Alexander Labell
02/19/2022

I. Design Notes

GapBuffer

The Gap Buffer class loads a single line of a string. It formats the string based on the cursor position, the text to the left is pushed all the way to the left and the text to the right is pushed all the way to the right. The middle is empty space. I decided to create counters for left and right based on shifting and manipulation of both the cursor_pos and newly added characters/strings. I created a private method called format that formats the array based on the cursor_pos and the counters.

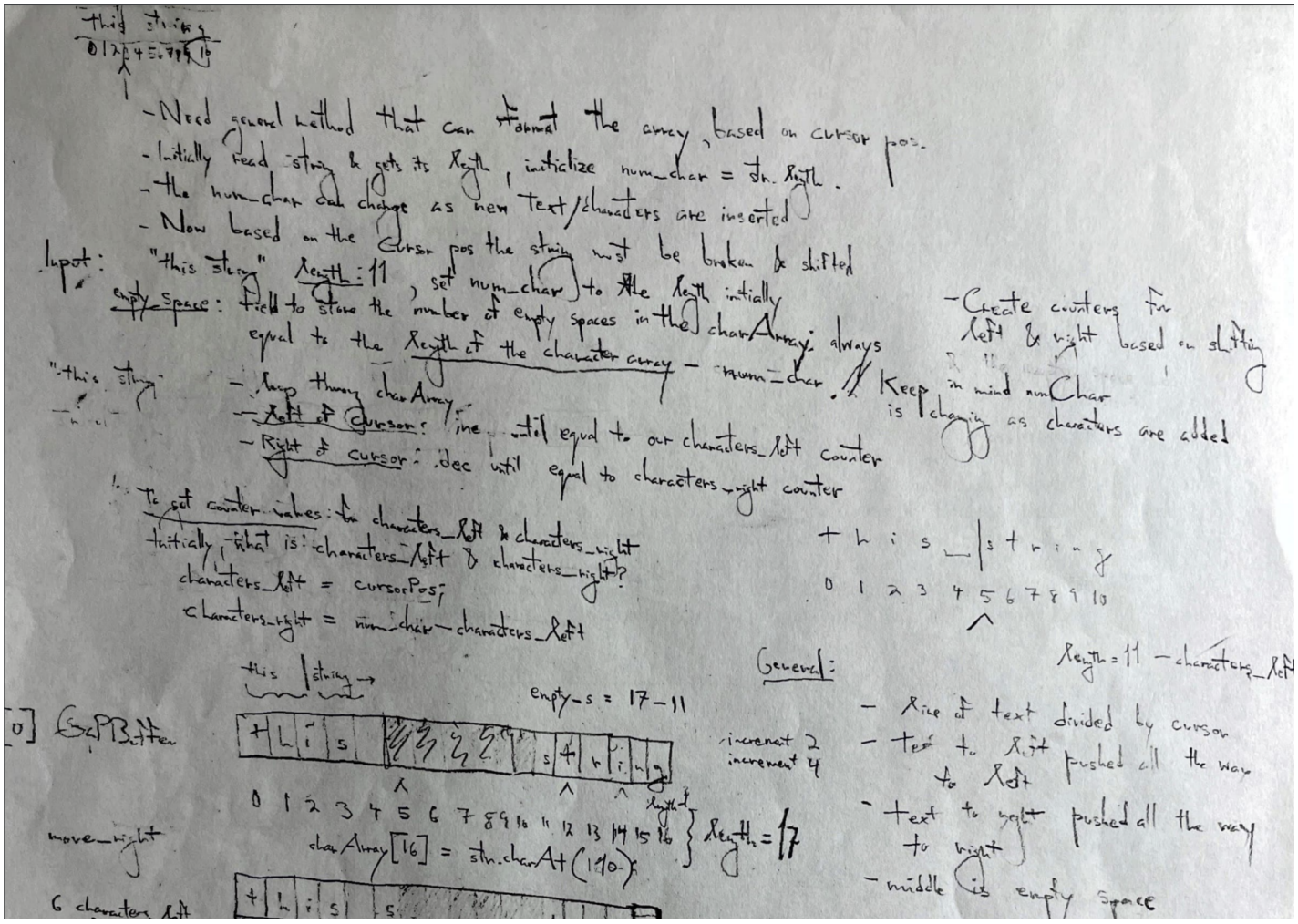


Figure 1 - Gap Buffer initial planning/design notes

Format Method

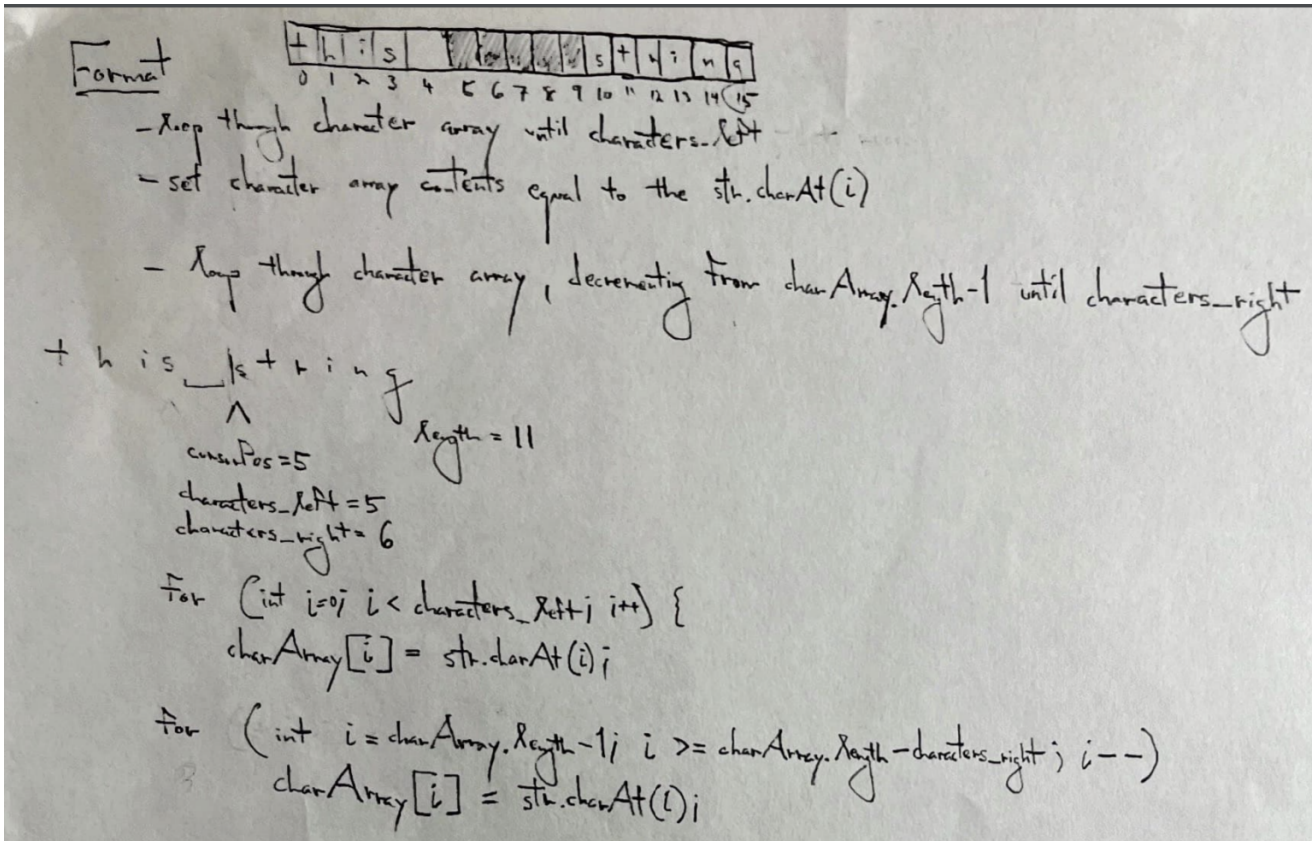


Figure 2 - Format method initial design

```
public void format() {
    for (int i=0; i<characters_left; i++) { //loop through character array until characters_left
        charArray[i] = str.charAt(i); //set character array contents equal to string on each iteration
    }
    //loop through character array, decrementing from charArray.length-1 until characters_right
    for (int i=charArray.length-1; i >= charArray.length-characters_right; i--) {
        charArray[i] = str.charAt(i-empty_space);
