CSEB5223 – SOFTWARE CONSTRUCTION & METHODS

SEMESTER 2 2025/2026

LAB 2 (ASSIGNMENT PART 1)

GITHUB REPOSITORY NAME:

cherrySLMS

SECTION: 01BT

PREPARED FOR:

TS. DR. LOO YIM LING

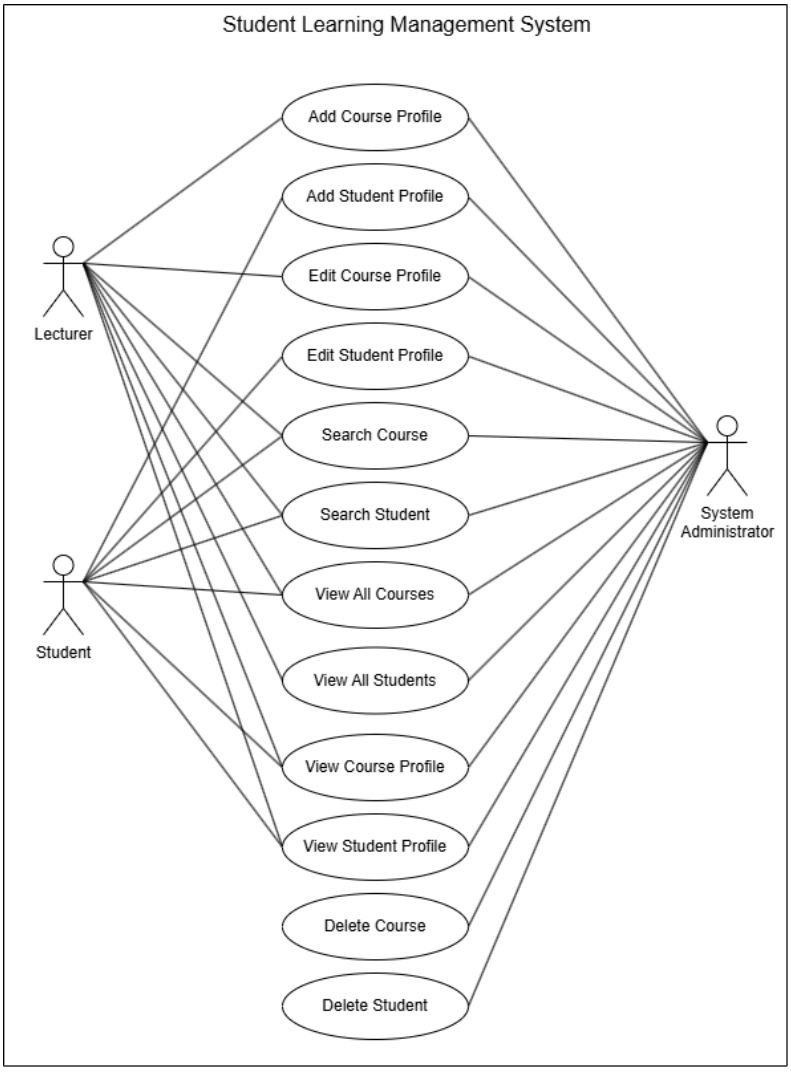
PREPARED BY:

NO.	NAME	ID
1	Nur Sahira Binti Mohd Nizam	BSW01085247
2	Aimi Syazwani Binti Mohd Ezanee	BSW01085441
3	Wan Nur Alisya Sofia Binti Wan Zainudin	BSW01085261

