Program #3 Helper

Prof Bill - May 2018

This is my helper for P3, Image Pal. thanks… yow, bill

Remember - test images link, wtkrieger.faculty.noctrl.edu/csc210-spring2018/images

Update - I have updated code in my program3 folder on the k: drive.

- ★ P3Helper - color distance and RGB methods
- ★ P3Palettes - predefined color palettes: web safe, xterm, NES, and grayscale

Roadmap

Follow my roadmap if you like.

Part 1 - GUI

Start your program. Get a simple GUI setup for you to add features.

1. Create Program3 project, no packages
2. Do GUI Hello, world (Program3 class?)
3. Create class for playing with images (TestImage class?)
    a. read file
    b. display in gui (fit into your space)
    c. write file
    d. show some basic image stats: num row, num columns, num pixels, etc
    e. Some help from the official Oracle Javadocs:
        i. docs.oracle.com/javase/8/javafx/api/javafx/scene/image/Image.html
        ii. docs.oracle.com/javase/8/javafx/api/javafx/scene/image/ImageView.html
4. I'm Professor-lazy, so I like the MenuBar in JavaFX. I used it in my P2. You can see that code on the k: drive. Or there's lots of examples on the internets.
    a. My menus {and submenus} so far: Program2 {About, Exit}, File {Open URL, Open, Save}, Palette {??? don't know yet}, Color Cruncher {???}

Part 2 - Count colors

Count the number of colors in an image using 2 structures: ArrayList, HashMap.

5. **Palette210**… class brainstorm = results. Huzzah! I've added our Palette210 interface to the k: drive. Copy these files to your folder. Use of this palette is not mandatory, but it is **highly recommended**. Email me if you have questions.
    a. **Palette210** interface - methods you need to count colors in an image
    b. **Palette210Entry** class - simple <key,value> object is <color,count> here; it's used in the Palette210 interface
6. **ArrayPal** - Code up ArrayPal is-a Palette210. Count colors using an array (ArrayList). Add him to your GUI.
7. **HashPal** - Code up HashPal is-a Palette210. Count colors using a HashMap. Add him to your GUI.
8. **Timing** - I time my color counters and print the results. I want to be able to summarize this in a table at the end.

Part 3 - Crunch colors

The goal of color crunching is to reduce the number of colors in an image to 256 or less. There's a lot of trial and error here.

9. My **ColorCruncher** class has the following attributes:
    a. The image
    b. The HashPal created using this image
    c. An array of Palette210Entry objects using the getEntryArray() method. Each entry has a replacementColor attribute which I can be set to change any color.
    d. The crunched image
10. **Matcher** algorithm - My Matcher algorithm uses an existing palette (like web safe colors) and replaces every color in my image to the closest color in the given palette. This is a pretty easy process, and it gives nice results quickly.
11. **Extractor** algorithm - My Extractor algorithm is 2 steps: 1) try to extract a "nice" color palette from the image, and then 2) run the Matcher using that palette.

Action Jackson says - Show me!
thanks… yow, bill

Research

Basic image links. Google on “javafx image png”

docs.oracle.com/javafx/2/api/javafx/scene/image/Image.html
docs.oracle.com/javase/8/javafx/api/javafx/scene/image/ImageView.html

Image coordinate system:

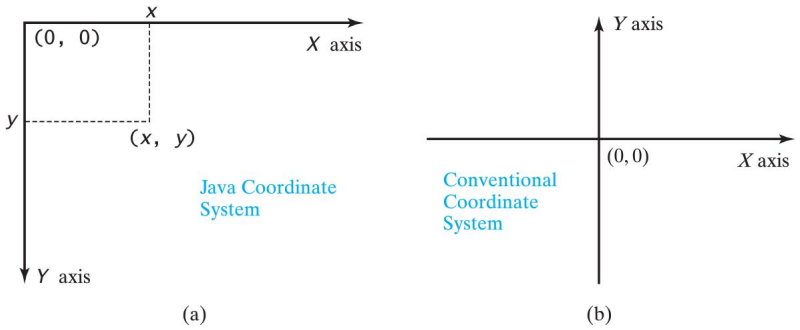

FIGURE 14.6 The Java coordinate system is measured in pixels, with `(0, 0)` at its upper-left corner.

Source: ksuweb.kennesaw.edu/~bsetzer/2302sp15/nanoc/output/notes/0218-javafx/

Scratch the surface:

- Nice background article: Reducing PNG file Size, medium.com/@duhroach/reducing-png-file-size-8473480d0476

- Wikipedia paage on reducing colors, en.wikipedia.org/wiki/Help:How_to_reduce_colors_for_saving_a_JPEG_as_PNG

Some Code

I added two Java files that you can use (optional!). They’re a couple simple GUI I coded up because it bugged me. The code is on my program3 folder on the k: drive.

