Assignment 1 Technical Report

Course: COSC2440 - Software Architecture: Design & Implementation
Semester: 2021A
Name: Truong Duc Khai
Student ID: s3818074

I. Introduction

This is a Java console application that manages student enrollment. Data about students, courses and enrolments are loaded from CSV files. This application consists of features such as viewing students and courses, CRUD on enrolment, saving data to CSV files, etc. These features will help increase the efficiency of managing student enrolments.

II. Software architecture

The architecture of this application consists of 3 layers: repository - service - menu, which helps to reduce class coupling and separate the business logic and presentation logic.

1. Repository

This is the layer that will access the CSV files when the program starts and save each data into their respective list so that the rest of the application can access them later.

The main interface for this layer is StudentEnrolmentManager (SEM), which provides methods to get the students/courses and CRUD on enrolments.

The main implementation for this interface is InMemoryStudentEnrolmentManager. As the name suggests, the data are loaded from the CSV files and saved inside a data structure called List, which only persists in memory, as the program runs. The methods of its implemented interface are overridden to access and mutate the data from those lists.

The instance of SEM will be injected into the other class for providing more extensions and useful methods on those base functionalities.

2. Service

This is the layer that interacts with the repository. Using the pattern called Dependency Injection, I will pass in the instance of SEM into the constructor. There are 3 main service classes in this application: StudentService, CourseService and EnrolmentService, which will take care of tasks such as filtering the data from the SEM and providing methods to display the data on the console.

By using these service classes, I separate the methods based on the model that they interact with, which avoid creating a clutter inside the SEM class with a bunch of methods. These methods will be further used by the Menu layer.

Additionally, I also create two more service classes for more utilities: CsvService and InputService. CsvService will handle the task of reading the data from the CSV files and converting them into objects which will be added to the SEM entity lists. InputService will take care of asking and validating user inputs, which is a fundamental part of the application.

3. Menu

As the name suggests, this layer will be the presentation of this console application. There are many menus that serve different purposes depending on the model that the user wants to interact with.

When the application runs, the MainMenu will run first, leaving the user with the options to navigate into sub-menus based on model or quit the application.

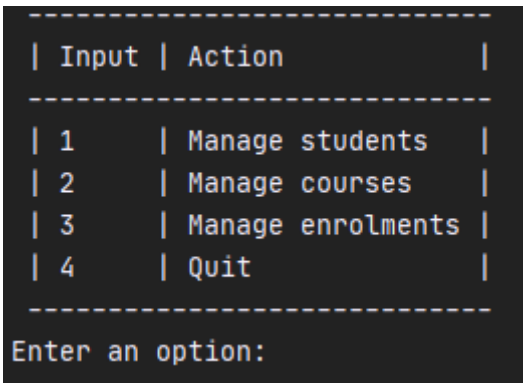


Figure 1. MainMenu console view

Within each submenus, the user can navigate back to the main menu by selecting the "Back" option, or select an option to perform a specific task. There are 3 sub-menus: StudentMenu, CourseMenu and EnrolmentMenu.

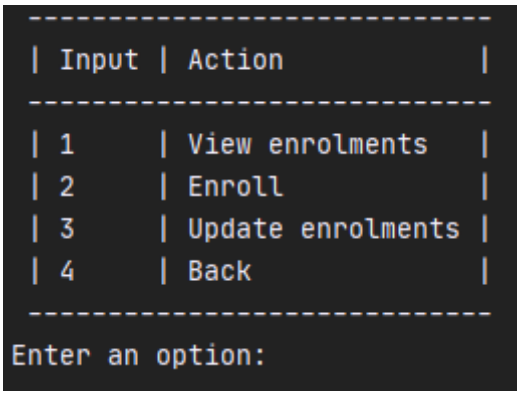


Figure 2. EnrolmentMenu console view

To make updating enrolment of a student in a semester easier, I create a menu called EnrolmentUpdateMenu, which will be opened after the user type in the student ID and the semester that they want to update the enrolment.

