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

1)

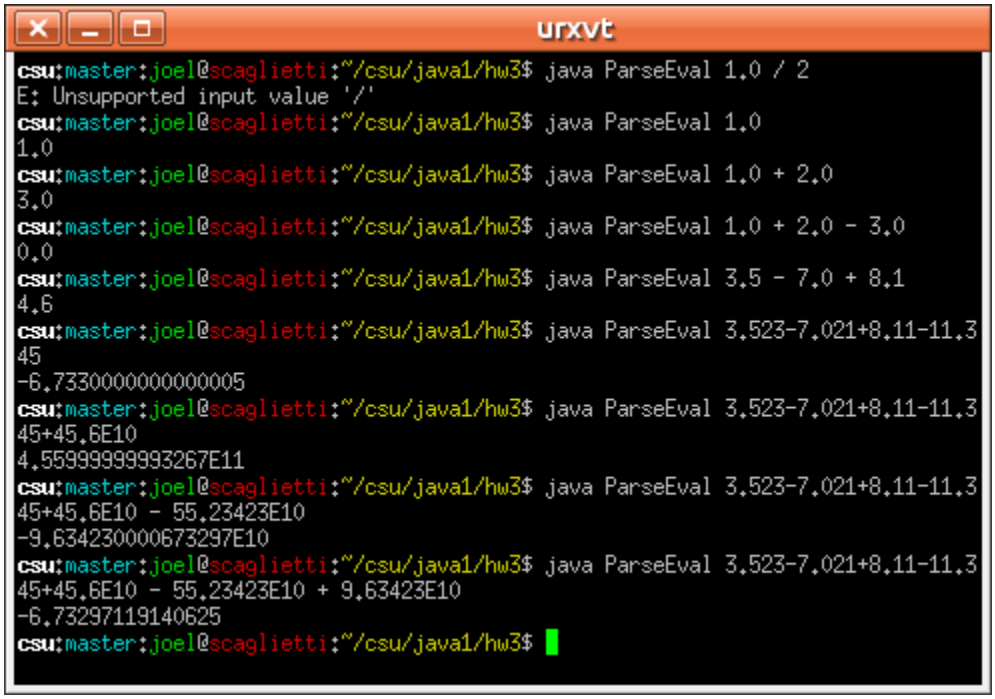
Source:

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>
```

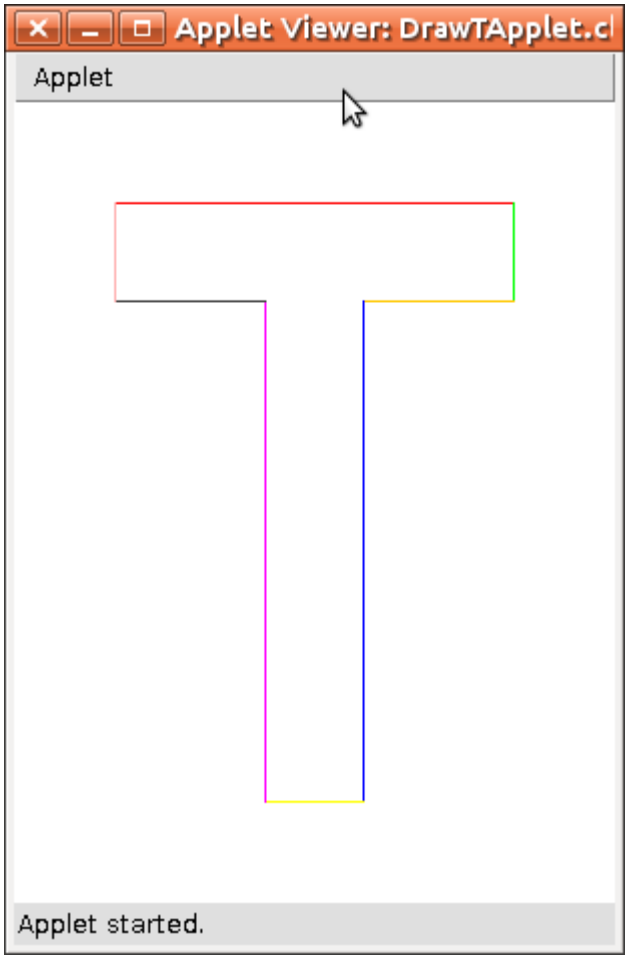
DrawTApplet.java:

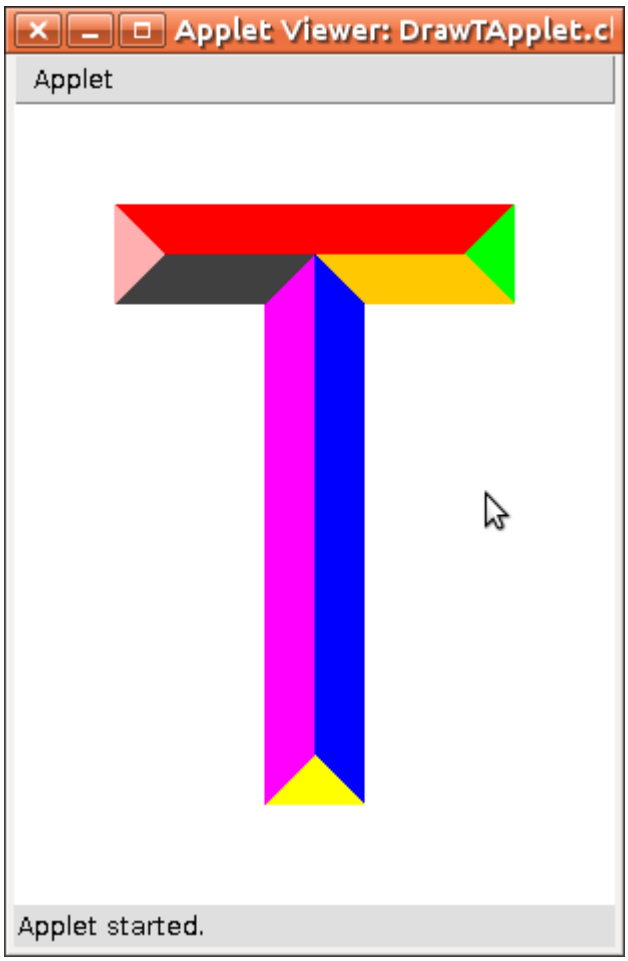
```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Graphics;

