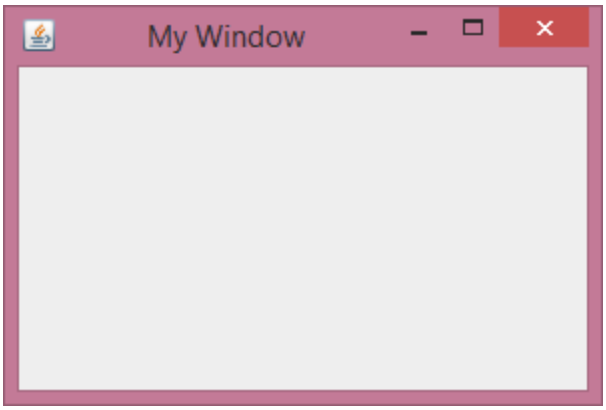
Basics of Drawing Lines, Shapes, Reading Images

To draw some simple graphics, we first need to create a window. The easiest way to do this in the current version of Java is to create a JFrame object which is part of the Swing library. When we create a JFrame object we actually create a new window on the screen. Here is an example:

```
import javax.swing.JFrame;
import java.awt.Graphics;

public class GTest extends JFrame
{
	public GTest()
	{
		setSize(300, 200);	// 300 pixels across, 200 down
		setDefaultCloseOperation(EXIT_ON_CLOSE);
		setTitle("My Window");
		setVisible(true); // Makes window visible
	}

	public static void main(String[] args)
	{
		GTest myWindow = new GTest(); // Title
	}
}
```

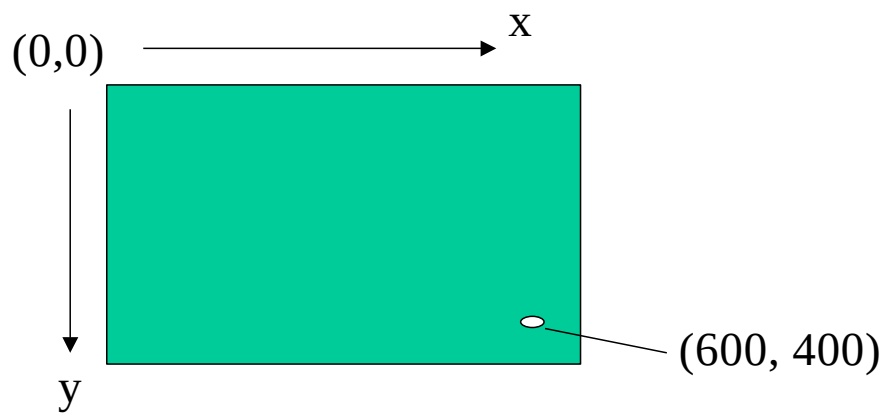
This program creates the following window and makes it visible:



The import statements give us access to Java graphics libraries. By making our class extend JFrame, we are inheriting all the code in the constructor to set the size, make it exit, and make it visible. The JFrame code already implements things like resizing, minimizing, etc.

Drawing in the JFrame

Once we have a JFrame, we will get access to a graphics object on the frame.
The graphics screen is set up as a grid of pixels where the upper left coordinate is 0,0. This is relative to where the canvas is on the viewframe. The x coordinate then grows out to the right, and the y coordinate grows down toward the bottom.



Here is some sample code that illustrates how to draw a rectangle and a string of text on the screen. Resist the urge to put your drawing commands outside the JFrame, like main. It is possible, but then your window won't refresh properly, because other factors might cause the window to have to refresh itself, and it can't if the code for how to draw the window is in main. **The idea is each object should know how to draw itself, and the place where the drawing happens is in the paint() method.**

```
import javax.swing.JFrame;
import java.awt.Graphics;
import java.awt.Color;

public class GTest extends JFrame
{

	// The paint method below determines what to
	// draw on the screen and is invoked by Java
	public void paint(Graphics g)
	{
		super.paint(g); // Must do this first
		g.setColor(Color.red);  // Sets color to red
		// NOTE: Title Bar about 22 pixels down
		// Border frame about 7 pixels across

		// Draw rectangle at X=10, Y=50,
		// width=100, height = 70  pixels
		g.drawRect(10,50,100,70);
		// Write text at X = 100, Y = 150
		g.drawString("Moof.", 100, 150);
	}

	public GTest()
```

```
        {
                setSize(300, 200);      // 300 pixels across, 200 down
                setDefaultCloseOperation(EXIT_ON_CLOSE);
                setVisible(true); // Makes window visible
        }

        public static void main(String[] args)
        {
                GTest myWindow = new GTest(); // Title
        }
}
```

This code produces the following image:

In this code, we are importing java.awt.Color to get access to some predefined colors.

