VidWorks Entertainment New Developer Training Program
Tutorial #1

Applets

NOTE: This tutorial series assumes that you know the basics of Java.

Intro to Applets

We have all seen the first program in DOS text mode...

```
/* Hello.java */

public class Hello {
      public static void main(String [] args) {
            System.out.println("Hello, World!");
      }
}
```

...but how would you like to see the classic "Hello, World!" in graphic mode!

```
/* HelloApplet.java */

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

public class HelloApplet extends Applet {
      public void paint(Graphics g) {
            g.drawString("Hello, World!", 30, 30);
      }
}
```

Unfortunately, you can't run this with the traditional "`java HelloApplet`" command. This is an applet, after all, not an application. Instead, we will embed this applet in an HTML document using the following tag:

```
<applet code="HelloApplet.class" width=300 height=200>
</applet>
```

You can open this HTML file in any browser that supports the Java plug-in to view the applet, or you can use "`appletviewer <HTML file>`", where `<HTML file>` is the name of the file that has the `applet` tag. Here's what your applet should look like:

Now, to tell you how this applet works:

`extends Applet` - all Applets are derived from the superclass `Applet`

`public void paint(Graphics g)` - "painting" is a term that means to fill in your applet with graphics. It is called every time your applet needs to be drawn or redrawn. The `Graphics` object is the object that you give commands to create various shapes, pictures or words.

`g.drawString("Hello, World!", 30, 30)` - the `drawString()` method invoked on the `Graphics g` object. The general form of this method is `drawString(String str, int x, int y)`. Thus, we see that this line draws the text "Hello, World!" at (30, 30).

NOTE: Coordinates are measured with the origin at the top-right corner of the screen, with x increasing to the right and y increasing downward.

Drawing Shapes

Hopefully, by now, you should understand how your first applet works. However, that's now all that applets can do. Here are other methods on the Graphics object that you can use to draw other things:

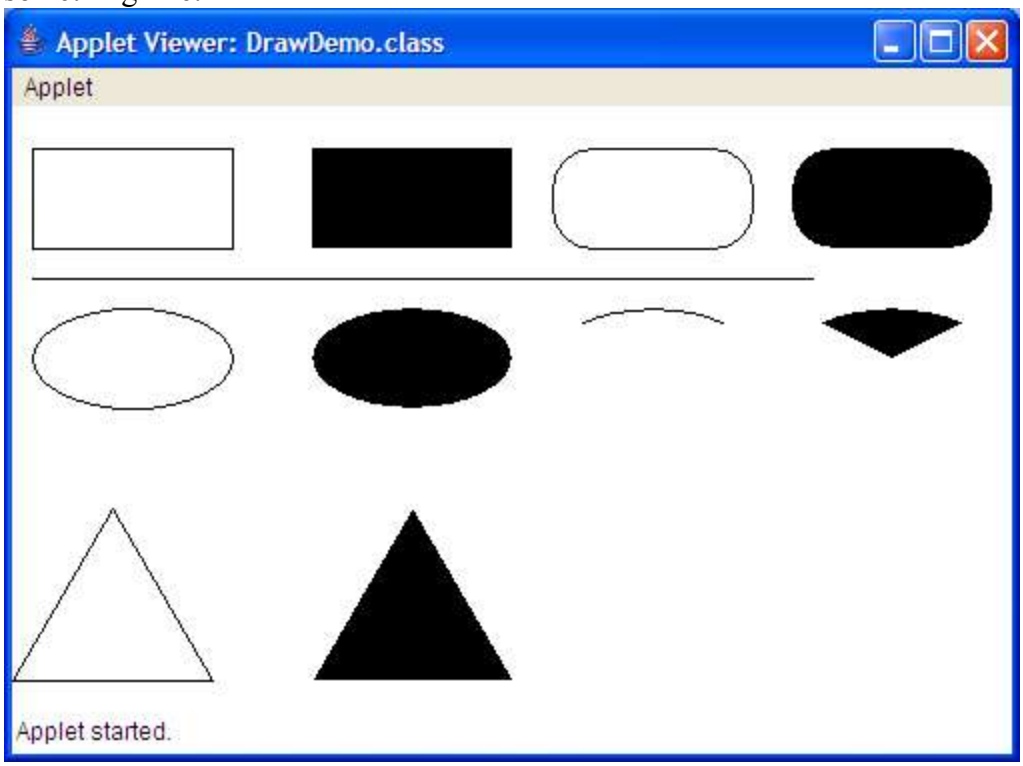
```
void drawRect(int x, int y, int width, int height)
void fillRect(int x, int y, int width, int height)
void drawRoundRect(int x, int y, int width, int height, int
 arcWidth, int arcHeight)
void fillRoundRect(int x, int y, int width, int height, int
 arcWidth, int arcHeight)
```

```
void draw3DRect(int x, int y, int width, int height,
 boolean raised)
void fill3DRect(int x, int y, int width, int height,
 boolean raised)
void drawOval(int x, int y, int width, int height)
void fillOval(int x, int y, int width, int height)
void drawArc(int x, int y, int width, int height, int
 startAngle, int arcAngle)
void fillArc(int x, int y, int width, int height, int
 startAngle, int arcAngle)
void drawPolygon(int[] xPoints, int[] yPoints, int nPoints)
void fillPolygon(int[] xPoints, int[] yPoints, int nPoints)
void drawLine(int x1, int y1, int x2, int y2)
void drawPolyline(int[] xPoints, int[] yPoints, int nPoints)
void drawString(String str, int x, int y)
```

