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: 169 G: 169 B: 169
- 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 difference between 255 and 252 versus 255 and 63 is drastic. 252 did not have noticeable changes while 63 almost appeared to be 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.

Take number of third rightmost digit (2^2), convert to base 10, divide the 183 by 2^2 and multiply back 2^2.
Convert result back to binary

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

183 ≡ 10110111 ← to be cleared

$2^2 = 4$ ← position left of bits to be cleared

183/4 ≒ 45.75 = 45

45 x 4 = 180

180 ≡ 10110100

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			4	
80	0101 0000			80	

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.

9. Are you able to discern a difference between the image with the two lowest bits cleared and the original? If so, describe the difference that you see. If not, provide a brief explanation of why this change is not noticeable.

There is a minute change in the brightness of the colours.

To summarize—clearing the rightmost two bits in a pixel's individual color component value does not change the color enough to be perceptible when viewing the picture. These bits can be cleared by dividing each color value (Red, Green, Blue) by 4 and then multiplying each by 4 and setting the pixel's color to a color represented by these values.

Setting Bits

As previously established, clearing the rightmost two bits of the Red, Green, or Blue values in the color of a pixel does not impact the color much. Because removing these values does not change the color in a perceptible way, it's possible to use these bits to store different information.

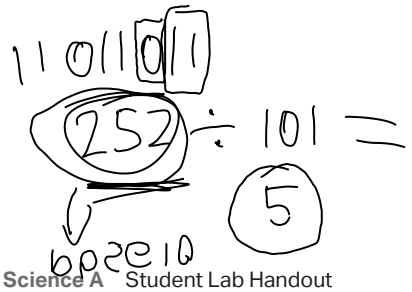
10. If dividing a decimal equivalent value by 4 removed the rightmost (lowest) two bits, what value would you need to divide by in order to remove the rightmost six bits, isolating the leftmost (highest) two bits in an eight-bit number?

64

In the code below, notice that the method `setLow` has a `Pixel p` and a `Color c` as parameters. This color is the 'new information' that will be stored in the now cleared bits. To store this information, a pixel is modified by adding the leftmost two bits of the color values of c (isolated by dividing by the value identified above) to the color values of `Pixel p`.

	Original Pixel Values		*Parameter Color c Value*		*After Call to setLow*	
	Decimal	***Binary***	***Decimal***	***Binary***	***Decimal***	***Binary***
Red	139	1000 1011	218	**11**01 1010	139	1000 10**11**
Green	66	0100 0010	112	**01**11 0000	65	0100 00**01**
Blue	16	0001 0000	214	**11**01 0110	19	0001 00**11**

So, in the code below, the rightmost two bits in the original pixel's colors are being set to the leftmost bits of another `Color c` by adding those bits from c to the color values of the pixel (after the pixel's rightmost two bits have been cleared).



enter 183
store
run through algorithm
|| convert to base-2
|| get position

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.

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?

Mod unknown number by 10, and add remainder back to quotient. Similar method for hundreds or thousands digit; mod by 100 and 1000.

Name: _____

Date: _____

ACTIVITY 2

Hiding and Revealing a Picture

arch.jpg

beach.jpg

In original form, `arch.jpg` is 360 X 480, while `beach.jpg` is 640 X 480.

1. Do you need to resize either of the images to fit one within the other? Why or why not?

NO, since arch.jpg's image does not go over beach.jpg's borders.

2. Identify the image that would need to be resized if you wanted to fit it into the other image. Explain the required modifications that would need to be made.

femaleLionAndTail.jpg, since it has the length and width extends past beach.jpg's borders.

Recall from Activity 1 that changing the lowest two bits of each color in all pixels of an image did not noticeably change the image. Taking advantage of this will allow hiding an image (`secret.jpg`) inside another image (`source.jpg`) by replacing the lowest two bits of each color in all pixels of `source.jpg` with the highest two bits of each color in all pixels of `secret.jpg`. Consider the following pixels (referring to Activity 1 to check the arithmetic):

- source pixel: `java.awt.Color[r=104,g=89,b=191]`
- secret pixel: `java.awt.Color[r=221,g=193,b=47]`
- combined pixel: `java.awt.Color[r=107,g=91,b=188]`
- revealed pixel: `java.awt.Color[r=192,g=192,b=0]`

3. If the top left pixel of `source.jpg` has the color `java.awt.Color[r=234,g=172,b=92]` and the top left pixel of `secret.jpg` has the color `java.awt.Color[r=120,g=34,b=196]` then what would be the color of the top left pixel of the combined image?

brown.

