EECS 1510: Object Oriented Programming
Project 8 – Final Project
220 Points Due by Monday April 30, 2018

Delivery Instructions: Delivered a printed submission to our TA Makan (Michael) Arastuie in North Engineering 1039, our normal lab, between 10:00 a.m. and 12:00 p.m. on Monday April 30. In case of difficulties, Makan's email address is

Makan.Arastuie@rockets.utoledo.edu

Note 1: This project may be done in pairs, but the pairs **cannot** be the same as on project.

Note 2: This final project has both a fixed part and an optional part. In both cases, the program must follow the 8 Program Standards given with Project 5. All previous projects are posted in the folder "Previous Projects".

Note 3: The optional part is for extra credit, and is somewhat open ended. Up to 150 extra points may be earned through extra credit.

Note 4: Grading and extra credit will be based on Human Engineering, Aesthetics, and Code Quality as well as functionality.

Fixed Part: Consider the phonebook application of Project 7. A solution to this project is given at the end of this document. You can use this code or your own code in Project 8 as a basis for this assignment. Either way is fine.

For the fixed part of this project (the 220 points) you must extend the program as follows:

- Extend the record format to allow both a first name and a last name. Keep the limit of 8 characters for the fist name and 8 characrers for the last name.
- Include a command "a" (for alphabetize) to put the phonebook in alphabetical order.
- Include a command "m" (for merge) to combine entries that may be duplicates or variants for a given person. For example, if John Smith has two entries, combine the notes into one entry, and if the phone numbers are different, put the second number in the notes.
- Construct a Graphical User Interface (GUI) for the user using JavaFX.

You must ake the program object oriented. This means having a separate class for the internal representation of a phonebook, with methods like add, list, or delete. It also means having a separate class for the GUI aspects of the program.

You must also follow the programming standards given in the file

Program Standards for Java Course.doc

posted in the folder "Administration".

Optional Part: For the optional part of this project you may extend the program in various ways. Some areas for extension are as follows:

- **Usability and Aesthetics:** Make the GUI especially pleasing to see and use. On example would be to make the list command give a nice listing.
- **Human Engineering:** Provide good human engineering for the user. This means being very tolerant of user errors and making it easy to use. For example, you might give the user an option to name the phonebook file, or you might check if the user tries to add another entry with the same name. Also, consider a simple Help command.
- **Reliability:** Make the program especially reliable. Try to make it so that the program will not crash even under incorrect inputs. For example, handle a missing file or prevent an array out of bounds error.
- **Maintainability**: Make the program especially well-structured and readable.
- **Functionality:** Enhance the functionality of the program in various ways, even small ways.

For functionality, one option is to add a command 'd' for delete to delete an entry corresponding to a name typed by the user. Another enhancement would be to check that a find or enter command actually has a non-null string for the name. A little more work would be to check the format of the phone number. For instance, check that a phone number has one of the following valid forms

```
419-460-1212      (419)460-1212      460-1212
```

(Note that one can use regular expressions for this.) Yet another option is to use a binary search to retrieve the phone number of a person.

Yet another option, likely to be even more work, would be to allow a partial match to find or delete an entry. For example, "F mi" would match any entry with "mi" in the name, for example "Smith" or "Hermit" or Mitchell. You may use the substring function in Java for this feature.

Final Submission: You must follow the posted file

Printed Submission of Projects 2018.doc

or points will be deducted. This means submitting a portfolio or dossier of sorts (a stapled or bound set of pages containing

- A separate FIRST page (or pages) itemizing the grounds for extra credit. *It is your responsibility to itemize all grounds for extra credit.*
- A printout of your code,
- A printout from several consecutive runs, illustrating the various features of your program. For example, you must show that the file I/O works.
- A printout from the list command

In the sample runs, each of the commands "e", "f", "l", "a", and "m" should be illustrated.

A solution similar to Project 7.

```
import java.io.*;
import java.util.*;
```

```
class Entry {
   public String name, number, note;
}

public class PhonebookFor1510 {
public static Entry[] contactList;
public static int num_entries;
public static Scanner stdin = new Scanner(System.in);

public static void main(String args[]) throws Exception{
   int i; char C;
   String code, Command;
   contactList = new Entry[200];
   num_entries = 0;

   readPhoneBook("phonebook.txt");
   System.out.println("Codes are entered as 1 to 8 characters.\nUse" +
         " \"e\" for enter," +
         " \"f\" for find," +
         " \"l\" for listing all the entries," +
         " \"q\" to quit.");

   Command = null;
   C = ' ';
   while(C != 'q'){
      System.out.print("Command: ");
      Command = stdin.next();
      C = Command.charAt(0);
      switch (C) {
      case 'e':  addContact();  break;
      case 'f':
         code = stdin.next();
         stdin.nextLine();
         i = index(code);
         if (i >= 0) displayContact(contactList[i]);
         else        System.out.println("**No entry with code " + code); break;
      case 'l':
         listAllContacts(); break;
      case 'q':
         CopyPhoneBookToFile("PhoneBook1.txt");
         System.out.println("Quitting the application. All the entries are "
               + "stored in the file PhoneBook1.txt"); break;
      default:
         System.out.println("Invalid command Please enter the command again!!!");
      }
   }
}

public static void readPhoneBook(String FileName) throws Exception {
   File F;
   F  = new File(FileName);
   Scanner S = new Scanner(F);

   while (S.hasNextLine()) {
      contactList[num_entries]= new Entry();
      contactList[num_entries].name   = S.next();
      contactList[num_entries].number = S.next();
```

```
      contactList[num_entries].note   = S.nextLine();
      num_entries++;
   }
   S.close();
}

public static void addContact() {
   String name = stdin.next();
   String number;
   stdin.nextLine();
   contactList[num_entries]     = new Entry();
   contactList[num_entries].name = name;

   System.out.print("Enter Number: ");
   number = stdin.nextLine();
   contactList[num_entries].number = number;

   System.out.print("Enter Notes: ");
   contactList[num_entries].note = stdin.nextLine();
   num_entries++;
}

public static int index(String Key) {
// Function to get the index of a key from an array
// if not found, returns -1
   for (int i=0; i < num_entries; i++) {
      if (contactList[i].name.equalsIgnoreCase(Key))
           return i; // Found the Key, return index.
   }
   return -1;
}

public static void displayContact(Entry contact) {
   System.out.println("--"+ contact.name+"\t"+
              contact.number+"\t"+
              contact.note);
}

public static void listAllContacts() {
   int i = 0;
   while (i < num_entries) {
      displayContact(contactList[i]);
      i++;
   }
}

public static void CopyPhoneBookToFile(String FileName) throws Exception{
   FileOutputStream out = new FileOutputStream(FileName);
   PrintStream P        = new PrintStream( out );

   for (int i=0; i < num_entries; i++) {
      P.println(contactList[i].name + "\t" + contactList[i].number +
              "\t" + contactList[i].note);
   }
}}
```