    }
}
```

Figure 3 - Format method source code

The format method is a private method, this method will format the charArray so that the correct characters appear at the beginning and end of the charArray. This method is important because it needs to be called in other methods as the cursor position is changed.

Movement Methods

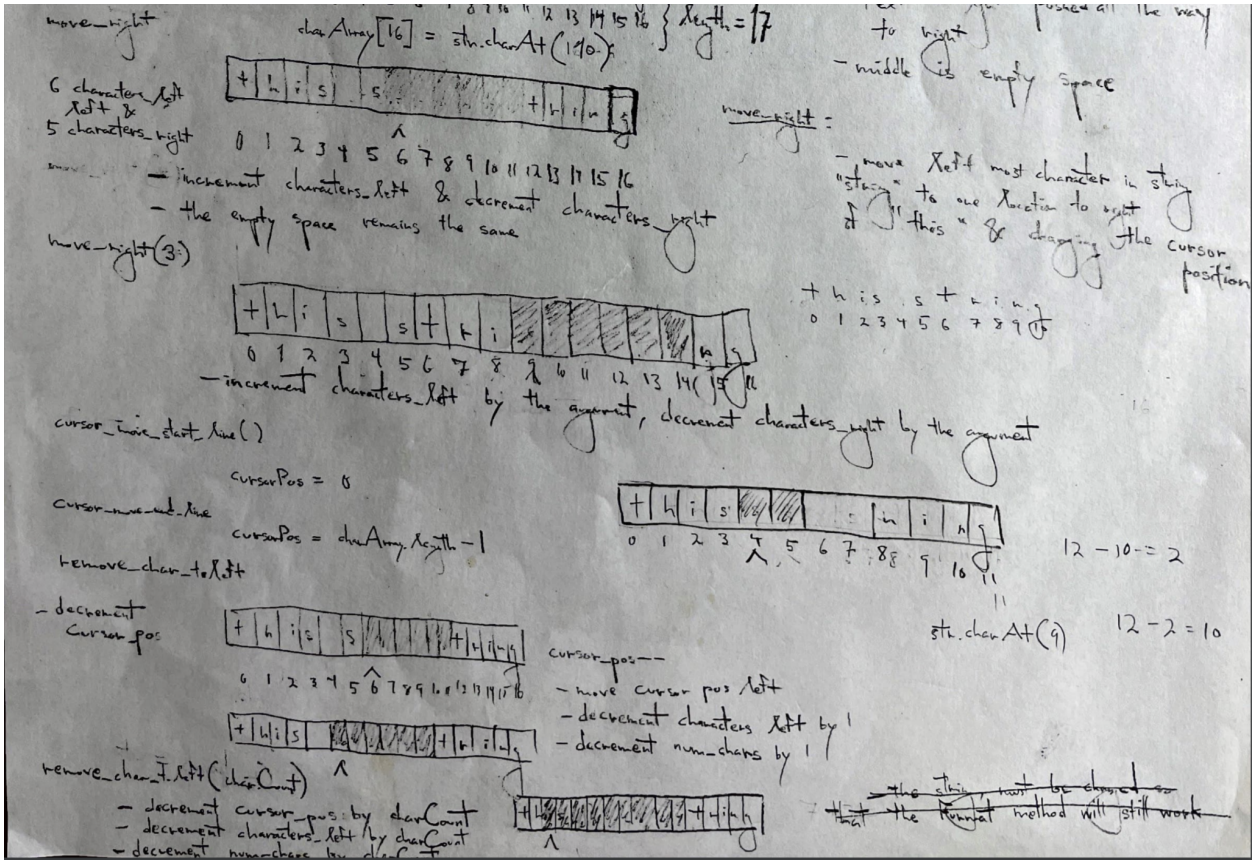


Figure 4 - Movement methods design

There were various cursor movement methods that were overriden and designed. In general they check if the cursor can be moved safely without generating an error. If the cursor will not fall out of the range then the cursor_pos (the field that holds the cursor's position) is changed and the characters to the left and right are manipulated. Finally the format method is called to ensure the string is organized inside the array properly. An example method is listed below.

```
@Override
public boolean cursor_left() {
    if (cursor_pos > 0) { //when we try to move the curs
        cursor_pos--;
        characters_left--;
        characters_right++;
        format(); //call format to format the charArray
        return true;
    }
    return false;
}
```

