

## Computational Media

### Project 01: Graphics and StdLib Setup

#### Marist School

#### Description:

In this class we will use Java, Eclipse, and Standard Library (stdlib.jar) from Princeton. The combination of Java and the Standard Library will provide a platform for us to run programs that use images and sound.

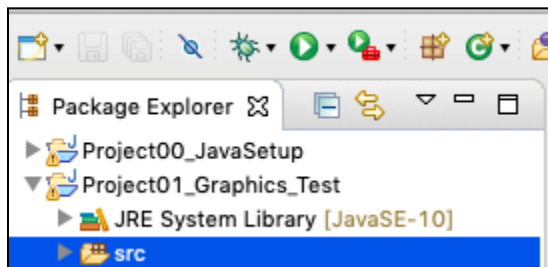
The Standard Library provides a wide array of tools for data visualization, statistics, and reading / writing various media files. The API for Standard Library can be found at:

<https://introcs.cs.princeton.edu/java/stdlib/>

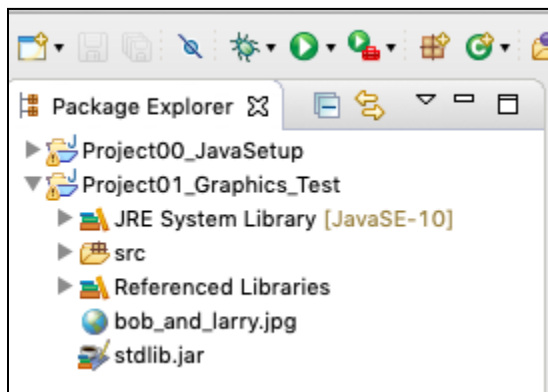
In this assignment you will setup and test Eclipse, Java, and the stdlib.jar. Then you will code several short examples to test and demonstrate the Standard Library. We will use StdLib and some of the Princeton lessons later in the Term.

#### Process:

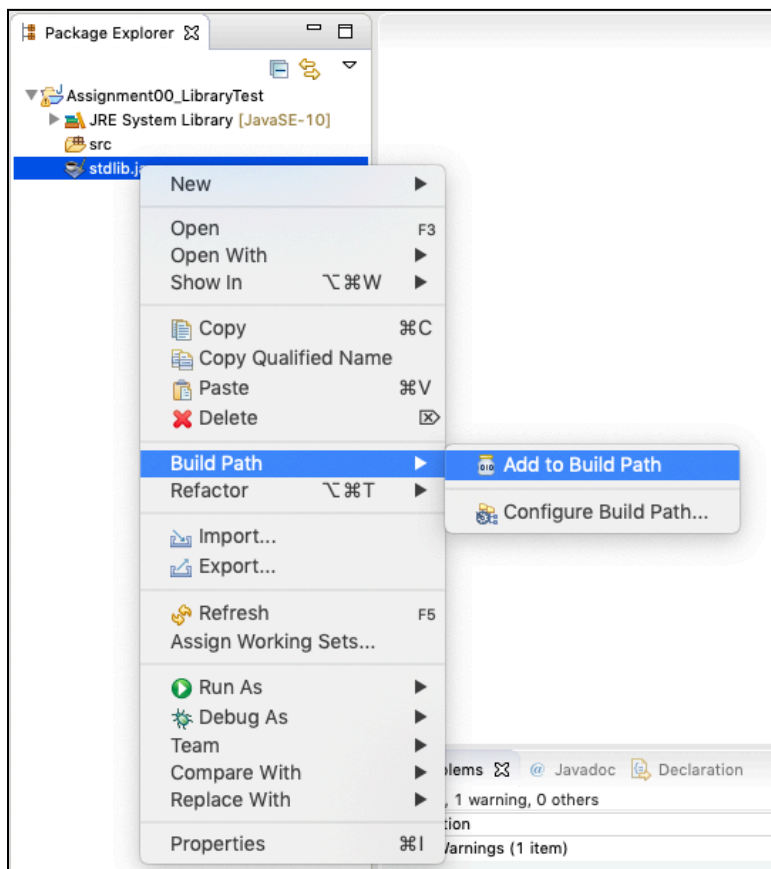
1. Download the stdlib.jar file from the Google Classroom or this link: [stdlib.jar](#)
2. Create a new Java Project in Eclipse and call it "Project01\_GraphicsTest"
3. Expand the project hierarchy in the "Package Explorer" until you see the "src" folder.



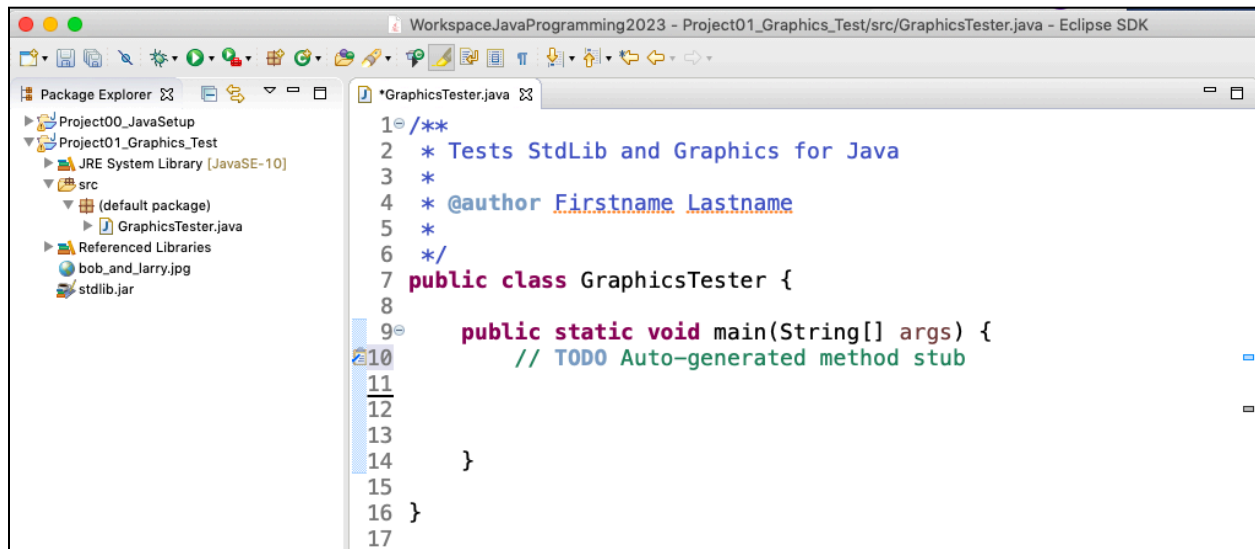
4. The `stdlib.jar` file should be in your Downloads folder. Copy `stdlib.jar` into the `Assignment00_LibraryTest` folder.



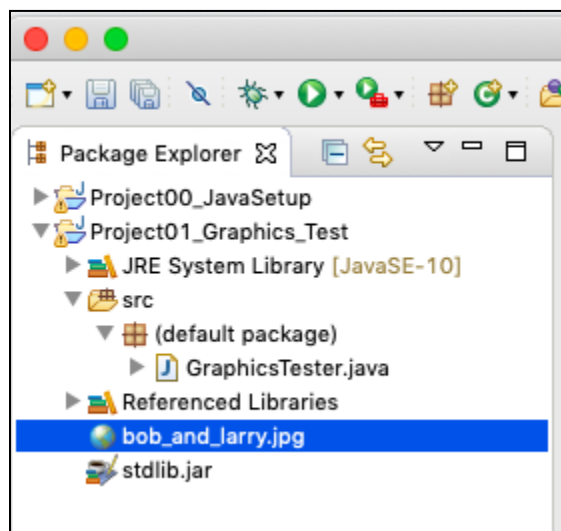
5. We now need to add the `stdlib.jar` file to the build path of the project. Right click on `stdlib.jar` and select “Build Path -> Add to Build Path”



6. With `stdlib.jar` on the build path, we will now create a Java class that will test features of the library. Right click on the “src” folder and create a new Java class called “GraphicsTester.java”. Add JavaDoc Comments as shown below:



7. We will now test the ability to read and display images. Find a small .jpg image file and place it into the project root directory. You can use “bob\_and\_larry.jpg” from the Google Classroom as a sample. ([https://nebomusic.net/javalessons/bob\\_and\\_larry.jpg](https://nebomusic.net/javalessons/bob_and_larry.jpg) )



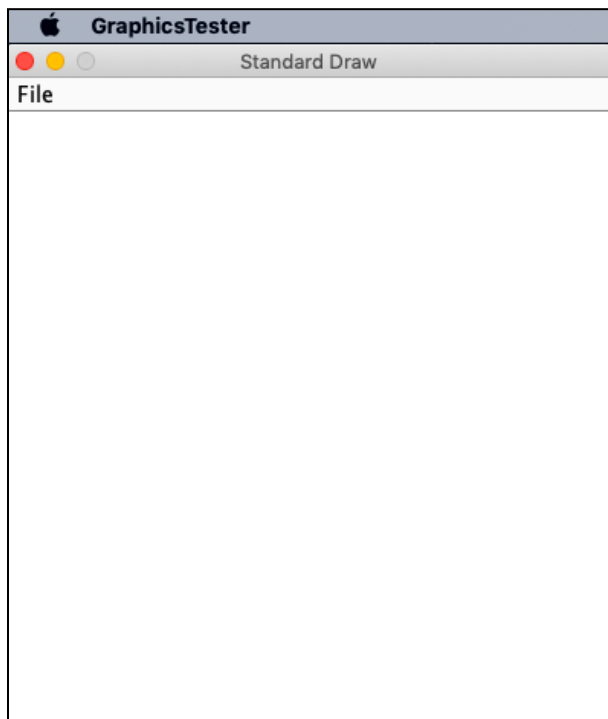
8. Examine the API for StdDraw at:

<https://introcs.cs.princeton.edu/java/stdlib/javadoc/Draw.html> .

We will now establish a canvas with dimensions of 400 by 400 pixels. The code below serves as an example (Lines 10 to 21):

```
*GraphicsTester.java
5  *
6  */
7  public class GraphicsTester {
8
9  public static void main(String[] args) {
10     // Setup Canvas variables for size
11     int width = 400;
12     int height = 400;
13
14     StdDraw.setCanvasSize(width, height);
15
16     // Set the Scale and Origin: Upper Left
17     StdDraw.setXscale(0, width);
18     StdDraw.setYscale(height, 0);
19
20     // Show Canvas
21     StdDraw.show();
22
23 }
24
25 }
26
```

9. Click the “Run” button and you should see the blank Canvas. Make sure to “close” the window when finished viewing.





9. Examine the API for StdDraw at: <https://introcs.cs.princeton.edu/java/stdlib/javadoc/Draw.html>

Notice there are three ways to draw an image onto the canvas:

void	<code>picture(double x, double y, String filename)</code>	Draws the specified image centered at (x, y).
void	<code>picture(double x, double y, String filename, double degrees)</code>	Draws the specified image centered at (x, y), rotated given number of degrees.
void	<code>picture(double x, double y, String filename, double scaledWidth, double scaledHeight)</code>	Draws the specified image centered at (x, y), rescaled to the specified bounding box.
void	<code>picture(double x, double y, String filename, double scaledWidth, double scaledHeight, double degrees)</code>	Draws the specified image centered at (x, y), rotated given number of degrees, and rescaled to the specified bounding box.

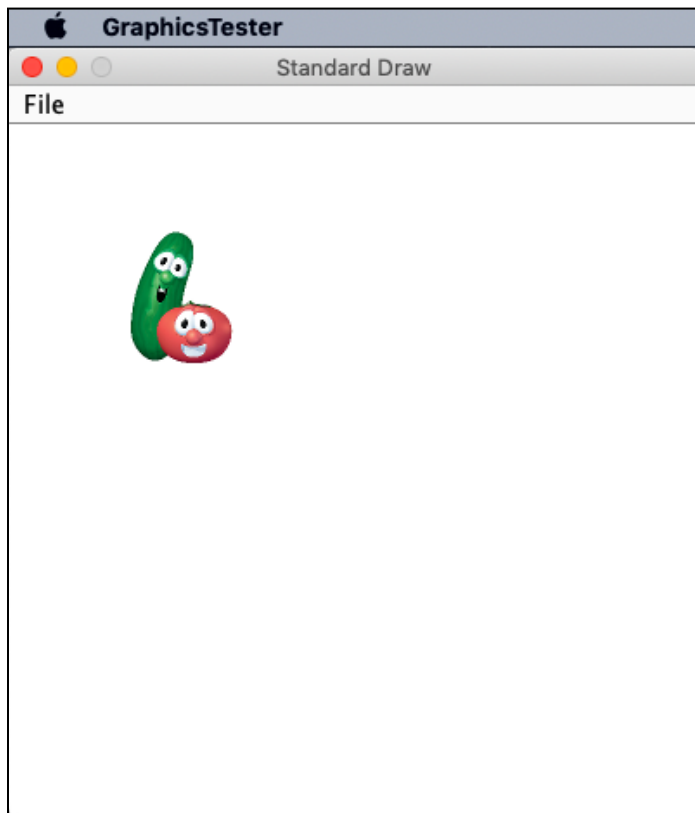
We will use

`picture(double x, double y, String filename, double scaledWidth, double scaledHeight)`

To draw a picture onto the canvas. The parameters (variables inside the parenthesis) describe how the values are used to control the properties of location and size of the image. Example code is show below: (Lines 20 to 24)

```
*GraphicsTester.java
1 /**
2  * Tests StdLib and Graphics for Java
3  *
4  * @author Firstname Lastname
5  *
6  */
7 public class GraphicsTester {
8
9     public static void main(String[] args) {
10         // Setup Canvas variables for size
11         int width = 400;
12         int height = 400;
13
14         StdDraw.setCanvasSize(width, height);
15
16         // Set the Scale and Origin: Upper Left
17         StdDraw.setXscale(0, width);
18         StdDraw.setYscale(height, 0);
19
20         // Variable pointing the the Bob and Larry file
21         String imgPath = "bob_and_larry.jpg";
22
23         // Draw Bob and Larry picture size 60, 80
24         StdDraw.picture(100, 100, imgPath, 60, 80);
25
26         // Show Canvas
27         StdDraw.show();
28     }
29 }
30
31 }
32
```

10. Click the “Play Icon and you will see the image”:



12. The StdLib also has methods for drawing lined and filled shapes:

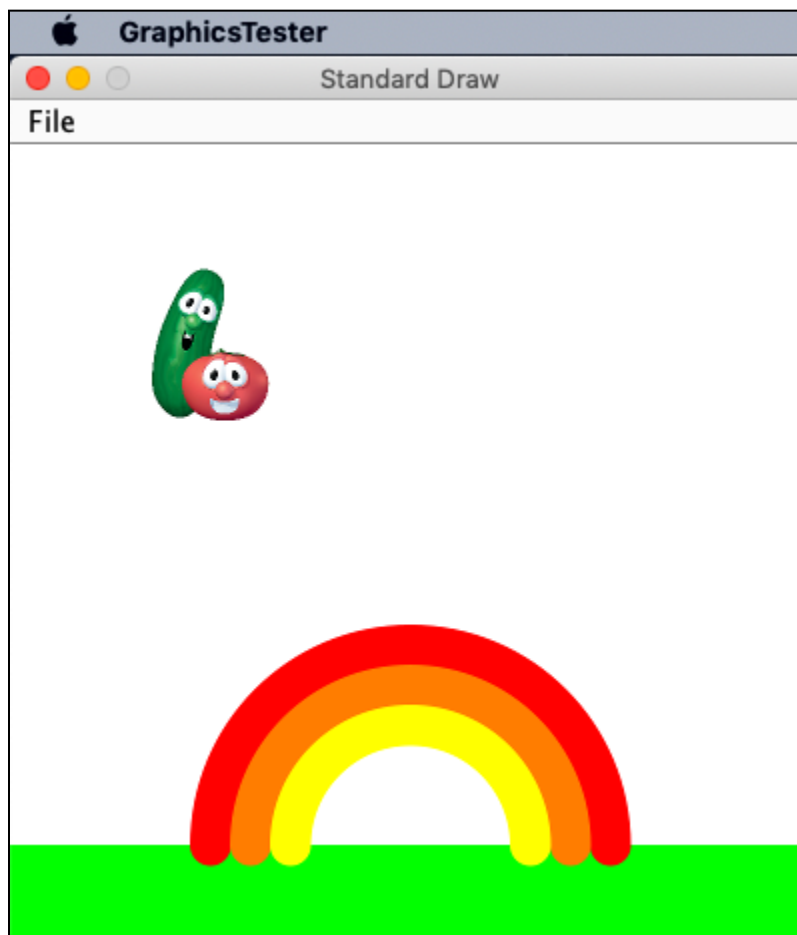
```
setPenColor(int red, int green, int blue)
setPenRadius(double radius)
```

```
arc(double x, double y, double radius, double angle1, double angle2)
circle(double x, double y, double radius)
ellipse(double x, double y, double semiMajorAxis, semiMinorAxis)
square(double x, double y, double halfLength)
rectangle(double x, double y, double halfWidth, double halfHeight)
polygon(double [] x, double [] y)
```

```
filledArc(double x, double y, double radius, double angle1, double angle2)
filledCircle(double x, double y, double radius)
filledEllipse(double x, double y, double semiMajorAxis, semiMinorAxis)
filledSquare(double x, double y, double halfLength)
filledRectangle(double x, double y, double halfWidth, double halfHeight)
filledPolygon(double [] x, double [] y)
```

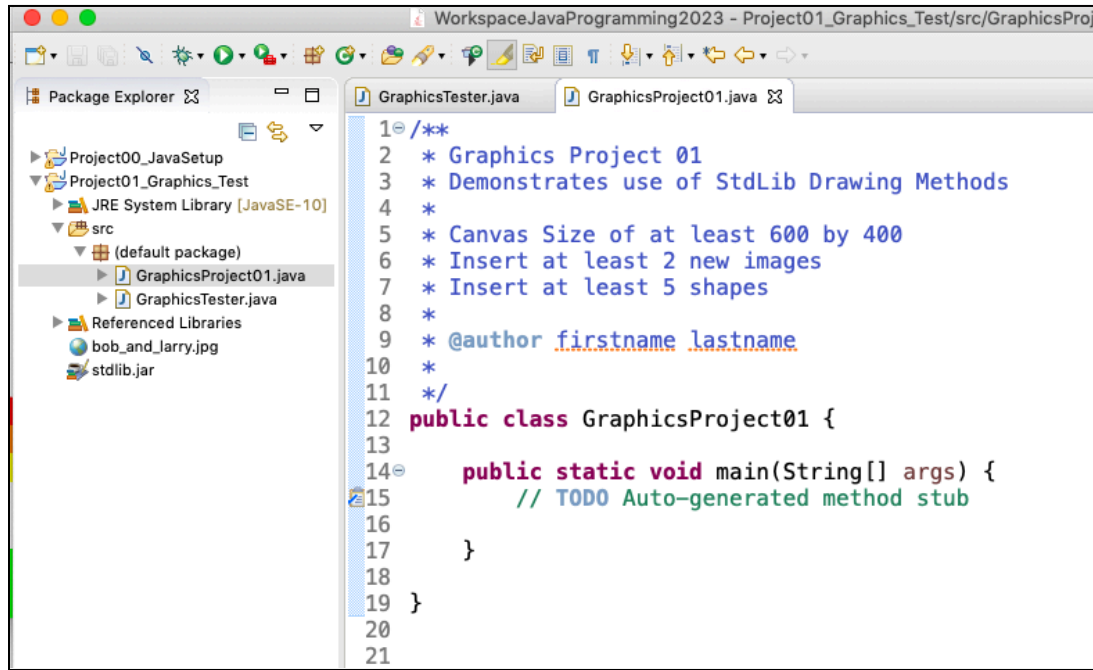
13. Some examples of shapes are shown below. Feel free to experiment!

```
26      // Examples with shapes:
27      StdDraw.setPenColor(0, 255, 0); // Green
28      StdDraw.filledRectangle(200, 375, 200, 25);
29
30      StdDraw.setPenColor(255, 0, 0); // Red
31      StdDraw.setPenRadius(0.04);
32      StdDraw.arc(200, 350, 100, 0, 180);
33
34      StdDraw.setPenColor(255, 125, 0); // Orange
35      StdDraw.setPenRadius(0.04);
36      StdDraw.arc(200, 350, 80, 0, 180);
37
38      StdDraw.setPenColor(255, 255, 0); // Yellow
39      StdDraw.setPenRadius(0.04);
40      StdDraw.arc(200, 350, 60, 0, 180);
41
```



## Requirements:

A. Create a new Java Class Called “GraphicsProject01”. Add JavaDoc comments as shown below:



B. Create a canvas size of at least 600 by 400 pixels with origin in upper left hand corner.

C. Find at least two new images and place the files into the Project01\_Graphics\_Test project. Place these into the Canvas

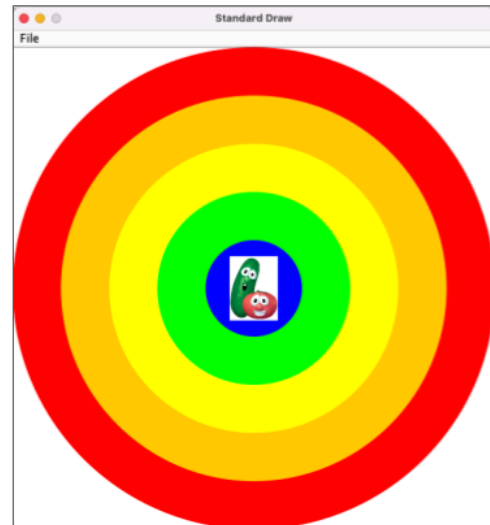
D. Place at least 5 shapes with at least 3 different colors and at least 3 different shape methods. Make sure to use comments for each shape to help the reader understand what code is trying to do. **Be creative!**

E. Submit following to Google Classroom:

- 1) Screenshot of the finished Canvas
- 2) GraphicsProject01.java source code file

