Joel Edwards
Course: Java Programming 1
Homework 3
April 9, 2011

1)

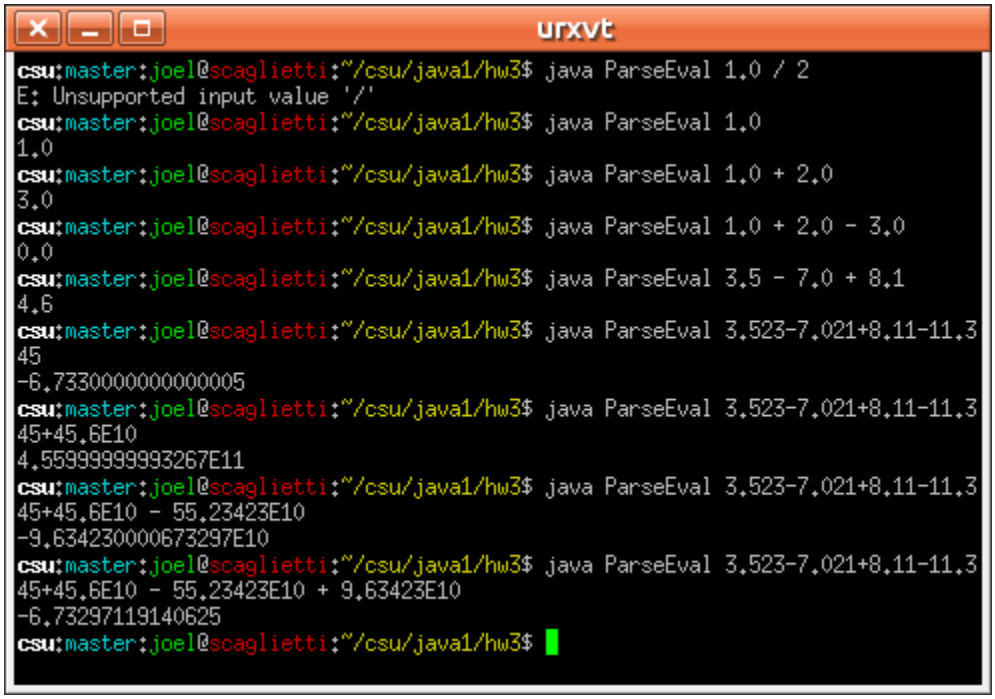
Source:

ParseEval.java:

```
import java.util.ArrayList;
import java.util.StringTokenizer;

class ParseEval
{
    public static final int NONE = 0;
    public static final int ADD = 1;
    public static final int SUBTRACT = 2;

    private static void error(String message) {
        System.out.println("E: " + message);
        System.exit(1);
    }

    public static void main(String[] args) {
        StringTokenizer tokenizer;
        ArrayList<String> tokens = new
ArrayList<String>(args.length);

        // Look for tokens within each token returned from the
command-line
        for (String arg: args) {
            tokenizer = new StringTokenizer(arg, "-+", true);
            while (tokenizer.hasMoreTokens()) {
                tokens.add(tokenizer.nextToken());
            }
        }

        boolean last_was_value = false;
        int     last_delimiter = ADD;
        double  value = 0;
        double  total = 0;
        int     token_index = 1;
        for (String token: tokens) {
            if ("+".compareTo(token) == 0) {
                if (!last_was_value) {
```

```
                    error("Adjacent operators are not allowed");
                }
                last_delimiter = ADD;
                last_was_value = false;
            } else if ("-".compareTo(token) == 0) {
                if (!last_was_value) {
                    error("Adjacent operators are not allowed");
                }
                last_delimiter = SUBTRACT;
                last_was_value = false;
            } else {
                try {
                    value = Double.parseDouble(token);
                    if (last_was_value) {
                        error("Adjacent values are not allowed");
                    }
                    last_was_value = true;
                    if (last_delimiter == SUBTRACT) {
                        total -= value;
                    } else if (last_delimiter == ADD) {
                        total += value;
                    } else {
                        // This should never occur
                        error("Unsupported arithmetic operation");
                    }
                } catch (NumberFormatException e) {
                    error("Unsupported input value '" +token+ "'");
                }
            }
            token_index++;
        }

        System.out.println("" + total);
    }
}
```

Output:



2)

Source:

DrawT.html:

```
<html>
    <head>
        <title>A Simple Program</title>
        <meta http-equiv="pragma" content="no-cache" />
    </head>
    <body>
        Here is the output of my program:
        <applet code="DrawTApplet.class" width="300" height="400">
        </applet>
    </body>
</html>
```

