Lab 9: Returning Values from Methods

Objectives:

1. To understand how methods return values
2. To use method return values

Preparation:

1. Go over the Lecture Notes: Topic 11.
2. Textbook Reading (optional): Chapter 5 section 5.2.5, 5.3.3

Exercise 1: Writing a Picture method that returns an integer

a) In Topic 11 of the Lecture Notes we saw a method for the Picture class called `countWhitePixels()` that returns the number of white pixels in a picture. This method is provided below. Using this method as a model, write a new method called `countNonWhitePixels()` that returns the number of pixels in a picture that are **not** white. Change **only** the method name and the `if` statement.

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

       if (pixelObj.getRed()==255 && pixelObj.getGreen()==255
           && pixelObj.getBlue()== 255)
           counter = counter + 1;
    }
   }
  return counter;
}
```

b) Test your new method in the Interactions pane on *caterpillar.jpg.* To see if you are getting the correct count returned, first call `countWhitePixels()` to see how many pixels are white. Then call your new method `countNonWhitePixels()`. What should be the value that it returns? (Hint: How many pixels are there in total for this picture?)

c) There is another way to approach the writing of the `countNonWhitePixels()` method, using the `countWhitePixels()` method that already exists. We can call a method that is in the Picture class from within another method that is in the Picture class (we have been doing this with getWidth() and getHeight() already, just as we do in the provided code below). So, fill in the blanks in the method `countNonWhitePixels2()` provided below to get the same result as in part a)

```
public int countNonWhitePixels2()
{
   int numPixels = this.getHeight()* this.getWidth();

   int numNotWhite = numPixels -  ____________________;

   return numNotWhite;
}
```

d) Test your new method in the Interactions pane on *caterpillar.jpg*

e) Optional: Rewrite the method `countNonWhitePixels2()` so that the body of the method consists of only one statement. (Hint: the `return` keyword can be followed by an expression.)

Exercise 2: Writing a Picture method that returns a boolean value

All of the *conditions* that we have seen in while loops and for loops and if statements are Boolean (logical) expressions, i.e. they evaluate to either `true` or `false.` In this exercise, you will write a method that returns `true` or `false`. What will be the return type of the method?

a) Create a new method in the Picture class that compares the size of two pictures. The method will be called `equalPictureSize` and will take one parameter that is a Picture object. It will have the header (you fill in the return type):
`public ___ equalPictureSize (Picture otherPicture)`
This method will be invoked on a Picture object. If `this` picture has the same width and height as the picture referenced by the parameter variable `otherPicture`, the method returns `true,` otherwise it returns `false`.

b) Test your new method in the Interactions pane by using two pictures that are the same size (for example *flower1.jpg* and *flower2.jpg*) and then two pictures that you know are of different height and width (for example *caterpillar.jpg* and *flower1.jpg*).

Exercise 3: Writing a Picture method that returns a Picture object

Provided below is the code for the Picture method `copyPicture()` from Topic 9 page 10. Recall that it is invoked on the target picture, and copies the source picture to the top left corner of the target picture.

a) First, you will write a new method for the Picture class called `copyLeftHalf()` that functions exactly like `copyPicture()`, but only copies the **left half** of the source picture to the target picture. It will have the header
`public void copyLeftHalf(Picture sourcePicture)`
(Hint: all you need to change in the body of the method is the limit on the `sourceX` coordinate!)

b) Test your new method with the image in *caterpillar.jpg* as the source picture, and *640x480.jpg* as the target picture. Why is there white space around the half-caterpillar picture?

c) Now suppose we wanted the target picture to be just the left half of the source picture, with **no** white space around it. In other words, we want our target picture to be of a size that *depends on the size of the source picture*. We discussed this idea in Topic 11 of the lecture notes: why and how to return a Picture object from a method. Write a different version of the method `copyLeftHalf()` of part a) so that it has the header:

```
public Picture copyLeftHalf()
```

that will be **invoked on the source picture**, and that will **return the target picture**.

(Hint: See how the `copyLeftRotation()` method in Topic 11 is changed from the version that has a header similar to your method of part a), to the version that has a header similar to the method you now want. Note that this includes adding code to your new version to create a new target picture that will be just the left half of the source picture. What should be the size of the new target picture?)
(Note that your new version of `copyLeftHalf()` has the same method name as your original `copyLeftHalf()` method. Why is this not a problem for the Java compiler?)

d) Test your new method using the following statements in the Interactions pane, choosing the file *caterpillar.jpg* as before.

```
Picture sourcePic = new Picture(FileChooser.pickAFile());
Picture targetPic =  sourcePic.copyLeftHalf();
targetPic.explore();

public void copyPicture (Picture sourcePicture)
{
   Pixel sourcePixel = null;
   Pixel targetPixel = null;
   // loop through the columns
   for (int sourceX = 0, targetX = 0;
        sourceX < sourcePicture.getWidth();
        sourceX++, targetX++)
   {
    // loop through the rows
     for (int sourceY = 0, targetY = 0;
          sourceY < sourcePicture.getHeight();
          sourceY++, targetY++)
     {

       // set the target pixel color to the source pixel color
       sourcePixel = sourcePicture.getPixel(sourceX,sourceY);
       targetPixel = this.getPixel(targetX,targetY);
       targetPixel.setColor(sourcePixel.getColor());
     }
   }
 }
```