

cs171, Introduction to Programming
Final Project: Steganography Application

200 points

Overview:

People use cryptography to send secret messages to one another without a third party overseeing the message. Steganography is a type of cryptography in which the secret message is hidden in a digital picture. Think of all those pixels in an image and each pixel has three color numbers — there are zillions of numbers in an image. If you were to change a few of these color numbers the resulting picture would probably look a lot like the original image; in fact, most people probably couldn't tell that you had changed the image at all. Steganography works by changing a few pixel color values; we will use selected pixel values to represent characters instead of a color value. Ofcourse, the resulting image will still look mostly like the original except that a few tiny "blips" might seem a little out of place if you look very closely. We can then send the image to a buddy and they can extract the message if they know which pixels to decode.

In this assignment you will be writing a java application that will enable you to exchange secret messages with another person.

Skills covered in this assignment:

Here are some of the skills you will need to learn in order to complete this assignment:

- Loading and saving bitmap files.
- Bit manipulation operations.

Requirements

- Project is to be submitted on email by 11:30am on Monday, May 1.

- You are to create an application called `Steganography.java`. All your code will be in this file. This is what you will submit on email.
- Your project is to work with the standard (original) `Picture.java` class. You shouldn't need any changes to this class in order to make your project work. You will not be submitting a `Picture.java` file. Instead, I will use my copy to run your program.
- There is a file `Secret.bmp` on the class web page. Encoded in this file is a question. Use your program to decode the message. Answer the question (in 255 chars or less). Then submit back to me your response encoded in a different bmp picture. You are to copy this bmp file in your file on the shared drive (before 11:30am, May 1). Of course, make sure your own program can decode the response you put in this picture; that way you can be sure that my program can decode the response too.

Bitmap Files

- First you will need to read your picture as a jpg and then save it in 24-bit bmp format. You will need to use bmp files for this assignment because jpg's are "lossy" meaning that what you write to the file may be changed slightly so that the resulting image can be stored more efficiently. Thus jpg will not work for steganography because jpgs will change the secret message when storing the file to disk. Here are the commands to save your file. You can give it the same name except be sure to put a `.bmp` file extension on the end. (For example, I loaded "Matt.jpg" and then saved "Matt.bmp").

```
> Picture p = new Picture(FileChooser.pickAFile());
> p = p.halve().halve();
> p.saveBMP(FileChooser.pickSaveFile());
```

- There is also a `loadBMP` method. You can probably guess how this works.
- Note that I reduced my image to $\frac{1}{4}$ original size because bmp files take a lot of memory. You will run in to less trouble if your image is smaller (say 100x100 or less).

Bit Manipulation

- You will need to be able to manipulate the bits stored in numbers. There are three basic bit manipulation operations: and, or, and shift. You will need all three.
- See the `BitExample.java` example to see how to use these different operations.

Interaction

- Prompt the user if they want to encode or decode a message.
- Use the `FileChooser` dialog to prompt the user for an input file.
- If encode, prompt the user for an input message. Encode the message into the picture (details below). Then use the `FileChooser` dialog to prompt the user for an output file. Save the new picture/message in this file (using `bmp` format).
- If decode, extract the message from the file. Print the message.

Encoding/Decoding Method

- You can extract the pixels of your target picture in one big array using the `texttgetPixels()` method.
- Use the first pixel (at spot 0) to hide the length of your message (number of characters). You will limit yourself to messages that are between 0 and 255 characters long.
- After that use every eleventh pixel to hide characters in your message. Start at pixel 11, then pixel 22, and so on until you hide all characters in your message.
- Every thing that you need to hide in a pixel is 8-bits long. The length (in the first pixel) is a byte. You can typecast all the unicode chars to bytes as well.
- Use the method below to hide each byte in an appropriate pixel.

Hiding Method

The problem with changing the red values in our encode/decode steps, is that these often cause quite visible changes in the resulting image. This is especially true if the pixels that are being changed are part of a large section of uniformly colored pixels – the "dots" stand out and are noticeable. As an option, we can change only the lower order bits of each pixel color (red, blue, and green). This will make subtle changes to each pixel's color and will not be as evident.

Remember that each pixel has three bytes: one byte for red, blue and green colors. Each byte has 8 bits to encode a number between 0 and 255. When we swap out the red color byte for a character, it is possible that we are changing the redness of that pixel by quite a bit. For example, we might have had a pixel with values of (225, 100, 100) which has lots of red, some green and some blue – this is basically a reddish pixel with a slight bit of pink color to it. Now suppose we are to store the character "a" in the red part of this pixel. An "a" is encoded as decimal number 97 so our new pixel becomes (97, 100, 100). Now we have equal parts of all three colors to produce a dark grey pixel. This dark grey is noticeably different than the dark pink we had before; it will definitely stand out in the image especially if the other nearby pixels are all dark pink.

We want a way to encode our message without making such drastic changes to the colors in the original image. If we only change the lowest bits of each pixel, then the numeric values can only change by a small percentage. For example, suppose we only change the last three bits (lowest three bits) – these are the bits that determine the "ones place", the "twos place" and the "fours place". We can only alter the original pixel color value by ± 7 . Let us think of our original pixel as a bits:

$$(r_7 r_6 r_5 r_4 r_3 r_2 r_1 r_0, g_7 g_6 g_5 g_4 g_3 g_2 g_1 g_0, b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0).$$

And our character (byte) as some bits:

$$c_7 c_6 c_5 c_4 c_3 c_2 c_1 c_0.$$

Then we can place three of these character bits in the lowest red pixel, three more in the lowest green pixel, and the last two in the lowest blue pixel as follows:

$(r_7 r_6 r_5 r_4 r_3 c_7 c_6 c_5, g_7 g_6 g_5 g_4 g_3 c_4 c_3 c_2, b_7 b_6 b_5 b_4 b_3 b_2 c_1 c_0)$.

If we had done this to the example of pixel (225, 100, 100) with character "a", we obtain:

```
original pixel = ( 11100001, 01100100, 01100100 )
"a"           = 01100001
new pixel     = ( 11100011, 01100000, 01100101 )
new pixel     = ( 227, 96, 101 )
```

Notice the new pixel of (227, 96, 101) is almost the same value as the old pixel of (225, 100, 100). There will be no noticeable color difference in the image! To retrieve the message, you simply extract the appropriate pixels from the rgb values to reconstruct the secret character.

To accomplish this, you will need to be handy with the "logical and" and "logical or" operators and also the "shifting" operator. Obtain a java reference book to research these operations. You might want to test them out on a small program first or on the Dr. Java command line.