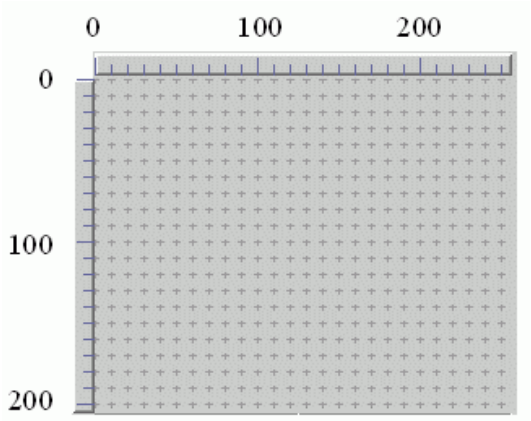
- **`JSlider`** :

GUI Layout

- Organizing GUI components
- Grouping GUI components
- Positioning GUI components

Absolute Layout

The "Graph Paper" Approach



Absolute Layout in Java

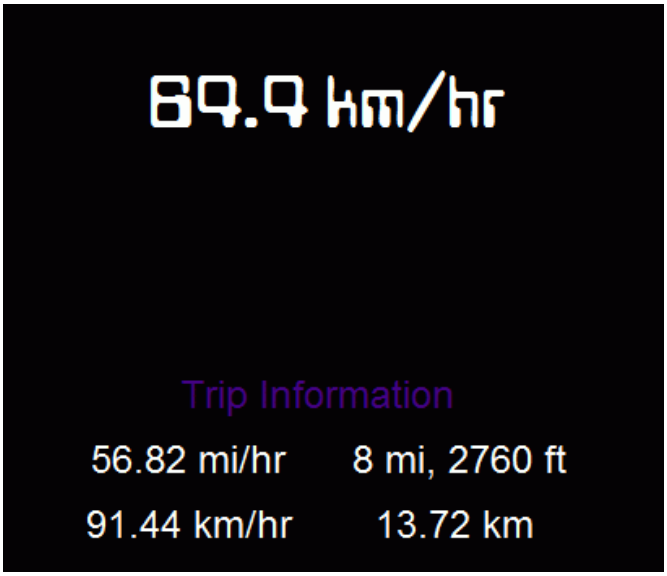
The **`null` `LayoutManager`**

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

/**
 * A frame that illustrates absolute layout with the null
 * layout manager
 *
 * @author  Prof. David Bernstein, James Madison University
 * @version 1.0
 */
```

```
public class NullLayoutFrame extends JFrame
{
    Container   contentPane;
    JButton     cancelButton, okButton;
    JLabel      titleLabel;


    /**
     * Default constructor
     */
    public NullLayoutFrame()
    {
       super("A Really Amazing Window!");
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

       contentPane = getContentPane();
       contentPane.setLayout(null);

       titleLabel = new JLabel("An Example of Absolute Layout");
       contentPane.add(titleLabel);
       titleLabel.setBounds(60,20,290,30);

       okButton = new JButton("OK");
       contentPane.add(okButton);
       okButton.setBounds(190,210,60,30);

       cancelButton = new JButton("Cancel");
       contentPane.add(cancelButton);
       cancelButton.setBounds(260,210,90,30);
    }

}
```

An Example of a "Nifty" Assignment

- The Important Concepts:
  - Simple arithmetic operators
  - Functions (i.e., static methods)
  - Integer arithmetic
- The Setting - A Dashboard for U.S. Cars in Europe:
  - Convert the measured speed (in mi/hr) to km/hr
  - Convert the trip distance (in ft) to mi,ft and km
  - Convert the trip distance (in ft) and duration (in hr) to average trip speeds (in mi/hr and km/hr)
- Making the Assignment More Engaging:
  - Instead of displaying the results on the console, display them on a GUI



The GUI for this "Nifty" Assignment

A Version that Assumes Students Understand Objects

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

/**
 * A Dashboard can be used to display the current speed, the trip
 * distance, and the average trip speed.
 *
 * Note: This is not a well-designed class. It is designed in such a way
 * that a single .class file can be given to introductory programming
 * students.
 *
 * @author  Prof. David Bernstein, James Madison University
 * @version 1.0 (Digital)
 */
public class Dashboard extends JFrame
{
    private boolean           showTripDistance, showTripSpeed;
    private double            speed, tripKM, tripSpeedKPH, tripSpeedMPH;
    private int               tripFeet, tripMiles;
    private JLabel            speedLabel, tripLabel, tripKMLabel, tripKPHLabel;
    private JLabel            tripMILabel, tripMPHLabel;

    private final Color GOLD   = new Color(203,182,119);
    private final Color PURPLE = new Color( 69,  0,132);


    /**
     * Default Constructor
     */
    public Dashboard()
    {
       super();
       Font          large, small;
       JPanel        contentPane;
       String        lookAndFeel;

       try
       {
          lookAndFeel = UIManager.getSystemLookAndFeelClassName();
          UIManager.setLookAndFeel(lookAndFeel);
       }
       catch (Exception e)
       {
          // Use the default look and feel
       }
```

```
      setTitle("DukeDash");
      setSize(640,640);
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      setResizable(false);
      contentPane = (JPanel)getContentPane();
      contentPane.setBackground(Color.BLACK);
      contentPane.setLayout(null);
      contentPane.setBackground(Color.BLACK);

      speed  =   0.0;

      showTripDistance     = false;
      showTripSpeed        = false;

      large = new Font("JACKIE", Font.PLAIN, 48);
      small = new Font(Font.SANS_SERIF, Font.PLAIN, 24);

      speedLabel = new JLabel();
      speedLabel.setFont(large);
      speedLabel.setBounds(160,160,320,160);
      speedLabel.setBackground(Color.BLACK);
      speedLabel.setForeground(Color.WHITE);
      speedLabel.setHorizontalAlignment(SwingConstants.CENTER);
      contentPane.add(speedLabel);

      tripLabel = new JLabel();
      tripLabel.setFont(small);
      tripLabel.setBounds(160,400, 320, 40);
      tripLabel.setBackground(Color.BLACK);
      tripLabel.setForeground(PURPLE);
      tripLabel.setHorizontalAlignment(SwingConstants.CENTER);
      contentPane.add(tripLabel);

      tripMPHLabel = new JLabel();
      tripMPHLabel.setFont(small);
      tripMPHLabel.setBounds(160,440, 160, 40);
      tripMPHLabel.setBackground(Color.BLACK);
      tripMPHLabel.setForeground(Color.WHITE);
      tripMPHLabel.setHorizontalAlignment(SwingConstants.CENTER);
      contentPane.add(tripMPHLabel);

      tripKPHLabel = new JLabel();
      tripKPHLabel.setFont(small);
      tripKPHLabel.setBounds(160,480, 160, 40);
      tripKPHLabel.setBackground(Color.BLACK);
      tripKPHLabel.setForeground(Color.WHITE);
      tripKPHLabel.setHorizontalAlignment(SwingConstants.CENTER);
      contentPane.add(tripKPHLabel);

      tripMILabel = new JLabel();
      tripMILabel.setFont(small);
      tripMILabel.setBounds(320,440, 160, 40);
      tripMILabel.setBackground(Color.BLACK);
      tripMILabel.setForeground(Color.WHITE);
      tripMILabel.setHorizontalAlignment(SwingConstants.CENTER);
      contentPane.add(tripMILabel);

      tripKMLabel = new JLabel();
      tripKMLabel.setFont(small);
      tripKMLabel.setBounds(320,480, 160, 40);
      tripKMLabel.setBackground(Color.BLACK);
      tripKMLabel.setForeground(Color.WHITE);
      tripKMLabel.setHorizontalAlignment(SwingConstants.CENTER);
      contentPane.add(tripKMLabel);

      setVisible(true);
   }



   /**
    * Repaint this component if it is visible
    */
   public void refresh()
   {
      speedLabel.setText(String.format("%5.1f km/hr", speed));

      if (showTripDistance || showTripSpeed)
      {
         tripLabel.setText("Trip Information");
      }
```

```
      if (showTripDistance)
      {
        tripMILabel.setText(String.format("%3d mi, %4d ft",
                                          tripMiles, tripFeet));
        tripKMLabel.setText(String.format("%6.2f km", tripKM));
      }

      if (showTripSpeed)
      {
        tripMPHLabel.setText(String.format("%5.2f mi/hr", tripSpeedMPH));
        tripKPHLabel.setText(String.format("%5.2f km/hr", tripSpeedKPH));
      }

      // This will put the calling thread to sleep briefly
      // so that the attributes can be changed in a loop
      // and appear to be animated
      try
      {
         Thread.sleep(100);
      }
      catch (InterruptedException ie)
      {
         // Ignore
      }
   }


   /**
    * Set the current speed (in km/hr)
    *
    * @param kph   The current speed
    */
   public void setSpeed(double kph)
   {
      speed = kph;

      refresh();
   }


   /**
    * Set the distance traveled on this trip in both kilometers
    * and miles,feet.  Note: This method does not check to ensure that
    * the two values are consistent
    *
    * @param km    The distance in kilometers
    * @param mile  The distance in whole miles
    * @param feet  The distance in "leftover" feet
    */
   public void setTripDistance(double km, int miles, int feet)
   {
      showTripDistance  = true;
      tripKM            = km;
      tripMiles         = miles;
      tripFeet          = feet;

      refresh();
   }



   /**
    * Set the average speed during this trip in both km/hr and
    * mi/hr. Note: This method does not check to ensure that
    * the two values are consistent
    *
    * @param kph   The average speed in kph
    * @param mph   The average speed in mph
    */
   public void setTripSpeed(double kph, double mph)
   {
      showTripSpeed = true;
      tripSpeedKPH  = kph;
      tripSpeedMPH  = mph;

      refresh();
   }
}
```