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.

4)
Graphics' *clearRect* method is useful for clearing portions of the Graphics' canvas, or the entire canvas.

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>
```

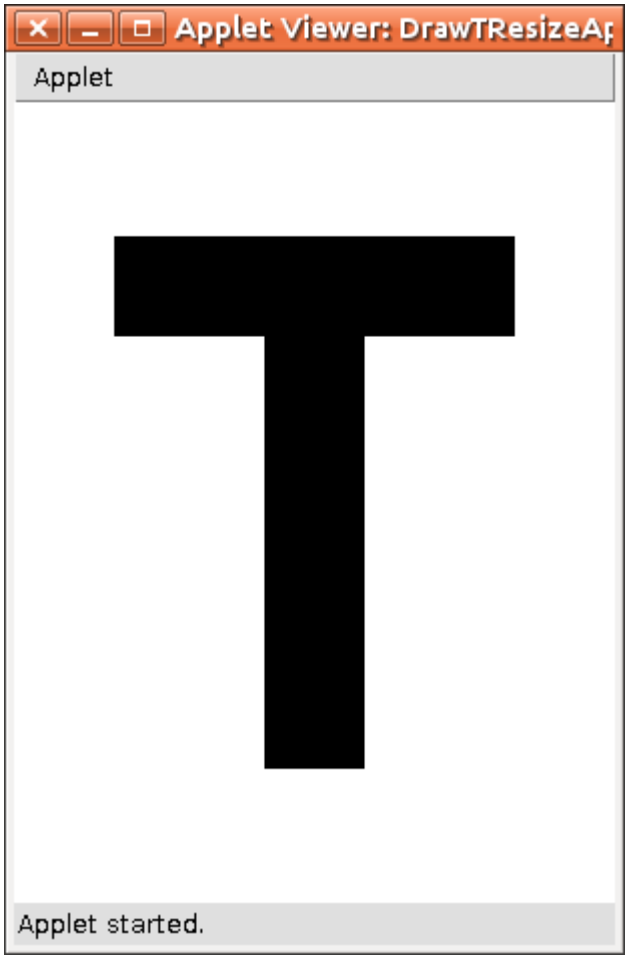
DrawTResizeApplet.java:

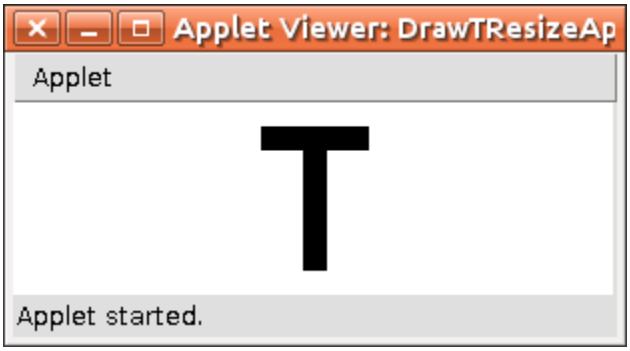
```
import java.applet.Applet;
import java.awt.Color;
import java.awt.Container;
import java.awt.event.ComponentListener;
import java.awt.event.ComponentEvent;
import java.awt.Graphics;

