Criterion C - Development

Table of Contents

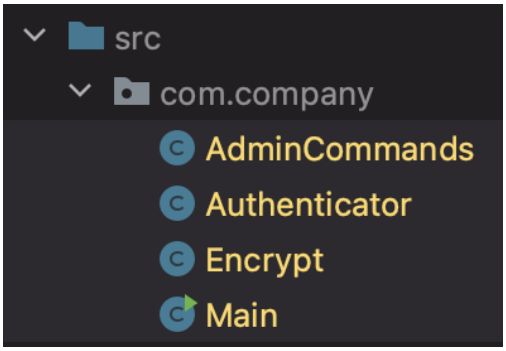
Encryption/Decryption

The most complex part of my program is the encryption and decryption system. Although there are Java libraries that can encrypt and decrypt using built-in methods, I decided to make my own unique text scrambling algorithm, so as to confuse anyone who might have a knowledge of how the importable Java encryption algorithm works. To get inspiration for my algorithm, I researched extensively on the internet to find other text-scrambling algorithms. The one that stood out to me the most was the Burrows-Wheeler Transform (BWT). The BWT rotates splits text into rows and columns, staggers the position of the columns by rotating them, and combines the text together again. I took inspiration from this for my own algorithm. I split the text taken from files into a 2-dimensional array, and then changed the positions of the letters as if I was flipping the square onto its side, with a 90º turn. Then, the text is combined and is scrambled in a way that is not legible to humans.

Classes

In the program I made for my client, I use 4 classes. The "Main" class acts as expected in Java programs, where the bulk of the code used to execute the other classes and run the main parts of the program is stored. The "AdminCommands" class holds the various methods I used to reduce the amount of code and repetition in the Main class, with some notable examples being "setUpFilenames" (which sets up the preexisting files in the host computer upon the first run of the program) and "writeFile" (which takes the content and the intended file path as input and writes the content to said file). The "Encrypt" class holds the "encode" and "decode" methods, which will be described in detail in the paragraph below. Finally, the "Authenticator" class provides the methods that check the username and password of the person trying to log in

against those of the list of authorized users, kept in separate files. Classes were used in this program to organize methods into different locations so that all of the methods don't pile up in Main.



Methods

A large part of my program is made up of methods. Asides from "setUpFilenames" and "writeFile", the two that I briefly touched on in the previous paragraph about classes, there are many more. In the "Encrypt" class, the two methods that perform the main function required by the client are stored: "encode" and "decode". Both methods take an input of the file path of the text file to take the content to encode/decode from and the path of the file to put the encoded/decoded content into. They then perform their intended function of encoding/decoding. I decided to use methods such as these in my program to simplify it for myself and other programmers and to make it less resource intensive on the host computer. Without the use of methods, I'd have to copy and paste the same 61-line encode method and the 58-line decode method several times throughout my program. This is applicable to all of the other methods I used, as I'd have thousands of lines of code if I had to duplicate the same chunk of code every time I needed to run something repetitive. Below is a screenshot of the encode method as an example.

```
static void encode(String startPath, String endPath) throws IOException {

    Path filePath = Path.of(startPath);
    String content = Files.readString(filePath);
    String[] splitContent = content.split("");
    int forLength = (int) Math.ceil(Math.sqrt(content.length()));
    String[][] originalArray = new String[forLength][forLength];
    String[][] scrambledArray = new String[forLength][forLength];

    int position = 0;
    for (int i = 0; i < forLength; i++) {

        for (int j = 0; j < forLength; j++) {

            if (position >= content.length()) {

                originalArray[i][j] = " ";

            } else {

                originalArray[i][j] = splitContent[position];
                position++;

            }

        }

    }

    position = 0;
    for (int i = 0; i < forLength; i++) {

        for (int j = 0; j < forLength; j++) {

            scrambledArray[i][j] = originalArray[j][i];
            position++;

        }

    }

    String output = "";
    for (int i = 0; i < forLength; i++) {

        for (int j = 0; j < forLength; j++) {

            output = output + scrambledArray[i][j];

        }

    }

    try (PrintWriter out = new PrintWriter(endPath, StandardCharsets.UTF_8)) {

        out.write(output);

    }

}
```

User Inputs

Based on what my client asked for, I decided that it was necessary to take user inputs in this program. In Java, the programming language used, a user input is taken by what is called a scanner. A scanner in Java is most often used to take input from the next line of a program's interface, used by the program asking a question to which the user responds on the line below. Scanners are necessary in this program to understand what the user intends to do, which I use to facilitate user's navigation through the program. They are also necessary when the user logs in, as they need to input their username and password to enter, as shown in the code snippet below.

```
System.out.println("\nPlease input the following
information to log in, or type \"end\" to end the
program.");

System.out.println("Username:");
Scanner usernameScanner = new Scanner(System.in);
String usernameString = usernameScanner.nextLine();

if (usernameString.equals("end")) System.exit(0);

System.out.println("Password:");
Scanner passwordScanner = new Scanner(System.in);
String passwordString = passwordScanner.nextLine();
```

ArrayLists

I used ArrayLists in my program as I needed a dynamic data type. A dynamic data type is one that doesn't get created with a set length (or space in memory), allowing an infinite number of items to be added, as long as the computer has the resources to handle it. The location where I used the ArrayList was to generate a list of filenames that would be used as parameters for all of my methods. I needed to use a dynamic data type in this scenario as my program has the option for admins to create new files. If I used a normal array, a static data type, it wouldn't work as I don't know how many files the end-user(s) will create, if any at all.

```
5 usages
public static List<String> filenames = new ArrayList<String>();
```

Nested Loops

Nested loops are a vital part of my program. Basically, a nested loop allows for 2-dimensional commands to be run, by looping fully through a row in the inner loop, while the outer loop triggers every time the full row finishes, switching columns. The most complex location where I used nested loops was in my "encode" and "decode" methods in my "Encrypt" class. I

encrypted the contents of the files by putting them into a 2-dimensional array, and then scrambled that up using nested loops that switched the positions of different items throughout the columns and rows. Attached below is an excerpt of the "encode" method that uses nested for loops.

```
int position = 0;
for (int i = 0; i < forLength; i++) {

    for (int j = 0; j < forLength; j++) {

        if (position >= content.length()) {

            originalArray[i][j] = " ";

        } else {

            originalArray[i][j] = splitContent[position];
            position++;

        }

    }

}
```

File Paths

To get the paths of files, I had to first import "java.nio.file.Files" and "java.nio.file.Path" into my program. With all of the text files being stored in a resource folder locally inside of the

program, the paths were easy to find. However, the imports were necessary to designate what was a file path and what wasn't inside of my program, allowing me to use paths as parameters of my methods.

```
static void decode(String startPath, String endPath) throws IOException {

    Path filePath = Path.of(startPath);
```

```
import java.nio.file.Files;
import java.nio.file.Path;
```

String Manipulation

As the basis of my program required turning the content of files into strings so that they could be used in the 2-dimensional arrays to be encoded and decoded, I had to manipulate strings many times. As shown in the example below, I took the content out of the file (using the file paths explained above), put it in a string, and split it into separate characters. Splitting the strings is classified as string manipulation. I had to do this as there was no other way for me to separate the characters of a string into arrays, as far as I was aware when programming the product.

```
Path filePath = Path.of(startPath);
String content = Files.readString(filePath);
String[] splitContent = content.split( regex: "");
int forLength = (int) Math.ceil(Math.sqrt(content.length()));
String[][] scrambledArray = new String[forLength][forLength];
String[][] originalArray = new String[forLength][forLength];
```