Using the GUI for this "Nifty" Assignment

If Students Understand Objects

```
Dashboard dashboard;
double    currentKPH, tripKPH, tripMPH;


dashboard = new Dashboard();
dashboard.setSpeed(currentKPH);
dashboard.setTripSpeed(tripKPH, tripMPH);
```

If Students Don't Understand Objects (Using a Slightly Different Implementation)

```
double    currentKPH, tripKPH, tripMPH;


Dashboard.setSpeed(currentKPH);
Dashboard.setTripSpeed(tripKPH, tripMPH);
```

Other Approaches to Layout (Not Considered Here)

- Relative Layout:

  Components are positioned relative to each other, rather than an absolute position

- Template Layout:

  Divide the container into sections and place one component in each section

- Hierarchical Template Layout:

  Subdivide the sections in a template layout

- Constrained Template Layout:

  Add constraints (e.g., doesn't expand) to sections of a template

Responding to GUI Components

1. Determine what events the component generates
2. Identify the interfaces that needs to be implemented
3. Create a class that implements that interfaces
4. Add "application logic" to the appropriate methods

Example: Responding to **`JButton`** Objects

- The Event(s):

  `ActionEvent`

- The Interface(s):

  `ActionListener`

- The Method(s)/Message(s):

  `void actionPerformed(ActionEvent event)`

Example: Responding to **`JButton`** Objects (cont.)

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

/**
 * An example that uses buttons
 *
 * @author  Prof. David Bernstein, James Madison University
 * @version 1.0
 */
public class ButtonFrame extends JFrame implements ActionListener
{
    Container   contentPane;
    JButton     cancelButton, okButton;
    JLabel      titleLabel;


    /**
     * Default constructor
     */
    public ButtonFrame()
    {
       super("A Really Amazing Window!");
       setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

       // Get the content pane
       contentPane = getContentPane();
       contentPane.setLayout(null);

       // Construct the widgets.components
       titleLabel   = new JLabel("A Button Example");
       okButton     = new JButton("OK");
       cancelButton = new JButton("Cancel");

       // Layout the content pane
       contentPane.add(titleLabel);
       titleLabel.setBounds(60,20,290,30);

       contentPane.add(okButton);
       okButton.setBounds(190,210,60,30);

       contentPane.add(cancelButton);
       cancelButton.setBounds(260,210,90,30);

       // Make this object an ActionListener on the buttons
       okButton.addActionListener(this);
       cancelButton.addActionListener(this);
    }


    /**
     * Handle actionPerformed message (required by ActionListener)
     *
     * @param event   The ActionEvent that generatedt his message
     */
    public void actionPerformed(ActionEvent event)
    {
       String       command;

       command = event.getActionCommand();
       if (command.equals("Cancel"))
       {
          titleLabel.setText("You can't cancel me!");
       }
       else if (command.equals("OK"))
       {
          titleLabel.setText("I'm OK, you're not.");
       }
    }

}
```

Rendering Engines

- Purpose:

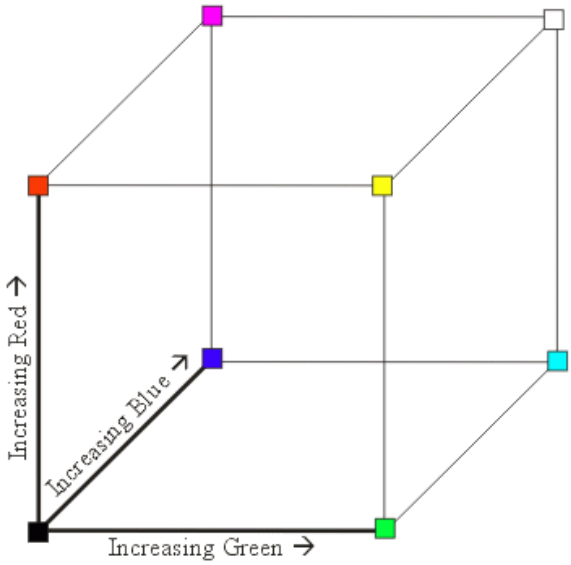
  Determine how to present *graphics primitives* on a visual output device

- In Java:

  The rendering engine is a `Graphics2D` object

Modeling Color

The (Linear) Color Cube



Modeling Color (cont.)

- The RGB Model:

  Begin with black then add red, green, and or blue

  An *additive* model

  Often used for displays/monitors

- The CMYK Model:

  Begin with white then remove cyan, magenta, and/or yellow

  A *subtractive* model

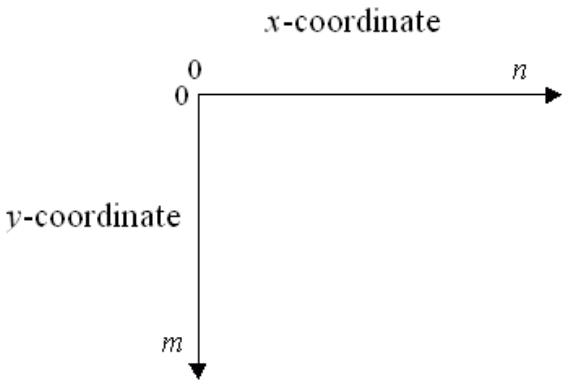
  Often used for printing

Coordinate Systems

- Coordinates:

  Quantities (linear and/or angular) that designate the position of a point in relation to a given reference frame

- In Java:



Rendering with a `Graphics2D` Object

- Geometric Shapes (see the **Shape** interface):

```
draw(Shape s)
```

```
fill(Shape s)
```

- Strings/Text:

```
drawString(String s, float x, float y)
```

- Images:

```
drawImage(Image i, int x, int y, null)
```

Classes that Implement the `Shape` Interface

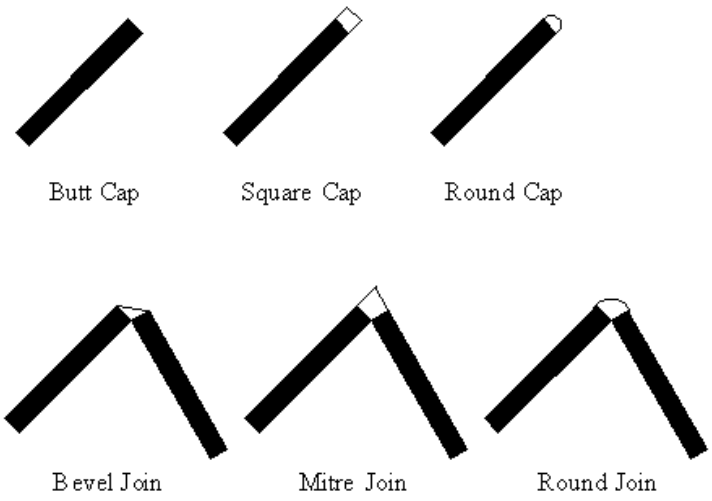
- 0-Dimensional:
  - Points (`Point2D`)
- 1-Dimensional:
  - Lines (`Line2D`)
  - Curves (`CubicCurve2D` and `QuadCurve2D`)
- 2-Dimensional:
  - Rectangles(`Rectangle2D`)
  - Polygons (`GeneralPath`)
  - Ellipses (`Ellipse2D`)

Stroking a Shape

- Stroke:
  - The line "style" (see the `Stroke` interface)
  - Set using the `setStroke` method in `Graphics2D`
- Color:
  - The line color (see the `Color` class)
  - Set using the `setColor` method in `Graphics2D`

Stroking a Shape (cont.)

Joins and Caps



Filling a Shape

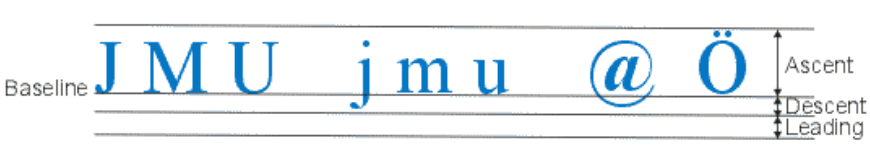
- Paint:
  - The fill "style" and color (see the **`Paint`** interface)
  - Set using the **`setPaint`** method in **`Graphics2D`**
- Types:
  - **`Color`**
  - **`GradientPaint`**
  - **`TexturePaint`**

Rendering Text/Strings

- Font:
  - The "glyphs" used for each character
  - Set using the **`setFont`** method in **`Graphics2D`**
- Properties:
  - Name
  - Size (in points)
  - Style (plain, italic, bold)

Rendering Text/Strings (cont.)

Line Metrics



The **x** and **y** passed to **drawString(String s, float x, float y)** are the left end of the baseline.

Rendering Text/Strings (cont.)

- **FontRenderContext**

  Keeps information about fonts

- **LineMetrics**

  Keeps information about font and line heights

Rendering Images

- **BufferedImage** Class:

  Extends the abstract `Image` class

  Objects can be read from a file using the static `read()` method in the `ImageIO` class

- The **drawImage(Image i, int x, int y, null)** Method:

  `x` and `y` determine where the the upper-left corner of the `Image` will be rendered

What GUI Component To Use?

- Motivation:

  You need a GUI component to render in

  You can extend any GUI component, you just need to override the `public void paint(Graphics g)` method

- A Good Class to Extend:

  `JComponent` which does very little in its `paint()` method

An Example

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

/**
 * A simple example of graphics
 *
 * @author  Prof. David Bernstein, James Madison University
 * @version 1.0
 */
public class Example extends JComponent
{
    public static final float[] DASHED_PATTERN = {10.0f, 10.0f};
    public static final float[] DOTTED_PATTERN = { 2.0f,  2.0f};


    /**
     * Render this JComponent
     *
     * @param g   The rendering engine to use
     */
    public void paint(Graphics g)
```

```
    {
       Font                 serifFont;
       GradientPaint        gradient;
       Graphics2D           g2;
       Line2D.Float         line;
       Path2D.Float         path;
       Rectangle2D.Float    rectangle;
       Stroke               dashed, dotted, solid;

       g2 = (Graphics2D)g;


       // Create some Stroke objects
       dashed = new BasicStroke(2.0f,
                                BasicStroke.CAP_BUTT,
                                BasicStroke.JOIN_MITER,
                                10.0f,
                                DASHED_PATTERN,
                                0.0f);

       dotted = new BasicStroke(2.0f,
                                BasicStroke.CAP_BUTT,
                                BasicStroke.JOIN_MITER,
                                10.0f,
                                DOTTED_PATTERN,
                                0.0f);

       solid  = new BasicStroke(2.0f,
                                BasicStroke.CAP_BUTT,
                                BasicStroke.JOIN_MITER);

       // Create a Font object
       serifFont = new Font(Font.SERIF, Font.PLAIN, 20);



       // Create and render some lines
       g2.setColor(Color.red);

       line = new Line2D.Float(0f,0f,100f,100f);
       g2.draw(line);

       line = new Line2D.Float(100f,100f,200f,200f);
       g2.setStroke(dashed);
       g2.draw(line);

       line = new Line2D.Float(200f,200f,300f,300f);
       g2.setStroke(dotted);
       g2.draw(line);


       // Render some text
       g2.setColor(Color.blue);

       g2.setFont(serifFont);
       g2.drawString("Hello World",40.0f,100.0f);


       // Create a Paint object
       gradient = new GradientPaint(300.0f,300.0f,Color.green,
                                    500.0f,300.0f,Color.yellow);

       // Create and render a rectangle
       rectangle = new Rectangle2D.Float(300f,300f,200f,100f);
       g2.setPaint(gradient);
       g2.fill(rectangle);
       g2.setStroke(solid);
       g2.setColor(Color.black);
       g2.draw(rectangle);

       // Create and render a Path2D
       path = new Path2D.Float();
       path.moveTo(100.0f,300.0f);
       path.lineTo(200.0f,400.0f);
       path.lineTo(100.0f,350.0f);
       path.lineTo(  0.0f,400.0f);
       path.lineTo(100.0f,300.0f);

       g2.draw(path);
    }
}
```

An Example: The Driver

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


/**
 * A simple example of graphics
 *
 * @author  Prof. David Bernstein, James Madison University
 * @version 1.0
 */
public class ExampleDriver
{
    /**
     * The entry point
     *
     * @param args   The command line arguments
     */
    public static void main(String[] args)
    {
       Container      contentPane;
       Example        example;
       JFrame         window;


       window      = new JFrame("Graphics Example");
       contentPane = window.getContentPane();
       contentPane.setLayout(null);
       example = new Example();
       example.setBounds(0,0,500,500);
       window.add(example);
       window.setSize(600,600);
       window.setVisible(true);
    }
}
```

An Example of a "Nifty" Assignment

- The Important Concepts:

  Conditionals

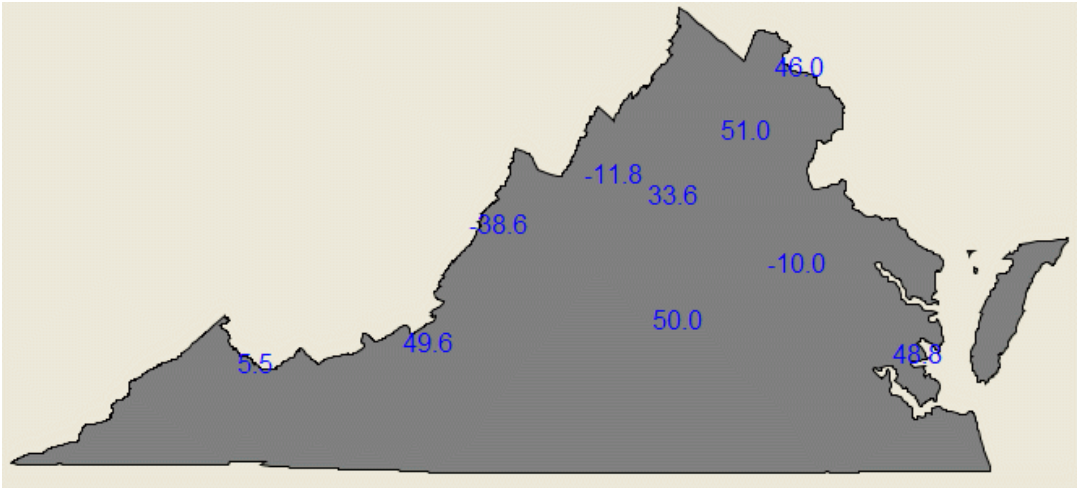
  Iteration

- The Setting - A Wind Chill Calculator:

  For several locations, convert the temperature and wind speeds to a wind chill value

- Making the Assignment More Engaging:

  Instead of displaying the results on the console, display them on a map



The GUI for this "Nifty" Assignment

A Version that Assumes Students Understand Objects