Most of these functions should be self-explanatory. If not a little experimentation should solve any and all questions. In any case, here's some sample code to show off some of these functions:

```
/* DrawDemo.java */

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

public class DrawDemo extends Applet {
      public void paint(Graphics g) {
            g.drawRect(10, 20, 100, 50);
            g.fillRect(150, 20, 100, 50);
            g.drawRoundRect(270, 20, 100, 50, 40, 40);
            g.fillRoundRect(390, 20, 100, 50, 40, 40);
            g.drawLine(10, 85, 400, 85);
            g.drawOval(10, 100, 100, 50);
            g.fillOval(150, 100, 100, 50);
            g.drawArc(270, 100, 100, 50, 45, 90);
            g.fillArc(390, 100, 100, 50, 45, 90);
            int x[] = {0, 50, 100};
            int y[] = {286, 200, 286};
            g.drawPolygon(x, y, 3);
            int x2[] = {150, 200, 250};
            g.fillPolygon(x2, y, 3);
      }
}
```

Running this in a 500x300 applet HTML object will get you output that looks something like:



Color

This is great and all, but it seems kinda... monochrome. So, let's jazz it up with some color!

Java has a class in the AWT package called `Color`, which supports many different colors. It comes with tons of predefined colors that you can access by calling `Color.black`, `Color.red`, `Color.green`, and the like.

In addition, `Color` has 2 constructors:

```
Color(int red, int green, int blue)
Color(float red, float green, float blue)
```

Note that for the integer Color constructor, each color only goes from 0 to 255. Using combinations of these, we can make millions of colors!

```
Color green = new Color(0, 255, 0);
Color purple = new Color(255, 0, 2550);
Color pink = new Color(255, 175, 175);
```

Two more constructors give us the alpha channel, aka, the transparency channel:

```
Color(int red, int green, int blue, int alpha)
Color(float red, float green, float blue, float alpha)
```

The lower the alpha channel, the more transparent.
Now, in order to use these colors, we'll need a few new functions:

`setBackground(Color bgcolor)` - this sets the background color of the applet; this is a member function of the `Applet` class