4. What would be the color of the top left pixel of the revealed image?

¯_(ツ)_/¯

5. Why are the lowest two bits of each color in all pixels in `source.jpg` replaced rather than the highest two bits?

Highest two bits would change the colour too much to be hidden.

6. Why are the highest two bits of each color in all pixels in `secret.jpg` used in the resulting image rather than the lowest two bits?

Adding the lowest two bits back into the highest two bits preserves the most information of the secret image.

7. After `arch.jpg` has been hidden in another image and then revealed, the revealed image is shown below. It almost looks pixelated. Why?

Some information of the pixels, specifically the lowest two bits, is lost through the hiding process.

8. Write the static method `canHide` that takes two pictures (source and secret) and checks picture sizes to make sure you can hide the secret in source. For now, this method should check if the two images are the same size, returning `true` if the two pictures have the same height and width, and `false` otherwise. This method will be modified in the following activity. Add code to `main` to test this method.

```
/**
 * Determines whether secret can be hidden in source, which is
 * true if source and secret are the same dimensions.
 * @param source is not null
 * @param secret is not null
 * @return true if secret can be hidden in source, false otherwise.
 */
public static boolean canHide(Picture source, Picture secret)
```

9. Write the static method `hidePicture` that takes two pictures (source and secret) and hides the secret in source using the algorithm previously discussed, returning the new picture. Add code to `main` to test this method.

```
/**
 * Creates a new Picture with data from secret hidden in data from source
 * @param source is not null
 * @param secret is not null
 * @return combined Picture with secret hidden in source
 * precondition: source is same width and height as secret
 */
public static Picture hidePicture(Picture source, Picture secret)
```

Tip

One iterative process can trigger a second iterative process, requiring the first process to wait while the second completes. Often the first iterative process provides input values through control variables for the second process. Regardless of where the iterative statement is in the overall program code, the only control variables that are changing are within that iteration statement.

10. Verify that the method `revealPicture` added to the `Steganography` class in Activity 1 still works as expected, namely when called with a picture (combined) reveals the secret picture by returning a new picture containing only the hidden pixels.

11. Write the `main` method which should construct two images and call `canHide` with them. If `canHide` returns `true`, the method calls `hidePicture`, calls `explore` on the picture returned, calls `revealPicture` and then calls `explore` on the new picture.

Check Your Understanding

Briefly discuss with a partner how the code for each of the implemented methods would need to change to allow a smaller image to be hidden in a larger image at a random location.

12. How could the hiding algorithm be altered so the revealed image is more like the original secret image? What effect would that have on the combined image?

During the hiding process, hide the lowest three bits in the highest three bits and vice versa in the revealing process preserves more information.

Name: _____

Date: _____

ACTIVITY 3

Identifying a Hidden Picture

1. In the `Steganography` class, after modifying `canHide` to allow the secret image to be smaller than the source image, write a second version of the `hidePicture` method that allows the user to place the hidden picture anywhere on the modified picture. This means adding two parameters for `startRow` and `startColumn` that represent the row and column of the upper left corner where the hidden picture will be placed.