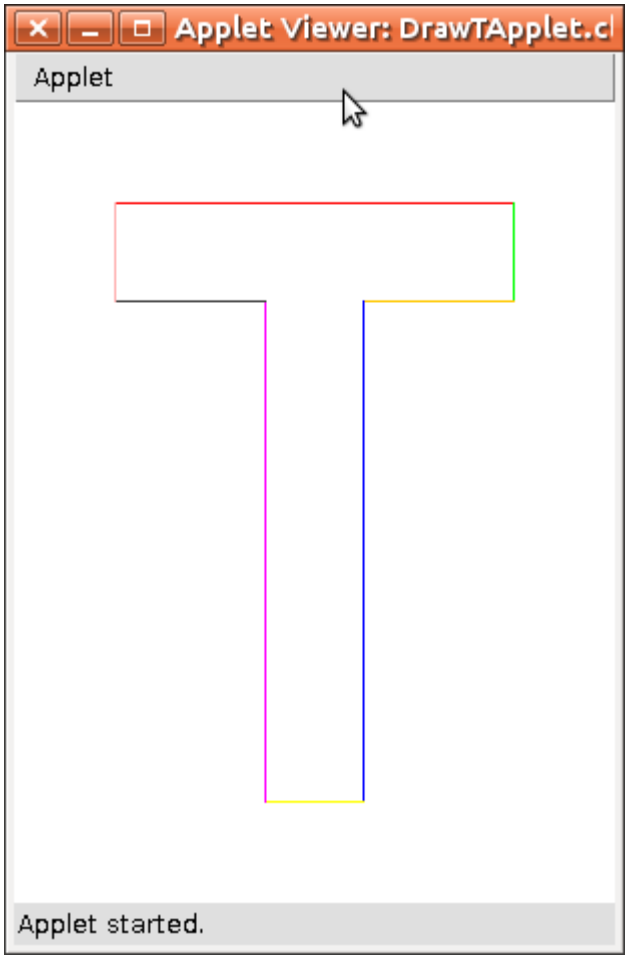
DrawTApplet.java:

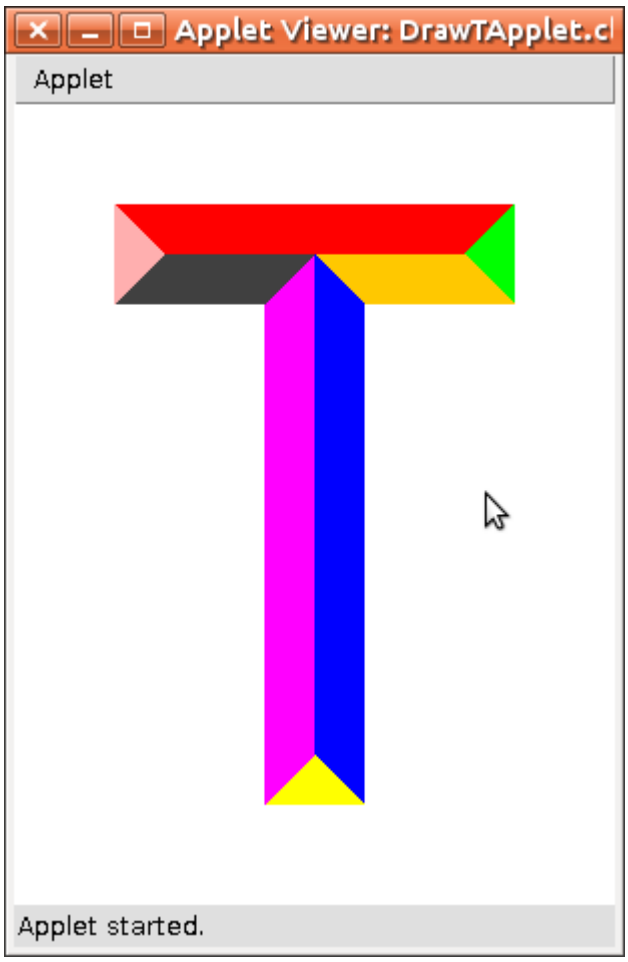
```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;

public class DrawTApplet
    extends Applet
{
    private static final int NEG = -1;
```

```
    private static final int POS = 1;

    private Graphics g = null;

    public void paint(Graphics g) {
        this.g = g;
        this.resize(300, 400);
        this.setBackground(Color.white);

        // Draw line segments
        this.drawPyramid(Color.RED,        1,  50, POS,  50, POS,
249, NEG,  50, POS);
        this.drawPyramid(Color.GREEN,      1, 249, NEG,  50, POS,
249, NEG,  99, NEG);
        this.drawPyramid(Color.ORANGE,     1, 249, NEG,  99, NEG,
174, NEG,  99, NEG);
        this.drawPyramid(Color.BLUE,       1, 174, NEG,  99, NEG,
174, NEG, 349, NEG);
        this.drawPyramid(Color.YELLOW,     1, 174, NEG, 349, NEG,
125, POS, 349, NEG);
        this.drawPyramid(Color.MAGENTA,    1, 125, POS, 349, NEG,
125, POS,  99, NEG);
        this.drawPyramid(Color.DARK_GRAY,  1, 125, POS,  99, NEG,
50, POS,  99, NEG);
        this.drawPyramid(Color.PINK,       1,  50, POS,  99, NEG,
50, POS,  50, POS);
    }

    public void drawLine(Color c, int x1, int y1, int x2, int y2) {
        this.g.setColor(c);
        this.g.drawLine(x1, y1, x2, y2);
    }

    public void drawPyramid(Color c, int depth,
                            int x1, int dx1,
                            int y1, int dy1,
                            int x2, int dx2,
                            int y2, int dy2) {
       for (int i = 0; i < depth; i++) {
           this.drawLine(c,
                         x1 + (dx1 * i),
                         y1 + (dy1 * i),
                         x2 + (dx2 * i),
                         y2 + (dy2 * i));
       }
    }
}
```