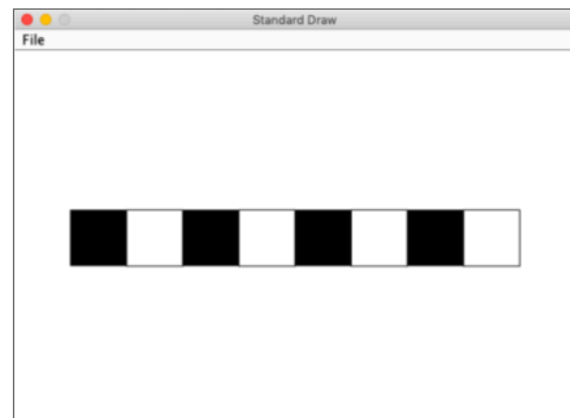
## 01 Drawing Challenge of the Day!

```
public class Challenge01 {  
    public static void main(String[] args) {  
        // Create Canvas  
        int width = 600;  
        int height = 600;  
  
        StdDraw.setCanvasSize(width, height);  
  
        StdDraw.setXscale(0, width);  
        StdDraw.setYscale(height, 0);  
  
        StdDraw.show();  
  
        // Start Drawing Code Here  
  
    }  
}
```



## 02 Drawing Challenge of the Day!

```
public class Challenge02 {  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        // Create Canvas  
        int width = 600;  
        int height = 400;  
  
        StdDraw.setCanvasSize(width, height);  
  
        StdDraw.setXscale(0, width);  
        StdDraw.setYscale(height, 0);  
  
        StdDraw.show();  
  
        // Start Drawing Code Here  
  
    }  
}
```



Hints: Starting Point = (90, 200)  
Squares are 60 x 60 Pixels

## 03 Drawing Challenge of the Day!

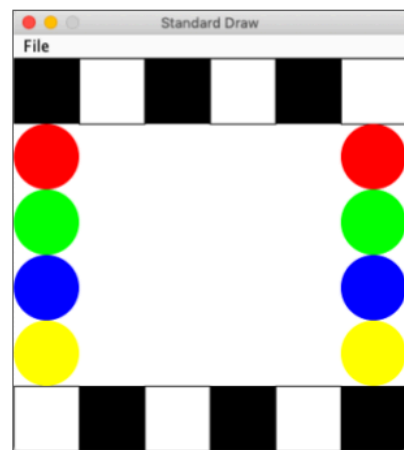
```
public class Challenge03 {  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        // Create Canvas  
        int width = 660;  
        int height = 400;  
  
        StdDraw.setCanvasSize(width, height);  
  
        StdDraw.setXscale(0, width);  
        StdDraw.setYscale(height, 0);  
  
        StdDraw.show();  
  
        // Start Drawing Code Here  
  
    }  
}
```



Hints: Starting Point = (40, 200)  
String path = "bob\_and\_larry.jpeg";  
StdDraw.picture(x, y, path, w, h, angle);

## 04 Drawing Challenge of the Day!

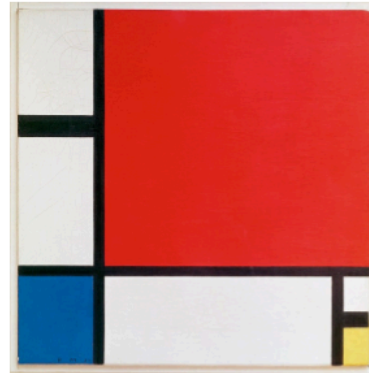
```
public class Challenge04 {  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
        // Create Canvas  
        int width = 360;  
        int height = 360;  
  
        StdDraw.setCanvasSize(width, height);  
  
        StdDraw.setXscale(0, width);  
        StdDraw.setYscale(height, 0);  
  
        StdDraw.show();  
  
        // Start Drawing Code Here  
  
    }  
}
```



Hints: Starting Point = (30,30)  
Shapes have radius of 30  
Can you use for loops?

## 05 Drawing Challenge of the Day!

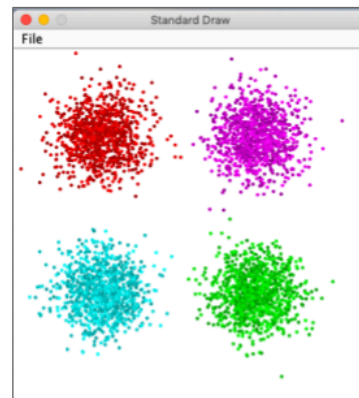
```
public class Challenge05 {  
    public static void main(String[] args) {  
        // Setup  
        int width = 400;  
        int height = 400;  
  
        StdDraw.setCanvasSize(width, height);  
  
        StdDraw.setXscale(0, width);  
        StdDraw.setYscale(height, 0);  
  
        // Show  
        StdDraw.show();  
    }  
}
```



Composition with Red, Blue and Yellow  
By Piet Mondrian 1930

## 06 Drawing Challenge of the Day!

```
import java.awt.Color;  
  
public class Challenge06 {  
    public static void main(String[] args) {  
        // Setup Canvas  
        int width = 400;  
        int height = 400;  
  
        StdDraw.setCanvasSize(width, height);  
  
        StdDraw.setXscale(0, width);  
        StdDraw.setYscale(height, 0);  
  
        StdDraw.show();  
    }  
}
```



Gaussian Random and Uniform Random

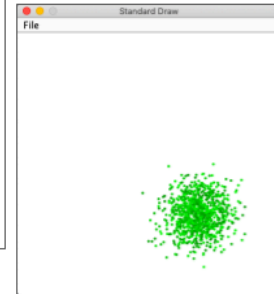
## Sample Code for Green Random Spots

```
// 1000 Random Green Spots
for (int i = 0; i < 1000; i++) {
    // Random Center of Spot at (275, 275) with deviation of 25
    double x = StdRandom.gaussian(275, 25);
    double y = StdRandom.gaussian(275, 25);

    // Random green color uniform between 150 and 255
    int g = StdRandom.uniformInt(150, 255);

    // Set Pen Color
    StdDraw.setPenColor(0, g, 0);

    // Draw the Random Circle with Radius of 2
    StdDraw.filledCircle(x, y, 2);
    StdDraw.show();
}
```



## StdRandom Methods (Pick a Random Number . . .)

StdRandom.gaussian(mu, sigma);	Chooses a random number with a "Gaussian" distribution. Values clumped towards the <i>mu</i> . Returns a double
StdRandom.gaussian(50, 5);	<i>mu</i> "Center of the value." Location for us sigma "Size of Distribution". Think as Size Looks like a Cluster of dots
StdRandom.uniformInt(min, max);	Chooses a random number between min and max. Returns an integer
StdRandom.uniformInt(100, 200);	Uniform distribution (like rolling a dice). Includes min and excludes max



**Computational Media**  
**Project 03: For Loops and Methods for Graphics**  
**Marist School**

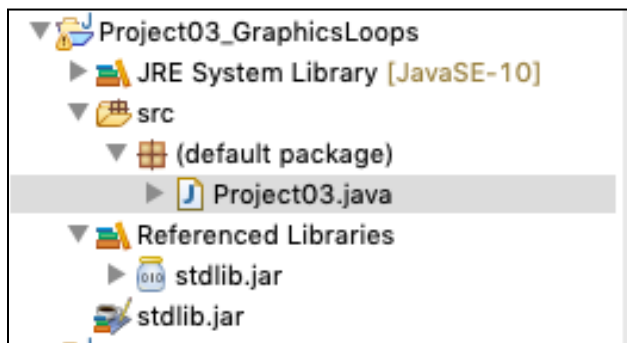
**Description:**

In this project you will implement three methods for creating graphics using for loops and nested for loops. The skills for this project include:

- Defining and calling void methods with parameters to create drawings.
- Using a nested for loop and two index variables to create two dimensional grids of graphics.
- Use counting variables, modulo, and conditionals to make decisions while creating the graphic.

**Project Setup:**

1. Create a new Java Project called “Project03\_GraphicsLoops” in Eclipse
2. Import the stdlib.java and Right-Click -> Add to Build Path.
3. Create a new Class in the “src” folder and call it “Project03.java”
4. The Project Setup should look like:



5. Add the JavaDoc Style comments for the Project03 class. They should look like the example below:

```
Project03.java
1 /**
2  *
3  * Project03: For Loops, Conditionals, Methods for Graphics
4  *
5  * @author Mr. Michaud
6  *
7  */
8 public class Project03 {
9
10     public static void main(String[] args) {
11         // TODO Auto-generated method stub
12
13     }
14
15 }
16
17
```

6. We will now add the Method Stubs for the project. Method Stubs provide the framework to develop code. We will also create the JavaDoc style comments for each method. These outline the structure of the code and then we will work within each method to write the functionality.

7. First, define the stub for the setup() method that will establish the drawing canvas:

```
14
15 /**
16  * Build the StdLib Canvas with inputs of
17  * width and height
18  * @param width
19  * @param height
20  */
21 public static void setup(int width, int height) {
22     // Set Canvas Size with StdLib
23
24     // Set Scale to put origin in upper left corner
25
26     // Show Canvas
27 }
28
```

8. Define the stub for the method for `makeCheckers()`. Be sure to include the JavaDoc comments as shown

```
28
29- /**
30     * Makes a Checkerboard with n by n squares
31     * with size being dimensions of squares in pixels
32     * @param n
33     * @param size
34     */
35- public static void makeCheckers(int n, int size) {
36     // Create Checkerboard
37
38 }
39
```

9. Define the stub for the method `makePyramid()`. Be sure to include the JavaDoc comments as shown.

```
39
40- /**
41     * Makes a pyramid with n number of rows.
42     * The top row has n squares. Then the next row has
43     * n-1 squares. This pattern continues until the last row
44     * has one square. The dimensions of the squares are determined
45     * by the parameter size
46     * @param n
47     * @param size
48     */
49- public static void makePyramid(int n, int size) {
50
51 }
52
```

10. Define the stub for the method `makeGradient()`. Be sure to include the JavaDoc comments as shown.

```
53-    /**
54     * Creates a centered Gradient image with bright in
55     * Center and moving towards dark on edges.
56     * Parameter n defines the number of steps in the gradient.
57     * You may choose the color range.
58     * @param n
59     */
60-    public static void makeGradient(int n) {
61
62    }
63
```

11. Define stub for creating a row of squares. We will work through this together in the directions for an example:

```
67
68-    /**
69     * Creates a row of n squares starting at x, y with size n
70     * This is a demonstration of using for loops
71     * to create drawing objects.
72     * @param x
73     * @param y
74     * @param n
75     * @param size
76     */
77-    public static void makeRowSquares(int x, int y, int n, int size) {
78        // Loop and Create Squares
79
80
81    }
82
```

## Work Through Directions for Row of Squares:

1. We will work through the steps to use a for loop to create a row of squares. The goal is to provide an example of how to iterate with a for loop and use math operators to set the location of each square.

The algorithm for creating the row of squares is as follows:

### Algorithm makeRowSquares: (Parameters $x$ , $y$ , $n$ , $size$ )

Loop  $n$  times:

$newX = i * size + size / 2 + x$  // Sets starting position for newX based on  $x$

Set Pen Color to Black

Draw Outline of Square at position  $newX$ ,  $y$ ,  $size / 2$

Set Pen Color to White

Draw Filled Square at position  $newX$ ,  $y$ ,  $size / 2$

First, find the makeRowSquares() method stub we defined above:

```
67
68= /**
69  * Creates a row of n squares starting at x, y with size n
70  * This is a demonstration of using for loops
71  * to create drawing objects.
72  * @param x
73  * @param y
74  * @param n
75  * @param size
76  */
77= public static void makeRowSquares(int x, int y, int n, int size) {
78     // Loop and Create Squares
79
80
81 }
82
```

2. Now create a for loop that will iterate from 0 to n exclusive as shown below

```
public static void makeRowSquares(int x, int y, int n, int size) {  
    // Loop and Create Squares  
  
    for (int i = 0; i < n; i++) {  
  
    }  
  
}
```

3. Inside the loop, we will calculate the new position for called newX based on the shift right in size and the starting position of x

```
public static void makeRowSquares(int x, int y, int n, int size) {  
    // Loop and Create Squares  
  
    for (int i = 0; i < n; i++) {  
  
        // Set Position for x (y stays the same)  
        int newX = i * size + size/2 + x;  
  
    }  
  
}
```

4. Now we will use the StdLib drawing commands to draw the boxes:

```
public static void makeRowSquares(int x, int y, int n, int size) {  
    // Loop and Create Squares  
  
    for (int i = 0; i < n; i++) {  
  
        // Set Position for x (y stays the same)  
        int newX = i * size + size/2 + x;  
  
        // draw (Black border and White Square)  
        StdDraw.setPenColor(0, 0, 0);  
        StdDraw.setPenRadius(0.005);  
        StdDraw.square(newX, y, size/2);  
        StdDraw.setPenColor(255, 255, 255);  
        StdDraw.filledSquare(newX, y, size/2);  
  
    }  
  
}
```

5. We also need to implement code for the setup() method. Find the setup() method and add code to set the canvas size: (Line 24 below)

```
15
16=  /**
17     * Build the StdLib Canvas with inputs of
18     * width and height
19     * @param width
20     * @param height
21     */
22= public static void setup(int width, int height) {
23     // Set Canvas Size with StdLib
24     StdDraw.setCanvasSize(width, height);
25
26     // Set Scale to put origin in upper left corner
27
28     // Show Canvas
29
30 }
```

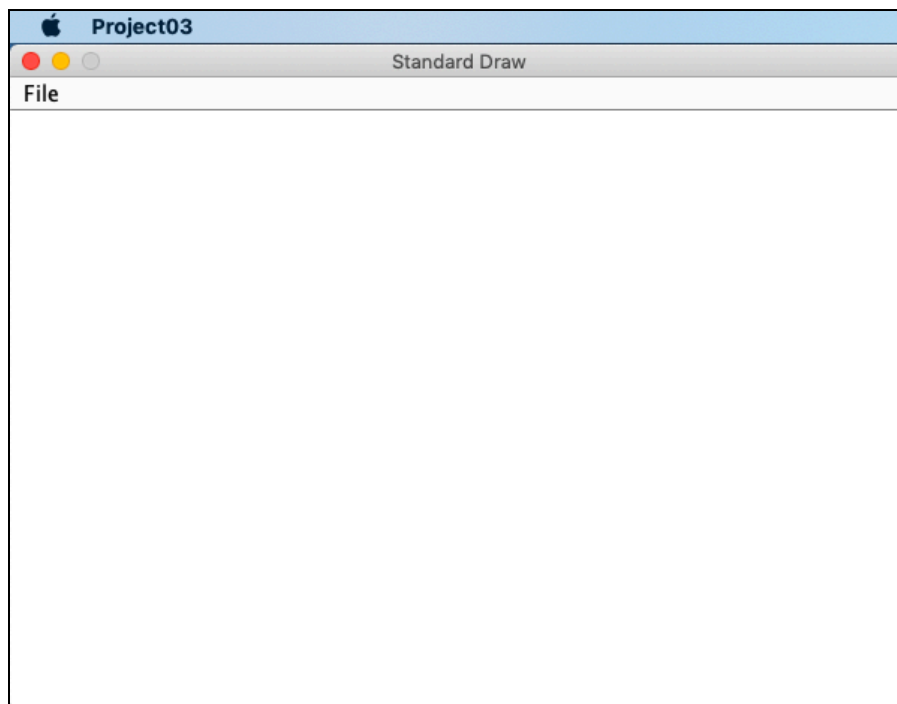
6. Now add code to set scale and show the canvas: (Lines 27-28 and Line 31)

```
15
16=  /**
17     * Build the StdLib Canvas with inputs of
18     * width and height
19     * @param width
20     * @param height
21     */
22= public static void setup(int width, int height) {
23     // Set Canvas Size with StdLib
24     StdDraw.setCanvasSize(width, height);
25
26     // Set Scale to put origin in upper left corner
27     StdDraw.setXscale(0, width);
28     StdDraw.setYscale(height, 0);
29
30     // Show Canvas
31     StdDraw.show();
32
33 }
34
```

7. We now can test the `setup()` method and the `makeRowSquares()` method. Find the `main()` method and call the `setup()` method to make a canvas that is 600 by 400 pixels:

```
1  /**
2   *
3   * Project03: For Loops, Conditionals, Methods for Graphics
4   *
5   * @author Mr. Michaud
6   *
7   */
8  public class Project03 {
9
10     public static void main(String[] args) {
11         // TODO Auto-generated method stub
12         setup(600, 400);
13     }
14 }
15
```

Run the method and you should see the canvas:

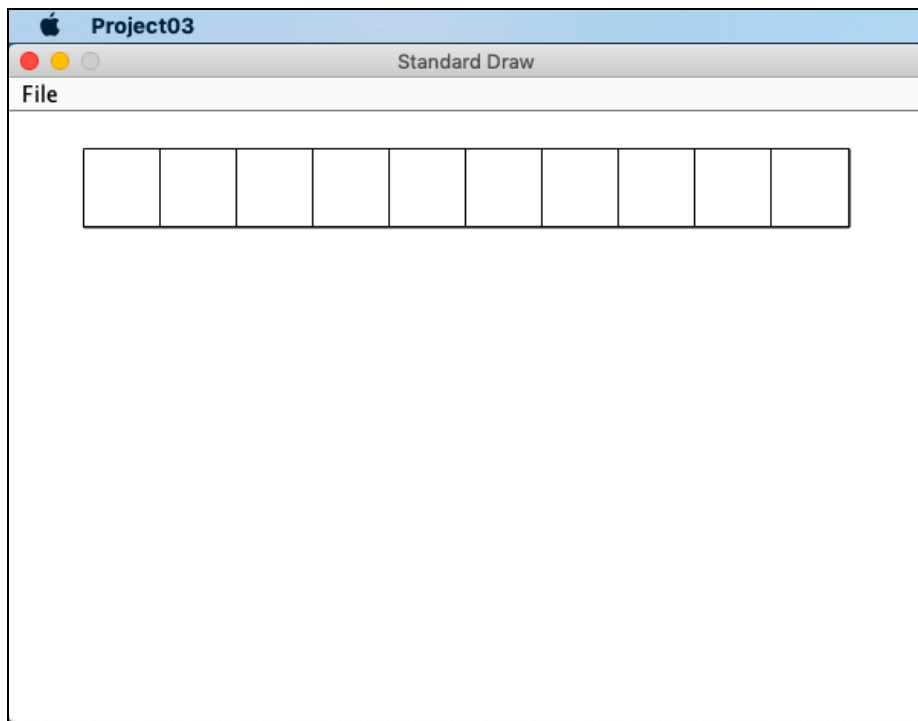




8. Now call the `makeRowSquares()` method. In the example below we create 10 squares that are 50 by 50 pixels in size starting at `x=50` and `y = 50`.

```
1 /**
2  *
3  * Project03: For Loops, Conditionals, Methods for Graphics
4  *
5  * @author Mr. Michaud
6  *
7  */
8 public class Project03 {
9
10     public static void main(String[] args) {
11         // TODO Auto-generated method stub
12         setup(600, 400);
13
14         makeRowSquares(50, 50, 10, 50);
15     }
16 }
```

9. Run the code and you should see:



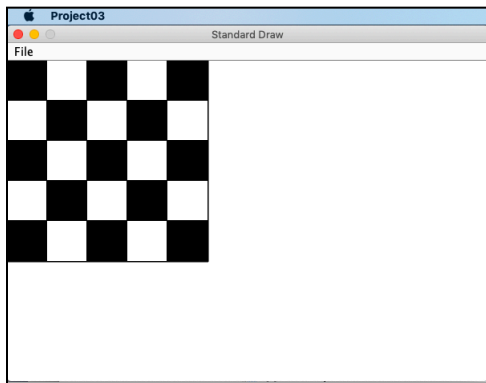
10. Congratulations - you have finished setting up and practicing with Java Project 03. The next section will outline the requirements.

