The objective is to add several methods to a class (by writing code) and demonstrate that those methods work as specified. We will explain in English: The methods will re-color Pixels from your picture in increasingly complex combinations. So, if you have not done Lab05, just go on with Lab06 BUT QUICKLY DO the Lab05 make-up steps when you come to them. For each of 4 lab tasks and the followup task below, we will explain:

1. What each method should be named, what the type of each parameter is, what the method should do when it is called, and how the parameters vary a detail of what it does. (Think about and discuss how this explanation is useful for somebody who did NOT write the method.)
2. How the method should do what it does (the algorithm). We already taught things like loops, so when we write "make a loop" you might have to review your project work, notes and/or ask a TA or classmate for help.

1. Make a new directory named Lab06. Get a copy of your chromakeyed picture into that directory.

   `cd CSI201`
   `mkdir Lab06`
   `cd Lab06`

   If you did not do Lab05, consult the Lab05 instructions, ask TA (copy from /usr/local/depts/cs/geintro/Fall13Photos/sc111111.JPG etc)

   `cp ../Lab05/sc111111.JPG .`

   (First read and try to remember the handful of command line tech tips below. Of course, use your own NetId in place of sc111111.JPG).

   See `cp` and some spaces, and then `../` with no spaces. That means: The copying command (`cp`) will start the path to your picture one DIRECTORY TREE LEVEL UP from your current working directory. Two dots means one step up.

   The next `Lab05` (no spaces before it) means it will continue into your `Lab05` directory (which should be one step below your CSI201 directory).

   The file name ending in `.JPG` (no spaces before it) specifies the name of the file to copy (which should be in your `Lab05` directory).

   After the `.JPG` there ARE SPACES and ONE SINGLE DOT. That last dot means that the destination for the copy is the current working directory. (In shell language, the single DOT means current working directory. Two dots `..` means the directory one step up from .)

   `ls -l` (That's hyphen - letter ell, not one, for LONG listing.) Sanity CHECK THAT YOU HAVE the copy right there in your current working directory; get help if you don't.)

2. Start DrJava, (reminder: /usr/local/depts/cs/geintro/drjava )

3. NEW: set certain preferences so less fancy but MORE EFFICIENT graphics are used. (Thanks for Andrew P. for finding out this tip.) Fewer bits processed by the itsunix servers and sent through UAs network make the response times faster for everybody.

   1. Edit->preferences -> display options -> toolbar buttons: text only
   2. Edit->preferences->display options->change the "look and feel" option to com.sun.java.swing.plaf.motif.MotifLookAndFeel
4. In the same way you extended G&E's `Turtle` class to define your own `ArtisticTurtle` class, you shall extend their `Picture` class to define your own `PaintablePicture` class. `PaintablePicture` inherits all the methods from `Picture` like `getPixel(int x, int y)`

It shall be a container holding the methods for new `Picture` operations you shall create and demonstrate today. Here's how:

```java
public class PaintablePicture extends Picture {
    public void testMeth(int parVar) {
        System.out.println("ArtisticTurtle's testMeth called with parameter value " + parVar);
        System.out.println("called ON object " + this);
        System.out.println("this Picture has width " + this.getWidth() + " and height " + this.getHeight());
        return;
    }
    public PaintablePicture(String parVar) { super(parVar); }
    public static void main(String[] a) {
        System.out.println("DONT RUN ME:I DONT WANT TO BE AN APP!");
    }
}
```

5. Start your demo app. and make sure (getting help to get it done fast) it works:

```java
public class Lab06Demo {
    public static void main(String[] a) {
        FileChooser.pickMediaPath();
        String filename = FileChooser.pickAFile();
        PaintablePicture myPref = new PaintablePicture(filename);
        myPref.testMeth(38);
        //The space below is for calling methods you will write today.

        myPref.explore(); //ALWAYS EXPLORE LAST!!
    }
}
```

