

Other interesting Picture topics

- Sepia Tone
- Blurring
- Edge Detection

Programming Design

- What makes good functions
- Debugging techniques

Class Methods versus Object Methods

- How do I automate a Java program
 - Ultimate goal -- make it "double clickable"
- How do I get input from the user so that a program can take paths of execution suited to a particular user?

For a change of pace, we'll start by talking about what makes good functions.

- How can we reuse variable names like **newpic** in both a function and in the Command Area?
- Why do we write the functions like this? Would other ways be just as good?
- Is there such a thing as a better or worse function?
- Why don't we just build in calls to **pickAFile** and **new Picture(...)**?

- We write functions as we do to make them *general and reusable*

- Programmers hate to have to re-write something they've written before
- They write functions in a general way so that they can be used in many circumstances.

- What makes a function *general and thus reusable*?

- A reusable function does *One and Only One Thing*

Compare the following two programs

```
public static void turnKatieRedHead()
{
    Color brownhair = new Color(42, 25, 15);
    Picture.setMediaPath("/Volumes/Personal Workspace/bressoud/cs171/MediaSources/");

    Picture.katiePicture = new Picture(Picture.getMediaPath("KatieFancy.jpg"));
    Pixel[] pixels = katiePicture.getPixels();
    Pixel mypixel;
```

```

for (int i=0; i < pixels.length; i++)
{
    // get the current pixel

    pixel = pixels[i];

    // check if in distance to brown and if so, double the red

    if (pixel.colorDistance(brownhair) < 50.0) {
        pixel.setRed(pixel.getRed() * 2.0);
    }
}

katiePicture.show();
}

```

```

public Picture turnRedHead(Color haircolor)
{
    Picture result = new Picture(getWidth(), getHeight());

    Pixel[] pixels = this.getPixels();
    Pixel mypixel;

    for (int i=0; i < pixels.length; i++)
    {
        // get the current pixel

        pixel = pixels[i];

        // check if in distance to brown and if so, double the red

        if (pixel.colorDistance(haircolor) < 50.0) {
            pixel.setRed(pixel.getRed() * 2.0);
        }
    }

    return result;
}

```

Of course, to get the same net end result as the first method, we need to do:

```

Picture.setMediaPath("/Volumes/PersonalWorkspace/bressoud/cs171/MediaSources/");
Picture katie = new Picture(Picture.getMediaPath("KatieFancy.jpg"));
Picture katiered = katie.turnRedHair(new Color(42, 25, 15));

```

but I can, without any recompilation, also do

```

Picture barb = new Picture(Picture.getMediaPath("barbara.jpg"));
Picture barbred = barb.turnRedHair(new Color(50, 30, 15));

```

Now compare these two methods/programs:

```

public Picture makeSunset()
{
    Picture result = new Picture(getWidth(), getHeight());
}

public Picture makeSunset()
{
    Picture p1 = this.reduceBlue();
}

```

```

public Picture makeSunset()
{
    Picture result = new Picture(getWidth(), getHeight());

    Pixel[] pixels = this.getPixels();
    Pixel mypixel;
    int value;

    for (int i=0; i < pixels.length; i++)
    {
        // get the current pixel

        pixel = pixels[i];

        value = pixel.getBlue();
        pixel.setBlue((int) (value * 0.7));

        value = pixel.getGreen();
        pixel.setGreen((int) (value * 0.7));
    }

    return result;
}

```

```

public Picture makeSunset()
{
    Picture p1 = this.reduceBlue();
    Picture result = p1.reduceGreen();

    return result;
}

```

```

public Picture reduceBlue()
{
    Picture result = new Picture(getWidth(), getHeight());

    Pixel[] pixels = this.getPixels();
    Pixel mypixel;
    int value;

    for (int i=0; i < pixels.length; i++)
    {
        // get the current pixel

        pixel = pixels[i];

        value = pixel.getBlue();
        pixel.setBlue((int) (value * 0.7));
    }

    return result;
}

```

```

public Picture reduceGreen()
{
    Picture result = new Picture(getWidth(), getHeight());

    Pixel[] pixels = this.getPixels();
    Pixel mypixel;
    int value;

    for (int i=0; i < pixels.length; i++)
    {
        // get the current pixel

        pixel = pixels[i];

        value = pixel.getGreen();
    }
}

```

```
        pixel.setGreen((int) (value * 0.7));
    }
    return result;
}
```

Does `makeSunset` do *one and only one thing*?

- Yes, but it's a higher-level, *more abstract thing*.
 - It's built on lower-level *one and only one thing*
- We call this *hierarchical decomposition*.
 - You have some *thing* that you want the computer to do?
 - Redefine that *thing* in terms of smaller *things*
 - Repeat until you know how to write the smaller things
 - Then write the larger things in terms of the smaller things.

What happens when we invoke a method?

If we do:

```
Picture p1 = new Picture(FileChooser.pickAFile());
```

```
Picture p2 = p1.turnRedHead(new Color(45, 25, 15));
```

Back to basic idea of substitution

1. First, evaluate arguments to methods.
2. Next evaluate method and substitute evaluated arguments (from step 1) for formal parameters.

So, the object created by `'new Color(45, 25, 15)'` ... or more precisely, the object reference