Saket Bakshi
Mr. Caces
AP Computer Science A
Picture Lab
5 May 2019

A1

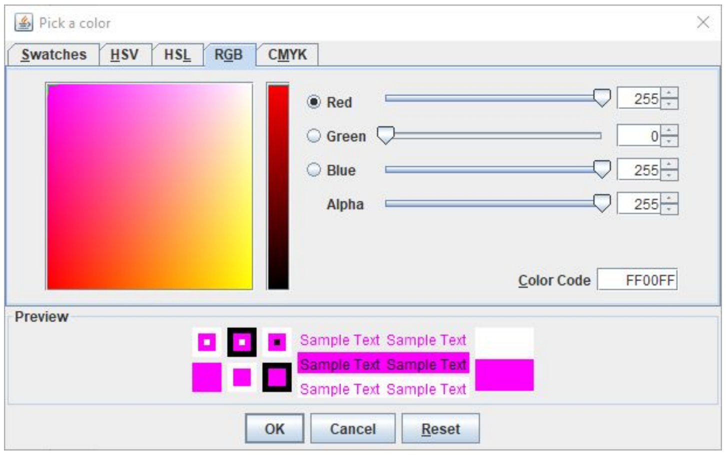
Questions

1. How many bits does it take to represent the values from 0 to 255?
    a. 8 bits
2. How many bytes does it take to represent a color in the RBG color model?
    a. 3 bytes
3. How many pixels are in a picture that is 640 pixels wide and 480 pixels high?
    a. 307200 pixels

A2

Questions

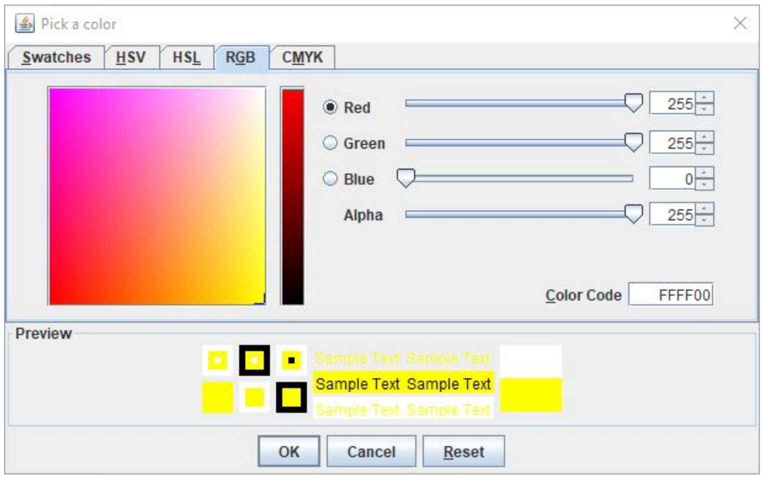
1. How can you make pink?
    a. R: 255
       G: 0