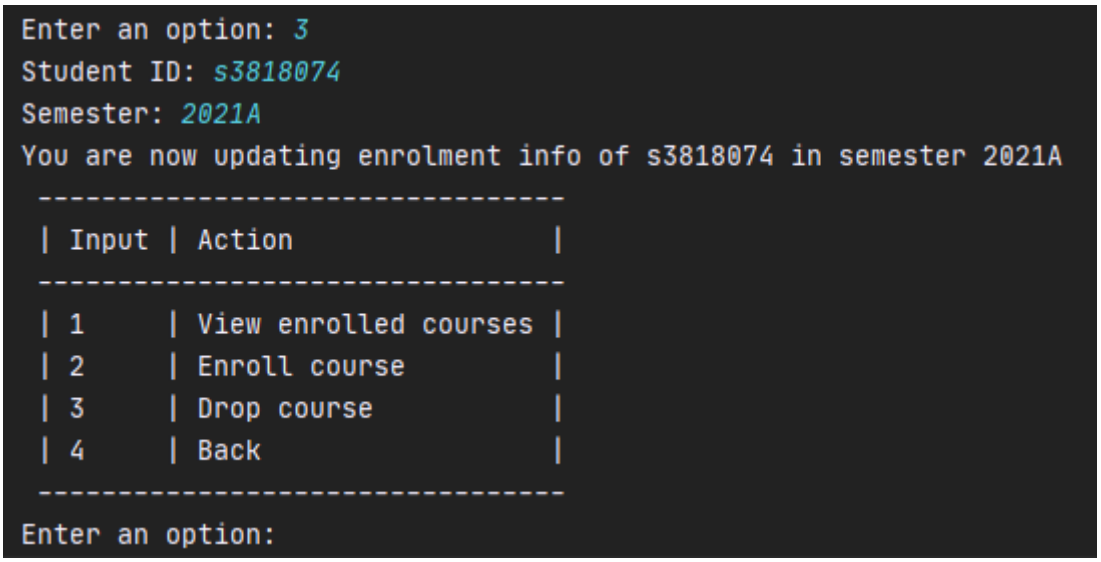


Figure 3. EnrolmentUpdateMenu console view

4. Helper

There are many helper classes that support the above layers in order to make the codebase of this application clean and reusable. Below are some notable ones:

a) Table

This class abstracts the formation of table view, which is used in displaying menu options and list data. The client will only need to care about data insertion, as this class has already hidden all the implementation details of formatting the table and calculating column lengths to fit the longest data.

b) InputField

This class abstracts the behaviour of a typical input field, such as validating the input based on client needs using lambda function and asking the input until it is valid.

In this application, even though empty input is invalid, I make it valid but function as a skip option. The menu will skip the task if the input is empty, so that the user will not be stuck inside the application in some cases such as not remembering the id of a student or a course.

