**Graph.java**

Author: Craig Jones

Class that allows the creation and display of the graphs of various geometric shapes.

public class **Graph** extends JPanel

**Overview**:

* + 1. This class contains 2 important elements:
       1. A JFrame - A window which will be the canvas for displaying various geometric shapes.
       2. A shapeList - a collection of lineList's. Each lineList is a collection of lines (defined by two endpoints) that describe a shape.
    2. This drawing works as follows:
       1. A JFrame is instantiated to create the display window.
       2. An image buffer (essentially a bitmap), which fits inside the JFrame window, is created to gather all the shapes which will be plotted.
       3. Once all the shapes have been rendered into the image buffer, the buffer is drawn into the JFrame.
    3. To use this class to display shapes:
       1. Instantiate a Graph object
       2. Create a lineList for each shape
       3. Add each lineList to the class shapeList
       4. Call PlotShapes() to plot the shapes into the JFrame window.

Note: PlotShapes() will analyze the extents of all the shapes to determine a scale factor to assure that all the shapes will fit into the JFrame window.

Note: The JFrame window uses a coordinate system which places the (0, 0) origin in the upper left corner of the window. The rendering code will transform the image so that the shapes will appear to be drawn in the more familiar Cartesian coordinate system with the (0, 0) origin in the lower left corner of the JFrame window.

Note: This class does not allow for negative coordinate values.

Note: Use the static main() method in this class definition as a template for displaying shapes. The “case DISPLAY:” code in isosceles.java also demonstrates how to use this functionality.

**Class Constants:**

int **TEXT\_EXTRA**: Used to add a little extra space at the bottom of the image buffer to allow for the "Scale

Factor: x" to be inserted.

int **FRAME\_SIZE**: Size of the JFrame window in pixels

int **FRAME\_OFFSET**: Number of pixels we will shift the image buffer, in both the X & Y directions, away

from the origin of the JFrame origin. This will prevent any part of the plotted shapes

from being plotted on the Frame window/ boundaries on all sides.

int **BUFFER\_SIZE**:The size of the image buffer.

**Public Variables:**

ArrayList<ArrayList<Line>> **shapeList**: A list of shapes (a collection of Line.lineList's) to be plotted into the

graph.

**Private Instance Variables:**

JFrame **frame**: The JFrame instance window into which the shapes will be plotted.

double **xMin, yMin, xMax, yMax**: The graphic extents of the entire collection of shapes.

double **scaleFactor**: The scale factor to be used to cause the entire group of shapes to fit within the confines of

the image buffer.

**Constructors:**

public **Graph**()

The default constructor is used.

public void **InitializeGraph**(String **shapeName**, Graph **graph**)

**shapeName** - The text to be shown in the title bar of the JFrame window.

**Graph** - Class instance reference obtained from a call to the Graph constructor.

While technically not a constructor, this is an extension to the default Graph constructor that does the actual initialization of the new graph context.

*Author’s Note: The reason the initialization is done in this method rather than in the Graph constructor is that the call to the frame.getContentPane().add(graph) method requires a reference to the graph object and I couldn't figure out how to pass the reference to the graph object while the object was still being constructed (using "this" doesn't seem to be legal in this situation). Kind of quirky I know but I'll see if I can fix it if I have time.*

**Public Methods:**

void **PlotShapes**()

Applies all the transformations to scale and position the shape coordinates in the image buffer and then renders them into the JFrame window.

@Override void **paint**(Graphics **g**)

**g** – Reference to a Graphics class instance obtained from the BufferedImage class instance.

Method invoked to render the image buffer into the JFrame window. It is called implicitly by the JFrame class instance as a result of a call to the JFrame setVisible(true) method.

**Private Methods:**

void **determineScaleFactor**()

Determines the best scale factor to use to map the shapes into the image buffer.

void **RefactorLines**()

Scale the line coordinates so that the plot of the shapes will fill the extent of the image buffer.

void **GetExtents(**Line **line**)

line – Reference to the Line object to be examined.

Checks the specified line against the current extent values and adjusts the extent values as needed.

**Private Methods (continued):**

Image **CreateImage**()

Returns a reference to an image buffer Image instance into which all the shapes from the Graph shapeList have been rendered.

int **doubleToInt**(double **value**, int **decimalPlaces**)

**value** – Double value to be converted to integer.

**decimalPlaces** – Number of places behind the decimal point to round the specified double value.

Converts the specified double precision number to integer after rounding it to the specified number of decimal places behind the decimal point.

