A Virtual Student Management System

Introduction

The assignment must be done using the BlueJ environment. All user input to the system, and system output to the user, must be done at the BlueJ terminal window. No other graphical interfaces are to be used (nor assessed) in your program.

Any points needing clarification should be discussed with your tutor in your lab classes. You should not make any assumptions about the program without consulting your tutor.

Learning outcomes

1) Design, construct, test and document small computer programs using Java.
2) Interpret and demonstrate software engineering principles of maintainability, readability, and modularisation.
3) Explain and apply the concepts of the "object-oriented" style of programming.

Specification

For this assignment you are required to write a program that implements a simple ***Virtual Student Management System*** for a very small, fictional, school. This section specifies the required functionality of the program. ***Only a simple text interface (using the BlueJ Terminal Window) is required for this program***; however, more marks will be gained for a program that is easy to follow with clear information/error messages to the user.

The aim of the ***Virtual Student Management System*** is for an administrative user (the school's enrolment staff – ***NOT*** students) to maintain a ***database***[1] of the students enrolled in various subjects in this fictional school. This school only offers 10 (ten) subjects, with various credit points allocated – see the list below for details:

[1] please note that in the context of this assignment, "database" refers to an **in-memory data structure** (e.g. a Java Collection), and ***NOT*** a real database in the traditional sense (e.g. SQL).

Subject Name	Credit Point
Basket Weaving	3
Social Media	3
Rocket Science	3
English	4
Maths	4
Finance	4
Operating Systems	5
Cyber Security	5
Systems Design	5
Programming	6

Note: although only 10 subjects are offered now, the school would like to introduce more subjects in the future – so you should not hardcode the above data into your program.

The aim of the system is to allow the staff to manage students, enrol them in subjects, and perform simple reports.

Even though this program is functionally very different from the program you wrote in Assignment 1, you should be able to re-use much of your previous code here - if you have designed the classes/logic in your previous program properly. This is one of the major benefits of an object-oriented program - the ability to re-use classes.

The ***Virtual Student Management System*** should provide the following features:

- maintains a list (using a ***Java Collection*** class) of **`Student`** objects
    - each **`Student`** object represents a person who is allowed to enrol in the fictional school
- maintains a list of exactly **10 `Subject`** objects (for now)
    - each **`Subject`** object represents a subject (with a credit point allocated to it) which is available in the fictional school
- enrols each student in multiple subjects (***up to a maximum of 15 credit points***)
- lists the details of an existing student

- produces a report of students based on some criteria
- loads a list of students (with their enrolled subjects) from a text file
- saves the list of current students (with their enrolled subjects) to a text file

You are to demonstrate the following programming techniques in your program:

- reading/writing data from/to text files
- using appropriate ***Java Collection*** class or classes to store data (including embedded objects)
- manipulating the data in the collection(s)
- performing simple searches, filtered by some given criteria
- using program constructs such as repetitions & selections
- using appropriate classes to represent objects in the program

You are also required to produce a **partial** ***Test Strategy*** for your program.

Program Logic

When the *Virtual Student Management System* starts, it should automatically load 2 text files:

- "**`students.txt`**" which contains details of all students currently stored in the system
- "**`subjects.txt`**" which contains details of all subjects currently stored in the system

The actual formats of these text files are described later in this document. The data loaded should be stored in some appropriate data structures (the "databases" – stored in memory). ***No other reading from or writing to file is required while the program is in operation, until the user chooses to exit, at which point the program saves all the data in memory back to the same text file*** (**`students.txt`**) - the **`subjects`** data does not need to be saved as it is not modified by this program (for now).

In other words, all the file I/O operations are performed automatically by the program, once at the start and once at the end, and require no interactions with the user.

The system will also store the **10 Subjects** in some appropriate data structures. Even though the school currently only offers 10 subjects, you must design your program so that it is easy to add more subjects in the future.

When the program is running, it should repeatedly display a **main menu** with these options:

```
(1) Add New Student

(2) Delete Student

(3) Suspend/Unsuspend Student

(4) Edit Student

(5) List Students by Subjects

(6) List Suspended Students

(7) List All Students

(8) Exit System
```

Option (1) allows the user to add a new student into the database. The user should be asked for the student's details, plus what subject(s) to "enrol" him/her in. Duplicate students (same name & ID) should be rejected.

- when entering subjects for the student, the subject names should be chosen from the list of known subjects (as read in from the file)

Option (2) allows the user to remove an existing student from the database.

Option (3) allows the user to find an existing student in the database, and mark the student's status as appropriate.

Option (4) allows the user to find an existing student in the database and edit his subject list (add or delete subjects). The user should be asked to enter a name to search for. If there are more than one student with the same name (but different IDs), the user should be asked to choose the correct one.

- when entering subjects for the student, the subject names should be chosen from the list of known subjects (as read in from the file)
- only the subject list is editable in this option

Option (5) allows the user to display a list of students enrolled in a subject. The user should be asked for a subject name; a list of all students (if any) enrolled in that subject should then be displayed. The user can also specify multiple subject names, to list all students enrolled in all of the specified subjects (e.g. all students enrolled in *English AND Maths AND Programming*). The output should not include suspended students.

Option (6) allows the user to display a list of all students who have been suspended.

Option (7) allows the user to display a list of all students enrolled in the school.

Option (8) exits the program. All the students currently in memory are automatically saved to "**`students.txt`**".

Inputs other than 1-8 should be rejected, and an error message printed. The menu should be displayed repeatedly, until the user chooses Option (8).