Table of Contents

1.0 Use Case and Class Design of Overall SLMS

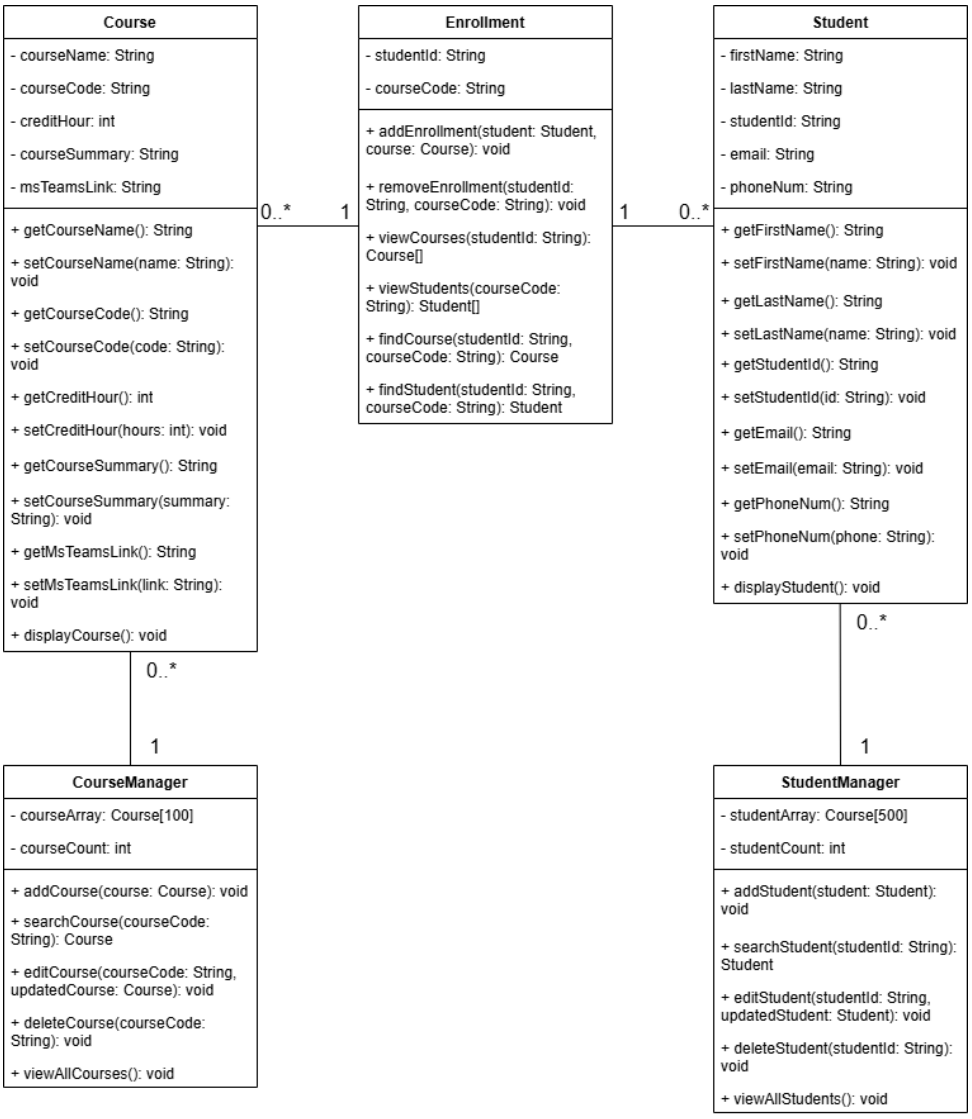
1.1 Use Case Design



- The use case diagram illustrates the main functionalities of the Student Learning Management System (SLMS) and the actors that interact with the system.
- Three actors are involved: Lecturer, Student and System Administrator, each with different access rights based on their roles.
- The Lecturer can add, edit, search, and view course and student profiles to support academic management activities.
- The Student can search and view course information and view their own student profile.

- The System Administrator has full access to the system, including adding, editing, searching, viewing, and deleting both course and student profiles.

1.2 Class Design



- The class diagram shows the main entities of the Student Learning Management System (SLMS) and how they are related.
- The Course class represents course details, while the Student class represents student information.

- The Enrollment class is used to model the relationship between students and courses.
- A student can be enrolled in multiple courses, and a course can have multiple students, which is represented through the Enrollment class.
- CourseManager and StudentManager classes are used to manage collections of students and courses, respectively.
- Each manager class maintains and manages multiple instances of its corresponding entity, representing a one-to-many relationship.

2.0 Documentation Formatting

To ensure clarity and consistency, all documentation regarding the Student Learning Management System (SLMS) must follow the following standardized format:

- Font: Times New Roman
- Font size: 11pt or 12pt
- Line spacing: 1.5
- Headings:
    - Main headings: Bold, 16 pt
    - Subheadings: Bold, 14 pt
- Page numbers included at the bottom right of pages
- Sections numbered logically (e.g., 1.0, 1.1, 1.2)
- Diagrams presented using standard UML notation
- Document format: .PDF or .DOCX

This ensures clarity and readability throughout the document while maintaining consistency in structure, terminology, and formatting. It also supports a professional academic presentation and improves overall comprehension for the reader.

3.0 Coding File Formatting

To ensure clean, consistent, and maintainable code development for the Student Learning Management System (SLMS), the following coding file formatting standards shall be applied:

- One class per file shall be implemented. The file name must exactly match the class name.
- Indentation shall use four spaces only. The use of tab characters is strictly prohibited.
- Opening braces must be written on the same line as the class, method, or control structure declaration.
- The maximum line length shall not exceed 100 characters to improve readability.
- Each method must include a brief comment describing its purpose and functionality.
- Meaningful and descriptive names must be used for all variables, methods, classes, and constants. Abbreviations should be avoided unless commonly understood.
- Consistent formatting shall be maintained throughout the entire project source code.

These standards improve readability, maintainability, collaboration efficiency, and overall consistency of the source code.

4.0 Naming Conventions

To ensure consistency, readability, and maintainability of the Student Learning Management System (SLMS), standardized naming conventions shall be applied throughout the project.

4.1 Class Naming

1. Class names shall use PascalCase.
2. Class names must represent nouns.
3. Each class name must match its file name exactly.

Examples from SLMS:

- Course
- Student

- Enrollment
- CourseManager
- StudentManager

4.2 Attribute Naming

1. Attributes shall use camelCase.
2. Attribute names must describe the stored data clearly.
3. All attributes shall be declared private to enforce encapsulation.

