Design Document for Timetable Manager

Alp Koçak 20220602056
Efe Serin 20210602055
Doğa Orhan 20210602043
Sude Teslime Daka 20220602207

Contents

1 Introduction

Timetable Manager is a standalone desktop application that can create, edit, and display timetables. The application is targeted for Windows desktops and will be developed using Java and JavaFX, with Scene Builder aiding in the GUI design. The application will be in English, as per the non-functional requirements.

2 Software Architecture

The software architecture for Timetable Manager is built around a simple and efficient design, focusing on two main components: backend logic and a user-friendly frontend. The backend will consist of core classes like **Course**, **Timetable**, and **Student** to handle data management and operations such as timetable creation, class assignments, and student enrollments. The frontend will provide a graphical interface that enables users to interact with these features easily. A **DatabaseHandler** class will manage data storage and retrieval using SQLite. This straightforward architecture ensures usability, maintainability, and reliability.

2.1 Structural Design

The system's architecture revolves around three core classes: **Student**, **Course** and **TimetableManager**. Each class has distinct responsibilities, and their interactions enable the system to fulfill all specified functional requirements. It interacts with the Student class, which holds individual student data and their enrolled courses, and the Course class, which manages course details, including time slots and enrolled students. This structure ensures efficient timetable generation and conflict resolution, with each class playing a specific role in managing data and operations within the system.