### Requirements for Project 03:

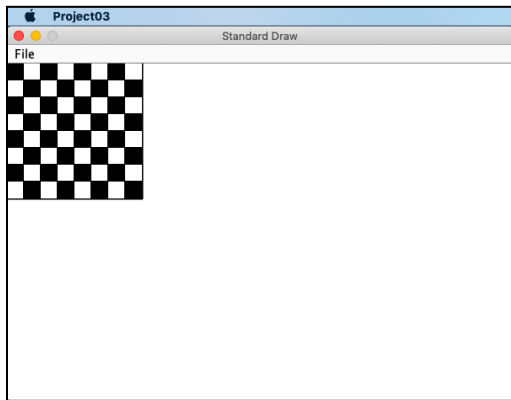
1. Implement the method `makeCheckers()` as shown below. The parameter  $n$  represents the  $n \times n$  dimensions of the checkerboard and the parameter `size` represents the size of each square.

```
28
29  /**
30   * Makes a Checkerboard with n by n squares
31   * with size being dimensions of squares in pixels
32   * @param n
33   * @param size
34   */
35  public static void makeCheckers(int n, int size) {
36      // Create Checkerboard
37  }
38
39
```

An example run of `makeCheckers(5, 50);` would produce:



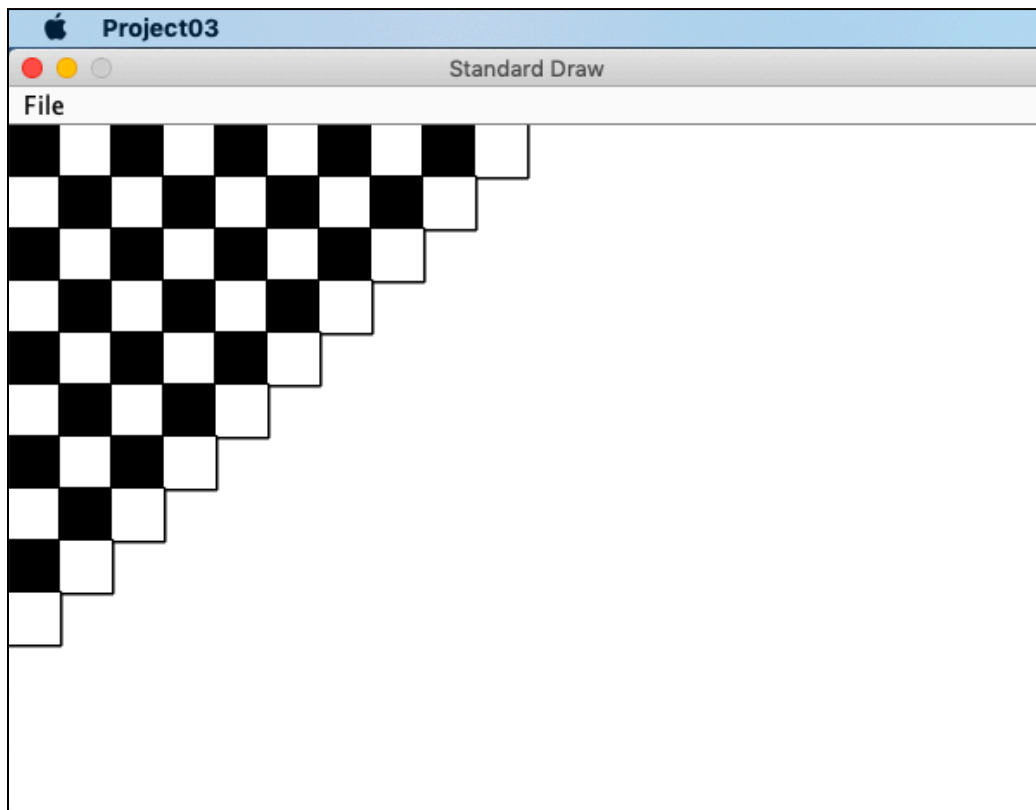
A run of `makeCheckers(8, 20);` would produce



2. Implement the method `makePyramid()` as shown below. The parameter `n` represents number of squares in the top row and then the pattern continues with `n-1` squares until there is 1 square left. The parameter `size` represents the size of each square.

```
39
40-  /**
41     * Makes a pyramid with n number of rows.
42     * The top row has n squares. Then the next row has
43     * n-1 squares. This pattern continues until the last row
44     * has one square. The dimensions of the squares are determined
45     * by the parameter size
46     * @param n
47     * @param size
48     */
49-  public static void makePyramid(int n, int size) {
50
51  }
52
```

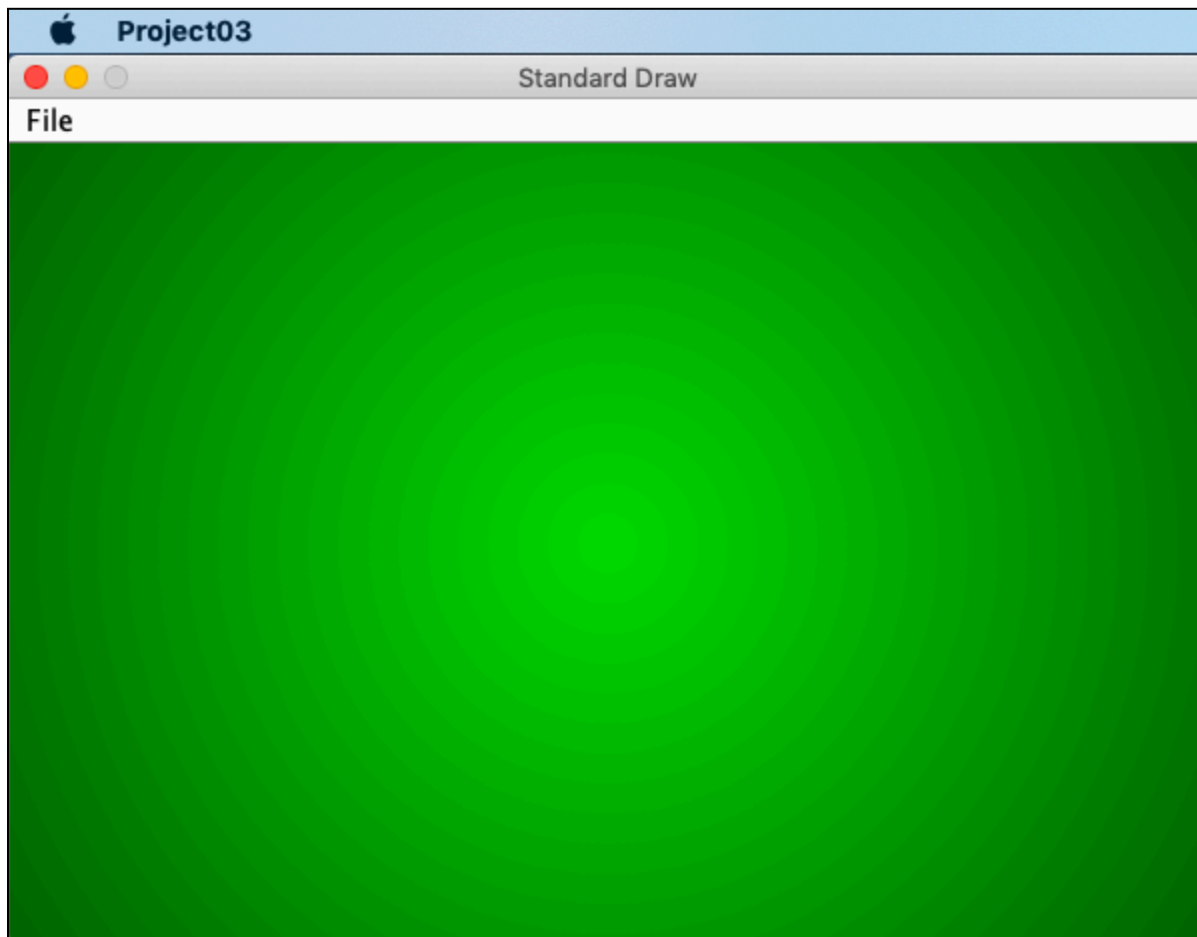
A run of: `makePyramid(10, 30)` would make:



3. Implement method `makeGradient(int n)` where the brightest part of the gradient is in the center and moves darker towards the edges. Think about drawing `n` circles starting with the large dark circle and moving smaller. You can choose the color range.

```
60
61=  /**
62    * Creates a centered Gradient image with bright in
63    * Center and moving towards dark on edges.
64    * Parameter n defines the number of steps in the gradient.
65    * You may choose the color range.
66    * @param n
67    */
68=  public static void makeGradient(int n) {
69
70  }
71
```

An example run of `makeGradient(200)` could look like:



### Hint for Checkerboard:

For the checkerboard a “nested for loop” control structure allows the programmer to define one loop inside another. An example in code is shown below:

```
/**
 * Demonstrates a Nested For Loop with n Rows and Columns
 * Output will be printing "* " representing a cell inside
 * A table with n Rows and n Columns
 * @param n
 */
public static void nestedLoopDemo(int n) {

    // Set number of ROWS and COLS
    int ROWS = n;
    int COLS = n;

    // Outer Loop for Rows
    for (int r = 0; r < ROWS; r++) {

        // Inner Loop for Cols
        for (int c = 0; c < COLS; c++) {
            // Do the Work
            System.out.print("* ");
        }
        // Add a new Line at end of Row
        System.out.println("");
    }

}
```

Notice that instead of `i` for the index variable, we are using `r` and `c`. The variable `r` will count the rows and `c` will count the columns. The output looks like this for running `nestedLoopDemo(6)`:

```
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
* * * * *
```

The example code below demonstrates a nested for loop printing the row and column numbers:

```
/**
 * Demonstrates a Nested For Loop with n Rows and Columns
 * Output will be printing "r0,c0 " representing a cell inside
 * A table with n Rows and n Columns
 * @param n
 */
public static void nestedLoopDemo2(int n) {

    // Set number of ROWS and COLS
    int ROWS = n;
    int COLS = n;

    // Outer Loop for Rows
    for (int r = 0; r < ROWS; r++) {

        // Inner Loop for Cols
        for (int c = 0; c < COLS; c++) {
            // Do the Work
            String cell = "r" + r + "c" + c + " ";

            System.out.print(cell);

        }
        // Add a 2 new Lines at end of Row
        System.out.println("");
        System.out.println("");
    }
}
```

The output looks like this for nestedLoopDemo2(6). Notice that the “y axis” is printed as the row and the “x axis” is the column.



```
Problems Javadoc Declaration Console
<terminated> Project03 [Java Application] /Library/Java/JavaV
r0c0 r0c1 r0c2 r0c3 r0c4 r0c5
r1c0 r1c1 r1c2 r1c3 r1c4 r1c5
r2c0 r2c1 r2c2 r2c3 r2c4 r2c5
r3c0 r3c1 r3c2 r3c3 r3c4 r3c5
r4c0 r4c1 r4c2 r4c3 r4c4 r4c5
r5c0 r5c1 r5c2 r5c3 r5c4 r5c5
```

Adapt this idea to make the checkerboard.

### Hint for Gradient:

1. Here is a hint for the gradient. Think about concentric circles with each circle getting lighter in color and smaller. Try the code below as a starting point: (Define method and then call in main)

```
/**
 * Sample Gradient
 * Generates 4 circles of graduating colors
 * Dark Green to light green
 */
public static void makeGradientSample() {

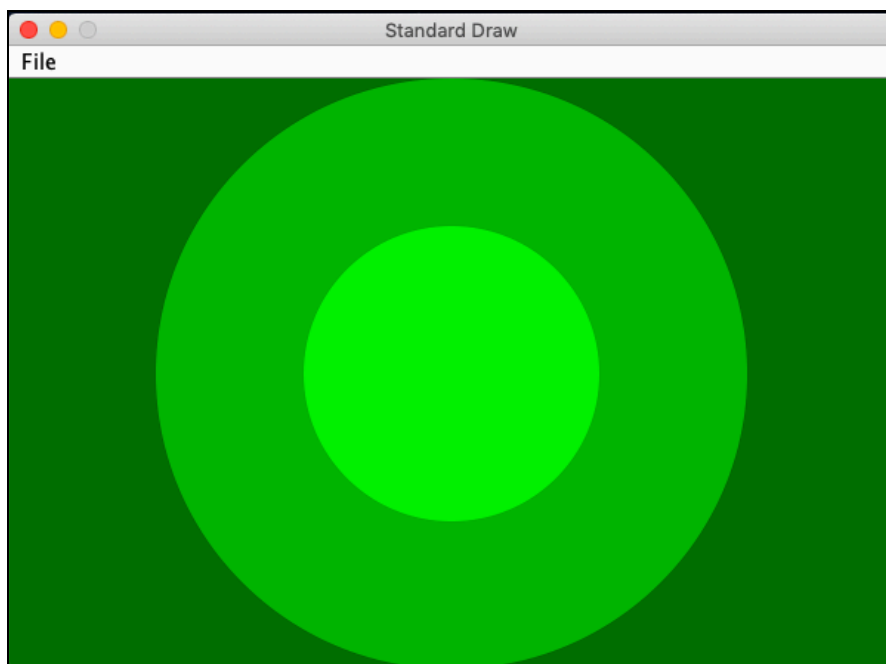
    // Outer Circle 1
    StdDraw.setPenColor(0, 50, 0);
    StdDraw.filledCircle(600/2, 400/2, 600);

    // Circle 2
    StdDraw.setPenColor(0, 110, 0);
    StdDraw.filledCircle(600/2, 400/2, 400);

    // Circle 3
    StdDraw.setPenColor(0, 180, 0);
    StdDraw.filledCircle(600/2, 400/2, 200);

    // Circle 4
    StdDraw.setPenColor(0, 240, 0);
    StdDraw.filledCircle(600/2, 400/2, 100);

}
```



**Deliverables:**

1. Create a Google Doc called “Lastname Project 03”.

2. Place a Heading on the Google Doc:

Java Project 02: For Loops and Methods for Graphics

Firstname Lastname

Programming in Java

Term 2, 2024

3. For each Method (makeCheckers, makePyramid, makeGradient) put the following:

- Name of Method
- Short description of the Method
- At least three screenshots of Method runs with different parameters
- Copy and paste the Code from your method (with the JavaDoc comments) into a table on the Google Doc.

Remember to use Times New Roman for prose and Consolas for code.

4. Submit the following to the Google Classroom:

- Google Doc “Lastname Project 03”
- Source Code file Project03.java



## Java programming for Image Manipulation

### Version 2024: Standard Library

#### Description:

The goal of these lessons is to apply the skills of nested iteration, math operations, conditionals, and manipulation of colors in pixels of images. A picture is a multidimensional collection of numerical data displayed in a way that humans interpret as a two dimensional color image.

The Standard Library from Princeton University (<https://introcs.cs.princeton.edu/java/stdlib/>) provides an API toolset for opening, editing, and saving images. We will use this library to explore and practice skills in traversing and editing arrays of data.

These series of lessons will work through various elemental techniques in image editing including:

- Opening, Editing, and Saving Image data with Picture Object
- Traversing 2D array of pixels and Editing values in color channels of Red, Green, Blue, and Alpha
- Rearranging pixel values for inverting and mirror effects
- Algorithms for creating grayscale and Duotone effects
- Combining images onto larger canvas such as tiling or collage
- Substitution algorithms used in greenscreen effects

The ultimate goal is to provide experience and practice skill in iterating through multidimensional data structures to provide the foundation for further study in computing algorithms such as path finding, graph theory, and machine learning.

#### Contents:

Lesson 01: Image Encoding  
Lesson 02: Setup Eclipse IDE and ImageManipulation project  
Lesson 03: Quickstart: Opening Images  
Lesson 04: Java Picture Functions  
Lesson 06: Visiting Pixels: writing the setRed() method  
Lesson 07: Assignment: setGreen() and setBlue()  
Lesson 08: Writing makeGrayscale() method  
Lesson 09: Saving Images to Files  
Lesson 00: Implement makeNegative() method  
Lesson 11: Reversing Images on X and Y Axis  
Lesson 12: Mirroring images on X, Y, and Diagonal Axis  
Lesson 13: Averaging Two Images  
Lesson 15: Combing Images onto a Canvas  
Lesson 16: Final Project Description

**Additional Content:**

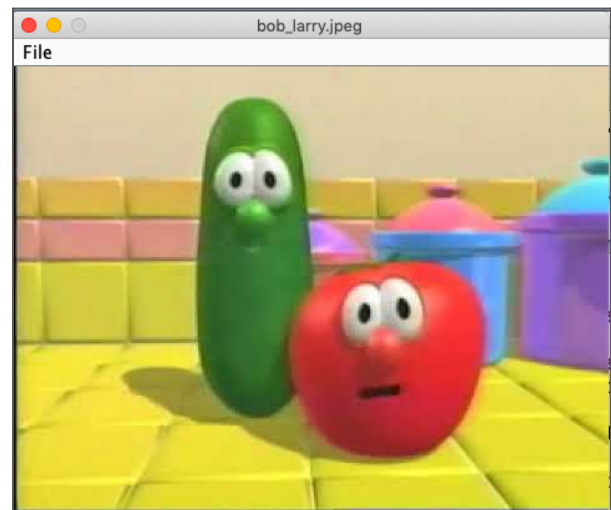
Tiling Project from Princeton COS126

Duotone Project from Princeton COS126

## Lesson 01: Image Encoding in Java

### Key Concepts:

- Format of Images in Computing
- Java Structure for Modeling Images
- Java Picture Object methods



### Format of Color Images in Java:

In computing, an image is stored as a two dimensional (2D) collection of pixels. By convention, the Y Axis are called the Rows and the X Axis are called columns. The origin (0, 0) is located in the upper left corner.

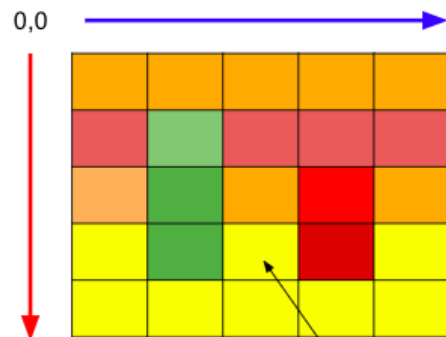
2 Dimensional Array of Pixels

Y Axis is called Rows

X Axis is called Columns

Origin in Upper Left Hand Corner

Model: A grid of colored squares



This pixel is  
at Row 3  
Column 2  
(2, 3)

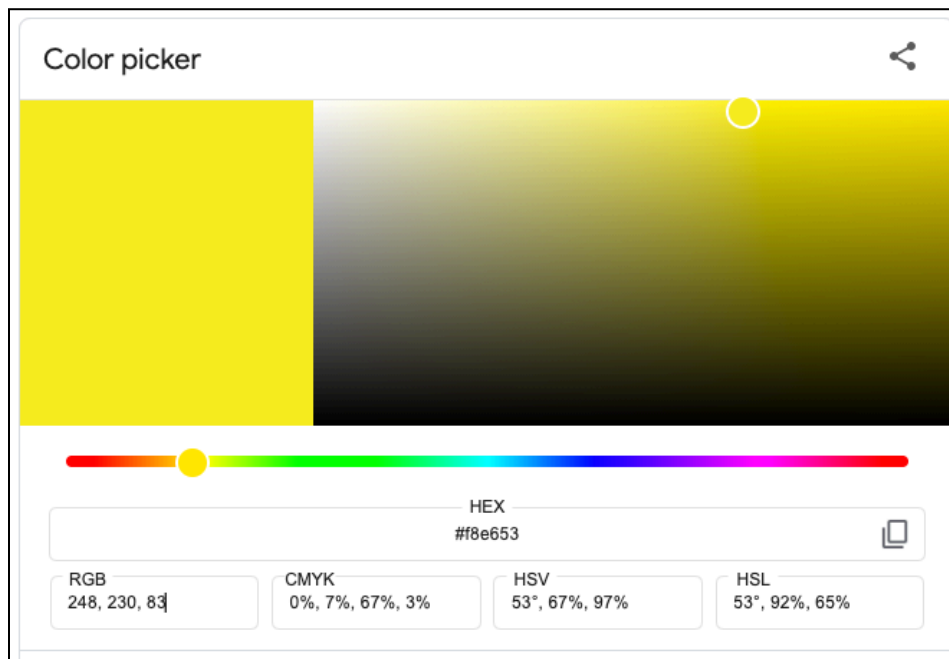
## Java structure for Pixels in Images:

A Picture Object is a grid of 2D Pixels. A Pixel contains four numerical values:

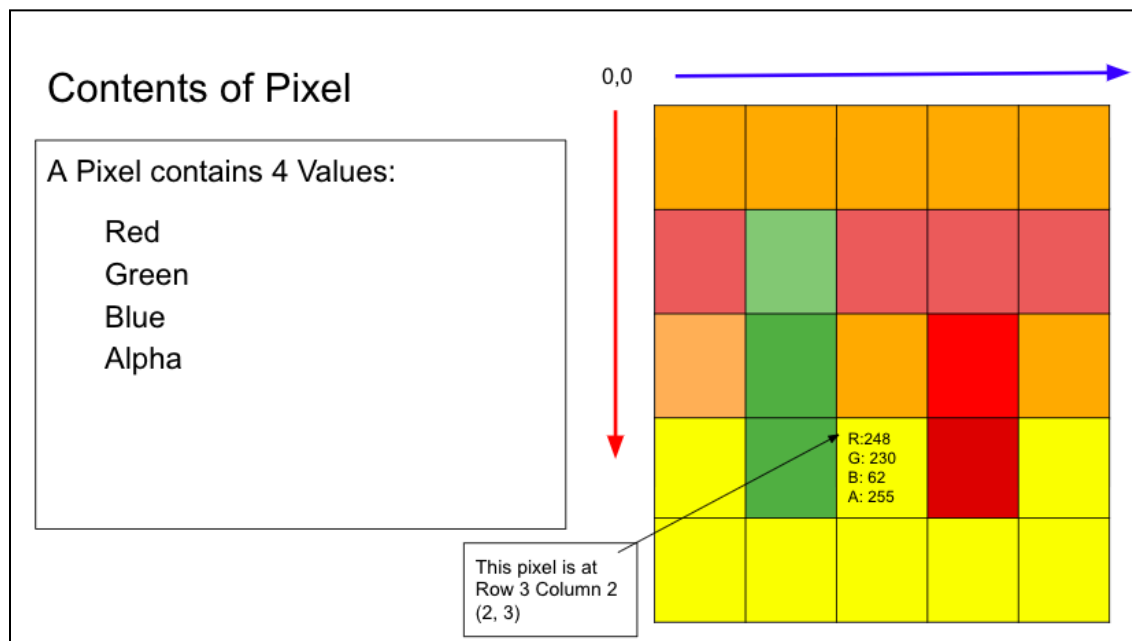
- Red
- Green
- Blue
- Alpha

Each value contains a number between 0 and 255. Think of a color channel as a light with a value of 0 being the light is off, and 255 the light is full strength. The Alpha channel controls the “transparency” of the pixel. An Alpha of 255 means the pixel is opaque and an Alpha of 0 means the pixel is transparent. Image formats such as .png, and .gif have “Alpha” channels. “.jpg” images do not have an Alpha Channel. For most of these lessons, we will work only with Red, Green, and Blue values.

An HTML color picker works in the same manner.



When referring to a specific Pixel in a Picture object, the returned object in Java is a Color Object.