Figure 5 - cursor_left method

Removing Text Methods

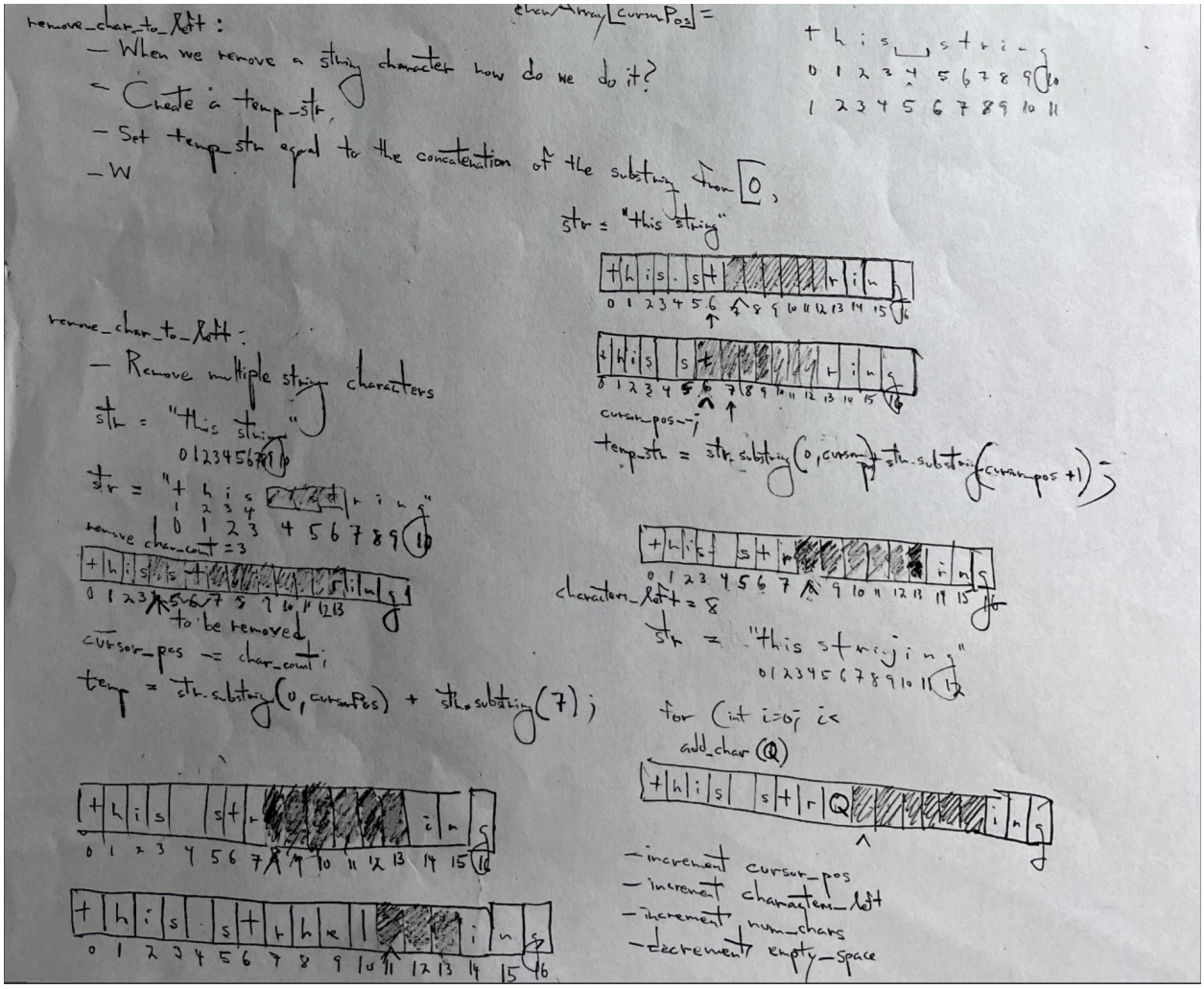


Figure 6 - Removing text methods design

For the methods to remove characters, first the cursor_pos is moved the number of characters to delete so that it is on the index of deletion. Next a temporary string is created and the stored string uses concatenation and the substring method to exlude the element(s) that should be deleted. The string reference is assigned the temporary string. Now, a character was deleted and therefore the characters_left should decrement, the number of characters in the string itself has decreased by the number of elements deleted and therefore num_chars is decreased and finally format is called to format the character array with the string correctly.