```
import java.awt.*;
import java.awt.geom.*;
import java.io.*;
import java.util.*;
import javax.swing.*;


/**
 * A component that displays information about temperatures on
 * a map
 *
 * @author  Prof. David Bernstein, James Madison University
 * @version 1.0
 */
public class MapComponent extends JComponent
{
    // Attributes
    private ArrayList<Shape>                shapes;
    private BufferedReader                  in;
    private HashMap<String, Point2D.Double> stations;
    private HashMap<String, Double>         temperatures;
    private JFrame                          frame;
    private JLabel                          maxLabel, minLabel;
    private JTextArea                       textArea;

    // Constants
    private final Font FIXED_WIDTH = new Font(Font.MONOSPACED, Font.PLAIN, 12);

    /**
     * Explicit Value Constructor
     *
     * @param region   The name of the region (e.g., "va")
     */
    public MapComponent(String region)
    {
       super();
       setFont(new Font(Font.SANS_SERIF, Font.PLAIN, 16));


       // Initialization
       shapes       = new ArrayList<Shape>();
       stations     = new HashMap<String, Point2D.Double>();
       temperatures = new HashMap<String, Double>();

       // Input
       try
       {
          readMap(region);
          readStations(region);
       }
       catch (Exception e)
       {
          e.printStackTrace();

       }
    }


    /**
     * Render this component
     *
     * @param g   The rendering engine to use
     */
    public void paint(Graphics g)
    {
       double           value;
       FontMetrics      metrics;
       Graphics2D       g2;
       Iterator<String> keys;
       Point2D          lonlat;
       String           key, text;
```

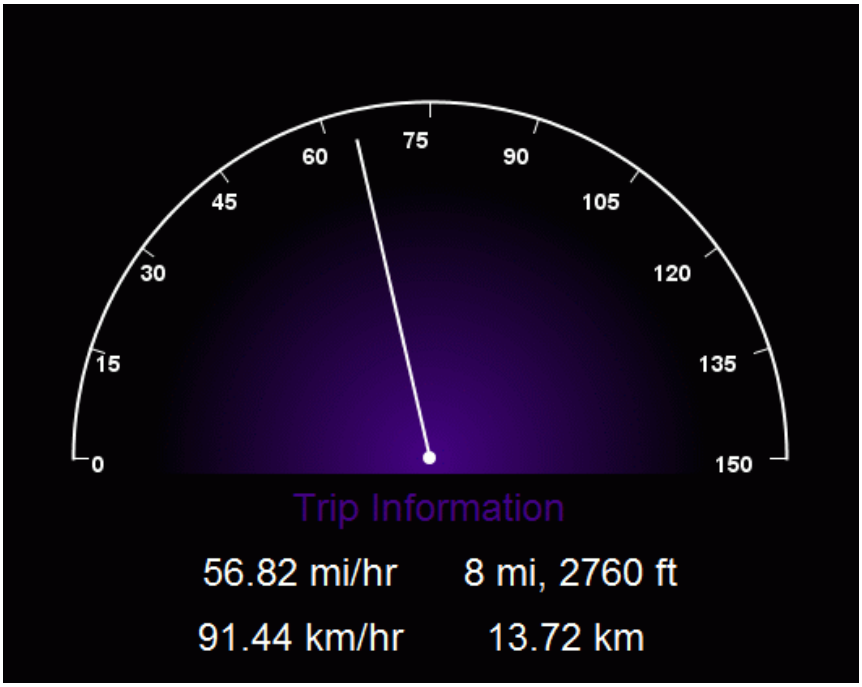
```
      super.paint(g);
      g2 = (Graphics2D)g;
      // Use antialiasing for shapes
      g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
                          RenderingHints.VALUE_ANTIALIAS_ON);
      // Use high-quality rendering
      g2.setRenderingHint(RenderingHints.KEY_RENDERING,
                          RenderingHints.VALUE_RENDER_QUALITY);
      // Use low-level "antialiasing for LCDs" for text
      g2.setRenderingHint(RenderingHints.KEY_TEXT_ANTIALIASING,
                          RenderingHints.VALUE_TEXT_ANTIALIAS_LCD_HRGB);
      metrics = g2.getFontMetrics(getFont());


      // Render the map
      for(Shape s: shapes)
      {
         g2.setColor(Color.GRAY);
         g2.fill(s);
         g2.setColor(Color.BLACK);
         g2.draw(s);
      }

      // Render the temperatures
      g2.setFont(getFont());
      keys = temperatures.keySet().iterator();
      while (keys.hasNext())
      {
         key    = keys.next();
         value  = temperatures.get(key);
         lonlat = stations.get(key);
         if (lonlat != null)
         {
            g2.setColor(Color.BLUE);
            text = String.format("%6.1f",value);
            g2.drawString(text,
                   (float)lonlat.getX()-(float)(metrics.stringWidth(text)/2.),
                   (float)lonlat.getY()-(float)(metrics.getHeight()/2.));
         }
      }
   }


   /**
    * Read the map
    *
    * @param region   The name of the region
    */
   private void readMap(String region) throws Exception
   {
      Shape                shape;
      String               id, line, token, type;
      StringTokenizer      st;

      in = new BufferedReader(new FileReader(region+".txt"));

      try
      {
         while ((line = in.readLine()) != null)
         {
            st = new StringTokenizer(line, "\t ");

            token = st.nextToken();
            type  = st.nextToken();

            // Read the ID if there is one
            id = null;

            if (st.hasMoreTokens())  token = st.nextToken();
            if (st.hasMoreTokens())  id = st.nextToken();


            shape = readPolygon();
            shapes.add(shape);
         }

         in.close();
      }
      catch (IOException ioe)
      {
```

```
         // Try again
      }
   }




   /**
    * Read a Polygon
    */
   private Shape readPolygon() throws IOException
   {
      boolean               keepReading;
      double                latitude, longitude;
      GeneralPath           polygon;
      String                line, token;
      StringTokenizer       st;

      polygon = null;
      keepReading = true;

      while (keepReading)
      {
         line = in.readLine();

         if ((line == null) || line.startsWith("END"))
         {
            keepReading = false;
         }
         else
         {
            st = new StringTokenizer(line, "\t ");

            token = st.nextToken();
            longitude = Double.parseDouble(token);

            token = st.nextToken();
            latitude = Double.parseDouble(token);

            if (polygon == null)
            {
               polygon = new GeneralPath();
               polygon.moveTo(longitude, latitude);
            }
            else polygon.lineTo(longitude, latitude);
         }
      }

      return polygon;
   }




   /**
    * Read the stations
    *
    * @param region   The name of the region
    */
   private void readStations(String region) throws Exception
   {
      BufferedReader        in;
      double                lat, lon, sign;
      double[]              p;
      String                icao4, line, token;
      StringTokenizer       st;

      in = new BufferedReader(new FileReader("stations-"+region+".txt"));

      try
      {
         while ((line = in.readLine()) != null)
         {
            st = new StringTokenizer(line, "\t");
            icao4  = st.nextToken();
            lon = Double.parseDouble(st.nextToken());
            lat = Double.parseDouble(st.nextToken());
            stations.put(icao4, new Point2D.Double(lon, lat));
         }
      }
      catch (Exception e)
```