Examples:

- courseName
- courseCode
- creditHour
- courseSummary
- msTeamsLink
- studentId
- firstName
- lastName
- email
- phoneNum
- courseArray
- studentArray
- courseCount
- studentCount

4.3 Method Naming

1. Methods shall use camelCase.
2. Method names must begin with a verb.
3. Getter and setter methods must follow standard accessor naming format.

Examples:

- getCourseName()
- setCourseName(String name)
- addCourse(Course course)
- searchStudent(String studentId)
- deleteCourse(String courseCode)
- viewAllStudents()
- displayCourse()

4.4 Constant Naming

- Constants shall use UPPER_CASE with underscores.
- Constants must be declared as static and final (if Java is used).

Example:

- MAX_COURSE_LIMIT
- MAX_STUDENT_LIMIT

4.5 Vocabulary Dictionary

To ensure a shared understanding among developers and stakeholders, the following standardized terminology shall be used:

Term	**Definition**
Student	A registered learner enrolled in one or more courses
Lecturer	A user responsible for managing course and student information
System Administrator	A user with full system access and management authority
Course	A subject offered within the SLMS containing course details

Enrollment	The association between a student and a course
Course Manager	A controller class responsible for managing course operations
Student Manager	A controller class responsible for managing student operations
Course Code	A unique identifier assigned to each course
Student ID	A unique identifier assigned to each student
Credit Hour	The academic credit value assigned to a course
MS Teams Link	Online meeting link associated with a course
Course Profile	Complete details of a course
Student Profile	Complete details of a student

These naming conventions and standardized vocabulary reduce ambiguity, improve collaboration efficiency and maintain consistency across the system.

5.0 Function and Methods Modularity

The Student Learning Management System (SLMS) shall follow modular programming principles to ensure scalability, maintainability, and clean system architecture.

5.1 Encapsulation Principle

1. All class attributes shall be declared as private.
2. Access to attributes shall be controlled using getter and setter methods.

This ensures:

- Data protection
- Controlled modification
- Improved security
- Easier debugging

5.2 Single Responsibility Principle

Each method shall perform only one specific task.

Examples:

- addCourse() – adds a new course only
- searchCourse() – searches for a course only
- editStudent() – updates student details only
- viewAllCourses() – displays course list only

Methods must not combine multiple responsibilities such as validation, storage and display logic in one function.

5.3 Getter and Setter Methods

Getter Methods

- Used to retrieve attribute values.
- Must not modify object state.

Examples:

- getStudentId()
- getCourseCode()
- getCreditHour()

Setter Methods

- Used to modify attribute values.
- Must include input validation where necessary.

Examples:

- setEmail(String email)
- setCreditHour(int hours)

5.4 Constructor Methods

Each class shall implement a constructor to initialize essential attributes during object creation.

Example:

- Course(String courseName, String courseCode, int creditHour, String courseSummary, String msTeamsLink)
- Student(String firstName, String lastName, String studentId, String email, String phoneNum)

5.5 Method Size and Complexity Control

To maintain readability:

- Each method should not exceed 30–40 lines of code.
- Nested control structures shall be minimized.
- Repeated logic must be extracted into separate helper methods.

5.6 Separation of Responsibilities

The system shall separate responsibilities into different classes:

Entity Classes:

- Course
- Student
- Enrollment

Management Classes:

- CourseManager
- StudentManager

Entity classes store data and basic behaviors while Manager classes handle system-level operations such as adding, editing, searching, and deleting.

This structure improves modularity and prevents tightly coupled code.

5.7 Reusability and Maintainability

Common logic such as searching or validation must be reusable and placed in dedicated methods instead of duplicating code across multiple classes.

This modular structure ensures:

- Easier debugging
- Reduced redundancy
- Improved scalability
- Cleaner code maintenance

6.0 Continuous Integration Rules

The Student Learning Management System (SLMS) project shall use GitHub as the platform for version control and continuous integration management. Continuous integration practices ensure stable development, controlled collaboration, and high-quality source code throughout the project lifecycle.

6.1 Branching Strategy

The following branching strategy shall be implemented to maintain structured development:

- The main branch shall be named main.
- Feature branches shall be created from the main branch for each new functionality.
- Feature branches shall follow the naming format:
    - feature/course-management
    - feature/student-management
    - feature/profile-management
- Feature branches must be merged back into the main branch only after approval.

6.2 Commit Message Format

All commit messages must follow a clear and standardized format to improve traceability and project documentation. Examples include:

- feat: add course profile creation function
- feat: implement student search feature
- fix: correct student id validation error
- refactor: improve course manager structure

Commit messages must be concise, descriptive, and written in lowercase after the prefix.

6.3 Merge Rules

The following rules shall be enforced before merging any branch into the main branch:

- A pull request must be created before merging.
- The code must compile successfully without errors.
- The code must comply with the defined naming conventions and formatting standards.
- Peer review approval is required before merging.

These continuous integration rules ensure structured collaboration, reduce integration conflicts, and maintain high software quality standards.