- **Dialog210** - a popup dialog box to display a message or ask the user for 1 piece of information. It’s modeled after the old, useful Java Swing methods: showMessageDialog and showInputDialog.
- **TextArea210** - a text area with nice methods: println and clear.

There’s more info on the k: drive.

Image File

/* I'm still playing with this, so it may change… */

It looks like two distinct cases for handling image files: 1) local and 2) URL.

Using local file name - Of course, use the JavaFX FileChooser. But you need to doctor the file name that comes back so it works with the Image ctor.

```
imageFile = selectedFile.toURI().toURL().toString();
```

Source: Scroll to the bottom for the FileChooser and string doctoring example,
teamtreehouse.com/community/setting-selected-image-file-to-imageview-in-javafx

My snippet, image from local file:

```
File file = fc.showOpenDialog(null);   // fc is my FileChooser
if (file != null) {
    try {
        String fileName = file.toURI().toURL().toString();
        System.out.println("File/Open: fileName=" + fileName);
        Image img = new Image(fileName);
        // put img in your ImageView or whatever
    } catch (MalformedURLException exc) {
        System.err.println( exc);    // weak error message to console
    }
}
```

Using URL - If you do this, it fails:

```
Image img = new Image( someUrl);
```

It's weird. It doesn't fail because of the url string. It fails because it takes too long and JavaFX freaks out. So, this works:

```
Image img = new Image( someUrl, false);
```

Setting that 2nd ctor parameter is called backgroundLoading… it "indicates whether the image is being loaded in the background".
Source: Image api, docs.oracle.com/javase/8/javafx/api/javafx/scene/image/Image.html
Source: URL help, bekwam.blogspot.com/2017/05/javafx-image-error-handling.html

My snippet, image from URL:

```
try {
    Image img2 = new Image(url, false);
```

```
    // ok to use img2 now
}
catch( Exception exc) {
    System.err.println( exc);    // lame console error msg
}
```

Etc

It looks like you can use equals() on JavaFX Color objects.
stackoverflow.com/questions/15370174/how-does-javafx-compares-colors

When coding HashPal, use HashMap<Color, PaletteEntry210>.
Fortunately, Color has a hashCode() method defined already, so get() and put() will do the right thing.
docs.oracle.com/javafx/2/api/javafx/scene/paint/Color.html#hashCode()

Fri May 4

It's **very important** that you test and run your Pals (ArrayPal and HashPal) on small images first. Get them working and then move on to the BIG boys.

Here are my results for two small test images:

For image wtkrieger.faculty.noctrl.edu/csc210-spring2018/images/solid_black.png

```
ArrayPal...
     image size=40,000 pixels
     palette size=1 colors
     time to count colors=0.005 seconds
```

For image wtkrieger.faculty.noctrl.edu/csc210-spring2018/images/noctrl_xs.jpg

```
ArrayPal...
     image size=480 pixels
     palette size=471 colors
     time to count colors=0.002 seconds
```

I'll add my results for the BIG noctrl image later.

Mon May 7

I want to reiterate something that's critical on a program this challenging…

> Always be **green!**
> Always be running.

To succeed… work on one small thing at a time. Victory. Celebrate. Then next item. Ask for help if you get stuck. Work on small images first.

Count colors debugging message - Boy, array counting is slow.
I put a debugging statement in my color counting loop, so I know it's still working. It prints a blurb every 10 rows. Like this:

```
PixelReader pr = this.img.getPixelReader();
for( int y = 0; y < this.img.getHeight(); y++) {
    if( verbose && (y % 10) == 0) {
        System.out.println( "countColors() - row=" + y);
    }
    for (int x = 0; x < this.img.getWidth(); x++) {
        // count color at pixel (x, y)
    }
}
```

Time your algorithms - To time the execution time of a piece of code…

```
long startMillis = System.currentTimeMillis();
// do something
long endMillis = System.currentTimeMillis();
// total time in milliseconds is (endMillis - startMillis)
```

For nice printing of large ints (with commas), doubles (set precision), and percentages... try **NumberFormat**, docs.oracle.com/javase/8/docs/api/java/text/NumberFormat.html

```
// regular formatter helps on integers and doubles
NumberFormat nf = NumberFormat.getInstance();
System.out.println( "my count = " + nf.format( count));
// special formatter for percentages
NumberFormat npf = NumberFormat.getPercentInstance();
System.out.println( "my percent=" + npf.format( percent));
```

Crunching colors

OK - after counting colors (**only after!**), we'll crunch the number of colors.
Here are some basics of my setup. Your mileage may vary.
My steps are:

1. Create a ColorCruncher class for my algorithm(s)
2. Turn my palette into an array of entries (Palette210Entry), so I can sort them and process them.
3. Print some stats to 1) check that my setup works, and 2) get a feel for what my image palettes look like.
4. Create a new image with color changes.