```
        {
        }
    }


    /**
     * Set the temperature for a particular station
     *
     * @param icao4        The 4-letter ICAO code for the station
     * @param temperature  The temperature
     */
    public void setTemperature(String icao4, double temperature)
    {
       if (icao4 == null) icao4 = "null";

       temperatures.put(icao4, temperature);
       repaint();
    }

}
```

Another Example

The Dashboard Revisited



Part 3

Animation

Definitions

- Animate:

  To give life or motion

- Animation:

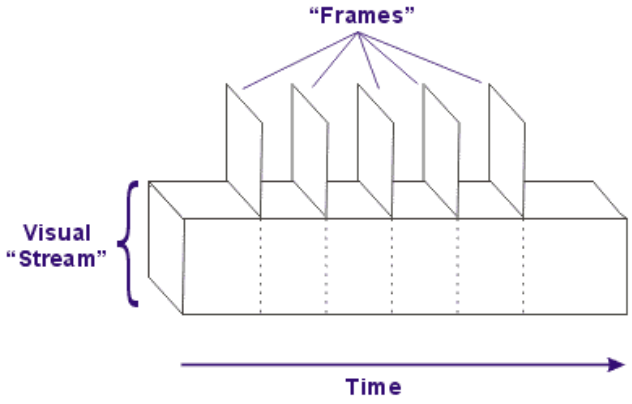
  A group of techniques that can make visual content appear to move

Techniques to Consider

- Sampled Dynamics (a.k.a. Frame Animation)
- Described Dynamics (a.k.a. Sprite Animation)

Sampling Dynamic Visual Content

An Illustration



Terminology for Sampled Dynamics

- *Frame Rate*:

  The number of frames per unit time

- *Animate*:

  To give life or motion

- *Frame Animation*:

  The "play back" of sampled dynamic visual content

Some Observations

- Each frame is thought of as a coherent whole
- Each frame consists of static visual content
- Each frame can be either sampled or described (and is completely independent of the time-based sampling)

Describing Dynamic Visual Content

- The Concept:

  Describe the way the visual "stream" changes over time

  Start with a set of components (i.e., the "participants" in the "action") and then *describe component-by-component changes*

- Analogy:

  A play with a script, stage directions, actors, sets, and a stage

Terminology for Described Dynamics

- Sprite:

  Common Usage - an imaginary being or elf

  Animation - a discrete piece of visual content that can respond to descriptions of changes

- Stage:

  The "component" that sprites act "on"

What's Involved?

- A **`Timer`** object that sends **`actionPerformed()`** messages
- An **`ActionListener`** on the **`Timer`** that updates state information each "tick"
- A **`JComponent`** that renders itself using the state information

A Simple Example

An Animated Rectangle

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

/**
 * A simple example of ad-hoc animation
 *
 * @author  Prof. David Bernstein, James Madison University
 * @version 1.0
 */
public class AnimatedRectangle extends JComponent implements ActionListener
{
    private float        direction, height, width, x, y;
    private Line2D.Float ground;
    private Timer        timer;


    /**
     * Default Constructor
     */
    public AnimatedRectangle()
    {
        super();

        width = 50.0f;
        height = 50.0f;

        x = 100.0f;
        y = 100.0f;
        direction = 1.0f;

        ground = new Line2D.Float(0f,480f,480f,480f);

        timer = new Timer(10, this);
        timer.start();
    }



    /**
     * Handle actionPerformed messages generated by the Timer
     * (Required by ActionListener)
     *
     * @param evt    The ActionEvent
     */
    public void actionPerformed(ActionEvent evt)
    {
        if      (y <= 100)        direction = 1.0f;
        else if (y+height >= 480) direction = -1.0f;
```

```
        y += direction;

        repaint();
    }



    /**
     * Render this Painting
     *
     * @param g   The rendering engine to use
     */
    public void paint(Graphics g)
    {
       Graphics2D            g2;
       Rectangle2D.Float     rectangle;

       g2 = (Graphics2D)g;
       g2.setColor(Color.BLACK);
       g2.draw(ground);

       g2.setColor(Color.RED);
       rectangle = new Rectangle2D.Float(x, y, width, height);
       g2.draw(rectangle);

    }
}
```

Another Simple Example

An Animated Curve

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

/**
 * A simple example of ad-hoc animation
 *
 * @author  Prof. David Bernstein, James Madison University
 * @version 1.0
 */
public class AnimatedCurve extends JComponent implements ActionListener
{
    private CubicCurve2D.Float   curve;
    private float[]              x, y;
    private Stroke               curveStroke;
    private Timer                timer;


    /**
     * Default Constructor
     */
    public AnimatedCurve()
    {
       super();

       curveStroke   = new BasicStroke(2.0f,
                                       BasicStroke.CAP_ROUND,
                                       BasicStroke.JOIN_MITER);

       setBackground(Color.white);


       x = new float[4];
       y = new float[4];

       x[0] =   0.0f;   y[0] =   0.0f;
       x[1] = 100.0f;   y[1] = 100.0f;
       x[2] = 300.0f;   y[2] = 300.0f;
       x[3] = 400.0f;   y[3] = 400.0f;

       curve = new CubicCurve2D.Float();
       updateCurve();
```

```
      timer = new Timer(100, this);
      timer.start();
   }



   /**
    * Handle actionPerformed messages generated by the Timer
    * (Required by ActionListener)
    *
    * @param evt    The ActionEvent
    */
   public void actionPerformed(ActionEvent evt)
   {
      x[0] += 1.0f;
      y[0] += 1.0f;

      x[1] += 1.0f;
      y[1] -= 1.0f;

      x[2] -= 1.0f;
      y[2] += 1.0f;

      x[3] -= 1.0f;
      y[3] -= 1.0f;

      updateCurve();
      repaint();
   }




   /**
    * Render this Painting
    *
    * @param renderer   The rendering engine to use
    */
   public void paint(Graphics   g)
   {
      Graphics2D     g2;

      g2 = (Graphics2D)g;

      g2.setColor(Color.black);

      if (curveStroke != null) g2.setStroke(curveStroke);

      if (curve != null) g2.draw(curve);
   }



   /**
    * Update the points in the curve
    */
   private void updateCurve()
   {
      curve.setCurve(x[0],y[0],x[1],y[1],
                     x[2],y[2],x[3],y[3]);
   }

}
```

User Interaction

- Mouse Events:

  A **`JComponent`** can have both **`MouseListener`** and **`MouseMotionListener`** objects

- Keyboard Events:

  A **`JComponent`** can have **`KeyListener`** objects

An Example of User Interaction

Whack-A-Duke

```
import java.awt.*;
import java.awt.event.*;
import java.awt.geom.*;
import java.io.*;
import java.util.Random;
import javax.imageio.*;
import javax.swing.*;

/**
 * An example of animation with user interaction
 *
 * @author  Prof. David Bernstein, James Madison University
 * @version 1.0
 */
public class WhackADuke extends    JComponent
                        implements ActionListener, MouseListener
{
    private Image              head;
    private int                direction, hits, speed, steps, x, y;
    private Random             rng;
    private Rectangle2D.Float  base;
    private Timer              timer;

    private final Color GOLD       = new Color(0xc2,0xa1,0x4d);

    private final Font  SCORE_FONT = new Font(Font.MONOSPACED, Font.PLAIN, 20);

    private final int HEAD_HEIGHT  = 70;
    private final int HEAD_WIDTH   = 70;
    private final int HIDDEN_STEPS = 10;

    /**
     * Default Constructor
     */
    public WhackADuke()
    {
       // Construct a random number generator
       rng = new Random();

       // Read the Madison head
       try
       {
          head = ImageIO.read(new File("madison.png"));
       }
       catch (IOException ioe)
       {
          head = null;
       }

       // Construct the base
       base = new Rectangle2D.Float(0f, 400f, 500f, 100f);