Additional Notes:

Both the main menu and the sub-menu must be displayed repeatedly after each menu operation, until the user chooses the appropriate exit option. All invalid inputs (including non-numeric or other special characters) should be rejected, and an error message printed.

If the user chooses **Options (1) or (4)** in the main menu, the **subject name** MUST be selected from a list which came from the data read (at the start of the program) from **`subjects.txt`**.

Your program must deal in a sensible way with any invalid values entered by the user.

For all the user interactions, the inputs/outputs can be formatted in many different ways. ***Discuss with your tutor regarding how you should implement these in your program***.

Your user interface need not be exactly as shown in the examples shown above. However, ***you should discuss with your tutor about what to include in your user interface.*** Keep it simple.

Important: see the *Program Design* section below for a description of what classes you need to implement in your program. Failure to implement those required classes will cause loss of marks.

Important Requirements

You should satisfy the following requirements when implementing your program:

- a **`Student`** object remembers the following data:
    - **`name: String`**
        - must not be a blank string
        - must be only alphabetic
        - may contain multiple words
    - **`ID: int`**
        - a 3-digit number between 111-999 (inclusive)
    - **`suspended status: boolean`**
    - **`subjects: a "database" of unique subjects`**
        - total number of credit points of the subjects must be <= 15
- a **`Subject`** object remembers the following data:
    - **`name: a String`**
        - must not be a blank string
        - can be numeric/alphabetic
    - **`credit points: int`**
        - a number between 1-6 (inclusive)
    - Note: for this program, the "subjects" are read from a text file ("**`subjects.txt`**"), and are not editable while the program is running.
- you may assume that the input data files are always in the correct formats (see below) - ie. there is no need to validate the actual data when reading them in.
- all program operations must be applied to the in-memory data structures - ***there must not be constant reading/writing to/from the data file***, except once at the start (when the program loads all data from the files) and once at the end (when the program saves all data back to the file, when it exits).
- *the program must not crash when accepting user inputs, regardless of what the user enters.*

- all student IDs are unique - if a student is already in the database, trying to add another with the same ID again should generate an error.
- there is no limit to how many students can be enrolled.
- each student can enrol in multiple subjects - as long as the subjects' total credit points do not exceed 15.
- all searches use exact matches (eg. "**andy**" will not match "**andy cheng**"); however, the search strings are ***not case-sensitive*** (eg. "**Sue Smith**" is considered to be the same as "**SUE smith**").

Input File Format

The first input data file (**students.txt**) has the following format for each line (note that there is no empty space in between a word and a comma). Note that the number of fields on each line can be different – the 4th field (shown in **green** below) onwards represent the multiple subjects the student can be enrolled in.

```
studentName,ID,suspendedStatus,subjectName1,creditPoint1,...
```

(the fields are separated by commas)

A sample **students.txt**:

```
David Smith,111,true,Basket Weaving,3,Programming,6,Maths,4

Andy Cheng,333,true,Programming,6,Operating Systems,5,Maths,4

Susan Dally,123,false

Zak Whatever,199,true,Rocket Science,3,Social Media,3,English,4,Basket Weaving,3

Russell Crowe,222,true,Maths,4

John Citizen,777,true

Jane Citizen,555,true,Social Media,3,Rocket Science,3,Systems Design,5
```

The second input data file (**subjects.txt**) has the following format for each line (note that there is no empty space in between a word and a comma).

```
subjectName,creditPoints
```

(the fields are separated by commas)

A sample **subjects.txt**:

```
Basket Weaving,3
Social Media,3
Rocket Science,3
English,4
Maths,4
Finance,4
Operating Systems,5
Cyber Security,5
Systems Design,5
Programming,6
```

You may assume:

- the strings which represent the various fields (**studentName/subjectName/e**tc) do not contain commas.
- this program does not provide the functionality to edit the **subject** list (i.e. use a text editor to edit "**subjects.txt**" if you wish to modify the list). However, keep in mind that the school may want to do this in future – so design your program so that this functionality can be easily added later.

Program Design

Your program must demonstrate your understanding of the object-oriented concepts and general programming constructs presented in FIT9131. Consider carefully your choice of classes, how they interact with each other, and the implementations of the fields and methods of each class.

You must use appropriate data structures to store the various objects (see below) in the program. ***If you do not understand what this means, ask your tutor now***.

You must be able to justify the choice of the data structures during your interview. You must document any additional assumptions you made.

Appropriate validations of values for fields and local variables should also be implemented. You should not allow an object of a class to be initialized/set to an invalid state (ie. put some simple validations in your mutator methods).

Discuss with your tutor what classes are appropriate, and how they interact with each other. The main requirements are:

(1) the students must be implemented as objects, and they must be stored in some appropriate Java Collections (e.g. an ArrayList of Student). The list of enrolled subjects for each student must also be stored in a Java Collection.

(2) the subjects must be implemented as objects, and they must be stored in some appropriate Java Collections (e.g. an ArrayList of Subject)

Your program must consist of at least these classes:

- **`School`** – main class, which contains the program's main logic
- **`Student`** – represents a single student
- **`Subject`** – represents a single subject
- **`StudentDatabase`** – represents a list of students
- **`SubjectDatabase`** – represents a list of subjects

Any other appropriate classes (e.g. a **`Menu`** class) are to be discussed with your own tutor

Your program must deal with invalid values entered by the user in a sensible manner. For instance, if a user enters "**`abc`**" instead of a number for the menu options, your program should not crash.

Exception handling should be used where appropriate.