Electronic Contact Book (ECB)

Assignment Due in Week 12

GENERAL DESCRIPTION

This is an individual programming assignment worth 20% of the assessment of this course.

In this individual assignment, you are going to develop a Java software application for electronic contact book (ECB). It's made up of two tasks:

Task 1: Design stage:

In this individual assignment, you are required to submit your class diagram design of the assignment. Your design should describe:

- All the classes needed
- The relationship of the classes.
- The fields and non-trivial methods (like getters and setters) of each class

Task 2: Coding stage:

In this individual assignment, you will create a Java software package to process the records in a phone book according to the given instructions/commands. Your Java software MUST provide ALL the following functionalities:

1. **Read from TWO inputs files: phone book file and instruction file**

   When the ECB system starts up, it assumes that an electronic phone book has the contact information as given in the ***phone book file***, and it manages the contact records according to the instructions in the ***instruction file***.

   - ***Phone book file*** contains contact information in a predefined format;
   - ***Instruction file*** lists instructions/commands to be performed on the records. The instructions/commands can be: "add", "delete", "query", and "save".

2. **Add a record (a person's contact details) to your phone book**

   For instance, the instruction

   - ***add*** *name Jo Bloggs; birthday 08-07-1980; phone 88884444; address 9001 Chester Crescent*

   is supposed to add/update a record for a person with name "Jo Bloggs", birthday 8/7/1980, phone number 88884444 and address "9001 Chester Crescent".

   Your ECB system checks whether this is an existing record:

   - If both person name and birthday are identical to those of an existing record in your phone book, the existing record will be updated with the new input information. E.g., update the items of *address, email*, and *phone*.

- Otherwise your system adds the new valid record to the list.

3. **Delete record(s) from your list by name**

   For instance, the instruction

   - ***delete*** *Jeff Vader*

     indicates deleting the record(s) with name "Jeff Vader" from the list.

   - ***delete*** *Jeff Vader ; 8-07-1980*

     indicates deleting the record with name "Jeff Vader" and birthday "08/07/1980" from the list.

4. **Query the records by person name, birthday, or phone number.**

   For instance,

   - **query** birthday 8-09-1991
   - **query** name David Joans
   - **query** phone 9110110

   NOTE: If there are more than one query results, all the query results should be saved in ascending order of person name and birthday.

5. **Save the resulting data collection to output file(s)**

   - **Save** the resulting data collection of the instructions of "add" and "delete" into the specified output file
   - **Save** the query results to a separate specified report file. When there are more than one "query" command, append the latest query results to the end of the report file. Separate the results of different query instructions using dash lines with query instructions.

DATA FORMAT

- **Input *phone book file*** has records as the following format:
  - Each record has information about a person. There may be 5 fields:

    1. ***name*** in the form of a string of forename(s) and surname, all on one line; and the name cannot include numeric or punctuation characters
    2. ***birthday*** in form of dd-mm-yyyy (leading zero in day or month may be omitted,

       i.e., both 05-04-2012 and 5-4-2013 are valid)
    3. ***phone*** in the form of a sequence of digits. You must *ignore* any leading zeroes.
    4. ***address*** in the form of a string that may span more than one line

5. ***e-mail*** which must be a valid e-mail address, e.g., a string with alphabetic, numeric and punctuation characters and an "at" (@) symbol inside and with no gaps, such as: abcdefg.123@gmail.com

- ***name*** and ***birthday*** are compulsory and required for all records when they are read in from the phone book file. That is, you may have a person record with name, birthday and just e-mail address, or another record with all five fields.
- Fields may occur in *any order*
- Records are separated by blank line(s)

- **Output Format**
  - The output file should have all the appropriate and valid records in it
  - The output records should follow a consistent format
  - Each field should fit on ONE line only
  - Again, records should be separated by single blank line.

EXAMPLE:

- "phonesample1.txt" is a sample input phone book file, with records as below:

name	Josephine Esmerelda Bloggs
birthday	13-5-1980
phone	99887766
email	j.e.bloggs@fanfare.com.au
name	Pac Man
birthday	27-03-1992
email	pac.man@gamebots.com
address	27 New Street, Birmingham, NSW,
	Australia
phone	000333998877

- "instructsample1.txt" is a sample instruction file with a single instruction: **Save**
- The output file is expected to contain the records as below:

name	Josephine Esmerelda Bloggs
birthday	13-5-1980
phone	99887766
email	j.e.bloggs@fanfare.com.au
name	Pac Man
birthday	27-3-1992
phone	333998877
email	pac.man@gamebots.com
address	27 New Street, Birmingham, NSW, Australia

IMPORTANT NOTES

1. Your code **must** make use of at least one collection e.g., ArrayList.
2. Your system must be able to handle both normal cases and difficult cases.
3. You MUST NOT build a graphical user interface.
4. You need to do systematic testing of the classes you create.
5. Do NOT hard-code the names for the input and output files.
6. You MUST be demonstrated during your laboratory class to your tutor in week 12. **Absenting the demo will incur a penalty of 4 marks**. If you miss the demonstration in week12 because of serious illness or misadventure, you should request ***Special Consideration*** using the appropriate forms **within a week**.
7. Recall that **the University takes plagiarism very seriously**, and it would be useful for you to review this policy here.

YOUR SOFTWARE *MUST*

- be put into the java project package with name "comp9103"
- be invoked in the format:

java comp9103.Comp phonebookfile instructionfile outputfile reportfile

where *Comp* is the name for the main class, ***phonebookfile, instructionfile, outputfile, and reportfile*** are names of files that will change between runs of the program.

Example: In the following example,

java comp9103.Comp phonesample05.txt instructsample05.txt outputfile05.txt reportfile05.txt

your software reads and parses contact information and corresponding instructions in ***phonesample05.txt*** and ***instructsample05.txt***, executes the valid instruction(s) on the contact record(s), and outputs all the resulting valid record(s) and query result(s) to ***outputfile05.txt*** and to ***reportfile05.txt*** respectively.

Your software should also be able to accept absolute path as input arguments. For example:

java comp9103.Comp d:\phonesample05.txt c:\instructsample05.txt d: \output\outputfile05.txt d:\output\reportfile05.txt

MARKING SCHEME

There are 4 parameters on which your submission will be marked:

1. **Design** [2 marks]
2. **Prototype Demo** [2 marks]: in week 12
3. **Implementation and testing** [14 marks]: at the final submission
4. **Standard of coding** [2 marks]: at the final submission.

SUBMISSION

Complete assignment

- Late submissions will incur a penalty of 2 marks **per day**.
- You must demonstrate your program to your tutor in your scheduled lab in week 12:
  - Your program (all the java source code), and
  - Documentation of your software design, e.g. UML diagram and description of your design.
- You must ALSO upload a single ***.zip*** archive onto Canvas before the deadline. The .zip archive must be named with your login ID, e.g. if your login name is "abcd1234" then the archive must be called "abcd1234.zip". It should contain:
  - Your program (all the java source code) in correct architecture, and
  - A description of the design of your program with UML diagram.