COMP 1600 Fall 2025

Lab 15: Steganography

Due by the end of class

The goal of this lab is to make a tool that can read secret messages hidden inside of image files. The art of encoding and decoding such secret messages is called steganography, and Wikipedia has a decent article about it **here**. The advantage of hiding a message this way is that, unlike some more obvious form of code or encryption, a potential eavesdropper will not even suspect that a message is being transferred.

Specification

Create a project called `Lab15`. Add a `SecretReader` class (with a `main()` method) as well as a `Picture` class. Delete everything inside the `Picture` class and paste in the code from **Picture.java**.

Input

Inside of the `main()` method in the `SecretReader` class, prompt the user to enter the name of the image file to read from. Read in this `String` and create a new `Picture` object using the file name in the constructor.

Revealing the Secret Text

The secret information is stored in the last bit of the red, green, and blue values for every pixel (whether or not those values are odd or even). If we consider two pixels, then there are six components to check for oddness or evenness. $2^6 = 64$, which is more than enough different possibilities for us to encode the 26 letters of the alphabet.

How do we get at it? First, make sure that you follow directions very, **very** carefully. This lab has a lot of technical details, and the output won't even be close with the smallest error. Make an `int total` variable and initialize it to zero. Also, make an `int count` variable and initialize it to zero.

Make two nested `for`-loops that loop over every pixel. The outer loop should loop over columns, and the inner loop should loop over rows. Get the `Color` value for the current pixel. Then, extract the red, green, and blue values from the `Color` object using the `getRed()`, `getGreen()`, and `getBlue()` methods.

Now comes the tricky part: We want to extract the 0 or 1 (the evenness or oddness) from the last bit of each color component and use those 0s and 1s to reconstruct a number between 0 and 63. To do

so, double the current value of `total`. Then, add to `total` the value of the red component modded by 2. Doing so will put that 1 or 0 into the value of `total`. Once again, double the current value of `total`. Doing so moves the 1 or 0 from the red up into the next place in its binary representation. Then, add to `total` the value of the green component modded by 2. Yet another time, double the current value of `total`. Then, add to `total` the value of the blue component modded by 2. Finally, add one to the current value of `count`.

Remember that we use **two** pixels to store all this information. That's why we use `count`. After you've done all of the above in the inner loop, check `count`. If `count` is even, then you've stored up data from two pixels into `total`. Only in that case, check the value of `total`. If `total` is less than 26, it's the representation of a letter. Remember that `char` values for letters are **not** from 0 to 25, they are from `'a'` to `'z'`. So, if `total` is less than 26, add `'a'` to the value, cast it to a `char` and output it to the screen. If it is exactly 26, print out a space. Otherwise, do nothing. (If we don't use all the pixels in an image for storing text, some will be leftover. The tool I used to hide data in the images set all the other pixels to make values larger than 26. By ignoring these values, we get only the message and not other gibberish.)

Once you have printed out a letter, a space, or ignored the value of `total` completely, set `total` back to zero. Remember: Only do this when you do the output step, that is, when `count` is even.

Output

After all that headache, the message should be output in all lower case, with only spaces and no other punctuation.

You can download the following image files for testing. Right-click each link and save the target. Make sure you put them in the `Lab15` directory.

- **secret1.png**
- **secret2.png**
- **secret3.png**

Here is sample output for the program on the file `secret1.png`. Please try to match the output formatting exactly.

```
What file would you like to read a secret from? secret1.png
if you diss doctor dre you diss yourself
```

Here is sample output for the program on the file `secret2.png`. Please try to match the output formatting exactly.

```
What file would you like to read a secret from? secret2.png
to be or not to be that is the question whether tis nobler in the mind to
suffer the slings and arrows of outrageous fortune or to take arms against a
sea of troubles and by opposing end them to die to sleep
```

You will have to get your reader working yourself to find out what the secret message in `secret3.png` is.

Turn In

Turn in your code by uploading `SecretReader.java` from the `Lab15\src` folder wherever you created your project to **Brightspace**. **Do not** upload the entire project. The `Picture.java` file is unnecessary. I only want the `SecretReader.java` file.

All work must be done individually. Never look at someone else's code. Please refer to the course policies if you have any questions about academic integrity. If you have trouble with the assignment, I am always available for assistance.