Alex Kurata
May 8, 2017
Software Development 1

Final Write Up

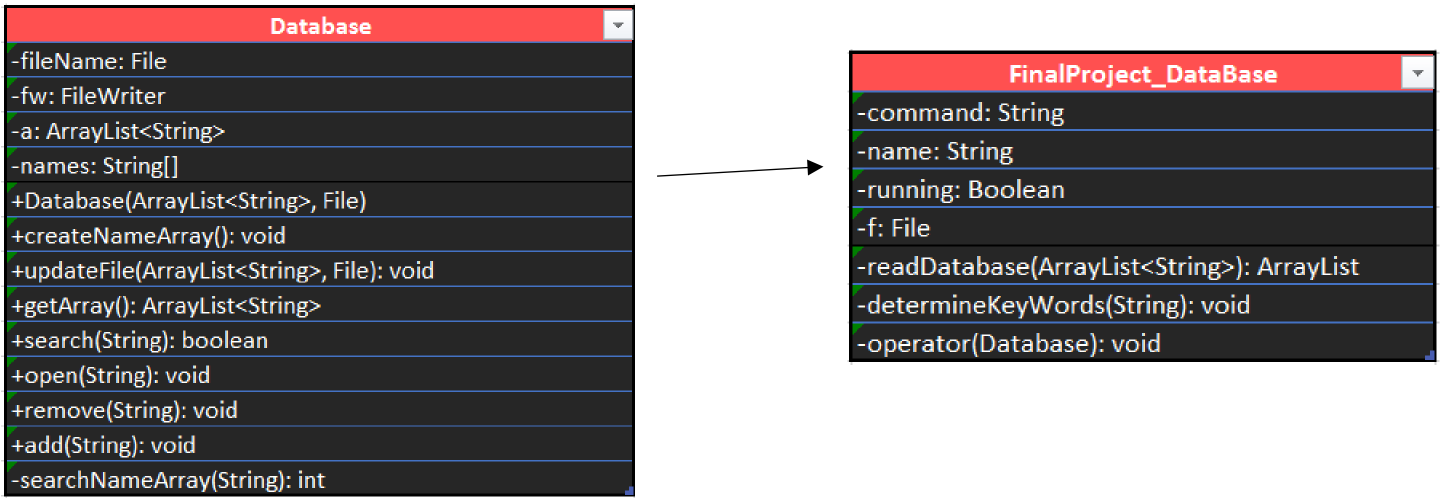
Abstract:

For this project, I decided to write a database management software. The code works in having two separate class files and a text document to hold all the string values of the database. The user is prompted to enter certain commands to either add, remove, search, the database or stop the function completely. The main premise of the program is to save file locations of other assignments from the past work I've completed in a text document.

Introduction:

I thought this project might be interesting since I will be taking classes relevant to database management, so it would be a good idea to try and at least see the complexity to the relative functionality of database management software. I attempted to see what basic commands could be implemented into a database, like adding or removing a file, or searching for the index of a file. Another goal was to try and use efficient algorithms to complete search or writing processes in a quick manner. Since string is a complicated means to compile a search algorithm for, I had difficulty in making an intuitive and effective way to complete each given command.

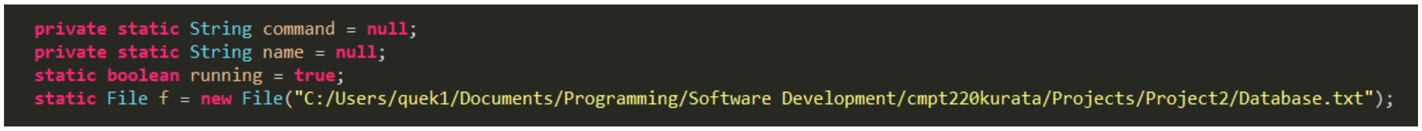
System Description:



Initialization:

The program operates between two class files, with one containing the main method, and another establishing the Database object. To start, FinalProject_Database.java initializes four static variables to be used throughout the program.

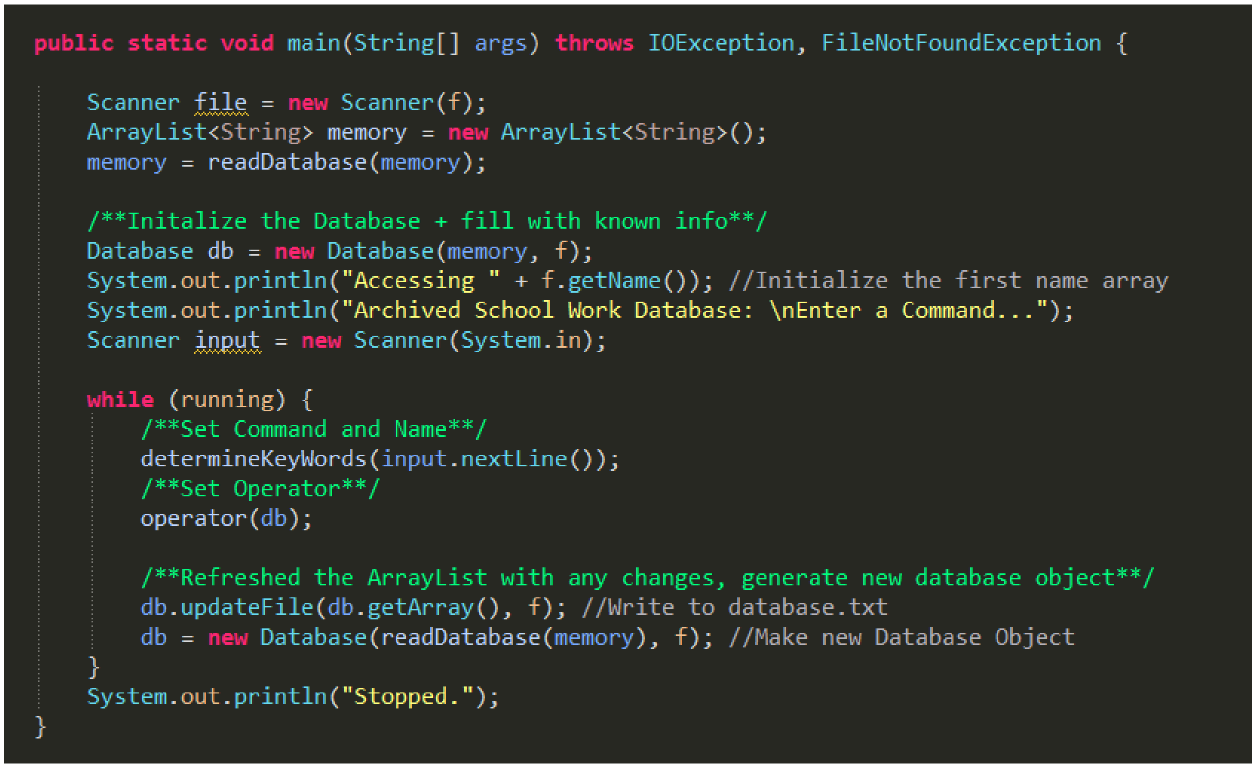
Command: is a string value that holds the command that comes from user input.
Name: is another string that holds the value of the file that is being operated on.
Running: is a Boolean value that is used to control a while loop that will continuously loop unless broken with this variable.
F: Contains the location of the Database.txt file itself. This file is used to save each file location.



```java
private static String command = null;
private static String name = null;
static boolean running = true;
static File f = new File("C:/Users/quek1/Documents/Programming/Software Development/cmpt220kurata/Projects/Project2/Database.txt");
```

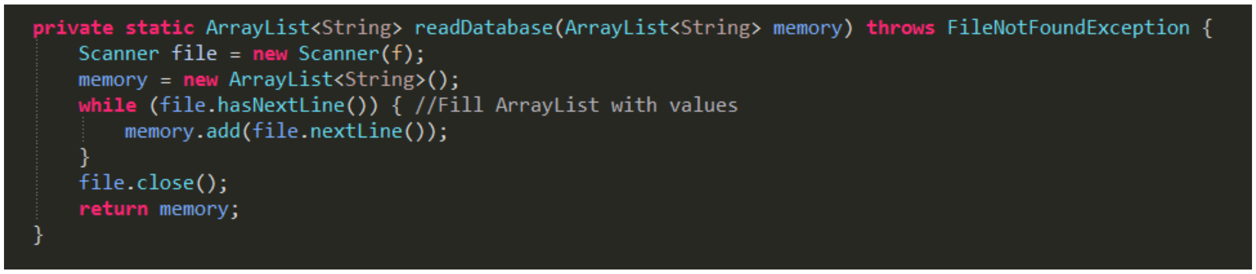
Main method:

Continuing the program goes to the main method. A scanner object is created an ArrayList<String> object is created, the ArrayList's purpose is to hold an indefinite list of the file names and to have others be added or remove at will, without the constraints of a normal array.



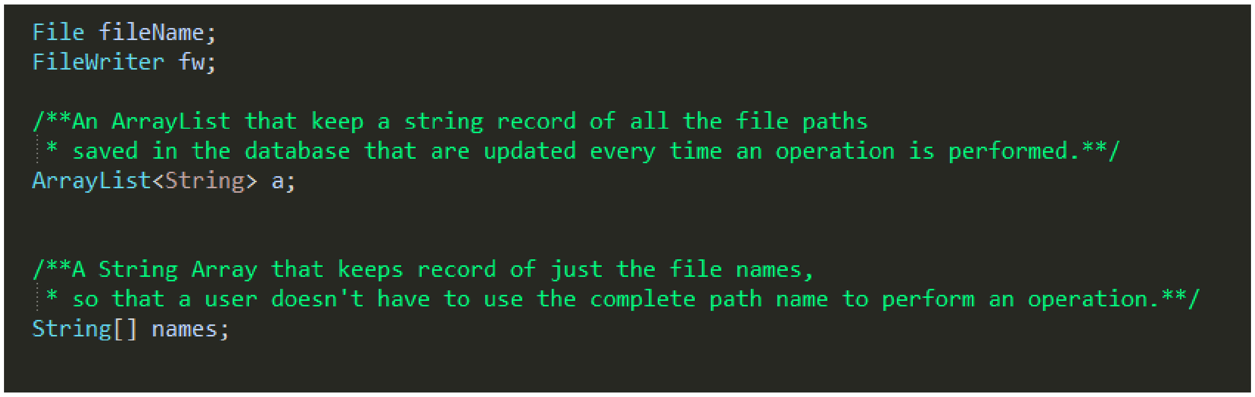
```java
public static void main(String[] args) throws IOException, FileNotFoundException {

    Scanner file = new Scanner(f);
    ArrayList<String> memory = new ArrayList<String>();
    memory = readDatabase(memory);

    /**Initalize the Database + fill with known info**/
    Database db = new Database(memory, f);
    System.out.println("Accessing " + f.getName()); //Initialize the first name array
    System.out.println("Archived School Work Database: \nEnter a Command...");
    Scanner input = new Scanner(System.in);

    while (running) {
        /**Set Command and Name**/
        determineKeyWords(input.nextLine());
        /**Set Operator**/
        operator(db);

        /**Refreshed the ArrayList with any changes, generate new database object**/
        db.updateFile(db.getArray(), f); //Write to database.txt
        db = new Database(readDatabase(memory), f); //Make new Database Object
    }
    System.out.println("Stopped.");
}
```

Following the initialization of the Memory ArrayList<String> object, the readDatabase method is now called. This method uses a Scanner object to read each line of the Database.txt file and fill each line of the array, this is a method because the ArrayList is often filled and remade often.

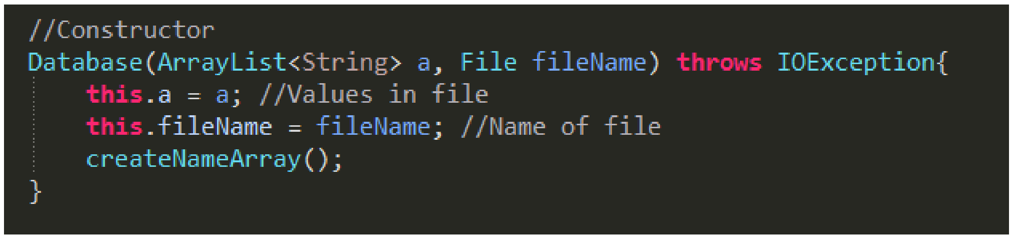


```
private static ArrayList<String> readDatabase(ArrayList<String> memory) throws FileNotFoundException {
    Scanner file = new Scanner(f);
    memory = new ArrayList<String>();
    while (file.hasNextLine()) { //Fill ArrayList with values
        memory.add(file.nextLine());
    }
    file.close();
    return memory;
}
```

Continuing down the main method, a Database object is now created from the other class file, passing along the ArrayList and file location. With the database object now made, the class begins by initializing a File and File Writer, ArrayList<String>, and String Array object. The String array is made to simplify user experience, since typing an entire file path can be tedious, the program also initializes a regular array to hold only the file name and extension.

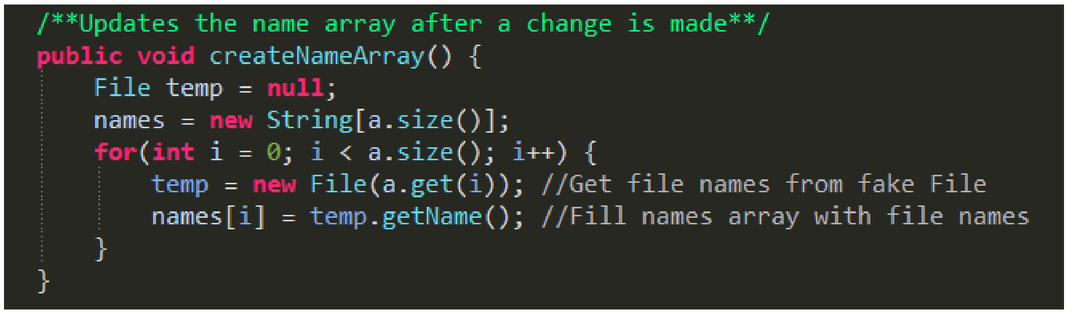


```
File fileName;
FileWriter fw;

/**An ArrayList that keep a string record of all the file paths
 * saved in the database that are updated every time an operation is performed.**/
ArrayList<String> a;


/**A String Array that keeps record of just the file names,
 * so that a user doesn't have to use the complete path name to perform an operation.**/
String[] names;
```



```
//Constructor
Database(ArrayList<String> a, File fileName) throws IOException{
    this.a = a; //Values in file
    this.fileName = fileName; //Name of file
    createNameArray();
}
```

The Database constructor also runs the createNameArray method. This is used to fill the string array with previously mentioned file names without the complete path.

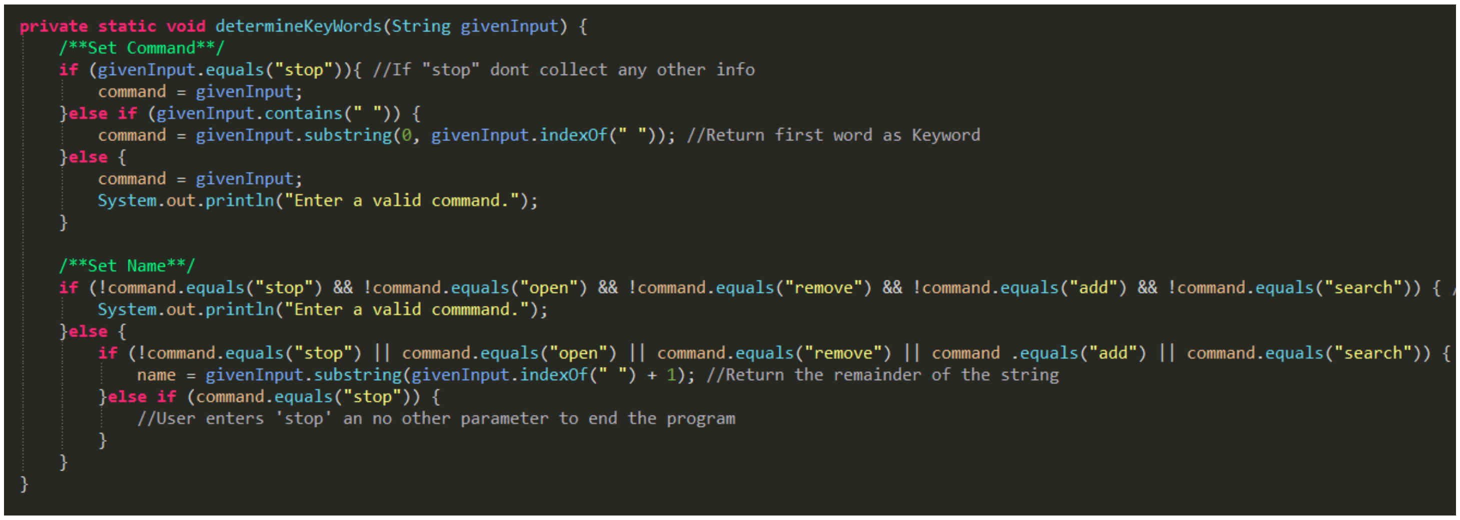


```
/**Updates the name array after a change is made**/
public void createNameArray() {
    File temp = null;
    names = new String[a.size()];
    for(int i = 0; i < a.size(); i++) {
        temp = new File(a.get(i)); //Get file names from fake File
        names[i] = temp.getName(); //Fill names array with file names
    }
}
```

The main method now prints a check statement to see make sure that the file being accessed is the correct one, and another statement to institute the user to enter something. A Scanner object now waits on user input.

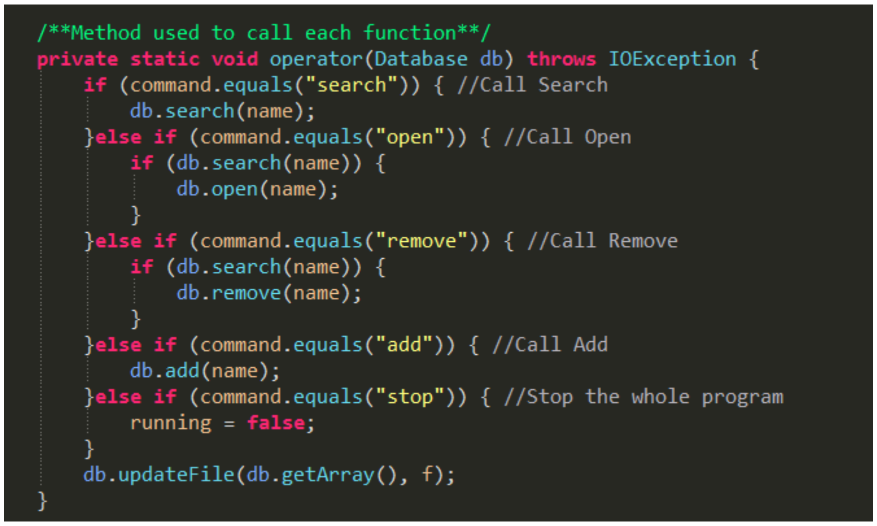
Following any user input, the program now loops through three new methods that consist most of the program: determineKeyWords, operator (add, remove, open, search), and updateFIle.

determineKeyWords: Runs two if statements to separate the string entered and determine what to do next. If the user enters "stop", command gets set to "stop" thusly ending the while loop and ending the program. If the user enters something containing a space, the program knows that it must contain multiple words since the commands are all only one word, and will thusly create a substring with the first word being the command and the second being the location. The second part of this method will now do several checks to see if it recognized the command set, and this will completely establish both the command and name variables.



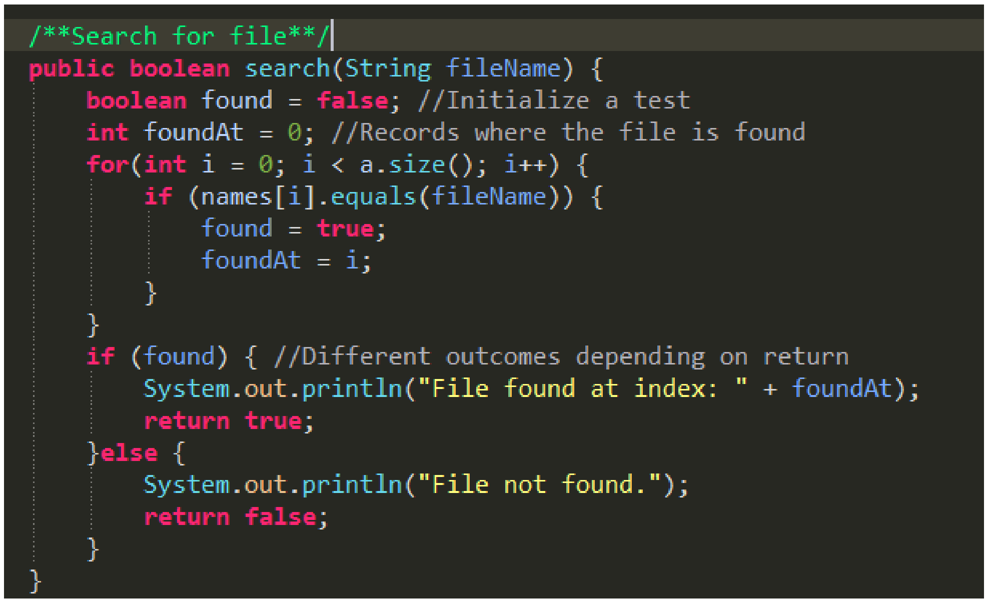
```java
private static void determineKeyWords(String givenInput) {
    /**Set Command**/
    if (givenInput.equals("stop")){ //If "stop" dont collect any other info
        command = givenInput;
    }else if (givenInput.contains(" ")) {
        command = givenInput.substring(0, givenInput.indexOf(" ")); //Return first word as Keyword
    }else {
        command = givenInput;
        System.out.println("Enter a valid command.");
    }

    /**Set Name**/
    if (!command.equals("stop") && !command.equals("open") && !command.equals("remove") && !command.equals("add") && !command.equals("search")) { /
        System.out.println("Enter a valid commmand.");
    }else {
        if (!command.equals("stop") || command.equals("open") || command.equals("remove") || command .equals("add") || command.equals("search")) {
            name = givenInput.substring(givenInput.indexOf(" ") + 1); //Return the remainder of the string
        }else if (command.equals("stop")) {
            //User enters 'stop' an no other parameter to end the program
        }
    }
}
```

Operator: Runs an "if" statement to identify the command given and running the appropriate methods in the Database object. For the commands "open" and "remove" the methods also runs search, before continuing to see if it can even find the file beforehand.



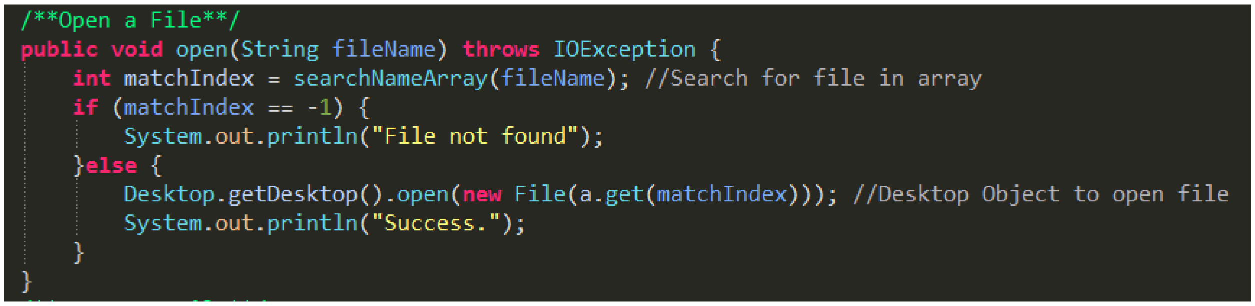
```java
/**Method used to call each function**/
private static void operator(Database db) throws IOException {
    if (command.equals("search")) { //Call Search
        db.search(name);
    }else if (command.equals("open")) { //Call Open
        if (db.search(name)) {
            db.open(name);
        }
    }else if (command.equals("remove")) { //Call Remove
        if (db.search(name)) {
            db.remove(name);
        }
    }else if (command.equals("add")) { //Call Add
        db.add(name);
    }else if (command.equals("stop")) { //Stop the whole program
        running = false;
    }
    db.updateFile(db.getArray(), f);
}
```

Search: This method will run a binary search algorithm to compare the passed string to find one in the array that resides in the database object. The method will print the index in which the string was found and return true, or it will just return false.

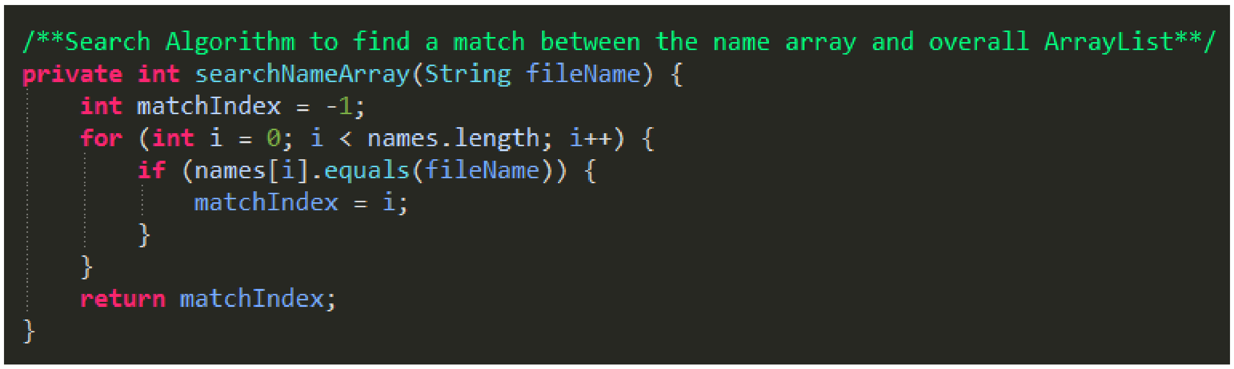


```java
/**Search for file**/
public boolean search(String fileName) {
    boolean found = false; //Initialize a test
    int foundAt = 0; //Records where the file is found
    for(int i = 0; i < a.size(); i++) {
        if (names[i].equals(fileName)) {
            found = true;
            foundAt = i;
        }
    }
    if (found) { //Different outcomes depending on return
        System.out.println("File found at index: " + foundAt);
        return true;
    }else {
        System.out.println("File not found.");
        return false;
    }
}
```

Open: This method will attempt to open a file passed in the default program. To begin the method runs another search algorithm, searchNameArray. This method simply return the index of a file, or an index that is out of bounds, -1. If the open method, is given the -1 index it will then print “File not found” and the loop outside will continue. If the file is found, the Desktop object is called to open the file at the matching index.

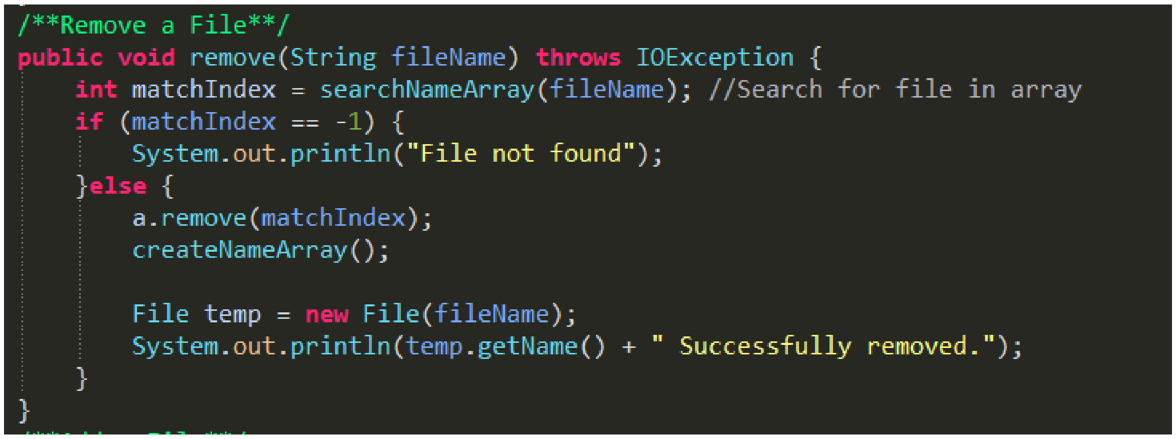


```java
/**Open a File**/
public void open(String fileName) throws IOException {
    int matchIndex = searchNameArray(fileName); //Search for file in array
    if (matchIndex == -1) {
        System.out.println("File not found");
    }else {
        Desktop.getDesktop().open(new File(a.get(matchIndex))); //Desktop Object to open file
        System.out.println("Success.");
    }
}
```



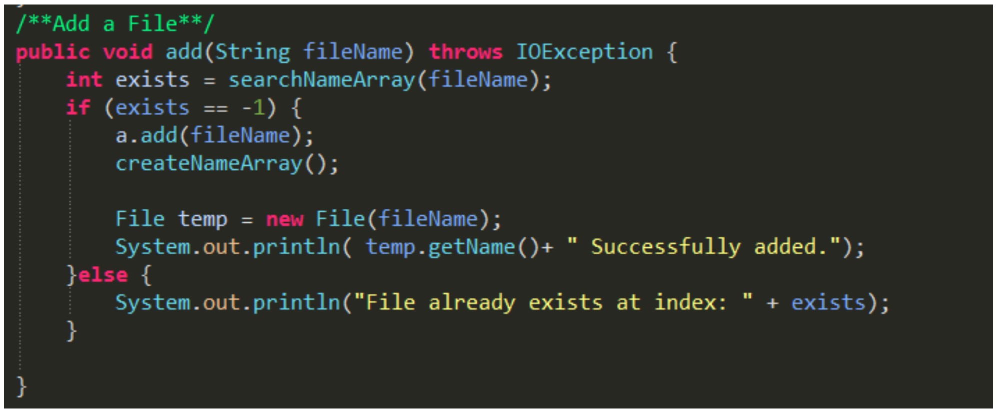
```java
/**Search Algorithm to find a match between the name array and overall ArrayList**/
private int searchNameArray(String fileName) {
    int matchIndex = -1;
    for (int i = 0; i < names.length; i++) {
        if (names[i].equals(fileName)) {
            matchIndex = i;
        }
    }
    return matchIndex;
}
```

Remove: This method's job is to simply find the index of the desired element, and remove it. Following the command being run, a new name array is formed and refilled with the new set of names.



```
/**Remove a File**/
public void remove(String fileName) throws IOException {
    int matchIndex = searchNameArray(fileName); //Search for file in array
    if (matchIndex == -1) {
        System.out.println("File not found");
    }else {
        a.remove(matchIndex);
        createNameArray();

        File temp = new File(fileName);
        System.out.println(temp.getName() + " Successfully removed.");
    }
}
```

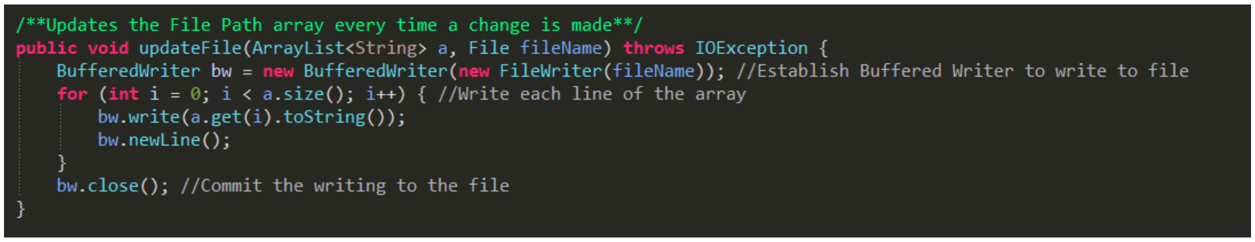
Add: This method, similar to remove, will search for the element and return "File already exists" if the file doesn't, the method will tack it on to the end of the ArrayList and create a new name array.



```
/**Add a File**/
public void add(String fileName) throws IOException {
    int exists = searchNameArray(fileName);
    if (exists == -1) {
        a.add(fileName);
        createNameArray();

        File temp = new File(fileName);
        System.out.println( temp.getName()+ " Successfully added.");
    }else {
        System.out.println("File already exists at index: " + exists);
    }

}
```

Stop: This is still a part of the same "if" statement, it simply sets the running variable to false and breaks the entire overarching while loop.

Back in the main method now, the updateFile method is called, and a new Database Object is created. updateFile uses the buffered writer object in the Database class to write over the file, it does this by printing each line of the ArrayList resident to the Database Object following any changes. A new version of the Object is created to read the entire database file again and refresh any changes made, and the while loop continues.



```
/**Updates the File Path array every time a change is made**/
public void updateFile(ArrayList<String> a, File fileName) throws IOException {
    BufferedWriter bw = new BufferedWriter(new FileWriter(fileName)); //Establish Buffered Writer to write to file
    for (int i = 0; i < a.size(); i++) { //Write each line of the array
        bw.write(a.get(i).toString());
        bw.newLine();
    }
    bw.close(); //Commit the writing to the file
}
```

User Manual

The program should be used by launching the FinalProject_Database.java as the main class file. Then enter one of the following commands to run their relevant functionality: “search”, “add”, “remove”, “open”, “stop” (lower case lettering only). Following the command without hitting space enter a file name for all commands except add and stop. Examples of this format is as follows:

- search FYS-Essay1.docx
- add C:\Users\cmpt220kurata\Projects\Project2\data\FYS-Essay1.docx
- open FYS-Essay1.docx
- stop

Conclusion

From the entirety of this project, I would say the system is not perfect by any means but I do feel like I accomplished a lot. Although this idea is not original or creative, I felt as though it was a more practical use of time, in that I don’t feel as though projects in the future will be too far off. There are some redundant forms of code used, and there are more practical ways to do them, but for a vanilla database management software I would say that it’s not too bad.

Overall my goals were to be able to create a working database management software and this project does fulfill that. It has some practical use, and mostly relies on other functions of a system to operate as well as more specific details, but the entirety of the Database.txt file is viewable in a succinct manner.