Red 255
Green 0
Blue 255
Alpha 255
Color Code FF00FF

       B: 255

2. How can you make yellow?
    a. R: 255
       G: 255

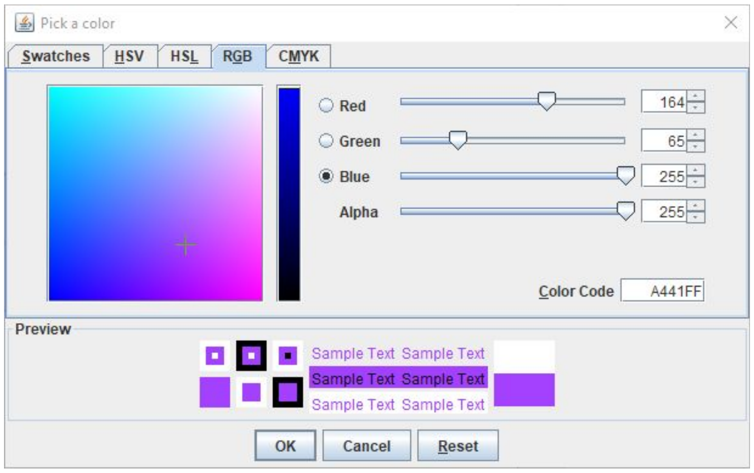


Red 255
Green 255
Blue 0
Alpha 255
Color Code FFFF00

       B: 0

3. How can you make purple?

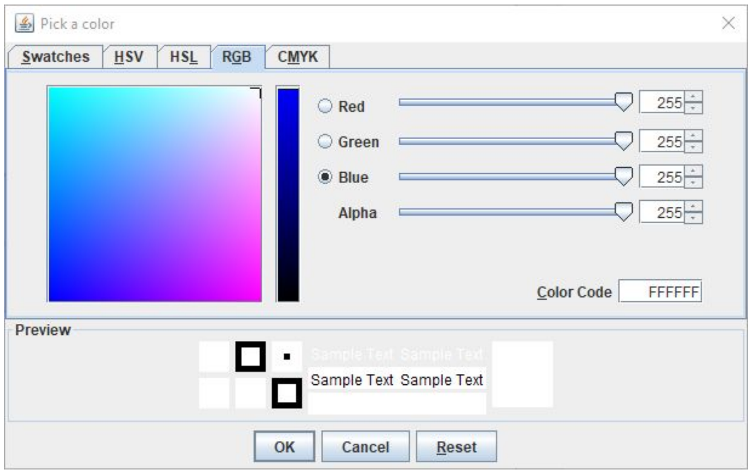
   a. R: 164
      G: 65



Red 164
Green 65
Blue 255
Alpha 255
Color Code A441FF

      B: 255

4. How can you make white?
   a. R: 255
      G: 255

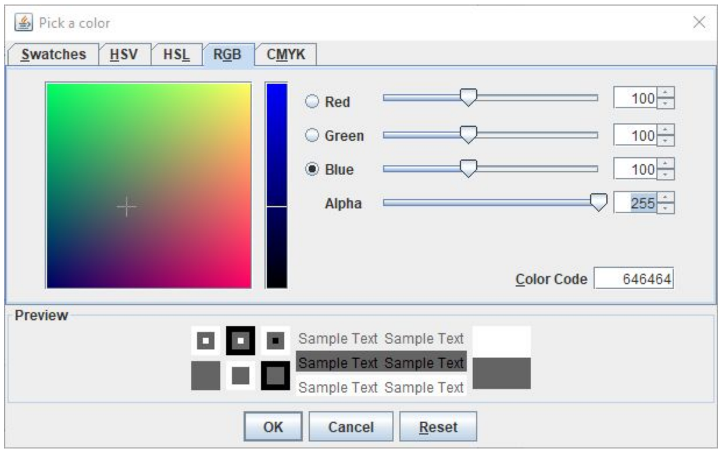


Red 255
Green 255
Blue 255
Alpha 255
Color Code FFFFFF

      B: 255

5. How can you make dark gray?

a. R: 100
   G: 100



Red 100
Green 100
[● Blue] 100
Alpha 255

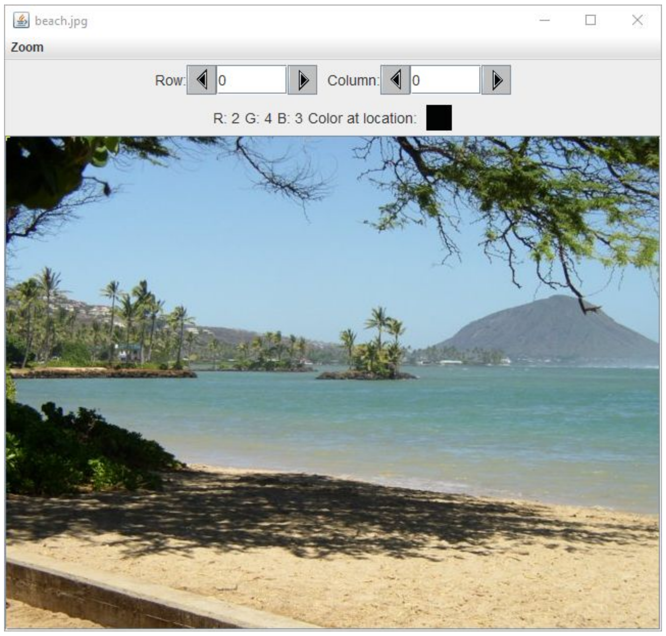
Color Code 646464

   B: 100

A3

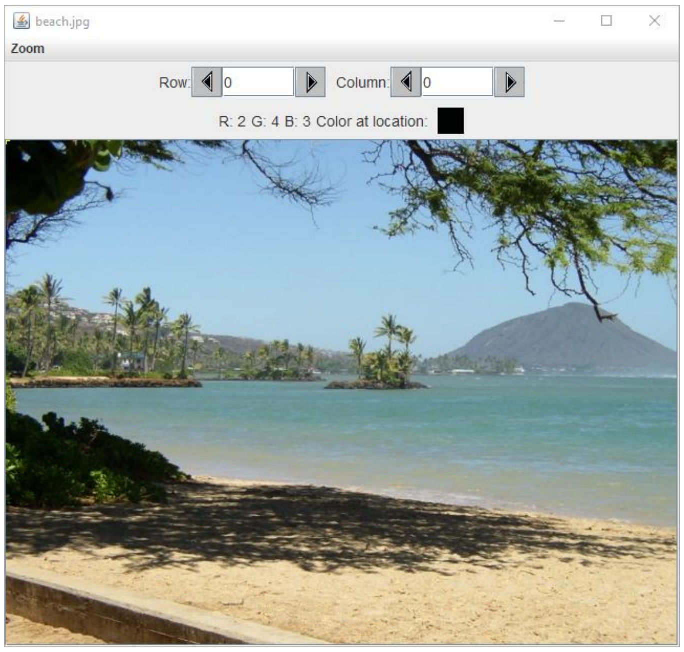
Questions

1. What is the row index for the top left corner of the picture?



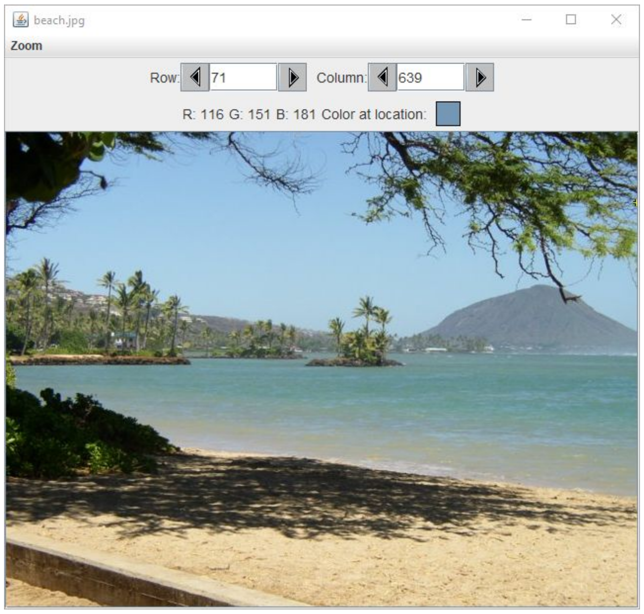
   a. 0

2. What is the column index for the top left corner of the picture?



   a. 0

3. The width of this picture is 640. What is the right most column index?

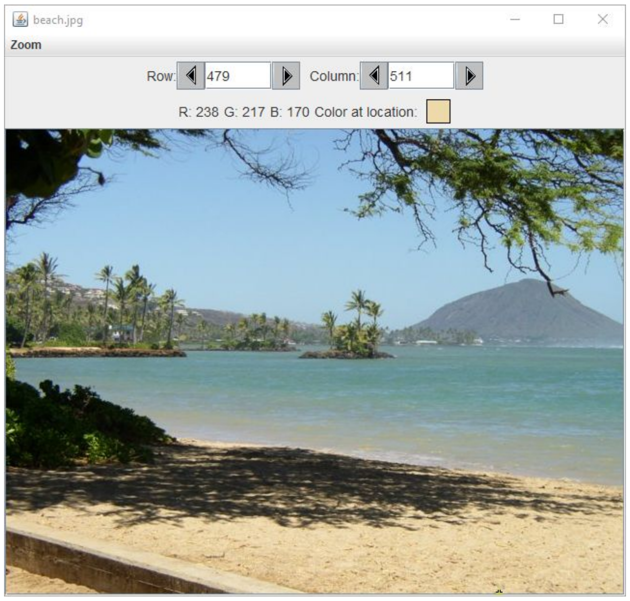


Row: 71 Column: 639

R: 116 G: 151 B: 181 Color at location:

   a. 639

4. The height of this picture is 480. What is the bottom most row index?

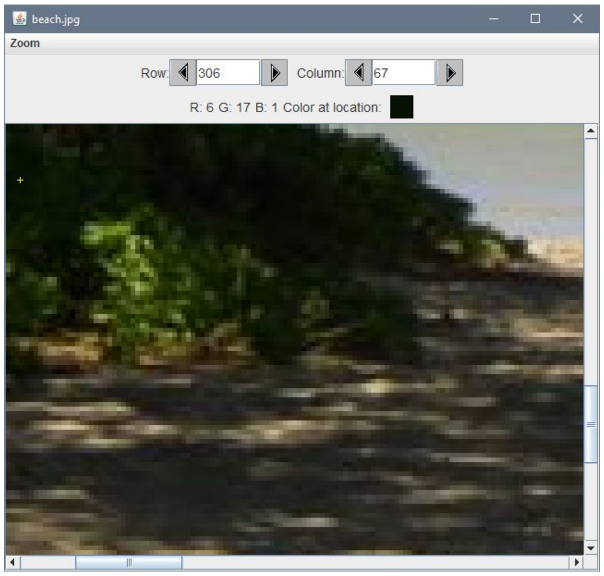


Row: 479 Column: 511

R: 238 G: 217 B: 170 Color at location:

   a. 479

5. Does the row index increase from left to right or top to bottom?
   a. Top to bottom
6. Does the column index increase from left to right or top to bottom?
   a. Left to right
7. Set the zoom to 500%. Can you see squares of color? This is called *pixelation*. Pixelation means displaying a picture so magnified that the individual pixels look like small squares.



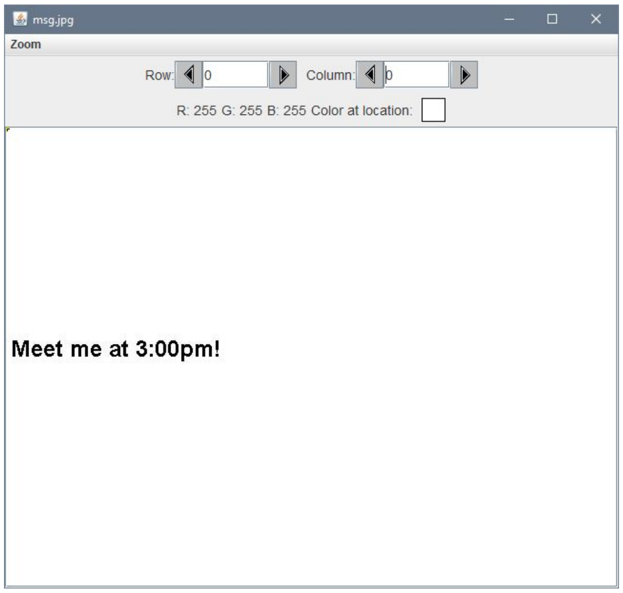
Row: 306 Column: 67

R: 6 G: 17 B: 1 Color at location:

a. Yes

Exercises

1. Modify the main method in the PictureExplorer class to create and explore a different picture from the images folder.
    a. public static void main(String args[])
      {
        Picture pix = new Picture("msg.jpg");
        pix.explore();
      }



Row: 0 Column: 0

R: 255 G: 255 B: 255 Color at location:

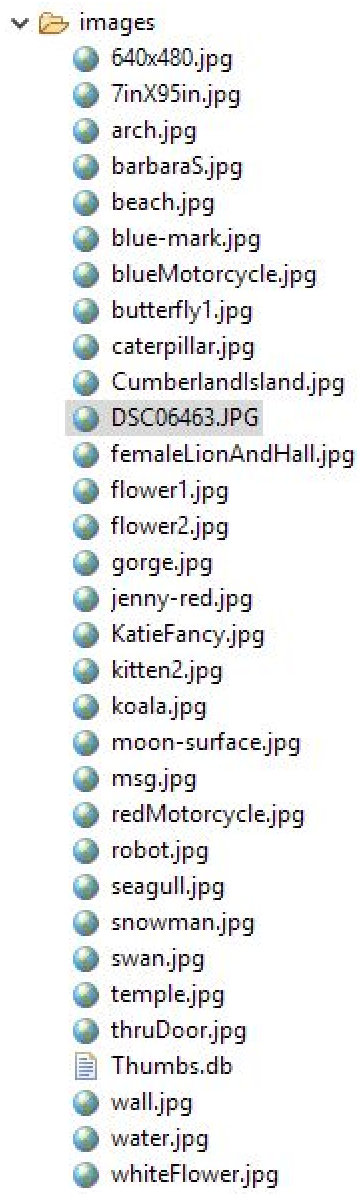
Meet me at 3:00pm!

    b.

2. Add a picture to the images folder and then create and explore that picture in the main method. If the picture is very large (for instance, one from a digital camera), you can scale it using the scale method in the Picture class.
   For example, you can make a new picture(“smallMyPicture.jpg”in the images folder) one-fourth the size of the original(“myPicture.jpg”)using:

```
Picture p = new Picture("myPicture.jpg");
Picture smallP = p.scale(0.25,0.25);
smallP.write("smallMyPicture.jpg");
```

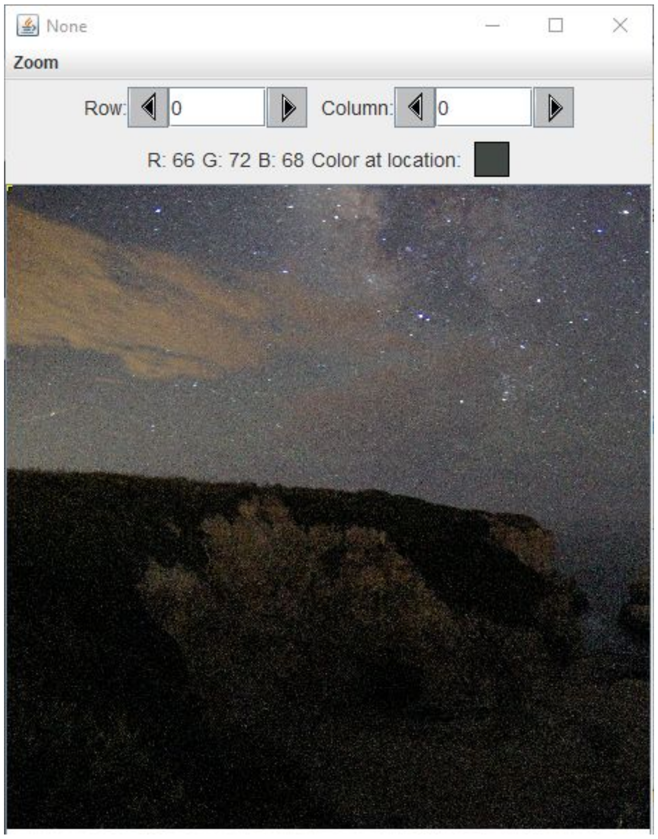
a. Added image “DSC06463.JPG” to images folder.



```
images
    640x480.jpg
    7inX95in.jpg
    arch.jpg
    barbaraS.jpg
    beach.jpg
    blue-mark.jpg
    blueMotorcycle.jpg
    butterfly1.jpg
    caterpillar.jpg
    CumberlandIsland.jpg
    DSC06463.JPG
    femaleLionAndHall.jpg
    flower1.jpg
    flower2.jpg
    gorge.jpg
    jenny-red.jpg
    KatieFancy.jpg
    kitten2.jpg
    koala.jpg
    moon-surface.jpg
    msg.jpg
    redMotorcycle.jpg
    robot.jpg
    seagull.jpg
    snowman.jpg
    swan.jpg
    temple.jpg
    thruDoor.jpg
    Thumbs.db
    wall.jpg
    water.jpg
    whiteFlower.jpg
```