### Key Commands in stdlib for Java Picture Object:

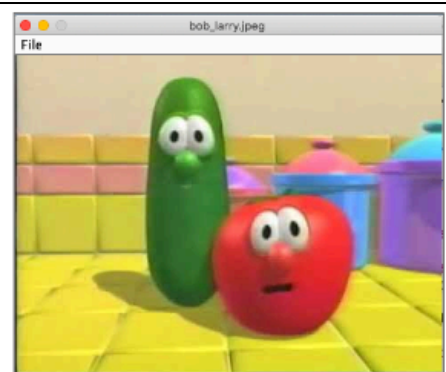
```
// Load an image into myPicture object
Picture myPicture = new Picture("bob_larry.jpeg");

// Get a Color at Location x, y
Color c = myPicture.get(54, 305);

// Get red value from Pixel p
int redValue = c.getRed();

// Create new Color Object
Color newC = new Color(123, 10, 55);

// Set pixel to new Color
myPicture.set(54, 305, newC);
```



## Lesson 02: Setup Eclipse IDE and ImageManipulation project

### Key Terms:

**Eclipse is an IDE** (Integrated Development Environment) used for writing and running code.

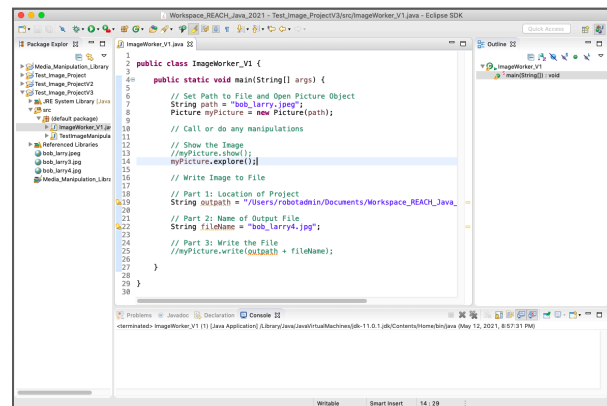
A **Jar File** is a Java Archive file that encapsulates a Library of functionality.

We will use the Jar file: **“stdlib.jar”**

**.jar** means “Standard Library” and it is the library for Java used by Princeton University in their programming classes.

Most universities and businesses will have their own custom libraries for Java and other languages to standardize and provide functionality for research and industry work.

MIT has the “Processing” IDE and Georgia Tech has used the Media Computation .jar file. These libraries change and evolve throughout the years.



### Eclipse IDE Overview

**Coding Area:** Editing current file

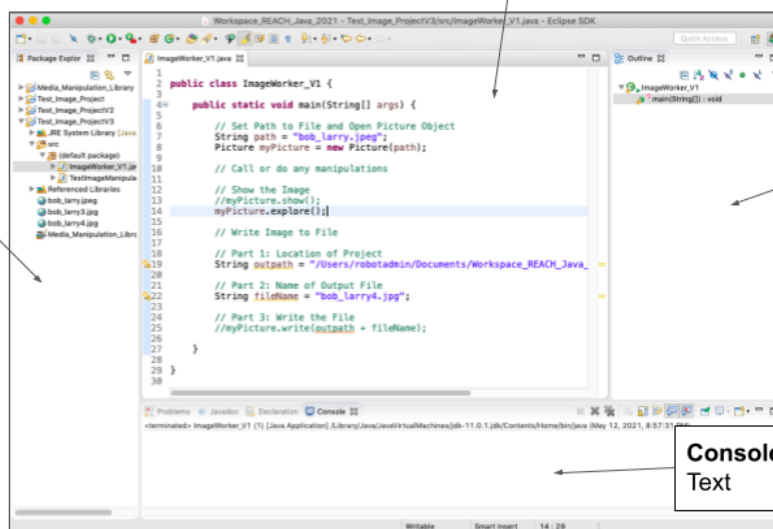
**Package Explorer:**

Current Projects in Workspace

**Outline:**

Structure of current Java Class

**Console:** Output Area for Text



## Workspace

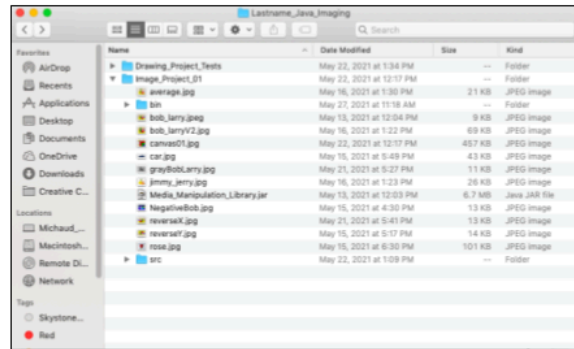
Folder on your computer that stores all the files associated with a Java Project

For Java Imaging, files will include the

.java source code

stdlib.jar

Any images we will edit



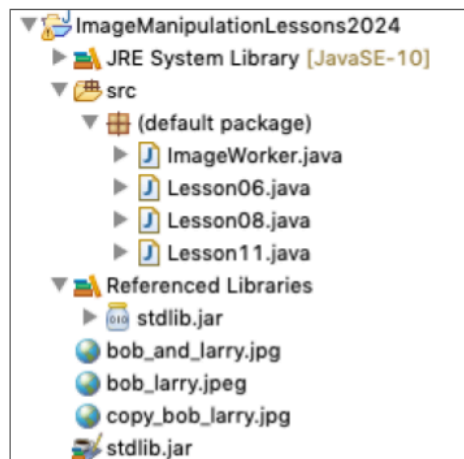
## Java Project

Collection of Libraries, Jar Files, Media Files, and your code that works together to create the project and program.

You will create Java files in the “src” folder under the “(default package)”

Images files are copied and placed in the project folder for editing.

New Images from program are saved here.

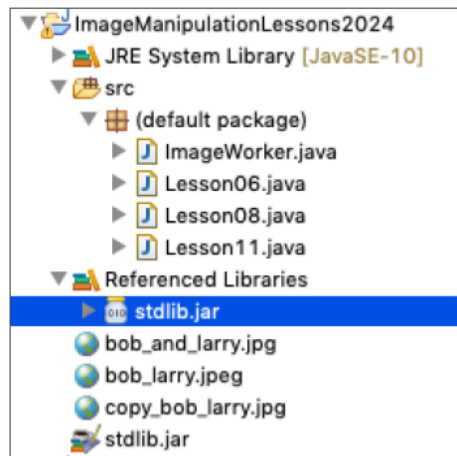


## Jar File: stdlib.jar

This library holds all code that allows us to use Java to manipulate and process Images and Sounds

Written at Princeton University

We will import and add this .jar file to the build path to allow our programs to work.





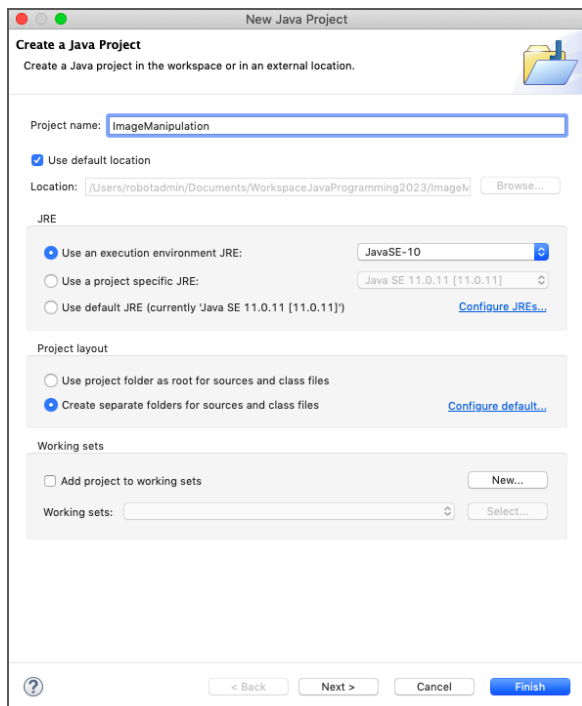
## Setup Eclipse, Make New Project, Import Library

### Overview: (Details follow)

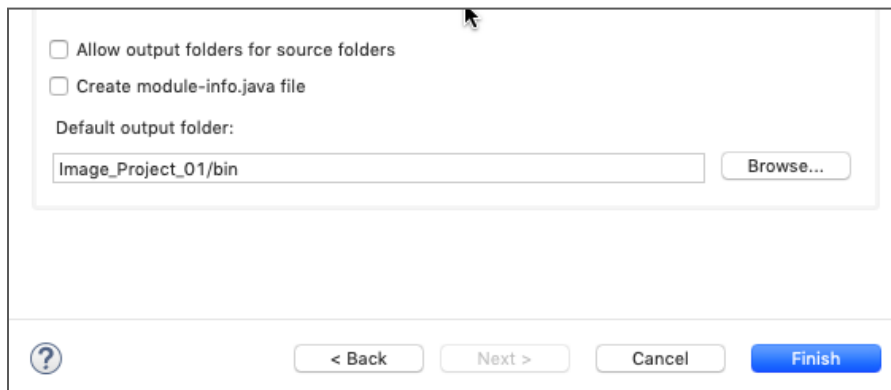
1. Create new Project “ImageManipulation”
2. Download the stdlib.jar file
3. Copy stdlib.jar file into project and add to build path

### Process:

1. Start Eclipse
2. Close the Welcome Tab if needed.
3. Select “File -> New -> Java Project” from the menu bar
4. Name the project “ImageManipulation” and click “Next”



5. Make sure the “Create module-info.java file” is unchecked:



6. Click “Finish”

7. Click on the links below to download the “stdlib.jar” file

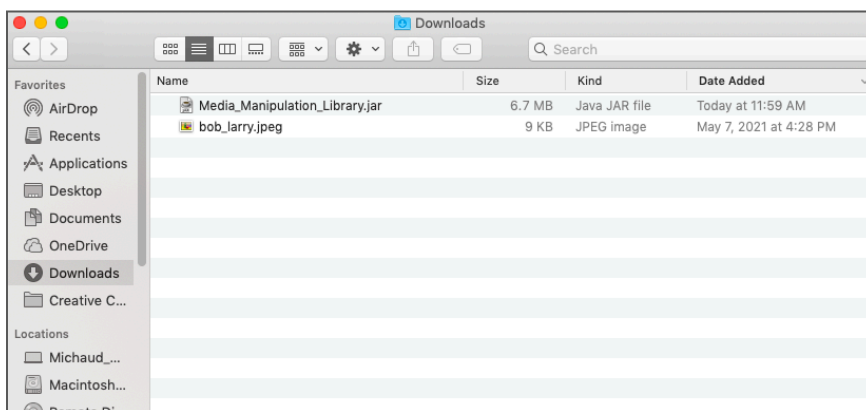
<https://nebomusic.net/javalessons/stdlib.jar>

8. Click “Allow” in your browser if needed.

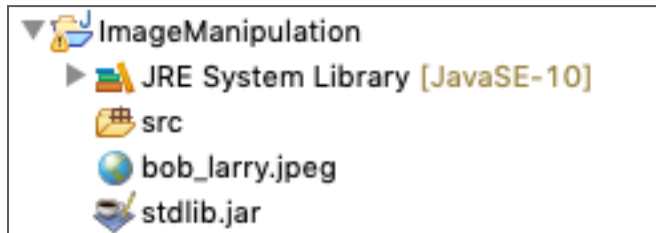
9. Click on the link below to download the “Bob and Larry” sample image (or find an image of your own. Make sure it is smaller than 800 by 800 pixels.

[https://nebomusic.net/javalessons/ImageManipulation/bob\\_larry.jpeg](https://nebomusic.net/javalessons/ImageManipulation/bob_larry.jpeg)

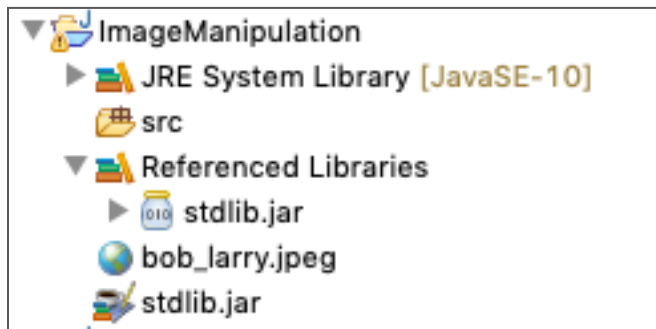
10. Find the .jar and image file in your downloads folder:



11. Select the stdlib.jar file and select “copy” (Control - c)
12. Go to your project in Eclipse.
13. Click on Project and select “Paste” to put the stlib.jar file into the project
14. Go back to your downloads folder and copy the sample image. Then paste the image into your project in Eclipse. The structure of the project should look like this:



15. We will now add the .jar file to the build path. Left click once on the stdlib.jar file.
16. Right click on the stdlib.jar file and select “Build Path - Add to Build Path”
17. The project structure will change and you will see “Referenced Libraries” and the .jar file.



18. Congratulations! You are ready to go and start the next Lesson!

## Lesson 03: Quickstart: Opening Images

### Overview of Steps

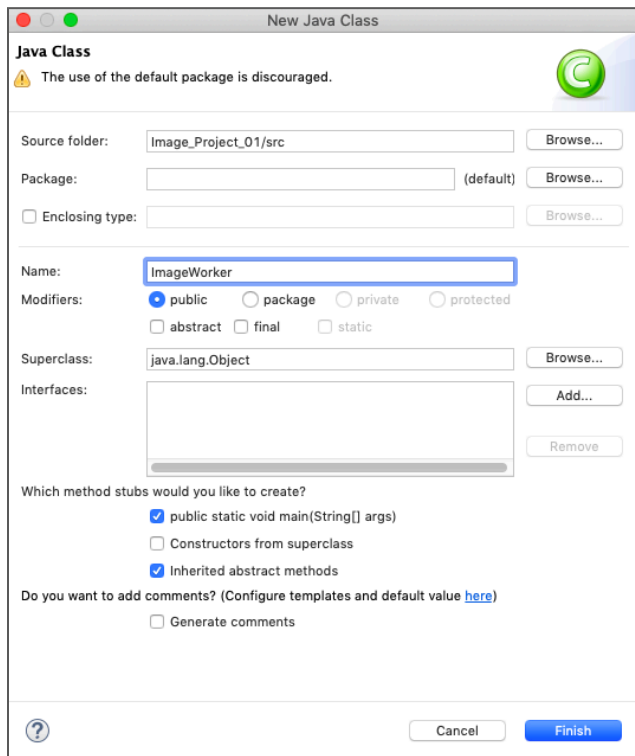
1. Create new Java File called "ImageWorker.java"
2. Define String for image path
3. Create myPicture object from the path
4. Write code to show the myPicture
5. Write code to explore myPicture

### Process: Create a New Java File called "ImageWorker.java"

1. Open Eclipse and your "ImageManipulation" project.
2. Find the "src" folder in the Project structure.
3. Right click on the "src" folder and select "New -> Class"

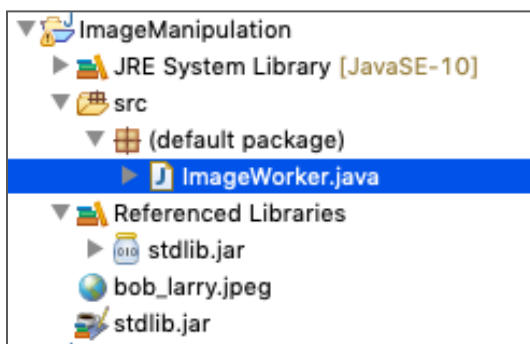
4. Name the Class “ImageWorker.java”

5. Make sure the box is checked “public static void main(String [] args)”



5. Click “Finish”

6. The new .java file should appear in the “src” folder

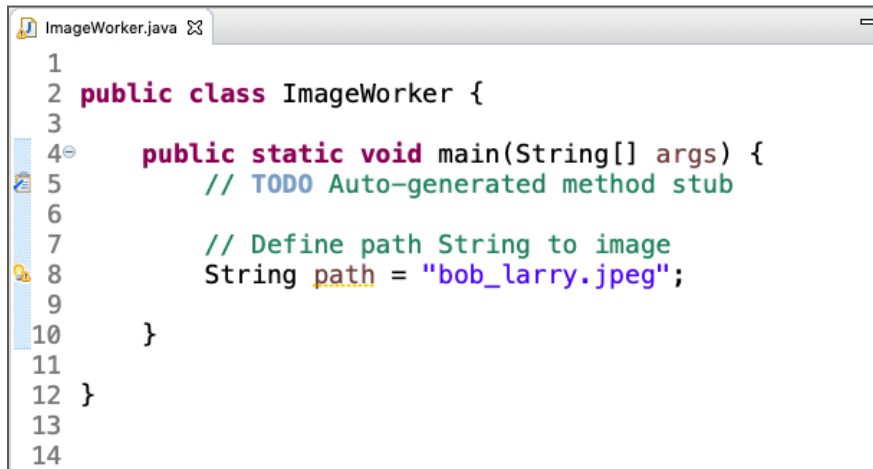


### Define String for Image Path:

7. Double click on the “ImageWorker” file.

8. This Java class format uses the public void main(). For now we will write code “inside” of public void main. Think of this like the “run” function in the previous code we have written.

9. Define a String to point to the location of the sample image “bob\_larry.jpeg”  
(Lines 7 and 8 in the sample)

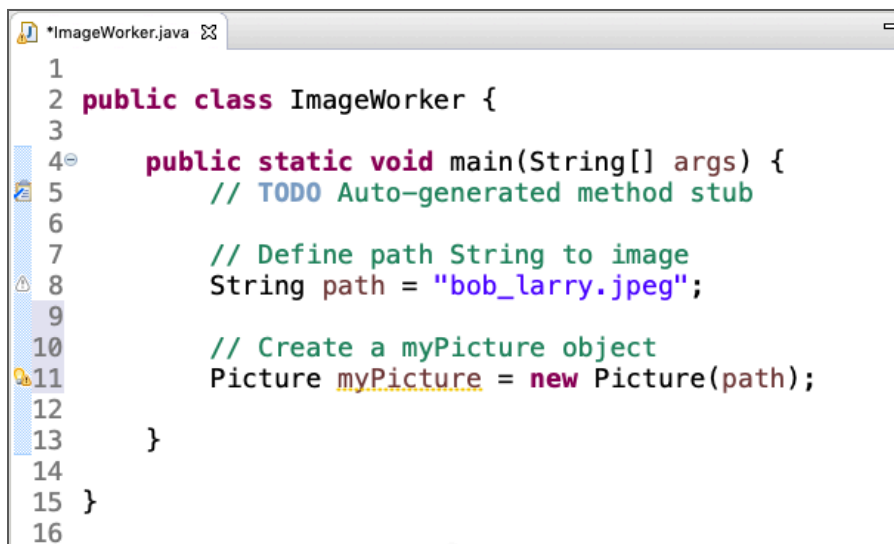


```
1
2 public class ImageWorker {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7         // Define path String to image
8         String path = "bob_larry.jpeg";
9
10    }
11
12 }
13
14
```

### Create a myPicture instance using the path

10. We will now create a Picture object “myPicture” that will hold the 2 dimensional array of pixel objects.

11. Lines 10 and 11 show how to create a Picture object “myPicture” using the path String.



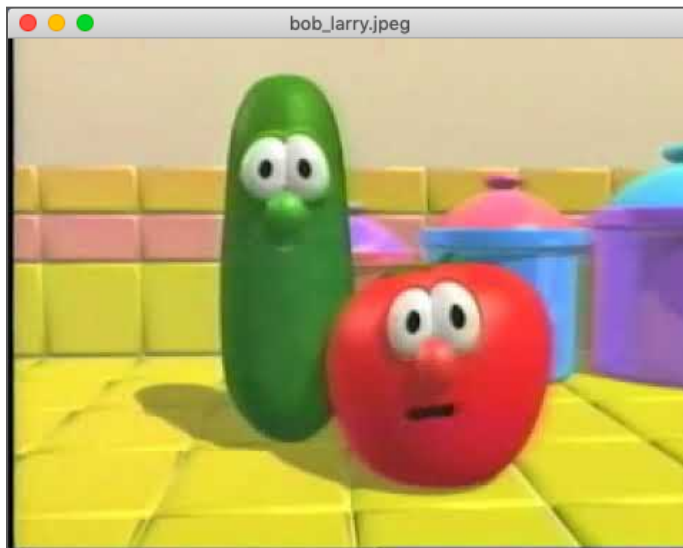
```
1
2 public class ImageWorker {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7         // Define path String to image
8         String path = "bob_larry.jpeg";
9
10        // Create a myPicture object
11        Picture myPicture = new Picture(path);
12
13    }
14
15 }
16
```

## Write code to show “myPicture”

12. Use the `.show()` method to display the picture. Lines 13 and 14 show the implementation.

```
ImageWorker.java
1
2 public class ImageWorker {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7         // Define path String to image
8         String path = "bob_larry.jpeg";
9
10        // Create a myPicture object
11        Picture myPicture = new Picture(path);
12
13        // Display myPicture
14        myPicture.show();
15    }
16
17 }
18 }
19
```

13. Save and run the code. The image should appear.

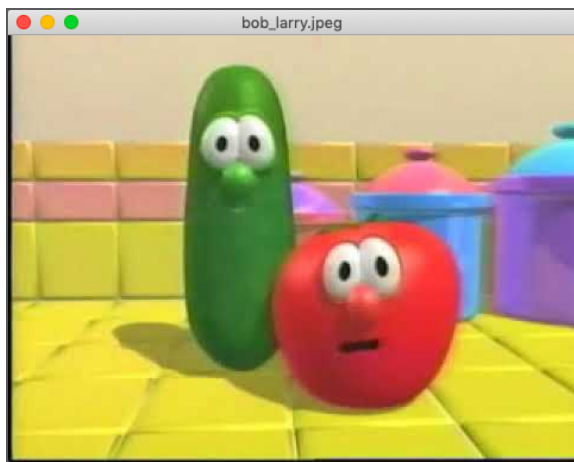


## Congratulations and Short Assignment:

You have successfully tested the Java Image code. Here is a short assignment:

1. Find a picture (use a small picture, less than 512 by 512 pixels).
2. Copy the picture into the Project folder.
3. Create a new String path2 to point to your picture file.
4. Create a new Picture Object and open the Picture using the .show() function.

```
ImageWorker.java
1
2 public class ImageWorker {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7         // Define path String to image
8         String path = "bob_larry.jpeg";
9
10        // Create myPicture object
11        Picture myPicture = new Picture(path);
12
13        // Display myPicture
14        myPicture.show();
15    }
16 }
17
18 }
19
```





Lesson 04: Java Picture Functions

04 Java Picture Functions

Types of Functions for Picture Object

- 1. Constructors
- 2. Accessor Functions
- 3. Viewing Functions
- 4. File Writing Functions

Picture.class

Picture

main(String[]) : void

height

image

isDisposed

isOriginUpperLeft

isVisible

jframe

title

width

Picture(int, int)

Picture(File)

Picture(String)

Picture(Picture)

actionPerformed(ActionEvent) : void

createGUI() : JFrame

equals(Object) : boolean

get(int, int) : Color

getJLabel() : JLabel

getRGB(int, int) : int

hasAlpha() : boolean

hashCode() : int

height() : int

hide() : void

isVisible() : boolean

save(File) : void

save(String) : void

set(int, int, Color) : void

setOriginLowerLeft() : void

setOriginUpperLeft() : void

setRGB(int, int, int) : void

setTitle(String) : void

show() : void

toString() : String

validateColumnIndex(int) : void

validateRowIndex(int) : void

width() : int

1. Constructors: Build a Picture object and return to program

String path = "bob\_larry.jpeg";  
Picture myPic = new Picture(path);

Picture

Picture()

Picture(String)

Picture(int, int)

Picture(Picture)

Picture(BufferedImage)

Type	Example	Description
Picture()	Picture myPic = new Picture();	Creates a picture object mPic with white pixels that is 200 pixels wide by 100 pixels tall.
Picture(String)	Picture myPic = new Picture(path);	Creates a picture object myPic out of an image file located at path.
Picture(int, int)	Picture myPic = new Picture(600, 400);	Creates a picture object myPic with white pixels that is 600 pixels wide by 400 pixels tall.
Picture(Picture)	Picture myPicA = new Picture(path); Picture myPicB = new Picture(myPicA);	Creates a picture object myPicB that is a copy and separate object from myPicA.