Output:

Single pixel lines were a little too thin, so I created the drawPyramid method in order to draw the T with line thickness determined by the *depth* parameter. The results were pretty interesting for a depth of 25 (second screenshot).





3)
String's *split* method(s) can quickly breaking up a String into an array of tokens based on more complex delimiters than those allowed by StringTokenizer. This would be useful for parsing log files.

4)
Graphics' *clearRect* method is useful for clearing portions of the Graphics' canvas, or the entire canvas. This proved useful in problem 5 when re-drawing the T.

5)

Source:

DrawTResize.html:

```
<html>
    <head>
        <title>A Simple Program</title>
        <meta http-equiv="pragma" content="no-cache" />
    </head>
    <body>
        Here is the output of my program:
        <applet code="DrawTResizeApplet.class" width="300"
height="400">
        </applet>
    </body>
</html>
```

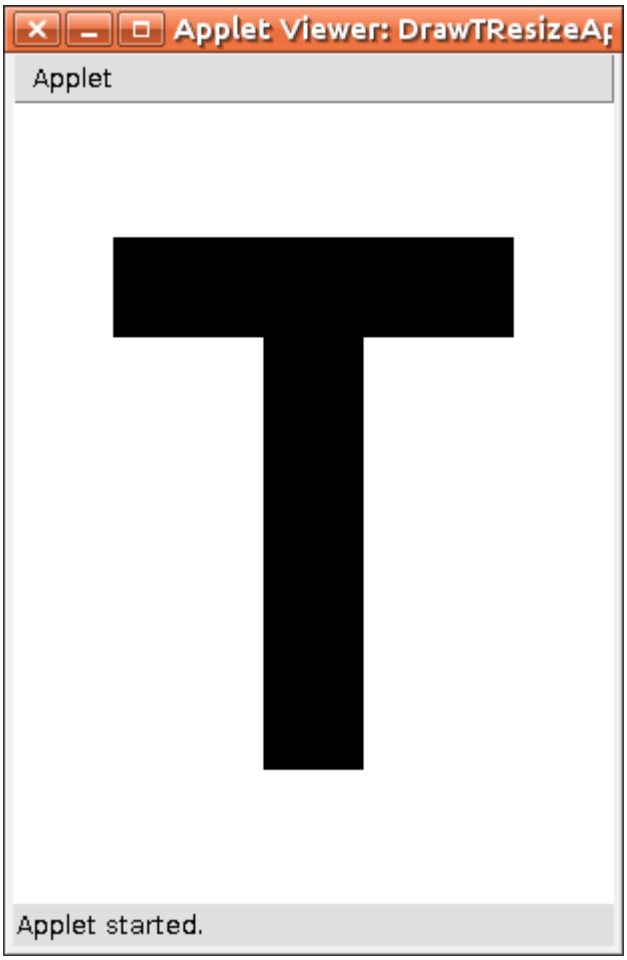
DrawTResizeApplet.java:

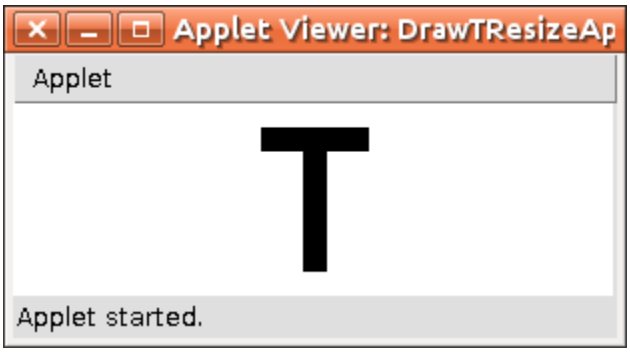
```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Container;
import java.awt.Dimension;
import java.awt.event.ComponentListener;
import java.awt.event.ComponentEvent;
import java.awt.Graphics;
import javax.swing.JApplet;
import javax.swing.JFrame;

public class DrawTResizeApplet
    extends JApplet
{
    private int w_div = 6; // Divide width by this value to determine
width of stem of the T
    private int h_div = 8; // Divide height by this value to
determine height of cross of the T

    public static void main(String[] args) {
        JFrame frame = new Jframe("Draw T");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setPreferredSize(new Dimension(300, 400));
        frame.setMinimumSize(new Dimension(30,40));

        JApplet applet = new DrawTResizeApplet();
        applet.init();
        applet.start();
        frame.add("Center", applet);
        frame.pack();
```

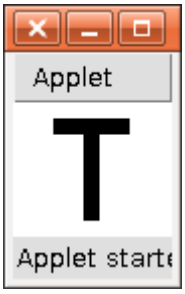
```
        frame.setVisible(true);
    }

    public void init() {
        // Here we handle the re-sizing of the Applet, and ignore all
other
        // events returned by the ComponentListener
        this.addComponentListener(new ComponentListener() {
            public void componentHidden(ComponentEvent e) { ; }
            public void componentMoved(ComponentEvent e) { ; }
            public void componentResized(ComponentEvent e) {
                ((DrawTResizeApplet)e.getComponent()).repaint();
            }
            public void componentShown(ComponentEvent e) { ; }
        });
    }

    public void start() {
        this.repaint(); // Draw the initial T prior to any resize
events
    }

    public void paint(Graphics g) {
        this.setBackground(Color.WHITE);
        g.setColor(Color.BLACK);
        int width  = this.getWidth();
        int height = this.getHeight();

        // We use these ratios to determine whether the T's dimension
should
        // be calculated with respect to width or height.
        double targetRatio = (double)(this.w_div) / (double)
(this.h_div);
        double dimRatio = (double)(width) / (double)(height);

        int top_x; // x-coordinate of upper left corner of the T's
cross
        int top_y; // y-coordinate of upper left corner of the T's
cross
        int top_w; // width of T's cross
        int top_h; // height (thickness) of T's cross
        int cnt_x; // x-coordinate of the upper left corner of the
T's stem
        int cnt_y; // y-coordinate of the upper left corner of the
T's stem
        int cnt_w; // width (thickness) of the T's stem
        int cnt_h; // height of the T's stem

        // Calculate dimensions with respect to height
```

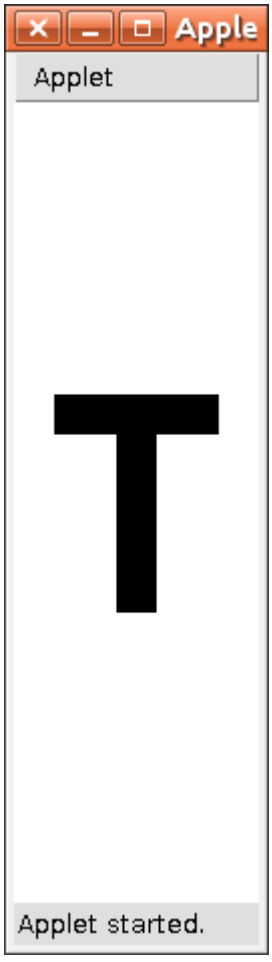
```
        if (dimRatio > targetRatio) {
            top_y = min_one(height / h_div);
            top_h = min_one(top_y);
            cnt_y = min_one(top_y);
            cnt_h = min_one(height - ((top_y) * 2));

            cnt_w = min_one(top_h);
            cnt_x = min_one(width / 2 - (cnt_w / 2));
            top_w = min_one(cnt_h * w_div / h_div);
            top_x = min_one(width / 2 - (top_w / 2));
        }
        // Calculate dimensions with respect to width
        else {
            top_x = min_one(width  / w_div);
            top_w = min_one(width - ((top_x) * 2));
            cnt_x = min_one((width / 2) - ((top_x) / 2));
            cnt_w = min_one(top_x);

            cnt_h = min_one(top_w * h_div / w_div);
            cnt_y = min_one(height / 2 - (cnt_h / 2));
            top_h = min_one(cnt_w);
            top_y = min_one(cnt_y);
        }

        g.clearRect(0, 0, width, height); // Remove previous T image
        g.fillRect(top_x, top_y, top_w, top_h); // Draw T's cross
        g.fillRect(cnt_x, cnt_y, cnt_w, cnt_h); // Draw T's stem
    }

    private static int min_one(int value) {
        return (value < 1) ? 1 : value;
    }
}
```

