What Stayed the Same:

At the lowest level, the program did not change. `Huffman`, the class which contains the methods for the creation of a Huffman tree, as well as implementing a tree for compression/expansion, functions as we planned. Having a design for the implementation of the Huffman compression algorithm was particularly helpful because it meant that we spent less time trying to decide on and create an implementation of the compression algorithm. Instead, that time went into trying to design a user-friendly program which would simply draw on the Huffman algorithm to compress a file.

What Changed:

Nearly everywhere else, the program was largely different. At the highest level, the GUI was simpler than we imagined. We wanted to have options to either select a file to compress using Finder (File Explorer on Windows), or to simply drag-and-drop a file into the program. Instead, the program only allows the user to select a file using Finder, because JavaFX does not have a mechanism to allow us to code in the drag-and-drop function.

That being said, we also added to the GUI. There is an option to have the program also generate a visual representation of the Huffman tree when performing a compression/expansion, which just lets an educated user see what the program is actually using to compress the file. We also added a log to the program, which displays the both the progression of the any compression or expansion actions, as well as error messages. We found this useful because it gives the program a way to express user-input errors, aside from the terminal. Having a log lets the user figure out why the program is unable to do what the user wants.

Another change was the addition of the `ColdenTab` class. Essentially this class performs all the functions we wanted `Compress` and `Expand` to do, except for the actual execution of the Huffman compression. `ColdenTab` serves as the parent of `Compress` and `Expand`, since both classes are so similar in their functionality, except for the direction in which they apply the Huffman compression. The existence of this class is particularly convenient because it means we don't have to write the same code twice.

The class `Artifact` was added to assist with the compression in `Compress` and `Expand`. This class contains methods that are used in writing the files generated by the algorithm. Having the methods defined in `Artifact` made the process of defining the compression and decompression methods much easier.

Similarly, the class `BitArray` was added to supplement the functionality of `Huffman`. Because the Huffman compression is dependent conserving as much physical space as possible, it relies heavily on storing information in bit and byte form. `BitArray` stores an `ArrayList`

of `Bytes`, as well as a number of methods for interacting with that data structure. Having `BitArray` objects makes the creation of the Huffman tree a much easier task.

Finally, the last deviation from our original project design is the `Graph` class. It's a very simple class, which takes in a `Huffman`, and draws that tree in a PDF using GraphViz, an external program. The user decides whether, as a part of the compression process, the program generates the Huffman tree as a PDF in addition to the compressed file. Having this class adds to the features the program can offer, although it is more tailored towards those who understand the algorithm implemented by the program.