# Module IV: Conditionals and Selective Image Manipulation

### Learning Objectives

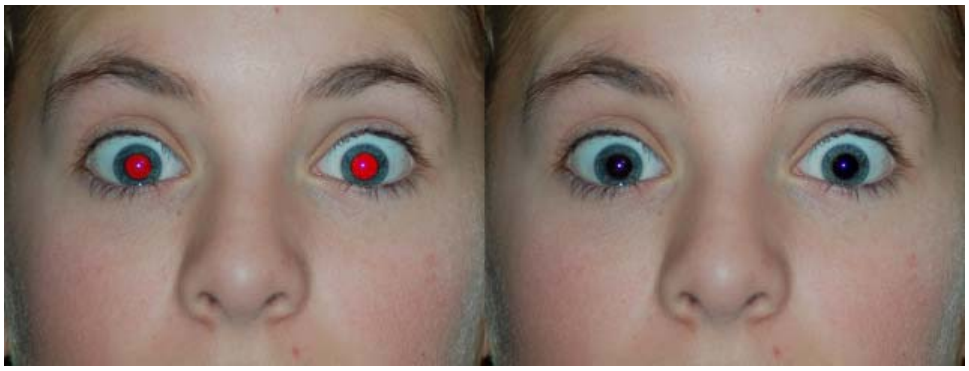
How to conditionally execute a statement or block of statements

How to remove red-eye from a picture

### Introduction

In the previous modules, we learnt how to manipulate images. However, we usually manipulated the whole image at once (*e.g.* removing the blue from all pixels or negating a whole image). But what happens if you only want to modify a specific part of an image? Can we do this?

The answer is yes (of course). A good example of this is red-eye removal. Red-eye happens frequently in pictures, when a camera flash is reflected off a subject's eyes, and makes photos look... creepy. We could always write an algorithm that removes all the red from a whole image, since that would definitely remove the red-eye. But if we remove the red, we also could be changing the color of the girl's clothing, which we do not want to do.



http://en.wikipedia.org/wiki/File:BoldRedEye-corrected.jpg

We'll discuss how to actually go about removing red-eye shortly, but first, we need to talk about conditionals.

#### Conditionals

If you've ever entered a contest, there were usually terms and *conditions*. This usually means that you could only win a prize if you met those conditions. These conditions are expressions that evaluate to either true or false. Let's look at a few examples.

#### Example Conditionals

Let's see how written conditions can be translated into Java. These are a few conditions that might be found on contest entries or other places:

- Must be 18 or older to enter
- Must be a legal resident of the Canada
- Must own a dog

All of these conditions can be expressed in the form of a yes/no question, like "Is [age] 18 or older?", or "Is [person] a legal resident of the Canada?". Once it is in this question form, it's easy to translate it to Java.

The question "Is [age] 18 or older?" can be translated to the Java expression age >= 18, where age is a variable representing how old a person is, and >= is asking whether that number is greater than 18.

You have also seen conditionals used in loops, where they are used to decide if a loop should continue or not. For example, in a for loop, the continuation area is a conditional that checks to see if a condition is met, and if so, the loop exits. This is the same for a while loop.

Notation... **==** ***versus*** **=**

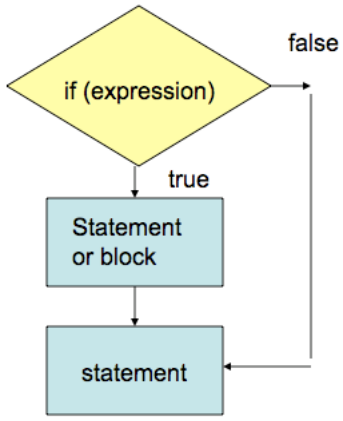
In Java, == and = have very different meanings. = is called the assignment operator and is used to *assign* values to variables. == on the other hand, is a comparison operator.
== *compares* 2 expressions, and returns true if both expressions equate to the same value.

Now that we have a way of translating from written conditions to Java expressions, let's talk more about conditional execution. Sometimes, we want a statement, or block of statements to execute only if some expression is true. To do this, we can use the if statement in Java.

The syntax for the if statement is like this:

```
if (condition) {
   // statement or block to execute
}
// next statement
```

The flow chart below also illustrates this concept:




If the conditional expression in the diamond (labeled "expression") evaluates to true, then the "statement or block" inside the if statement executes. Once that is completed, the next statement executes (after the if block of your code). If the conditional expression in the diamond is , then the "statement or block" is skipped, and the next statement executes (after the if block of your code).

