

Working with Pixels on Images

Images provide a nice visual way to see loops in action. We'll use nested loops to process images to do things you might normally run in a paint program.

First, 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 **MyKids.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;

public class ShowImage extends JFrame
{

    private static BufferedImage image = null; // Stores image

    public void paint(Graphics g)
    {
        super.paint(g);
        g.drawImage( image, 5, 35, null);    // Draws the image
        // Upper left corner at 5,35 (avoid drawing over border)
    }

    public ShowImage()
    {
        setSize(810, 645);   // Size of the window in pixels
        setDefaultCloseOperation(EXIT_ON_CLOSE);
        setVisible(true);
    }

    //-----
    /**
     * Main method. You have to add the "throws Exception" at
     * the end here. Later we will see a better way to do this
     * using "try" and "catch".
     */
    public static void main(String[] args) throws Exception
    {
        ShowImage myWindow = new ShowImage();

        // These next two lines read the image from the file
        File input = new File("MyKids.jpg");
        image = ImageIO.read(input);

        myWindow.repaint();
    }
}
```

Here is what this program displays:



There is a lot of material here that is mysterious for now, but don't worry about that – we'll just focus on how to manipulate the picture image.

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: 130 green: 144 blue: 153
At x=794,y=0 the color is red: 134 green: 148 blue: 157
...

```

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

```
for (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:

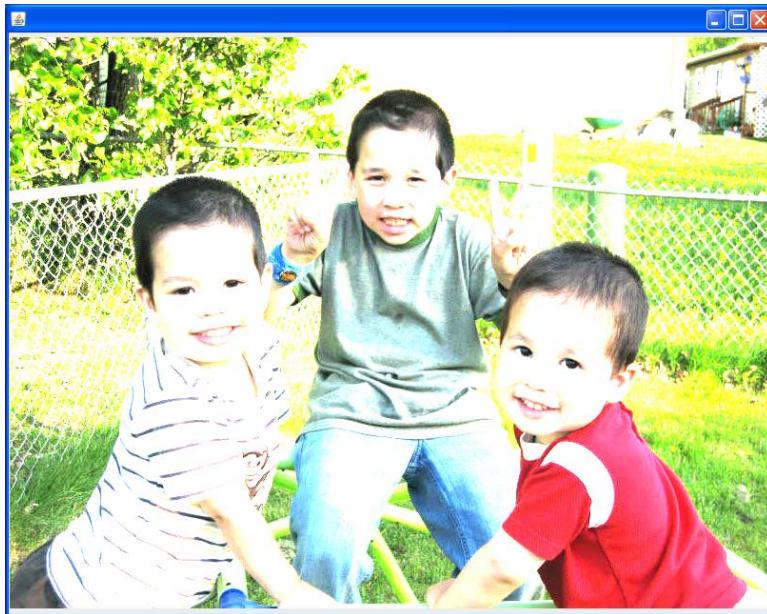
```
r = (int) (getRed(pixel) * 2.5);
g = (int) (getGreen(pixel) * 2.5);
b = (int) (getBlue(pixel) * 2.5);
```

However, this results in a program crash:

```
Exception in thread "main" java.lang.IllegalArgumentException: Color parameter outside 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:

$$\begin{aligned} \text{Gray} &= (\text{Red} + \text{Green} + \text{Blue}) / 3 \\ \text{Red} &= \text{Gray} \\ \text{Green} &= \text{Gray} \\ \text{Blue} &= \text{Gray} \end{aligned}$$

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:

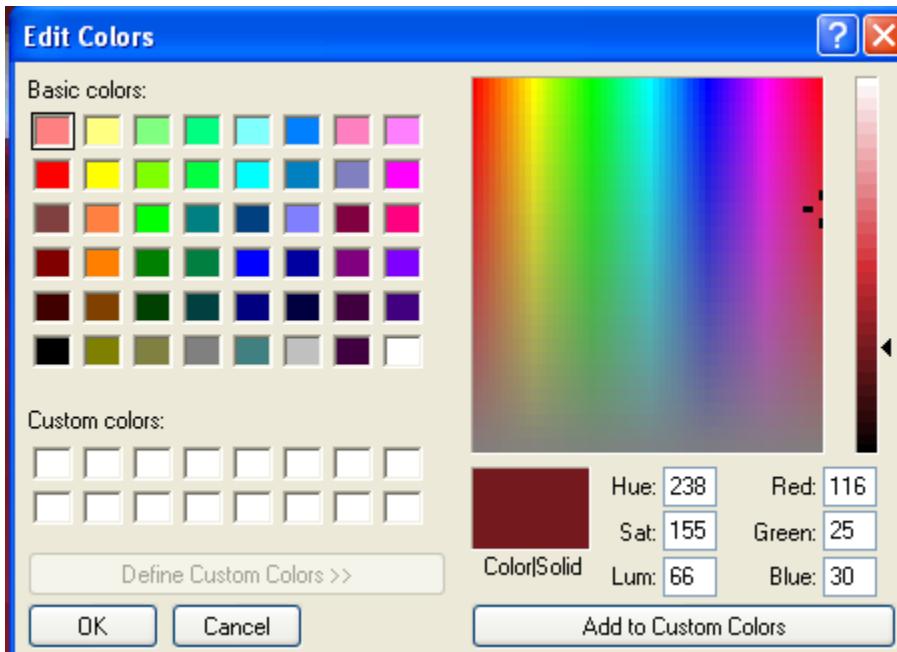


Selective Color Changes

Let's say that red is no longer our favorite color and we would like to turn the color of the red on the shirt 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 almost always twice as big as the green and almost always twice as big as the blue.

If we loop over the image and find all pixels in this range, we can make them much greener by decreasing the amount of red and increasing the amount of green:

```

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 > 2*g) && (r > 2*b))
        {
            r = r / 3;
            g = g * 3;
            if (g > 255)
                g = 255;
            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 some of the lip and ear is inadvertently changed as well. We could increase our threshold and get less flesh tones, but then fewer pixels on the shirts would be changed.



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 red shirt, in this case from X=512 to the width of the image, and Y=385 to the height of the image.



```
for (y = 385; y < image.getHeight(); y++)
{
    for (x = 512; 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 > 2*g) && (r > 2*b))
        {
            r = r / 3;
            g = g * 3;
            if (g > 255)
                g = 255;
            int newColor = new Color(r,g,b).getRGB();
            image.setRGB(x,y,newColor);
        }
    }
}
```

We still get a little bit of the cheek but miss the ear and lips. If we used separate loops or additional if statements to skip the cheek pixels then we could perfect the color change.



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.