       // Initialize
       direction = 0;
       x         = 0;
       y         = (int)base.getY();
       speed     = 2;

       // Make this object a MouseListener on itself
       addMouseListener(this);

       // Construct and start the timer
       timer = new Timer(10, this);
       timer.start();
    }

    /**
     * Handle actionPerformed messages (required by ActionListener)
     *
     * @param  event   The ActionEvent that generated the message
     */
    public void actionPerformed(ActionEvent event)
    {
       if (direction == 0) // The head isn't moving
```

```
      {
         steps++;
         if (steps > HIDDEN_STEPS)
         {
            x = rng.nextInt(400);
            y = (int)base.getY();
            steps = 0;
            direction = -1;
         }
      }
      else if (direction == -1) // The head is moving up
      {
         y += direction * speed;

         if (y <= ((int)base.getY() - HEAD_HEIGHT))
         {
            direction = 1;
         }
      }
      else if (direction == 1) // The head is moving down
      {
         y += direction * speed;

         if (y >= (int)base.getY())
         {
            direction = 0;
         }
      }

      repaint();
   }

   /**
    * Handle mouseClicked messages (requried by MouseListener)
    *
    * @param event  The MouseEvent that generated the message
    */
   public void mouseClicked(MouseEvent event)
   {
      int       mouseX, mouseY;

      // Get the position of the mouse
      mouseX = event.getX();
      mouseY = event.getY();

      // Check for a whack
      if ((direction != 0) &&
          (mouseX >= x) && (mouseX <= x+HEAD_WIDTH) &&
          (mouseY >= y) && (mouseY<=  y+HEAD_WIDTH))
      {
         hits++;
      }
   }

   /**
    * Handle mouseEntered messages (requried by MouseListener)
    *
    * @param event  The MouseEvent that generated the message
    */
   public void mouseEntered(MouseEvent event)
   {
   }

   /**
    * Handle mouseExited messages (requried by MouseListener)
    *
    * @param event  The MouseEvent that generated the message
    */
   public void mouseExited(MouseEvent event)
   {
   }


   /**
    * Handle mousePressed messages (requried by MouseListener)
    *
    * @param event  The MouseEvent that generated the message
    */
   public void mousePressed(MouseEvent event)
   {
   }
```

```
    /**
     * Handle mouseReleased messages (requried by MouseListener)
     *
     * @param event  The MouseEvent that generated the message
     */
    public void mouseReleased(MouseEvent event)
    {
    }

    /*
     * Render this JComponent
     *
     * @param g   The rendering engine to use
     */
    public void paint(Graphics g)
    {
       float               height, width;
       Graphics2D          g2;
       Rectangle2D         background;


       g2 = (Graphics2D)g;

       // Clear the screen
       background = getBounds();
       width      = (float)background.getWidth();
       height     = (float)background.getHeight();
       g2.setColor(Color.WHITE);
       g2.fill(background);

       // Render the head
       g2.drawImage(head, x, y, null);

       // Render the base
       g2.setColor(GOLD);
       g2.fill(base);

       // Render the score
       g2.setColor(Color.BLACK);
       g2.setFont(SCORE_FONT);
       g2.drawString("Score: "+hits, 300, 100);
    }
}
```

A More Object-Oriented Example

A **`Fish`** Class

```
import java.awt.*;
import java.awt.image.*;
import java.util.*;


/**
 * A Fish that "swims" in an interesting way.
 *
 * @author  Prof. David Bernstein, James Madison University
 * @version 1.0
 */
public class Fish
{
    protected BufferedImage[]     images;
    protected int                 initialSpeed, maxX, maxY, speed, x, y;
    protected int                 state, stateChange;
    protected int                 ticks, ticksInState;



    private static final int     INITIAL_LOCATION = -320;
    private static final Random  rng = new Random();

    /**
     * Explicit Value Constructor
     *
     * @param threeImages  The three Image objects for this Fish
     * @param width        The width of the fishtank
```

```
     * @param height       The height of the fishtank
     * @param speed        The normal speed
     */
    public Fish(BufferedImage threeImages, int width, int height, int speed)
    {
       int     imageHeight, imageWidth;

       if (threeImages != null)
       {
          imageHeight = threeImages.getHeight(null);
          imageWidth  = threeImages.getWidth(null)/3;

          images = new BufferedImage[3];
          for (int i=0; i<3; i++)
          {
             images[i] = threeImages.getSubimage(i*imageWidth,0,
                                                 imageWidth,imageHeight);
          }
       }

       maxX = width;
       maxY = height;

       x    = rng.nextInt(maxX);
       y    = rng.nextInt(maxY);

       this.speed        = speed;
       this.state        = 0;
       this.stateChange  = 1;
       this.ticksInState = 20 - 2*speed;
    }


    /**
     * Paint this Fish
     *
     * @param g   The rendering engine to use
     */
    public void paint(Graphics g)
    {
       ticks += 1;
       if (ticks > ticksInState)
       {
          ticks = 0;
          state += stateChange;
          if      (state == 2) stateChange = -1;
          else if (state == 0) stateChange =  1;
       }

       x += speed;

       if (x > maxX)
       {
          x     = INITIAL_LOCATION;
          y     = rng.nextInt(maxX);
       }

       if (images[state] != null) g.drawImage(images[state], x, y, null);
    }
}
```

A More Object-Oriented Example (cont.)

The **FishTank**

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.*;
import java.io.*;
import java.util.Random;
import javax.imageio.*;
import javax.swing.*;

/**
 * An example of animation
 *
 * @author  Prof. David Bernstein, James Madison University
```

```
 * @version 1.0
 */
public class FishTank extends    JComponent
                      implements ActionListener
{
    private BufferedImage      ocean;
    private Fish[]             fish;
    private Timer              timer;

    private static final Random  rng = new Random();


    /**
     * Default Constructor
     */
    public FishTank()
    {
       BufferedImage         threeImages;

       try
       {
          ocean        = ImageIO.read(new File("ocean.png"));
          threeImages  = ImageIO.read(new File("fish.png"));
       }
       catch (IOException ioe)
       {
          ocean       = null;
          threeImages = null;
       }

       fish = new Fish[5];
       for (int i=0; i<5; i++)
          fish[i] = new Fish(threeImages, 640, 480, rng.nextInt(3)+1);

       timer = new Timer(10, this);
       timer.start();
    }

    /**
     * Handle actionPerformed messages (required by ActionListener)
     *
     * @param  event   The ActionEvent that generated the message
     */
    public void actionPerformed(ActionEvent event)
    {
       repaint();
    }

    /*
     * Render this JComponent
     *
     * @param g   The rendering engine to use
     */
    public void paint(Graphics g)
    {
       Graphics2D           g2;


       g2 = (Graphics2D)g;

       if (ocean != null) g2.drawImage(ocean, 0, 0, null);

       for (int i=0; i<fish.length; i++) fish[i].paint(g);
    }
}
```

An Example of a "Nifty" Assignment

- The Important Concepts:

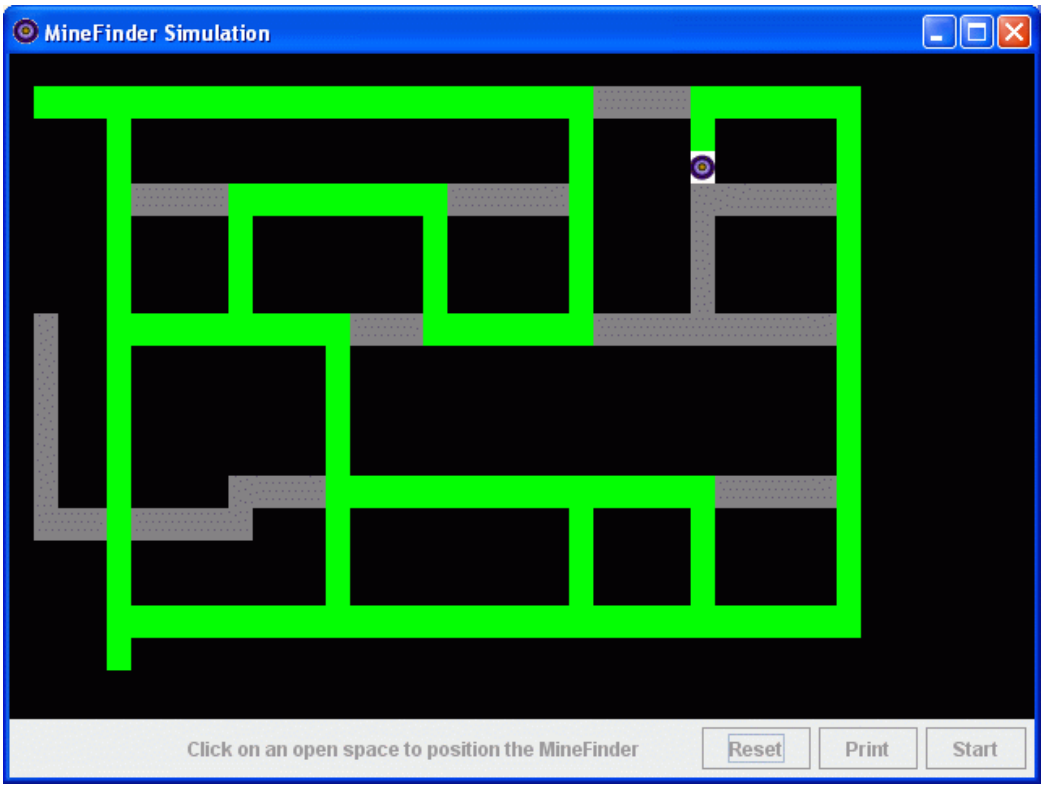
  Recursion

- The Setting - A Robot Minesweeper/Vacuum:

  Search for mines in a building/vacuum a building

- Making the Assignment More Engaging:

Instead of displaying the results on the console, animate the robot



The GUI for this "Nifty" Assignment

The Abstract Class that Students Must Extend

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

/**
 * An abstract encapsulation of a Robot simulator
 *
 * @author  Prof. David Bernstein, James Madison University
 * @version 1.0
 */
public abstract class Robot
{
    private boolean         show;
    private int             column, delay, row;
    private int[][]         map;
    private JFrame          window;
    private JLabel[][]      labels;


    private static final int WALL    = 0;
    private static final int OPEN    = 1;
    private static final int VISITED = 2;



    /**
     * Explicit Value Constructor
     *
     * @param show  true to include a visual simulation of this Robot
     */
    public Robot(boolean show)
    {
       this.show = show;
       delay = 10;
    }
```

```
/**
 * Can the Robot move (one "spot") in the given direction?
 *
 * @param d   The Direction of interest
 * @return    true/false if the Robot can/can't move
 */
protected boolean canMove(Direction d)
{
   return checkMapFor(WALL, d);
}


/**
 * Check the map for a WALL, OPEN, VISITED
 *
 * @param d  The Direction to check (relative to the current location)
 */
private boolean checkMapFor(int value, Direction d)
{
   boolean  result;
   int      i, j;

   i = row    + d.getDeltaY();
   j = column + d.getDeltaX();

   result = false;

   if (inBounds(i, j))
   {
      if (map[i][j] > value) result = true;
   }

   return result;
}



/**
 * Create a window that shows the behavior of this Robot
 */
private void createWindow()
{
   JComponent   contentPane;


   window = new JFrame();
   window.setTitle("Roombarch Simulation");
   window.setSize(400,400);
   contentPane = (JComponent)window.getContentPane();
   contentPane.setLayout(new GridLayout(map.length, map[0].length));

   labels = new JLabel[map.length][map[0].length];


   for (int i=0; i<map.length; i++)
   {
      for (int j=0; j<map[0].length; j++)
      {
         labels[i][j] = new JLabel(" ");
         labels[i][j].setOpaque(true);
         labels[i][j].setForeground(Color.RED);
         contentPane.add(labels[i][j]);

         if      (map[i][j] == WALL)
            labels[i][j].setBackground(Color.BLACK);
         else if (map[i][j] == OPEN)
            labels[i][j].setBackground(Color.GRAY);
      }
   }
   window.setVisible(true);

}


/**
 * Has the Robot already been to the location in the
 * given Direction?
 *
```

```
     * @param d  The Direction of interest (relative to the current location)
     * @return   true/false if the Robot has/hasn't been to the given Direction
     */
    protected boolean haveBeen(Direction d)
    {
       return checkMapFor(OPEN, d);
    }


    /**
     * Check to see if the point i,j is inside of the map
     *
     * @param i   The row
     * @param j   The column
     */
    private boolean inBounds(int i, int j)
    {
       boolean result;

       result = false;

       if ((i >= 0) && (j >= 0) &&
           (i < map.length) && (j < map[0].length)) result = true;


       return result;

    }



    /**
     * Move the Robot (one "spot") in the given Direction (if possible)
     *
     * @param d  The Direction to move in
     */
    protected void move(Direction d)
    {
       int      i, j;

       if (canMove(d))
       {
          if (show)
          {
             labels[row][column].setText(" ");
             labels[row][column].setBackground(Color.WHITE);
          }


          row    += d.getDeltaY();
          column += d.getDeltaX();
          map[row][column] = 2;

          if (show)
          {
             labels[row][column].setText("*");
             labels[row][column].setBackground(Color.WHITE);
             window.repaint();
          }

          try
          {
             Thread.sleep(delay);
          }
          catch (InterruptedException ie)
          {
          }
       }
    }


    /**
     * Turn the Robot on
     *
     * @param room  The file containing the simulated room
     */
    public void powerOn(String room) throws IOException
    {
       char                  c;
       File                  roomFile;
       int                   lineLength, maxLength;
```

```
      Scanner               scanner;
      String                line;
      Vector<String>        lines;

      maxLength = 0;
      lines = new Vector<String>();


      roomFile = new File(room);
      scanner  = new Scanner(roomFile);
      while (scanner.hasNext())
      {
         line = scanner.nextLine();
         lineLength = line.length();

         if (lineLength > maxLength) maxLength = lineLength;
         lines.add(line);
      }


      map = new int[lines.size()][maxLength];
      for (int i=0; i<map.length; i++)
      {
         for (int j=0; j<map[i].length; j++)
         {
            map[i][j] = OPEN;
            line = lines.elementAt(i);
            if (j < line.length())
            {
               c = line.charAt(j);
               if (c == '*')
               {
                  row    = i;
                  column = j;
                  map[i][j] = OPEN;
               }
               else if (c == '0')
               {
                  map[i][j] = WALL;
               }
            }
         }
      }
      if (show) createWindow();
   }


   /**
    * Print the map (in its current state)
    *
    * "0" indicates a wall, " " indicates a location that
    * has been visited, and "." indicates a location that
    * has not been visited
    */
   public void printMap()
   {
      for (int i=0; i<map.length; i++)
      {
         for (int j=0; j<map[0].length; j++)
         {
            if      (map[i][j] == WALL)    System.out.print("0");
            else if (map[i][j] == VISITED) System.out.print(" ");
            else if (map[i][j] == OPEN)    System.out.print(".");
         }
         System.out.println();
      }
   }


   /**
    * Set the delay (in milliseconds) been moves
    *
    * @param delay   The delay (in milliseconds)
    */
   public void setDelay(int delay)
   {
      this.delay = delay;
   }
```

```
    /**
     * Start cleaning/moving
     */
    public abstract void start();



    /**
     * Stop moving/cleaning
     */
    public void stop()
    {
       printMap();
       if (show) window.dispose();
    }

