Lab 6: Using For Loops and Nested For Loops

Objectives:

1. To practise using for loops and nested for loops.
2. To practise using for loops with one-dimensional arrays of pixels
3. To practice using nested for loops with two-dimensional arrays of pixels.

Preparation:

1. Go over the Lecture Notes: Topic 5, Topic 6, Topic 7.
2. Textbook reading (optional): Chapter 4 sections 4.3.6 - 4.3.10, Chapter 5 section 5.1

Exercise 1: Using for loops

In this Exercise, you will practice writing for loops and nested for loops.

1. In the Interactions pane, type a for loop that prints 10 asterisks on a line. (Hint: change the while loop in Lab 5, Exercise 1 a) to a for loop.
2. Reset the Interactions pane and type the following ***nested for loops*** that are intended to print 5 rows of 10 asterisks each. The ***outer loop*** counts the rows and the ***inner loop*** counts the asterisks per line. You need to fill in the blanks with the appropriate numeric values. (Note that loop counters do not always have to start with 1!)

```
for (int row = 1;  row <= _____ ; row ++)
{
   for (int count = 0; count < _____ ; count ++)
   {
    System.out.print("*");
   }
System.out.println();
}
```

Check that your code prints the correct output.

3. Now type the statement
`System.out.println(row);`
What happens? Why?

4. Use nested for loops as in step 2 to print the following:
*
**
***
****
*****
Hint: You can use the `row` variable to determine the condition on which the for loop ends:

when the `row` variable contains 1, how many asterisks should be printed on a line? when the `row` variable contains 2? when it contains 3?

Exercise 2: Looping through Pixels with a for loop

In this Exercise, you will access pixels of an image by using its one-dimensional array representation, as returned by the Picture method `getPixels().`

1. In the Interactions pane, make a variable `pictureObj` refer to a picture that contains the image in the file *butterfl2y.jpg* that is in your mediaSources folder. (Refer back to Lab 4 Exercise 1 if you need to.) Show it on the screen using the `explore`() method.

2. Get the one-dimensional array representation of the image as follows (reference: Topic 5 pages 24-27):

```
Pixel[] pixelArray = pictureObj.getPixels();
```

3. Type the following statement, filling in the length of the array in order to show how many pixels there are in the image (Hint: see Lab 5 Exercise 1 step 4)

```
System.out.println("The number of pixels in the image is " +
__________ );
```

4. Here is a while loop that accesses the first 100 pixels in `pixelArray` and outputs the coordinates and colour of each of these pixels (you do not need to type this in):

```
int i = 0;
while (i < 100) {
   Pixel pixelObj = pixelArray[i];
   System.out.println("At position " + i +" pixel at ("
      + pixelObj.getX() +","+ pixelObj.getY() +
      ") colour is: " + pixelObj.getColor());

   i++;
}
```

Modify the above code to **use a for loop** instead of a while loop, and type it into the Interactions pane.

5. Now print out the information about the pixels in the array at positions 1, 3, 5, 7, 9, 11, … 99. That is, print out the information about all pixels at odd numbered positions in the array, from position 1 up to and including position 99.

Exercise 3: Creating a General Method

In class we presented the `decreaseRed` method (Topic 6 slide 32) that decreases the red component of all pixels in a picture by half. A limitation with the method presented is that it only can decrease the red by half. What if we wanted to decrease by 20% or some other amount? What if we wanted to increase the red values? In class we then discussed (slide 43) writing a general method to change the amount of red in a picture, that takes a parameter of type `double` that indicates the amount by which to change the red. For example, if we wanted to decrease the red by 20%, the amount passed in would be 0.8; if we wanted to increase it by 20%, the amount passed in would be 1.2 In this Exercise, you will be writing such a general method and adding it to the Picture class.

1. The code for the `decreaseRed` method is provided at the end of this exercise. Add this method to your *Picture.java* (you can use the *Picture.java* that you downloaded for Lab 5)
2. Recompile the changed file *Picture.java* and fix any errors. In the Interactions pane, test the method by using the following statements and choosing the file *butterfly2.jpg* for the image file:

   ```
   String fileName = FileChooser.pickAFile();
   Picture pictureObj = new Picture(fileName);
   pictureObj.explore();
   pictureObj.decreaseRed();
   pictureObj.explore();
   ```

   You can check the new red values in the Picture Explorer.
3. Now add a new method called `changeRed` to *Picture.java*, that takes a double parameter indicating how much to change the red. Use the method header
   `public void changeRed(double howMuch)`
   In the code for your method definition, you should **use a for loop**. (Hint: model it on the provided code for `decreaseRed`, except change the while loop to a for loop)
4. Recompile the changed file *Picture.java* and fix any errors. Test your new method `changeRed` in the Interactions pane as in step 2, with some parameter of your choice that **increases** the red in the image, using the image in *butterfly2.jpg* again. Check that the red values were changed correctly.

Provided `decreaseRed` method:

```
public void decreaseRed()
  {
    Pixel[] pixelArray = this.getPixels();
    Pixel pixelObj = null;
    int index = 0;
    int value = 0;

    // loop through all the pixels
    while(index < pixelArray.length)
    {
```

```
        // get the current pixel
        pixelObj = pixelArray[index];

        // get the red value
       value = pixelObj.getRed();

       // decrease the red value
       value = (int) (value * 0.5);

        // set the pixel's red value
        pixelObj.setRed(value);

        // increment the index
       index++;
      }
    }
```

Exercise 4: Looping through Pixels with Nested For Loops

In Topic 6, we implemented a method `clearBlue()` of the Picture class that cleared the blue in all the pixels of an image, by using the one-dimensional array representation of the image (`pixelArray`) and a single for loop. In this Exercise, you will implement a method that does the same thing, going through the **two-dimensional array** representation of an image **using nested for loops.**

1. The almost completed code for a method `clearBlue2()` of the Picture class using nested for loops is given below. Add the method definition to your Picture class, filling in the blanks. (Hint: see Topic 7, slides 14 to 20). Does this method go through the pixels row-wise or column-wise?

```
public void clearBlue2()
  {
    Pixel pixelObj;
    // loop through the columns (x direction)
    for (int x = 0; x < this._____________; x++)
    {
      // loop through the rows (y direction)
       for (int y = 0; y < _____________; y++)
       {
        // get pixel at the x and y location
        pixelObj = this.getPixel(x,y);

        // set its blue to 0
        pixelObj._____________________;

       } //end of inner loop
    } // end of outer loop
   }
```

2. Recompile the changed file *Picture.java* and fix any errors. In the Interactions pane, test your method by using the following statements and choosing the file *butterfly2.jpg* for the image file:

```
String fileName = FileChooser.pickAFile();
Picture pictureObj = new Picture(fileName);
pictureObj.explore();
pictureObj.clearBlue2();
pictureObj.explore();
```

You can check that all the blue values are 0 in the Picture Explorer.

3. Now change your method `clearBlue2`() so that it goes through the pixels **row-wise.**

4. Recompile the changed file *Picture.java* and fix any errors. Test your changed method `clearBlue2()` in the Interactions pane as in step 2, choosing *butterfly2.jpg* again. Check that the blue values are all 0.