Assignment 1

SUMMARY
In this assignment you will add three methods to the Picture class (Picture.java) to frame and panel pictures.

DEADLINE
This assignment is due on Friday, February 28 at 11:00 pm.

DESCRIPTION

FRAME
Create a new method:
public Picture frame(int frameSize, Color frameColor)
in Picture.java. This method will create and return a new picture object. The new picture object will contain the original picture inside a frame. The thickness of the frame is given by the parameter frameSize. The color of the frame is given by the parameter frameColor. The new picture object will be larger than the original because of the frame.

PANEL
Create a new method:
public Picture panel(int columns, int rows, Color dividerColor)
in Picture.java. This method will create and return a new picture object. The new picture object will be the original picture with vertical and horizontal divider lines drawn on top. The columns parameter will specify the number of column segments that will result. That is, the number of vertical lines to be drawn is one less than the number of columns. e.g. four columns require three divider lines. Likewise for rows. The dividerColor parameter will specify the color of the divider lines. Divider lines should be one pixel thick and be evenly spaced. See the examples for reference.

PANELFRAME
Create a new method:
public Picture panelFrame(int columns, int rows, int frameSize, Color dividerFrameColor)
in Picture.java. This method will create and return a new picture object. The new picture object will be the original picture both paneled and framed. The columns and rows parameters determine the paneling columns and rows. The thickness of the frame is given by the parameter frameSize. The dividerFrameColor parameter will serve as both the dividerColor and frameColor.
Remove Red Eye

• Red eye is when the flash from the camera is reflected from the subject’s eyes

• We want to change the red color in the eyes to another color
  – But not change the red of her dress

Red Eye Algorithm

• We can find the area around the eyes to limit where we change the colors
  – Using `pictureObj.explore()`

  – But we still just want to change the pixels that are “close to” red.

  – We can find the distance between the current color and our definition of red

  • And change the color of the current pixel only if the current color is within some distance to the desired color

Detailed Red Eye Algorithm

• Pass the x and y of the starting location, and the x and y value of the end location.

• Using a nested loop through x and y:
  – Get the pixel at this x and y
  – Get the distance between the pixel color and red

  • If the distance is less than some value (167) then change the color to some passed new color

Conditional Execution

• Sometimes we want a statement executed only if some expression is true

  – We can use the “if” statement in Java

    \[
    \text{if}(\text{colorDistance < value})
    \]

    \[
    \text{Statement or block to execute} \\
    \text{next statement}
    \]
Using if Exercise

- Open DrJava and try this in the interactions pane

```java
int x = 2;
if(x > 1) System.out.println("X is > 1");
System.out.println("X is " + x);
x = 0;
if(x > 1) System.out.println("X is > 1");
System.out.println("X is " + x);
```

Color Distance

- The distance between two points is computed as

\[
\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}
\]

- There is a method in the Pixel class to do this

```java
public void removeRedEye(int startX, int startY,
int endX, int endY, Color newColor)
{
    Pixel pixelObj = null;

    // loop through the pixels in the rectangle defined by the
    // startX, startY, and endX and endY
    for(int x = startX; x < endX; x++)
    {
        for(int y = startY; y < endY; y++)
        {
            // get the current pixel
            pixelObj = this.getPixel(x,y);

            // if the color is near red then change it
            if(pixelObj.colorDistance(Color.RED) < 167)
                pixelObj.setColor(newColor);
        }
    }
}
```
Testing removeRedEye

- **Picture** `p = new Picture(".../jenny-red.jpg");`
- `p.explore();`
- **Color** `colorObj = new Color(150,150,200);`
- `p.removeRedEye(123,96,135,107,colorObj);`
- `p.explore();`

Edge Detection

- Find the areas of high contrast and turn pixels is this area black
  - Turn all other pixels white