## 2. Accessor Functions: Gets information from Picture

```
String path = "bob_larry.jpeg";  
Picture myPic = new Picture(path);
```

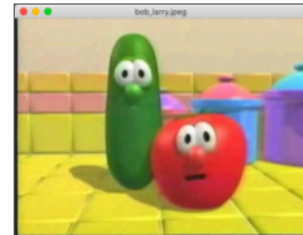
Accessor functions usually start with the word "get"

Type	Example	Description
height()	<code>int height = myPic.height();</code>	Returns an integer representing the height in pixels of the picture.
width()	<code>int width = myPic.width();</code>	Returns an integer representing the width in pixels of the picture.
get(int x, int y)	<code>Color c = myPic.get(100, 100);</code>	Returns a Color object

## 3. Viewing Functions: Displaying Picture to Screen

```
String path = "bob_larry.jpeg";  
Picture myPic = new Picture(path);  
myPic.show();
```

Type	Example	Description
show()	<code>myPic.show();</code>	Displays the Picture myPic to the computer screen in a Java Applet.



## 4. File Writing Functions: Saving data to image files

```
String path = "bob_larry.jpeg";  
Picture myPic = new Picture(path);  
String newPath = "newPicture.jpg";  
myPic.save(newPath);
```

Type	Example	Description
save(String)	<pre>String newPath = "newPicture.jpg";  myPic.save(newPath);</pre>	<p>Write the contents of a Picture object to the file system as a .jpg file. Remember to use the file extension ".jpg".</p> <p>In some computer systems you may have to specify a specific path back to the Project in the Workspace.</p>

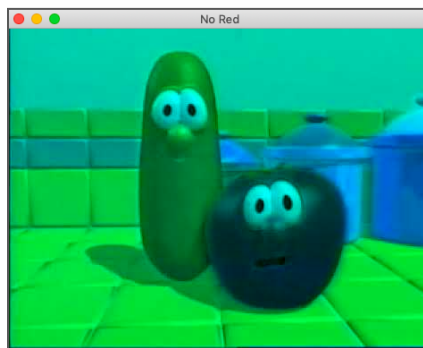
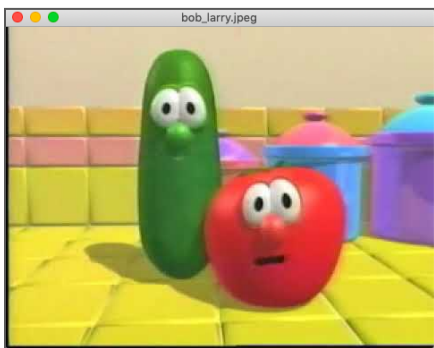
## Lesson 06: Visiting Pixels: writing the setRed() method

**Objective:** Write a function to visit all the Pixels in a Picture object and set the Pixels to the same red value.

### Skills Needed:

- Open a Picture Object
- Use the get(x, y) and Color Objects
- Using For Loops in a “nested” style
- Using the .show() method

### Work-through Lesson - Step by Step Instructions

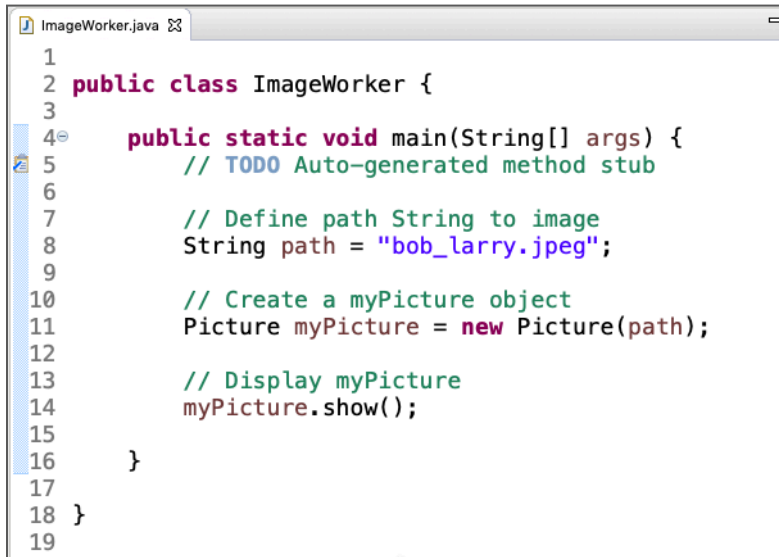


### Overview of Process:

1. Open or write the ImageWorker.java to open and display the sample picture
2. Define the function  
`public static Picture setAllRed(Picture p, int red)`
3. Create a copy of the picture
4. Use nested for loops to change all red pixel values of the copy.
5. Create a new class called “Lesson\_06”
6. Test the function and show in the void main() function of “Lesson\_06”

## Open or write the ImageWorker.java

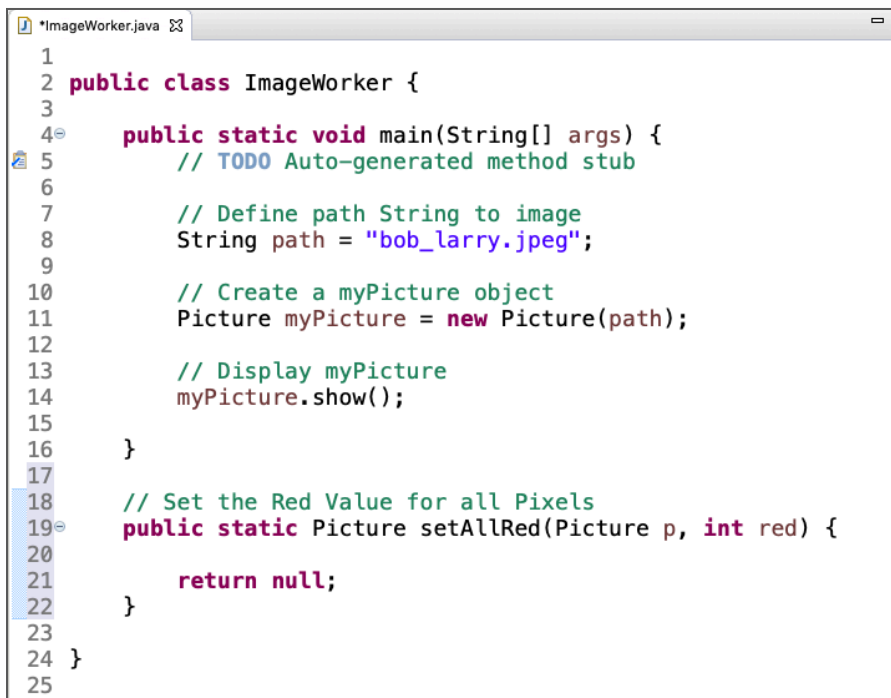
1. Open Eclipse and find the Project you created from 03 Quickstart Opening Exploring and Viewing Images. If you do not have this code, use the directions from 03 to set up the project.
2. The starting code is shown below:



```
1
2 public class ImageWorker {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7         // Define path String to image
8         String path = "bob_larry.jpeg";
9
10        // Create a myPicture object
11        Picture myPicture = new Picture(path);
12
13        // Display myPicture
14        myPicture.show();
15    }
16 }
17
18 }
19
```

### Define function setAllRed(Picture p, int red)

3. Start below the curly bracket the closes the main() function.
4. Write the function definition for setAllRed(Picture p, int red)  
(Lines 18 through 22 in example)
5. Notice that we have a 'return null' is written on line 21. This is a placeholder for a needed return statement.



```
1
2 public class ImageWorker {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7         // Define path String to image
8         String path = "bob_larry.jpeg";
9
10        // Create a myPicture object
11        Picture myPicture = new Picture(path);
12
13        // Display myPicture
14        myPicture.show();
15    }
16
17    // Set the Red Value for all Pixels
18    public static Picture setAllRed(Picture p, int red) {
19
20        return null;
21    }
22
23
24 }
25
```

### Create a copy of the parameter Picture p

6. Inside the function, create a Picture object named “output” to hold a copy of the incoming picture.

7. Change the return statement to return output.

```
// Set the Red Value for all Pixels
public static Picture setAllRed(Picture p, int red) {

    // create copy named output
    Picture output = new Picture(p);

    // Return the output
    return output;
}
```

Note the process:

- a. Define output
- b. Do the work
- c. Return the output

### Setup nested for loops to visit each pixel value

We are going to visit each pixel and change the red values. This is the “Do the work” part of the process

8. Write a for loop to move through all the y values

9. Write a for loop inside the y loop for the x values.

10. Notice the placement of the curly brackets to “nest” the Loop. We will go one row at a time and visit each value in the row. The next page will outline how to change the value.

```
// Set the Red Value for all Pixels
public static Picture setAllRed(Picture p, int red) {

    // create copy named output
    Picture output = new Picture(p);

    // Visit all the Pixels and change red value
    for (int y = 0; y < output.getHeight(); y++) {

        for (int x = 0; x < output.getWidth(); x++) {

        }

    }

    // Return the output
    return output;
}
```



## Get pixel at (x, y) and change red value

Now we will get the pixel at x, y from the output copy and change the red value.

11. Use the .get() function to get the Color at (x, y) from the loop
12. Create a new Color substituting input red and existing getGreen() and getBlue()
13. Set the Color value of the output image at pixel x, y with new Color.

```
// Set the Red Value for all Pixels
public static Picture setAllRed(Picture p, int red) {

    // create copy named output
    Picture output = new Picture(p);

    // Visit all the Pixels and change red value
    for (int y = 0; y < output.height(); y++) {

        for (int x = 0; x < output.width(); x++) {
            // Get Color at Pixel x, y
            Color c = output.get(x, y);

            // Create new Color with red value
            Color newC = new Color(red, c.getGreen(), c.getBlue());

            // Set the new Pixel color at x, y
            output.set(x, y, newC);
        }
    }

    // Return the output
    return output;
}
```

## Create a new class called “Lesson\_06”

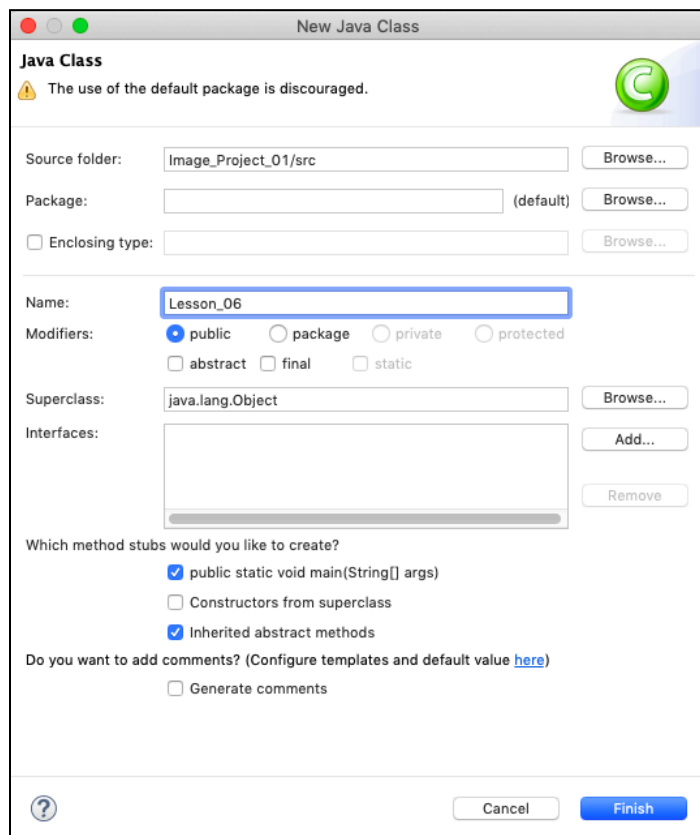
We are going to start making separate classes for each lesson to test the functions.

14. Right click on the “src” folder and select “New-Class”

15. Name the class “Lesson\_06”

16. Make sure the box “public static void main(String [] args) is checked.

17. Click “Finish”



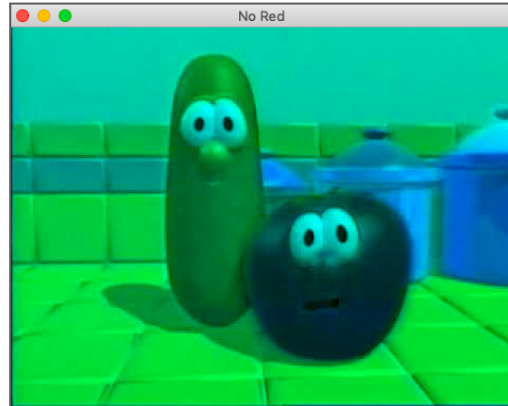
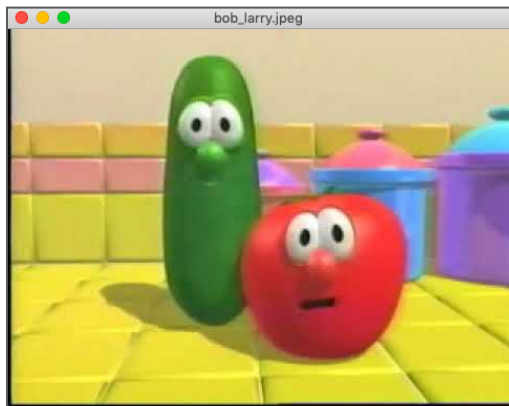
18. You should now be in the new class “Lesson\_06”

## Test the setAllRed() in the main function of Lesson\_06

19. Go back to the main function
20. Use the ImageWorker.setAllRed() function to create a new Picture object named "zeroRed" with red values set to 0. (Experiment with different values)
21. Set the Title of zeroRed to "No Red"
22. Use the .show() function to display the original and changed picture.

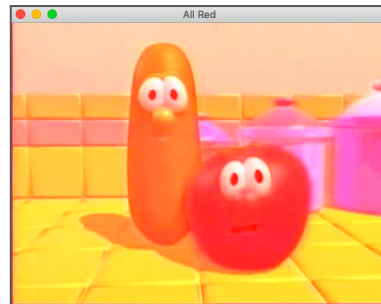
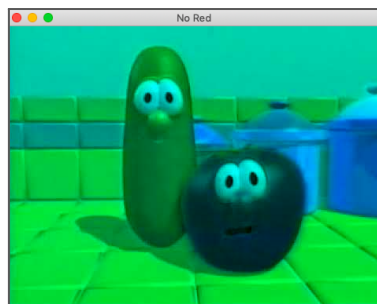
```
1
2 public class Lesson_06 {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7         // Define path String to image
8         String path = "bob_larry.jpeg";
9
10        // Create a new myPicture object
11        Picture myPicture = new Picture(path);
12
13        // Test the setAllRed() function
14        Picture zeroRed = ImageWorker.setAllRed(myPicture, 0);
15
16        // Set Title of zeroRed
17        zeroRed.setTitle("No Red");
18
19        // Display myPicture
20        myPicture.show();
21        zeroRed.show();
22    }
23 }
24
25 }
26
```

**Run the Program and note the two Images**



**Exercises:**

1. In the main() function of Lesson\_06, use ImageWorker.setAllRed() with the “Bob and Larry” picture and create and display a picture with red set to 255.
2. Import your own picture (less than 512x512 pixels) and try the setAllRed() function with different red values.



## Lesson 07: Assignment: setGreen() and setBlue()

### 07 Assignment: Implement setAllGreen() and setAllBlue()

**Objective:** Write functions to visit all the Pixels in a Picture object and set the green or blue pixels to the same value.

**Skills Needed:**

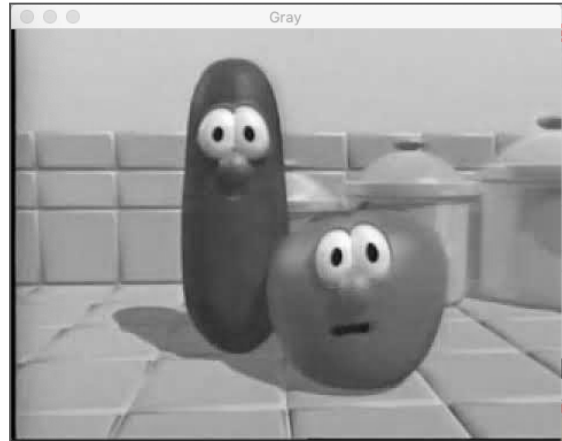
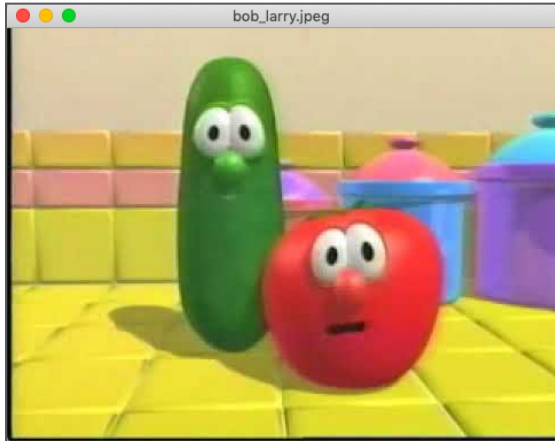
- Open a Picture Object
- Use the get(x, y) and set() functions
- Using For Loops in a “nested” style
- Using the .show() method

**Requirements Lesson** - You have to write code from list of Requirements

### Requirements

1. Open the “ImageWorker” project with setAllRed()
2. Write the Code Stubs as shown here for setAllGreen() and setAllBlue()
3. Complete the code to implement the setAllGreen() and setAllBlue() functions.
4. Create a Java class called “Lesson\_07” Using your own image, test and show the ImageWorker.setAllGreen() and ImageWorker.setAllBlue() functions with at least two different values for each function.
5. Display the original and the test images (You will have 5 images total)

## Lesson 08: Writing makeGrayscale() method



**Objective:** Write a function to visit all the Pixels in a Picture object and compute grayscale value to create a grayscale image.

**Skills Needed:**

- Open a Picture Object

- Use the `get(x, y)` and `set(x, y)` functions.

- Use the Color object and `getRed()`, `getGreen()`, and `getBlue()` methods

- Using For Loops in a “nested” style

- Reading Math formulas and converting to Java

- Using the `.show()` method

**Work-through Lesson** - Step by Step Instructions

## Overview of Process:

1. Open ImageWorker.java file.
2. Define the function stub in ImageWorker.java  
`public static Picture makeGrayscale(Picture p)`
3. Create a copy of the picture
4. Use nested for loops to visit, compute gray value, and write in each pixel of copy.
5. Create Lesson\_08 class and test in the main() function.

## Open ImageWorker.java file

1. Open Eclipse and find the Project “ImageManipulation”.
2. Open the ImageWorker.java file Leave the setAllRed(), setAllBlue() and setAllGreen() functions in the code. (Do not delete anything!)
3. Scroll down to below the setAllBlue() function. (Towards the end of the file)

## Define function makeGrayscale(Picture p)

4. In ImageWorker.java, find a starting place below your “setAll” functions.
5. Write the function definition for makeGrayscale(Picture p)
6. Notice that we have a ‘return null’ is written in function. This is a placeholder for a needed return statement.

```
// Make Gray scale image
public static Picture makeGrayscale(Picture p) {

    return null;
}
```

### Create a copy of the parameter Picture p

7. Inside the function, create a Picture object named “output” to hold a copy of the incoming picture.

8. Change the return statement to return output.

```
// Make Gray scale image
public static Picture makeGrayscale(Picture p) {

    // create copy named output
    Picture output = new Picture(p);

    // Return the output
    return output;
}
```

Note the process:

- a. Define output
- b. Do the work
- c. Return the output



## Setup nested for loops to visit each pixel value

We are going to visit each pixel to compute the new gray scale value. This is the “Do the work” part of the process

9. Write a for loop to move through all the y values

10. Write a for loop inside the y loop for the x values.

11. Notice the placement of the curly brackets to “nest” the Loop. We will go one row at a time and visit each value in the row. The next page will outline how to compute the gray scale value.

```
// Make Gray scale image
public static Picture makeGrayscale(Picture p) {

    // create copy named output
    Picture output = new Picture(p);

    // Visit all the Pixels and compute gray value
    for (int y = 0; y < p.getHeight(); y++) {

        for (int x = 0; x < p.getWidth(); x++) {

        }

    }

    // Return the output
    return output;
}
```

## Compute gray scale value and assign to Pixel

Now we will get the red, green, and blue values for each pixel and compute the grayscale value. The math is:

$$\text{gray} = (\text{red} + \text{green} + \text{blue}) / 3$$

Thus, we will average the color values of each pixel. There are other formulas for computing gray scale. ([See GIMP resource here](#)) We are using the average formula.

