CS1110, Fall 2009. Assignment A4. Color models. Due: Midnight Friday Oct. 16 on the CMS

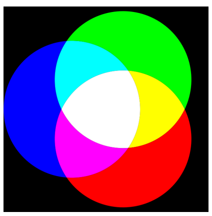
This assignment: (1) introduces three color models that are used in computing and graphics, (2) gives you practice in writing functions, and (3) introduces you to an extension of the testing and debugging methodologies you have learned about so far.

Start early on this assignment and keep track of your time. Just before you submit, tell us in a comment at the top of class `A4Methods` how many hours you took to complete this assignment.

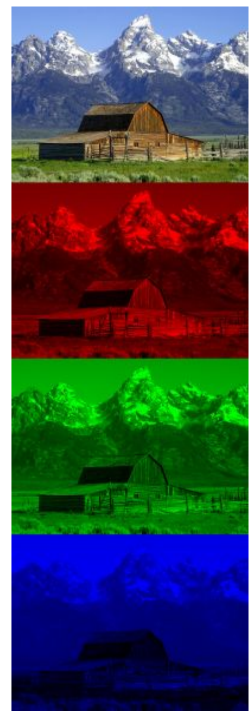
This assignment has been given before. We give it again because it is such a good one, although we have made some changes. We ask you not to cheat by obtaining a copy of the earlier solution or a version handed in by another student. Such cheating helps no one, especially you, and it makes more unnecessary work for us. We are working hard to make this a good course, and in return we expect you to be honest. We do not take kindly to cheating.

You may work with one partner. Form your group on the CMS (send invitation *and* accept invitation) *several days* before the assignment is due. Remember: it is dishonest for partners to split the work in half and work independently. Program and test and debug together, alternating driving (using the keyboard and mouse).

In writing this assignment, we made heavy use of articles on the various color models on Wikipedia.



Color model RGB



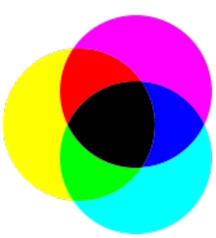
The RGB model is an additive model in which **red**, **green**, and **blue** are mixed to produce other colors, as shown in the image to the left. Black is the absence of color; white is the maximum presence of all three. In the upper right is a colored image. Below it is its separation into red, green, and blue. (Here is a high resolution version of these four images.) In the three separation panels, the closer to black a point is, the less of that color it has. For example, the white snow is made up of a large amount of all three colors, whereas the brown barn is made up of red and green with very little blue.

RGB has its roots in the 1953 RCA color-TV standards and in the Polaroid camera. RGB is used in your TV and computer monitors, and hence on web pages.

In the RGB model, the amount of each of red (R), green (G) and blue (B) is represented by a number in the range 0..255. Black, being the absence of the three main colors, is [0, 0, 0]; white, being the maximum presence of all three, is [255, 255, 255]. There are 16,777,216 different colors. Class java.awt.Color has some constants that you can use for some of the possible colors. For example, Color.magenta is [255, 0 , 255] (in the RGB model) and Color.orange is [255, 200, 0]. Web page http://en.wikipedia.org/wiki/List_of_colors gives (non-Java) names to many colors in the RGB model.

In some graphics systems on the web, RGB is used with Java **double** numbers in the range 0..1.0, so that 255 is actually given by 1.0. Later, in calculations, we convert each number in the integer range 0..255 to a **double** number in the range 0..1.0, calculate, and then convert back to 0..255.

The complementary color of RGB color [r, g, b] is the color [255-r, 255-g, 255-b]. It is like a color negative.