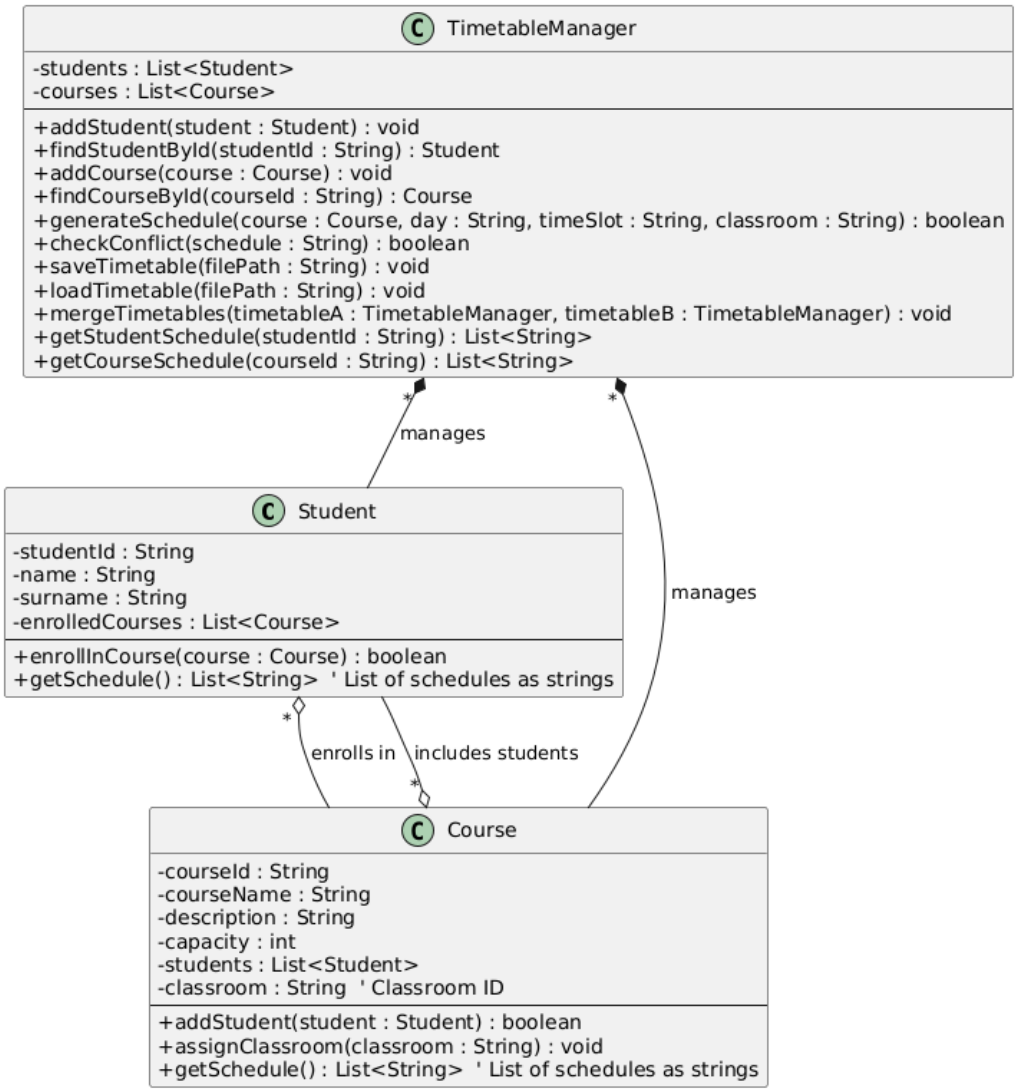

Figure 1: Class Diagram for Timetable Manager

As shown in Figure 1, the class diagram illustrates the relationships and methods of these core components.

2.1.1 Class Overview

TimetableManager Class: The `TimetableManager` class acts as the central controller of the application. It **manages** the collection of students and courses and facilitates critical functionalities like schedule generation, conflict checking, and **data persistence**.

Key Methods:

- `addCourse(course: Course): void` Adds a new course to the system, enabling administrators to define its details such as name and description.
- `generateSchedule(course: Course, day: String, timeSlot: String, classroom: String): boolean` Assigns a classroom and time slot to a course while checking for conflicts.
- `checkConflict(schedule: String): boolean` Ensures no scheduling conflicts exist before enrolling a student or assigning a classroom.

Functional Requirements:

- **Requirement 1:** The user must be able to add a course.
- **Requirement 2:** The user must be able to assign classrooms to courses compatible with the capacity.
- **Requirement 3:** The user must be able to enroll students in courses without schedule conflicts.

This functionality ensures that the system meets Functional Requirements 1, 2, and 3.

Student Class: The Student class represents an individual student enrolled in the system. Each student has a unique identifier, name, and surname, and a list of courses they are enrolled in.

Key Methods:

- `enrollInCourse(course: Course): boolean` Checks for scheduling conflicts and, if none exist, enrolls the student in the specified course.
- `getSchedule(): List<String>` Retrieves the student's weekly schedule, listing all enrolled courses.

This class directly supports Functional Requirement 3 by ensuring that students can only enroll in non-conflicting courses.

Course Class: The `Course` class encapsulates the details of a course, including its *ID*, *name*, *description*, and *associated classroom*. It also **tracks** the list of students enrolled in the course.

Key Methods:

- `addStudent(student: Student): boolean` Adds a student to the course if capacity allows and no scheduling conflicts exist.
- `assignClassroom(classroom: String): void` Allocates a classroom to the course.

This class meets Functional Requirement 2 by ensuring classrooms are assigned based on the course capacity and prevents over-allocation.

2.1.2 Relationships

- A many-to-many relationship exists between `Student` and `Course`, as each student can enroll in multiple courses, and each course can include multiple students.
- A one-to-one relationship exists between `Course` and `Classroom`, where each course is assigned a single classroom, represented by the `classroom` attribute.
- A many-to-many relationship exists between `Timetable` and `Course`, as each timetable can include multiple courses, and each course can be scheduled in multiple timetables.