The general algorithm is written below. We will implement the code on the next page.

### Algorithm: Convert to Gray scale

For Each Color  $c$  at  $(x, y)$  from source Picture:

$R$  = red from Color  $c$

$G$  = green from Color  $c$

$B$  = blue from Color  $c$

$$\text{gray} = (R + G + B) / 3$$

    Create new Color with gray values

    Set color of output pixel at  $(x, y)$  to new Color

**Note:** Try this yourself inside the nested loop before looking at next slide

## Coding for Grayscale average algorithm

12. Use the algorithm and code sample below to implement creating a new color for each pixel

### Algorithm: Convert to Gray scale

For Each Color  $c$  at  $(x, y)$  from source Picture:

R = red from Color  $c$

G = green from Color  $c$

B = blue from Color  $c$

$\text{gray} = (R+G+B) / 3$

Create new Color grayColor with gray values

Set color of output at  $(x, y)$  to grayColor

```
public static Picture makeGrayscale(Picture p) {  
    // create output  
    Picture output = new Picture(p);  
  
    // Visit all the Pixels and compute gray value  
    for (int y = 0; y < output.height(); y++) {  
        for (int x = 0; x < output.width(); x++) {  
            // Get pixel color  
            Color c = output.get(x, y);  
  
            // Get Red, Green, and Blue  
            int red = c.getRed();  
            int green = c.getGreen();  
            int blue = c.getBlue();  
  
            // Math  
            int gray = (red + green + blue) / 3;  
  
            // Create new Color  
            Color grayColor = new Color(gray, gray, gray);  
  
            // Set pixel at x, y to grayColor  
            output.set(x, y, grayColor);  
        }  
    }  
  
    // Return the output  
    return output;  
}
```

## Create Lesson\_8 Class for Testing

13. Right click on the “src” folder and select “New-Class”
14. Name the class “Lesson\_08”
15. Make sure the box “public static void main(String [] args) is checked.
16. Click “Finish”

**Java Class**

The use of the default package is discouraged.

Source folder: Image\_Project\_01/src Browse...

Package: (default) Browse...

☐ Enclosing type: Browse...

Name: Lesson\_08

Modifiers: ☒ public ☐ package ☐ private ☐ protected  
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object Browse...

Interfaces: Add...  
Remove

Which method stubs would you like to create?

☒ public static void main(String[] args)  
☐ Constructors from superclass  
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))  
☐ Generate comments

? Cancel Finish

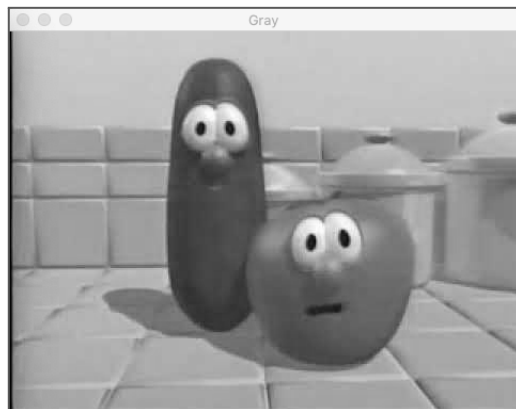
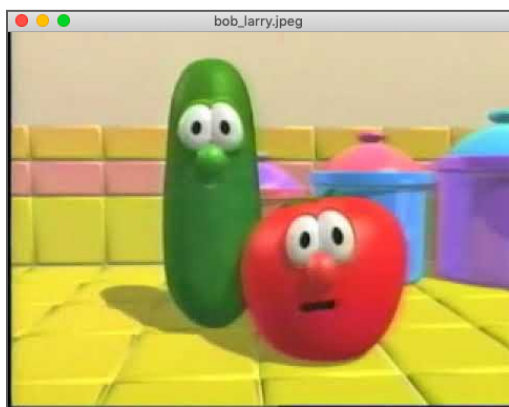
## Test Function and show in void main() of Lesson\_08

17. Find the static void main() function in Lesson\_08

18. Add code to create a new Picture object using the ImageWorker.makeGrayscale() function.

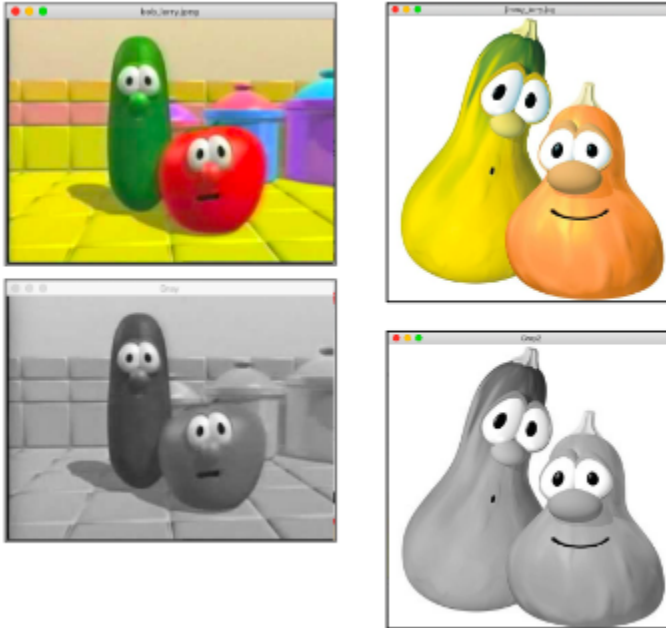
19. Call the .show() command to see the gray scale picture.

```
1
2 public class Lesson_08 {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7         // Define path String to image
8         String path = "bob_larry.jpeg";
9
10        // Create myPicture object
11        Picture myPicture = new Picture(path);
12
13        // Test Gray scale
14        Picture gray = ImageWorker.makeGrayscale(myPicture);
15        gray.setTitle("Gray");
16
17        //Display muPicture
18        myPicture.show();
19        gray.show();
20    }
21 }
22
23 }
24
```



## Exercises:

1. Import your own picture (less than 512x512 pixels) test with the `makeScaleFunction()`. Show the original picture and the grayscale picture.



2. For a challenge, write a function `makeLuminosity(Picture p)` and implement the Luminosity math as shown at: <https://docs.gimp.org/2.6/en/gimp-tool-desaturate.html>.

### Luminosity

The graylevel will be calculated as

$$\text{Luminosity} = 0.21 \times R + 0.72 \times G + 0.07 \times B$$

You will need to cast between `int` and `double` datatypes. Test the new function in the `void main()`.

## Lesson 09

### 09 Writing Images to Files

**Objective:** Now that we can do some pixel changes to images, we want to save the new pictures to the file system.

**Skills Needed:**

- Open a Picture Object
- Use one of our functions to create an edited picture
- Define Strings for the output path and output name
- Use the .write() function to save the new picture to the file system.

**Work-through Lesson** - Step by Step Instructions

Note: Depending on your computer, you may have to define the Path String to a very specific location on your file system.

This example uses OSX / Unix / Linux file structures.

### 09 Writing Images to Files Procedure

1. Open Lesson\_08 java file. Use the ImageWorker.makeGrayscale() function to make gray scale image.
2. Define a String fileName that is the name and extension of the file you wish to save.
3. Use the .save() function to save the gray scale image.
4. Use F5 to refresh the Project and see the new file.

Note: Depending on your computer, you may have to define the Path String to a very specific location on your file system.

This example uses OSX / Unix / Linux file structures.

## 1. Open Lesson\_08.java file and make a new image

1. Open the Lesson\_08 file and create an image object from one of your images.
2. Use the makeGrayscale() function to make a black and white image.
3. Use the .show() function to show and test both images.

```
1 public class Lesson_08 {
2
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7         // Define path String to image
8         String path = "bob_larry.jpeg";
9
10        // Create myPicture object
11        Picture myPicture = new Picture(path);
12
13        // Test Gray scale
14        Picture gray = ImageWorker.makeGrayscale(myPicture);
15        gray.setTitle("Gray");
16
17        //Display muPicture
18        myPicture.show();
19        gray.show();
20    }
21
22
23 }
24
```

## 2. Define String fileName that has name and extension

1. Add code to define String fileName has name and file extension for file you wish to save.

```
public class Lesson08 {
    public static void main(String[] args) {
        // TODO Auto-generated method stub

        // Define path String to image
        String path = "bob_larry.jpeg";

        // Create myPicture object
        Picture myPicture = new Picture(path);

        // Test Gray scale
        Picture gray = ImageWorker.makeGrayscale(myPicture);

        // Define name
        String fileName = "grayBobLarry.jpg";

        // Display myPicture
        myPicture.show();
        gray.show();
    }
}
```



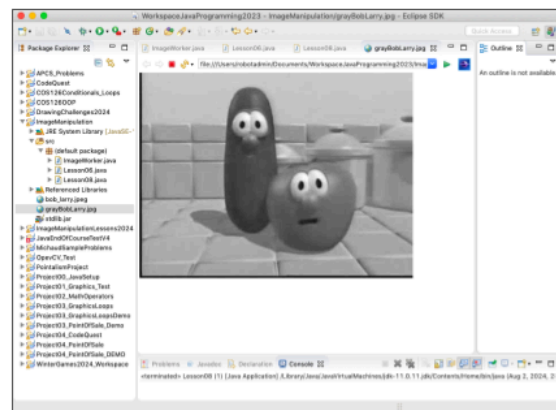
### 3. Use the .save() function to save the gray scale image

1. Use the .write(String path) function to save the Picture object to the file system.
2. Comment out the .show() commands for myPicture and gray.
3. Run the Code. You will not see any immediate results. (Go to next slide . . .)

```
public class Lesson08 {  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
  
        // Define path String to image  
        String path = "bob_larry.jpeg";  
  
        // Create myPicture object  
        Picture myPicture = new Picture(path);  
  
        // Test Gray scale  
        Picture gray = ImageWorker.makeGrayscale(myPicture);  
  
        // Define name  
        String fileName = "grayBobLarry.jpg";  
        gray.save(fileName);  
  
        // Display myPicture  
        //myPicture.show();  
        //gray.show();  
    }  
}
```

### 5. Refresh the Package Explorer

1. Right click on the Project Name "ImageManipulator" and select "Refresh" to refresh the project and see the file written to the system.
2. You can also press F5 to refresh the view.
3. Double click on the file to see the saved image.



We will use the .save() function to save images we create and edit in future lessons.

## 10 Assignment: Implement makeNegative() Function

**Objective:** Implement makeNegative() Function to “reverse” the colors of an image.

**Skills Needed:**

- Open a Picture Object
- Use the get(x, y), Color Objects, and math operations
- Using For Loops in a “nested” style
- Read and interpret mathematical procedure (algorithm)
- Use the .save() function to save your work.

**Requirements Lesson** - You have to write code from list of Requirements

## 10 Assignment: Algorithm for Negative Images

**Algorithm: Negative Images**

For each Color at pixel (x, y) of input Picture:

Get Color c from pixel at (x, y)

R = red from Color c

G = green from Color c

B = blue from Color c

// Subtract each color channel from 255 (Max value)

newRed = 255 - R

newGreen = 255 - G

newBlue = 255 - R

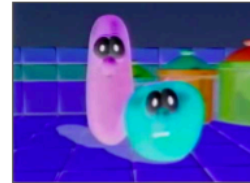
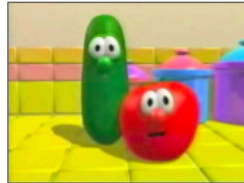
Create a new Color newC with newRed, newGreen, and newBlue values

Assign newColor to output pixel at (x, y)

## 10 Assignment: Requirements

1. Open the "ImageWorker" class.
2. Scroll down and Write the Code Stub as shown for makeNegative() function.
3. Complete the code to implement the makeNegative() function.
4. Create a new class called "Lesson\_10"
5. Using "Bob and Larry" image, test and save your negative as "NegativeBob.jpg".
6. Test the makeNegative() on your own image and save as "MyNegative.jpg".

```
// Make Negative  
public static Picture makeNegative(Picture p) {  
    return null;  
}
```



Hint: Use Assignment 08  
Writing makeGrayscale()  
function as a guide to setting  
up makeNegative()

## Lesson 11 Reversing Images on X and Y Axis

**Objective:** Write functions to visit all the Pixels in a Picture object and output a image Reversed on the X or Y axis.

**Skills Needed:**

Open a Picture Object

Use the get(x, y) and

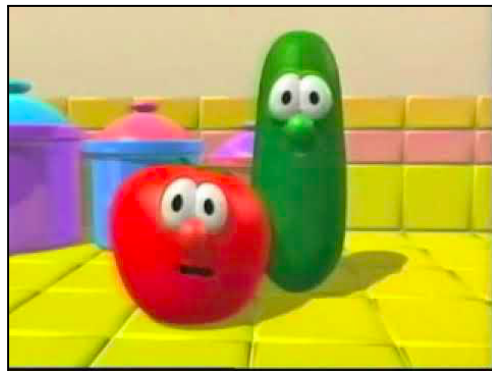
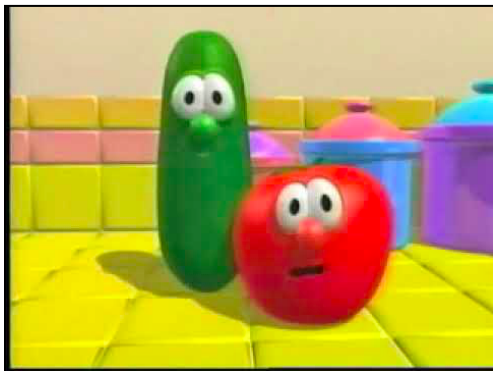
Color objects.

Using For Loops in a “nested” style

Iterating backwards and forwards through loops

**Work-through Lesson** - Step by Step Instructions for X

**Assignment Lesson** - Implement from Requirements for Y



## Overview: Reversing Images on X Axis:

1. Open the ImageWorker.java file.
2. Define function stub for reverseOnX(Picture p)
3. Setup output object and nested for Loop
4. Implement the pixel color transfer
5. Create new class "Lesson\_11".
6. In class Lesson\_11 import a picture and test the ImageWorker.reverseOnX() function.

### Open the ImageWorker.java file

1. Open the ImageWorker.java
2. Scroll down below the functions to find a line define the new function. Be sure to leave all the functions you have defined so far as you will need them in later lessons.
3. Write the function sub for reverseOnX(Picture p) in the ImageWorker class. Note the use of return null for the sub.

```
// Reverse on X function
public static Picture reverseOnX(Picture p) {
    return null;
}
```

### Setup the output object and next for loop:

4. Create a Picture object "output"
5. Setup nested for loop iterating through y and x axis:

```
// Reverse on X function
public static Picture reverseOnX(Picture p) {

    // Setup output Picture
    Picture output = new Picture(p);

    // Do the Work - nested for loop
    for (int y = 0; y < p.getHeight(); y++) {
        for (int x = 0; x < p.getWidth(); x++) {

        }
    }

    // Return statement
    return output;
}
```

## Concept: Reversing values on Row of Pixels

Before we implement the math to transfer pixels, let us visualize a “row” of pixels that only hold one number (like an Array of integers):

100	150	200	225	263
-----	-----	-----	-----	-----

Each location on the row has an address. In the above example of five elements the addresses of the “x” position of each element would be:

0	1	2	3	4
100	150	200	225	263

To reverse the numbers, we will copy them into a blank array in reverse order:

0	1	2	3	4

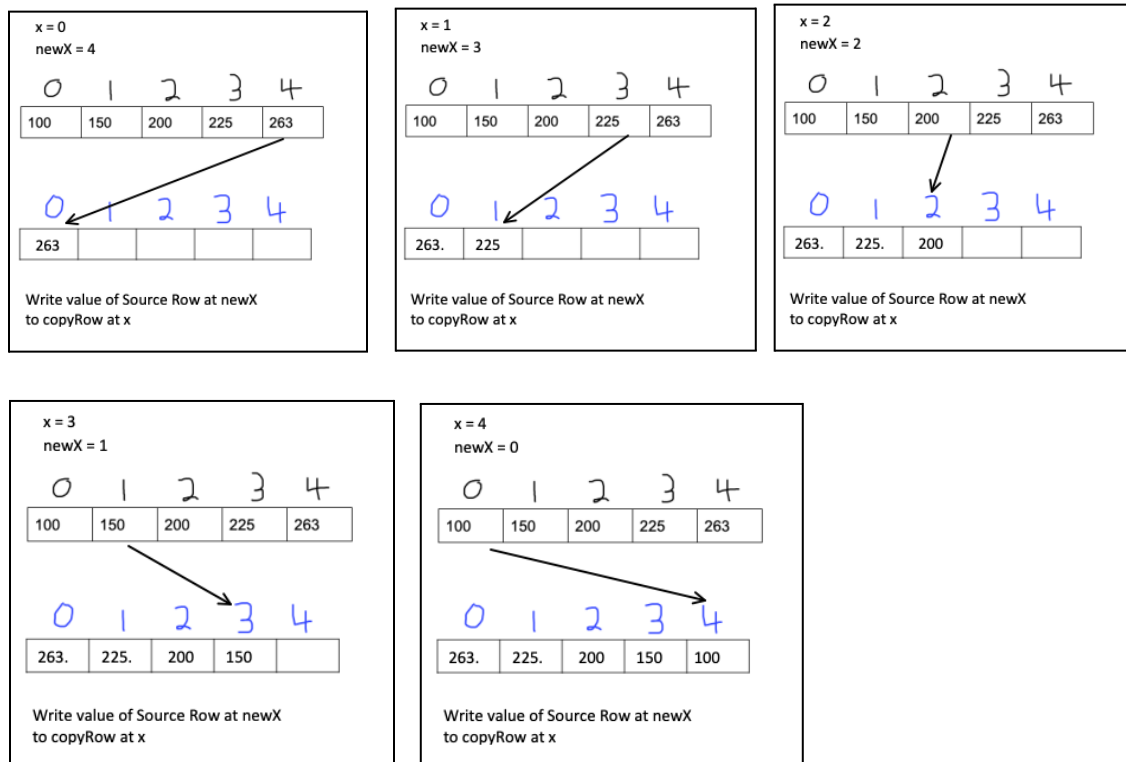
We are going to setup a T-Chart that tracks the values of “x” (index of original row) and “newX” (values for blank array):

x	newX
0	4
1	3
2	2
3	1
4	0

Note that the mathematical relationship between x and newX is:

$$\text{newX} = 4 - x$$

So, we are going to “get” the Color from the original Row at “newX” and then write the Color at the copy row at X:



Thus, for each row of the image, we are going to work forwards counting on x and then create a new variable newX = width - x - 1. Then we transfer the color from the source picture at newX to the output picture at x.

```
for (int x = 0; x < p.width(); x++) {
    // set newX to move backwards
    int newX = p.width() - 1 - x;

    // Get source color
    Color c = p.get(newX, y);

    // Assign color to output
    output.set(x, y, c);
}
```



## Implement the pixel color transfer

6. Setup a “newX” that is equal to the width of the image - 1 - the current x position.
7. Get the source Color c at  
(newX, y)
8. Write the source Color c to the output pixel at (x, y)

```
// Reverse on X function
public static Picture reverseOnX(Picture p) {

    // Setup output Picture
    Picture output = new Picture(p);

    // Do the work – nested for loop
    for (int y = 0; y < p.height(); y++) {
        for (int x = 0; x < p.width(); x++) {
            // set newX to move backwards
            int newX = p.width() - 1 - x;

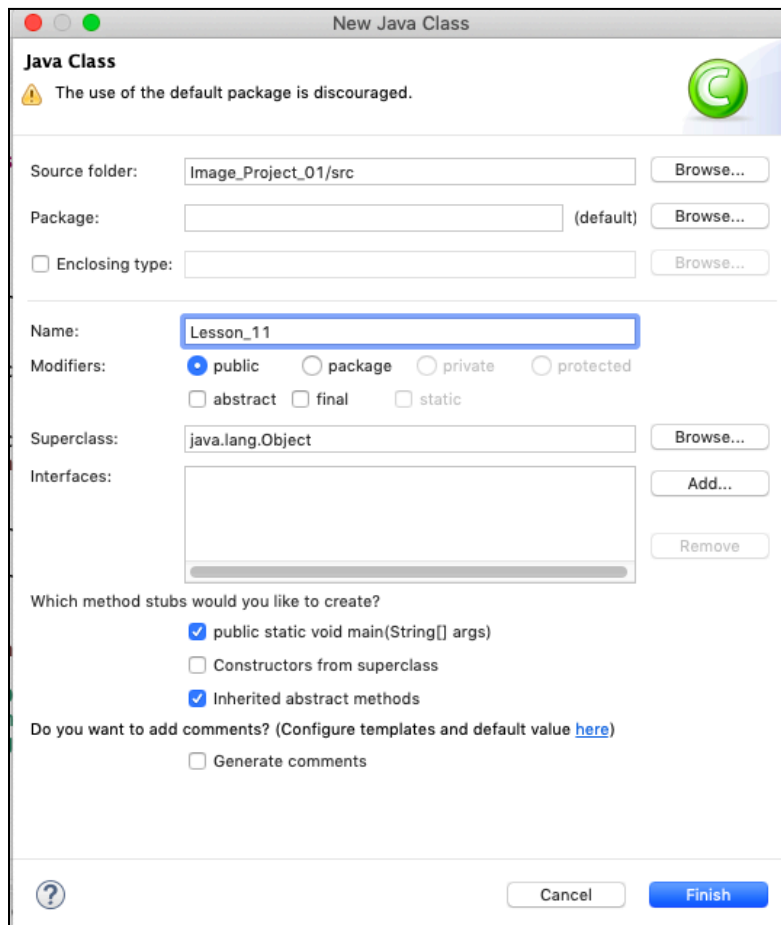
            // Get source color
            Color c = p.get(newX, y);

            // Assign color to output
            output.set(x, y, c);
        }
    }

    // Return statement
    return output;
}
```

