CIS 573 Software Engineering - Homework #6

Siddharth Srivatsa

1.
I had not written an Junit tests during HW1. I implemented two tests to check the two methods in my code – The function to encrypt a file, and the function to read a file. Both these methods take in a file as an input. A test file has been attached with the uploaded zip folder which needs to be placed in the src folder of the project. I have attached the test case in the zip folder.

2.
a) I did not know how to catch exceptions using try blocks, and I had used if statements instead, manually trying to catch exceptions. An easy way to fix this would be to use try and catch blocks to handle exceptions. (In the main function, Line 30). By doing this, I will not be checking for each condition manually and I will let the try anc catch handle exceptions if they occur. This will reduce execution time, and also make it better code.

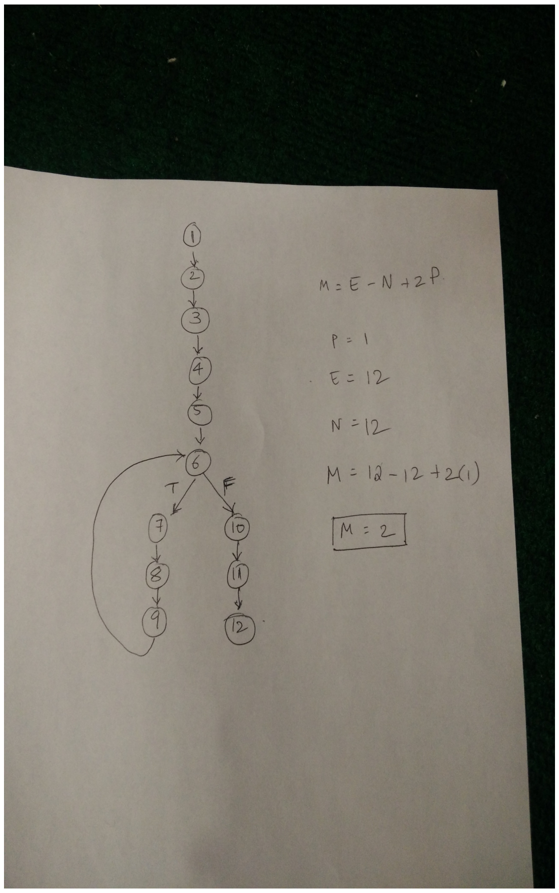
```
            // Check if path of the folder is valid
            if(!folder.isDirectory()) {
                  System.out.println("Invalid path, Exiting program");
                  System.exit(0);
            }

            // Check if the folder is empty
            if(folder.list().length<1) {
                  System.out.println("Empty directory, Exiting program");
                  System.exit(0);
            }

            if(!folder.canRead()) {
                  System.out.println("Folder cannot be read, Exiting program");
                  System.exit(0);
            }

            // Check if number of arguments is 1
            if(args.length != 1)
          {
              System.out.println("Invalid number of arguments, Exiting program");
              System.exit(0);
          }
```

b) While mapping from frequency model to decrypt the encrypted file, I did it very inefficiently. I built a new map so that I could compare between the counts of the model and the file to decrypt. This was a problem because the counts were values in the original map, and I did not know how to sort the map based on the values instead of keys. This was very inefficient, since I had to loop through the original map and re-build a new map by interchanging the values and keys. Instead I should have used a method to iterate over the values.
I cannot point to the exact line of code since this happens at different functions – Lines 182, and 92.

```
static <K,V extends Comparable<? super V>>
SortedSet<Map.Entry<K,V>> entriesSortedByValues(Map<K,V> map) {
                SortedSet<Map.Entry<K,V>> sortedEntries = new
TreeSet<Map.Entry<K,V>>(
                    new Comparator<Map.Entry<K,V>>() {
                        @Override public int compare(Map.Entry<K,V> e1,
Map.Entry<K,V> e2) {
                            int res = e1.getValue().compareTo(e2.getValue());
                            return res != 0 ? res : 1;
                        }
                    }
                );
                sortedEntries.addAll(map.entrySet());
                return sortedEntries;
            }
```

3.
a) Since I have used only a single class, I calculated the cohesion of the methods and the fields in that class. I used 6 methods and 3 fields, giving A = and M = 6. With this I calculated the cohesion using LCOM to be 0.7333. From this I can tell that there is a high lack of cohesion in my code which is not good. I should have made the code more cohesive to reduce the value of LCOM.

b) I used the "McCabe metric" for the fileToStr Method (Line 256). I got a value of 2. (Please see attached image) This means that if someone else implements the same method we can use the metric to comapre the two implementations. As it is this number is abstract and does not make sense.



4.

a) Did not adhere to Java conventions – (Variable names started with capital letter `File NextFile`)

b) Non intuitive method and variable names -
- `(Line 225) getError(File originalFile, String decrypted)` was the method for calculating the number of correctly decrypted characters)
-`(Line 259) BufferReader br = new BufferedReader(new FileReader(readFile))` as the file reader for reading each file. Could have been f
-`(Line 260) StringBuilder sb = new StringBuilder()` as the data type for concatenating lines from the file to make a new string
- `(Line 66) Map<Character, Integer> freqModelInv = new TreeMap<Character, Integer>();` was the name used for the inverted frequency model.

c) (Line 232) Incorrect spacing and indentation (`for (int i=0; i<original.length(); i++)`)

d) Useless comments, and random pieces of code commented out. I had tried few things while implementing my algorithm and I had not removed them, at many places there were also comments which did not really help with the readability or the understandability of the code. Rather, they hurt it more.

5.

a) Message chain – `(Line 82 and Line 84)`
```
freqModel.put(entry.getValue(),entry.getKey());
freqModelInv.put(entry.getKey(),freqModelInv.get(entry.getKey())
+entry.getValue());

Refactoring -
letter = entry.getKey();
count = entry.getValue();
freqModel.put(count, letter);

letter = entry.getKey();
count = entry.getKey + freqModelInv.get(letter)
freqModel.put(count, letter);
```

b) Primitive obsession – Used map to store the model. This resulted in me using an inverted model to build it using the frequency counts. After that I had to flip the keys and values so that I could iterate through the model based on the counts while decrypting.
```
Map<Character, Integer> freqModelInv = new TreeMap<Character, Integer>();
Map<Integer, Character> freqModel = new TreeMap<Integer, Character>();

Refactoring -
Should have used a different map or a custom data type to store the letters with
their counts and iterate more efficiently over both keys and values easily.
```

c) Duplicate code – There are several places where code has been duplicated and I could have used a method for doing the same thing.
Eg Line 129, 156, 186, 216, 235
```
      if (Character.isUpperCase(c)) {
            c = Character.toLowerCase(c);
      }
```

```
Refactoring – Extract method. I should have created a separate method that checked
if the character was uppercase and returned the lowercase character.
```

d) Long class – I used a single class for all operations. In hindsight, I should have used as many classes as possible – One for the main function to call the other methods, one class for reading a file, one class for encrypting a file, one class for building the model, one class for decrypting the file.