Activity Diagram: The following Activity Diagram outlines the functional flow of the system, integrating the operations of `TimetableManager`, `Student`, and `Course` classes:

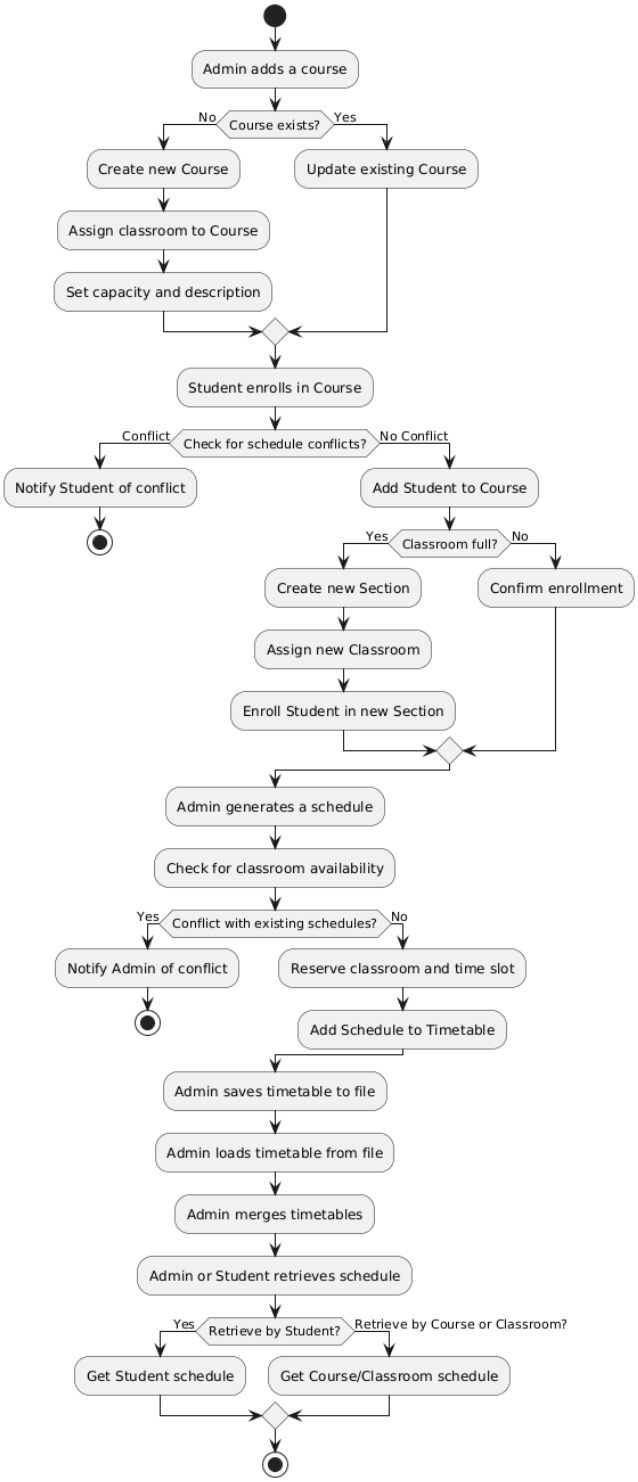

Figure 2: Activity diagram to visually explain how program will work.

2.2 Behavioral Design

While the structural design of the classes allows for storing data in memory, the following methods are required to perform operations such as **searching**, **saving**, **loading**, and **merging** within the software.

2.2.1 Searching Timetable for Students or Classes

The system will allow users to search for and view a specific student's timetable. The Timetable Manager class will store all timetable data in a "timetable" variable, which holds a collection of **Course** objects. Each Course object contains information about the *course name*, *classroom*, and *enrolled students*. To search for a specific student's schedule, the system will iterate through the courses to identify the student's enrollment. The Student class will have a unique identifier, such as a *studentID*, to ensure that the correct student's data is retrieved.
This functionality will meet Functional Requirement 4.

> **Functional Requirement 4:** The user must be able to search for a student's course schedule by week.

Functional Requirement 5 states that the users shall be able to search for a classroom's schedule. The Timetable Manager will allow users to search for the weekly schedule of a specific classroom. The system will go through the timetable data and retrieve all courses assigned to the given classroom, providing a clear and organized view of the classroom's usage.
This functionality will meet Functional Requirement 5.

> **Functional Requirement 5:** The user must be able to check classroom schedules for any given week.

