STUDENT GRADE MANAGEMENT SYSTEM

1. Project Purpose and Features

The purpose of this project is to design and implement a **Student Grade Management System** using the principles of **Object-Oriented Programming (OOP)** in C++. The system aims to provide a structured and reliable way to manage students, courses, and grades while ensuring modularity, readability, and data persistence. This project demonstrates how real-world academic processes can be modeled using classes, relationships, and well-defined responsibilities.

The system allows users to manage student information by adding and removing students from the system. Each student is uniquely identified by an ID and is associated with personal information such as name, surname, and department. This approach ensures that student data is organized and easily accessible throughout the application.

Courses are centrally defined within the system and include essential information such as course code, course name, and credit value. By managing courses in a centralized structure, the system prevents invalid course entries and ensures consistency when grades are recorded. This design choice also simplifies grade validation and GPA calculation.

One of the core functionalities of the system is **entering and updating grades** for students. For each course, students receive grades for midterm, project, and final exams. These grades are combined using predefined weight percentages (midterm 30%, project 20%, final 50%) to calculate a course average. The system supports updating grades, allowing corrections or changes when necessary.

The application calculates student performance using a **credit-based weighted GPA system**. Each course contributes to the GPA proportionally to its credit value, reflecting real university grading systems. This ensures that high-credit courses have a greater impact on the overall GPA than low-credit courses. The GPA is normalized to a 4.00 scale, making the results intuitive and standardized.

In addition to individual student evaluation, the system provides **class-level statistics**. These statistics include class average GPA, median GPA, highest and lowest GPA, standard deviation, and counts of students above and below the class average. Furthermore, the system identifies honor students (GPA ≥ 3.50) and students in the academic risk zone (GPA < 2.00), offering meaningful insights into overall class performance.

Another important feature is the ability to **rank students by GPA**, enabling a clear comparison of academic performance. The system also generates a detailed **transcript** for each student, displaying enrolled courses, credits, numerical averages, letter grades, total credits, and cumulative GPA.

To ensure data persistence, the system supports **file input/output operations**. All student and grade data can be saved to a file and later loaded back into the system. This feature prevents data loss between program executions and simulates real-world academic record management systems.

Overall, this project successfully combines object-oriented design, data management, and practical academic requirements into a functional and extensible application. The system is designed to be easily maintainable and provides a strong foundation for future enhancements such as graphical user interfaces or database integration.

2. Class Design (with UML Diagram)

The class design of the Student Grade Management System is based on a clear separation of responsibilities and reflects the core principles of object-oriented programming. The system consists of four main components: **GradeManager, Student, Course, and Grade**. Their structure and relationships are illustrated in the UML class diagram presented in the report.

The GradeManager class acts as the central controller of the system. It is responsible for managing students and courses, handling all grade-related operations, and performing GPA and statistical calculations. This class maintains a collection of Student objects and a map of Course objects, enabling efficient access and management. Core functionalities such as adding and removing students, entering and updating grades, ranking students by GPA, generating transcripts, displaying class statistics, and performing file input/output operations are encapsulated within this class. By centralizing these responsibilities, the overall system structure remains clean, organized, and easy to maintain.

