RMIT UNIVERSITY

STUDENT ENROLMENT SYSTEM

Technical report

Author: Nguyen Luu Quoc Bao

Supervisor/ Lecturer: Mr. Vu Thanh Minh

COSC2440 | S3877698

Table of Contents

OVERVIEW

Student Enrolment System is widely implemented at almost universities all over the world [1]. In this project, a console demonstration of this system is developed with Java. This console app has functions to read CSV files and convert them to valid data, which can be used by the system to perform tasks including creating, reading, updating, and deleting (CRUD) enrolments. Besides, the app can export CSV reports on the users' queries in case they request.

FLOW DESCRIPTION

There are three layers on this app including system, service, and menu. The purpose of separating is to minimize the coupling between classes, which can maximize the efficiency of the program. Besides, this architecture facilitates the extendibility and maintainability of the app.

In brief, when users start the program, the Menus is called for getting commands. Those menus then called the services relevant request so that those services can interact with data in the system layer for responding the result.

In detail, when users start the program, the system will prompt a message to ask if they want to use their file. If yes, the system will ask them to enter the file name. Otherwise, the system will automatically use the default file in the system. A notable point is that all CSV file is stored in the data directory. Next, the

system will populate the relevant data and allocate them into three lists including student list, course list, and enrolment list. After that, the system will display the main menu under a table (view Appendix 2). There are four options: manage students, manage courses, manage enrolments, and quits. The user will choose an option by entering the relevant toggle key number. The menu will run infinitely until the user chooses the quit option. In case the wrong toggle key is entered, the menu will ask them to correctly input the toggle again. For student managing, there are three options: view all students, view all students in a course, and back to the main menu. Users can choose an option by toggling the key and providing relevant information such as IDs and getting output performed by the services in the menus. In the case of student and course managing, the user can choose to save the result to CSV files, which are stored on the reports directory. Similarly, there are four options in course management comprising view all courses, view all courses in a semester, view all courses of a student in a semester, and back to the main menu. In terms of enrolment management, users can choose to directly add an enrolment by providing student id, course id, and semester, or they can choose the update option to add or drop course of a student in a semester.

System

This layer acts as a backend of the application. By using CsvService class to access CSV files and populate them to appropriate data for the app.

On this layer, there is an interface called StudentEnrolmentManager which covers all the necessary functions to manage students, courses, and enrolments. This interface is implemented by the StudentEnrolmentSystem class, which override all methods (CRUD) in tandem with implementing some additional method such as getFileName.

The StudentEnrolmentSystem is supported with CsvService class, which comprises all methods to interact with CSV files and convert them into Model (Student, Course, or Enrolment) data. Those converted data are stored on three lists for later retrieves.

Another noticeable point is that when an instance of StudentEnrolmentSystem is initialized, users can choose to use their data by providing the file name. Otherwise, the system will automatically populate data with the default file name, which is already provided by the developer.

Service

This layer acts as the APIs of the application. By taking StudentEnrolmentManager as an attribute, those services (including StudentService, CourseService, and EnrolmentService) can use the functions of the system to interact with the data. A StudentEnrolmentManager must be passed in the constructor whenever an instance of service is initialized.

On this layer, there is three main service class consisting of StudentService, CourseService, and EnrolmentService. In terms of StudentService, this class has two methods including getting all students and getting a student by ID. The implementation of those functions basically calls the relevant functions on the StudentEnrolmentSystem and responds with filtered results. About CourseService class, there are three methods including get all courses, get course by semester, and getting course of a student by semester. The logic of those methods is to call relevant methods on the system. EnrolmentService also follows those executions above.

Another noticeable point is that all the validation about constraints will be executed on this layer. For example, before returning all courses of a student, the service will call suitable functions to validate the existence of that student by retrieving ID.

Menu

This layer acts as the frontend of the application. By taking relevant services classes as an attribute, those menus can call service functions based on the input of the user. There is a parent abstract class called Menu. Other's menus are extended from this class. Each menu has a different list of commands, which can be displayed in the show menu method.

On this layer, there are five menus including StudentMenu, CourseMenu, EnrolmentMenu, UpdateMenu, and MainMenu. UpdateMenu is a composition with the EnrolmentMenu as it cannot exist without the EnrolmentMenu. MainMenu has StudentMenu, CourseMenu, EnrolmentMenu as attributes. Once getting users' commands, the MainMenu will decide to run a suitable Menu to perform tasks called from services.

Those menus will check the display format of each response before printing it out to the user. For example, it will print "Student does not exist" in case the ID is not valid instead of Null, which enhances the usability of the app. Those displays are supported by the utility class called Table, which takes lists and displays them under a table format. (View Appendix 2).

Utilities

CsvValidator

This class is used to check each row of the CSV file is on true format and all fields are not null. If a row is invalid, it will throw an exception. This class is used when converting CSV to Model data.

CsvWriter

This class is used to write data from users' query to a CSV report. This class can convert the data in a Student or Course list to CSV format and write them to a new file in the report's directory.

DataConverter

This class is used to convert Date to String and String to Date using SimpleDateFormat, which is used while converting CSV file to model data and displaying Student data.

Table

This class is used to display responses (lists) under the format of table to enhance the user-experiences. This class can take lists of models and display them to the screen as tables.

METHODS

Method name	Params	Return	Functionality	Pseudo code
getAllEnrolments	none	List<Enrolment>	Get list of enrolments	**begin** **return** list **end**

getOneEnrolment	- sId: student id
	- cId: course id
	- semester: semester
		Enrolment if found, null if not found	Get one enrolment by finding in the list with sId, cId, and semester	**begin**
				  for each e **in** list
				  if all params matched
				  do return e
				  return null
				end

addEnrolment	- sId: student id
	- cId: course id
	- semester: semester
		Boolean: true if added successfully, false if added fail.	Take information about enrolment, convert it to Enrolment and add to the list	**begin**
				  if exist **in** list **return** false
				  else add new object of Enrolment to the list
				  return true
				end

removeEnrolment	- sId: student id
	- cId: course id
	- semester: semester
		Boolean: true if removed successfully, false if removed fail.	Take information about enrolment, find that enrolment on the list and remove if exist	**begin**
				  if exist **in** list **return** false
				  else delete Enrolment from the list
				  return true
				end

getAllCourses	none	List<Course>	Get list of courses	**begin** **return** list **end**

getCourseById	cId: course id	Course if found, null if not found	Take an id and find course with that id in the list	**begin**
				  for each c **in** list
				  if id matched
				  return c
				  return null
				end

getAllStudent	none	List<Student>	Get list of students	**begin** **return** list **end**

getStudentById	sId: student id	Student if found, null if not found	Take an id and find student with that id in the list	**begin**
				  for each s **in** list
				  if id matched
				  return s
				  return null
				end

DIAGRAMS

Use case diagram

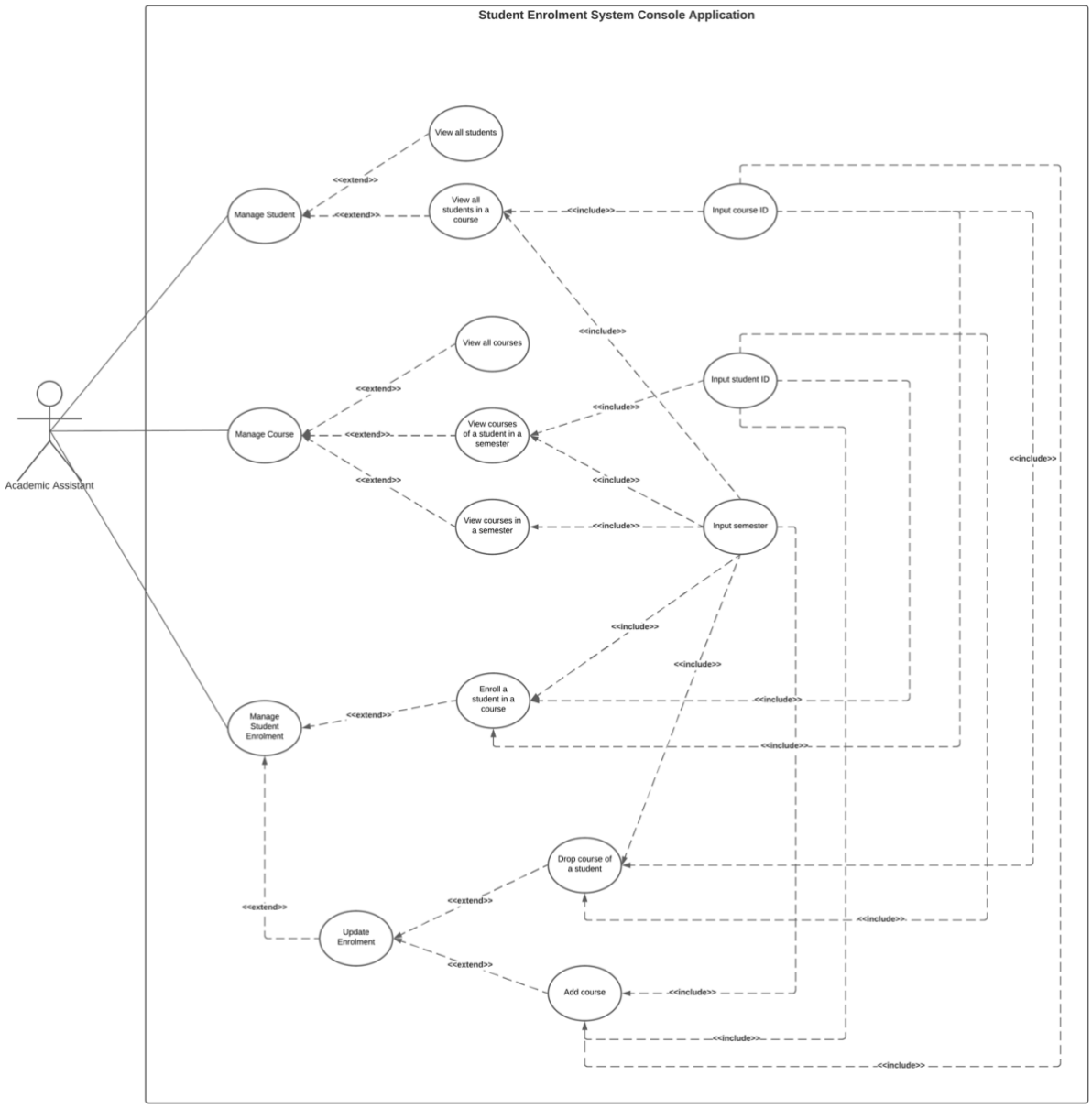

Figure 1. Use case diagram (PDF Attached on ZIP file - uml directory)

Class diagram

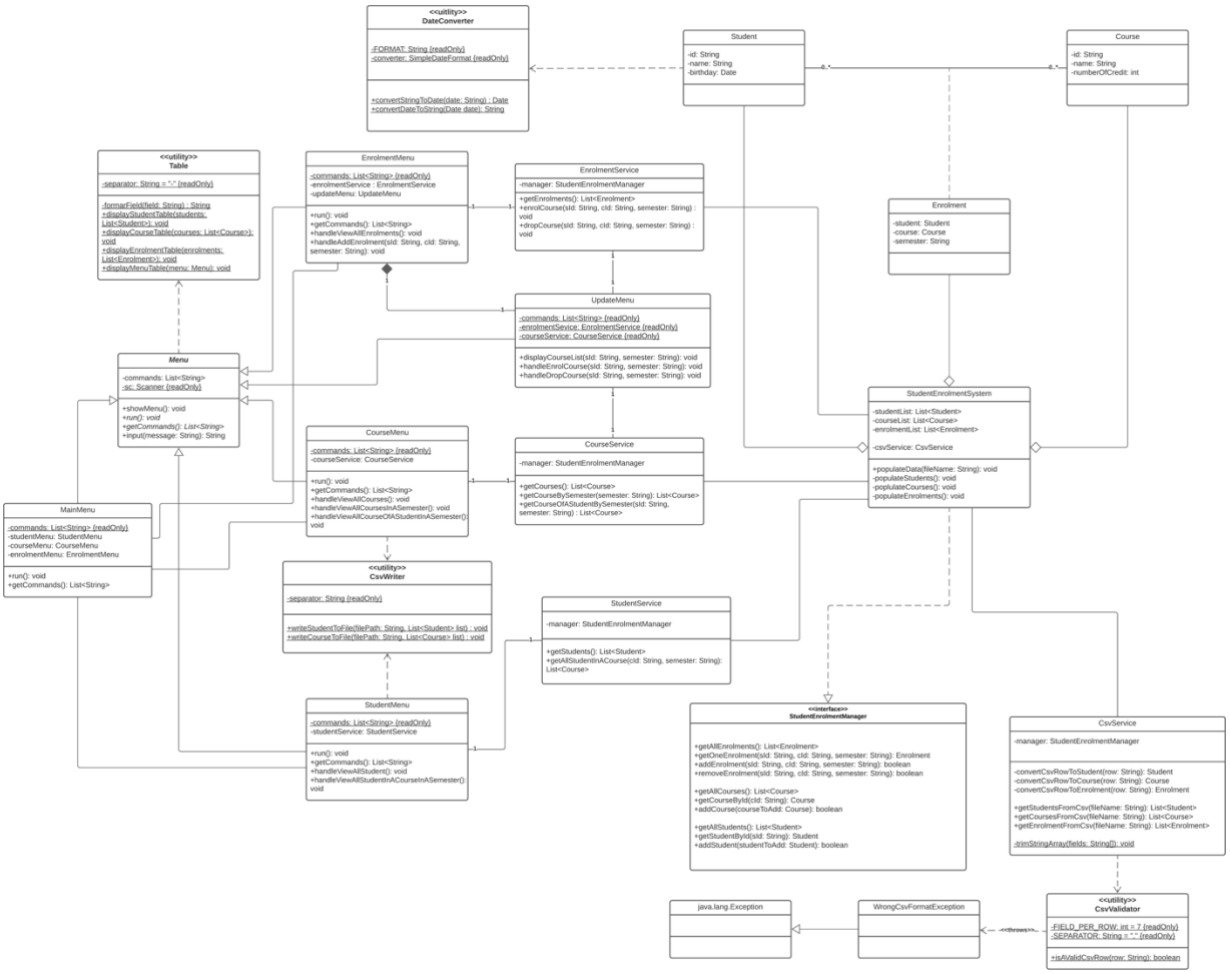

Figure 2. Class diagram (PDF Attached on ZIP file – uml directory)

APPENDIX

Appendix 1. GitHub: https://github.com/s3877698/Student-Enrolment-System

Appendix 2. Example of menu under table format

```
********** Managing Enrolment **********
-----------------------------------------------------------------
|  Toggle  |                     Command                        |
-----------------------------------------------------------------
|    1     |                View all enrolments                 |
|    2     |                  Add an enrolment                  |
|    3     |                  Update enrolment                  |
|    4     |                        Back                        |
-----------------------------------------------------------------
```

REFERENCES

[1] H. K. Massoud and R. Ayoubi, "Do flexible admission systems affect student enrollment? Evidence from UK universities", *Journal of Marketing for Higher Education*, vol. 29, no. 1, pp. 84-101, 2018. Available: 10.1080/08841241.2018.1562507.