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?

Yes, definitely, as the sizes of each image are different in length.

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.

The arch.jpg could be resized to a length of 640 rather than 360, and therefore the 2d array of pixels would have to be extended in length by 280.

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?

233, 172, 95

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

64, 0, 192

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

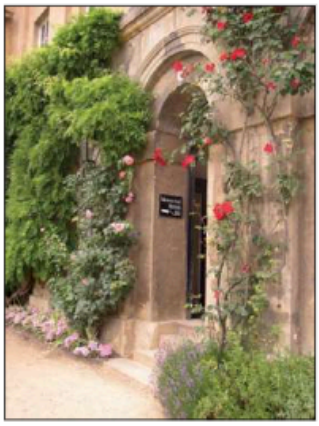
Because the highest two bits will change the entire image drastically. However, the lower two bits have minimal impact on the overall image, while storing data.

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?

Because the highest two bits hold the biggest, or more significant value, and are a stronger representation of the original hidden 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?

The values of the rgb measurements are all rounded down to a nearby multiple of 64, which results in nearby values getting clumped into similar combinations of colors.



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?

The first 3 digits of the secret image could be pushed into the first 3 digits of the combined image. This would make the combined image slightly more different from its original, but store more data for the secret image.

```
/* Determines whether secret can be hidden in source, which is * true if source and secret are the same dimensions.
 * -k @param source is not null * @param secret is not nu I -k @return true if secret can be hidden in source, false otherwise.
 */
public static boolean canHide (Picture source, Picture secret)
{
    return (source.getHeight()==secret.getHeight() && source.getWidth()==secret.getWidth());
}
/* Creates a new Picture with data from secret hidden in data from source *
 * @param source is not nu 1 *
 * @param secret is not null *
 * @return combined Picture with secret hidden in source
 * @author AKSHAT GARG 11.27.24
 * precondition: source is same width and height as secret -k
 */
public static Picture hidePicture (Picture source, Picture secret){
    Pixel[][] pixels = source.getPixels2D();
    Pixel[][] secpix = secret.getPixels2D();
    testClearLow(source);
    for (int r = 0; r < pixels.length; r++)
    {
        for (int c = 0; c < pixels[0].length; c++) {
            Color col = secpix[r][c].getColor();
            Color orig_col = pixels[r][c].getColor();
            pixels[r][c].setRed(col.getRed()/64+orig_col.getRed());
            pixels[r][c].setGreen(col.getGreen()/64+orig_col.getGreen());
            pixels[r][c].setBlue(col.getBlue()/64+orig_col.getBlue());
        }
    }
    return source;
}
```