# Assignment-6

# (Use classes and objects, implement OOP, file I/O, and Exception handling)

# (Total: 200 points)

## Learning Objectives

- Get familiar with defining classes and create objects
- Get familiar with using ArrayList
- Apply basic OOP concepts
- Hands-on practices on reading data from .csv file.
- Hands-on practice on process text by using String class methods: e.g., startsWith()
- Hands-on practice on using try and catch blocks and throw statement to handle FileNotFoundException, and IOException

## Instructions

**Lab-5: Search phone number list** [100pts]: Download program files "**A6.zip**" from canvas assignment drop box in Canvas course site. Unzip the zipped folder, there are four files: **ResidentPhone.java, PhoneBook.java, PhoneBookTest.java**, and **sampleData.csv**.

**Part-1[40pts]:** Open **ResidentPhone.java**, and complete the definition of a class named **ResidentPhone** in that file. The ResidentPhone class should contain the following fields and methods:

- [5pts] three private instance variables(fields): ***last_name, first_name,*** and ***phone_number***.
- [10pts] two constructors: one default constructor (i.e., without parameters), and one constructor with two parameters. The parameters are used to provide initial values for resident name and phone number.
- [20pts] public methods: ***getLastName*** to return the last name of a ResidentPhone object. ***setLastName*** to set last name for a ResidentPhone object. ***getFirstName*** to return the first name of a ResidentPhone object. ***setFirstName*** to set first name for a ResidentPhone object. ***getPhoneNumber*** to return the phone number of a ResidentPhone, and ***setPhoneNumber*** to set a phone number for a ResidentPhone object;
- [5pts] public method: ***toString*** to return ResidentPhone info as a String.

**Part-2[50pts]**: Open **PhoneBook.java**, and complete the definition of a class named **PhoneBook** in that file. The PhoneBook class should contain the following fields and methods:

- [5pts] One private instance variable(field): an ArrayList named ***PhoneList*** of ResidentPhone objects.
- [10pts] Public method: ***addPhone*****.** The method has three parameters: parameter ***firstName*** is used to accept a resident first name as an argument, parameter ***lastName*** is used to accept a resident last name as an argument, and parameter ***number*** is used to accept a phone number as an argument. addPhone method adds a new ResidentPhone object to the ArrayList.of the ResidentPhone objects stored in a PhoneBook object instance.
- [25pts] Pubic method: ***searchPhone***. The method has one parameter: parameter ***key*** is used to accept a search keyword entered by users, and return an ArrayList of ResidentPhone objects in

which the resident names match the search keyword (see in Part-3 for more details about matching criteria).

[Hint: use String class method ***startsWith(String prefix)*** can check if a string stars with the specified prefix.
(https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html#startsWith(java.lang.String)**.**

- [10pts] Public method: **print**. The method prints out all resident phone info (including resident names and phone numbers) stored in the ArrayList of ResidentPhone.

**Part-3[55pts]:** Open **PhoneBookTest.java**, in the main method, add java code to complete the following steps:

(1) [5pts] Create a PhoneBook object instance. Use that PhoneBook object to create an ArrayList of ResidentPhone objects. The ResidentPhone objects in that ArrayList in the PhoneBook object are created based on the resident names and phone numbers read in from **sampleData.csv** file. The sampleData.csv contains the following list of names and phone numbers.

```
Harrison,Rose,555-2234
James,Jean,515-9098
Smith,William,312-1785
Smith,Brad,440-9924
```

(2) [20pts] Prompt users to enter data file name, and use try-catch blocks to handle FileNotFoundException, and other IOException separately. If user entered a data file other than the sampleData.csv or if did not find the sampleData.csv, then throw the FileNotFoundException, and then use a catch block to create a pop-up message says: "Cannot find the sample data file! Try again!", and prompt user to re-enter the file name until a correct file name (e.g., sampleData.csv) is entered.

(3) [10pts] Read in the data from the data file if no IOException occurs, and use PhoneBook object created in step-1 and its ***addPhone*** method to create ResidentPhone objects and add them to the ArrayList of ResidentPhone stored in that PhoneBook object.

(4) [5pts] Call ***print*** method of the PhoneBook object to print out the list of resident name and phone numbers from the ArrayList of ResidentPhone stored in that PhoneBook object.

(5) [15pts] Prompt users to enter a person's name or the first few characters of a name, and then use the PhoneBook object to search for the matched resident names (call the ***searchPhone*** method). If find matched resident names, display all matched names and corresponding phone numbers. If cannot find matched resident names, then print out a message as shown in the following sample outputs:

For example, if the user enters "Sm" or "Smith", the program should display the following names and phone numbers to the console area.

**Find Customers:**
**Smith, William: 312-1785**
**Smith, Brad: 440-9924**

If the user enters "Tina", the program should display the following message to the console area.

**Did not find customer with name starts with "Tina"**

**Part-4[55pts]:**

[10pts] In order to test your program, open "**sampleData.csv**" file (e.g., in excel or in a text editor) add 10 more persons' names and phone numbers following the same format: last name, first name, xxx-xxxx, and then save the file (should contain at least 14 persons' name and phone number records in the file).

[40pts] Compile and test your program to make sure there is no syntax, logical, or run-time error. Create a testing report (a pdf or a word file) named "**PhoneNumSearch _TestingReport**" that shows at least 4 testing cases: with or without exceptions, find or not find matched names; and the testing results from running your program for those testing cases.

[5pts] Submit your **sampleData.csv, PhoneBookTest.java, ResidentPhone.java, PhoneBook.java, and PhoneNumSearch_TestingReport** to the Canvas assignment drop box. If there are some bugs in the program that you cannot figure out, leave a message in the dropbox or in your program to mention them.

***Evaluation Criteria for Individual Lab Assignments***

- **Timeliness:** The assignments should be completed on time. Please see policy for late assignments below. Deadlines are given for all assignments.
- **Completeness:** All parts of a given assignment are to be submitted at the same time. However, if you have not completed an assignment by the time it is due, you are better off submitting what you have rather than nothing.
- **Accuracy:** The assignment has been completed according to the directions given. The deliverable delivered is what was asked for. Program needs to be run-able.
- **Efficiency:** The code should be concise and clear.
- **Readability:** Appropriate comments are required to help others understand. Also the code should be well formatted with nice alignment.

Please note: All your Java programs in your assignments should meet the above requirements. If in the case that there are bugs in your program that you did not be able to figure out, always remember to leave message or comments in the dropbox or in your source code to report the problems; otherwise, half of your assignment grade will be taken away.