public class DrawTResizeApplet
    extends Applet
{
    private   Graphics g      = null;
    private   int      w_div  = 6; // Divide width by this value to
determine width of stemp of the T
    private   int      h_div  = 8; // Divide height by this value to
determine height of cross of the T

    public void paint(Graphics g) {
        this.g = g;

        this.setBackground(Color.WHITE);
        this.g.setColor(Color.BLACK);

        // 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) {
```

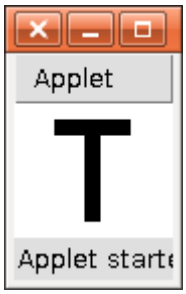
```
                DrawTResizeApplet a =
(DrawTResizeApplet)e.getComponent();
                a.redrawT();
            }
            public void componentShown(ComponentEvent e) { ; }
        });

        this.redrawT(); // Draw the initial T prior to any resize
events
    }

    public void redrawT() {
        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 = (float)(this.w_div) / (float)
(this.h_div);
        double dimRatio = (float)(width) / (float)(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 = height / h_div;
            top_y = (top_y < 1) ? 1 : top_y;
            top_h = top_y;
            top_h = (top_h < 1) ? 1 : top_h;
            cnt_y = top_y;
            cnt_y = (cnt_y < 1) ? 1 : cnt_y;
            cnt_h = height - ((top_y) * 2);
            cnt_h = (cnt_h < 1) ? 1 : cnt_h;

            cnt_w = top_h;
            cnt_w = (cnt_w < 1) ? 1 : cnt_w;
            cnt_x = width / 2 - (cnt_w / 2);
```

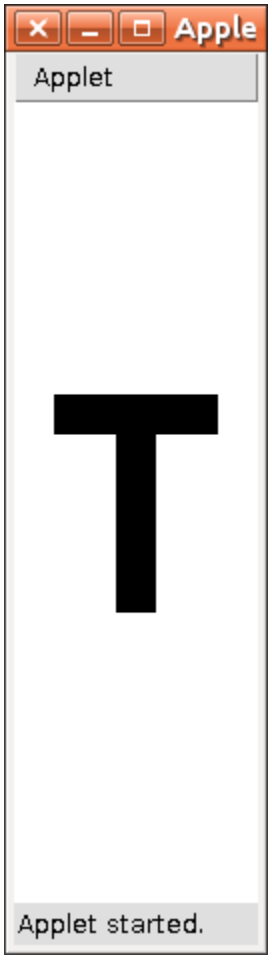
```
            cnt_x = (cnt_x < 1) ? 1 : cnt_x;
            top_w = cnt_h * w_div / h_div;
            top_w = (top_w < 1) ? 1 : top_w;
            top_x = width / 2 - (top_w / 2);
            top_x = (top_x < 1) ? 1 : top_x;
        }
        // Calculate dimensions with respect to width
        else {
            top_x = width  / w_div;
            top_x = (top_x < 1) ? 1 : top_x;
            top_w = width - ((top_x) * 2);
            top_w = (top_w < 1) ? 1 : top_w;
            cnt_x = (width / 2) - ((top_x) / 2);
            cnt_x = (cnt_x < 1) ? 1 : cnt_x;
            cnt_w = top_x;
            cnt_w = (cnt_w < 1) ? 1 : cnt_w;

            cnt_h = top_w * h_div / w_div;
            cnt_h = (cnt_h < 1) ? 1 : cnt_h;
            cnt_y = height / 2 - (cnt_h / 2);
            cnt_y = (cnt_y < 1) ? 1 : cnt_y;
            top_h = cnt_w;
            top_h = (top_h < 1) ? 1 : top_h;
            top_y = cnt_y;
            top_y = (top_y < 1) ? 1 : top_y;
        }

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

Output:


Applet Viewer: DrawTResizeAp
Applet
Applet started.


Applet Viewer: DrawTResizeAp
Applet
Applet started.


Applet
Applet starte


Apple
Applet
Applet started.