6. Lab05 Makeup (and reminder to everybody) Add and test this code (only if you didn't already do Lab05): Just before calling the `explore()` method:

```
java.awt.Color purple = new java.awt.Color(175, 0, 175);
myPref.getPixel(0, 5).setColor(purple);
```
Get the app to run, magnify the explore, and make sure you see the purple pixel at location (0, 5), near the upper right hand corner. Then add and test that you can put a purple spot on your nose (code this BEFORE `explore( )` of course!)

```java
myPref.getPixel(???, ???).setColor( purple );
myPref.getPixel(???,+1, ???).setColor( purple );
myPref.getPixel(???,+1, ???-1).setColor( purple );
myPref.getPixel(???-1, ???+1).setColor( purple );
myPref.getPixel(???-1, ???-1).setColor( purple );
```

You must first use the explorer window to find the x, y Pixel coordinates locating a `Pixel` near the middle of your nose, and fill in those two integers instead of the `??` above! (`??` in the spot for an expression is illegal Java syntax).

(Note: `myPref.getPixel(???, ???).setColor( purple );` is a shorter way of coding `pixRef = myPref.getPixel(???, ???); pixRef.setColor( purple );` and it doesn't need the variable named `pixRef`.

1 **Purple spot method**

- Method name: `purpleSpot`
- Parameters and types: `int x` and `int y`
- It should make a purple spot as your code from Lab05 (or above) did. You might use the same size and shape, or make an improvement if you like.
- Parameter value effects: The values of x and y should locate where the purple spot should be.
- How: Write `public void purpleSpot(int x, int y) {` before the spot making code from Lab05 (like the 5 lines above), and remember the `}` after it. Remember to put in code to declare and put a `java.awt.Color` reference value into the `purple` variable (just code `purple = new java.awt.Color( ..etc., and MAKE SURE:

  - `this` is used to refer to this `Picture` for getting a `Pixel`. For example code

    ```java
    Pixel pixRef;
    pixRef = this.getPixel( x, y );
    pixRef.setColor( purple );
    ```

    instead of

    ```java
    pixRef = pRef.getPixel( x, y );
    ```

  - Optional shorter form:

    ```java
    this.getPixel( x, y ).setColor( purple );
    ```

- Review what parameter variable names you used (I used `x` and `y`) and use mathematical expressions involving those parameter variables to help locate which `Pixels` to make purple. DON'T KEEP USING THE PLAIN NUMBERS from Lab05 (for example, code `(x+1, y-1)` and NOT `(321+1, 205-1)`
- TEST AND DEMONSTRATE that spot like last week: Note again where your nose is and use its coordinates in a CALL OPERATION CALLING `purpleSpot` that you code in `Lab06Demo`
2 Purple horizontal line method

- Method name: `purpleHLine`
- Parameters and types: `int xmin, int y, and int length`
- It should make a purple horizontal line of width 1 (pixel).
- Parameter value effects: The leftmost pixel to be colored purple should have location (coordinates) `xmin` and `y`. The number of pixels to be made purple should be `length`. In other words, `length`, a number of pixels at `(xmin, y), (xmin+1, y), (xmin+2, y) up to (xmin+length-1, y)` should be made purple. But, you MUST do it in a LOOP!
- How: Write the method definition with the 3 parameter variable declarations, and begin and end the body with `{ }`. Then write code to get the purple `Color`. Follow that by code for a LOOP whose body repeats length times and colors purple just one the correct pixels, each time it is repeated.
- TEST AND DEMONSTRATE like the spot last week: Note again where your nose is. Make the purple line start about an inch left of your face, go through your nose, and come out about symmetrically on the other side of your face.

