CS100J Fall 2007 Assignment A8. Steganography Due 30 November

1. Introduction

This assignment is an extension of the last assignment, A7. In this assignment, you will add one more feature to that program: hiding (and then retrieving) a text message in a jpg file!

You may work with one other person, as usual. It is expected that you will work together, designing and implement this assignment. It is dishonest for each of you to work independently on separate pieces of the assignment and then put the pieces together.

2. Steganography"

According to Wikipedia (en.wikipedia.org/wiki/Steganography), steganography "is the art and science of writing hidden messages in such a way that no one apart from the intended recipient even realizes there is a hidden message." In contrast, in cryptography, the existence of the message is not disguised but the content is obscured. Quite often, steganography deals with messages hidden in pictures.

To hide a message, each character of the message can be hidden in one or two pixels of an image, by modifying the red, green, and blue values for the pixel(s) so slightly that the change is not noticeable.

Each character is represented using the American Standard Code for Information Interchange (ASCII) as a three-digit integer. We allow only characters that can be represented in ASCII —all the obvious characters you can type on your keyboard are ASCII characters. See page 6.5 of the ProgramLive CD for a discussion of ASCII.

For the normal letters, digits, and other keyboard characters like `$` and `@`, you can get its ASCII representation simply by casting it to **int**. For example, `(int)'B'` evaluates to the integer 66, while `(int)'k'` evaluates to 107.

We can hide character `'k'` in a pixel whose RGB values are 199, 222, and 142 by replacing the least significant digit of each color component by a digit of the integer representation 107 of `'k'`:

Original pixel				**Pixel with 'k' hidden**		
Red	Green	Blue	hide `'k'`, which is 107	Red	Green	Blue
199	222	142	⟶	19**1**	22**0**	14**7**

This change in each pixel is so slight that it will not —cannot— be noticed just by looking at the image.

Decoding the message, the reverse process, requires extracting the last digit of each color value of a pixel and forming the ASCII value of a character from the three extracted values. In the above diagram to the right, extract 1, 0, and 7 to form 107, and cast this integer to **char**.

Extracting the message does *not change the image*. The message stays in the image forever.

Three problems for you to solve. You will write code to store a message `m` in the pixels of the image in row-major order, starting with pixel 0, 1, 2, … Think about the following issues and solve them.

1. You need some way to recognize that the image actually contains a message. Thus, you need to place something in pixels 0, 1, 2, … that has very little chance of appearing in a real image. You can't ever be *sure* that an image without a message doesn't start with those pixels, but the chances should be small.

2. You have to know where the message ends. You can do this in several ways —place the length of the message in the first pixels in some way (in which case you will want to restrict the length of the message to something reasonable), put some unused character at the beginning and end of the message, or use some other scheme.

3. The largest value of a color component (e.g. blue) is 255. Suppose the blue component is 252 and you try to hide 107 in this pixel; the blue component should be changed to 257, but this impossible because a color components are ≤ 255. There are several ways to solve this problem, including using *two* pixels for each character of the message. Each solution has its advantages and disadvantages. We ask you to think about this problem, come up with at least two ways to solve the problem, and implement one of them.

As you can see, this assignment is a bit less defined than the previous ones. *You* get to solve some little problems yourself. Part of this assignment will be to document and discuss your solutions.

Your task

(a) Decide on how you will solve the problems mentioned in points 1, 2, and 3 given above. As you design and implement this assignment, write a short essay that documents at least two solutions to each of the three problems mentioned above, discusses their advantages and disadvantages, and indicates what your solutions are. Advantages could be: easiest to implement, least amount of pixels used in hiding an image, least time spent in hiding a message — whatever. When you are finished with this assignment, insert this essay as a comment at the beginning of class `ImageMaintainer`.

Feel free to discuss points 1, 2, and 3 with TAs or consultants. They will not tell you *how* to solve these problems. But they will discuss your ideas with you.

(b) Complete the body of procedures `hide` and `reveal` in class `ImageMaintainer`. These two methods will hide a message and reveal the message in the jpg image. When you design method `reveal`, make sure it attempts to extract the message only if its presence can be detected. (Note that the spec for reveal should say "Return null if no message detected." and not "Return null if no image detected.").

(c) Look at the declaration of fields in class `ImageJFrame`. You will see two buttons: `buttonhide` and `buttonreveal`. Scroll down to method setup and look at the method calls under "Build box buttonBox of buttons". Insert method calls to add buttons `buttonhide` and `buttonreveal` to `buttonBox`. Then, in the next set of instructions, register "this" class as an "action listener" for these two buttons.

Now, when you create a new `ImageJFrame` object, you should see the hide and reveal buttons. The message that is hidden is the string of text that appears in the text area just below the images in the GUI. Take a look at procedure `actionPerformed` and see what it does when one these two buttons are clicked.

Submit both files `ImageJFrame` and `ImageMaintainer` on the CMS by the due date —after inserting your essay as a comment at the beginning of `ImageMaintainer`.

Writing, testing and debugging your code

Methods `hide` and `reveal` can be difficult to debug. Here some hints to help you.

1. We encourage writing other methods besides `hide` and `reveal`. You get to decide which ones to write as you design and implement this assignment. Be sure to specify each method appropriately in a comment before it.
2. It will be difficult to use a JUnit testing class here, because it will be difficult to get it to access a picture appropriately. You do not have to use one.
3. Use function `ImageMap.toString(pixel)` to get a readable String representation of a pixel.
4. Do *not* assume that you can debug simply by calling `hide` and then `reveal` to see whether the message comes out correctly. The best thing to do is to write method `hide` and debug it, before going on to `reveal`.
5. In the methods that you write, insert `System.out.println(…);` statements to print out information, so that you can see what your code you is doing. This is the easiest way to determine whether your code is working correctly. But you will have to continually change these statements in order to keep the amount of output to something manageable.
6. Start with extremely short messages to hide —1, 2, or 3 characters— and first check that the initial pixels, which are supposed to indicate that a message is hidden, are correct.