When Java goes to display what is in the window, it invokes the `paint()` method. The parameter is a Graphics object that supports many drawing functions. Here we are using `drawRect` and `drawString`.

`drawRect(x1,y1,width,height)` draws a rectangle with the upper left corner at (x1,y1) and the specified width and height.

`drawString(string, x1, y1)` draws the text string at coordinates x1, y1.

It is important to note that our drawing coordinates include pixels used for the title bar and the frame around the border. With my widget scheme the title bar is about 23 pixels, and the border about 7 pixels wide. However, if we try to draw something at that location, we get nothing. For example, if the drawRect is:

```
g.drawRect(0,0,100,70);
```

Then the output will look like:

Colors

Here are different colors available using the color class:

Color.black	Color.darkGray	Color.gray
Color.blue	Color.green	Color.lightGray
Color.cyan	Color.magenta	Color.orange
Color.pink	Color.red	Color.white
Color.yellow		

To create our own color, we can specify the color we want in RGB (red, green, blue) color model. This is an additive color model that is somewhat like mixing colors of light or like mixing paints. The idea is that any color is represented as some combination of red, green, and blue. To specify a color use:

```
new Color(redvalue, greenvalue, bluevalue);
```

where each value is in the range 0-255.

For example:

```
new Color(0,255,200);
```

Creates a green-blue color, with stronger green than blue. If red, green, and blue are all 0 then this is the color black. If red, green, and blue are all 255 then this is the color white. Red and green makes yellow, red and blue makes magenta, blue and green makes cyan, etc.

More drawing functions

Here is a list of other drawing functions you can use with the graphics object:

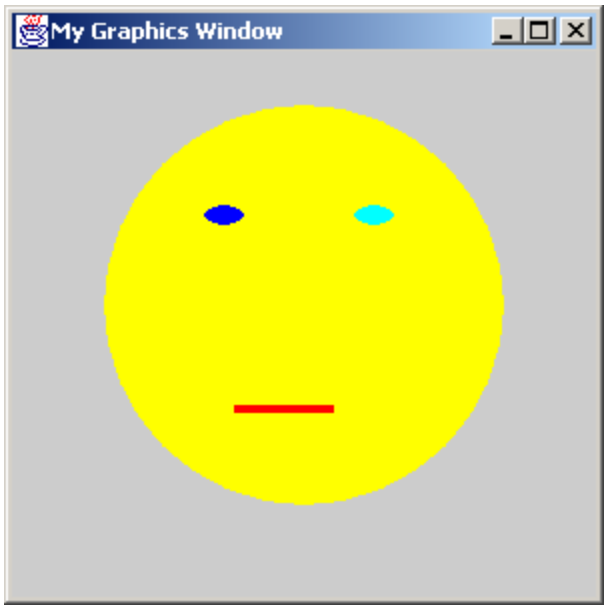
setColor(color)
Sets the current color to a color defined as above
drawLine(int x1, int y1, int x2, int y2)
Draws a line in the current color from x1,y1 to x2,y2
Can use drawLine(x1,y1,x1,y1) to draw a single pixel at x1,y1
drawOval(int x, int y, int width, int height)
Draws an oval in the current color within a box at upper left corner x,y
With specified width and height in pixels
Use width = height to draw a circle
drawRect(int x, int y, int width, int height)
Draws a rectangle in the current color with upper left corner at x,y
With specified width and height in pixels
drawArc(int x, int y, int width, int height, int startAngle, int arcAngle)
Draws an arc in the current color at the coordinates and specified angle
fillOval(int x, int y, int width, int height)
Draws an oval in the current color within a box at upper left corner x,y
With specified width and height in pixels
Use width = height to draw a circle
fillRect(int x, int y, int width, int height)
Fills a rectangle in the current color with upper left corner at x,y
With specified width and height in pixels
fillArc(int x, int y, int width, int height, int startAngle, int arcAngle)
Fills an arc in the current color at the coordinates and specified angle

There are many more, for a full list see
http://docs.oracle.com/javase/7/docs/api/java/awt/Graphics.html

Here is an example using some of these functions to draw an ambivalent yellow face:

```
public void paint(Graphics g)
{
	super.paint(g);
	g.setColor(Color.yellow);
	g.fillOval(50,50,200,200);
	g.setColor(new Color(0,0,255));		// Blue left eye
	g.fillOval(100,100,20,10);
	g.setColor(new Color(0,255,255));	// Blue-Green right eye
	g.fillOval(175,100,20,10);
	g.setColor(new Color(255,0,0));		// Red mouth
	g.fillRect(115,200,50,4);
}
```

This program draws the following image:



How to draw a curved mouth?