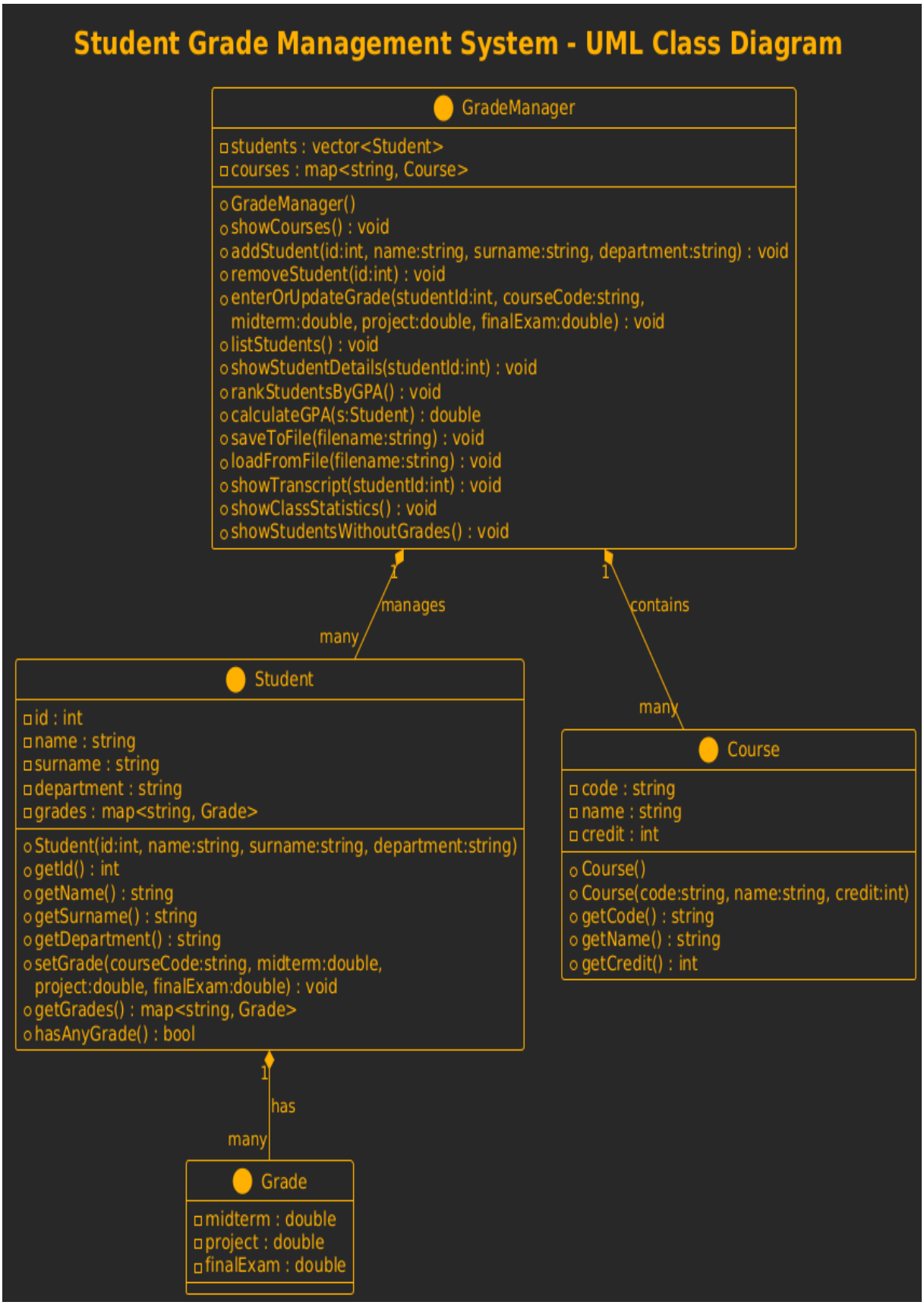
The Student class represents individual students in the system. Each student object stores personal information including student ID, name, surname, and department. Additionally, a student maintains a collection of grades associated with enrolled courses. This relationship is modeled using composition, as grades do not have an independent lifecycle outside the student. The Student class provides methods to set and retrieve grades, access student information, and check whether any grades have been recorded, ensuring that student-related data remains self-contained and logically grouped.

The Grade component is used to represent assessment data for a course. It stores midterm, project, and final exam scores. Since a grade is meaningful only in relation to a specific student, it is not implemented as an independent entity with standalone behavior. Instead, it is owned by the Student class, reinforcing the use of composition in the system design.

The Course class defines the academic courses available in the system. Each course includes a course code, course name, and credit value. Courses are centrally managed by the GradeManager class to ensure data consistency and to prevent invalid course entries. The credit value of each course plays a critical role in calculating the weighted GPA, allowing the system to accurately reflect academic importance.

