Due date: April 6, 2018 in the submit folder

This assignment will provide you with the opportunity to develop a more comprehensive program than what you have been doing. This assignment requires the development of one integrated program consisting of a number of classes.

**Code Development and Design**

As in assignment 1, you will be provided with a start BlueJ folder. Please refer to the similar section in the Assignment 1 write-up for instructions.

**Problem Scenario: Online Phone Book**

Fly By Night Consulting is a small company that is computerizing their company telephone directory. Each employee has a four-digit telephone number (i.e. a local). You are to write a menu driven program that will retrieve information, and allow the user to update the directory.

**General Requirements:**

The current directory is stored alphabetically by last name in a file named **phone.txt**. Each line in the file contains the data for each employee. The data for each employee is last name, first name, initial, department name and telephone local in that order. For simplicity, assume that no two people can have the same last name, and that everyone has a middle initial.

The client interface presents a menu to the user. This menu has the following options:

P – Print a list of all persons in the directory

L – look up a person’s phone number

R – create a report with a department phone list

C - change a listing

A – add a new listing

D - delete a listing

E - exit

P – This prints the contents of the entire directory. It will use the same format as Option L.

L – if this option is chosen, the user should be asked for the person’s last name. The directory will be searched, and if the name is there (this should be case insensitive) the person’s info is displayed. An appropriate message will be printed if the employee is not found in the list.

R – if this option is chosen, the user should be asked for the name of the department, whether the report should be written to a file or to the screen. If to a file, ask for the name of the file to which the report should be written. Each entry in the directory will be examined, and if the employee is in the requested department (ignore case), the person’s info should be written in a report format. The report should have appropriate heading information.

C - this option is for updating the info for an employee. The user should be asked for the person’s last name, and the directory will be searched using this. If the name is there (this should be case insensitive), the info will be displayed. In this case, the department, or local phone number, or both may be changed. The program must ask which fields need to be changed, then do so.

A - this option is for adding a new employee into the directory. For simplicity, assume that the last name is unique, that is, no two persons will have the same last name. If the employee is already in the directory, the program should print an appropriate message and not add the employee. Otherwise, add the record into the directory in the correct place (i.e. in order).

D - this option is for deleting an employee from the directory. If s/he is not found in the list, the program should print an appropriate message.

E - this option is for terminating the program. When this is chosen, the contents of the directory written back into a file (different from the input file) in the same format as the input file. The next time the program is run, it will read the latest version of the phone directory data.

**Program Requirements**

You will need to create and complete the following classes. The start folder will contain the skeleton for these classes.

1. The **Person** class

The **Person** class contains the data for each employee that is contained in the directory. It has the following as instance variables:

1. Last name
2. First name
3. Initial
4. Department
5. Telephone number

It should also have the usual accessors and mutators. In addition, it should have a methods to print the information required in Options L and R. The format of this information for Option L should be similar to:

Ruben B. Yumol Marketing 9876

For Option R, the format of the data (regardless of whether on screen or to a file) should be similar to:

Ruben B. Yumol 9876

This class should also be documented to meet the JavaDoc requirements as shown in class.

1. The **Directory** class

The **Directory** class is the main processing class for this application. It will contain the actual directory as well as all the processing methods required in the menu. Because the number of employees is relatively small, the data, which is located in a file is loaded into an ArrayList of **Persons.**

All of the above processing use this ArrayList, both for searches and also for processing the requirements. When Option ‘E’ is chosen, before ending the program, the data in the ArrayList should be saved in another file in the same format as the original data file. Thus, there should be a method to write the contents of the ArrayList into a file. Thus, the company will always have the most current information for its data which can be loaded the next time the application is used.

Develop the necessary methods in order to fulfill the requirements of each of the menu items. Most of these requirements will involve performing some process on the ArrayList. For most of the options, you will need to search the ArrayList to determine if the person exists in the directory. Assume that you will use the last name as the search criteria.

For Option P, the report format should look like:

Fly By Night Consulting

Name Department Telephone Number

Ruben B. Yumol Marketing 9876

Clark S. Kent Accounting 5689

For Option R, the report format should look like:

Marketing Department

Name Telephone Number

Ruben B. Yumol 9876

Clark S. Kent 5689

There could be other helper methods that may be required in order to assist in the processing. Searching if the person is in the directory could one such method.

Note: This class should not receive any input from the user as the user is not allowed to access this class directly.

1. The **UserInterface** class

This is the class the user interacts with. The user should not be able to directly interact with the **Directory** class or the **Person** class. In addition, this class interacts only with the **Directory** class but not the **Person** class.

You will be given the skeleton for this class which contains a menu as a separate method. This method will perform the following functions:

1. Before any processing can be done, the data must be loaded into the ArrayList. Hence, you will need to create a **Directory** object and then use the appropriate method of the **Directory** object to load the data. The required input is the filename of the data which is then passed as a parameter to the **Directory** object’s method that loads the data.
2. After the data is loaded, the menu is presented and the user is asked for his choice. It is important that if the user’s choice is not one of the valid choice, the user is informed of such.
3. For a valid user choice, the choice is processed according to the requirements. This often involves invoking the **Directory** object’s method appropriate to fulfill the requirement.
4. For Option A, since String is not a primitive type, you cannot compare two string variable using == or <=. You will need to use one of the methods provided with the string class. The best one for this assignment is compareToIgnoreCase. This works as follows. If str1 and str2 are String variables, the expression str1.compareToIgnoreCase(str2) returns a 0 if the two strings are equal (ignoring case), a negative number if str1 comes before str2 (e.g. if you were looking them up in a dictionary), and a positive number if str1 would come after str2 in the dictionary. You can check the Java 8 API for more information.
5. For Option R, the program should be able to deal with printing the report on the screen or to a file. If the file is required, the filename is to be inputted.
6. When the exit option is chosen, the data in the ArrayList should be written to a file. This should receive the filename as input and the appropriate **Directory** object method is called to finish the processing. The file name is passed as a parameter.

Aside from the menu choice, assume that the user is a smart user and the he will enter the proper data for all the other required input. No error checking will be required. Ensure that your code follows the Coding Conventions and Documentation standard for this class.

**Development Hints**

This is a rather large and complex program than what you are used to. Hence, it is imperative that you develop and test your program incrementally. Here are some hints to assist you in developing the program.

1. Develop one class at a time. Since the **Person** class is the core of the application, it might be a good idea to develop this class first. Identify the required instance variables

and the required methods perhaps by drawing the class diagram.

Develop the class one method at a time. Once a method is developed, test the method. You can write a test class that can do this which you can later remove.

1. Next, develop the **Directory** class since this uses the **Person** class. As before, determine the required variable. Since this is a processing class, there probably is no need for instance variables except perhaps for the ArrayList. If this will need a Scanner, declare it as an instance variable. Determine the required methods. Ensure that this will have methods to perform the processing for each of the options identified in the menu. In addition, there may be other helper methods that will be required.

As above, develop one method at a time and test each one. When you are initially developing this class, it might be a good idea to manually load the data into the ArrayList rather than loading the data from the file. This way, you can developed and test the entire class that can process the data without worrying about file input. Once you are satisfied that this class meets the requirements, then you can work on the file input for loading the data.

1. Lastly, develop the **UserInterface** class. This will probably the smallest of the three classes and should be straightforward.

**Submission:**

Name your Project Folder: <lastname>\_<firstname>\_asg3. Submit this folder or the zipped version to the submit drive no later than the due date.