New main method:

```
public static void main( String args[])
 {
   Picture pix = new Picture("DSC06463.JPG");
   Picture smallPix = pix.scale(0.15,  0.15);
   smallPix.explore();
 }
```



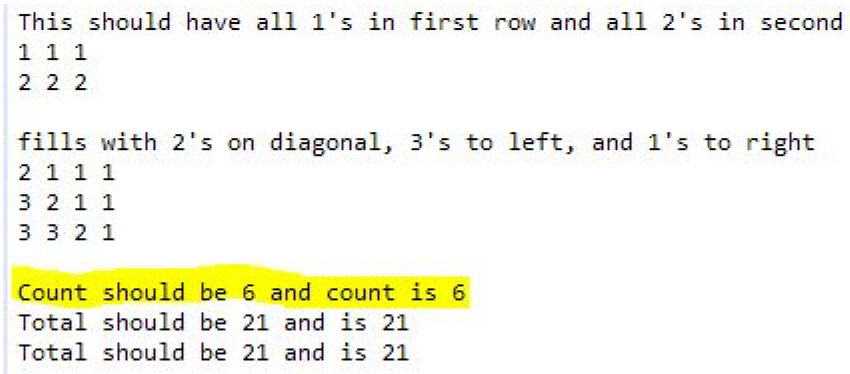
R: 66 G: 72 B: 68 Color at location:

A4

Exercises

1. Write a getCount method in the IntArrayWorker class that returns the count of the number of times a passed integer value is found in the matrix. There is already a method to test this in IntArrayWorkerTester. Just uncomment the method testGetCount() and the call to it in the main method of IntArrayWorkerTester.
    a. public int getCount(int number)

```
{
        int count = 0;
        for(int a = 0; a < matrix.length; a++)
        {
                for(int b = 0; b < matrix[0].length; b++)
                {
                        if(number == matrix[a][b])
                                Count++;
                }
        }
        return count;
```



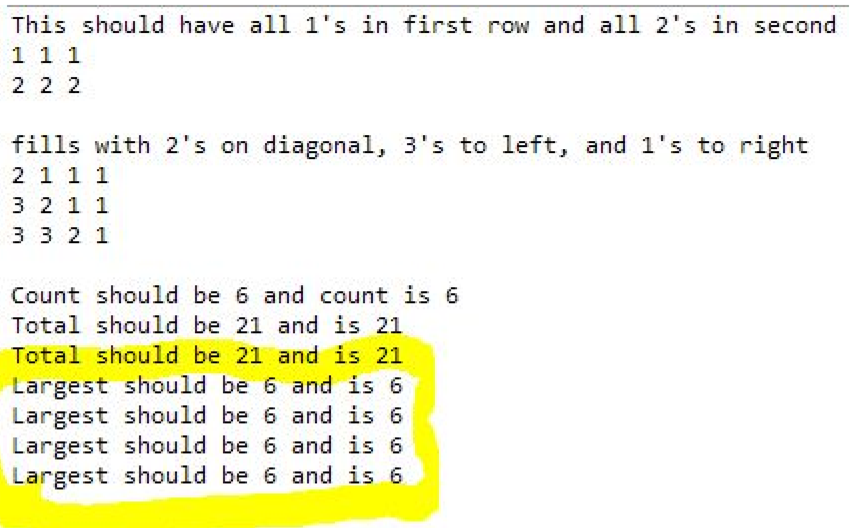
```
This should have all 1's in first row and all 2's in second
1 1 1
2 2 2

fills with 2's on diagonal, 3's to left, and 1's to right
2 1 1 1
3 2 1 1
3 3 2 1

Count should be 6 and count is 6
Total should be 21 and is 21
Total should be 21 and is 21
```

```
}
```

2. Write a getLargest method in the IntArrayWorker class that returns the largest value in the matrix. There is already a method to test this in IntArrayWorkerTester. Just uncomment the method testGetLargest() and the call to it in the main method of IntArrayWorkerTester.
    a. public int getLargest()

```
{
        int max = 0;
        for(int a = 0; a < matrix.length; a++)
        {
                for(int b = 0; b < matrix[0].length; b++)
                {
                        if(max < matrix[a][b])
                                max = matrix[a][b];
                }
        }
```

return max;

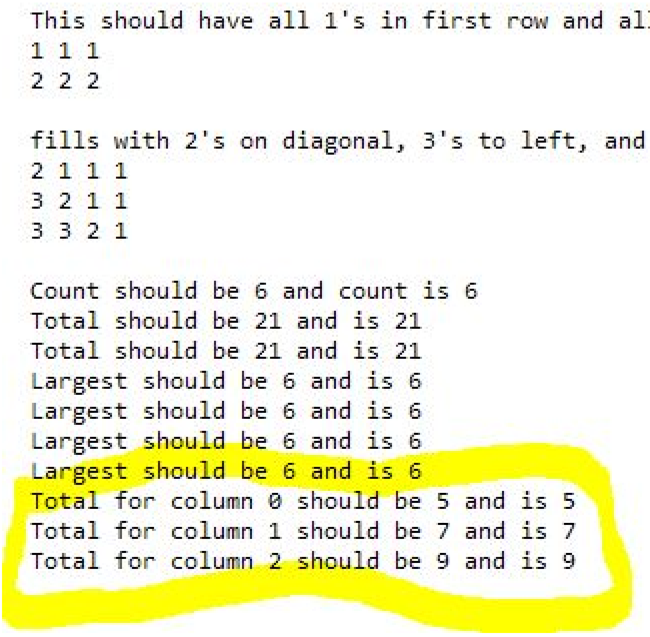


```
This should have all 1's in first row and all 2's in second
1 1 1
2 2 2

fills with 2's on diagonal, 3's to left, and 1's to right
2 1 1 1
3 2 1 1
3 3 2 1

Count should be 6 and count is 6
Total should be 21 and is 21
Total should be 21 and is 21
Largest should be 6 and is 6
Largest should be 6 and is 6
Largest should be 6 and is 6
Largest should be 6 and is 6
```

}

3. Write a getColTotal method in the IntArrayWorker class that returns the total of all integers in a specified column. There is already a method to test this in IntArrayWorkerTester. Just uncomment the method testGetColTotal() and the call to it in the main method of IntArrayWorkerTester.
    a. public int getColTotal(int b)

```
{
        int total = 0;
        for(int a = 0; a < matrix.length; a++)
                total += matrix[a][b];
        return total;
```



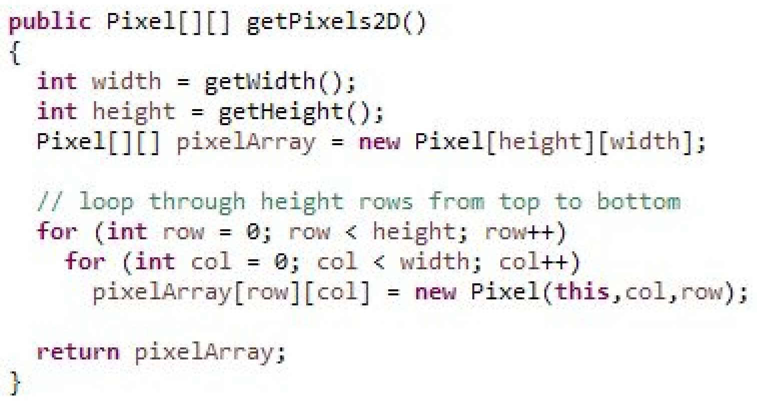
```
This should have all 1's in first row and all 2's in second
1 1 1
2 2 2

fills with 2's on diagonal, 3's to left, and 1's to right
2 1 1 1
3 2 1 1
3 3 2 1

Count should be 6 and count is 6
Total should be 21 and is 21
Total should be 21 and is 21
Largest should be 6 and is 6
Largest should be 6 and is 6
Largest should be 6 and is 6
Largest should be 6 and is 6
Total for column 0 should be 5 and is 5
Total for column 1 should be 7 and is 7
Total for column 2 should be 9 and is 9
```

}

A5

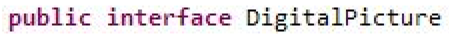
Questions

1. Open Picture.java and look for the method getPixels2D. Is it there?
    a. No
2. Open SimplePicture.java and look for the method getPixels2D. Is it there?
    a. Yes



```
public Pixel[][] getPixels2D()
{
  int width = getWidth();
  int height = getHeight();
  Pixel[][] pixelArray = new Pixel[height][width];

  // loop through height rows from top to bottom
  for (int row = 0; row < height; row++)
    for (int col = 0; col < width; col++)
      pixelArray[row][col] = new Pixel(this,col,row);

  return pixelArray;
}
```

3. Does the following code compile?
    DigitalPicture p = new DigitalPicture();
    a. No, you cannot initialize an object from an interface.