c) Option

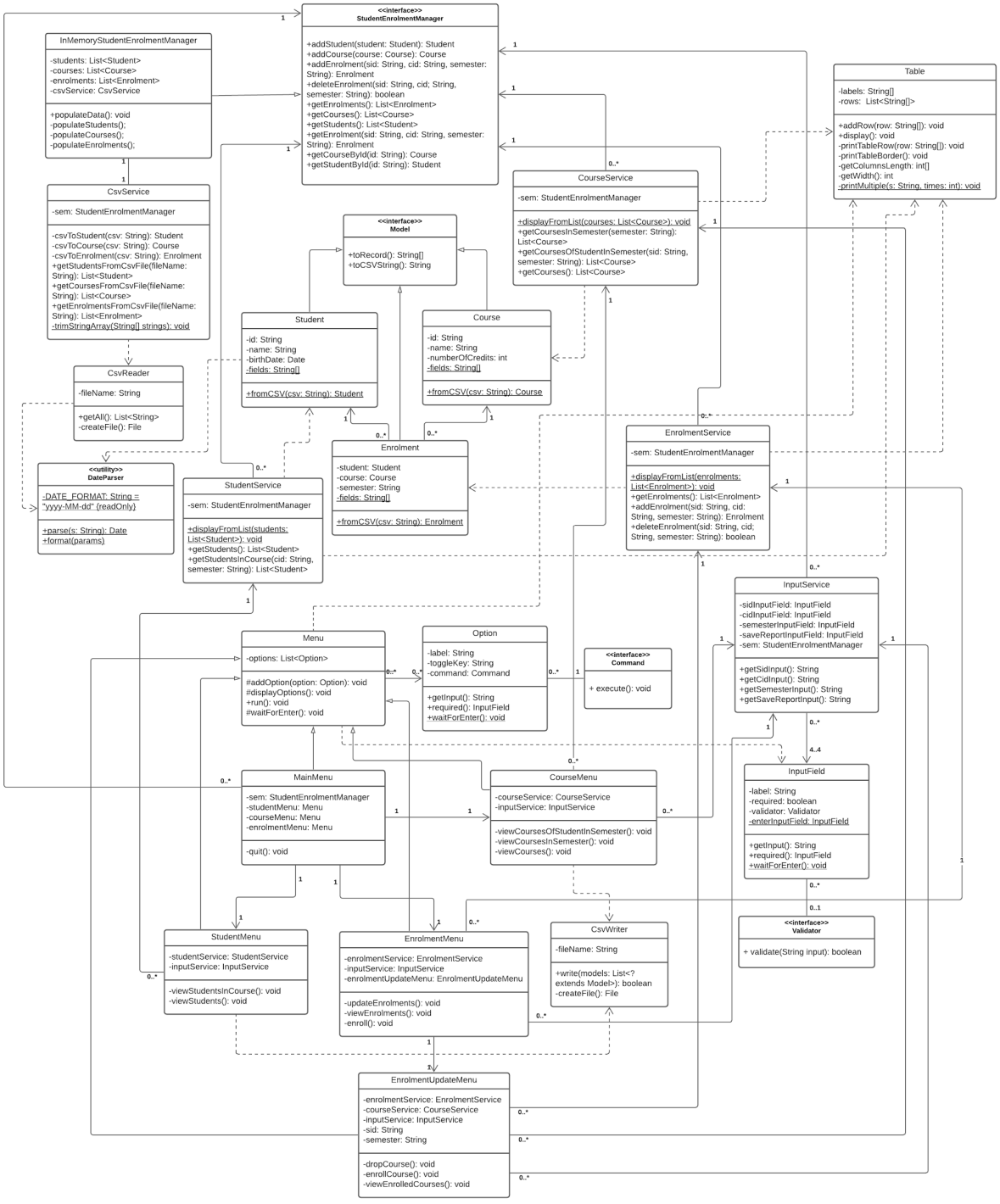
This class represents the option inside a menu, which has a label, a toggle key, which is the input string to activate the option, and a callback function when the input is activated. Using this class, I do not have to write the print statements manually on each menu and do switch-case to handle different tasks based on input. Each menu will have a list of options that will be displayed in the console and perform the correct task based on user input.

```
addOption(new Option( label: "View enrolments",  toggleKey: "1", () -> {
    viewEnrolments();
    waitForEnter();
    run();
}));
addOption(new Option( label: "Enroll",  toggleKey: "2", () -> {
    enroll();
    waitForEnter();
    run();
}));
addOption(new Option( label: "Update enrolments",  toggleKey: "3", () -> {
    updateEnrolments();
    run();
}));
addOption(new Option( label: "Back",  toggleKey: "4", () -> {

}));
```