2.2.2 Swapping Classroom

The class swap function allows students to swap classes. When a user initiates a class swap, the system verifies that the target class has sufficient capacity for the course and is available within the required time slot. If these conditions are met, the swap is performed and the timetable is updated accordingly. If the target class cannot meet these conditions, the system notifies the user and ensures that the timetable remains consistent and conflict-free.
This functionality will meet Functional Requirement 6.

Functional Requirement 6: The user must be able to perform a classroom swap, checking availability for that classroom.

2.2.3 Adding New Sections to Courses

When there are more students than the capacity of the designated class, the system allows additional course sections to be created. When a new request is made, the system places the user in a different class that meets the capacity requirements. This provides scalability for multi-person classes by allowing each student to enroll in the course without exceeding the class limit.
This functionality will meet Functional Requirement 7.

Functional Requirement 7: The user must be able to add new sections to a course if the classroom capacity is full.

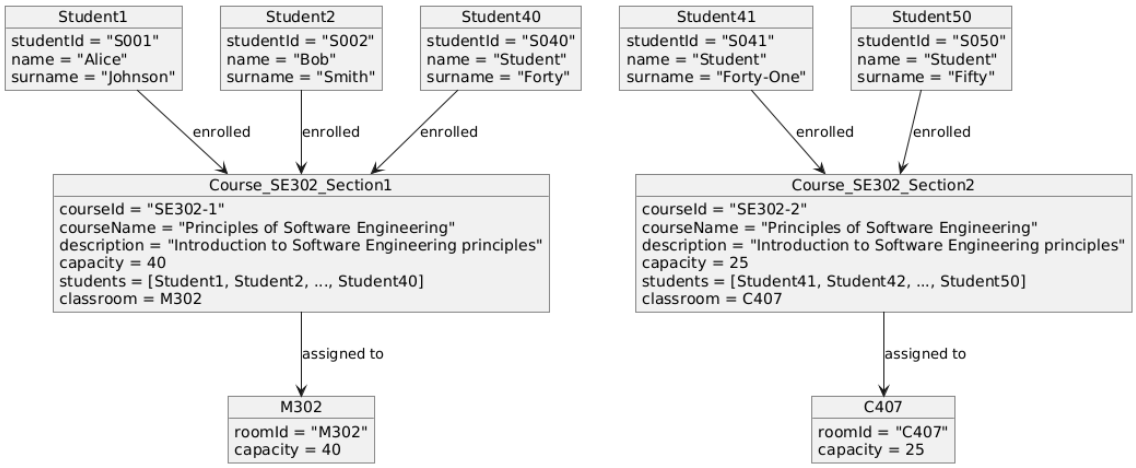

Figure 3: Object diagram for adding a new section when exceeding classroom capacity.

2.2.4 Importing Data from File

To simplify data entry, the app includes functionality to import class and course data from a CSV file. When a user uploads a file, the system processes the data, validates it, and populates the timetable. The user is given the option to select a file to import, and the system handles errors such as incorrect formats or missing data.
This functionality will meet Functional Requirement 9.

Functional Requirement 9: The user must be able to import classrooms and courses data in CSV file format.

2.2.5 Persistent Storage of Timetable Manager

The timetables created by the software shall be saved to and loaded from the disk. This allows users to store their timetable data and continue updating or reviewing it at a later time. The timetable data will be serialized for storage. A Timetable Manager object can be serialized and written to a file. Later, the data can be deserialized, allowing it to be loaded back into the software. To implement this, the Timetable Manager class will provide methods for saving and loading data. Specifically, the **"Save"** method will be a public function, while the **"Load"** method will be static, allowing it to be used without the need to create a Timetable Manager object first. Both methods will require a file path, specifying the file to be operated on.
This functionality will meet Functional Requirements 8, 10, and 11.

Functional Requirement 8: The user must be able to save, load, and update the timetable data.
Functional Requirement 10: The system must be able to save the current timetable to a file on the file system.
Functional Requirement 11: The system must be able to load a previously saved timetable from a file.