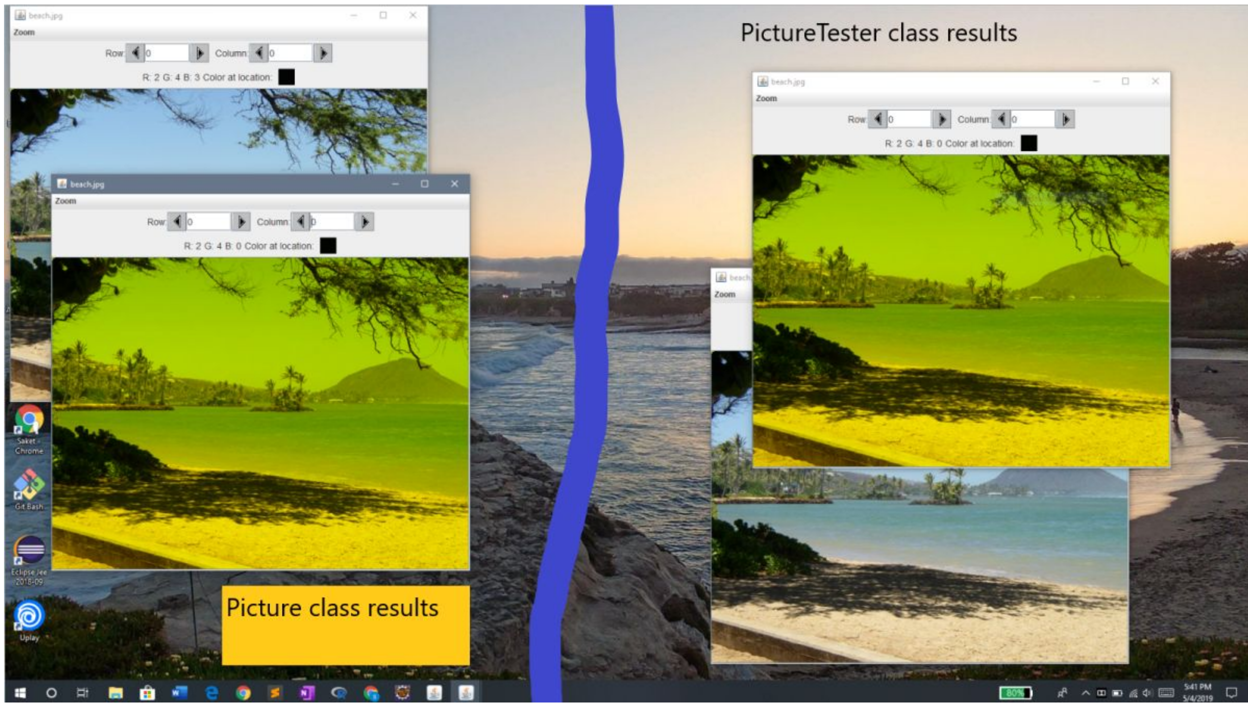


```
public interface DigitalPicture
```

4. Assuming that a no-argument constructor exists for SimplePicture, would the following code compile?
    DigitalPicture p = new SimplePicture();
    a. Yes
5. Assuming that a no-argument constructor exists for Picture, does the following code compile?
    DigitalPicture p = new Picture();
    a. Yes
6. Assuming that a no-argument constructor exists for Picture, does the following code compile?
    SimplePicture p = new Picture();
    a. Yes
7. Assuming that a no-argument constructor exists for SimplePicture, does the following code compile?
    Picture p = new SimplePicture();
    a. No

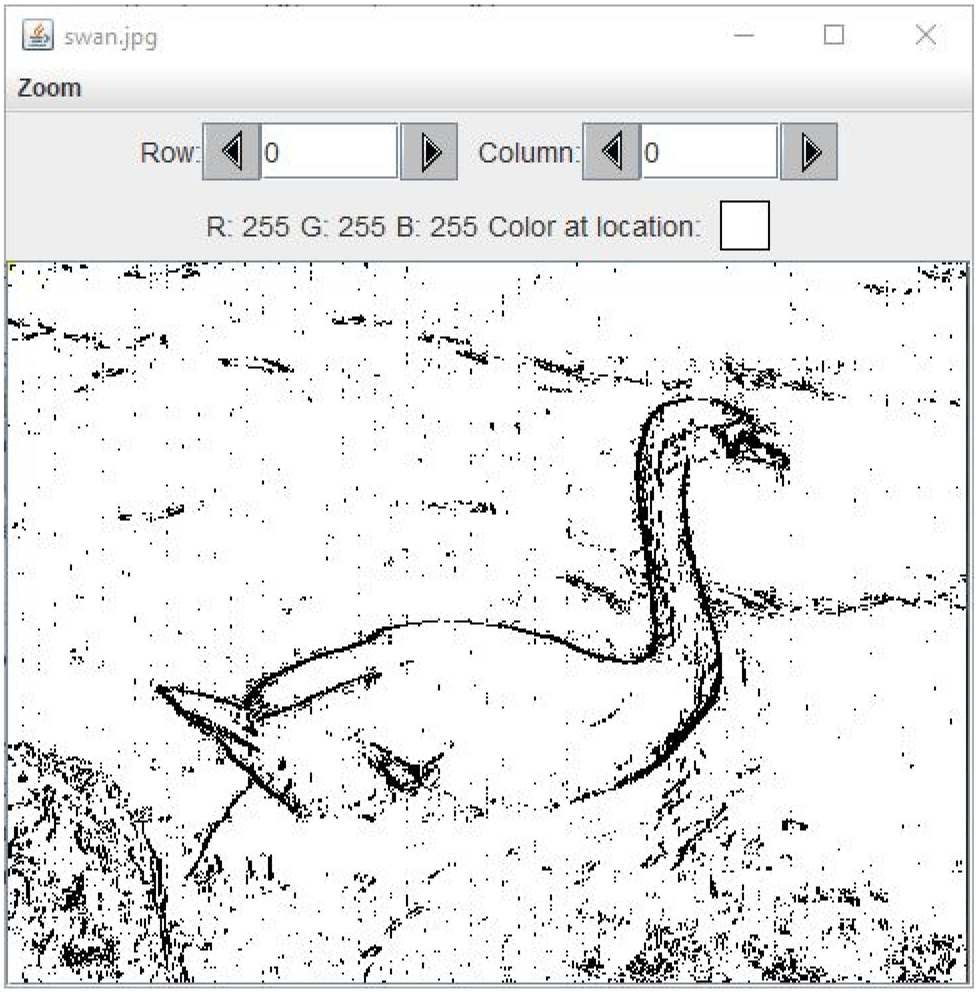
Exercises

1. Open PictureTester.java and run its main method. You should get the same results as running the main method in the Picture class. The PictureTester class contains class (static) methods for testing the methods that are in the Picture class.

a. 
PictureTester class results
Picture class results
beach.jpg
Zoom
Row: 0 Column: 0
R: 2 G: 4 B: 3 Color at location:
R: 2 G: 4 B: 0 Color at location:

2. Uncomment the appropriate test method in the main method of PictureTester to test any of the other methods in Picture.java. You can comment out the tests you don't want to run. You can also add new test methods to PictureTester to test any methods you create in the Picture class.

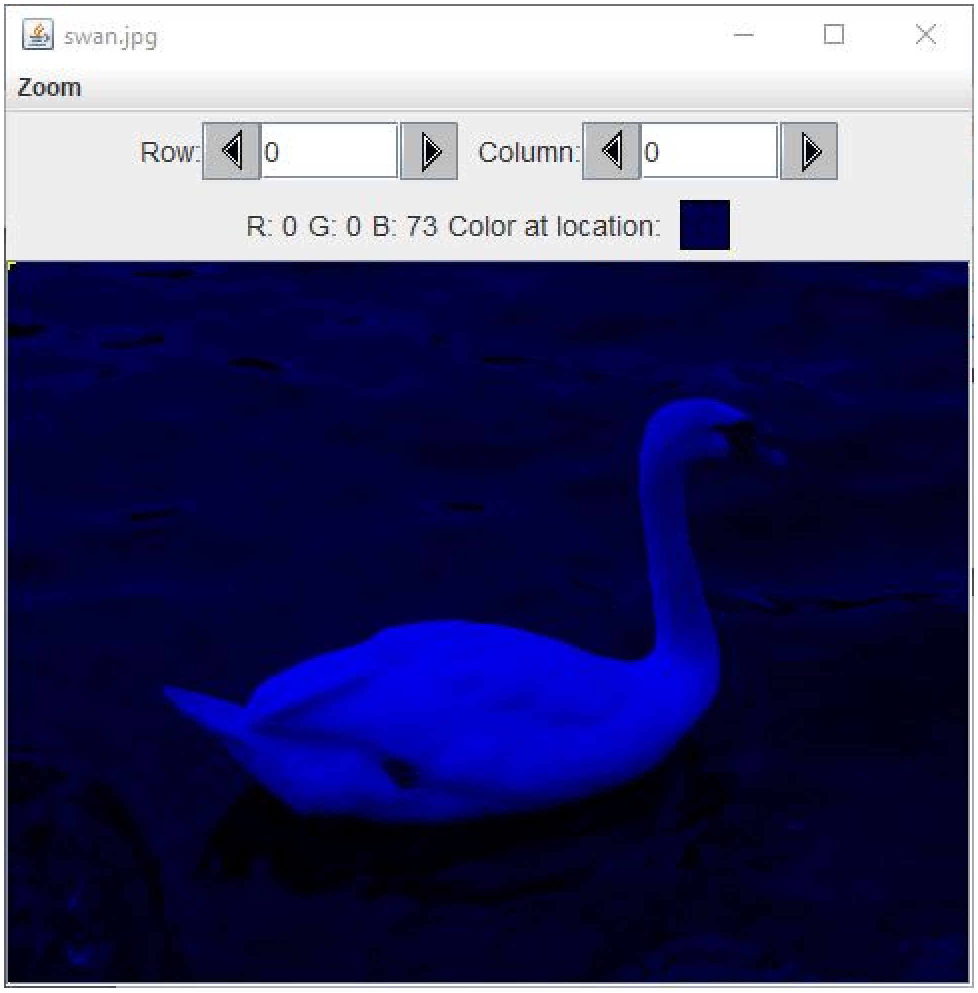
a. Tested testEdgeDetection();



R: 255 G: 255 B: 255 Color at location:

3. Using the zeroBlue method as a starting point, write the method keepOnlyBlue that will keep only the blue values, that is, it will set the red and green values to zero. Create a class (static) method to test this new method in the class PictureTester. Be sure to call the new test method in the main method in PictureTester.
    a. public void keepOnlyBlue()

```
{
        Pixel[][] pixels = this.getPixels2D();
        for(Pixel[] rowArray : pixels)
        {
                for(Pixel pixelObj : rowArray)
                {
                        pixelObj.setGreen(0);
                        pixelObj.setRed(0);
                }
        }
}
```