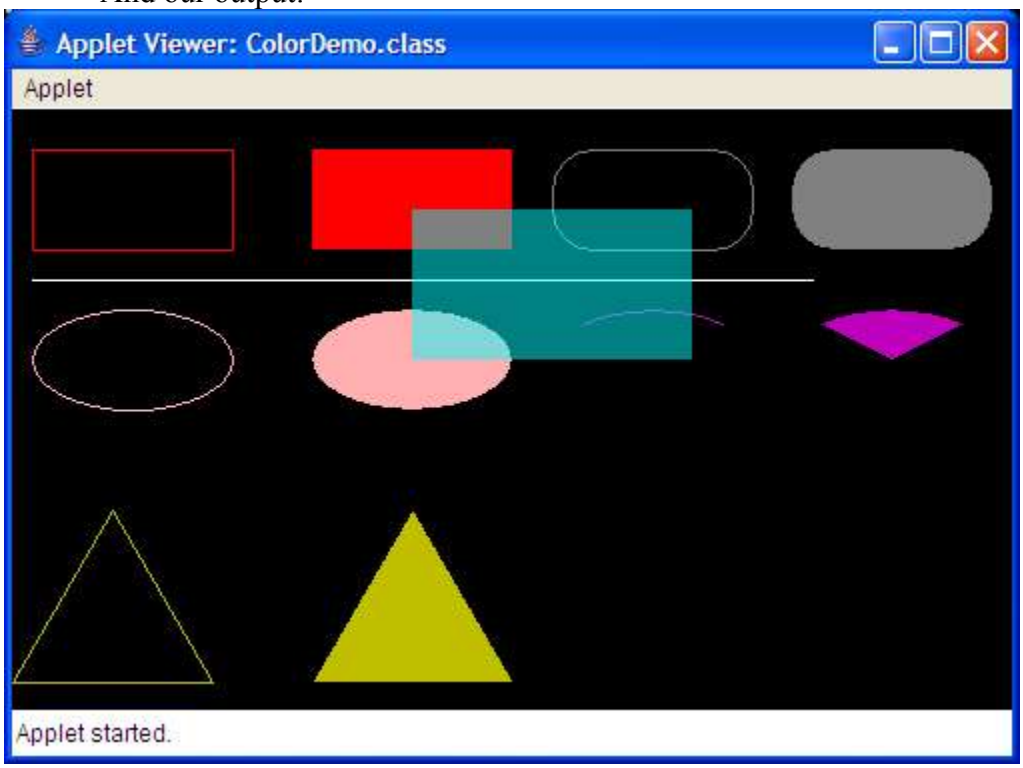
`setColor(Color color)` - this sets the current color to the given color. *All* subsequent draw or fill commands will use this color; this is a member function of the `Graphics` class

Now, let's use these methods to Color up our drawing demo:

```
/* ColorDemo.java */

import java.awt.*;
import java.applet.Applet;

public class ColorDemo extends Applet {
      Color aqua = new Color(0.75f, 0.0f, 0.75f);
      Color darkyellow = new Color(0.75f, 0.75f, 0.0f);
      Color cellophane = new Color(0.0f, 1.0f, 1.0f, 0.5f);

      public void init() {
            setBackground(Color.black);
      }

      public void paint(Graphics g) {
            g.setColor(Color.red);
            g.drawRect(10, 20, 100, 50);
            g.fillRect(150, 20, 100, 50);

            g.setColor(Color.gray);
            g.drawRoundRect(270, 20, 100, 50, 40, 40);
            g.fillRoundRect(390, 20, 100, 50, 40, 40);

            g.setColor(Color.white);
            g.drawLine(10, 85, 400, 85);

            g.setColor(Color.pink);
            g.drawOval(10, 100, 100, 50);
            g.fillOval(150, 100, 100, 50);

            g.setColor(aqua);
            g.drawArc(270, 100, 100, 50, 45, 90);
            g.fillArc(390, 100, 100, 50, 45, 90);

            g.setColor(darkyellow);
            int x[] = {0, 50, 100};
            int y[] = {286, 200, 286};
```

```
            g.drawPolygon(x, y, 3);
            int x2[] = {150, 200, 250};
            g.fillPolygon(x2, y, 3);

            g.setColor(cellophane);
            g.fillRect(200, 50, 140, 75);
        }
}
```

And our output:



Much better, yes?

Now, there's just one loose end, that `init()` method. Where did it come from? What does it do? It turns out that our `ColorDemo` class inherited this method from `Applet`. `init()`, like `paint()`, is automatically called by the browser. Unlike `paint()`, `init()` is generally only called once, when the applet is first loaded. Usually, you put code that you'd put in the constructors of other objects into the `init()` function.

Fonts

We can manipulate colors, but often times, for text, that's not enough. Consider our Hello World example. If our text had to always look like that, it'd be incredibly dull. Fortunately, there exists the class `Font`, which allows us to set the fonts for our text, with this constructor:

```
Font(String name, int style, int size)
```

Some things to note: the `name` argument takes the name of the font. In this case, we should use "Serif", "SansSerif", "Monospaced", "Dialog", or "DialogInput". Granted, we *could* name a specific font, like **comic sans**, but that's generally risky because if the font doesn't exist on the computer that's running the applet, then you cannot predict the behavior of that particular applet.

As for the style field, the options are `Font.PLAIN`, `Font.ITALIC`, and `Font.BOLD`. Note that these are all bitmasks, so you can use bitwise operations to combine them. For example, if you want a font that's both ***italicized and bolded***, you can use `Font.ITALIC|Font.BOLD`. (Note that it doesn't make any logical sense to try to combine `Font.PLAIN` another font style.)

There are two methods that can be used to set a font (incidentally, they both have the same name): `setFont(Font font)`. Applet has inherited one of these functions, and Graphics has the other. If you call Applet's version (usually in the `init()` method) it'll set that to the default font of the entire applet. On the other hand, if you call Graphics' version (usually in the form `g.setFont(nameOfFont)`), it'll use that font for the rest of the `paint()` method, but the next call to `paint()` will reset the font to the default font. Although both methods do nearly the same thing, the biggest difference between them is stylistic.

Here's an example that illustrates the use of fonts:

```
/* FontDemo.java */

import java.awt.*;
import java.applet.Applet;

public class FontDemo extends Applet {
	Font serif = new Font("Serif", Font.ITALIC, 24);
	Font sansSerif = new Font("SansSerif", Font.BOLD, 14);
	Font mono = new Font("Monospaced", Font.PLAIN, 18);
	Font dialog = new Font("Dialog", Font.ITALIC|Font.BOLD, 20);
	Font dialogIn = new Font("DialogInput", Font.PLAIN, 18);

	public void init() {
		setFont(serif);
	}

	public void paint(Graphics g) {
		g.drawString(serif.getName(), 50, 50);

		g.setFont(sansSerif);
```

```
            g.drawString(sansSerif.getName(), 50, 100);

            g.setFont(mono);
            g.drawString(mono.getName(), 50, 150);

            g.setFont(dialog);
            g.drawString(dialog.getName(), 50, 200);

            g.setFont(dialogIn);
            g.drawString(dialogIn.getName(), 50, 250);
        }
}
```

And if you run this as a 300x300 applet, you'll get this:

Applet Parameters

You know the prototype for the `main()` function? How it looks like `public static void main(String args[])`? Yeah, we remember what that `String args[]` is for. It's to pass command-line arguments to your program. But there's no `main()` function in an applet! So what do we do? First, let's introduce a new HTML tag:

```
<param name=myName value="It's a string!">
```

Put this tag between the `<applet>` and `</applet>` tags to make it work. This tag basically creates a parameter called "myName" with the value "It's a string!"

This is used in conjunction with the follow method in Applet:

```
String getParameter(String paramName)
```

Note that all parameter names and values are Strings. If this is the case, then, how do you pass a numerical parameter? With an integer parser, of course!

```
int num = Integer.parseInt(getParameter("numericParam"));
```

The following example demonstrates one use for Applet Parameters:

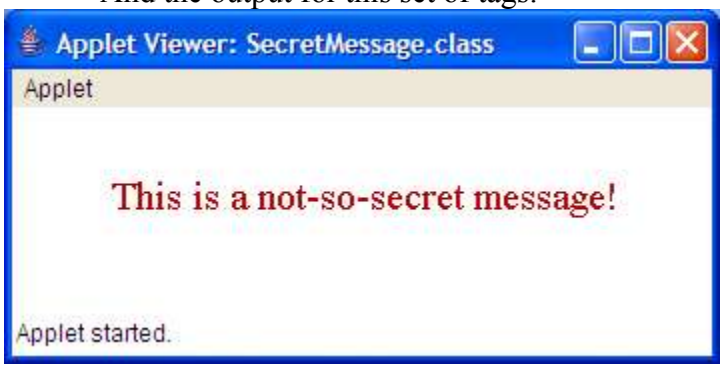
```
/* SecretMessage.java */

import java.awt.*;
import java.applet.Applet;

public class SecretMessage extends Applet {
      Color backColor, textColor;
      Font textFont;
      String message, fontName;
      int fontSize, bgRed, bgGreen, bgBlue, fgRed, fgGreen, fgBlue;

      public void init() {
            fontName = getParameter("fontName");
            fontSize = Integer.parseInt(getParameter("fontSize"));
            textFont = new Font(fontName, Font.PLAIN, fontSize);
            setFont(textFont);

            bgRed = Integer.parseInt(getParameter("bgRed"));
            bgGreen = Integer.parseInt(getParameter("bgGreen"));
            bgBlue = Integer.parseInt(getParameter("bgBlue"));
            backColor = new Color(bgRed, bgGreen, bgBlue);
            setBackground(backColor);

            fgRed = Integer.parseInt(getParameter("fgRed"));
            fgGreen = Integer.parseInt(getParameter("fgGreen"));
            fgBlue = Integer.parseInt(getParameter("fgBlue"));
            textColor = new Color(fgRed, fgGreen, fgBlue);
            setForeground(textColor);

            message = getParameter("message");
      }

      public void paint(Graphics g) {
            g.drawString(message, 50, 50);
      }
}
```

