Winter 2018, McGill, CCCS-301 Programming Techniques 2

Homework 1

Due: February 19, 2018, at 6:00pm.

Submission must be made electronically via MyCourses site.

Late submissions will NOT be accepted.

PLAGIARISM WARNING

This is individual work. The works you submit as solutions are routinely uploaded to a plagiarism-detection system that uses metrics to detect works that are more similar than others. It makes no difference whether you were a **donor** or **recipient** of the copied work – the consequences are the same. Do not risk having such incident on your permanent academic record and being irreversibly removed from the program. You may consult with others, including colleagues from the same class. But do your own work yourself and do not share your work with anyone.

TASK DESCRIPTION

Main concepts that are part of this task include:
User input, advanced Scanner use, menu, basic use of exceptions.

Create Java project `Homework1` in Netbeans.

You will create a program which manipulates records of data with people's name and age.

It then displays a menu with 4 options:

```
1. Display records
2. Add a record
3. Delete a record
4. Modify a record
5. Exit.
```

You should implement all 5 options appropriately, as described below.

Once one option has been chosen and executed you will display the menu again and wait for the user to make another choice.

Internal Data Structures

You will create class `PersonData` which will contain fields `String name`, `int age`, and `String email`.

You will then encapsulate these fields by creating public accessors and making fields private.

You will add a constructor that takes three parameters `name`, `age`, `email` and uses them to initialize the fields of newly created object.

In the *setters* you will verify that name is at least 2 characters long and age is between 0 and 150 (inclusive), and email looks like a valid email (use Regular Expressions matching). If they do not your setter will throw ArgumentInvalidException with message passed to its constructor explaining what the problem exactly was, e.g. “Email seems invalid”.

ArgumentInException is a **checked** exception defined as follows. You should indent it better.

```
class ArgumentInvalidException extends Exception {
      ArgumentInvalidException() { super(); }
      ArgumentInvalidException(String msg) { super(msg); }
      ArgumentInvalidException(String msg, Throwable cause)
                { super(msg, cause); }
}
```

Main class

In the main class, Homework1, you will add a `static` field of type `ArrayList<PersonData>` which you will use to store a collection of objects `PersonData`.

You will also define a static field input of type Scanner and initialize it as follows:

```
static Scanner input = new Scanner(System.in);
```

You will also define a `static` helper method `inputInt()` that returns `int`. That method will use the `Scanner` in `input` to retrieve the integer value entered by user. If what user entered doesn't parse as integer value Scanner's method will throw an exception. You are to catch this exception and ask user for integer input again, until user provides a valid integer. Then you are to return that integer as the result of `inputInt()` method.

<u>Hint:</u> put an infinite loop around try-catch.

You will also define a `static` method `displayMenu()` that will display the menu of 5 items shown at the beginning.

You will also define a `static` method `getMenuChoice()` that returns an integer. This method will ask user for an integer value (menu choice) and will continue asking until both conditions are satisfied: a) value entered is a valid integer, b) value entered is an integer within the 1-5 scope. Once user enters a valid choice you will return that choice as an integer.

Menu item: Display records

When displaying a record each record is numbered, so that option “Delete record” can ask for a record number. Records are displayed one record per line, e.g.:

```
1: Jerry, 45, jerry@gmail.com
2: Timmy Toe, 32, timmy@toe.com
3: Hilary Hanoe, 73, hilary@hotmail.com
```

Menu item: Add a record

You will ask user for 3 inputs of name, age, email. When asking for age you will use `inputInt()` method you defined. When instantiating PersonData object remember you need to try/catch the possible exception. If an exception occurred inform the user that adding record failed. If there was no exception inform the user that the operation succeeded. Example interaction:

```
Enter name: J
Entere age: abc3
Invalid integer, try again: zzz
Invalid integer, try again: 39
Enter email: j@jj.com
Error creating record: Name too short.
```

Note: the error message "Name too short" was extracted using `getMessage()` method on the Exception you caught.
Note: the repeated "Invalid integer, try again:" display was created by `inputInt()` method.

Menu item: Delete a record

You will ask user for record number using inputInt() and then delete this record from collection. If the record didn't exist you will display an error message. If the record was successfully deleted you will display message confirming it. In both cases the program will continue back to the menu.

Menu item: Modify a record

Similar to Add a record, but you will first ask user, using inputInt() for the number of record to be replaced. Then you will verify whether such record exists. If not you will display an error message and continue to menu. If record does exist you will ask for new values, identical to "Add a record".

Making data persistent

On application start, before displaying the menu, your program must load data lines from file "data.txt", if such file exists. If it does not exist execution continues normally. That is done by calling `loadDataFromFile()` method as one of the first things in the main() method of the program. That method will load data from file, line by line into `personList` using PersonData constructor that takes `dataLine` - single line of the file.

Just before application exits (when user selected "5 – Exit" option) your program will call `saveDataToFile()` method in order to save data from `personList` to "data.txt" file. If the file does not exist it will be crated. If it does exist it will be overwritten. Each element of the list is transformed into its textual representation, as a single line using `toDataString()` method.

PARTIAL CODE

Here's a partial code that should make it easier to get started. Note that this is provided solely as an illustration of some of the above stated requirements, and as a starting point which you are expected to have to modify and build upon.

```
class PersonData {
     // TODO: This class is incomplete
     private String name;
     private int age;
     private String email;
     // constructor used when creating objects from user inputs
     PersonData(String name, int age, String email) { … }
     // constructor used to read from data file lines
     PersonData(String dataLine)
                    throws ArgumentInvalidException { … }
     String toDataString() { … }
}

public class Homework1 {

     static Scanner input = new Scanner(System.in);

     static ArrayList<PersonData> personList = new ArrayList<>();

     static int inputInt() { /* TODO  */ }

     static void displayMenu() { /* TODO */ }

     static int getMenuChoice() { /* TODO */ }

     static void loadDataFromFile() throws IOException { /* TODO */ }

     static void saveDataToFile() throws IOException { /* TODO */ }

     public static void main(String [] args) {
          // TODO
     }

}
```

GENERAL NOTES

For maximum marks your program must conform to industry best practices, such as (but not limited to): naming conventions, proper decomposition of the tasks and structuring of code, exception handling and propagation.

WHAT TO SUBMIT

ZIP file of your entire Netbeans project.