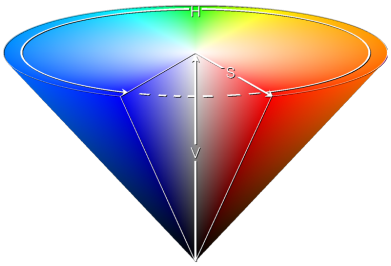


Color model CMYK

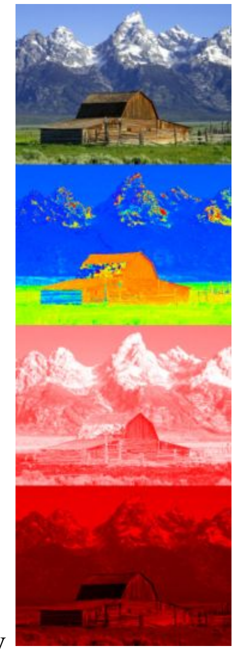
On your ink-jet printer, you buy expensive ink cartridges in the colors cyan (RGB color [0, 255, 255]), magenta [255, 0, 255], yellow [255, 255, 0], and black [0, 0, 0]. Black is referred to using K, for "Key plate". Theoretically, you need only CMY, but the K (the black) is there because without it, black doesn't look black enough. Also, to get black would require adding a lot of the CMY colors, making the paper too wet. Finally, text is usually black, so you save a lot of ink by having the additional black cartridge. Since white (the usual paper color) is the absence of the four main colors, CMYK is a "subtractive" system, in contrast to the additive system RGB.

To demonstrate, in the upper right, we show you an image; below it is its separation into cyan, magenta, and yellow. To the right of that, you see the same image separated into four components; C, M, Y, K. Much less of the CMY colors is needed to make the image when black is also used. (Here is an enlarged version of the CMY image and an enlarged CMYK image.)

In our CMYK system, each of the four components is represented by a **double** value in the range 0..1.0.



Color Model HSV (or HSB)



The HSV model, used heavily in graphics applications, was created in 1978 by Alvy Ray Smith. Artists prefer the HSV model over others because of its similarities to the way humans perceive color. HSV can be explained in terms of the cone that appears to the left.

H, the *Hue*, defines the basic color. H is an angle in the range 0 ≤ H< 360, if one views the top of the cone as a disk. Red is at angle 0. As the angle increases, the hue changes to orange, yellow, green, light blue, dark blue, violet, and back to red. The image in this paragraph shows the angles for some colors.

S, in the range 0 ≤ S ≤ 1, is the *Saturation*. It indicates the distance from the center of the disk. The lower the S value, the more faded and grayer the color. The higher the S value, the stronger and more vibrant the color.

V, the *Value*, also called the *Brightness*, is in the range 0 ≤ V ≤ 1. It indicates the distance along the line from the point of the cone to the disk at the top. If V is 0, the color is black; if 1, the color is as bright as possible.

To the right at the top is a picture. Below it we see its hue, saturation, and brightness components. The hue component shows color. The snow has color, but its saturation is low, making it almost grayish. Look at the various components of the image —the sky, the green grass, the snow, the dark side of the barn, etc.— to see how each component H, S, and V contributes. See more detail in this high-resolution version.

A GUI (Graphical User Interface) that provides understanding, and your assignment

Click on the link a4.jar to download a java application that is the desired result of this assignment (or download it from the course webpage). After downloading it, double-click its icon, and a GUI will appear on your monitor with two color panes and sliders for all the RGB, HSV, and CMYK components.

The initial color of the left pane is green, and the pane to its right contains the complementary color. In the title, you see the RGB values for green. In addition, the color pane and complementary-color pane contain information on the RGB, HSV, and CMYK values for the current colors in the panes.

On the bottom, you will see fields into which you can type numbers for the RGB, HSV, and CMYK components. After typing in their RGB numbers, click button inRGB and the panels and sliders will change to represent that RGB value; similarly for the other color models. The numbers should be in their respective ranges: 0..255 for R, G, B; 0.0..1.0 for S, V, C, M, Y, K; and 0..359.9 for H. If a negative number is input, 0 is used instead; if too large a number is input, the maximum allowed is used instead. You should use this feature when ou want to develop test cases. Your can type in values and see what *our* program does and then add test cases to see whether your program does the same.

When you move one slider, the colors change accordingly —and other sliders change too, so that all three models (RGB, CMYK, and HSV) always register the same color.

As a first move, very slowly move the Hue slider H up and watch how the colors (and the other sliders) change. Then change the saturation S and value V sliders to see their effect. Play around like this, while reading about the color models, to get an understanding of how they relate to each other. It's neat!

Code you need to write

Here are five "skeleton" files for classes `A4`, `HSV`, `CMYK`, `A4Methods`, and `A4Tester`. Download the .zip file, either from here or from the course webpage, store the unzipped files in a new directory, compile them in DrJava, and execute `A4.main(null);`. You will see the GUI as discussed above, but the text will not display properly and the two panels will be green. Also, the RGB sliders work, but the other sliders have no effect. Your job is to write and test out, *one by one*, the methods in class `A4Methods`. As you do, more and more of the GUI will work properly.