3 Purple vertical line method

- Method name: `purpleVLine`
- Parameters and types: `int x, int ymin, and int length`
- It should make a purple vertical line of width 1 (pixel).
- Parameter value effects: The topmost pixel to be colored purple should have location (coordinates) `x` and `ymin`. The number of pixels to be made purple should be `length`. In other words, `length`, a number of pixels at `(x, ymin), (x, ymin+1), (x, ymin+2) up to (x, ymin+length-1)` should be made purple. Again, MUST do it in a LOOP!
- How See how above!
  - Tip for programming speed: Copy and paste your code, and then edit it to make the line vertical instead of horizontal. You MUST change the names of the parameter variables for the sake of clarity to a human reader!
- TEST AND DEMONSTRATE: Make the vertical line be visible anywhere you like.

4 Thick Purple horizontal line method

- Method name: `purpleThickHLine`
- Parameters and types: `int xmin, int y, and int length`
- It should make a purple horizontal line of width 3 (pixels).
- Parameter value effects: The top and left most pixel to be colored purple should have location (coordinates) `(xmin and y-1)` and the bottom and right most pixel should have location `(xmin+length-1, y+1)`. The number of pixels to be made purple should be `length*3`. In other words, the thick line is composed of three thin lines of the parametrized length. The first thin line should begin at `(xmin, y-1)`, the second at `(xmin, y)` and the third at `(xmin, y+1)`
How: Write the method definition with the 3 parameter variable declarations and begin and end the body. Then write code to get the purple Color. Now you have a choice for how to do it: (Idea: you and a neighbor decide so you do it one way and your neighbor do it the other. Then compare your answers.)

- Way 1: Use a loop again. Do it fast: First, copy your code from `purpleHLine`. Add two code lines into the body of the loop. Put them near your original code to color one pixel. One of them should color the pixel right above that one pixel. The other should color the pixel right below that one pixel.

- Way 2: Code the body of `purpleThickLine` with NO LOOP AT ALL. Simply code `this.purpleHLine( ..., ... );` THREE separate times with suitable expressions for the parameter values in each of the three calls. Tip for programming speed: Copy and paste your code, and then edit it to make the line vertical instead of horizontal. You MUST change the names of the parameter variables for the sake of clarity to a human reader!

- TEST AND DEMONSTRATE: Make the vertical line be visible anywhere you like.

**Lab06 Followup Task: Chromakey controlled purple horizontal line.**

- Method name: `purpleHLineOnGreenOnly`
- Parameters and types: `int xmin, int y, and int length`
- It should make a purple horizontal line of width 1 (pixel). However, it should recolor ONLY pixels that are green, leaving the others alone. The effect will be that the line drawn through your face will appear ONLY BEHIND YOU.

  Study G&E's programs 45 and 46 for clues to how to use an if statement to make the decision to color a Pixel purple or just skip it. Those clues tell how to code getting the red, green and blue intensity values from a Pixel, and make the computer add and compare them to make a decision about whether a Pixel is really green or not. You will have to interchange the roles of green and blue, since Prof. Guzdial's screen was blue. Find those programs on pages 489-491 of the UA custom edition or pages 203-205 in the red covered paper edition. (Ask a friend, TA or the prof to show you that if you don't have the G&E material handy.)

  - Parameter value effects and how: Just like in purpleHLine except the recoloring instruction line is enclosed within the BODY (between the `{ ... }`) of an `if`-statement.

  - TEST AND DEMONSTRATE like the spot last week: Note again where your nose is. Make the purple line start about an inch left of your face, go through your nose, and come out about symmetrically on the other side of your face. BUT..the line should appear BEHIND your head!

  - For LAB CREDIT, upload your two Java files and two class files to Blackboard under Lab06 followup.

For the future: Develop a method that has a Color parameter plus parameters that determine a rectangle (with horizontal and vertical sides). The method should color in the given rectangle with the given color. Super challenge: Remove the horizontal and vertical restriction. Tip: Look up line rasterization on the web, and/or [http://groups.csail.mit.edu/graphics/classes/6.837/F02/lectures/6.837-7_Line.pdf](http://groups.csail.mit.edu/graphics/classes/6.837/F02/lectures/6.837-7_Line.pdf)