Software Development in Java

Case Study
Notes on Assignment

<u>QUIZ 2 in the Lab Session</u>
Assignment Part one

Notes on Assignment: ECB

Inputs – phone book file

- text files contains contact information in a predefined format
  - each field on a separate line
  - fields may occur in any order
  - contacts are separate by blank line(s)

- Invalid fields and invalid contacts should be ignored.

Inputs – instruction file

- A text file with *instructions on each* line.
- The instructions include:
  - Add,
  - Delete,
  - Query,
  - Save
- Each instruction begins with one of these four command words followed by a list of parameters (except for "save")
- Instructions should be processed following their orders in file
- If there are errors in the parameter(s)/command, ignore the invalid instruction and proceed to the next instruction

Outputs – output file

- A text file which records contacts information organized in consistent format, i.e. the order of fields should be consistent.
- Final results of the contacts after executing a series of “Add” and “Delete” commands in instruction file

Outputs – report file

- A text file which records the reports for each “query” command
- Results of each “query” command should be separated from others using dash lines.
- Results of a “query” command should only reflect the current status of the contacts list, i.e. any commands following this “query” command should not affect the results of this “query”.

Outline

start up

- read in the command-line arguments args[0], args[1], args[2] and args[3] (these will be the file names of phone book, instruction, output and report files)

read in contacts from phone book file

- File f = new File(args[0]);
- and make a Scanner (don’t forget the try/catch block!) scanning the file

read in instructions from an instruction file

- f = new File(args[1]);
- and make a Scanner scanning the data file

Scanning the data file

```
let in be a Scanner object, accessing the phone book file
while in has something to read {
    read a line
    if the line is not empty then
        scan a contact
    else //have finished a contact
    }
```

scan a contact

```
make a new Scanner for the line
read the first word, call it word
if word is "name" then
    read the rest of the line and record the name of the contact
else if word is "birthday" then
    read the valid date and record it
else if word is "phone" then
    read the valid content and record it
else if word is "email" then
    read the valid content and record it
else if word is "address" then
    read the valid content and record it
```

- **NOTE:** the fields can be in any order, a loop is needed

read in instructions from the instruction file

```
let in be a Scanner accessing the instructions file

while in has something to read{
    read a line & make another Scanner object for the line

    let command be the first word of the line
    if command is "add" then
        add a stock of food with the valid information on the rest of the line

    else if command is "delete" then
        find the food with the valid information on the rest of the line
        remove the item in the collection

    else if command is "QUERY" then
        query the items and may sort the results
        save the query results

    else
        invalid instructions should be ignored
}
```

Add

To add a contact we must scan all the information on the line.

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

So break the line into fields:

```
   . . .
 if command is "add" then
     read the rest of the line into a String
     split the String into multiple Strings representing each parameter
     validate the parameters
     make a contact record if the parameters are valid
     add the contact to the collection
```

Making a contact record with a Contact Class

- Fields, constructors, and methods
- Methods for validating fields:
  - hasValidName:
    - return true if and only if the name(s) is correct.
  - hasValidBirthday:
    - return true if and only if birthday is a correct date
  - hasValidEmail:
    - return true if and only if the email address conforms to the common email format.

…

More Notes

- Refer to the sample files for the formats of output and report files. DO NOT invent your own format.
- **We will check academic dishonesty by using software packages as well as manual investigation. The suspected cases will be passed to school / university for further process.**