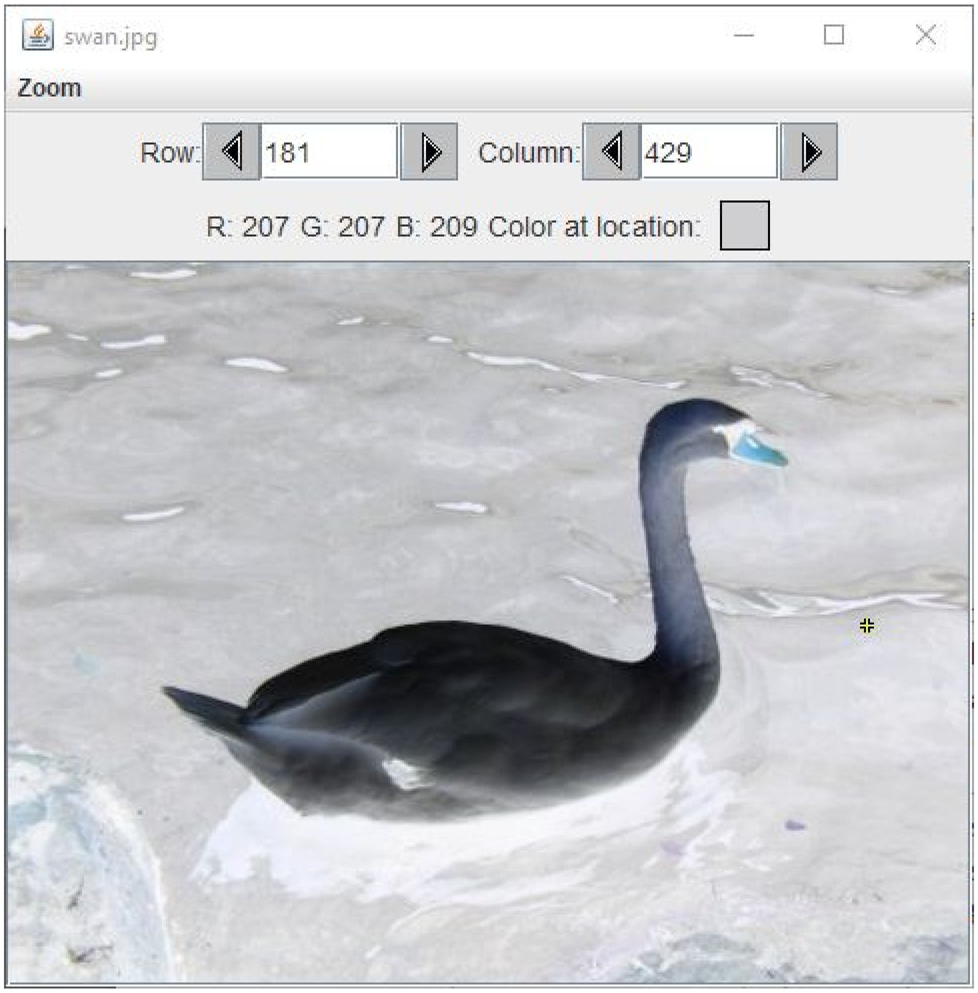


R: 0 G: 0 B: 73 Color at location:

4. Write the negate method to negate all the pixels in a picture. To negate a picture, set the red value to 255 minus the current red value, the green value to 255 minus the current green value and the blue value to 255 minus the current blue value. Create a class (static) method to test this new method in the class PictureTester. Be sure to call the new test method in the main method in PictureTester.
    a. public void negate()

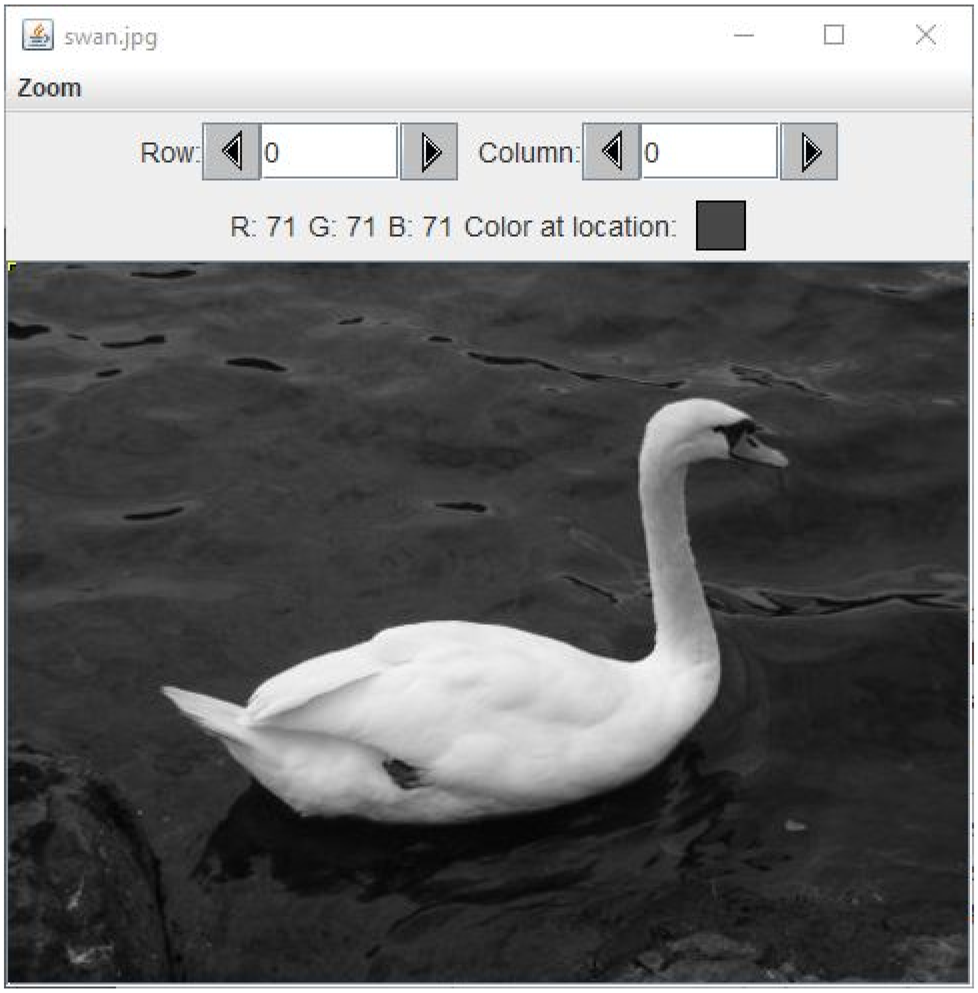
```
        {
                Pixel[][] pixels = this.getPixels2D();
                for(Pixel[] rowArray : pixels)
                {
                        for(Pixel pixelObj : rowArray)
                        {
                                pixelObj.setGreen(255 - pixelObj.getGreen());
                                pixelObj.setRed(255 - pixelObj.getRed());
                                pixelObj.setBlue(255 - pixelObj.getBlue());
                        }
                }
```

}



5. Write the grayscale method to turn the picture into shades of gray. Set the red, green, and blue values to the average of the current red, green, and blue values (add all three values and divide by 3). Create a class (static) method to test this new method in the class PictureTester. Be sure to call the new test method in the main method in PictureTester.
    a. public void grayscale()

```
        {
                Pixel[][] pixels = this.getPixels2D();
                for(Pixel[] rowArray : pixels)
                {
                        for(Pixel pixelObj : rowArray)
                        {
                                int average = pixelObj.getBlue() +
pixelObj.getGreen() + pixelObj.getRed();
                                average = average / 3;
                                pixelObj.setBlue(average);
                                pixelObj.setGreen(average);
                        pixelObj.setRed(average);
                    }
                }
            }
```

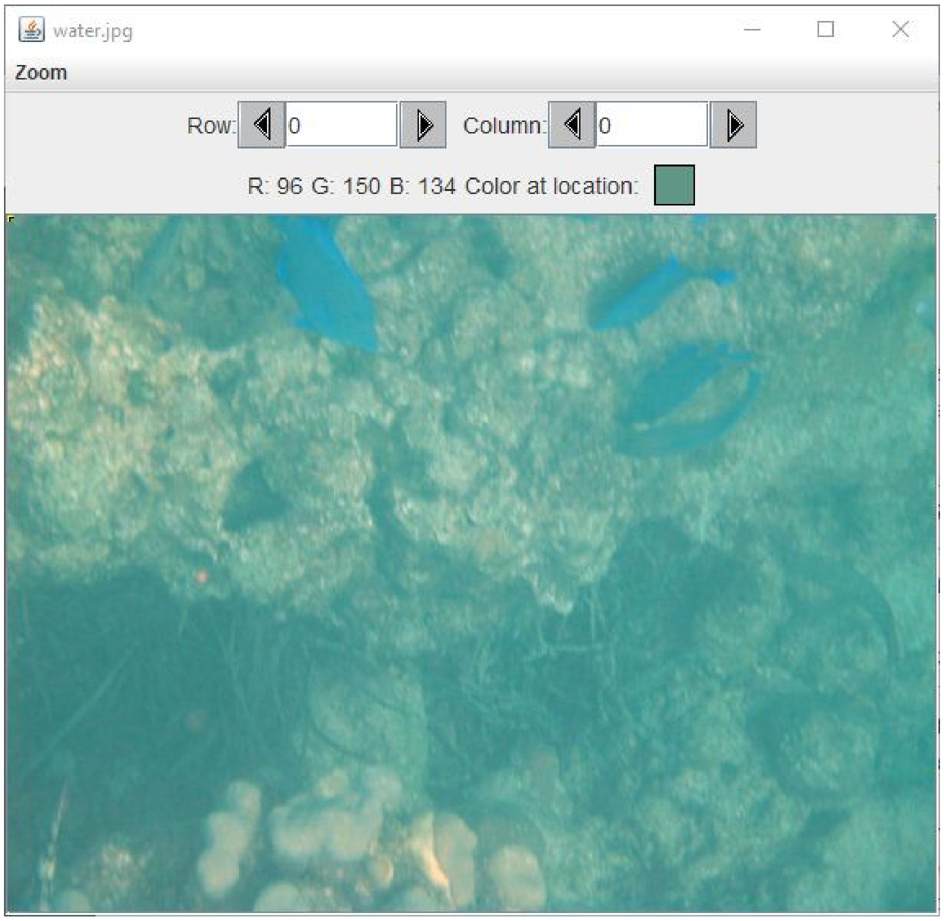


R: 71 G: 71 B: 71 Color at location:

6. Challenge — Explore the “water.jpg” picture in the images folder. Write a method fixUnderwater() to modify the pixel colors to make the fish easier to see. Create a class (static) method to test this new method in the class PictureTester. Be sure to call the new test method in the main method in PictureTester.
    a. public void fixUnderwater()

```
            {
                Pixel[][] pixels = this.getPixels2D();
                for(Pixel[] rowArray : pixels)
                {
                    for(Pixel pixelObj : rowArray)
                    {
                        pixelObj.setRed(pixelObj.getRed() * 3);
                    }
                }
```

```
}
```



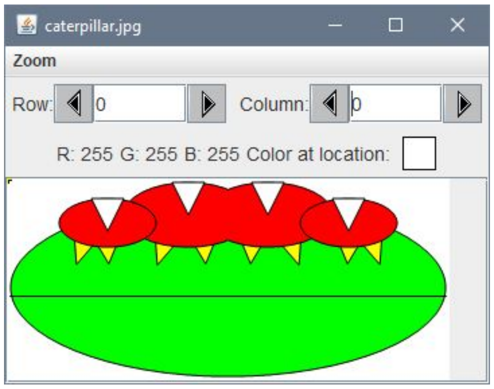
R: 96 G: 150 B: 134 Color at location:

A6

Exercises

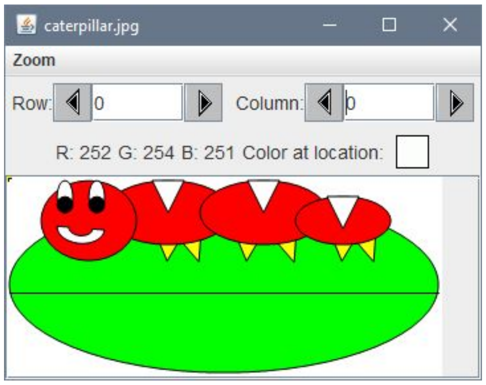
1. Write the method mirrorVerticalRightToLeft that mirrors a picture around a mirror placed vertically from right to left. Hint: you can copy the body of mirrorVertical and only change one line in the body of the method to accomplish this. Write a class (static) test method called testMirrorVertical in PictureTester to test this new method and call it in the main method.
   a. public void mirrorVerticalRightToLeft()

```
      {
            Pixel[][] pixels = this.getPixels2D();
        Pixel leftPixel = null;
        Pixel rightPixel = null;
        int width = pixels[0].length;
        for (int row = 0; row < pixels.length; row++)
        {
          for (int col = 0; col < width / 2; col++)
          {
            leftPixel = pixels[row][col];
            rightPixel = pixels[row][width - 1 - col];
            leftPixel.setColor(rightPixel.getColor());
          }
        }
      }
```



Row: 0 Column: 0

R: 255 G: 255 B: 255 Color at location:



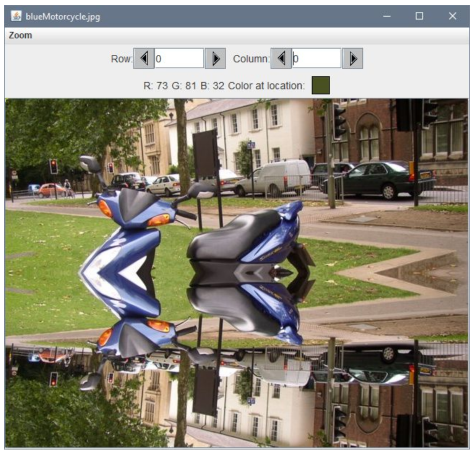
Row: 0 Column: 0

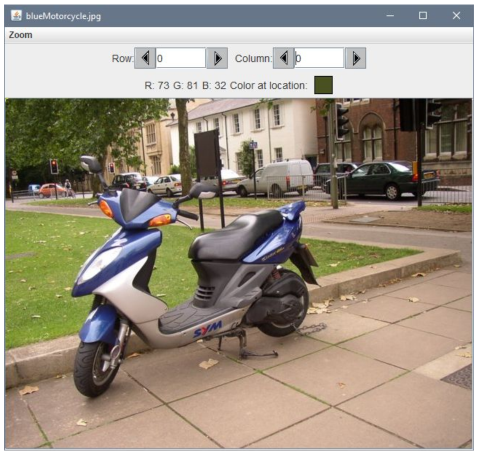
R: 252 G: 254 B: 251 Color at location:

2. Write the method mirrorHorizontal that mirrors a picture around a mirror placed horizontally at the middle of the height of the picture. Mirror from top to bottom as shown in the pictures below (Figure 8). Write a class (static) test method in PictureTester to test this new method and call it in the main method.
   a. public void mirrorHorizontal()

```
      {
            Pixel[][] pixels = this.getPixels2D();
        Pixel topPixel = null;
        Pixel bottomPixel = null;
        int height = pixels.length;
```

```
    for (int row = 0; row < height / 2; row++)
    {
      for (int col = 0; col < pixels[0].length; col++)
      {
        topPixel = pixels[row][col];
        bottomPixel = pixels[height - 1 - row][col];
        bottomPixel.setColor(topPixel.getColor());
      }
    }
  }
```

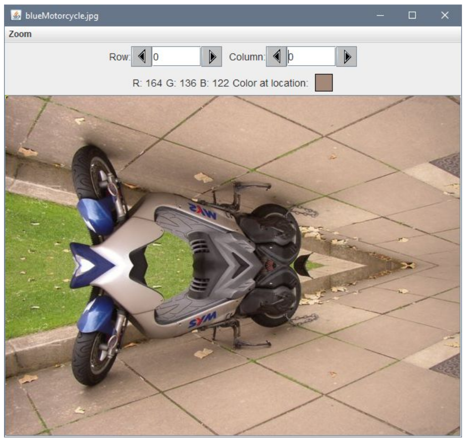


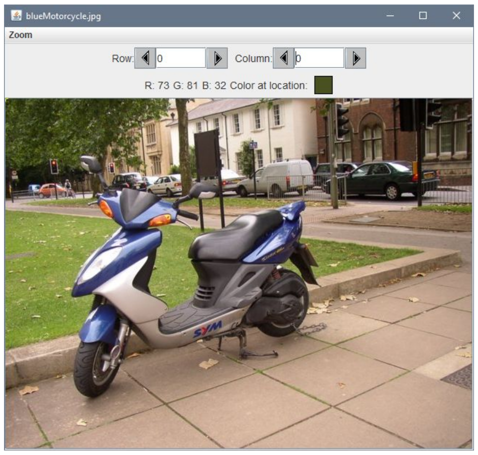


3. Write the method mirrorHorizontalBotToTop that mirrors the picture around a mirror placed horizontally from bottom to top. Hint: you can copy the body of mirrorHorizontal and only change one line to accomplish this. Write a class (static) test method in PictureTester to test this new method and call it in the main method.
    a. public void mirrorHorizontalBotToTop()

```
  {
          Pixel[][] pixels = this.getPixels2D();
    Pixel topPixel = null;
    Pixel bottomPixel = null;
    int height = pixels.length;
    for (int row = 0; row < height / 2; row++)
    {
      for (int col = 0; col < pixels[0].length; col++)
      {
        topPixel = pixels[row][col];
        bottomPixel = pixels[height - 1 - row][col];
        topPixel.setColor(bottomPixel.getColor());
      }
    }
```

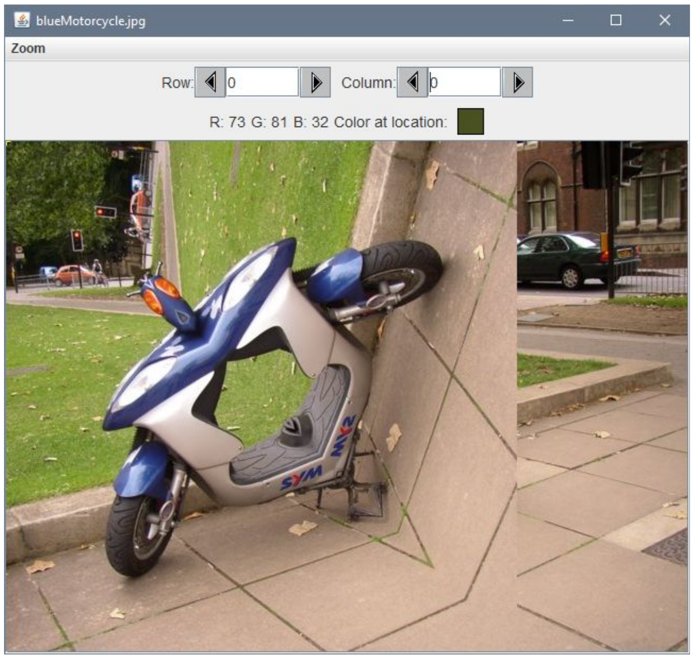
```
}
```





4. Challenge — Work in groups to figure out the algorithm for the method mirrorDiagonal that mirrors just a square part of the picture from bottom left to top right around a mirror placed on the diagonal line (the diagonal line is the one where the row index equals the column index). This will copy the triangular area to the left and below the diagonal line as shown below. This is like folding a square piece of paper from the bottom left to the top right, painting just the bottom left triangle and then (while the paint is still wet) folding the paper up to the top right again. The paint would be copied from the bottom left to the top right as shown in the pictures below (Figure 9). Write a class (static) test method in PictureTester to test this new method and call it in the main method.
   a. public void mirrorDiagonal()

```
        {
                Pixel[][] pixels = this.getPixels2D();
                Pixel leftPixel = null;
                Pixel rightPixel = null;

                int max = pixels.length;
                if (pixels[0].length < max)
                        max = pixels[0].length;

                for (int row = 1; row < max; row++)
                {
                        for (int col = 0; col < row; col++)
                        {
                                leftPixel = pixels[row][col];
                                rightPixel = pixels[col][row];
                                rightPixel.setColor(leftPixel.getColor());
                        }
                }
        }
```



R: 73 G: 81 B: 32 Color at location:

A7

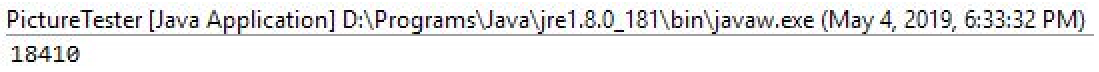
Questions

1. How many times would the body of this nested for loop execute?
   for (int row = 7; row < 17; row++)
       for (int col = 6; col < 15; col++)
   a. 90
2. How many times would the body of this nested for loop execute?
   for (int row = 5; row <= 11; row++)
       for (int col = 3; col <= 18; col++)
   a. 112

Exercises

1. Check the calculation of the number of times the body of the nested loop executes by adding an integer count variable to the mirrorTemple method that starts out at 0 and increments inside the body of the loop. Print the value of count after the nested loop ends.
   a. public void mirrorTemple()

```
 {
   int mirrorPoint = 276;
   Pixel leftPixel = null;
   Pixel rightPixel = null;
   int count = 0;
   Pixel[][] pixels = this.getPixels2D();
   // loop through the rows
   for (int row = 27; row < 97; row++)
   {
     // loop from 13 to just before the mirror point
     for (int col = 13; col < mirrorPoint; col++)
     {

       leftPixel = pixels[row][col];
       rightPixel = pixels[row]
                        [mirrorPoint - col + mirrorPoint];
       rightPixel.setColor(leftPixel.getColor());
       count++;
     }
   }
   System.out.print(count);
 }
```

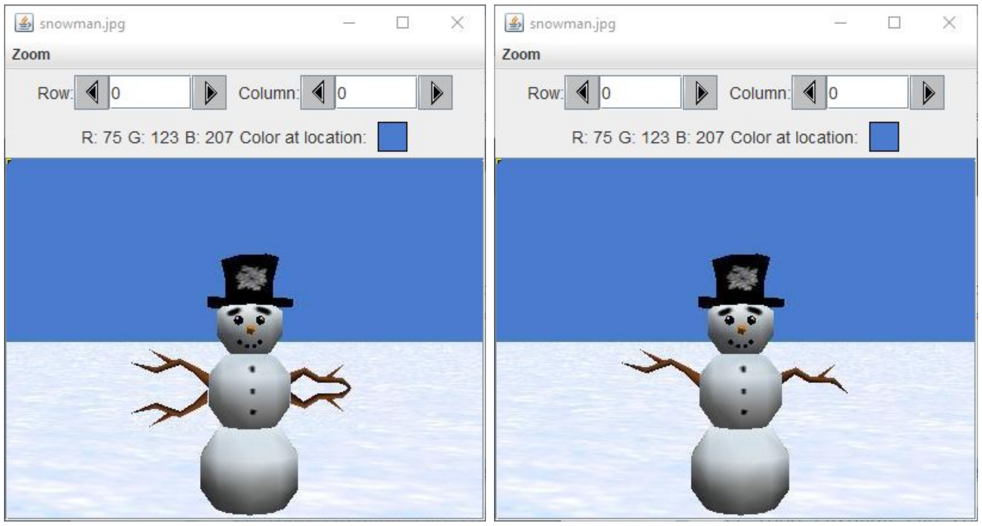


```
PictureTester [Java Application] D:\Programs\Java\jre1.8.0_181\bin\javaw.exe (May 4, 2019, 6:33:32 PM)
18410
```

2. Write the method mirrorArms to mirror the arms on the snowman (“snowman.jpg”) to make a snowman with 4 arms. Write a class (static) test method in PictureTester to test this new method and call it in the main method.
    a. public void mirrorArms()

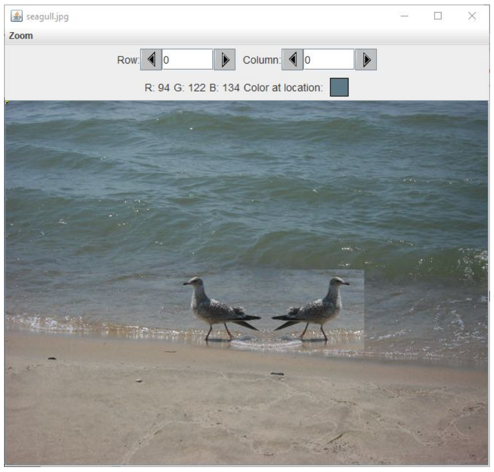
```
        {
                Pixel topPixel = null;
                Pixel botPixel = null;
                Pixel[][] pixels = this.getPixels2D();

                // loop through the rows
                for (int row = 155; row < 191; row++)
                {
                // loop through the columns
                for (int col = 98; col < 169; col++)
                {
                topPixel = pixels[row][col];
                botPixel = pixels[191-row+191][col];
                botPixel.setColor(topPixel.getColor());
                }
                }

                // loop through the rows
                for (int row = 155; row < 191; row++)
                {
                // loop through the columns
                for (int col = 238; col < 296; col++)
                {
                topPixel = pixels[row][col];
                botPixel = pixels[191-row+191][col];
                botPixel.setColor(topPixel.getColor());
                }
                }
```

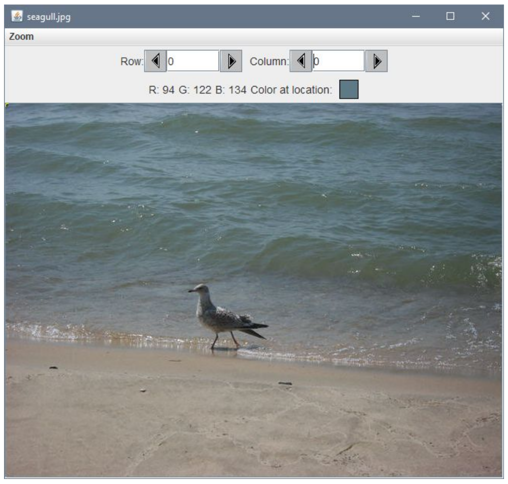
```
}
```


snowman.jpg
Zoom
Row: 0
Column: 0
R: 75 G: 123 B: 207 Color at location:
snowman.jpg
Zoom
Row: 0
Column: 0
R: 75 G: 123 B: 207 Color at location:

3. Write the method mirrorGull to mirror the seagull (“seagull.jpg”) to the right so that there are two seagulls on the beach near each other. Write a class (static) test method in PictureTester to test this new method and call it in the main method.
   a. public void mirrorGull()

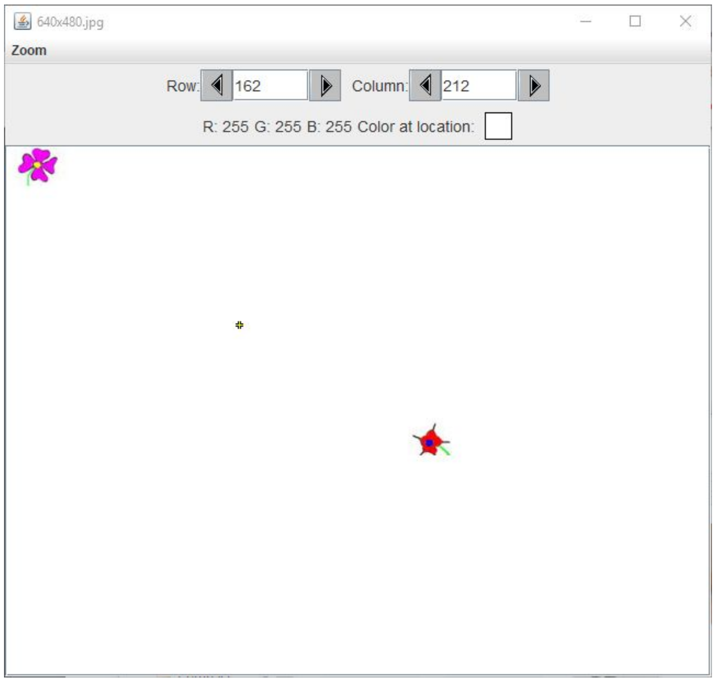
```
{
        int mirrorPoint = 350;
        Pixel leftPixel = null;
        Pixel rightPixel = null;
        Pixel[][] pixels = this.getPixels2D();

        // loop through the rows
        for (int row = 225; row < 332; row++)
        {
                // loop from 13 to just before the mirror point
                for (int col = 219; col < mirrorPoint; col++)
                {
                        leftPixel = pixels[row][col];
                        rightPixel = pixels[row][mirrorPoint - mirrorPoint];
                        rightPixel.setColor(leftPixel.getColor());
                }
        }
}
```


seagull.jpg
Zoom
Row: 0 Column: 0
R: 94 G: 122 B: 134 Color at location:

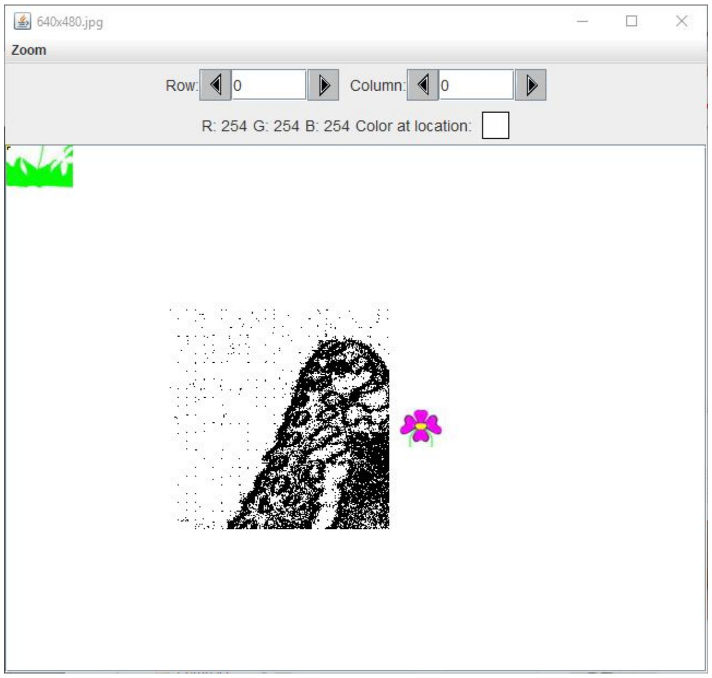

seagull.jpg
Zoom
Row: 0 Column: 0
R: 94 G: 122 B: 134 Color at location:

A8

Exercises

1. Create a second copy method that adds parameters to allow you to copy just part of the fromPic. You will need to add parameters that specify the start row, end row, start column, and end column to copy from. Write a class (static) test method in PictureTester to test this new method and call it in the main method.
   a. public void createCollage2()
```
{
  Picture flower1 = new Picture("flower1.jpg");
  Picture flower2 = new Picture("flower2.jpg");
  this.copy2(flower1, 0, 20, 35, 80, 0, 0);
  this.copy2(flower2, 10, 0, 50, 55, 240, 350);
  this.write("collage.jpg");
}
```


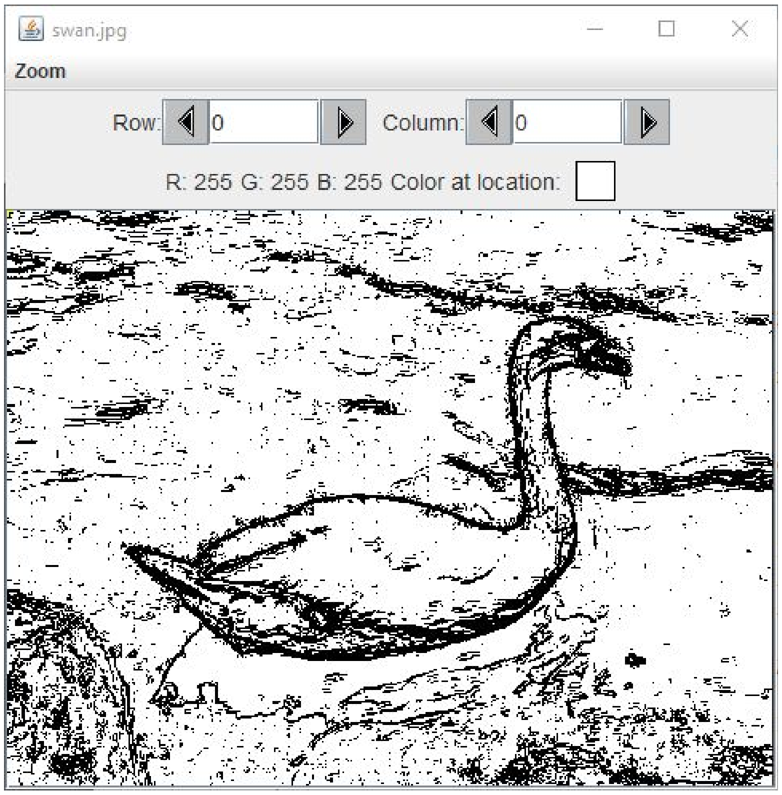
2. Create a myCollage method that has at least three pictures (can be the same picture) copied three times with three different picture manipulations and at least one mirroring. Write a class (static) test method in PictureTester to test this new method and call it in the main method.
   a. public void myCollage()
```
{
        Picture first = new Picture("flower1.jpg");
        first.mirrorDiagonal();
        this.copy2(first, 60, 20, 99, 80, 0, 0);

        Picture second = new Picture("flower1.jpg");
        second.mirrorVertical();
        this.copy2(second, 0, 20, 35, 80, 240, 350);

        Picture third = new Picture("butterfly1.jpg");
        third.edgeDetection(10);
        this.copy2(third, 0, 0, 200, 200, 150, 150);
        this.write("collage2.jpg");
}
```



A9

1. Notice that the current edge detection method works best when there are big color changes from left to right but not when the changes are from top to bottom. Add another loop that compares the current pixel with the one below and sets the current pixel color to black as well when the color distance is greater than the specified edge distance.
   a. public void edgeDetection2(int edgeDist)

```
        {
                Picture copy = new Picture(this);
                Pixel leftPixel = null;
            Pixel rightPixel = null;
            Pixel[][] pixels = this.getPixels2D();
            Color rightColor = null;

            for (int row = 0; row < pixels.length; row++)
            {
              for (int col = 0;
                   col < pixels[0].length-1; col++)
              {
                leftPixel = pixels[row][col];
                rightPixel = pixels[row][col+1];
                rightColor = rightPixel.getColor();
                if (leftPixel.colorDistance(rightColor) > edgeDist)
                  leftPixel.setColor(Color.BLACK);
                Else
                  leftPixel.setColor(Color.WHITE);
              }
            }

            Pixel[][] copyPixels = copy.getPixels2D();
            Pixel topPixel = null;
            Pixel botPixel = null;
            Color botColor = null;
            for(int row = 0; row < copyPixels.length-1; row++)
            {
                    for(int col = 0; col < copyPixels[0].length; col++)
                    {
                            topPixel = copyPixels[row][col];
                            botPixel = copyPixels[row+1][col];
                            botColor = botPixel.getColor();
                            if(topPixel.colorDistance(botColor) > edgeDist)
                            {
                                    pixels[row][col].setColor(Color.BLACK);
                            }
            }
        }
    }
```

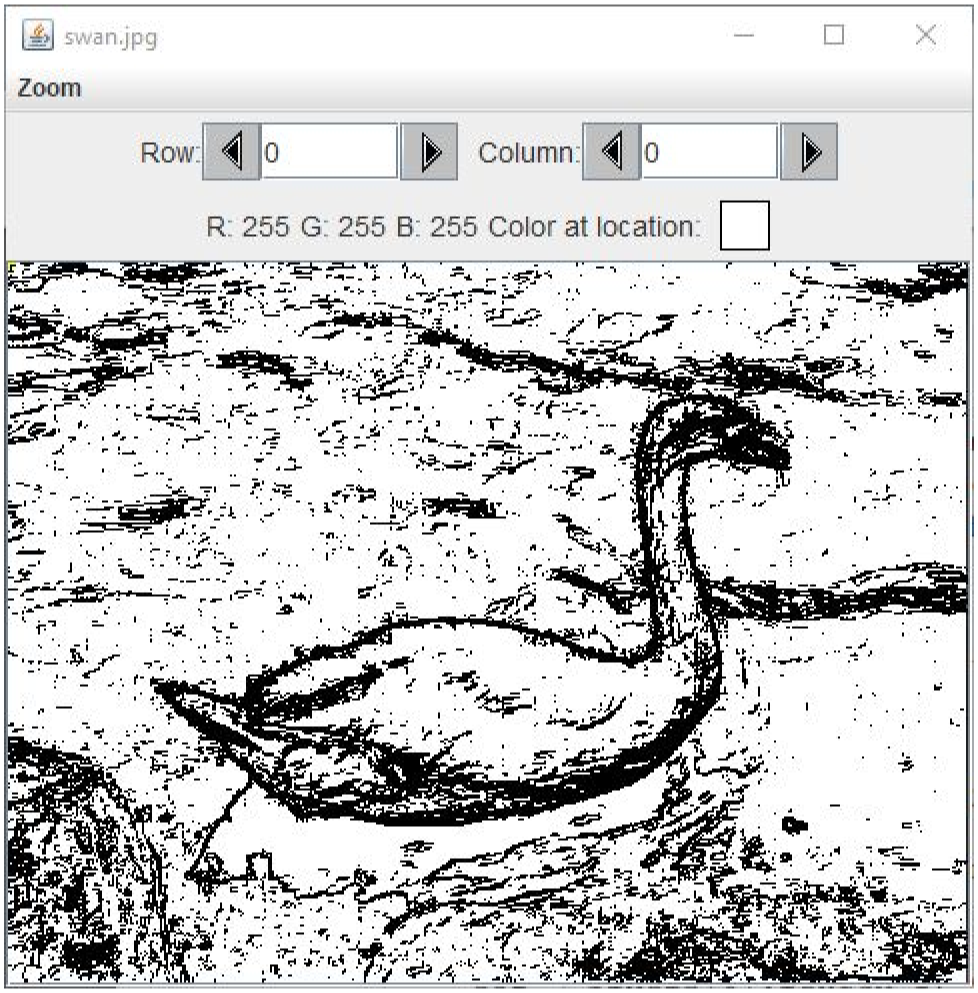


R: 255 G: 255 B: 255 Color at location:

2. Work in groups to come up with another algorithm for edge detection.
    a. public void edgeDetection3(int edgeDist)

```
    {
            Picture copy = new Picture(this);
            Pixel leftPixel = null;
      Pixel rightPixel = null;
      Pixel[][] pixels = this.getPixels2D();
      Color rightColor = null;

      for (int row = 0; row < pixels.length; row++)
      {
        for (int col = 0;
             col < pixels[0].length-1; col++)
        {
          leftPixel = pixels[row][col];
          rightPixel = pixels[row][col+1];
          rightColor = rightPixel.getColor();
          if (leftPixel.colorDistance(rightColor) > edgeDist)
            leftPixel.setColor(Color.BLACK);
          Else
            leftPixel.setColor(Color.WHITE);
        }
      }

      Pixel[][] copyPixels = copy.getPixels2D();
      Pixel topPixel = null;
      Pixel botPixel = null;
      Color botColor = null;
      for(int row = 0; row < copyPixels.length-1; row++)
      {
            for(int col = 0; col < copyPixels[0].length; col++)
            {
                topPixel = copyPixels[row][col];
                botPixel = copyPixels[row+1][col];
                botColor = botPixel.getColor();
                if(topPixel.colorDistance(botColor) > edgeDist)
                {
                    pixels[row][col].setColor(Color.BLACK);
                }
            }
      }

      Pixel[][] copyPix2 = copy.getPixels2D();
      Pixel firstPixel = null;
      Pixel diagPixel = null;
      Color diagColor = null;
      for(int row = 0; row < copyPix2.length-1; row++)
      {
            for(int col = 0; col < copyPix2[0].length-1; col++)
            {
                firstPixel = copyPix2[row][col];
                diagPixel = copyPix2[row+1][col+1];
                diagColor = diagPixel.getColor();
                if(firstPixel.colorDistance(diagColor) > edgeDist)
                {
                    pixels[row][col].setColor(Color.BLACK);
                }
            }
      }
```

```
        }
```



R: 255 G: 255 B: 255 Color at location: