Project Report, Module 3
Course Manager

Lukas Arnroth (h17lukar)

October 28, 2017

Contents

1 Introduction

In this project report I will present an outline to my project in which I have created a personal course manager. A large part of this report will focus on the three classes *Menu*, *CourseList* and *Course* as their interaction makes the program. In section 3 a brief user guide will be presented. The main idea of the program is to give a simple tool for a student to keep track of his or her courses. The user is able to save the contents between sessions through a text file – making the program almost usable!

2 Classes

In this section the classes will be presented along with the entire program presented as a UML diagram.

2.1 Course

The Course class is the base of the program with the responsibility of holding all the information about the individual courses, such as course name and hours studied etc.

Looking at figure 1 it should be noted that the class' to string method is overloaded. If a boolean is supplied to the to string method it will basically be printed in the format of a .csv file, simplifying the problem of reading the data back in to the program. In figure 1 it is shown that Course has a constructor which takes a string and a boolean, this is for constructing an object based on a line in a text file written by the progam.

2.2 CourseList

Next part is the CourseList class which serves as a placeholder for all the courses. It seems natural to separate the task of having the information of a course and the task of storing all these courses into two separate classes. The CourseList therefore corraborates with both classes in figure 1, it has to know the Course instances.

The information given back to the user from this class is the information which is aggregated across all courses. For example, having the information of one instance of Course will not give us the total hours studied for *all* courses (unless the array only holds one course that is). It collaborates with the Course class through methods such as addCourse which takes a course object and appends it to the array list. The Course class mainly collaborates with the menu option through CourseList. Whenever the user is displayed the information of a course, it is done through the .get() method and the to string method of the Course class.

2.3 Menu

This class has the responsibility of tying the classes together. It works by sending the user between getMenu, which prints the menu and asks the user to choose an option which takes the user to userChoice were the option choosen will be executed. It has han instance of both the Course and CourseList class. The user only gets access to the CourseList and Course classes through the Menu class. So the purpose of this class is to enable the user to utilize the Course and CourseList classes.

The program can be summed up as the user being sent between the two methods of the Menu class!

2.4 UML diagram

In figure 1 the program is presented as UML diagrams.

3 User Guide

In this final section I will give a brief user guid along with some comments of my work. Each option from the original menu, apart from *(6) Exit* which is just a System.exit(0), will be introduced. The menu has the following row-based design and will ask the user for a value of 1-6 to use the different options which is displayed in figure 2. Note that there is no input validation for this entry, so not following the instructions in the bottom of figure 2 will cause the program to break down. If an integer is entered which is not 1 to 5 it will be caught by the final else statement of the userChoice method of the Menu class (see the UML diagram of section 2.4) which terminates the program. This is not something the user should expect and there should be some validation of the user choice.

3.1 Add Course

This option will take the user through some required entries for a course to registered to the course list. In figure 3 an example is shown where the course *Multivariate Analysis* is added. The instance of the CourseList class has been updated and now the entire contents of it is displayed, which is the one course.

After the user has entered all required entries, and they have been validated[1], the menu gets printed again. Note that if the user where to add a course with a title that already exists in the instance of the CourseList class, it wouldn't get acceptet to the list of courses – all cours titles has to be unique. On a final note of adding courses it should be mentioned that the user can currently have invalid combinations of grade and credits (such as grade being *G* and course credits being 0).

3.2 Remove Course

This option is straight forward. It will ask the user for a course name, and unless the course list is empty, keep doing so until a name which can be located in the course list is found (ignoring case). Once a valid title has been given, the chosen course will be removed from the CourseList instance. If the CourseList instance is empty, no search will be performed by the userChoice method.

[1]validation includes: lettergrade has to be U, G or VG (set to U if anything else is encountered). Credits has to be a value between 0 and 30 (set to 0 if anything else) and hours has to be a value between 0 and 500

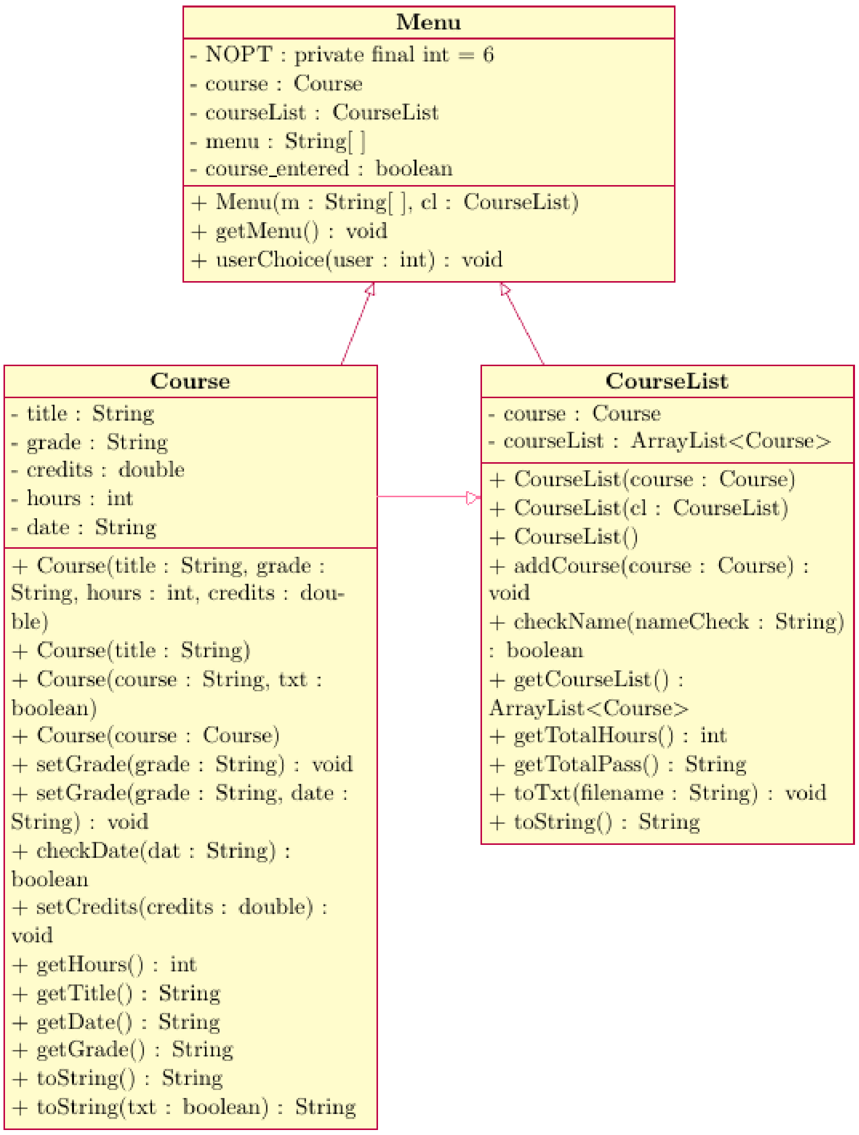
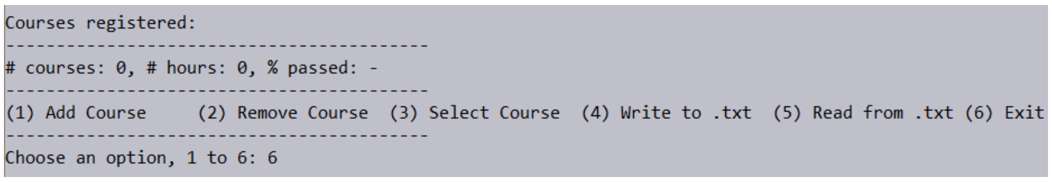
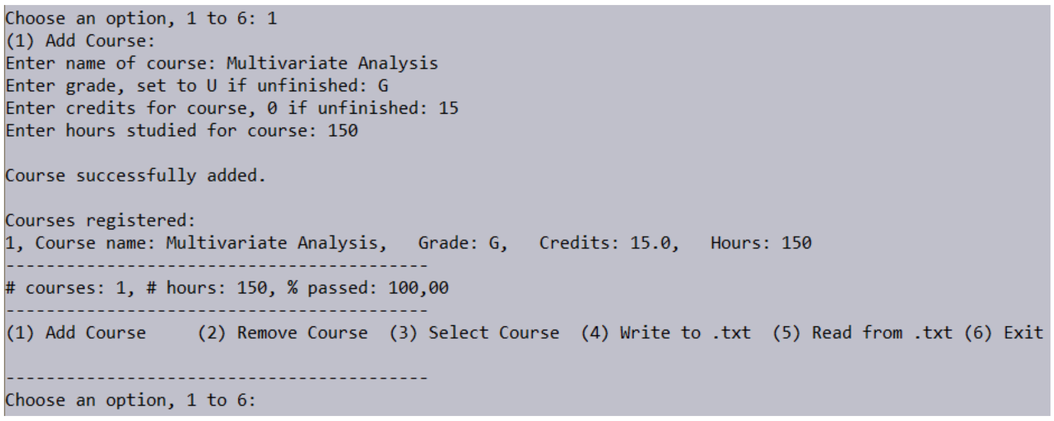

Menu
- NOPT : private final int = 6
- course : Course
- courseList : CourseList
- menu : String[ ]
- course_entered : boolean
+ Menu(m : String[ ], cl : CourseList)
+ getMenu() : void
+ userChoice(user : int) : void
Course
- title : String
- grade : String
- credits : double
- hours : int
- date : String
+ Course(title : String, grade : String, hours : int, credits : double)
+ Course(title : String)
+ Course(course : String, txt : boolean)
+ Course(course : Course)
+ setGrade(grade : String) : void
+ setGrade(grade : String, date : String) : void
+ checkDate(dat : String) : boolean
+ setCredits(credits : double) : void
+ getHours() : int
+ getTitle() : String
+ getDate() : String
+ getGrade() : String
+ toString() : String
+ toString(txt : boolean) : String
CourseList
- course : Course
- courseList : ArrayList<Course>
+ CourseList(course : Course)
+ CourseList(cl : CourseList)
+ CourseList()
+ addCourse(course : Course) : void
+ checkName(nameCheck : String) : boolean
+ getCourseList() : ArrayList<Course>
+ getTotalHours() : int
+ getTotalPass() : String
+ toTxt(filename : String) : void
+ toString() : String

Figure 1: UML diagram



```
Courses registered:
------------------------------------------
# courses: 0, # hours: 0, % passed: -
------------------------------------------
(1) Add Course     (2) Remove Course  (3) Select Course  (4) Write to .txt  (5) Read from .txt (6) Exit
------------------------------------------
Choose an option, 1 to 6: 6
```

Figure 2: Main menu of course manager



```
Choose an option, 1 to 6: 1
(1) Add Course:
Enter name of course: Multivariate Analysis
Enter grade, set to U if unfinished: G
Enter credits for course, 0 if unfinished: 15
Enter hours studied for course: 150

Course successfully added.

Courses registered:
1, Course name: Multivariate Analysis,   Grade: G,   Credits: 15.0,   Hours: 150
------------------------------------------
# courses: 1, # hours: 150, % passed: 100,00
------------------------------------------
(1) Add Course     (2) Remove Course  (3) Select Course  (4) Write to .txt  (5) Read from .txt (6) Exit

------------------------------------------
Choose an option, 1 to 6:
```