The relationships between classes are intentionally designed without using inheritance. There are no “is-a” relationships in the problem domain; therefore, the system relies on has-a and manages relationships instead. GradeManager manages multiple Student and Course objects, while each Student owns multiple Grade objects. This design choice improves clarity, reduces unnecessary complexity, and closely aligns with real-world academic systems.

Overall, the UML class diagram provides a clear visualization of the system architecture, class responsibilities, and object relationships. The chosen design promotes modularity, maintainability, and extensibility, making the system easy to understand and suitable for future enhancements.


Student Grade Management System - UML Class Diagram
GradeManager
students : vector<Student>
courses : map<string, Course>
GradeManager()
showCourses() : void
addStudent(id:int, name:string, surname:string, department:string) : void
removeStudent(id:int) : void
enterOrUpdateGrade(studentId:int, courseCode:string, midterm:double, project:double, finalExam:double) : void
listStudents() : void
showStudentDetails(studentId:int) : void
rankStudentsByGPA() : void
calculateGPA(s:Student) : double
saveToFile(filename:string) : void
loadFromFile(filename:string) : void
showTranscript(studentId:int) : void
showClassStatistics() : void
showStudentsWithoutGrades() : void
1
manages
many
1
contains
many
Student
id : int
name : string
surname : string
department : string
grades : map<string, Grade>
Student(id:int, name:string, surname:string, department:string)
getId() : int
getName() : string
getSurname() : string
getDepartment() : string
setGrade(courseCode:string, midterm:double, project:double, finalExam:double) : void
getGrades() : map<string, Grade>
hasAnyGrade() : bool
Course
code : string
name : string
credit : int
Course()
Course(code:string, name:string, credit:int)
getCode() : string
getName() : string
getCredit() : int
1
has
many
Grade
midterm : double
project : double
finalExam : double

3. Key OOP Concepts Used

The Student Grade Management System is designed by applying fundamental **Object-Oriented Programming (OOP)** principles to ensure clarity, modularity, and maintainability. The most important OOP concepts used in this project are explained below.

Encapsulation

Encapsulation is achieved by defining class attributes as private and providing controlled access through public member functions. For example, student information such as ID, name, surname, department, and grades are stored as private members within the Student class. Access to this data is provided through getter methods and controlled update functions. This approach protects internal data from direct modification and ensures data integrity across the system.

Composition

The system heavily relies on composition to model real-world relationships. Each Student object owns multiple Grade objects, meaning that grades cannot exist independently of a student. Similarly, the GradeManager class manages collections of Student and Course objects. These relationships are modeled using composition to reflect strong ownership and lifecycle dependency, which aligns well with the academic domain.

Single Responsibility Principle

Each class in the system has a clearly defined responsibility. The Student class is responsible only for storing student-related data and grades. The Course class represents course information such as code, name, and credit value. The GradeManager class handles all business logic, including student management, grade processing, GPA calculations, statistics generation, and file input/output operations. This separation of responsibilities improves readability and simplifies maintenance.

Use of Collections

Standard C++ collection classes are used to efficiently manage data. A vector is used to store students, allowing dynamic addition and removal, while maps are used to associate course codes with Course objects and grades with courses. This choice enables fast lookup, organized data storage, and scalable system behavior.

File Input/Output (File I/O)

The system uses file input and output operations to provide data persistence. Student and grade information can be saved to a file and loaded back into the system in later executions. This feature ensures that data is not lost between program runs and simulates real-world academic record management systems.

Avoidance of Unnecessary Inheritance

Inheritance is intentionally not used in this project because there are no meaningful “is-a” relationships in the problem domain. Instead, the system relies on composition and object collaboration, resulting in a simpler and more accurate design.