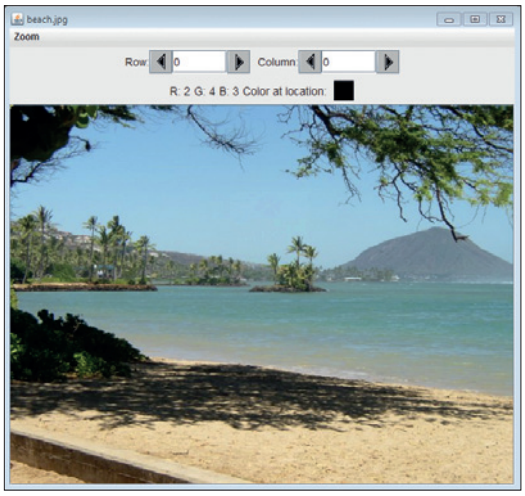
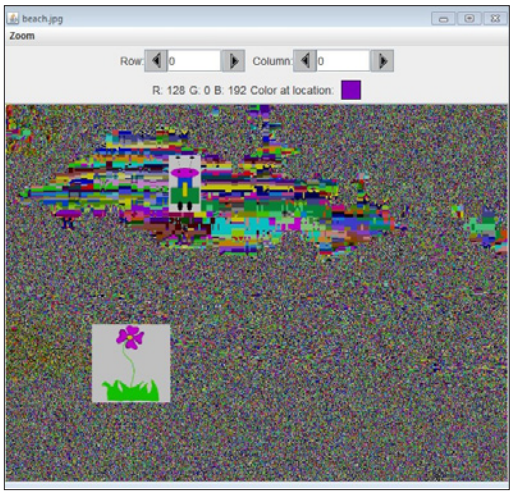
Sample code:

```
Picture beach = new Picture("beach.jpg");
Picture robot = new Picture("robot.jpg");
Picture flower1 = new Picture("flower1.jpg");
beach.explore();

// these lines hide 2 pictures
Picture hidden1 = hidePicture(beach, robot, 65, 208);
Picture hidden2 = hidePicture(hidden1, flower1, 280, 110);
hidden2.explore();

Picture unhidden = revealPicture(hidden2);
unhidden.explore();
```

Results:

`hidden2` ***With Pictures Hidden:***	`unhidden` ***After Calling*** `revealPicture`***:***
	

2. Write a method `isSame` that takes two pictures and checks to see if the two pictures are exactly the same (returns `true` or `false`).

Sample code:

```
Picture swan = new Picture("swan.jpg");
Picture swan2 = new Picture("swan.jpg");
System.out.println("Swan and swan2 are the same: " +
   isSame(swan, swan2));
swan = testClearLow(swan);
System.out.println("Swan and swan2 are the same (after clearLow run on swan): "
  + isSame(swan, swan2));
```

Results:

```
Swan and swan2 are the same: true
Swan and swan2 are the same (after clearLow run on swan): false
```

3. Write a method `findDifferences` that takes two pictures and makes a list of the coordinates that are different between them (returns an `ArrayList` of points/coordinates).

Sample code:

```
Picture arch = new Picture("arch.jpg");
Picture koala = new Picture("koala.jpg");
Picture robot1 = new Picture("robot.jpg");
ArrayList<Point> pointList = findDifferences(arch, arch2);
System.out.println("PointList after comparing two identical pictures " +
   "has a size of " + pointList.size());
pointList = findDifferences(arch, koala);
System.out.println("PointList after comparing two different sized pictures " +
   "has a size of " + pointList.size());
Picture arch2 = hidePicture(arch, robot1, 65, 102);
pointList = findDifferences(arch, arch2);
System.out.println("Pointlist after hiding a picture has a size of  "
   + pointList.size());
arch.show();
arch2.show();
```

Results:

```
PointList after comparing two identical pictures has a size of 0
PointList after comparing two different sized pictures has a size of 0
PointList after hiding a picture has a size of 2909
```

Original Arch	*Arch with Hidden Robot*	*Points in* `PointList` *are Colored*
		

4. Write a method `showDifferentArea` that takes a picture and an `ArrayList` and draws a rectangle around part of picture that is different (returns a `Picture`).

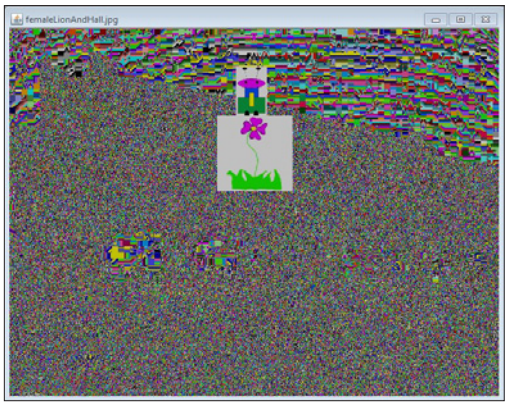
Tip

To provide instructions for the computer to process many different input values, selection statements may need to be nested together to form more than two branches and options. Pathways can be broken down into a series of individual selection statements based on the conditions that need to be checked and nesting together the conditions that should only be checked when other conditions fail or succeed.