3 Graphical User Interface

The GUI acts as the interface where the program's code and methods interact with the user. Through this GUI, users shall be able to add, search, swap, edit, delete, and visually see their actions. The GUI will be constructed using JavaFX and its libraries. The program interface will have the following components to enable users to effectively utilize its features:

- **Programs:** There will be program screens for all constructed programs. Through these screens, the users shall be able to view all selected courses and their placement in timetables alongside other courses.

- **Search Bar:** There will be the following actions.

  - **Search Bar Results:** The search bar shall display either the entered department's courses or just the lectures themselves.
  - **Add Course:** The listed courses shall be selectable in order to be added to the program.

- **Selected Courses:**
  - **Remove:** After the courses are selected, they shall be viewable from a different window. The user will be able to remove the desired course or remove all of the courses from the window.

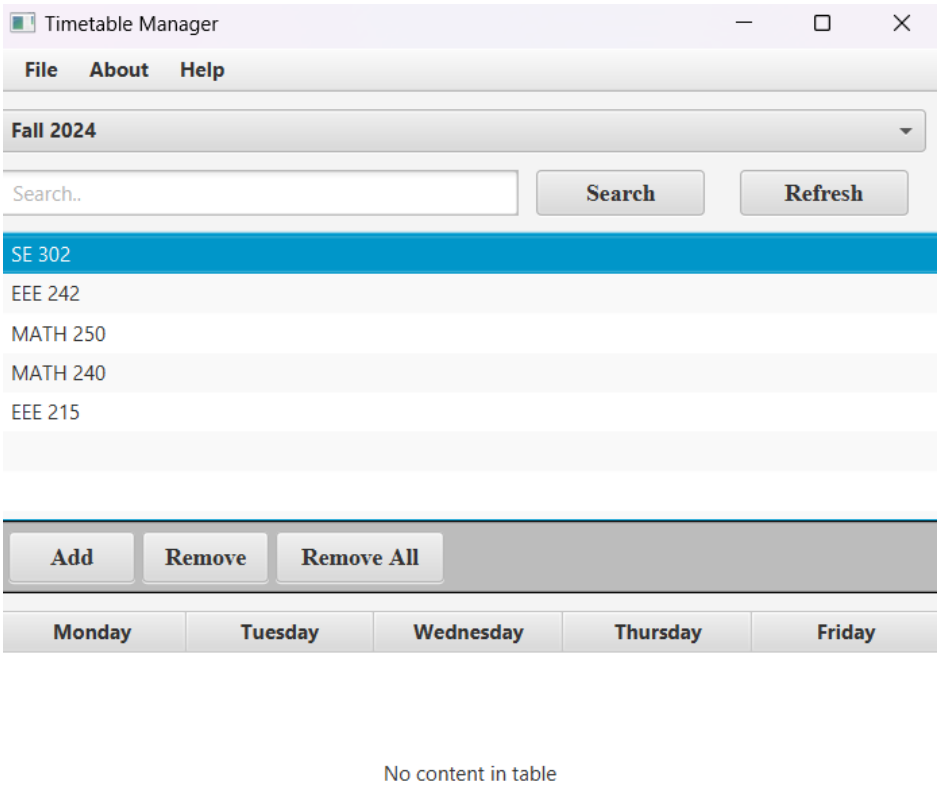

Figure 4: GUI for the Timetable Manager

- **File:** There will be the following actions.
  - **Import:** The user shall be able to import the desired database from their computer.
  - **Load:** Previously imported databases can be loaded.

  - **Save:** Users shall be able to save any changes.
  - **Export:** This option will allow users to select a specific location to save their changes.

- **Help:** The user shall be able to use the software after reading this manual. It will be accessible within the software, it will not be online.
  - **User Manual:** The help menu will provide access to a user manual that explains how to use the program effectively.
  - **About:** This section will provide general information about the Timetable Manager application.

This will meet Functional Requirement 12.

Figure 5: File and Help for the Timetable Manager

Functional Requirement 12: The software must have a manual that will be displayed with a "Help" menu item.