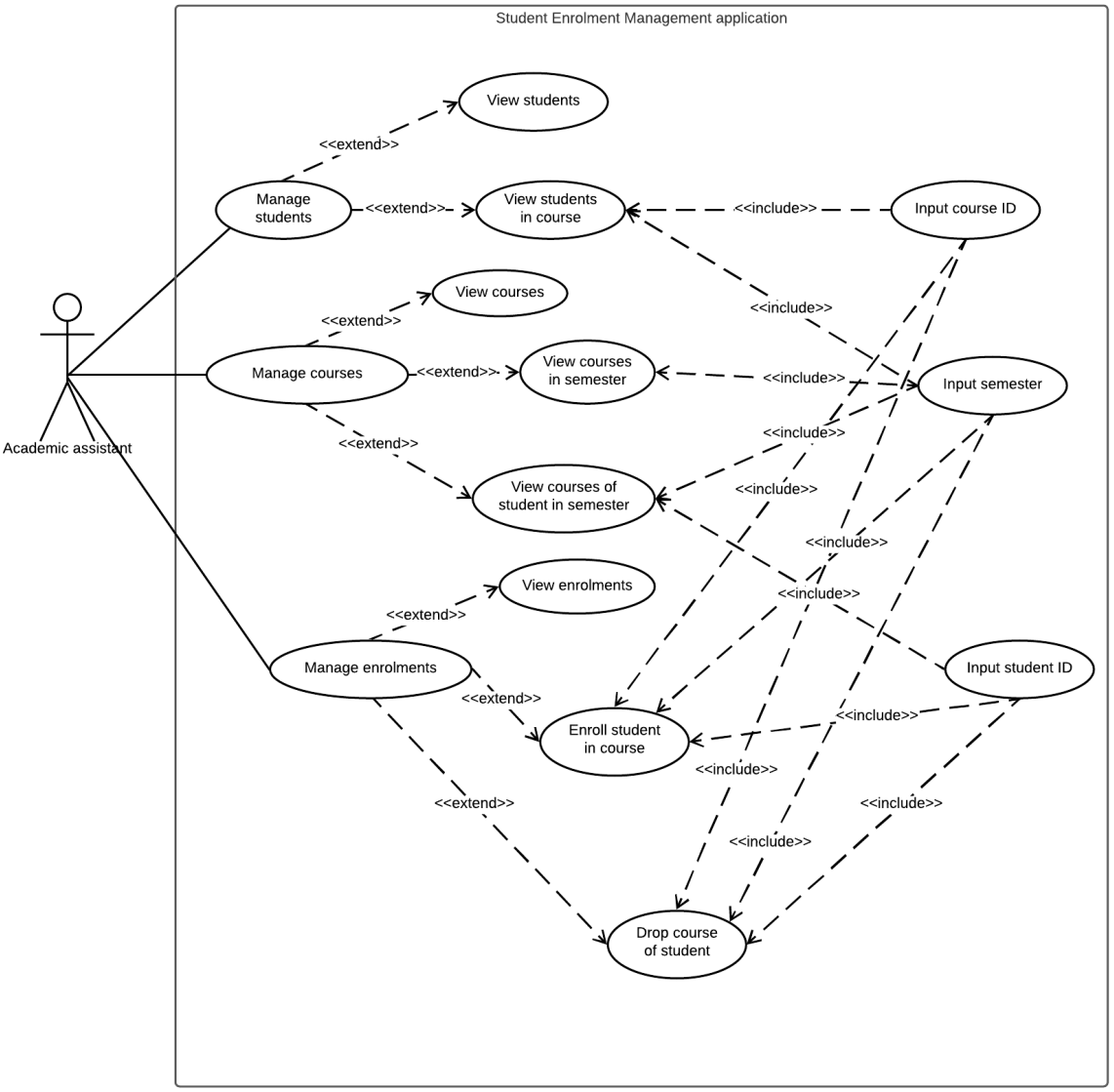
Figure 4. EnrolmentMenu implementation using Option list

III. Class diagram



(The pdf file of this class diagram can be found at the root of the zip file in case this one is not clear enough)

IV. Use case diagram



(The pdf file of this use case diagram can be found at the root of the zip file in case this one is not clear enough)

V. GitHub

https://github.com/khaitruong922/cosc2440-a1

VI. How to run

Go to the "src" folder and run the Main class.

VII. Tests

Tests are located in "src/test", which cover many cases of all methods in the SEM interface. The tests are written using JUnit Jupiter 5.4.2.