And it's accompanying HTML tags:

```
<!--SecretMessage.html-->
<applet code="SecretMessage.class" width=350 height=100>
      <param name=fontName value="Serif">
      <param name=fontSize value=20>
```

```
      <param name=bgRed value=255>
      <param name=bgGreen value=255>
      <param name=bgBlue value=255>
      <param name=fgRed value=192>
      <param name=fgGreen value=0>
      <param name=fgBlue value=0>
      <param name=message value="This is a not-so-secret message!">
</applet>
```

And the output for this set of tags:



You should definitely play around with the tags to get a feel of this applet.

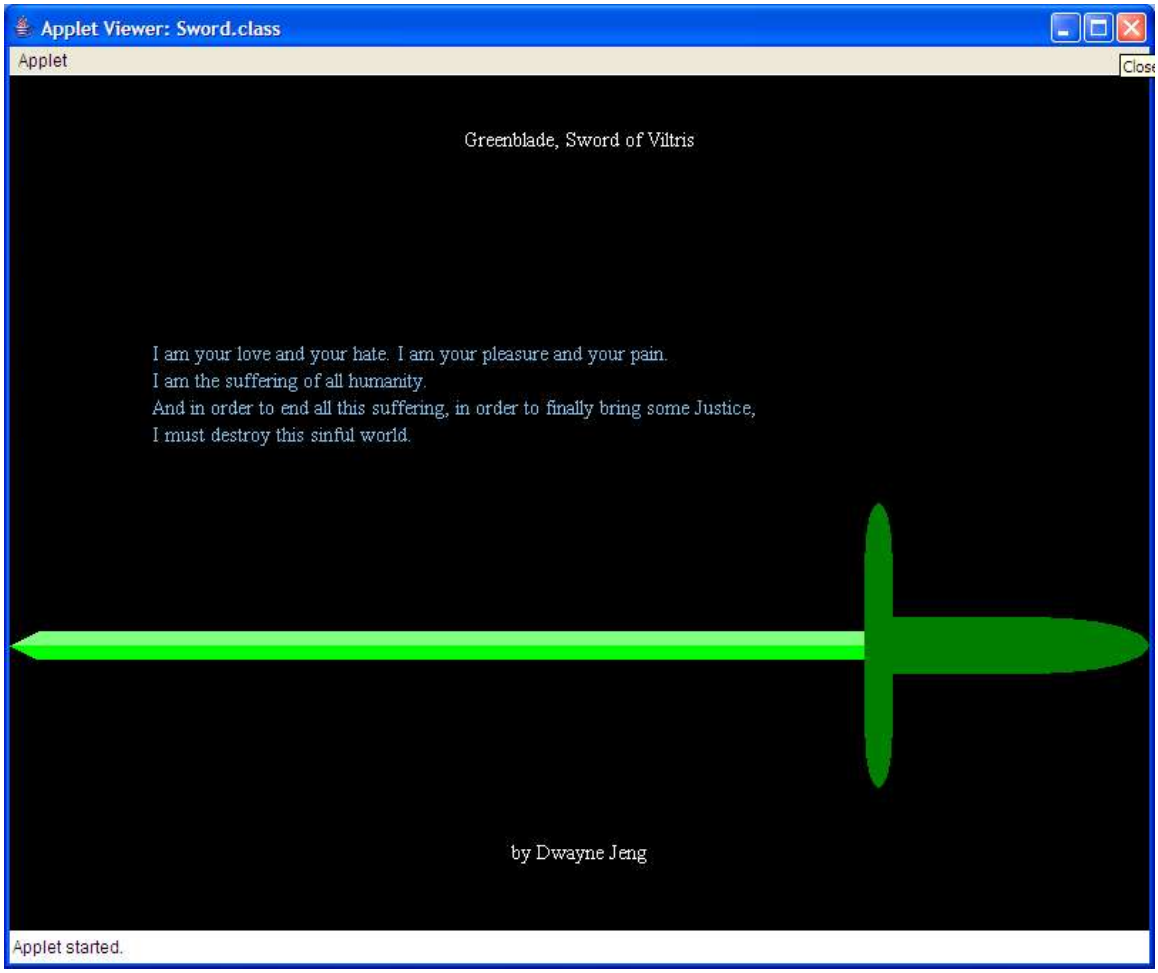
Conclusion

Using these basic building block functions, we can make all sorts of crazy pictures. Here's an example that makes use of fonts, colors, shapes, text, an interesting use for the `init()` method, and some new material that I'm sure you can figure out on your own:

```
/* Sword.java */

import java.awt.*;
import java.applet.Applet;

public class Sword extends Applet {
      Font font;
      FontMetrics metrics;
      Color metalGreen = new Color(127, 255, 127);
      Color darkGreen = new Color(0, 127, 0);
      final int NUMQUOTES = 4;
      int x[][] = {{0, 20, 600, 600}, {0, 20, 600, 600}};
      int y[][] = {{400, 410, 410, 400}, {400, 390, 390, 400}};
      int titleLeft, nameLeft, height;

      Color random;
      int randQuote;

      public void init() {
            setBackground(Color.black);
            setForeground(Color.white);
```

```
            font = new Font("Serif", font.PLAIN, 14);
            setFont(font);
            metrics = getFontMetrics(font);
            titleLeft = (800 - metrics.stringWidth("Greenblade, Sword of
Viltris")) / 2;
            nameLeft = (800 - metrics.stringWidth("by Dwayne Jeng")) /
2;
            height = metrics.getHeight();

            float randRed = (float) (0.5 + 0.5 * Math.random());
            float randGreen = (float) (0.5 + 0.5 * Math.random());
            float randBlue = (float) (0.5 + 0.5 * Math.random());
            random = new Color(randRed, randGreen, randBlue);