Working with Pixels on Images

Here is a program that demonstrates how to read an image file from the disk and display it in a JFrame window. In this case I have a picture named **`kids.jpg`** that is stored in the same directory as the Java program.

```
import javax.swing.JFrame;
import java.awt.Graphics;
import java.awt.Color;
import java.awt.image.BufferedImage;
import java.io.File;
import javax.imageio.ImageIO;
import java.util.Scanner;
import java.io.IOException;

public class ShowImage extends JFrame
{
  private static BufferedImage image = null;
  // Will hold the image we load from disk

  public void paint(Graphics g)
  {
    super.paint(g);
    // Draw the image we loaded on the screen
    g.drawImage(image, 5, 35, null);
  }

  public ShowImage()
  {
```

```
    setSize(810, 645);
    setDefaultCloseOperation(EXIT_ON_CLOSE);
    setVisible(true);
  }

  public static void main(String[] args)
  {
    ShowImage window = new ShowImage();
    try
    {
        File input = new File("kids.jpg");
        image = ImageIO.read(input);
    }
    catch (IOException e)
    {
        System.out.println(e);
    }
    window.repaint();  // Forces paint to be called again
  }
}
```

Here is what this program displays:

Let's show how we can access individual colors of the image. Add the following code to the main method right before the `myWindow.repaint()`:

```
    int x;
    int pixel;
    // Get pixel color values for the first line
    for (x = 0; x < image.getWidth(); x++)
    {
```

```
      pixel = image.getRGB(x, 0);
      Color c = new Color(pixel);
      int r,g,b;
      r = c.getRed();
      g = c.getGreen();
      b = c.getBlue();
      System.out.print("At x=" + x + ", y=0 the color is ");
      System.out.println("red: " + r + " green: " + g + " blue: " + b);
   }
```

This code will output the Red, Green, and Blue values of each pixel on the first horizontal line of the image. Here is a sample of the output:

```
At x=793,y=0 the color is red: 150 green: 159 blue: 164
At x=794,y=0 the color is red: 147 green: 156 blue: 161
…
```

We can also set the color of pixels if we like. Consider the following loop:

```
        for (int x = 0; x < image.getWidth(); x++)
        {
                int redColor = new Color(255,0,0).getRGB();
                image.setRGB(x,0,redColor); // Set to red
        }
```

This code loops through each pixel on the top row and sets its color to red (255 red, 0 green, 0 blue). This is shown below (the top line is turned to red).

Image Brightness

If we wanted to set every pixel to red, we would just need a nested loop so that we process every row in addition to the columns. An example is shown below. However, it doesn't turn every pixel to red – can you guess what it will do?

```
    int x,y;
    int pixel;

      for (y = 0; y < image.getHeight(); y++)
      {
        for (x = 0; x < image.getWidth(); x++)
        {
            pixel = image.getRGB(x, y);
            Color c = new Color(pixel);
            int r,g,b;
            r = c.getRed();
            g = c.getGreen();
            b = c.getBlue();
            r = (int) (r / 2.5);
```

```
            g = (int) (g / 2.5);
            b = (int) (b / 2.5);
            int newColor = new Color(r,g,b).getRGB();
            image.setRGB(x,y,newColor);
        }
    }
```

In this case we decrease the red, green, and blue components by 2.5. This darkens the entire image. If we used a large enough value each pixel would become black.

If we wanted to brighten the image, we might try changing the code so we multiply by 2.5 instead of dividing by 2.5:

```
r = (int) (getRed(pixel) * 2.5);
g = (int) (getGreen(pixel) * 2.5);
b = (int) (getBlue(pixel) * 2.5);
```

However, this results in a program crash:

```
Exception in thread "main" java.lang.IllegalArgumentException: Color parameter o
utside of expected range: Green
        at java.awt.Color.testColorValueRange(Color.java:310)
        at java.awt.Color.<init>(Color.java:395)
        at java.awt.Color.<init>(Color.java:369)
        at ShowImage.main(ShowImage.java:58)
Press any key to continue . . .
```

This is because a color's red, green, or blue component cannot be larger than 255. We can compensate for this by limiting the maximum value to 255:

```
r = (int) (r * 2.5);
if (r > 255)
      r = 255;
g = (int) (g * 2.5);
if (g > 255)
      g =255;
b = (int) (b * 2.5);
if (b > 255)
      b = 255;
```

When we "clip" the color red, green, or blue at 255 this does result in a washed-out picture. However, this is likely preferable to the funky colors.