Adding Text Methods

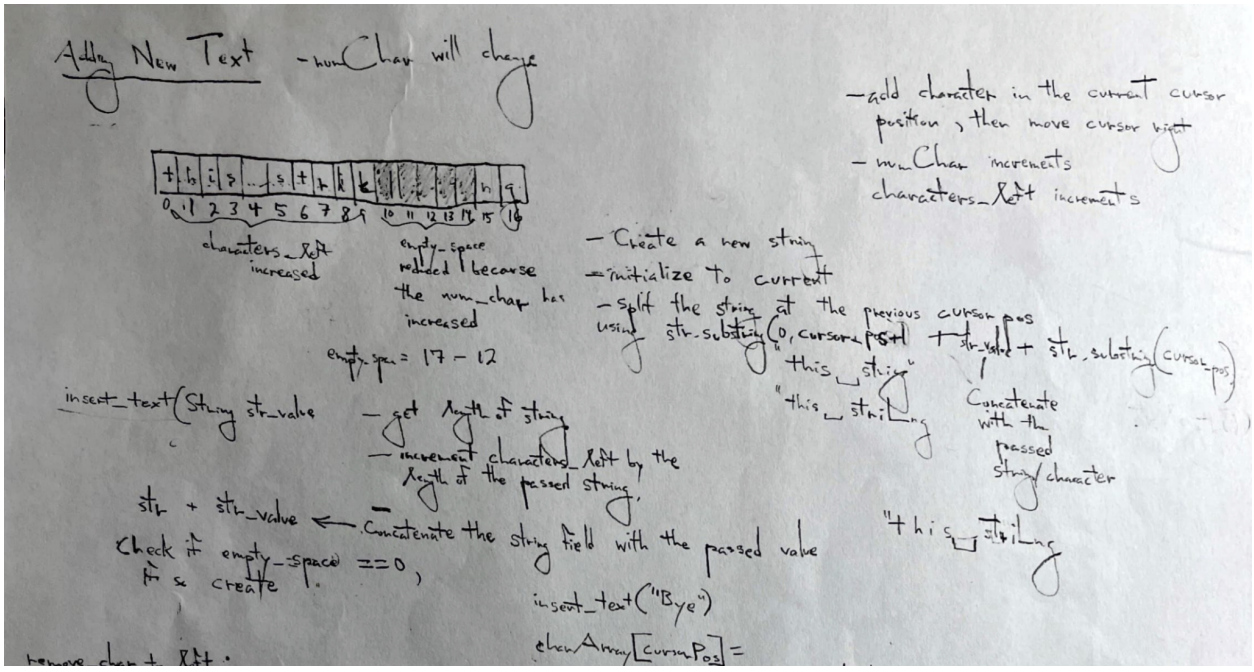


Figure 7 - Adding text methods design

For adding text, a new temp string is created. Now, the string field is split at the current cursor_pos and concatenated with the passed string. The number of characters added is increased by the length of the passed string argument. The string reference str is set to the temporary created string, now the field will reflect the new string. Next the empty space is decreased by the characters added. A conditional is created that checks if the empty_space is less than or equal to one. If so; we have overflowed, there must always be an empty space of 1 because that is where the cursor is stored, therefore false is returned. Otherwise, the str_value is added to the array at the cursor position using a for loop and adding characters one at a time. Finally, the cursor_pos should move to the right the number of characters added, the number of characters on left is incremented by the number of characters added, increment num_chars by the number of characters added.

Buffer Structure

BufferStructure is an object that holds an array of GapBuffer objects. It specifies a cursor_pos in the first dimension so that lines can be accessed and thus individual characters within those lines as we move into the 2nd dimension of the 2-D structure.