4. Example Input/Output Screenshots

1. Main Menu of the Student Grade Management System

```
1. Add Student
2. Remove Student
3. Enter / Update Grade
4. List Students (with GPA)
5. Show Student Details
6. Rank Students by GPA
7. Save Data to File
8. Load Data from File
9. Show Transcript
10. Class Statistics
11. Show Students Without Grades
0. Exit
```

2. Adding a New Student to the System

```
Choice: 1
Enter Student ID: 1
Enter Name: Ali
Enter Surname: Arslan
Enter Department: Computer Engineering
Student added successfully.
```

3.Removing an Existing Student from the System

```
Choice: 2
ID: 1
Student removed successfully.
```

4.Displaying Available Courses and Entering Grades

```
Choice: 3

AVAILABLE COURSES
CODE         COURSE NAME                                CR
CMPE1101     INTRODUCTION TO PROGRAMMING                4
CMPE1604     OBJECT ORIENTED PROGRAMMING                4
CMPE2105     DATA STRUCTURES AND ALGORITHMS             4
CMPE3110     OPERATING SYSTEMS                          4
CMPE3111     DATABASE DESIGN AND MANAGEMENT             4
CMPE3112     INTRODUCTION TO DATA SCIENCE               4
CMPE3209     COMPUTER ORGANIZATION                      4
ECON1005     INTRODUCTION TO ECONOMY AND FINANCE        3
ELEC2151     CIRCUIT THEORY                             4
MATH1102     CALCULUS I                                 4
MATH1161     LINEAR ALGEBRA                             3
MATH1202     CALCULUS II                                4
MATH2128     DIFFERENTIAL EQUATIONS                     3
MATH2162     DISCRETE MATHEMATICS                       3
PHYS1101     PHYSICS I                                  4
PHYS1201     PHYSICS II                                 4
Enter Student ID: 1
Enter Course ID: CMPE1604
Enter Midterm (%30): 70
Enter Project (%20): 80
Enter Final (%50): 78
Grade saved.
```

5. Listing Students with Their GPA Values

```
Choice: 4
ID: 1 | Ali  Arslan | GPA: 3.04
ID: 2 | Zeynep  Yigit | GPA: 2.96
ID: 3 | Esma Demir | GPA: 3.48
```

6. Viewing Detailed Grade Information of a Student

```
Choice: 5
Student ID: 2

Student: Zeynep  Yigit
CMPE1604 -> M:70 P:65 F:80
```

7.Ranking Students by GPA

```
Choice: 6
Esma Demir -> GPA: 3.48
Ali  Arslan -> GPA: 3.04
Zeynep  Yigit -> GPA: 2.96
```

8. Saving Student and Grade Data to a File

```
Choice: 7
Data saved to file.
```

9. Loading Student and Grade Data from a File

```
Choice: 8
Data loaded from file.
```

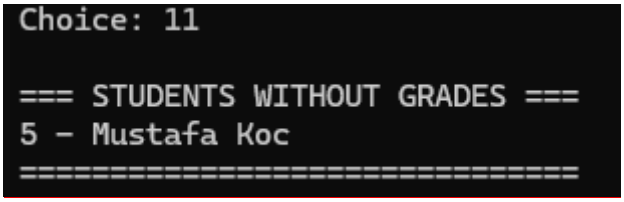
10. Displaying a Student Transcript

```
Choice: 9
Student ID: 3

========== TRANSCRIPT ==========
Esma Demir | Department: Computer Engineering

CODE        COURSE NAME                          CR    AVG       LETTER
-------------------------------------------------------------------
CMPE1604    OBJECT ORIENTED PROGRAMMING          4     87        BA

Total Credits: 4
GPA (4.00): 3.48
================================
```

11. Displaying Class Statistics

```
Choice: 10

========== CLASS STATISTICS ==========
Total Students      : 3
Class Avg GPA       : 3.16
Median GPA          : 3.04
Highest GPA         : 3.48
Lowest GPA          : 2.96
Std Deviation       : 0.228619
Above Avg Students  : 1
Below Avg Students  : 2
Honor Students      : 0 (GPA >= 3.50)
Risk Zone Students  : 0 (GPA < 2.00)
======================================
```

12. Listing Students Without Any Grades



Project Information:

Project Title: Student Grade Management System

Course: Object-Oriented Programming

Department: Computer Engineering

Institution: Istanbul University

Instructor: Dr. Ezgi ZORARPACI

Semester: 2025–2026 Spring

Developers:

Feyza KORKMAZ ,

Rayiha SÖNMEZ ,

Beytullah AYDIN .