SOFT6017 – Problem Solving & Programming 2 (ITM1/WEB1)

Individual Assignment Part 2 (10%)

Due Date: 12th May @ 5pm

OVERVIEW

In part 1 of the individual assignment, you developed a single-user module registration system with no persistent storage. An issue with this is that once the program ends, the choices made are lost.

What would be desirable is if the choices could be made permanent, i.e. saved to a file. This will allow for subsequent runs of the program to pick up where the last run ended. It would also be desirable if the program could handle multiple students.

In this administration system, only 2 elective module choices are to be stored per student, not the 6 required in part 1. It is assumed that the other 4 modules have already been chosen and are not in the available elective modules list.

It would also be desirable to have the ability to add new modules (old modules are never removed from the system).

Also required is a security mechanism that can **authenticate** the credentials (i.e. a username and password) of administrative users and determine whether they are **authorised** to add modules or choose modules.

EXISITING CODE

You have a choice. You can either continue on from where you left off with part 1 of the assignment, or you can use the pre-prepared version (see the additional attached program in the electronic version of the assignment on Blackboard). If you decide to continue from your previous effort (which would be encouraged if you were happy with your efforts), you will not be penalised for any inadequacies carried forward from the first version. You may use the supplied code as inspiration for new code if you want to use your previous effort as a starting point.

PERSISTENCE SPECIFICATION

The module file (modules.txt):

When the program starts, it should load the available modules from a file in this format:

```
ModuleCode~ModuleTitle~Capacity~CurrentEnrolment
```

e.g.

```
SOFT6017~Problem Solving & Programming 2~40~21
MATH6000~Essential Mathematical Skills~30~7
```

And so on with additional modules.

Capacity refers to the maximum number of students that can be enrolled on the module. CurrentEnrolment refers to the number of students that have so far been registered on the module. Each time you save a student choice, you must update the CurrentEnrolment. Obviously, **CurrentEnrolment should never exceed the Capacity.**

In this file format, the **delimeter** is the "~" character, also known as the Tilda character. It is generally not used in module titles and so is a good delimeter. However, it means that any module title should not contain a Tilda as this would adversely affect the splitting up of records into the 2 fields.

Just before the program ends, the module data in memory should be written to the modules.txt file, overwriting the old data.

The users file (users.txt):

Also when the program starts up, the users of the system should be loaded. This file is in the format:

```
UserID~Password~Name
```

e.g.

```
bsmith~cork123~Barry Smith
joconnor~myfoot~June O'Connor
henright~liverpool18~Harry Enright
```

The module choices file (choices.txt):

When the program starts up, the module choices should be loaded. This will allow an administrator to check the current choices of a student, if they exist, before making any changes or additions.

The records in the file are in the following format:

```
StudentID~StudentName~ModuleCode1~ModuleCode2
```

e.g.

```
R01234567~Rob Lyle~MATH6000~SOFT6017
R00459982~Rachel Madder~SOFT6030~unspecified
```

Just before the program ends, you must write all of the choices data stored in memory to the choices.txt file, overwriting the old contents.

FUNCTIONAL SPECIFICATION

When the program starts, a login screen will display for the Module Registration System (MRS).

```
CIT MRS – LOGIN
----------------------------------------
User ID: bsmith
Password: cork123
```

If correct credentials are supplied, the main menu will be displayed.

```
CIT MRS - Bob Smith – Main Menu
----------------------------------------
1 – Add Student
2 – List Students
3 – Add Module
4 – List Modules
5 – Add Student Choice
6 – Remove Student Choice
7 – Exit

Choose option >>
```

When adding a student:

```
CIT MRS – Bob Smith – Add Student
----------------------------------------
Student ID: R05554554
First Name: Anna
Surname: Harrison

Student with id R05554554 added successfully.
```

Listing students:

```
CIT MRS – Bob Smith – List Students
----------------------------------------
++++++++++++++++++++++++++
Student ID: R01234567
Name: Rob Lyle
Choice 1: MATH6000
Choice 2: SOFT6017
++++++++++++++++++++++++++
Student ID: R00459982
Name: Rachel Madder
Choice 1: SOFT6030
Choice 2: unspecified
++++++++++++++++++++++++++
Student ID: R05554554
Name: Anna Harrison
Choice 1: unspecified
```

```
Choice 2: unspecified
```

Adding a module:

```
CIT MRS – Bob Cole – Add Module
---------------------------------------
Module Code: TEST6000
Module Title: Test Title
Max. Allocation: 40

Module added successfully.
```