Edge Detection Algorithm

- To find areas of high contrast
  - Loop from `y = 0` to `y < height - 1`
  - Loop from `x = 0` to `x < width`
    - Get the pixel at the `x` and `y` (top pixel)
    - Get the pixel at the `x` and `(y + 1)` bottom pixel
    - Get the average of the top pixel color values
    - Get the average of the bottom pixel color values
    - If the absolute value of the difference between the averages is over a passed limit
      - Turn the pixel black
      - Otherwise turn the pixel white

Use if and else for two possibilities

- Sometimes you want to do one thing if the expression is true
- and a different thing if it is false

```java
int x = 200;
if (x < 128)
    System.out.println("<128");
else
    System.out.println(">=128");
```
Edge Detection Exercise

- Write a method edgeDetection that takes an input limit
  - And turns all pixels black where the absolute value of the difference between that pixel and the pixel below is greater than the passed limit
  - And turns all pixels white where the absolute value of the difference between that pixel and the below pixel is less than or equal to the passed limit

```java
/**
 * Method to detect edges in a picture.
 * Color edges black and non-edges white.
 */
public void edgeDetection()
{
    Pixel topPixel = null;
    Pixel bottomPixel = null;
    for(int x = 0; x < this.getWidth(); x++)
    {
        for(int y = 0; y < this.getHeight()-1; y++)
        {
            topPixel = this.getPixel(x,y);
            bottomPixel = this.getPixel(x,y+1);
            if(topPixel.colorDistance(bottomPixel.getColor()) > 25)
                topPixel.setColor(Color.BLACK);
            else
                topPixel.setColor(Color.WHITE);
        }
    }
}
```

Testing Edge Detection

- `Picture p = new Picture(".../butterfly1.jpg");`
- `p.explore();`
- `p.edgeDetection(10);`
- `p.explore();`

Sepia-Toned Pictures

- Have a yellowish tint, used to make things look old and “western.”
Sepia-toned Algorithm

- First make the picture grayscale.
- Change the shadows (darkest grays) to be even darker ($0 \leq \text{red} < 60$)
  - Decrease all the colors.
- Make the middle grays a brown color ($60 \leq \text{red} < 190$)
  - Decrease blue.
- Make the highlights (lightest grays) a bit yellow ($190 \leq \text{red}$)
  - Increase red and green
  - Or decrease blue

We need to be able to do some more things with our expressions before we can implement this.

We have one condition that has two parts:

$$60 \leq \text{red} < 190$$

This says “red is greater than or equal to 60 and red is less than 190.”

How do we do this in Java?

Conditional Operators

- We can check if several things are true - AND
  - Using && (evaluation stops if the first item is false)
  - Using & (to always evaluate both operands)

- We can check if at least one of several things is true - OR
  - Using || (evaluation stops if the first item is true)
  - Using | (to always evaluate both operands)

- We can check if only one of the things is true – Exclusive OR
  - Using ^

Using && (And)

- Check that a value is in a range
  - For instance, is some value between 0 and 255 (inclusive) for valid pixel color values.
    $$0 \leq x \leq 255$$
    is written in Java as:
    
    $$0 \leq x && x \leq 255 \ // \ is \ the \ same$$
    $$x \geq 0 && x \leq 255 \ // \ is \ the \ same$$
    $$x \geq 0 && 255 \geq x \ // \ is \ the \ same$$
Using && (And)

- Check if a pixel is a certain color.
- For example, white.
- White is red, green and blue all being 255
- In Java, this is:
  \[
  \text{if (red == 255 && green == 255 && blue == 255)}
  \]

Using || (Or)

- Check if at least one of several things is true
  - For instance, that at least one of red or green is 0.
  - In Java, this would be
    \[
    \text{if (red == 0 || green == 0 || blue == 0)}
    \]

Count White Pixels Exercise

- Write a method that counts the number of white pixels (red = 255, blue = 255, green = 255) in a picture.
- The method should return the number of white pixels.
- Use it to count the number of white pixels in caterpillar.jpg.

