**EECS 1510: Object Oriented Programming**

**Project 20 – Final Project**

**200 Points** Due by 1:00 p.m., Monday April 28, 2014

**Delivery Instructions:** Must be delivered as a printed submission to the TA Kris

Armstrong in PL 2600, the Engineering College Library, between 10:00 a.m. and

12:00 p.m. on Monday April 28.

Kris’s email address is Kris.Armstrong@rockets.utoledo.edu

**Note 1:** This project may be done in pairs, but the pairs **cannot** be the same as on the previous project. Pair projects will be evaluated at a slightly higher standard, at about a 20 points level.

For example, a submission worth 240 points submitted alone would be counted as 220 points if submitted by a pair.

**Note 2:** This final project has both a fixed part and an optional part. In both cases, the program must follow the 6 program Standards given with Project 4 (all previous projects are still posted in the folder “Previous Projects”).

**Note 3:** The optional part is for extra credit, and is somewhat open ended. Up to 100 extra points may be earned through extra credit.

Consider the phonebook application of Project 6. A solution to this project is given at the end of this document. You can use this code or your own code in Project 6 as a basis for this assignment. Either way is fine.

For the fixed part of this project (the 200 points) you must extend the program as follows:

The phonebook is to be kept in alphabetical order. When a new entry is to be added, put the entry into the appropriate location in the array.

Construct a Graphical User Interface (GUI) for the user. Do not use JOptionPane.

You must use Java Swing.

Make 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.

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.

**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 well or prevent an array out of bounds error.

**Maintainability**: Make the program especially well-structured and readable.

**Functionality:** Enhance to 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. (Note that one can use regular expressions for this.). Yet another option is to use a binary search to retrieve a phone number.

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 “Submission of Projects.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 initial 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", “d”, and "q" should be illustrated.

**My Project 6.**

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 's':

sortList(); 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 sortList() {

int i;

Entry temp;

temp = new Entry();

for(int j=0; j < num\_entries; j++) {

for(i = j+1; i < num\_entries; i++) {

if(contactList[j].name.compareToIgnoreCase(contactList[i].name)>0) {

//Swap the entries if their names are not in ascending order

temp = contactList[j];

contactList[j] = contactList[i];

contactList[i] = temp;

}

}

}

System.out.println("Sorted the list. Press l to list all the contacts");

}

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);

}

}

}