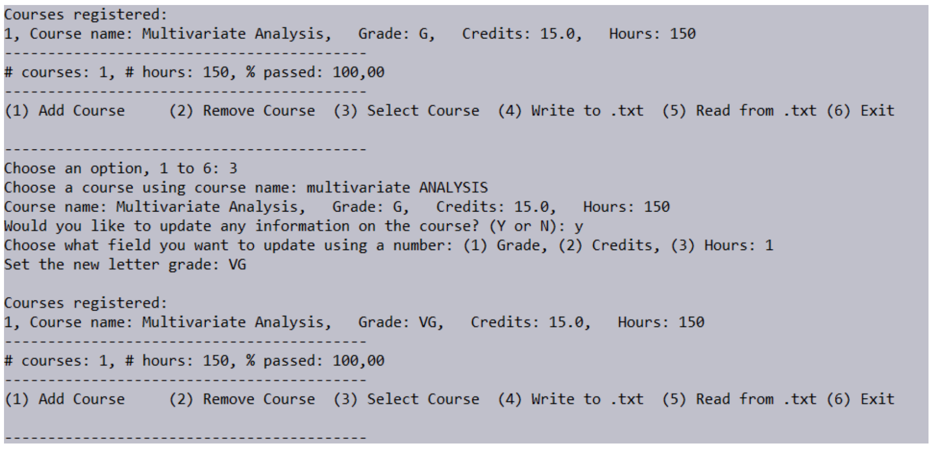
Figure 3: Add course

3.3 Select Course

This option is the tool through which the user can update information about the course (not the title name however, that is fixed). If this option is chosen, the user will be asked to enter a course name and prompted to re-enter a course name if none was found in the course list with a matching title. Once a course has been chosen it's information will be displayed along with the option of making changes to it. If the user chooses to do so there will be several options avaible. The user can change the grade, credits and hours. Note that there is input validation here, as with the option of adding a course.
In figure 4 an example of the option is displayed. A course is chosen and its grade is updated from G to VG.

Note that if changing a grade from U to G or VG, the user will be prompted to enter a date of passing the course. The date will be validated before accepted. [2] After the grade has been changed and a date set, the user can't set the grade back to U.

[2]Format is yy-mm-dd. Year has to be between 00 and 17, month has to be between 01 and 12 and day has to be between 1 and 31

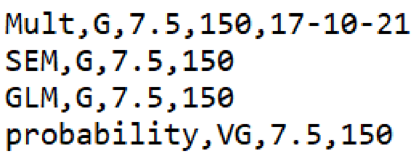


```
Courses registered:
1, Course name: Multivariate Analysis,   Grade: G,   Credits: 15.0,   Hours: 150
------------------------------------------
# courses: 1, # hours: 150, % passed: 100,00
------------------------------------------
(1) Add Course     (2) Remove Course  (3) Select Course  (4) Write to .txt  (5) Read from .txt (6) Exit

------------------------------------------
Choose an option, 1 to 6: 3
Choose a course using course name: multivariate ANALYSIS
Course name: Multivariate Analysis,   Grade: G,   Credits: 15.0,   Hours: 150
Would you like to update any information on the course? (Y or N): y
Choose what field you want to update using a number: (1) Grade, (2) Credits, (3) Hours: 1
Set the new letter grade: VG

Courses registered:
1, Course name: Multivariate Analysis,   Grade: VG,   Credits: 15.0,   Hours: 150
------------------------------------------
# courses: 1, # hours: 150, % passed: 100,00
------------------------------------------
(1) Add Course     (2) Remove Course  (3) Select Course  (4) Write to .txt  (5) Read from .txt (6) Exit

------------------------------------------
```

Figure 4: Select Course

3.4 Write to .txt

This option allows the user to write all the information in the instance of the CourseList to a .txt file. The information will be read using the overloaded to string method of the Course class.



```
Mult,G,7.5,150,17-10-21
SEM,G,7.5,150
GLM,G,7.5,150
probability,VG,7.5,150
```

Figure 5: Text file format

In figure 5 the way the function writes the course information is displayed for four courses.

3.5 Read from .txt

This option allows the user to read the information from a .txt file (with the exact format written by the *Write to .txt* option). Currently there is some problem with reading from a text file that the courses will not be displayed in the menu until a new course has been added. The constructor in the Course class which takes the row from the .txt file and creates an instance of the Course class is the one which takes a string and a boolean.

As a sidenote, using a comma in their course title is a simple way of causing the program to break down. Looking at the code for the constructor in the Course class which takes a string and a boolean, it should be evident that if there's some unexpected commas in the line the construtor wouldn't work. One way of dealing with this is to let the user enter commas in the program but remove them once read into the .txt file and inform the user.

The user can go around the input validation by changing the values directly in the text file generated by the program. In my program I make the assumption that the text file hasn't been altered by the user. So if the user where to open the text file written from the program and change the hours of a course to 5000, this would be accepted by the program.

4 Final Remarks

I do feel that this program could have used a little more planning from the get go. The program wasn't really planned with all the tasks in mind, resulting in quite the patchwork. With that said, it does perform all the specified tasks in the mini project.