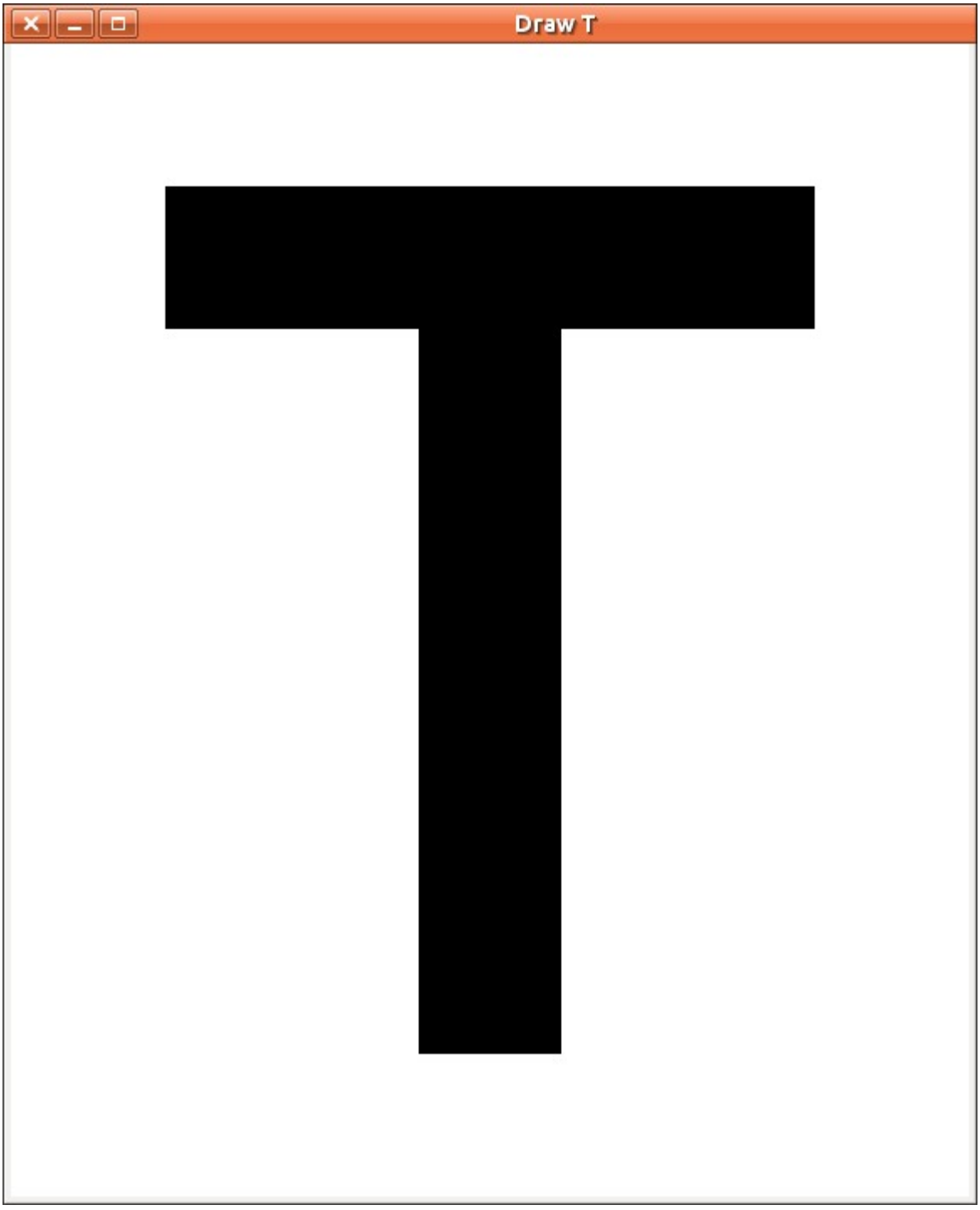
Output:
The applet worked well when making the T smaller. However, I could not find a way to increase the size of the drawing area when the applet viewer's size was increased (bottom applet screenshot). I set up the program such that it could operate as an application (final screenshot) to confirm the size increase was simply an applet viewer issue. The application appears to behave as desired.










Draw T