Sample code:

```
Picture hall = new Picture("femaleLionAndHall.jpg");
Picture robot2 = new Picture("robot.jpg");
Picture flower2 = new Picture("flower1.jpg");
// hide pictures
Picture hall2 = hidePicture(hall, robot2, 50, 300);
Picture hall3 = hidePicture(hall2, flower2, 115, 275);
hall3.explore();
if(!isSame(hall, hall3))
{
   Picture hall4 = showDifferentArea(hall,
      findDiffferences(hall, hall3));
   hall4.show();
   Picture unhiddenHall3 = revealPicture(hall3);
   unhiddenHall3.show();
}
```

Results:

Bounding Rectangle	*Unhidden Pictures*
	

Check Your Understanding:

5. This activity is focused on being able to hide the secret image in a random location within the source image. One aspect that was not mentioned was how to determine an appropriate random location to hide the secret image. Assuming source and secret are both known (and therefore the height and width of each could be determined through method calls), how would you generate random `row` and `column` values to start hiding secret?

Complete the following expressions:

`int row` = _____

`int column` = _____

6. Are your coordinates guaranteed to fit secret within source? If not, modify the above expressions to ensure that there is room to fit the entire secret image within source.

No.

Name: _____

Date: _____

ACTIVITY 4

Hiding and Revealing a Text Message

Information can be represented in various ways, as seen throughout this lab with images being stored as 2D arrays of pixels. All information stored on a computer is stored as sequences of bits, and how those bits are interpreted can completely change their meaning. Consider the following eight-bit sequence—0100 1101. It's impossible to know what it represents without context. It could be the integer 77, the character 'N', the amount of red in a pixel, or any number of things.

The biggest consideration when determining how to represent information is knowing how many items need to be represented, since this will ultimately determine the number of bits that will be used. For example, early monitors were not capable of displaying the variance in color that current monitors can display, so colors were represented using far fewer bits. As monitors have improved and become capable of displaying more colors, the number of bits used to store color has increased so that all possible colors that can be displayed have a unique representation.

1. How many items can be represented with the following numbers of bits:

2 bits? ___ 4 bits? ___ 8 bits? ___

This activity involves hiding and then revealing a text message in a picture. The ideas are the same as those involved in hiding and revealing a picture, but the text message must be split up and reassembled more carefully.

Recall that to hide a picture, the leftmost two bits in each color for each pixel from the secret image are stored in the source image as the rightmost two bits in each matching color/pixel. The picture can be revealed by taking those rightmost bits and making them the leftmost bits again.

In this activity, you're going to store messages consisting of uppercase letters and spaces in a picture. To do this, the 26 letters will be represented by numbers (1–26 where A = 1, etc.), a space will be represented using the value 27, and a 0 will represent the end of the string. In this way, messages will be able to be coded using 28 distinct integer values.

By encoding a message in this way, five bits will be needed to hold each letter or space. Recall that five bits can hold the decimal values ranging from 0 to 31 so there are even a few extra bits in case punctuation is desired. The desire is for the text to be truly hidden in the picture, so only the rightmost two bits in each color value of each pixel are used to store the coded message. Since there are three color components per pixel, each coded character will be split into three pairs of two bits where the leftmost bit is always 0. Again, more characters such as punctuation can be added

later to the messages being stored. This six-bit scheme is convenient for storing in three colors, but also gives room to grow later if necessary.

Here's an example of the encoding of a message. Suppose "HELLO WORLD" was the message. Using the 28-integer code, it would be: 8, 5, 12, 12, 15, 27, 23, 15, 18, 12, 4, 0.

Create a method to encode the string into these integers. Type the following code into `Steganography.java`:

```
/**
 * Takes a string consisting of letters and spaces and
 * encodes the string into an arraylist of integers.
 * The integers are 1-26 for A-Z, 27 for space, and 0 for end of
 * string. The arraylist of integers is returned.
 * @param s string consisting of letters and spaces
 * @return ArrayList containing integer encoding of uppercase
 *      version of s
 */
public static ArrayList<Integer> encodeString(String s)
{
   s = s.toUpperCase();
   String alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
   ArrayList<Integer> result = new ArrayList<Integer>();
   for (int i = 0; i < s.length(); i++)
   {
      if (s.substring(i,i+1).equals(" "))
      {
         result.add(27);
      }
      else
      {
         result.add(alpha.indexOf(s.substring(i,i+1))+1);
      }
   }
   result.add(0);
   return result;
}
```