## Create a new Class called “Lesson\_11”

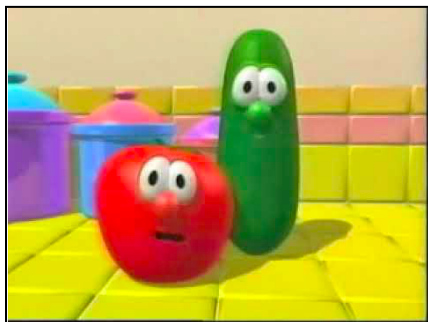
9. Right click on the “src” folder.
10. Select “New - Class”
11. Name the class “Lesson\_11”
12. Make sure the box “public static void main(String [] args)” is checked
13. Click “Finish”



## Test and write output to File system.

14. Go to the main() function on Lesson\_11
15. Use the ImageWorker.reverseOnX() to make a new Picture object called reverseX
16. Set the fileName String
17. Use the .write() function to save reverseX as "reverseX.jpg".

```
public class Lesson11 {  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
  
        // Define path String to image  
        String path = "bob_larry.jpeg";  
  
        // Create myPicture object  
        Picture myPicture = new Picture(path);  
  
        // Create mirrorX  
        Picture reverseX = ImageWorker.reverseOnX(myPicture);  
  
        // Save picture to file  
        String fileName = "reverseX.jpg";  
        reverseX.save(fileName);  
    }  
}
```



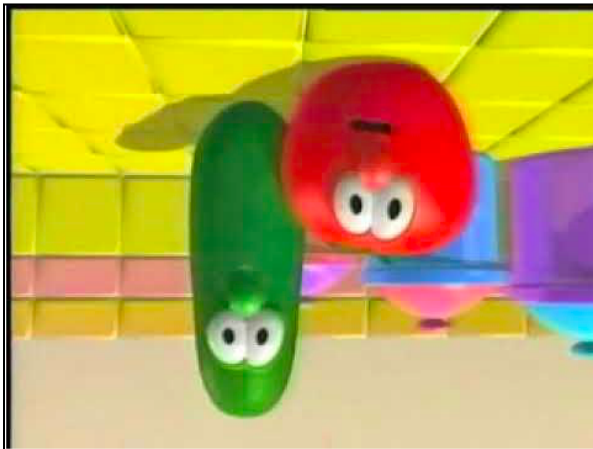
### Assignment: Implement reverseOnY(Picture p)

1. In the ImageWorker.java file, create a function stub public static Picture reverseOnY(Picture p)
2. Implement code to reverse image on Y axis. (Hint, you can change the nested loop to move through X first).

```
for (int x = 0; x < p.width(); x++) {  
    for (int y = 0; y < p.height(); y++) {  
    }  
}
```

3. Test ImageWorker.reverseOnY() in main function of Lesson\_11 and write results of test to File system as “reverseY.jpg”

```
// Reverse on Y  
public static Picture reverseOnY(Picture p) {  
    return null;  
}
```



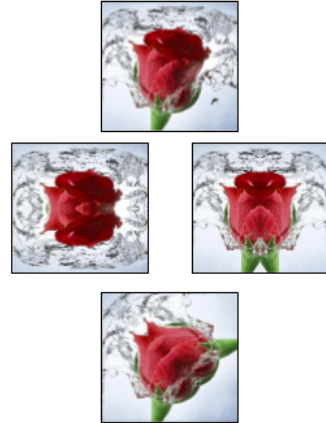
## 12 Assignment: Mirror on X, Mirror on Y, Mirror on Diagonal

**Objective:** Write functions to visit all the Pixels in a Picture object and output a image Mirrored on the X, Y, or Diagonal axis.

**Skills Needed:**

- Open a Picture Object
- Use the get(x, y) Accessors and Modifiers
- Using For Loops in a “nested” style
- Developing techniques for iterating halfway through loops

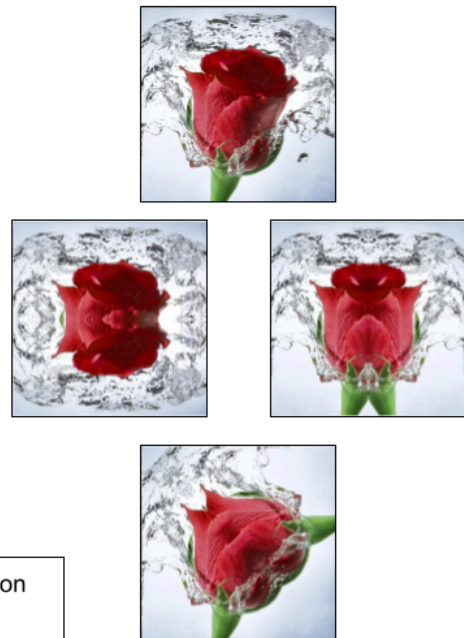
**Assignment Lesson** - Implement from Requirements



## 12: Requirements

1. On ImageWorker.java, create a function stub  
public static Picture mirrorOnX(Picture p)
2. On ImageWorker.java, create function stub  
public static Picture mirrorOnY(Picture p)
3. On ImageWorker.java, create function stub  
public static Picture mirrorOnDiag(Picture p)
4. Implement code for functions.
5. Create a new Java file called “Lesson\_12”
6. Test in main() function of Lesson\_12 and write  
outputs to File System

Hint: These work best on  
square images



## Lesson 13 Averaging Pictures

### 13 Averaging Pictures

**Objective:** Write function to combine (average) two images.

**Skills Needed:**

- Finding two images with same dimension (512 x 512)
- Definition of mean (average) and formula
- Iterating through Pixels and computing mean
- Writing pixels to output image
- Saving to file

**Work-through Lesson** - Step by Step for averaging to Arrays

**Assignment Lesson** - Implement averaging for images.

### Averaging Pictures: Mathematical Background

**Mean:** Average of a set of numbers

**Example:**

$$x = 10$$

$$y = 6$$

**Mean of x and y is computed as:**

$$\text{mean} = (x + y) / 2$$

$$\text{mean} = (10 + 6) / 2$$

$$\text{mean} = 16 / 2 = 8$$

$$\text{mean} = 8$$

## Averaging Pictures: Apply to array of numbers

**Mean:** Average of a set of numbers

**Example:**

A = 

10	8	7	12	15
----	---	---	----	----

B = 

4	10	9	7	5
---	----	---	---	---

avg = 

7	9	8	9.5	10
---	---	---	-----	----

**Algorithm: Average Arrays**

Given Array A

Given Array B // same size as A

Initialize C = new Array size of A

For i = 0, i < Size of A, i = i + 1

    newValue = (A[i]+B[i]) / 2

    C at index i = newValue

Return C

## Work through: Average two Arrays

**Overview:**

1. Create new Java Class "ArrayTester.java"
2. Write function `public static double [] averageArrays()`
3. Write a function `public static void printArray()`
4. Create two Arrays in void main()
5. Test function and print results.

**Algorithm: Average Arrays**

Given Array A

Given Array B // same size as A

Initialize C = new Array size of A

For i = 0, i < Size of A, i = i + 1

    newValue = (A[i]+B[i]) / 2

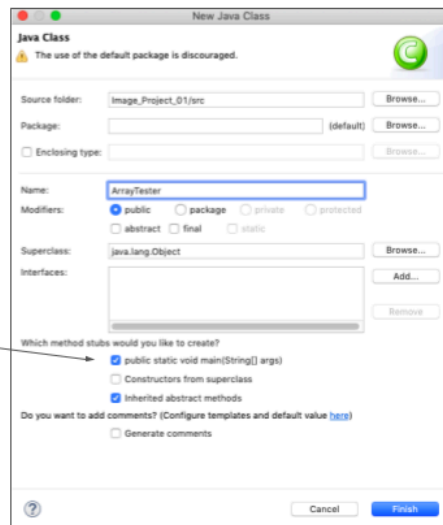
    C at index i = newValue

Return C

## 1. Create new Java Class “ArrayTester.java”

### Overview:

1. Right click on the (default package) icon and select “New -> Class”
2. Name the class “ArrayTester” and make sure the option “public static void main(String [] args)” is checked.
3. Click “Finish”



## 2a. Start function averageArrays(double [] a, double [] b)

### Overview:

1. Start below the void main() function and write the function stub for public static double [] averageArrays(). (Lines 9 through 13 in example)
2. Note the return null as a placeholder for now.

```
1 public class ArrayTester {  
2  
3  
4     public static void main(String[] args) {  
5         // TODO Auto-generated method stub  
6     }  
7  
8  
9     // Average Arrays Function  
10    public static double [] averageArrays(double [] a, double [] b) {  
11  
12        return null;  
13    }  
14 }  
15  
16  
17
```



## 2b. Define output for averageArrays() function

### Overview:

1. Inside the averageArrays() function, initialize an array named output that is the same length as the input array a.
2. Change the return statement to return output.

```
// Average Arrays Function
public static double [] averageArrays(double [] a, double [] b) {

    // Initialize output array
    double [] output = new double [a.length];

    // Return output object
    return output;
}
```

## 2c. Use for loop to do math and average elements of a and b

### Overview:

1. Implement algorithm below and use for loop to iterate through the input arrays and compute mean.

#### Algorithm: Average Arrays

Given Array A  
Given Array B // same size as A

Initialize C = new Array size of A

For i = 0, i < Size of A, i = i + 1  
    newValue = (A[i]+B[i]) / 2  
    C at index i = newValue

Return C

```
// Average Arrays Function
public static double [] averageArrays(double [] a, double [] b) {

    // Initialize output array
    double [] output = new double [a.length];

    // Do the work: Use for loop and compute averages
    for (int i = 0; i < a.length; i++) {

        // Compute mean at index i
        double newValue = (a[i]+b[i]) / 2;

        // Write mean to output at index i
        output[i] = newValue;

    }

    // Return output object
    return output;
}
```

### 3. Write function printArray(double [] a)

#### Overview:

1. Implement the function printArray() that will display the values of an array to the console.
2. Define this below the averageArrays() function.
3. We will use this later.

```
// Printing contents of Array
public static void printArray(double [] a) {

    String output = "[";

    for (int i = 0; i < a.length-1; i++) {
        String value = String.valueOf(a[i]);
        output += value + ", ";
    }

    output += String.valueOf(a[a.length-1]) + "]";

    System.out.println(output);

}
```

### 4. Create two Arrays in void main()

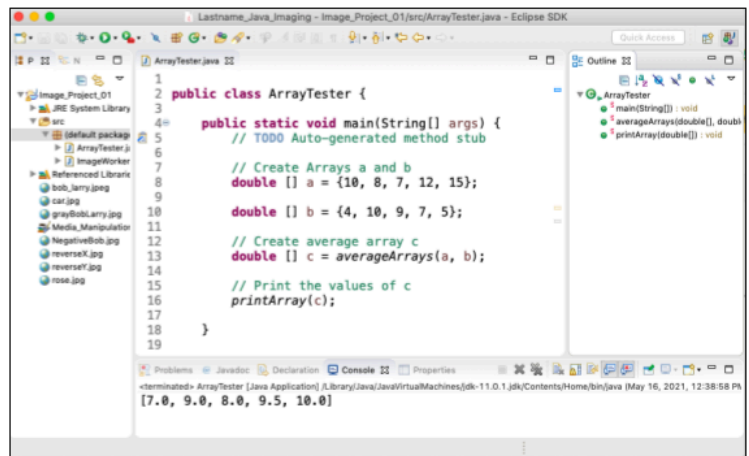
#### Overview:

1. Go back to the void main() function and define two Arrays a and b as shown on lines 7 through 10.

```
1
2 public class ArrayTester {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7         // Create Arrays a and b
8         double [] a = {10, 8, 7, 12, 15};
9
10        double [] b = {4, 10, 9, 7, 5};
11
12    }
13
14 }
```

## 5. Test function and print results.

1. Use the `averageArrays()` function to create a new array called `c`.
2. Use the `printArray()` function to display the output.
3. Run the code. The values of the output in the console should be the average of each element of the input.



The screenshot shows the Eclipse IDE with the `ArrayTester.java` file open. The code defines a `public class ArrayTester` with a `main` method. Inside `main`, two double arrays `a` and `b` are created. Array `a` contains the values `{10, 8, 7, 12, 15}` and array `b` contains `{4, 10, 9, 7, 5}`. The `averageArrays(a, b)` function is called to create array `c`. Finally, `printArray(c)` is called to display the results. The console output at the bottom shows the array `[7.0, 9.0, 8.0, 9.5, 10.0]`.

```
1 public class ArrayTester {
2
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7         // Create Arrays a and b
8         double [] a = {10, 8, 7, 12, 15};
9
10        double [] b = {4, 10, 9, 7, 5};
11
12        // Create average array c
13        double [] c = averageArrays(a, b);
14
15        // Print the values of c
16        printArray(c);
17
18    }
19 }
```

Console Output: `[7.0, 9.0, 8.0, 9.5, 10.0]`

## Averaging Images: Overview of Algorithm

### Algorithm: Averaging Pictures

Given: Picture A

Given: Picture B // Same width and height as A

Initialize output Picture C as copy of A // Or blank image at same size as A

For each pixel in A at location (x, y)

    Initialize colorA as Color at pixel(x, y) in A

    Initialize colorB as Color at pixel(x, y) in B

        Initialize averageRed as (colorA's red + colorB's red) / 2

        Initialize averageGreen as (colorA's green + colorB's green) / 2

        Initialize averageBlue as (colorA's blue + colorB's blue) / 2

        Create new Color colorC with values of (averageRed, averageGreen, averageBlue)

        Set color at output(x,y) as colorC

Return Picture C

## Averaging Images: Setup

We will now apply the skill of averaging values in 1D Arrays to images.

1. Find two images of same size. I suggest you use 512 x 512 pixels. Save these pictures to your "ImageManipulation" area on Eclipse.
2. Go to the ImageWorker.java file
3. Define the function stub for  
`public static Picture averagePictures(Picture a, Picture b)`
4. Implement averagePictures() using algorithm on previous slide
5. Create a new Java class called "Lesson\_13"
6. In the main method of Lesson\_13.java, define String paths for two pictures and create Picture objects.
7. Test the ImageWorker.averagePictures() function on your two pictures in Lesson\_13.java

## Averaging Images: Requirements

1. Find two images of same dimensions. You can use [PictureA](http://nebomusic.net/javalessons/ImageManipulation/bob_larryV2.jpg) and [PictureB](http://nebomusic.net/javalessons/ImageManipulation/jimmy_jerry.jpg) as samples to get started.

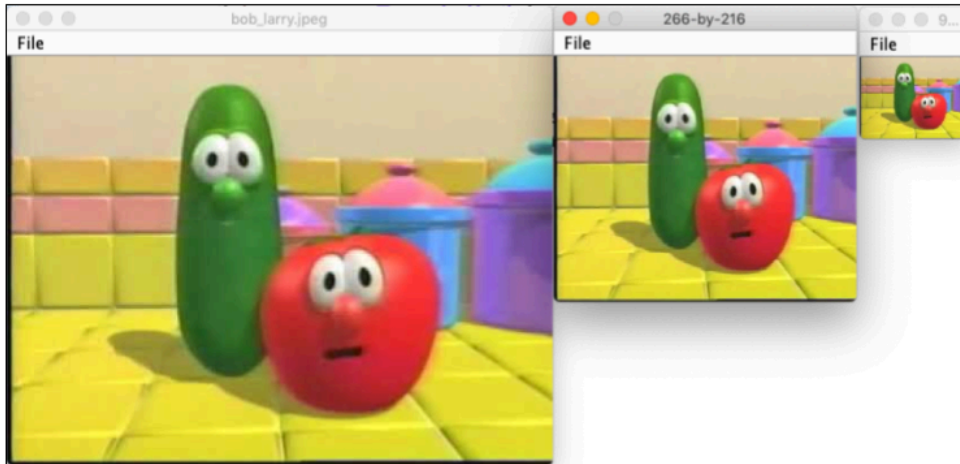
PictureA: [http://nebomusic.net/javalessons/ImageManipulation/bob\\_larryV2.jpg](http://nebomusic.net/javalessons/ImageManipulation/bob_larryV2.jpg)

PictureB: [http://nebomusic.net/javalessons/ImageManipulation/jimmy\\_jerry.jpg](http://nebomusic.net/javalessons/ImageManipulation/jimmy_jerry.jpg)

2. Implement the averagePictures(Picture a, Picture b) function in ImageWorker
3. Create new class "Lesson\_13"
4. Test the ImageWorker.averagePictures(Picture a, Picture b) function in the void main() of Lesson\_13.
5. Save the output of the average image to the file system.
6. Make sure to run this algorithm on your own pictures in addition to the samples.

## Lesson 14 Resizing Images

### resize() Method Operation

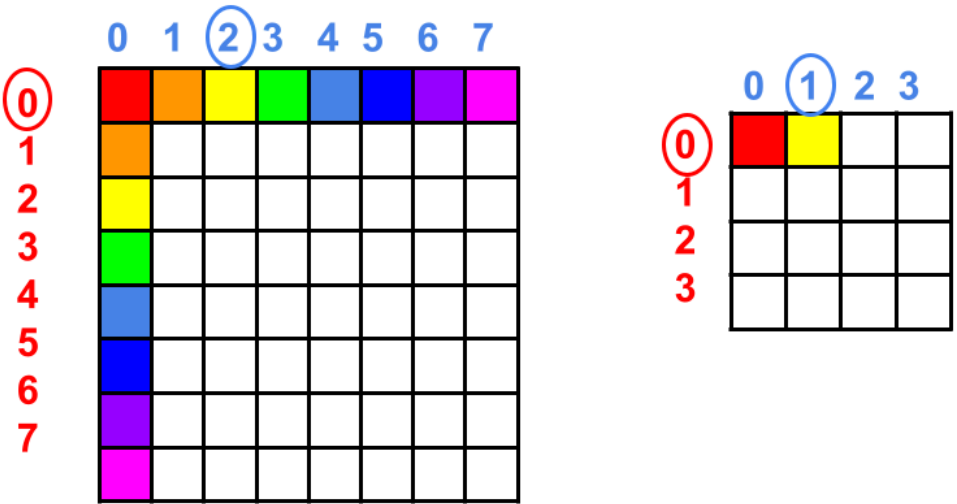


### resize () Method Operation:

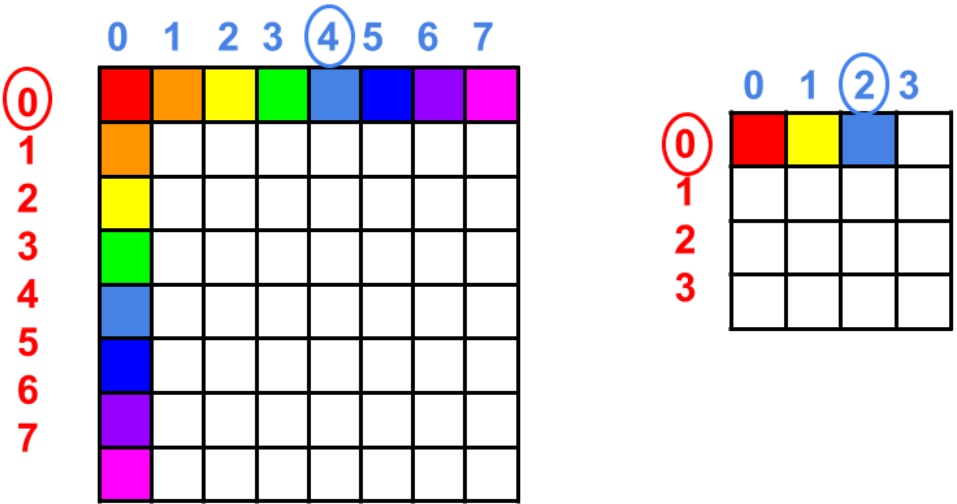
	0	1	2	3	4	5	6	7
0	Red	Orange	Yellow	Green	Light Blue	Blue	Purple	Pink
1	Orange							
2	Yellow							
3	Green							
4	Light Blue							
5	Blue							
6	Purple							
7	Pink							

	0	1	2	3
0				
1				
2				
3				

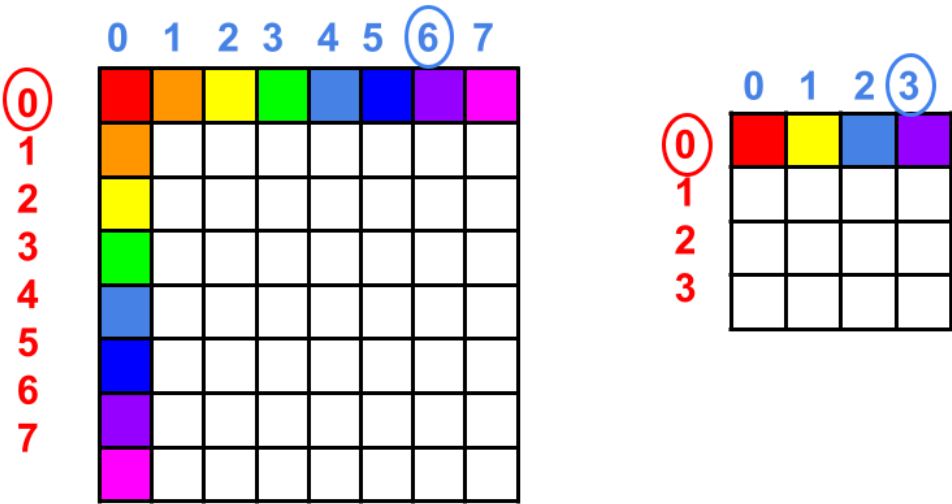
resize () Method Operation:



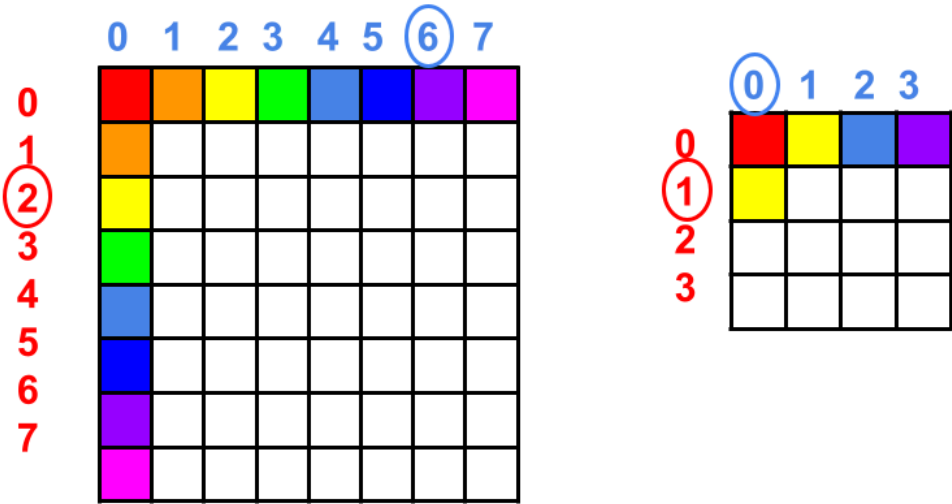
resize () Method Operation:



resize () Method Operation:



resize () Method Operation:





```

public static Picture resize(Picture p, int newWidth, int newHeight) {
    // Create output object
    Picture output = new Picture(newWidth, newHeight);

    // Calculate delta values
    double deltaX = p.width() * 1.0 / newWidth;
    double deltaY = p.height() * 1.0 / newHeight;

    // Loop and Move Pixel values
    for (int y = 0; y < output.height(); y++) {
        for (int x = 0; x < output.width(); x++) {
            int targetX = (int) (deltaX * x);
            int targetY = (int) (deltaY * y);

            Color c = p.get(targetX, targetY);
            output.set(x, y, c);
        }
    }

    // Return output object
    return output;
}

```

## Lesson 15 Creating New Image and Combining Images

**Objectives:** Create a Blank Picture Object to a Specific Size  
Write a function to Copy pixels from one  
Picture object to Another.