Changing Color Values - Grayscale

We can also use our basic nested loop to easily convert an image to grayscale. A color of gray is one in which the red = green = blue. Large values are white and small values are black. An easy way to make an grayscale image out of color is to set each color value to the average of all three:

Gray = (Red + Green + Blue) / 3
Red = Gray
Green = Gray
Blue = Gray

Here is an example:

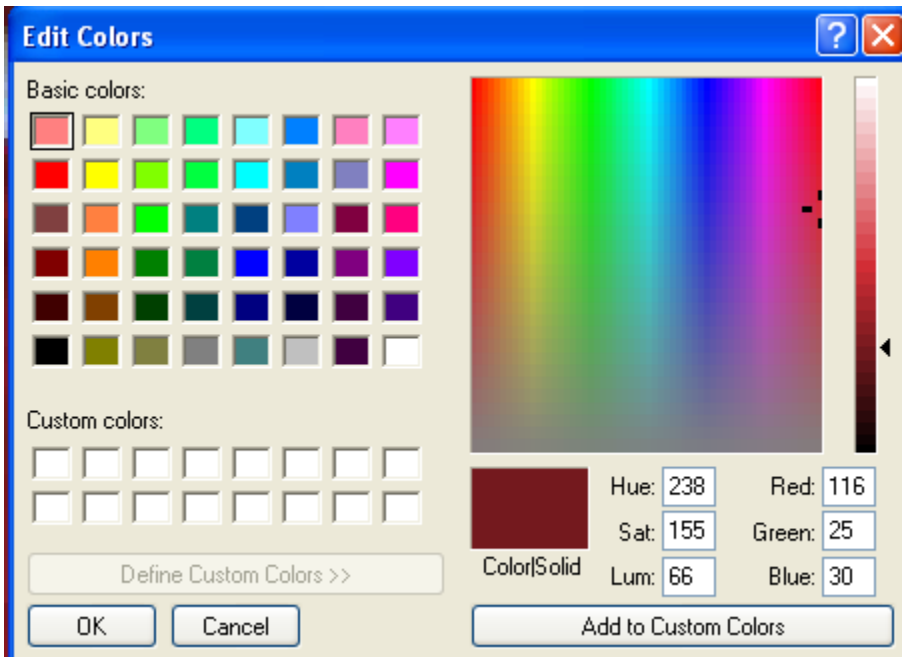
```
int x,y;
int pixel;

  for (y = 0; y < image.getHeight(); y++)
  {
    for (x = 0; x < image.getWidth(); x++)
    {
        pixel = image.getRGB(x, y);
        Color c = new Color(pixel);
        int r,g,b;
        r = c.getRed();
        g = c.getGreen();
        b = c.getBlue();
        int gray = (r + g + b) / 3;
        int newColor = new Color(gray,gray,gray).getRGB();
        image.setRGB(x,y,newColor);
    }
  }
```

The image with the kids becomes this:

Selective Color Changes

Leading up to our rock thin slice GeoPixelCount program, we can selectively change colors in the image. In the original picture, there is some red in the shirt for the kid in the middle. Let's change the red to green.

Using a paint program (Paint in Windows will do) we can examine the range of pixel values and coordinate locations for the red shirt. In Paint we can use the eye dropper tool to find coordinates, select a color, and find its RGB from the Edit Colors menu.



Using this we can see that the red is typically twice as much as the green and blue. If we loop over the image and find all pixels with this property (in our case, 1.8 times as much to give a little tolerance), we can make them much greener by swapping the red and green amounts:

```
int x,y;
int pixel;

  for (y = 0; y < image.getHeight(); y++)
  {
    for (x = 0; x < image.getWidth(); x++)
    {
        pixel = image.getRGB(x, y);
        Color c = new Color(pixel);
        int r,g,b;
        r = c.getRed();
        g = c.getGreen();
        b = c.getBlue();

        if ((r > 1.8*g) && (r > 1.8*b))
        {
              int temp = r;
              r = g;
              g = temp;
              int newColor = new Color(r,g,b).getRGB();
              image.setRGB(x,y,newColor);          }
    }
  }
```

The result is close, but not quite there! We change the color on most of the shirt, but lots of other stuff got changed to green too.

One way out of this problem would be to make separate loops that apply only to a small area instead of the entire image. For example we could make a loop that only changes pixels in the general area of the shirt, in this case from X=319,Y=317 to 392,397 to change the bottom part of the shirt.

```
for (y = 317; y < 397; y++)
{
  for (x = 319; x < 392; x++)
  {
      pixel = image.getRGB(x, y);
      Color c = new Color(pixel);
      int r,g,b;
      r = c.getRed();
      g = c.getGreen();
      b = c.getBlue();

      if ((r > 1.8*g) && (r > 1.8*b))
      {
            int temp = r;
            r = g;
            g = temp;
            int newColor = new Color(r,g,b).getRGB();
            image.setRGB(x,y,newColor);          }
  }
}
```

It still needs some work but if we used separate loops or modify the formula for "red" then we could get more of the red pixels while avoiding the skin pixels.

A very similar process is done when performing red-eye reduction on an image in a photo editing program. The user typically selects the eye region (so the program knows what area to look for) and changes any reddish pixels in that area to dark pixels.