Let's try this out with some real code. Create a Netbeans project and observe what happens.

```
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);
```

In the example above, we only executed ONE statement after the ***if*** statement. However, we can execute a whole series of statements as well. We call this a *block* of statements. The syntax for conditionally executing a block of statements is as follows:

```
if (expression) {
   // statement1
   // statement2
   // ...
   // ...
}
next statement
```

As you can see, in both if statements, we enclosed the inner statement or statements within '{' and '}'. This tells Java that the statement or statements should only be executed when the conditional expression is true.

> Note... Stylistically, when using an if statement, you should always indent the statement or block of statements following the if statement, as this makes it easier to read. For more information, refer to Writing Clean Code.

### Back to red-eye removal

Now that we know how to use conditional statements in Java, let's get back to red-eye removal. Let's first take a look at an algorithm we can use for red-eye removal. At a high level, the algorithm should look for pixels that are close to red in the area around the eyes. More specifically:

1. We only want to change the pixels that are "close to" red
2. We can find the distance between the current color and our definition of red
    a. Change the color of the current pixel only if the current color is within some distance to the desired color.

So that's the overview of the algorithm, but it's not detailed enough for us to actually write code for it. Here's a more detailed look at the algorithm, assuming we have the coordinates for the area around the eyes (x, y, w, h):

1. Loop from x to w
    a. Loop from y to h
        a. Get the pixel at the current x and y
        b. Get the distance between the pixel color and red
        c. If the distance is less than some value (167), change the color to some passed new color

Now, that algorithm is more developed and can be translated fairly easily into code. But what is this color distance thing? Colors don't have distances, do they? Well, they do, kind of. If we take each color as a point in a 3-dimensional space, we can calculate the distance between 2 colors fairly easily.

The distance between 2 points (http://www.purplemath.com/modules/distform.htm) is computed using the pythagorean theorem, (http://en.wikipedia.org/wiki/Pythagorean_theorem) like so: **distance = square_root$((x1-x2)^2 + (y1-y2)^2)$**

So, to compute the distance between 2 colors, we modify the equation slightly and come up with distance = **square_root$((red1-red2)^2 + (green1-green2)^2 + (blue1-blue2)^2)$**

If all that math has got you down, don't fret. Some programmer somewhere decided that lots of people might need to calculate this, and wrote a method for it in the Pixel class. So you can now use the following code:
**double dist = pixelObj.colorDistance(color1);**

Now that you've seen the algorithm, let's see what this actually looks like in code.

```
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
   // startX, startY, endX and endY
   for (int x = startX; x < endX; x++)    {
      for (int y = startY; y < endY; y++)  {
         //get the current pixel
         pixelObj = getPixel(x, y);

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

Finally, to test it out, we can use the following code. We've given an example that works for the picture used in our illustration above. You can always substitute a different file in if you prefer. We have specified that any red should be replaced with black in this example.

See Netbeans project MediaNine.

**learn by doing**

Write a method called countWhite to count the number of pixels close to Color.WHITE with a maximum distance of 10.

**solution**

We have to keep in mind of the following aspects: - How should we keep track of the number of white pixels we have seen so far? - Do we need x and y coordinates for this? - How would you decide when to add 1 to the number of white pixels?

```
public void countWhite() {
   Pixel[] pixelArray = this.getPixels();
   int numWhite = 0;

   // loop through all the pixels in the picture
   for (int i = 0; i < pixelArray.length; i++) {
      if (pixelArray[i].colorDistance(Color.white) < 10) {
         numWhite++;
      }
   }
   System.out.println("Number of white pixels: " + numWhite);
}
```

Now that you know how to change eye color, here's an added challenge. Take a photo of a friend, and try to change either their eye color or the color of their clothing. Can you write one method to do change colors of things, and call it several times with different parameters to handle eye color and clothing color and hair color? You can use the same basic principles that we have covered here (color distance, conditionals) to do this.

**Summary**

In this module, we learnt:

- How to use the if statement to conditionally execute a statement, or block of statements
- How to apply conditional statements to do useful things, like remove red-eye from pictures
- The syntax for an if statement:

```
if (condition)  {
   statement1
   statement2
     :
     :
}
next statement
```