ColorCruncher class - I want all my crunching code in one place for now. I definitely do **not** want it embedded in my GUI. I may end up adding more classes for more algorithms, but for now, I have just one cruncher.

```
public class ColorCruncher {
    /** the image being crunched */
    private final Image img;
    /** the color palette of the image */
    private final Palette210 pal;
    /** array of color entries from the palette */
    private final Palette210Entry[] colorArray;

    // code here
}
```

Array of entries - In the cruncher I want to turn my palette into an array of entries, so that I can play. Two steps:

1. Add method to our Palette210 interface, `Palette210Entry[] getEntryArray()`
2. Code method in my ArrayPal and HashPal classes.
    a. ArrayPal is easy… ArrayList has a toArray() method
    b. HashPal is a little tougher… lil help:
    stackoverflow.com/questions/1090556/java-how-to-convert-hashmapstring-object-to-array

I created a Comparator to sort the array by count. Arrays.sort() is very handy.
docs.oracle.com/javase/8/docs/api/java/util/Arrays.html

Stats first - Do I start crunching colors like a madman. No way. I decided to look at my array an print some stats. How many colors have a low count (like 1)? The top 256 colors… how many pixels in the image do represent? And so on...

Writable images - There's probably many ways to change your pixels. Here's one.
We know this…

> *A JavaFX **Image** has a **PixelReader** class we can use to view (and count) the pixels in an image using its **getColor()** method.*

Well, there's an analogous setup for writing images as well.

> *A JavaFX **WritableImage** has a **PixelWriter** class we can use to change the pixels in an image using its **setColor()** method.*

Nice! Here's the official Javadoc.

> docs.oracle.com/javafx/2/api/javafx/scene/image/WritableImage.html
> docs.oracle.com/javafx/2/api/javafx/scene/image/PixelWriter.html

My first attempt… change all my pixels to purple.

```
PixelWriter pw = wimg.getPixelWriter();
for( int y = 0; y < wimg.getHeight(); y++) {
    for( int x = 0; x < wimg.getWidth(); x++) {
        pw.setColor(x, y, Color.PURPLE);
    }
}
```

Set it in your GUI ImageView… purple!
If we can set all our pixels to purple, then maybe we set them to anything we like.
Tomorrow!

Sat May 12

Matcher is a nice place to start when reducing colors. It uses an existing color list and matches colors in the image to those in the list.

Here's the pseudocode:

```
// change colors in image to use only those in the colorList
matcher( image, hashPal, colorList)
        for each entry in Palette210 {
                c = entry.color
                c2 = findClosestColor( colorList, c)
                entry.replacementColor = c2
        }
        use hashPal to replace colors in image

// return the closest color in the list to a given color
Color findClosestColor( colorList, c)
        closestColor = null
        closestDist = 0
        for eachColor in colorList {
                dist = color distance from eachColor to c
                if closestColor == null  OR  dist < closestDist
                        closestColor = eachColor
                        closestDist = dist
        }
        return closest
```

- On all color crunching algorithms, use the HashPal to explore and find replacement colors. Changing the colors in the actual image should be your last step. Reason: The image is much BIGGER than the HashPal.
- I have 3 color distance methods in P3Helper class on the k: drive.
- I have 4 pre-mixed color palettes in my P3Palettes class on the k drive. The palettes are: Web safe colors, xterm colors, Nintendo (NES) colors, and grayscale colors.
- A nice Matcher quality: you can use < 256 colors if you want. In fact, I can run Matcher with a grayscale palette with 2 colors (black and white). It's cool.

I actually wrote my **Extractor** algorithm first. I played around and then noticed how easy it would be to do Matcher, so I did it. Results for Extractor are just okay right now. But it's early.

- ➔ Top256 - My first attempt was to sort the entry array by count and use the 256 most used colors. This wasn't very good. Often, images have thousands of different colors, many very close to each other. So, the top 256 usually gave mediocre results.
- ➔ Top256plusplus - Better! I am trying to pick smarter Top 256 colors. I still use the sorted array, but I only adding colors that are not "too close" to those already in my Top 256 list. Better, but I'm still playing.
- ➔ Now that I pulled Matcher out, my Extactor algorithms have a nice, common structure:

```
colorList extractedColors = Extractor()
run Matcher( image, hashPal, extractedColors)
```

Matcher is a nice first algorithm to try. It's pretty easy and you can use it with other things that create color lists later (like my Extractor).

Hope this helps. Please email me any questions.
Have a great weekend.
thanks… yow, bill