Name: _____

Date: _____

ACTIVITY 1

Exploring Color

Look at the description of colors in Activity A1 of the Picture Lab. Each pixel's color is represented by a triplet of decimal numbers (RGB), where R represents the amount of red in the color, G represents the amount of green, and B represents the amount of blue. These decimal numbers range from 0 to 255. A larger number represents more of that color. Also described in that activity is the idea that the computer stores the color values in binary, with each value being represented in 8 bits, also known as a byte.

1. Answer the following review questions using the given website. Clicking on the color name or HEX value will give you the decimal value.
https://www.w3schools.com/colors/colors_names.asp

- What are the RGB values for White? R: 255 G: 255 B: 255
- What are the RGB values for Silver? R: 192 G: 192 B: 192
- What are the RGB values for Coral? R: 255 G: 127 B: 80

Clearing Bits

Colors can be manipulated by changing the individual color values. If you think about these values as binary, changing the values can be accomplished through clearing bits (setting to 0) or setting bits (setting to 1).

Consider a value of 255 (in decimal) for red. In binary, this is eight 1s (representing $2^7+2^6+2^5+2^4+2^3+2^2+2^1+2^0$). If the two leftmost bits are cleared (set to 0), the result is 0011 1111 in binary, or 63 in decimal. If instead, the two rightmost bits of 255 are cleared, the result is 1111 1100, or 252 in decimal.

2. Run `main` in `ColorChooser.java`, click on the RGB tab, and set red at 255, green at 0 and blue at 0. Notice the color (in the Preview area), which is bright red. Now set red to 252 and note the color change. Finally set red to 63 and note the difference from when red was 252. In one or two sentences, describe the differences between the different colors of red that you observed.

The red drops from a bright red color to a darker, maroon color. I think when the value is lower there is less light and so it turns to a mix of red and black.

If changes are made to bits on the left-hand side of a binary value, it has more impact on the magnitude of the number represented than if changes are made to bits on the right-hand side. From the previous example, clearing the left two bits of 255 resulted in 63 while clearing the right two bits resulted in 252. For colors, this means that clearing two bits on the right-hand side doesn't change the color very much, while clearing two bits on the left-hand side changes the color significantly.

While changing the bits is easy if the number is in binary (just change those bit positions to 0), this course deals with decimal numbers. It will be helpful to clear and set bits by performing operations on decimal numbers. Consider the following numbers, shown in both decimal and binary format:

Original Decimal	*Original Binary*	*Altered Binary*	*Altered Decimal*
183	1011 0111	1011 0100	180
5	0000 0101	0000 0100	4
80	0101 0000	0101 0000	80

Note that in a number where the rightmost two bits are already 0, this clearing of bits makes no difference in its value as in the example of the decimal number 80 above.

To figure out how to clear the last two bits on the right, consider that as the positions in the binary number change, moving from right to left, powers of 2 increase by 1. So, the rightmost position is 2^0, the next position is 2^1, etc. **Dividing a decimal number by 2 using integer division has the effect of removing the rightmost bit in the binary representation of the number.**

For example, if the number 183 (represented in binary as 1011 0111) is divided by 2, all the bits in the binary representation move to the right and the result is 91 (represented in binary as 0101 1011). Whether the rightmost bit was 0 or 1, it's now gone. If the resulting decimal value is multiplied by 2, all the bits in the binary representation move to the left. 91 (0101 1011 in binary) times 2 is 182 (1011 0110 in binary). Note that the rightmost bit is now cleared.

3. What if we want to clear (set to zero) the rightmost **two** bits? With a group, determine the steps needed to accomplish this.

Find value % 4 to isolate the last two bits. Then subtract this from value.

4. On your own, try your algorithm with the value 183. Record your process, and the intermediate values generated, in the space below.

183 % 4 = 3, 183-3 = 180, 180 is the answer.

5. **Complete the Table:** Try this process of dividing by 4 and multiplying by 4 with the other numbers in the leftmost column of the table and verify that the result for each will be the number in the second to last column.

Original Decimal	*Original Binary*	*Altered Decimal After Dividing by 4*	*Altered Binary After Dividing by 4*	*Altered Decimal After Multiplying by 4*	*Altered Binary After Multiplying by 4*
183	1011 0111	45	10 1101	180	1011 0100
5	0000 0101	1	00 0001	4	0000 0100
80	0101 0000	20	01 0100	80	0101 0000

Changing Colors

The above operation can be done on colors of pixels in Java.

6. Create a `Steganography` class, which will only have static methods and use the `Picture` class to manipulate images. This class will be executable so include a `main` method which will be implemented later. You must add the code `import java.awt.Color;` to the top of your file.

Add the following method to the `Steganography` class.

```
/**
 * Clear the lower (rightmost) two bits in a pixel.
 */
public static void clearLow( Pixel p )
{
   /* To be implemented */
}
```

7. In the area specified "To be implemented," implement your algorithm to clear the rightmost two bits from each of the color components R, G, and B of the given `Pixel p`.

8. Add a `static` method `testClearLow` that accepts a `Picture` as a parameter and returns a new `Picture` object with the lowest two bits of each pixel cleared.

Change `main` in `Steganography.java` to contain the following lines:

```
Picture beach = new Picture ("beach.jpg");
beach.explore();
Picture copy = testClearLow(beach);
copy.explore();
```

Run `main` and compare the two pictures.