Another method is necessary to decode a list of integers back into a string. Make the method `public` to test but then make it `private`.

```
/**
 * Returns the string represented by the codes arraylist.
 * 1-26 = A-Z, 27 = space
 * @param codes encoded string
 * @return decoded string
 */
public static String decodeString(ArrayList<Integer> codes)
{
   String result="";
   String alpha = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
   for (int i=0; i < codes.size(); i++)
   {
      if (codes.get(i) == 27)
      {
         result = result + " ";
      }
      else
      {
         result = result +
             alpha.substring(codes.get(i)-1,codes.get(i));
      }
   }
   return result;
}
```

To hide the string in the picture, one more method is needed. This method takes an integer between 0 and 63 and splits it into three two-bit pairs.

```
/**
 * Given a number from 0 to 63, creates and returns a 3-element
 *  int array consisting of the integers representing the
 *  pairs of bits in the number from right to left.
 * @param num number to be broken up
 * @return bit pairs in number
 */
private static int[] getBitPairs(int num)
{
   int[] bits = new int[3];
   int code = num;
   for (int i = 0; i < 3; i++)
   {
      bits[i] = code % 4;
      code = code / 4;
    }
   return bits;
}
```

2. You're now ready to create the method that hides the message. Complete the following method in the `Steganography` class.

```
/**
 * Hide a string (must be only capital letters and spaces) in a
 *  picture.
 *  The string always starts in the upper left corner.
 *  @param source picture to hide string in
 * @param s string to hide
 * @return picture with hidden string
 */
public static void hideText(Picture source, String s)
```

3. Now, complete the following method to be able to return the secret message hidden in the image.

```
/**
 * Returns a string hidden in the picture
 * @param source picture with hidden string
 * @return revealed string
 */
public static String revealText(Picture source)
```

4. In the `main` method, add code to test hiding and revealing a message.

Check Your Understanding

5. Given the representation scheme used, would it be possible to represent both uppercase and lowercase letters? What about digits? Exactly how many characters can be represented using the six-bit encoding scheme?

NO. Yes. 64 characters.

6. When storing the secret message, a special value was used to signify the end of the message. Discuss with a partner what would happen if there was no way to signal the end of a message. Which methods would change? Describe how the behavior would be different.

There may be unnecessary characters created, creating confusion as to which characters were part of the intended text.

Name: _____

Date: _____

ACTIVITY 5

Open-Ended Activity

This open-ended activity requires you to develop a program on a topic that interests you. As a class, spend a few minutes reviewing the requirements of the open-ended activity.

Requirements:

- Create a program with a `main` method.
- Create at least one new method that is called from `main` (can be part of another class, such as `Steganography`) that takes at least one parameter.
- Traverse elements in a 2D array or do parallel traversals of multiple data structures.
- Modify some elements in a data structure based on the identified purpose.

In addition, review the provided scoring guidelines so that you understand what you'll be expected to explain once you're done completing your program.

It's strongly recommended that the implementation of the program involve collaboration with another student. Your selected program can be anything that you choose that meets the requirement and allows you to demonstrate your understanding.

Before beginning, make sure that you understand the expectations for the activity.

- Who will you be working with? Are you allowed to work with a partner? In a group of three or four?
- Among the members of your group (or with your partner), how will the implementation be completed?
- If you'll be using pair programming, will your teacher be instructing you when to switch driver and navigator, or is this something that you need to keep track of?
- What should you do if your group/pair is stuck? Does your teacher want you to come straight to them? Are you allowed to ask another group?

Tip

For groups that choose to traverse a 2D array in something other than row-major order:

There is a specific nested iterative structure to traverse elements in a 2D array in row-major order. This structure can be modified to traverse in column-major order by switching the nested iterative statements, making sure to adjust the bounds appropriately. Additional modifications can be made to traverse rows or columns in different ways, such as back and forth or up and down. The execution of the outer loop can impact the values and ranges of the inner loop.

Check Your Understanding

Once your program has been implemented and tested, answer the following questions on your own:

1. Describe the development process used in the completion of the project.

2. Provide the method header for one method that you implemented that takes at least one parameter. Explain why you chose the given parameters, including type, and why you made the method static or non-static. How would your code have been affected if you had made a different decision?

3. Provide a code segment where the elements in a data structure are traversed. Other than specific syntax, explain how using a different data structure would change the complexity of your code. Provide an equivalent code segment to the one included above that uses a different data structure.