Listing Modules:

```
CIT MRS – Bob Cole – List Modules
---------------------------------------
+++++++++++++++++++++++++++++++++++++++++++
Module Code: SOFT6017
Module Title: Problem Solving & Programming 2
Allocated: 21/40
+++++++++++++++++++++++++++++++++++++++++++
Module Code: MATH6000
Module Title: Essential Mathematical Skills
Allocated: 7/30
+++++++++++++++++++++++++++++++++++++++++++
Module Code: TEST6000
Module Title: Test Title
Allocated: 0/40
```

Adding a student choice:

```
CIT MRS – Bob Cole – Add Student Choice
---------------------------------------
Student ID: R05554554
Module Code: TEST6000

Student choice added successfully.
```

Removing a student choice:

```
CIT MRS – Bob Cole – Remove Student Choice
---------------------------------------
Student ID: R01234567

Found the following student details...
Name: Rob Lyle
Choice 1: MATH6000
Choice 2: SOFT6017

Remove option 1 or 2? 1
```

```
Module MATH6000 removed from student R01234567.
```

Exit option:

```
CIT MRS – Bob Cole – Exit
-----------------------------------------
Exiting....
Modules saved.
Choices saved.
Goodbye.
```

TESTING

Add a JUnit test class that will test at least 3 methods that are new in Part 2. Candidate methods would be ones that return a true or false or a simple value, such as an index position – or you might test that a method has added or removed from an array as expected.

FURTHER INSTRUCTIONS

- Modularise your code with appropriate methods to avoid duplication and aid testability.
- Remember: if you can't test any methods with JUnit, your design may be at fault.
- Use Javadoc (we will cover this after Easter, if you haven't used it already) to document your methods. Also use standard comments in other parts of your code when additional explanation is required.
- Add as much validation as you can, for example:
  - adding a module choice for a student that already has both modules chosen
  - adding a module choice for a non-existent module code
  - adding a module choice for a module that is already at its maximum allocation limit (to test this, add a new module with a very small max allocation, e.g. 3 students)
  - adding a student id that already exists
  - adding a module code that already exists
  - entering an incorrect user id or password
  - entering an incorrect menu option
  - entering a non-integer value for maximum allocation

ADVICE

Data Structures

The data structures (should use parallel arrays in this assignment) in part 2 will be significantly different than part 1 to accommodate more students for the module choices. Also the available modules data structure requires 2 additional fields: maximum allocation and current allocation. There is also a new data structure required for the administrative users.

Prototyping

Start with getting the menu system working with some basic print outs from each piece of functionality, as follows:

```
CIT MRS - Bob Smith – Main Menu
---------------------------------------
1 – Add Student
2 – List Students
3 – Add Module
4 – List Modules
5 – Add Student Choice
6 – Exit

Choose option >> 1

This method will add a student

CIT MRS - Bob Smith – Main Menu
---------------------------------------
1 – Add Student
2 – List Students
3 – Add Module
4 – List Modules
5 – Add Student Choice
6 – Exit

Choose option >> 3

This method will add a module

CIT MRS - Bob Smith – Main Menu
---------------------------------------
1 – Add Student
2 – List Students
3 – Add Module
4 – List Modules
5 – Add Student Choice
6 – Exit

Choose option >> 5

This method will add a student choice
```

Then continue to develop each new module / method to replace the simple printouts. This approach will get the skeleton in place for your program.

Module by Module

A good place to start might be reading in all of the data from the files to fill the data structures (your parallel arrays). You might decide to have 1 module per file read. You will be reading from the files

record by record. You should use the String.split("~") approach to create an array of fields from each record, which you can then assign to the parallel arrays. You will be reading in Strings. For the *maximum allocation* and *current allocation* fields, you will need to convert from string to integer.

You can do it like this:

```
// assigning an integer value to the modules parallel int arrays
MaxAllocations[i] = Integer.parseInt(moduleFields[2]);
CurrAllocations[i] = Integer.parseInt(moduleFields[3]);
```

Reuse of Code from Part 1

Reuse as much of the code from part 1 as you can. For example, the code to search for a module code within an array and returning its index position can be reused in part 2.

MARKING SCHEME

A rubric will be added shortly to the electronic assignment on Blackboard.

GUIDELINES

- You must submit your entire eclipse project – you should zip up the project folder in your workspace, including all classes (main program and JUnit class) and text files (users.txt, modules.txt, choices.txt), and attach it to the electronic assignment.
- Please refer to the student regulations with regard to penalties for plagiarism. Copying the code of another student is a serious offence and will be punishable for both students – the person who wrote the original code and the person who copied it. It is remarkably easy to check for plagiarism. It isn't worth the risk and will stand you in good stead for the 50% final exam where you will need to be able to produce your own work under a time pressure.
- As usual, there will be a 10 mark penalty for submitting up to 1 week late, a 20 mark penalty for submitting between 1 and 2 weeks late, and 100 mark deduction for being more than 2 weeks late.