Lab 4

Introduction to
Computer Science
ICSI201
Fall 2012

SUMMARY

Picture, **Pixel**, and **Color** objects. Just read & do, 1 thing at a time. 1 attendance point for showing the TA all questions on the handout ANSWERED; plus 1 for attending.

PRE-LAB AND AFTER LAB STUDY

You're expected to be able to reproduce from previous labs: (a) Making a directory under your CSI201 directory for Lab04 exclusively, (b(writing, saving into that directory and running Java programs, and (c) having the bookClasses ready to use. Secs. 3.6.1, 3.6.2, 4.2 of the book fill in a few more lab topic details. The Albany way: you will save your work in the **main()** method, NOT runs commands manually as in the book. Just read and follow the directions below one at a time, and with care.

START DRJAVA

Start DrJava with the following command:

```
/usr/local/depts/cs/geintro/drjava
```

OPEN A PICTURE

To start, create a Hello World program and verify it compiles and works. What is the name of class containing your program_____ ?and what is the name of the **.java** file containing its class definition_____(Please write the **.java** extension too.) ?

First program the code below and write what it prints at each step: Remember: Type the code **between** **main**'s outer onion rings: **public static void main(String a[ ])** { ... }

Obtain a ticket (a **variable**) from the supply room whose value can refer to a **String**. Digital image files stored on a computer disk, etc., like all other files, are located by a pathname, which is a string of characters. Study and copy the code below to make the variable, document its purpose, and try to print what is written on the ticket, i.e., print the value of the variable **fileName**:

```
String fileName;  //The PURPOSE is to refer to the name of a file.
System.out.println(fileName);
```

Write the description of the error:_____
This kind of error is a compiling or compile time error.

Add the code to make sure **fileName** is initialized (which should fix the compile time error):

```
fileName = "/usr/local/depts/cs/geintro/mediasources/beach.jpg";
```

(The sequence MUST BE: (1) Obtain the ticket, uninitialized. (2) Initialize it. (3) Print what's on it.)

Now for a **Picture**. Code the commands: Obtain a 2nd ticket, uninitialized, suitable for referring to a **Picture** object. Next, make a new **Picture** object that encodes of the data from the above digital image file. Finally, make sure the computer can refer to that **Picture** in the future by copying its location onto your 2nd ticket. The code below does all that:

```
Picture myPic;
myPic = new Picture(filename); //1 line to make a Picture AND copy.
```

Finally, code the command to call the **show** method on that **Picture** just made, which calls a method of the Picture programmed to make the computer to display the image to YOU:

```
myPic.show();
```

(Whether or not it works, turn over right away for both help and to go on!)

When you get the program to compile (***often*** you will have to correct more syntax errors), and command the computer to "run" it, and all goes well, you will see an image!

I anticipate one popular runtime BUG. If your program has this bug, the computer will print

```
There was an error trying to open /bla/blabla.jpg
```

and show in the new window: **`couldn't load /bla/blabla.jpg`**, white on black, instead of a pretty beach scene. The cause of this bug is that you didn't type EXACTLY the pathname of file containing the image of the beach. We can't anticipate what reading and typing mistakes students will make, so we wrote "**`/bla/blabla.jpg`**" to represent a generic, mistyped pathname.

IF you get ANY OTHER kind of runtime bug, get help from a neighbor or a TA (don't waste time!)

REUSE VARIABLES

"Reuse a variable" is Guzdial and Ericson's way of saying: "Erase what's on a ticket and then write on on that ticket again". The Java assignment operator **`=`** combines erasing with re-writing.

So, you can reuse **`fileName`** and **`myPic`** to refer to different **String** and **Picture** objects without "declaring new variables." That means simply you do not demand (and make the Army pay for!) two more tickets beyond the two (named **`filename`** and **`myPict`**) you already have on your desk.

This time we command the file chooser to pick the file name. Use the following commands and select the **`butterfly.jpg`** image from the same directory
(**`/usr/local/depts/cs/geintro/mediasources`**)

```
fileName = FileChooser.pickAFile();
myPic = new Picture(fileName);
myPic.show();
```

Notice that you didn't declare any new variables this time. What happened to the beach picture? What did G&E program **`show( )`** to do: Pop up a 2nd window?_____ Modify the 1st? _____

WORK WITH THE PICTURE

Now let's work with the butterfly image. What are its dimensions? Add the code:

```
System.out.println( myPic.getWidth() );//Purpose: Tell people the width.
System.out.println( myPic.getHeight() );
```

You've added code to make your program print the dimensions. What did your program print for the dimensions? Width_____ Height_____

WORK WITH ONE PIXEL AT A TIME

As introduced in the textbook, a digital image, and the data values stored in a **Picture**, are organized into a grid of pixels. (Very different from reality, paintings or etchings!) You can get at a specific pixel by addressing it with the method
getPixel(*X-COORDINATE*, *Y-COORDINATE* **)**. Try it out:

```
System.out.println( myPic.getPixel( 0, 0) );
```

Which pixel is this? Notice how the pixel's information is described in terms of red, green, and blue intensities. Remember, intensity values range from 0 to 255. You create a custom color by creating a **Color** object with **new java.awt.Color(** *RED_LEVEL*, *GREEN_LEVEL*, *BLUE_LEVEL* **)**.
First you might import the Color class so you can more easily work with **Color** objects:

```
import java.awt.Color; //put this at the TOP of your .java file.
```

If you don't, you will have to type **java.awt.Color** each time, instead of **Color**. So most people do.

Let's use the following code to create an object to recolor a **Pixel**:

```
Color brightCol = new Color(255,0,0);
```

What color will this be? Red?_____ Green?_____ Blue?_____ (check one.)

Ok, now let's put it all together. You can set or change the color of one single pixel with the **setColor(***Color reference***)** method. After you change the color , you need to command with the *pictureRef*.**repaint()** method to display the changes. Try it out:

```
myPic.getPixel(0,0).setColor(brightCol);
//Single line of code to (1) Get at a Pixel and then (2) set its color
myPic.repaint( );
myPic.explore( );  //Purpose: Let a person analyze Picture details!
```

Explore butterfly picture. Do you see the change? It is small, but this is the size of one pixel.

Pay special attention to the first command of the last three. There is a lot happening here. It calls TWO object methods, but on which objects are they called? Which object is **getPixel()** a method of? How about **setColor()**? Hint: those 2 objects are different! If you don't understand this statement, ask your TA.

TRY IT ON YOUR OWN

Now try this out on your own. Using the commands from above (but changing the parameters) set the bottom right pixel to blue. Hint: You should not declare any new variables.
If you have any problems ask for help!

GET CREDIT

Make sure the TA records your attendance. After lab, ARCHIVE UP and upload to Blackboard the Lab04.zip or other archive containing your Lab04 directory and in it, a .java file which, when the TA compiles and runs it, reproduces the lab work described above.