**Skills Needed:**

Using Nested For Loops with Picture objects  
Getting and Setting Color Values from Pixels  
Using functions we have already written to  
create new edited Picture Objects

**Work-through Lesson** - Step by Step for creating blank Object and placePicture() function.

### Overview of Process:

1. Create a new class called "Lesson\_15"
2. Use the Picture(int x, int y) constructor to create a blank canvas in the main method of "Lesson\_15"
3. Go to the "ImageWorker" file and implement the placePicture() function.
4. Go back to "Lesson\_15" and experiment with placing Picture objects, using effects functions, and resizing existing pictures.
5. Write canvas Picture object to file.

## Create new Java class “Lesson\_15” and import two pictures

1. Create new Java Class “Lesson\_15”.
2. In the main method, import two pictures. You may use any pictures you like. I am going to use the rose.jpg and bob\_larry.jpg.

```
1
2 public class Lesson_15 {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7         // Define path String to image
8         String pathBob = "bob_larryV2.jpg";
9         String pathRose = "rose.jpg";
10
11         Picture bob = new Picture(pathBob);
12         Picture rose = new Picture(pathRose);
13
14     }
15 }
16
17
```

## Use the Picture(int x, int y) constructor to create a blank canvas.

3. We will now make a blank Picture object called “canvas” using the Picture(x, y) constructor.
4. Because the source pictures are 512 x 512 pixels, we are going to make the canvas 2048 wide and 2048 tall.

```
1
2 public class Lesson_15 {
3
4     public static void main(String[] args) {
5         // TODO Auto-generated method stub
6
7         // Define path String to image
8         String pathBob = "bob_larryV2.jpg";
9         String pathRose = "rose.jpg";
10
11         Picture bob = new Picture(pathBob);
12         Picture rose = new Picture(pathRose);
13
14         // Make a Canvas
15         Picture canvas = new Picture(2048, 2048);
16
17     }
18 }
19
20
```

## Define the placePicture() function in ImageWorker

5. Go to the ImageWorker class.
6. Scroll down and write a function stub for public static Picture placePicture().
7. Use a return null as the placeholder.
8. We will start the implementation on next page

```
// Place Picture Function  
public static void placePicture(Picture p, Picture canvas, int x, int y) {  
  
}
```

### Parameters:

Picture p: Picture to place

Picture canvas: Destination for p

int x: X position for starting point (Upper Left corner)

int y: Y position for starting point (Upper Left corner)

## Setup the Nested For Loop

9. Setup the nested for loops to copy the pictures from p to canvas. Notice we will use 'r' for vertical (Y) positions and 'c' for horizontal (X) positions.

```
// Place Picture Function
public static void placePicture(Picture p, Picture canvas, int x, int y) {

    // Nested For Loop
    for (int r = 0; r < p.height(); r++) {
        for (int c = 0; c < p.width(); c++) {

            }
        }
    }
}
```

## Compute destination (x, y) and copy Pixel Colors

10. Inside the nested for loop, compute dX and dY using the x and y parameters.

11. Get the Color at (c, r) from p. Name this color "pS"

12. Set the color of canvas at (dX, dY) to the Color from p at (c, r) we named "pS"

```
// Place Picture Function
public static void placePicture(Picture p, Picture canvas, int x, int y) {

    // Nested For Loop
    for (int r = 0; r < p.height(); r++) {
        for (int c = 0; c < p.width(); c++) {

            // Set destination values for canvas
            int dX = c + x;
            int dY = r + y;

            // Get original color from input Picture p
            Color pS = p.get(c, r);

            // Set the canvas destination pixel to pS color
            canvas.set(dX, dY, pS);

        }
    }
}
```

## Experiment with placing Picture objects, using effects functions, and resizing existing pictures in Lesson\_15 class

13. Go back to Lesson\_15 and find the main() function. Create some new images using the functions you wrote.

14. I am going to experiment with the setAllRed(), setAllBlue(), setAllGreen() from ImageWorker

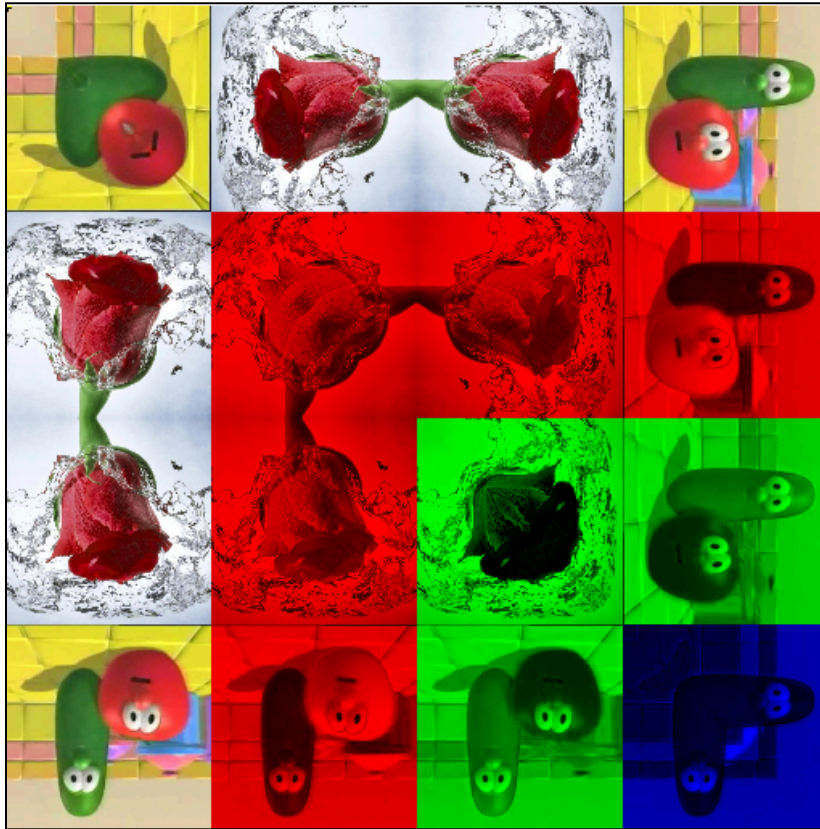
15. Use the ImageWorker.placePicture() to put these on the canvas.

16. Use canvas.show() to test.

```
public class Lesson15 {  
  
    public static void main(String[] args) {  
        // TODO Auto-generated method stub  
  
        // Get a Picture from file  
        String pathBob = "bob_larryV2.jpeg";  
        Picture bob = new Picture(pathBob);  
  
        // Make a Canvas  
        Picture canvas = new Picture(2048, 2048);  
  
        // Edit Pictures  
        Picture redBob = ImageWorker.setAllGreen(bob, 0);  
        redBob = ImageWorker.setAllBlue(redBob, 0);  
  
        Picture greenBob = ImageWorker.setAllRed(bob, 0);  
        greenBob = ImageWorker.setAllBlue(greenBob, 0);  
  
        Picture blueBob = ImageWorker.setAllGreen(bob, 0);  
        blueBob = ImageWorker.setAllRed(blueBob, 0);  
  
        // Place on Canvas  
        ImageWorker.placePicture(bob, canvas, 0, 0);  
        ImageWorker.placePicture(redBob, canvas, 512, 0);  
        ImageWorker.placePicture(greenBob, canvas, 1024, 0);  
        ImageWorker.placePicture(blueBob, canvas, 1536, 0);  
  
        // Show Canvas  
        canvas.show();  
    }  
}
```

### Write Picture to File:

1. Experiment with editing / writing code to manipulate your images.
2. Write these images to the canvas.
3. Save the canvas to the File system as “canvas.jpg”.



## 16 Final Project: Create a Picture Collage with Java

**Objectives:** Create a collage using a collection of images and applying / creating effects with Java.

**Skills Needed:**

- Create image editing functions
- Importing Images as Picture Objects
- Editing Images with Java functions
- Creating Canvas and arranging Images
- Saving to File System

**Project** - Apply what you have learned and create a product

## 16 Final Project: Project Setup and Planning

1. Create a new class called "Final\_Project" in "Image Project 01"
2. Collect source images and place them in the "Image Project 01" project on Eclipse.
3. Develop a written plan or sketch of what you want to do with the images.
4. Plan out any new editing functions (turning images, adding borders, picture in picture . . .)
5. Create a Picture object canvas in "Final\_Project" to place edited pictures on.
6. Implement any additional functions in ImageWorker
7. Create Code on "Final\_Project"
8. Test and Refine



## Ideas . . . Try these in your project

1. Research the functions to change the size of a picture:
2. Write functions to turn pictures 90 degrees right or left.
3. Research photo filters and try to apply with Java
4. Experiment with Image “Averaging” to simulate Opacity

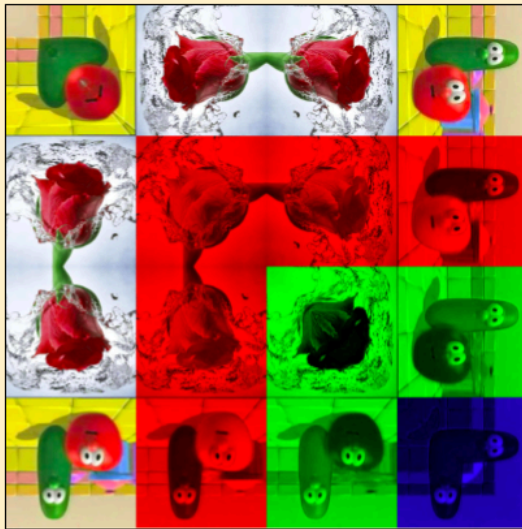
## Requirements

1. Use at least 4 different source images in artwork.
2. Use at least 4 image manipulation functions from ImageWorker (these are functions you defined in lessons)
3. Create and use at least one new manipulation function in ImageWorker (see previous slide for suggestions)
4. Use one of the scaling functions to change the size of images.
5. Collage canvas must be at least 1000 by 800 pixels.
6. Collage canvas must have at least 10 different images (can be repeats with various edits using functions)

## Deliverables (To Google Classroom or your Website)

1. Source Code for your ImageWorker.java
2. Source Code for your Final\_Project.java  
Put your name in comments on the first line  
// Firstname Lastname
3. Your Collage in .jpg format.
4. A Google Slide with the following:
  - $\frac{2}{3}$  of the Slide will be your Collage
  - $\frac{1}{3}$  of the Slide will be a short paragraph describing the design
  - Slide will be formatted and attractive

### ***Collage Title by Firstname Lastname***



Short paragraph on how you made the collage goes here. Describe your design process, function calls, and patterns you used to create the artwork.

If you created some of your own picture functions, describe them here and what they do.

## Project 01: Duotone Filter (by Kevin Wayne)

A *duotone filter* involves using two colors to create a new image. In particular, a duotone filter is a way to reproduce an image, using combinations of only two ink colors, *color1* and *color2*.

It is a popular effect for photographers and digital artists. (Here's a nice [site](#) for experimentation.)

To apply a duotone filter to an image, consider each pixel of a source image one at a time:

- Let  $(r, g, b)$  denote the *red*, *green*, and *blue* components, respectively of the pixel. Each component is an integer between 0 and 255.
- Let  $(r_1, g_1, b_1)$  and  $(r_2, g_2, b_2)$  denote the *red*, *green*, and *blue* components of *color1* and *color2* respectively.
- Change the color of the pixel from  $(r, g, b)$  to  $(r', g', b')$  by applying the following formulas:
  - First, compute the *monochrome luminance* of the given color as an intensity between 0.0 and 255 using the NTSC formula:
    - $luminance = (0.299r + 0.587g + 0.114b) / 255.0$
  - Then, compute  $r', g', b'$  using the formulas:
    - $r' = luminance * r_1 + (1 - luminance) * r_2$
    - $g' = luminance * g_1 + (1 - luminance) * g_2$
    - $b' = luminance * b_1 + (1 - luminance) * b_2$
    - When computing  $r', g', b'$  round the result to the nearest integer so that  $r', g', b'$  are integers between 0 and 255.

**Hint for rounding:**

```
double red = 1.75;

int redRounded = (int) Math.round(red);
```

## Requirements:

Write a program `Duotone.java` that applies a duotone filter to an image, and displays the results in a window. You may use `StdPicture` or `Picture` to read, modify, and display the picture. More information on `StdPicture` and example can be found below:

```
public class StdPicture
```

<code>void read(String filename)</code>	<i>initialize picture from file filename</i>
<code>int width()</code>	<i>return the width of the picture</i>
<code>int height()</code>	<i>return the height of the picture</i>
<code>int getRed(int col, int row)</code>	<i>return the red component of pixel (col, row)</i>
<code>int getGreen(int col, int row)</code>	<i>return the green component of pixel (col, row)</i>
<code>int getBlue(int col, int row)</code>	<i>return the blue component of pixel (col, row)</i>
<code>void setRGB(int col, int row, int r, int g, int b)</code>	<i>set the color of pixel (col, row) to (r, g, b)</i>
<code>void show()</code>	<i>display the picture in a window</i>
<code>void save(String filename)</code>	<i>save the picture to file filename</i>

*API for our library of static methods for standard picture*

## Example:

```
public class DuoTone {
    public static void main(String[] args) {

        // Define path String and Open Picture
        String path = "bob_larryV2.jpg";
        StdPicture.read(path);

        // Read colors at pixel (10, 10)
        int red = StdPicture.getRed(10, 10);
        int green = StdPicture.getGreen(10, 10);
        int blue = StdPicture.getBlue(10, 10);

        // Show Picture on Screen
        StdPicture.show();

    }
}
```

For example, given a photo of Johnson Arch:



Duotone Image:



In this sample execution,  
*color1* is Princeton orange (245, 128, 37) and  
*color2* is black (0, 0, 0).

Take a photo of a building, statue, gate, or other structure on campus and select two different colors for  $(r_1, g_1, b_1)$  and  $(r_2, g_2, b_2)$ . Apply the duotone filter to that image.

**Deliverables:**

1. Create a Google Doc named "Lastname Duotone Project". Set the font to Times New Roman 12.

2. Place a header on the document in the following format:

Firstname Lastname Class Name Period, Term, Year Project Name
--

3. In the body of the document, place the following:

- Screenshot of the original image you selected to apply the Duotone program.
- Screenshot of the resulting Duotone image from your Code
- Screenshot of the color patch for your color1 including the *red*, *green*, and *blue* values.
- Screenshot of the color patch for your color2 including the *red*, *green*, and *blue* values.
- (You can use any HTML Color Picker to create the color patch)
- A short paragraph outlining your process for creating the DuoTone Code.

4. Create a Table in the document and paste your code from DuoTone.java in the table. Make sure the code is in `consolas` font.

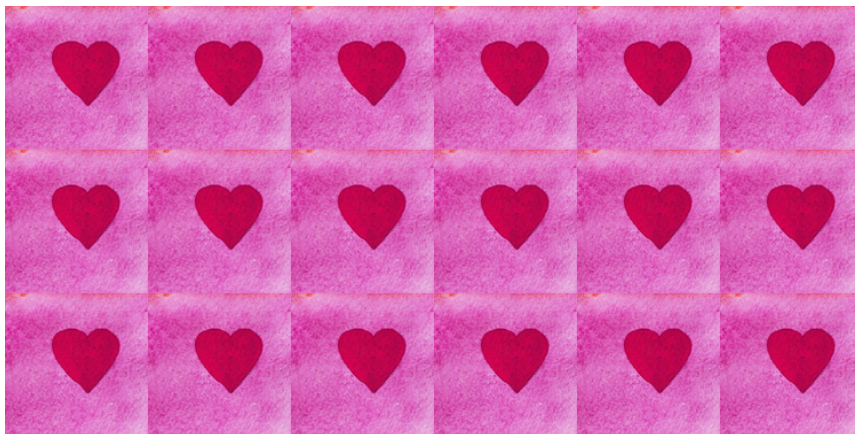
5. Submit the Google Doc "Lastname Duotone Project" to the Google Classroom.

## Project 02: Rectangular tile of an image (by Kevin Wayne)

Write a program to create a *rectangular tile of an image*. A rectangular tile of an image is created by repeating copies of an image in a rectangular grid, with a specified number of columns and rows. For example, consider the following image:



Here is a 6-by-3 rectangular tile:



## Requirements:

1. Write a program `Tile.java` that takes three command-line arguments (the file name of the image, the number of columns, and the number of rows) and displays a rectangular tile of the image, as described above. Using the [Picture API](#) data type, organize your program using the following API:

```
public class Tile {  
  
    // Returns a cols-by-rows tiling of the specified image.  
    public static Picture tile(Picture picture, int cols, int rows) {  
        // Complete Code Here  
  
    }  
  
    public static void main(String[] args) {  
        // Define Picture Object, number of rows, number of columns  
        String path = "tile.png";  
        int width = 6;  
        int height = 3;  
  
        // Call the tile method and show the resulting image  
  
    }  
}
```

2. Run the program with three different input tile images with different width and height values. You may use the tile images provided on the Google Classroom in the .zip file or you may create your own tiles. (Create images less than 128 by 128 pixels).



### Example Program Executions:

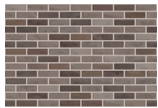
princeton.png with width of 1 and height of 1



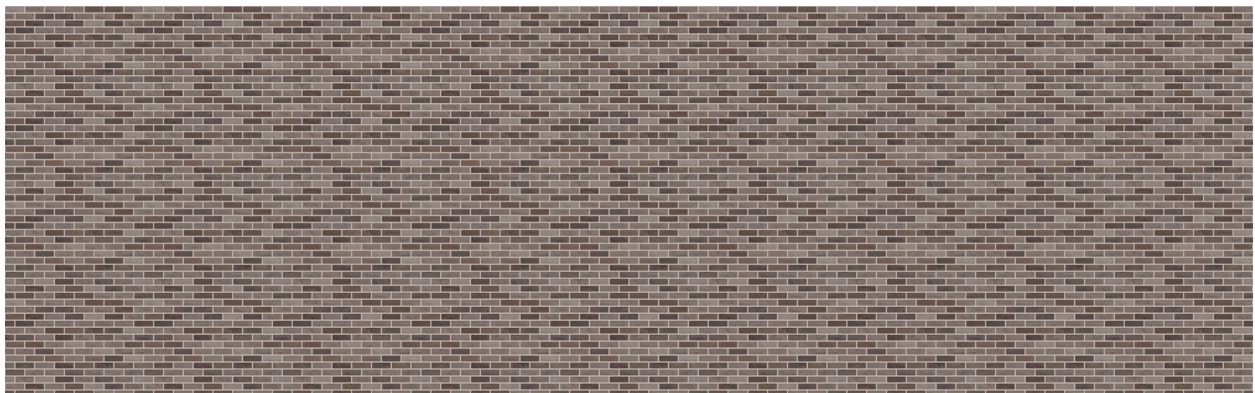
princeton.png with width of 5 and height of 2



bricks.png with width of 1 and height of 1



bricks.png with width of 9 and height of 4



**Deliverables:**

1. Create a Google Doc named “Lastname Tile Project”. Set the font to Times New Roman 12.

2. Place a header on the document in the following format:

Firstname Lastname Class Name Period, Term, Year Project Name
--

3. In the body of the document, place the following:

- Screenshot of the original tile images (at least 3 different images)
- Screenshot of the resulting tile images (at least 3 examples)
- Label each image set with the width and height.
- A short paragraph outlining your process for creating the Tile Code.

4. Create a Table in the document and paste your code from Tile.java in the table. Make sure the code is in consolas font.

5. Submit the Google Doc “Lastname Tile Project” to the Google Classroom.