**Mutators and Accessors:**

private void **SetShapeList**(ArrayList<ArrayList<Line>> shapeList)

public ArrayList<ArrayList<Line>> **GetShapeList**()

private void **SetJFrame**(JFrame **frame**)

private JFrame **GetJFrame**()

private void **SetScaleFactor**(double **scaleFactor**)

private double **GetScaleFactor**()

private void **SetXMin**(double **xMin**)

private double **GetXMin**()

private void **SetXMax**(double **xMax**)

private double **GetXMax**()

private void **SetYMin**(double **yMin**)

private double **GetYMin**()

private void **SetYMax**(double **yMax**)

private double **GetYMax**()

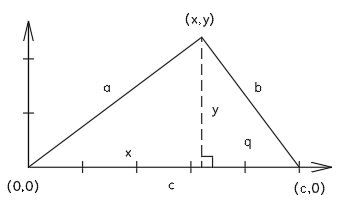
**Public Static Methods:**

public static **void PlotTriangleFromLineLengths**(double **A**, double **B**, double **C**)

A, B,C - Lengths of the three legs of a triangle

Given the lengths of the three legs of a triangle, plot the triangle. A discussion of the algorithm used follows:

Given the known lengths of three sides of a triangle, a, b, and c, a valid set of coordinates describing that triangle can be ascertained. (Constraint: the sum of the two shortest sides have to be greater than the length of the longest side, otherwise the sides cannot describe a closed triangular figure).



Consider the above triangle. By drawing a perpendicular line from the base to the apex, two right triangles are created. Note that q = c – x.

Using the Pythagorean theorem we get the following two equations:

and

Solving for in the first equation in terms of *x* yields:

Substituting and for in the second equation yields:

Expanding the term gives:

And simplifying the equation gives:

Moving the non-x terms to the right side of the equation yields:

Multiplying both sides by -1:

And solving for *x* gives an equation in which we can now use the given values *a*, *b*, and *c* to derive the value of *x*:

We can now solve for the value of *y* by substituting *x* back into the derived equation for :

=>

By arranging the side such that the longest leg coincides with the X axis, we can also deal with scalene triangles.

**Test Interface/Sample:**

This static main() method allows the Graph.java file to be run as a separate program via the Debug/Debug File option in NetBeans.

It can be used as a template for drawing various shapes.

import app.shapes.Graph;

import app.shapes.Line;

import app.shapes.Point;

import java.util.ArrayList;

public static void main(String[] args) {

// instantiate a new graph object

Line line;

ArrayList<Line> lineList;

Point pt1, pt2;

Graph graph;

// Instantiate a graph object and initialize it

graph = new Graph();

graph.InitializeGraph("Isosceles Triangles", graph);

// Create a new line list for the polygon

lineList = Line.CreateLineList();

// Create a new line object from two points

// Note:Tthe two points are deep copied into the line object by the

// Line(pt1, pt2) constructor.

pt1 = new Point(0.0, 0.0);

pt2 = new Point(2.07055, 0.0);

line = new Line(pt1, pt2);

lineList.add(line);

// Create a 2nd line to add to the line list (the 1st point is the same

// as above)

pt2.setPointX(1.035276);

pt2.setPointY(3.8637);

line = new Line(pt1, pt2);

lineList.add(line);

// Create a 3rd line to add to the line list (the 2nd point is the same

// as above)

pt1.setPointX(2.07055);

pt1.setPointY(0.0);

line = new Line(pt1, pt2);

lineList.add(line);

// Add the lineList to the shapeList

// Note: Outside of this class, you will need to use

// GetShapeList().add(lineList) instead of this below:

graph.shapeList.add(lineList);

// Adding a second new polygon

lineList = Line.CreateLineList();

pt1 = new Point(9.0, 8.0);

pt2 = new Point(13.0, 5.0);

line = new Line(pt1, pt2);

lineList.add(line);

pt2.setPointX(17.0);

pt2.setPointY(8.0);

line = new Line(pt1, pt2);

lineList.add(line);

pt1.setPointX(13.0);

pt1.setPointY(5.0);

line = new Line(pt1, pt2);

lineList.add(line);

graph.shapeList.add(lineList);

// Plot the shapes to the JFrame window.

graph.PlotShapes();

// Plot the triangle from the lengths of 3 lines using the public static interface.

PlotTriangleFromLineLengths(5.0, 3.0, 4.0);

} // main()