<table>
<thead>
<tr>
<th>Conditional</th>
<th>Operand 1</th>
<th>Operand 2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>And</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>And</td>
<td>true</td>
<td>false</td>
<td>false</td>
</tr>
<tr>
<td>And</td>
<td>false</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>And</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>
### Truth Table

<table>
<thead>
<tr>
<th>Conditional</th>
<th>Operand 1</th>
<th>Operand 2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Or</td>
<td>true</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Or</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>Or</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Or</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

### Exclusive Or

<table>
<thead>
<tr>
<th>Conditional</th>
<th>Operand 1</th>
<th>Operand 2</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exclusive Or</td>
<td>true</td>
<td>true</td>
<td>false</td>
</tr>
<tr>
<td>Exclusive Or</td>
<td>true</td>
<td>false</td>
<td>true</td>
</tr>
<tr>
<td>Exclusive Or</td>
<td>false</td>
<td>true</td>
<td>true</td>
</tr>
<tr>
<td>Exclusive Or</td>
<td>false</td>
<td>false</td>
<td>false</td>
</tr>
</tbody>
</table>

### Not Conditional Operator

- Use `!` To change the value to the opposite
  - `!true` is false
  - `!false` is true

- The `!` operator can be used to change the meaning of expressions:
  - `!(x < 5)` is “x is not less than 5”
  - `!(x == 10)` is “x is not equal to 10”
Not Conditional Operator

• A not conditional operator applied to a complex conditional changes it:
  \[-!(\text{op}1 \&\& \text{op}2)\] is equal to \(!\text{op}1 \mid\mid !\text{op}2\)
  \[-!(\text{op}1 \mid\mid \text{op}2)\] is equal to \(!\text{op}1 \&\& !\text{op}2\)

• This is known as De Morgan’s Law

Using Multiple If Statements

• If we are doing different things based on a set of ranges

```java
if(0 <= x && x <= 5)
    System.out.println("Fail");
if(5 < x && x <= 10)
    System.out.println("D");
if(7 < x && x <= 8)
    System.out.println("C");
if(8 < x && x < 9)
    System.out.println("B");
if(9 <= x)
    System.out.println("A");
```

```java
if(0 <= x && x < 6)
    if(0 <= x && x < 6)
        Statement or block
    if(6 < x && x < 7)
        Statement or block
    if(7 < x && x < 8)
        Statement or block
    if(8 < x && x < 9)
        Statement or block
    if(9 <= x)
        Statement or block
```
Using Multiple If Statements

• If we are doing different things based on a set of ranges
  0 <= x < 6
  6 <= x < 7
  7 <= x < 8
  8 <= x < 9
  9 <= x
• You don’t need to check if x >= 6 since the first if block would have executed

if (0 <= x && x < 6)
  Statement or block
if (6 <= x && x < 7)
  Statement or block
if (7 <= x && x < 8)
  Statement or block
if (8 <= x && x < 9)
  Statement or block
if (9 <= x)
  Statement or block

Conditionals with > 2 Choices

if (0 <= x && x <= 5)
{
}
else if (x <= 10)
{
}
else // what is x?
{
}
// tint the shadows darker
if(redValue < 60)
{
    redValue = redValue * 0.9;
    greenValue = greenValue * 0.9;
    blueValue = blueValue * 0.9;
}

// tint the midtones a light brown by reducing the blue
else if(redValue < 190)
{
    blueValue = blueValue * 0.8;
}

// tint the highlights a light yellow by reducing the blue
else
{
    blueValue = blueValue * 0.9;
}

// set the colors
pixelObj.setRed((int) redValue);
pixelObj.setGreen((int) greenValue);
pixelObj.setBlue((int) blueValue);

Testing sepiaTint

• String file = FileChooser.getMediaPath("gorge.jpg");
• Picture p = new Picture(".../gorge.jpg");
• p.explore();
• p.sepiaTint();
• p.explore();