Because we have now advanced to more complex programs, you should employ the following **enhanced testing methodology**. First, as usual, you should write test cases in class `A4MethodsTester` using `assertEquals` statements *before* you write the method body, for the purposes of checking that your method has the correct final behavior. (In the skeleton of class `A4Tester` we have put a few test cases in for you, but you should add more.) Also, as usual, organize your test cases into logically-coherent test procedures. But *in addition*, as you write each method, add test cases to ensure there is complete test coverage of the "internals" of the method: *each statement or expression within the method body should be exercised during at least one test case*. Note that this means that it's fine for one test case to exercise multiple statements or expressions; but it is not OK for some statement/expression (such as the contents of an "else" clause) to not be tested at all.

We also introduce the following **debugging idea**. We strongly suggest that you use `System.out.println` statements to help you track down bugs. Procedure `System.out.println` prints out the String given to it as an argument. Here is an example of its use. Suppose you are trying to find an error in your method RGB2HSV (or in a method that calls RGB2HSV) and that RGB2HSV changes a variable h at line 36. Then, you might insert at line 37 a statement like

```
System.out.println("RGB2HSV: h is " + h);
```

When the method is executed, this `println` statement causes information to be printed out about the value of h within method RGB2HSV, and if that value isn't what you expected, you know something about the source of the error you're tracking down!

Incidentally, if you import System.out.*, then you can write `println` instead of `System.out.println`.

About objects that contain RGB, CMYK, and HSV values.

(a) Class `java.awt.Color` is used to maintain RGB colors (use `import java.awt.*;` to access this class). Create a new RGB color object using `new Color(r, g, b)`, where `r`, `g`, and `b` are in the range 0..255. Given an RGB object `rgb`, obtain its three components using `rgb.getRed()`, `rgb.getGreen()`, and `rgb.getBlue()`.

(b) Use the given classes `CMYK` and `HSV` for objects that contain CMYK and HSV. Take a look at the classes; their use should be obvious. Note that the toString functions call functions in A4Methods, which you will write.

Write (AND TEST) your functions, one at a time, in the following order. Read the text below, but also make sure you carefully read the specifications and comments in the provided code; while you should always do so anyway, in this assignment, sometimes we give hints in the specs/comments about how best to write your functions.

0. Function complementRGB(Color). We gave you some partial code that you need to fix. We provide a test case for it in class A4Tester (should you add more?), which won't work until you rewrite the function.

1. Function truncate2Five(double). There is a stubbed-in return statement in it so that the program compiles. Write this function, using the guidance given as comments in the body. We provided test cases for it in class `A4Tester (should you add more?)`. This function will be needed later.

2. Function round2Five(double). This is the function we really want to use, and it is most easily written by calling truncate2Five appropriately. There is a stubbed-in return statement in it so that the program compiles. Write this function, using the guidance given as comments in the body. We provided test cases for it in class `A4Tester` (should you add more?). This function will be used later to get values to appear in the GUI in a consistent format --always 5 characters long.

2. Function toString(Color). Write this function. When this is written, RGB values will appear in the title of the GUI and in the two color panes. We provided one test case for this function in class `A4tester`; that is all that is needed.

3. Function toString(CMYK). Write this function. When written, CMYK values will appear in the two color panes. This function should call function `round2Five` to round each CMYK value to 5 characters. We do NOT provide test cases for this function. You write at least two of them.

4. Function toString(HSV). Write this function. When written, HSV values will appear in the two color panes. This function should call function `round2Five` to truncate each HSV value to 5 characters. We do NOT provide test cases for this function. You write at least two of them.

5. Function RGB2CMYK. This function converts an RGB value to a CMYK value. When you get it working, moving the RGB sliders will cause the CMYK sliders to move as well. There are several different ways to convert, depending on how much black is used in the CMYK model. Our conversion uses as much black as possible. Let R, G, and B be the color components of the RGB in the range 0..1.0 (not 0..255!!). Then the conversion is as follows:

1. Compute $C' = 1 - R$, $M' = 1 - G$, and $Y' = 1 - B$.

2. If C', M', and Y' are all 1, use the CMYK value (0, 0, 0 , 1).
   If not, compute and use:
   K = minimum of C', M', and Y',
   $C = (C' - K) / (1 - K)$, $M = (M' - K) / (1 - K)$, $Y = (Y' - K) / (1 - K)$.

That's it! That's not too bad, is it? Providing test cases is a bit problematic because **`double`** values are only approximations to the real values, and slightly different ways of computing might produce different results. To show you how to do this, we provide in `A4Tester` three test cases, testing only the rounding of the values to 5 characters (except in the two first, extreme cases), since that is what appears in the color panes of the GUI.

6. Function CMYK2RGB. This function converts a CMYK value to an RGB value. When you get it working, moving the CMYK sliders will cause the RGB sliders to move as well. Let C, M, Y, and K be the color components of the CMYK value, all in the range 0..1.0. Then the conversion is as follows:

$$R = (1 - C)(1 - K), \quad G = (1 - M)(1 - K), \text{ and } B = (1 - Y)(1 - K).$$

This produces RGB values in the range 0..1.0, and they must be converted to the range 0..255. *Use rounding*. We'll say it again because in previous semesters students have not complied: *USE ROUNDING*.

Put at least two test cases for this function in class `A4Tester`. To figure out what test cases to use, you can use the GUI that we provided (execute a4.jar) in order to find out what the values in one color model should be in a different color model..

7. Function RGB2HSV. This function converts an RGB value to an HSV value. *Warning:* class Color has a function `RGBtoHSB`, but it produces output in a different format than what we want (and uses a construct you haven't learned about yet). So, in this case, do *not* use this pre-existing code. Our experience it is easier (and more educational) for students to write this function from scratch than to figure out how to correctly employ Color.RGBtoHSB.

When you get it working, moving the RGB sliders will cause the HSV sliders to move as well. Here's how the conversion works.

Let `MAX` be the maximum and `MIN` be the minimum of the (`R`, `G`, `B`) values. `H` will satisfy `0 ≤ H ≤ 360` and `S, V, R, G`, and `B` will be in `0..1`.

H is given by 5 different cases:

(a) MAX = MIN: H = 0.
(b) MAX = R and G ≥ B: H = 60.0 * (G - B) / (MAX - MIN) .
(c) MAX = R and G < B: H = 60.0 * (G - B) / (MAX - MIN) + 360.0
(d) MAX = G: H = 60.0 * (B - R) / (MAX - MIN) + 120.0
(e) MAX = B: H = 60.0 * (R - G) / (MAX - MIN) + 240.0

S is given by: if MAX = 0 then 0, else 1 – MIN/MAX.

V = MAX.

You have to provide at least 5 test cases in class A4Tester, so that each expression in the cases for H are evaluated in at least one test case.

8. Function HSV2RGB. This function converts an HSV value to an RGB value. When you get it working, everything in the GUI should work.

Let `Hi = floor(H/60)`, `f = H/60 – Hi`, `p = V(1-S)`, `q = V(1-fS)`, `t = V(1-(1-f)S)`.

Then `R`, `G`, and `B` depend on the value `Hi` as follows:

If `Hi = 0`, then `R = V, G = t, B = p`
If `Hi = 1`, then `R = q, G = V, B = p`
If `Hi = 2`, then `R = p, G = V, B = t`
If `Hi = 3`, then `R = p, G = q, B = V`
If `Hi = 4`, then `R = t, G = p, B = V`
If `Hi = 5`, then `R = V, G = p, B = q`

This produces RGB values in the range 0..1.0, and they must be converted to the range 0..255. *Use rounding*. We'll say it again because in previous semesters several students have not complied: *USE ROUNDING*.

Provide at least 6 test cases for this function because of the 6 possible values of `Hi`.

Note: Wikipedia uses

Hi = floor(H/60) % 6 and f = H/60 - floor(H/60).

Because H is in the range 0 <= H < 360 (degrees), the two formulas given below have the same value, so we use the simpler one.

floor(H/60) % 6 and floor(H/60)

Submission of the assignment

We remind you to place a comment at the top of class A4Methods that indicates how much time you spent on this assignment. Make sure that class A4Methods is indented properly. Make sure that class A4Tester has the appropriate test cases. Submit files A4Methods.java and A4Tester.java on the CMS by the due date.

We hope you enjoyed this assignment and found it instructive.