In `Steganography.java` add the following method:

```
/**
 * Set the lower 2 bits in a pixel to the highest 2 bits in c
 */
public static void setLow (Pixel p, Color c)
{
   /* To be implemented */
}
```

11. In the area specified "To be implemented," implement the process described above to replace the lowest two bits of each color value with the highest two bits of color value of the parameter `c`.

12. Add a `static` method `testSetLow` that accepts a `Picture` and a `Color` as parameters and returns a new `Picture` object with the lowest two bits of each pixel set to the highest two bits of the provided color.

Change `main` in `Steganography.java` to contain the following lines:

```
Picture beach2 = new Picture ("beach.jpg");
beach2.explore();
Picture copy2 = testSetLow(beach2, Color.PINK);
copy2.explore();
```

Note that again, the two pictures appear to be identical, yet looking at individual pixels, you'll see that the color values differ between 0 and 3.

13. To see a representation of the hidden image, the rightmost two bits for each color component need to become the most significant (leftmost) bits of the components of a new color. With a group, determine the algorithm needed to reveal the 'hidden' picture using pseudocode.

Take the rightmost two bits by taking the value % 4, then multiply that value by 64 to make them the leftmost bits of a new color, then add them to the new color % 64 to isolate the last 6 digits of it. This makes 2 digits from the first color and 6 from the second, to make 8 original bits.

Add the following method to `Steganography.java`:

```
/**
 * Sets the highest two bits of each pixel's colors
 * to the lowest two bits of each pixel's colors
 */
public static Picture revealPicture(Picture hidden)
{
   Picture copy = new Picture(hidden);
   Pixel[][] pixels = copy.getPixels2D();
   Pixel[][] source = hidden.getPixels2D();
   for (int r = 0; r < pixels.length; r++)
```

```
{
   for (int c = 0; c < pixels[0].length; c++)
   {
      Color col = source[r][c].getColor();
      /* To be Implemented */
   }
}
return copy;
```

14. In the area specified "To be implemented", implement the process to isolate the rightmost two bits of the color values of `col` and move them to the leftmost position in `copy`.

Add the following to the `main` method:

```
Picture copy3 = revealPicture(copy2);
copy3.explore();
```

These lines take the previously hidden color and then reveal it. These techniques will be explored more in Activity 2.

Check Your Understanding

The same techniques that were used to isolate bits can be used to isolate different components in a decimal number (1s, 10s, 100s, etc). Discuss with a partner when you would need to isolate different parts of a decimal number.

15. On your own, answer the following question and then discuss with a partner:

How would you isolate the tens digit from a decimal number of unknown size? What about the hundreds or thousands digit?

Divide the value by its wanted value, such as the 10s digit, then take that value % 10. For example, 1289's tens digit, 1289/10 -> 128. 128%10 = 8, which is the tens digit.

```java
import java.awt.Color;
/**
 * Write a description of class Steganography here.
 *
 * @author Akshat Garg
 * @version 11.27.24
 */
public class Steganography
{
    // instance variables - replace the example below with your own
    private int x;

    /**
     * Constructor for objects of class Steganography
     */
    public Steganography()
    {
        // initialise instance variables
        x = 0;
    }
    public static void main(String[] args)
    {
        Picture beach = new Picture ("beach.jpg");
        beach.explore();
        Picture copy = testClearLow(beach);
        copy.explore();

        Picture beach2 = new Picture ("beach.jpg");
        beach2.explore();
        Picture copy2 = testSetLow(beach2, Color.PINK);
        copy2.explore();

        Picture copy3 = revealPicture(copy2);
        copy3.explore();

    }
    /**
     * Clear the lower (rightmost) two bits in a pixel
     */
    public static void clearLow(Pixel p)
    {
        /* To be implemented*/
        p.setRed((p.getRed()/4)*4);
        p.setGreen((p.getGreen()/4)*4);
        p.setBlue((p.getBlue()/4)*4);
    }
    public static Picture testClearLow(Picture pic)
    {
        Pixel[][] pixels = pic.getPixels2D();
        for (Pixel[] rows:pixels){
            for (Pixel p: rows){
                clearLow(p);
            }
        }

        return pic;
    }
```

```java
/**
 * Clear the lower (rightmost) two bits in a pixel
 */
public static void setLow(Pixel p, Color c)
{
    /* To be implemented*/
    p.setRed(p.getRed()/4*4+c.getRed()/64*64);
    p.setGreen(p.getGreen()/4*4+c.getGreen()/64*64);
    p.setBlue(p.getBlue()/4*4+c.getBlue()/64*64);
}
public static Picture testSetLow(Picture pic, Color c)
{
    Pixel[][] pixels = pic.getPixels2D();
    for (Pixel[] rows:pixels){
        for (Pixel p: rows){
            setLow(p,c);
        }
    }

    return pic;
}
/** Sets the highest two bits of each pixel's colors* to the lowest two bits of each pixel's coloros */
    public static Picture revealPicture(Picture hidden)
    {
        Picture copy = new Picture(hidden);
        Pixel[][] pixels = copy.getPixels2D();
        Pixel[][] source = hidden.getPixels2D();
        for (int r = 0; r < pixels.length; r++)
        {
            for (int c = 0; c < pixels[0].length; c++) {
                Color col = source[r][c].getColor();
                pixels[r][c].setRed((col.getRed()%4)*64+col.getRed()%64);
                pixels[r][c].setGreen((col.getGreen()%4)*64+col.getGreen()%64);
                pixels[r][c].setBlue((col.getBlue()%4)*64+col.getBlue()%64);
            }
        } return copy;
    }
```