            randQuote = (int) Math.floor(Math.random() * NUMQUOTES);
      }

      public void paint(Graphics g) {
            g.drawString("Greenblade, Sword of Viltris", titleLeft, 50);
            g.drawString("by Dwayne Jeng", nameLeft, 550);

            g.setColor(Color.green);
            g.fillPolygon(x[0], y[0], 4);
            g.setColor(metalGreen);
            g.fillPolygon(x[1], y[1], 4);
            g.setColor(darkGreen);
            g.fillRect(600, 340, 20, 120);
            g.fillOval(600, 300, 20, 80);
            g.fillOval(600, 420, 20, 80);
            g.fillRect(620, 380, 100, 40);
            g.fillOval(640, 380, 160, 40);

            g.setColor(random);
            if (randQuote == 0) {
                  g.drawString("I am your love and your hate. I am your
pleasure and your pain.", 100, 200);
                  g.drawString("I am the suffering of all humanity.",
100, 200 + height);
                  g.drawString("And in order to end all this suffering,
in order to finally bring some Justice,", 100, 200 + (2

* height));
                  g.drawString("I must destroy this sinful world.", 100,
200 + (3 * height));
            } else if (randQuote == 1) {
                  g.drawString("Weak! Why are they all so weak?!?", 100,
200);
            } else if (randQuote == 2) {
                  g.drawString("War brings Unity. Unity brings Peace.
Peace brings Justice.", 100, 200);
            } else if (randQuote == 3) {
                  g.drawString
("HAHAHAHAHAhahahahahAHAHAHAHAHAhahA!!!!!", 100, 200);
            }
      }
}
```

And here is one instance of our output (shrunken in size, since this applet runs in 800x600 mode):



Notice that every time we run this applet, the quote will look different.

"Wouldn't it be easier," you might ask, "to simply make an image rather than doing all that drawing for the sword?"

Well, the answer is yes, but the point of this example was to have fun with shapes and text and stuff. We'll be covering images handling in a later tutorial.

Homework Assignment!

Make your own neat little graphic using shapes and the other material we have learned in this tutorial. Creativity counts!