}
```

Another Example of a "Nifty" Assignment

- The Important Concepts:

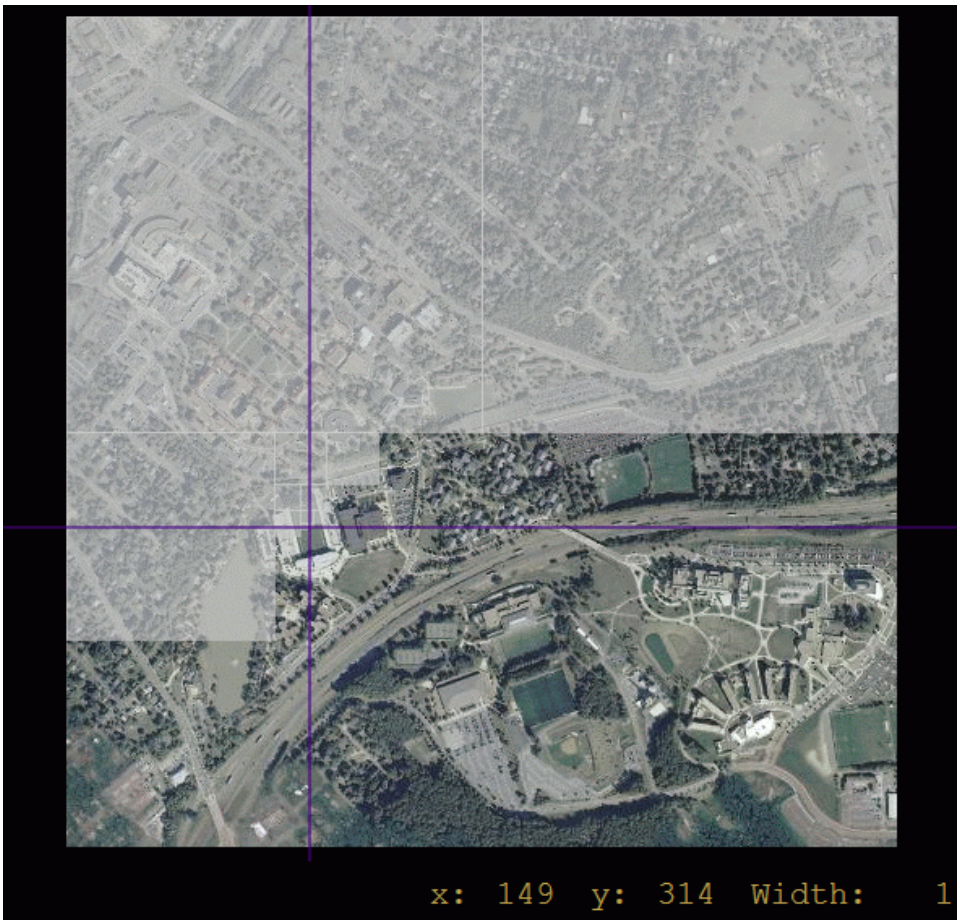
  Recursion

- The Setting - People in Purple:

  Search for aliens in and around JMU

- Making the Assignment More Engaging:

  Instead of displaying the results on the console, use animated crosshairs



Part 4

Other Examples

A Fancier GUI

- The Important Concepts:

  Objects

- The Setting:

  Scoring dives in a diving competition

- Making the Assignment More Engaging:

  Instead of using the console use a scorebaord

JMU Invitational

Name	Dive	Difficulty	Judge 0	Judge 1	Judge 2	Judge 3	Judge 4	Judge 5	Judge 6	Score
Flintstone, Wilma	103B	1.7	5.5	5.0	6.0	6.5	6.0	6.5	5.0	9.86
Rubble, Betty	405B	3.4	4.5	4.5	4.5	5.0	5.5	5.0	5.0	16.32

Another Fancier GUI

- The Important Concepts:

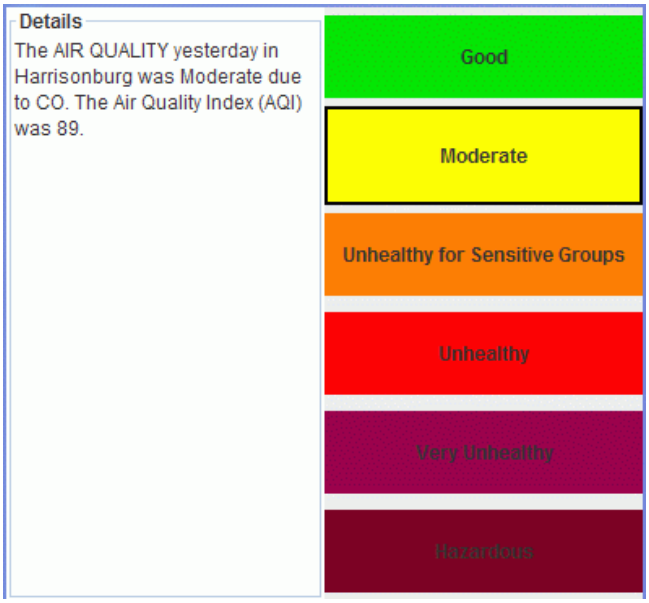
  Designing classes

- The Setting:

  An air quality monitoring system

- Making the Assignment More Engaging:

  Instead of using the console use a visual report



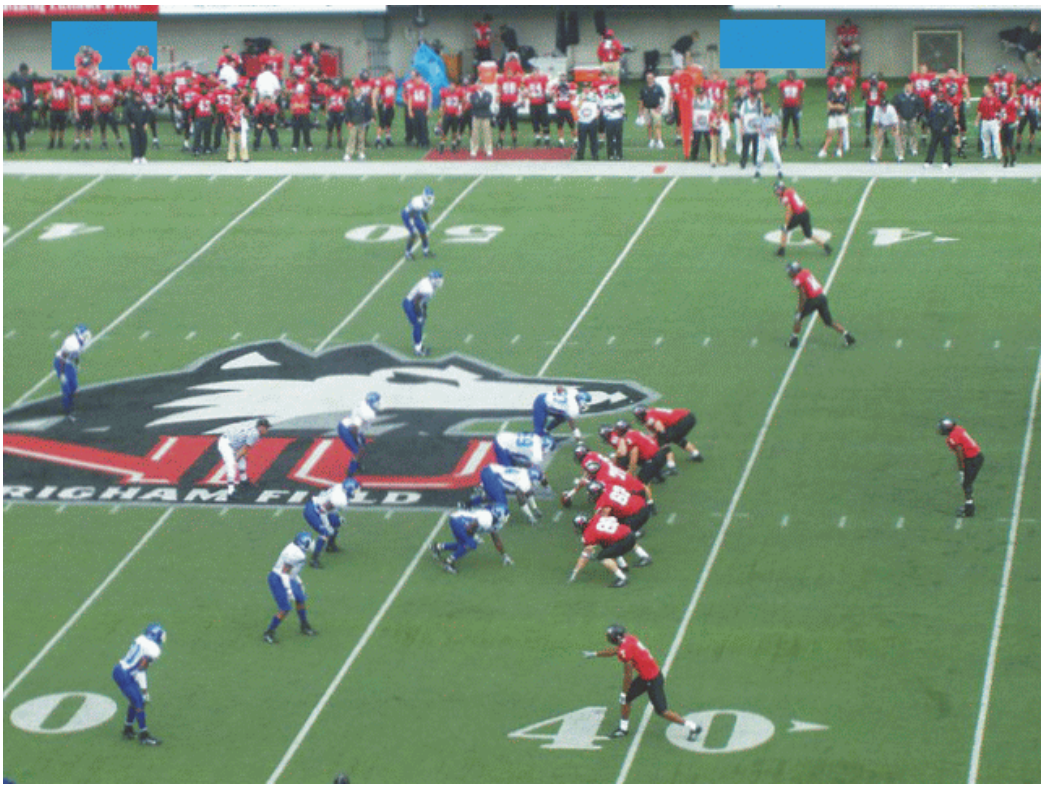
Using Bitmap Images

- The Important Concepts:

Two-dimensional arrays (i.e., arrays of arrays)

- Making the Assignment More Engaging:

  Add advertisiments to a television broadcast



- An Advanced Assignment:

  Add line-of-scrimmage and first-down lines