

Getting Started with Images

Color

You can follow the link below to find the Java API specification; this is a listing of all the classes that come prepackaged with Java. In the listing at the left, find and click on the `Color` class.

<http://java.sun.com/j2se/1.4.2/docs/api/>

Notice that `Color` comes with several predefined colors as constants. There are also methods to get the red/blue/green components of a color; each of these will return a number in the range of 0-255. Try the following commands on the Dr. Java interactions pane:

```
import java.awt.Color;
Color c = Color.orange;
System.out.println(c);
c.getRed();
c.getGreen();
c.getBlue();
```

That first line (the import statement) is necessary because Dr. Java doesn't always "load" all the prebuilt classes because there are too many of them and it would be a waste of computer memory. So we need to instruct Dr. Java to load or import the `Color` class so that we can use it.

Now search for the `Pixel` class in the listing at the left. Do you find it?

The reason that `Pixel` isn't there is that this class is not one of the prebuilt classes that comes with Java. Instead, the authors of our textbook have constructed the `Pixel` class. It should be in the directory that contains all of our book classes. Use Dr. Java to open the `Pixel` class so that it is displayed in our editor window. Be careful not to make changes to `Pixel` just yet.

JavaDoc

It is a little hard to read the `Pixel` class. Wouldn't it be nicer to have a listing and description of `Pixel` and the other book classes that is similar to the one that

comes with the Java API? Well, fortunately there is a way.

Open all of the java files in your book classes folder; there are about a dozen of them. Once they are all open, click the "Javadoc" button at the top. It will prompt you to create a directory (accept the defaults) and then it will take a while. You might see some warning messages at the bottom when it finishes, these are ok.

If your browser does not automatically open the new Javadoc description, you will need to do it by hand. With your browser (Safari) select "open" and then go to the book classes folder. In here you will find the new "doc" folder that was just created. Inside the doc folder, you will find "index.html". Click and open this file. Viola, up should pop a description/listing of all the book classes. Click on the one about `Pixel` and examine it.

Pixel

As you read about the `Pixel` class you will notice that the Javadoc description does not include any of the `private` items from the class. This makes sense since the API description is meant to be the public interface to the class (the instruction manual) and you don't necessarily need to inform the user about all the internal details of your class that they can't change or use anyway. The `Pixel` class has methods that allow us to get and set the color components, we can also get the x,y coordinates.

Picture

Open the `Picture` class in your API listing. Read about all the methods that come with `Picture`.

You may notice there are only two; actually one constructor and one `toString` method. Where are all the other methods? In fact, they are "inherited" from a different class. The `Picture` class is built from (or *extends*) the `SimplePicture` class. Open the `SimplePicture` class and read about this. Anything that is in a `SimplePicture` object is also in a `Picture` object.

The distinction is a little like the difference in buying cars. You can buy a base level Chevy Cavalier or you can buy a Cavalier with all the extra options. They are both Cavaliers and both do the same thing, but the loaded version comes with extra goodies that are not part of the base model. The `SimplePicture` is the base model while the `Picture` class is the loaded version with extra good-

ies. We will be adding to the `Picture` class to incorporate our own extra features.

In `SimplePicture` we see that the last constructor accepts a `String` filename as input. Where have we seen filenames as strings before? Yup, the `FileChooser` method – notice it is here as part of the book class listing. Try the following where you are sure to select your picture `jpg` file when prompted by the file chooser:

```
Picture p = new Picture(FileChooser.PickAFile());
p.show();
```

You should have seen your picture pop up. In the statements above, we have combined several steps into one. We could have done this same thing using separate steps:

```
String myFileName;
myFileName = FileChooser.pickAFile();
Picture p;
p = new Picture(myFileName);
p.show();
```

Manipulating Images

Remember our picture is really a matrix of pixels that contains many rows and many columns. Let us find out how big our picture is. Can you look at the `SimplePicture` API to see how we might find out how wide and how high our picture is (measured in number of pixels)?

```
p.getWidth()
p.getHeight()
```

Notice if you leave the semicolon off the end (as is done above), Dr. Java will “print” the result of the statement. Normally, legal java statements all must end in semicolons so this feature is only supported in Dr. Java. It is really just a shortcut for:

```
System.out.println(p.getWidth());
System.out.println(p.getHeight());
```

I want to see if I can retrieve one of the pixels from my image. I will aim for a pixel in the middle (half way in terms of width) and top (y coordinate/height is 0) because there are some bright green pixels here.

```
Pixel pix = p.getPixel(p.getWidth/2,0);
pix.getRed()
pix.getGreen()
pix.getBlue()
```

For me, the RGB colors were (52,96,71) – yours will likely be different. Notice the green is the largest component which confirms my guess that this is a green pixel I had retrieved.

Can you now change the pixel so that it is bright red? After you do so, you will need to execute the show method for your picture again so that the change is visible.

```
pix.setRed(255);
pix.setGreen(0);
pix.setBlue(0);
p.show();
```

If you look very very closely you should be able to see a tiny bright red dot.

Arrays of Pixels

There are times when we may want to examine and/or change each pixel in the image. It is too slow to type commands that access each (x,y) pixel one at a time. Instead, we will grab the entire collection of pixels in one big list that is more convenient and faster to work on.

We can grab all the pixels at once using:

```
Pixel array[] = p.getPixels();
```

Now we have a variable called `array` this is a listing of all the pixels in the image. There are likely to be several thousand. Since my image is 640 wide by 320 high, there are 204800 pixels in the array. Each pixel has an *index* in the array. The indices start at number 0 and go up to 204799 (notice the unusual numbering convention).

We can access the pixels with a loop that performs the same operation on each pixel:

```
for (int i = 0; i < array.length; i++ )
    array[i].setRed(255);
p.show();
```

We have now set the red component of each pixel to its brightest setting. After the `show()` method, our image should have a very red tint to it.