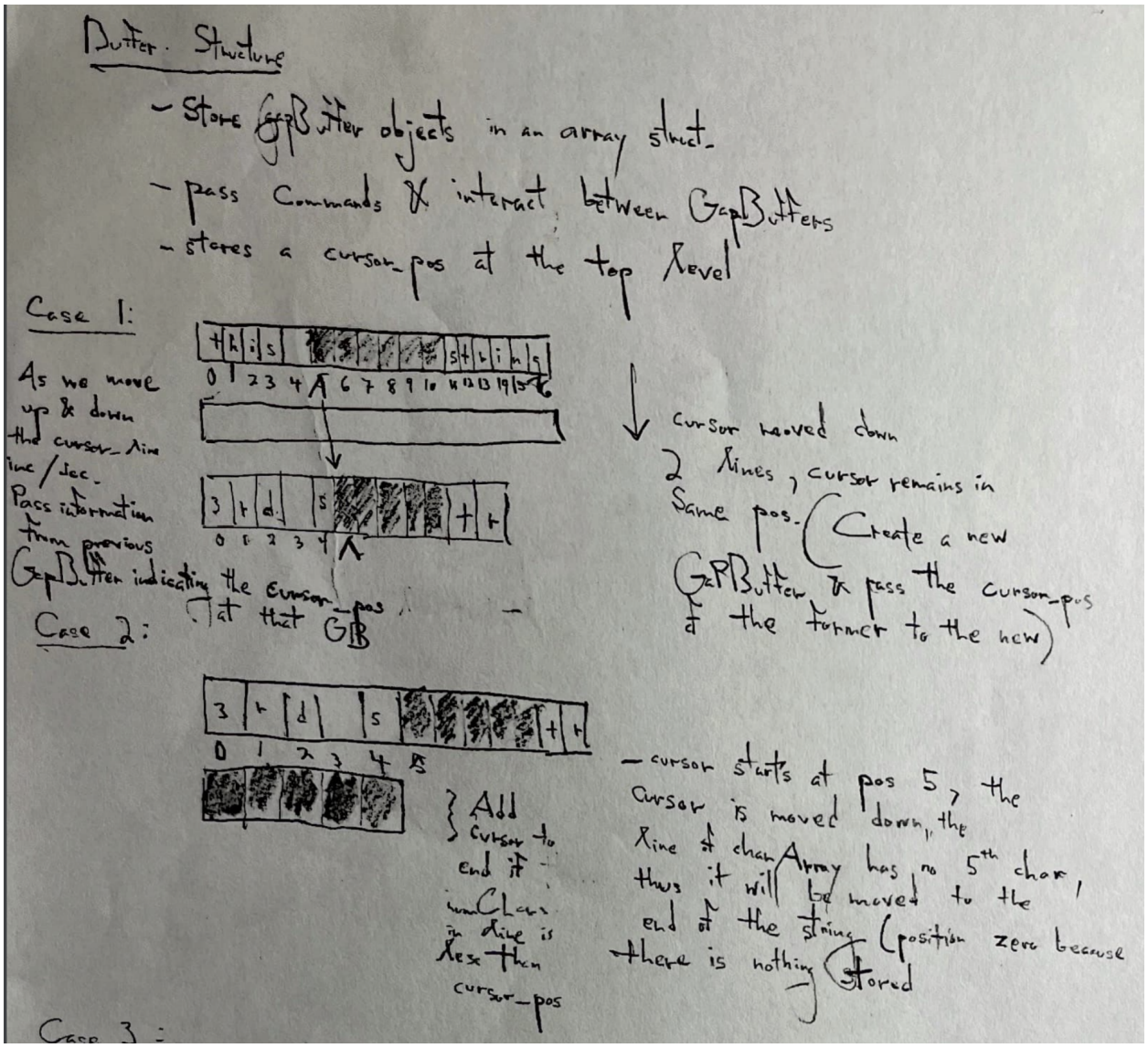


Figure 8 - Buffer structure initial design/planning

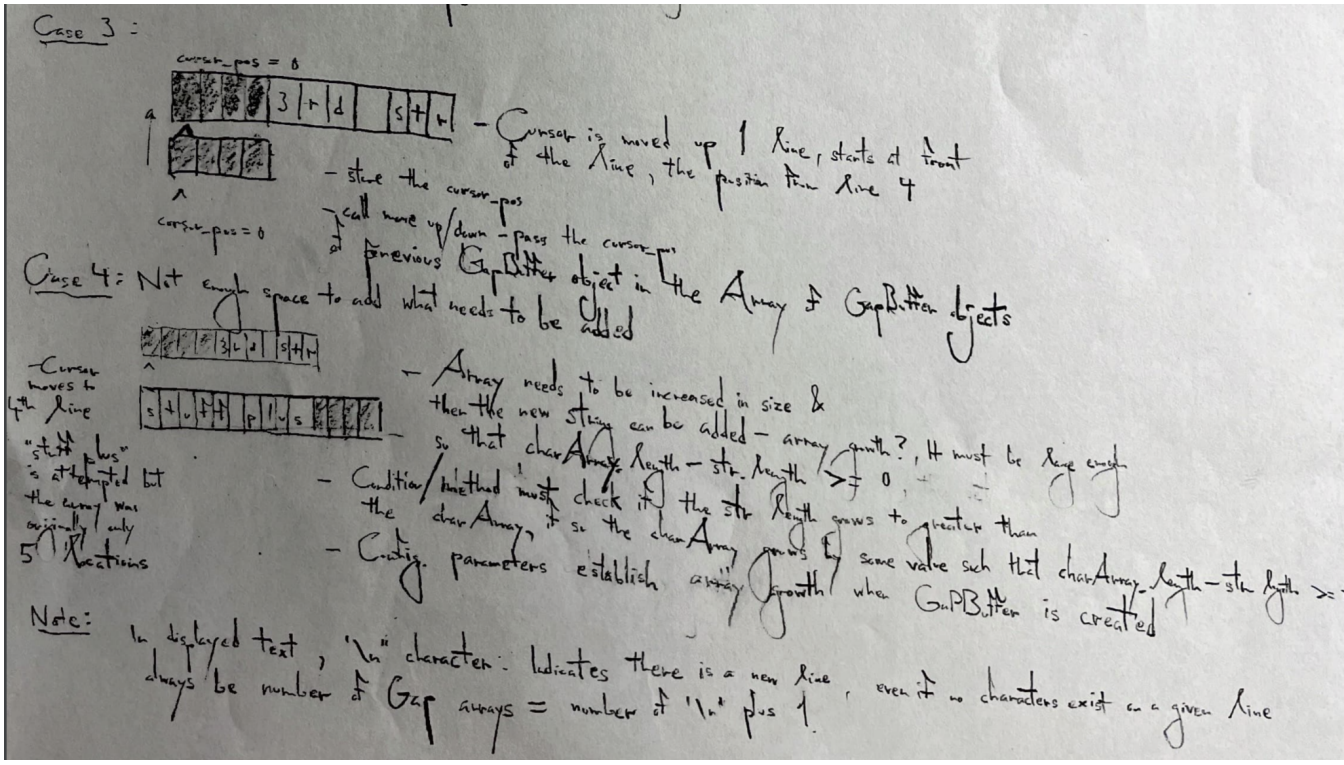


Figure 9 - Buffer structure case analysis. The purpose of case analysis is to think about possible inputs and their outputs before applying physical tests to ensure that the basic logic is relatively bug free.

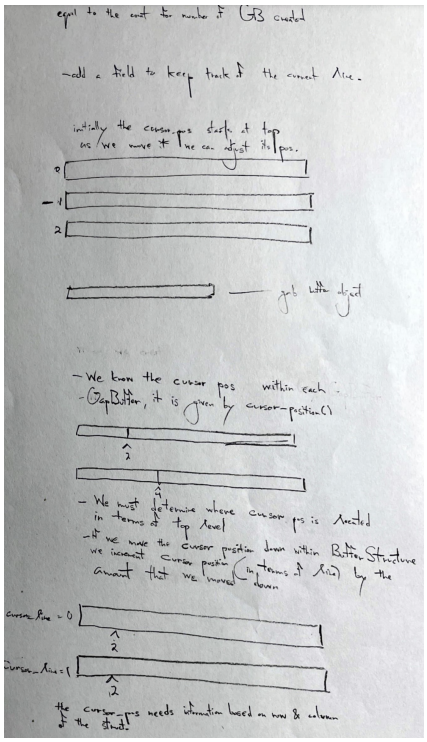


Figure 10 - Buffer structure case analysis cont.

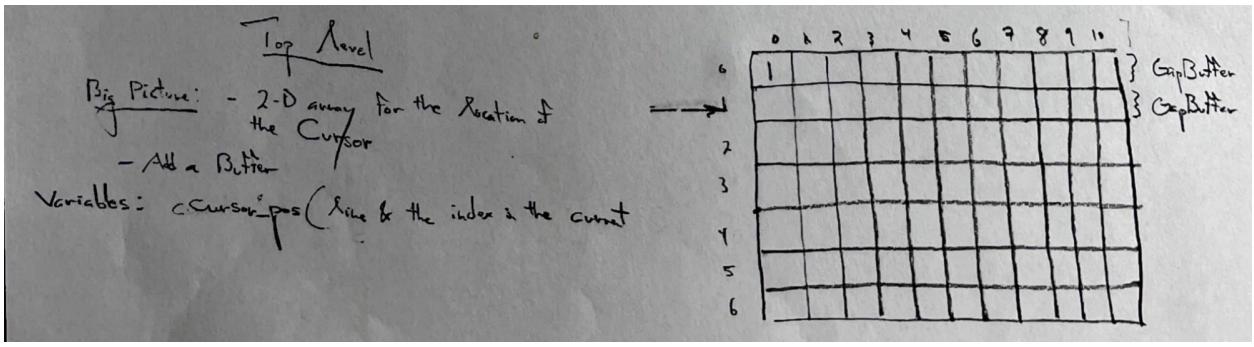


Figure 11 - Buffer structure top level design/planning

inc_gb_count

This method was created because everytime we add GapBuffers to the array we must have a cnt that increments, if the count is greater than the length of the gb_array we must increase the size of the array by creating a new array and setting the length equal to the overflow ie. cnt - gb_array.length. This method is described below.

check_overflow

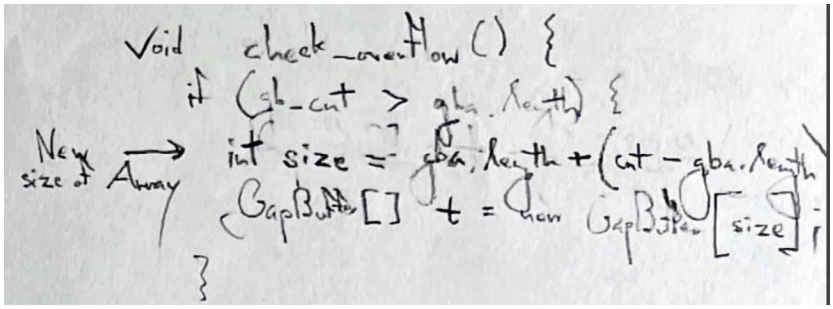


Figure 12 - check_overflow method pseudo code

Line loading methods

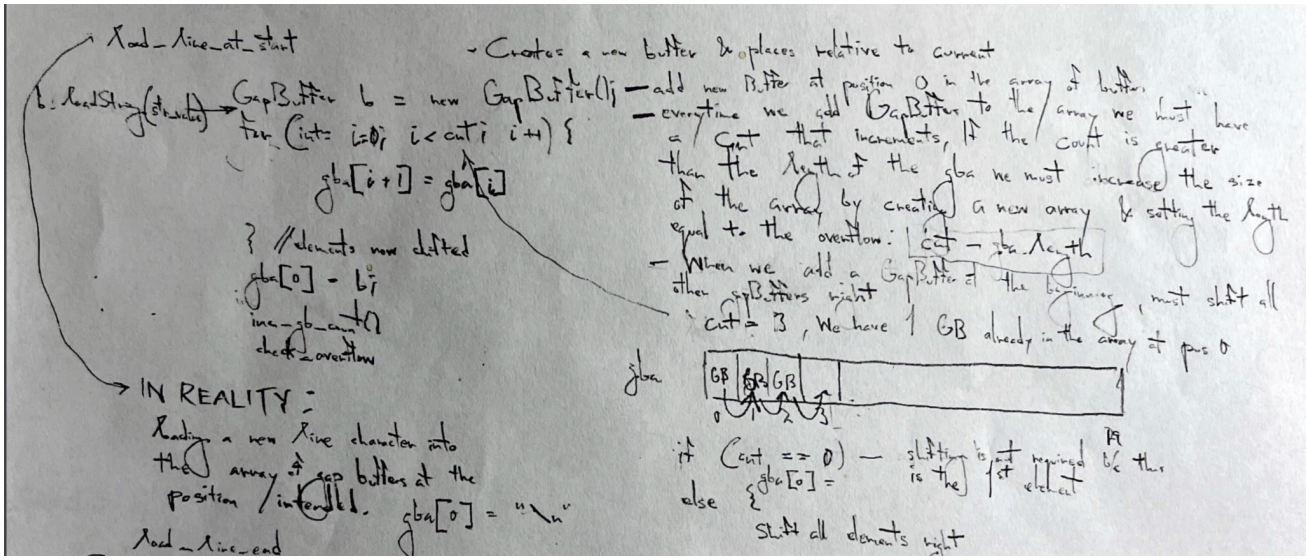


Figure 13 - load_line_at_start method design

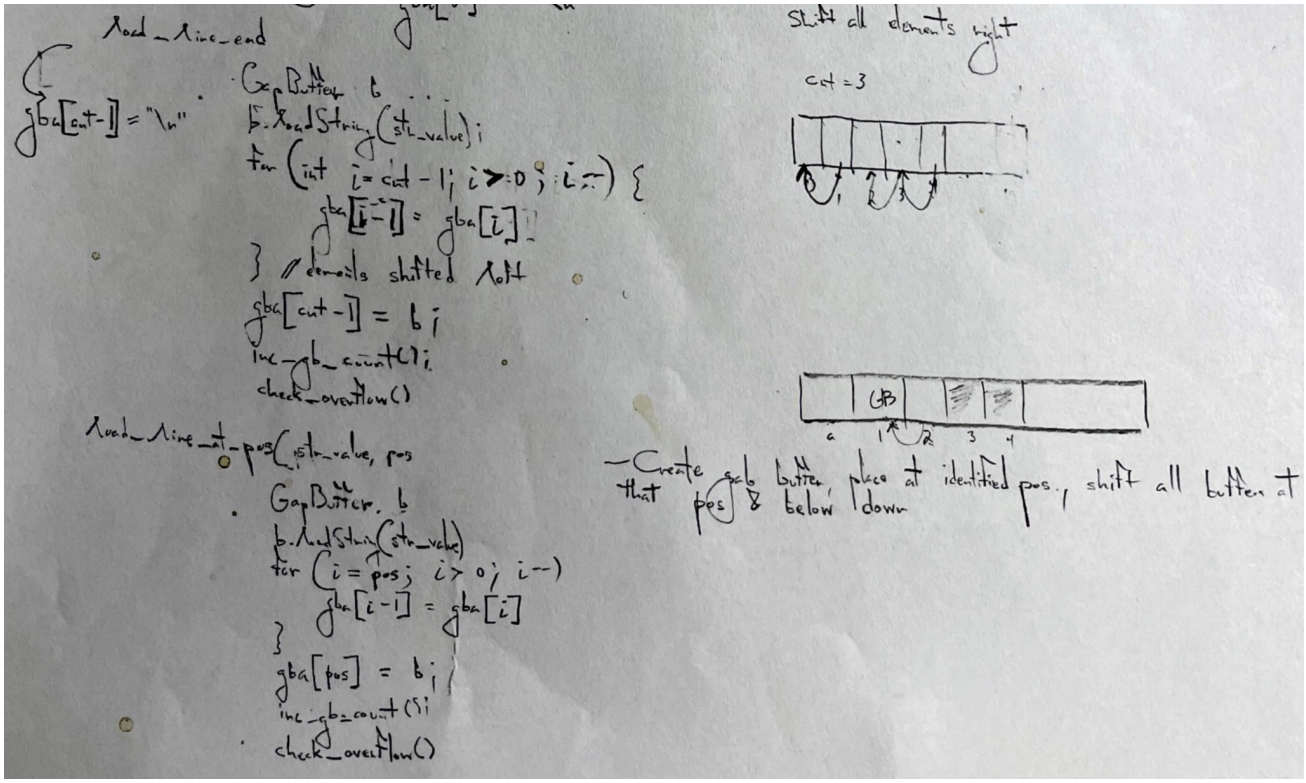


Figure 14 - load_line_end and load line at a position method design

Cursor movement methods

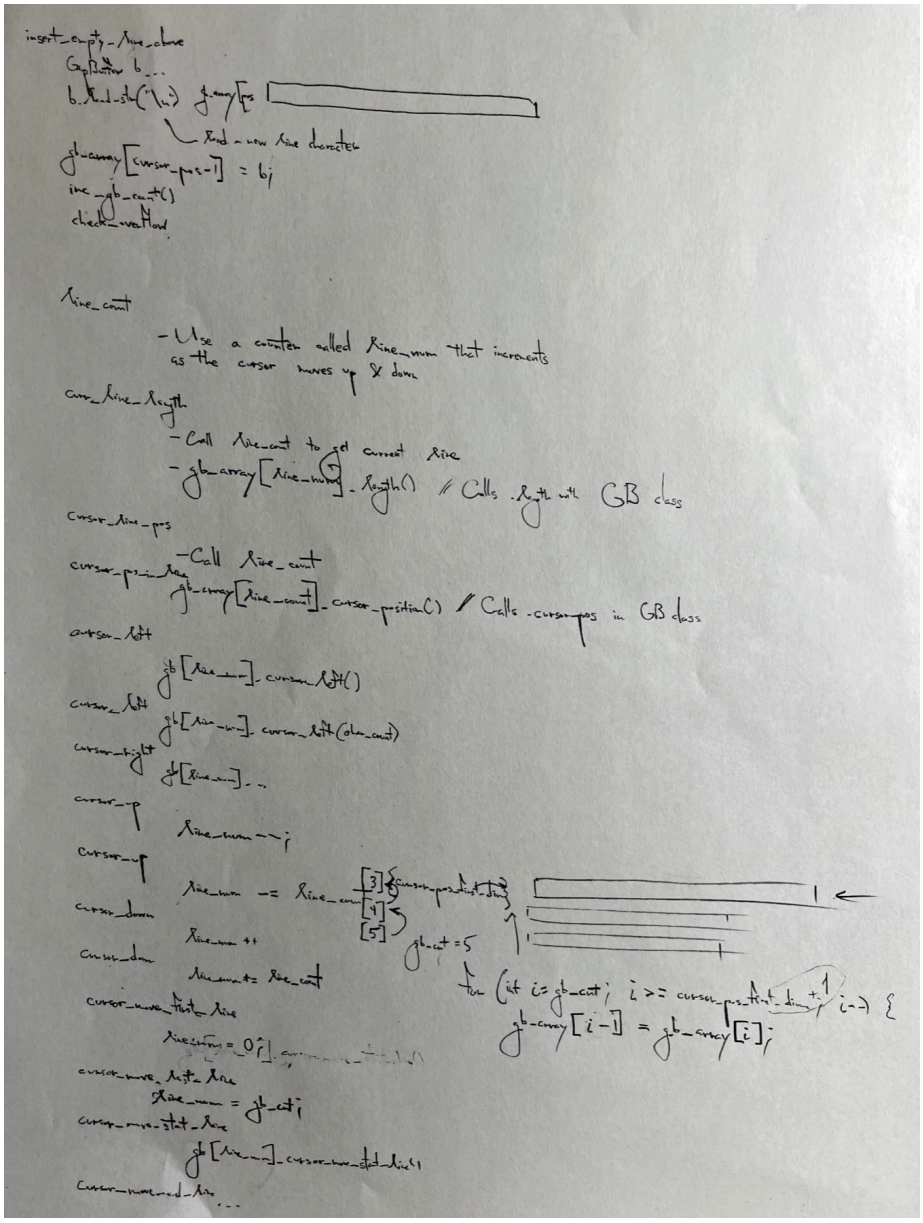


Figure 15 - cursor movement method design

Allows the cursor to be moved up and down in the 2-D array structure.

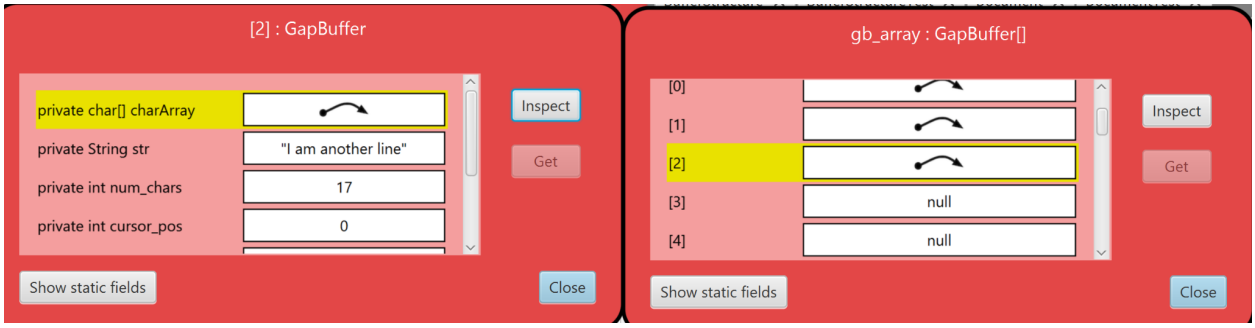
Document

All interactions with data structures go through this class. This acts as the top level class in the design.

load_file

Load file creates a new FileInputStream and primes a scnr to read using the DocumentIO class. A line is read and loaded into the buffer structure. Each line in the file is loaded into BufferStructure starting with index 0 using the load_line_at_start() method.

It proved somewhat difficult to test this using the available methods given. I therefore relied on visual inspection. The following illustrates why I passed the test for load_file




```
@Test
@DisplayName("testing loading file given a filename")
public void load_file_tb() throws IOException {
    //testing strategy for this method has some ambiguity
    //what follows is an attempt to create a file, initially
    //with text, load the contents into the BufferStructure
    //restore the contents that were loaded into the BufferS
    Document doc = new Document(); //create a new Document o
    File file = new File("filename.txt");
    FileOutputStream fos = new FileOutputStream(file); //wri
    PrintStream ps = new PrintStream(fos);
    ps.println("I am a line");
    ps.println("I am another line");
    ps.println("another line of text");
    doc.load_file("filename.txt"); //I visually inspected th
    doc.store_file("filename.txt"); //
}
```

Figure 16 - load_file test image and source code

store_file

There is some ambiguity between the Document and DocumentIO class here, I am not exactly sure what the desired functionality should be. I assume I should be using FileOutputStream and PrintStream objects to write to the file but that was not

implemented in the interface. Regardless, the class will populate the file with the contents of the buffer structure.

The remaining methods within Document are straightforward, they simply call methods within the BufferStructure class using an instance of BufferStructure called b_struct.

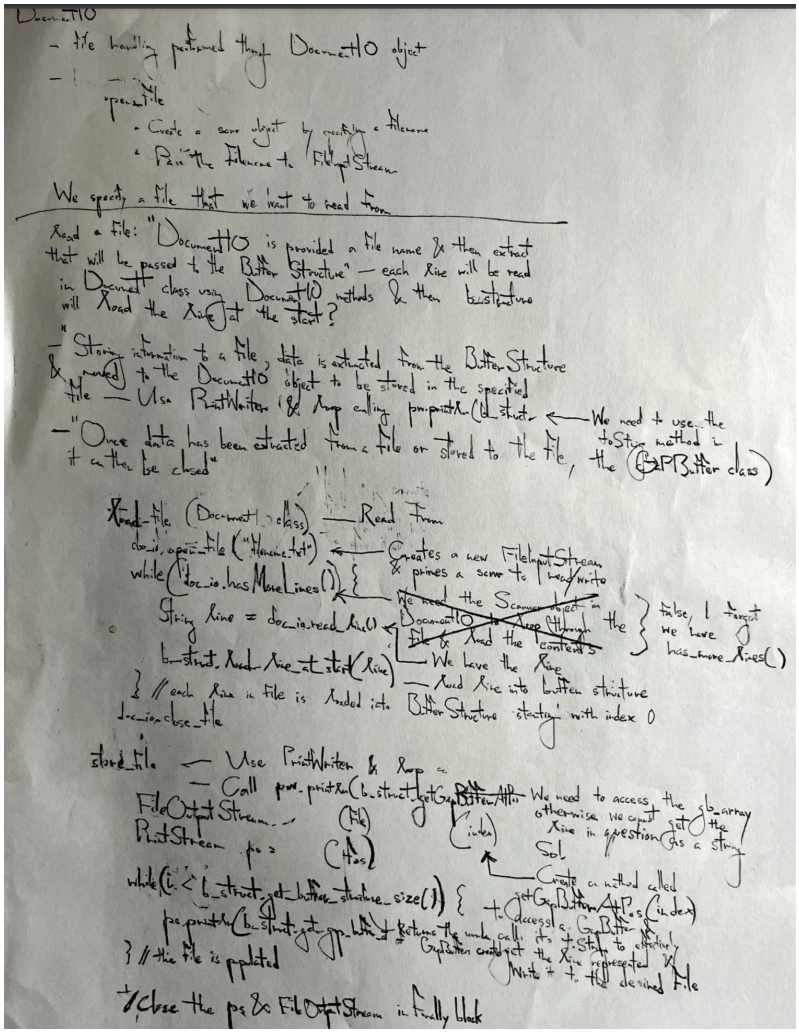


Figure 17 - load_file and store_file method design

Appendix A

Raw design notes for GapBuffer & BufferStructure classes:
https://acrobat.adobe.com/link/track?uri=urn:aaid:scds:US:f370399c-4446-44ae-a37d-47410236202f

Raw design notes link for Document class:
https://acrobat.adobe.com/link/track?uri=urn:aaid:scds:US:8c811541-2a92-4d22-b508-2040d6ef9e32