Information On Running The Programs

(First: as it turns out, WhichImage.java still needs work. However, the others should be test-ready.)

No command line information is needed; the programs will prompt you for all necessary file names.

The image/text files in the base directory are provided to allow easy testing of the various programs. You may use other files as desired. However, note that not all images are well-suited for steganography. For example, an image with large areas of pure white will degrade quickly when data is masked within it. Images with a wide range of colors used are preferable.

Also note that these programs do not support the masking of data in images with transparencies.

Here are details on the available programs:

- HideTextInImage.java/HideImageInImage.java:
  - → The user is prompted for the name of an image file which will serve as host for hidden text/image data, then for the text/image file containing the data to be hidden.
    - If the string entered does not correspond to an actual file, the user is prompted to try again until an actual file is located.
  - → The data may be hidden in one, two, or three of the least significant bits, and in one of five pixel patterns (every pixel, every even-numbered pixel, every odd-numbered pixel, every third pixel, every prime-numbered pixel).
  - → Each of these requires a different amount of space, and so some require a larger image than others.
  - → The program will calculate which combinations of LSB and pixels used will fit in the image file chosen. If none will fit, the user is informed of such and the program will exit. Otherwise, the user is prompted to choose a combination from a menu.

- In all menu selection scenarios, as well as all yes/no prompts, only the first character of the user response is considered. This is known and, for the moment, considered acceptable.

→ The user is then prompted to enter the name of the file to which the image (now containing hidden text) will be written. The suffix ".png" is automatically appended to the user's input.

- If this will lead to an extant file being overwritten, the user is given the chance to choose a different name.

→ The new file is created. The user has successfully hidden data in an image file.

➢ RetrieveTextFromImage.java/RetrieveImageFromImage.java:

→ User is prompted for the name of an image file from which data will be extracted.

- If the string entered does not correspond to an actual file, the user is prompted to try again until an actual file is located.

→ The user is then prompted to enter the name of the file to which the extracted text will be written. The suffix ".txt" (or ".png") is automatically appended to the user's input.

- If this will lead to an extant file being overwritten, the user is given the chance to choose a different name.

→ User is prompted to choose yes/no on whether to use an "ignorant search," which searches every pixel and only one least significant bit. For demonstration purposes only – to accurately retrieve data, choose "no."

→ The data is extracted to the destination chosen.

➢ WhichImage.java

→ Determines which image file is most likely to contain masked text.

→ User enters image files one at a time, then enters "stop" to stop adding images.

- Invalid file names are ignored, and the user is informed that no such file exists.

→ Each image is tested. Twelve possible combinations are checked – three (the number of least significant bit patterns used) times four (the number of pixel patterns used). This program does *not* currently test for images in which data has been hidden in the prime-numbered pixels.

→ For each pattern, the ratio of letters, numbers, and common punctuation marks to total characters is determined. The highest ratio for each image is reported to the user. Ideally, it will be clear which image(s) contain masked text and which do not.

Known/potential issues:

- When taking user input for yes/no decisions, or when the user is selecting from a menu, only the first character of the input is considered. If this proves problematic, it can be changed.
- Some of these programs can run out of heap memory space, especially when large images are being used. If this occurs, run the program with "-Xmx1g" following the word "java" and the name of the program.
- When encoding text files, only ASCII characters are considered. This is unlikely to be changed.
- Text files created in Linux may be missing line breaks when unmasked in a Windows environment. This may or may not be addressed.

Features not yet implemented:

- The ability to encrypt text files with single-character substitution before hiding data in an image.
- The ability of WhichText.java to test for text hidden in only the prime-numbered pixels.
- The ability of WhichText.java to test for text encrypted with single-character substitution.