

COURSE ENROLLMENT AND GRADE MANAGEMENT SYSTEM

This project is a simple command-line based course management system that allows administrators to manage courses and students. The system provides functionality to enroll students in courses, assign grades to students, and calculate overall course grades for each student. It consists of four main classes: Student, Course, CourseManagement, and AdministratorInterface.

Student: This class represents a student.

- name: A String variable that stores the name of the student.
- id: An integer variable that stores the ID of the student.
- enrolledCourses: A Map that stores the courses the student is enrolled in, mapped to the grade the student has in that course.

```
© Student.java × © Course.java © CourseManagement.java © AdministratorInterface.java
import java.util.ArrayList;
import java.util.HashMap;

public class Student {
    private String name;
    private int ID;
    private ArrayList<Course> enrolledCourses = new ArrayList<>();
    private HashMap<Course, Integer> courseGrades = new HashMap<>();

    public Student(String name, int ID) {
        this.name = name;
        this.ID = ID;
    }

    public int getID() {
        return this.ID;
    }

    public void setID(int newID) {
        this.ID = newID;
    }

    public String getName() {
        return this.name;
    }

    public void setName(String newName) {
        this.name = newName;
    }

    public ArrayList<Course> getEnrolledCourses() {
        return this.enrolledCourses;
    }

    public void enrollCourse(Course newCourse) {
        if (!this.enrolledCourses.contains(newCourse)) {
            this.enrolledCourses.add(newCourse);
            this.courseGrades.put(newCourse, 0);
            CourseManagement.updateStudentCourses(this, enrolledCourses);
            Course.enrolledStudentsCount++;
        }
    }

    public void assignGrade(Course course, int grade) {
        this.courseGrades.put(course, grade);
    }
}
```

Course: This class represents a course.

- name: A String variable that stores the name of the course.
- code: An integer variable that stores the course code.
- maxCapacity: An integer variable that stores the maximum capacity of the course.
- enrolledStudents: An ArrayList that stores the students enrolled in the course.

```
© Student.java × © Course.java × © CourseManagement.java © AdministratorInterface.java
1  import java.util.ArrayList;
2  import java.util.HashMap;
3
4  public class Course {
5      private String name;
6      private int courseCode;
7      private int maximumCapacity;
8      private ArrayList<Student> enrolledStudents = new ArrayList<>();
9      private HashMap<Student, Integer> studentGrades = new HashMap<>();
10     public static int enrolledStudentsCount;
11
12     public Course(String name, int courseCode, int maximumCapacity) {
13         this.name = name;
14         this.courseCode = courseCode;
15         this.maximumCapacity = maximumCapacity;
16     }
17
18     public int getCourseCode() {
19         return this.courseCode;
20     }
21
22     public void setCourseCode(int newCode) {
23         this.courseCode = newCode;
24     }
25
26     public int getMaximumCapacity() {
27         return this.maximumCapacity;
28     }
29
30     public void setMaximumCapacity(int newCapacity) {
31         this.maximumCapacity = newCapacity;
32     }
33
34     public String getName() {
35         return this.name;
36     }
37
38     public void setName(String newName) {
39         this.name = newName;
40     }
41
42     public ArrayList<Student> getEnrolledStudents() {
43         return this.enrolledStudents;
44     }
45
46     public void enrollStudent(Student student) {
47         if (this.enrolledStudents.size() < this.maximumCapacity) {
48             this.enrolledStudents.add(student);
49             this.studentGrades.put(student, 0);
50         } else {
51             System.out.println("Error: Course is at maximum capacity");
52         }
53     }
54 }
```

CourseManagement: This class is the main entry point of the application. It contains the main method and several static methods and variables to manage the course and student data.

```
© Student.java  © Course.java  © CourseManagement.java ×  © AdministratorInterface.java
1  import java.util.ArrayList;
2  import java.util.HashMap;
3
4  public class CourseManagement {
5      private static final ArrayList<Course> courses = new ArrayList<>();
6      private static final HashMap<Student, ArrayList<Course>> studentCourses = new HashMap<>();
7      private static final HashMap<Student, Integer> studentGrades = new HashMap<>();
8
9      public static void updateStudentCourses(Student student, ArrayList<Course> courses) {
10         studentCourses.put(student, courses);
11     }
12
13     public static void addCourse(String name, int code, int maximumCapacity) {
14         Course course = new Course(name, code, maximumCapacity);
15         courses.add(course);
16     }
17
18     @ public static void enrollStudent(Student student, Course course) {
19         student.enrollCourse(course);
20     }
21
22     @ public static void assignGrade(Student student, Course course, int grade) {
23         student.assignGrade(course, grade);
24     }
25
26     @ public static void calculateOverallGrade(Student student) {
27         if (student.courseGrades.isEmpty()) {
28             studentGrades.put(student, 0);
29         } else {
30             int sum = 0;
31             for (int grade : student.courseGrades.values()) {
32                 sum += grade;
33             }
34             studentGrades.put(student, sum / student.courseGrades.size());
35         }
36     }
37
38     public ArrayList<Course> getStudentCourses(Student student) {
39         return studentCourses.get(student);
40     }
41
42     @ public static Student[] getStudentList() {
43         return studentCourses.keySet().toArray(new Student[0]);
44     }
45
46     @ public static Course[] getCourseList() {
47         return courses.toArray(new Course[0]);
48     }
49
50     public static int getStudentOverallGrade(Student student) {
51         return studentGrades.getOrDefault(student, 0);
52     }
53 }
```

- `courseList`: A static `ArrayList` that stores all the `Course` objects.
- `studentList`: A static `ArrayList` that stores all the `Student` objects.
- `scanner`: A static `Scanner` object used to get user input from the command line.
- `main`: The main method that initializes the `Scanner` and starts the program.
- `start`: A static method that displays the administrator menu and handles user choices.
- `addCourse`: A static method that adds a new course to the course list.
- `enrollStudent`: A static method that enrolls a student in a course.
- `assignGrade`: A static method that assigns a grade to a student for a course.
- `displayCourses`: A static method that displays information about all courses.
- `displayStudents`: A static method that displays information about all students.

Static methods and variables are used in the `CourseManagement` class to track enrollment and grade-related information across multiple instances of the `Student` and `Course` classes. This allows the data to be accessed and modified from anywhere in the program without having to create an instance of the class.

Administrator: The Administrator class is designed to manage the course and student data. It has several methods to handle various tasks, such as adding a course, enrolling a student, assigning a grade, displaying courses, and displaying students.

- **addCourse:** A static method that takes parameters for the course name, course code, and maximum capacity. It creates a new Course object with the provided data and adds it to the `courseList`.
- **enrollStudent:** A static method that takes parameters for the student's name, ID, and the course code for the course you want to enroll them in. It creates a new Student object with the provided data, adds it to the `studentList`, and updates the `enrolledStudents` list in the respective Course object.
- **assignGrade:** A static method that takes parameters for the student's ID and the course code for the course you want to assign a grade for, along with the grade itself. It finds the respective Student and Course objects and updates the grade in the `enrolledCourses` map of the Student object.
- **displayCourses:** A static method that iterates through the `courseList` and prints out information about each course, including the course name, code, maximum capacity, and the students enrolled in the course.
- **displayStudents:** A static method that iterates through the `studentList` and prints out information about each student, including their name, ID, and the courses they are enrolled in along with their grades.

```
1 import java.util.Arrays;
2 import java.util.Optional;
3 import java.util.Scanner;
4
5 public class AdministratorInterface {
6     private static final Scanner scanner = new Scanner(System.in);
7
8     public static void main(String[] args) {
9         System.out.println("Welcome to the Administrator Interface");
10        start();
11    }
12
13    private static void start() {
14        System.out.println("\n(A:Add_New_Course B:Enroll_Student C:Assign_Grades D:Calculate_Overall_Grades E:Exit)");
15
16        String input = scanner.nextLine().toUpperCase();
17
18        switch (input) {
19            case "A":
20                addCourse();
21                break;
22
23            case "B":
24                System.out.println("New Student? Y/N");
25                String newStudent = scanner.nextLine().toUpperCase();
26
27                if (newStudent.equals("Y")) {
28                    enrollNewStudent();
29                } else if (newStudent.equals("N")) {
30                    enrollExistingStudent();
31                } else {
32                    System.out.println("Error: Unexpected Input");
33                    start();
34                }
35                break;
36
37            case "C":
38                assignGrades();
39                break;
40
41            case "D":
42                calculateOverallGrades();
43                break;
44
45            case "E":
46                System.out.println("Goodbye!");
47                System.exit(0);
48
49            default:
50                System.out.println("Error: Unexpected input");
51                start();
52                break;
53        }
54    }
55
56    public static void addCourse() {
57        System.out.println("Enter Course Name:");
58        String name = scanner.nextLine();
59
60        System.out.println("Enter Course Code:");
61        int code;
62        try {
63            code = Integer.parseInt(scanner.nextLine());
64        } catch (NumberFormatException e) {
65            System.out.println("Error: Invalid input for Course Code. Please enter a valid integer.");
66            start();
67            return;
68        }
69    }
```

```

68
69     System.out.println("Enter Maximum Capacity:");
70     int maxCapacity;
71     try {
72         maxCapacity = Integer.parseInt(scanner.nextLine());
73     } catch (NumberFormatException e) {
74         System.out.println("Error: Invalid input for Maximum Capacity. Please enter a valid integer.");
75         start();
76         return;
77     }
78
79     CourseManagement.addCourse(name, code, maxCapacity);
80
81     System.out.println("Successfully added " + name + "\nCourse code: " + code);
82     start();
83 }
84
85 public static void enrollNewStudent() {
86     System.out.println("Enter Student Name:");
87     String name = scanner.nextLine();
88
89     System.out.println("Enter Student ID:");
90     int ID = scanner.nextInt();
91     scanner.nextLine();
92
93     boolean studentExists = false;
94     for (Student existingStudent : CourseManagement.getStudentList()) {
95         if (ID == existingStudent.getID()) {
96             System.out.println("Error: ID already exists");
97             studentExists = true;
98             break;
99         }
100     }
101     if (studentExists) return;
102
103     Student student = new Student(name, ID);
104
105     while (true) {
106         System.out.println("Enter Course Code:");
107         int code = scanner.nextInt();
108         scanner.nextLine();
109
110         boolean courseExists = Arrays.stream(CourseManagement.getCourseList()).anyMatch(course -> course.getCourseCode() == code);
111
112         if (courseExists) {
113             Optional<Course> courseOpt = Arrays.stream(CourseManagement.getCourseList()).filter(course -> course.getCourseCode() == code).findFirst();
114
115             if (courseOpt.isPresent()) {
116                 Course course = courseOpt.get();
117                 CourseManagement.enrollStudent(student, course);
118                 System.out.println("Successfully enrolled " + student.getName() + " to " + course.getName());
119                 break;
120             }
121         } else {
122             System.out.println("Error: Course Code " + code + " doesn't exist");
123         }
124     }
125     start();
126 }
127

```



```

128 private static void enrollExistingStudent() {
129     System.out.println("Enter Student ID:");
130     int ID = scanner.nextInt();
131     scanner.nextLine();
132     Student student = getStudent(ID);
133
134     if (student == null) {
135         System.out.println("Error: ID doesn't exist");
136         enrollExistingStudent();
137
138     } else {
139         System.out.println("Enter Course Code:");
140         int code = scanner.nextInt();
141         scanner.nextLine();
142         Course course = getCourse(code);
143
144         if (course != null) {
145             System.out.println("Error: Course Code " + code + " doesn't exist");
146             enrollExistingStudent();
147
148         } else {
149             CourseManagement.enrollStudent(student, course);
150             System.out.println("Successfully enrolled " + student.getName() + " to " + course.getName());
151             System.out.println("Student ID: " + ID);
152             start();
153         }
154     }
155 }
156
157 private static void assignGrades() {
158     System.out.println("Enter Student ID:");
159     int ID = scanner.nextInt();
160     scanner.nextLine();
161     Student student = getStudent(ID);
162
163     if (student == null) {
164         System.out.println("Error: ID doesn't exist");
165         assignGrades();
166
167     } else {
168         System.out.println("Enter Course Code:");
169         int code = scanner.nextInt();
170         scanner.nextLine();
171         Course course = getCourse(code);
172
173         if (course == null) {
174             System.out.println("Error: Course code" + code + " doesn't exist");
175             assignGrades();
176
177         } else {
178             System.out.println("Enter Grade (1-100):");
179             int grade = scanner.nextInt();
180             scanner.nextLine();
181
182             CourseManagement.assignGrade(student, course, grade);
183             System.out.println(
184                 "Successfully added " + grade + " to " + student.getName() + " in " + course.getName());
185             start();
186         }
187     }
188 }

```

```

189
190     private static void calculateOverallGrades() {
191         System.out.println("Enter Student ID:");
192         int ID = scanner.nextInt();
193         scanner.nextLine();
194         Student student = getStudent(ID);
195
196         if (student == null) {
197             System.out.println("Error: ID doesn't exist");
198             assignGrades();
199
200         } else {
201             CourseManagement.calculateOverallGrade(student);
202             System.out.println("Successfully calculated Overall Grade");
203             System.out.println(student.getName() +
204                 "'s overall grade is " + CourseManagement.getStudentOverallGrade(student));
205             start();
206         }
207     }
208
209     @ ~ private static Student getStudent(int ID) {
210         ~ for (Student student : CourseManagement.getStudentList()) {
211             ~ if (ID == student.getID()) {
212                 ~ return student;
213             }
214         }
215         ~ return null;
216     }
217
218     @ ~ private static Course getCourse(int code) {
219         ~ for (Course course : CourseManagement.getCourseList()) {
220             ~ if (code == course.getCourseCode()) {
221                 ~ return course;
222             }
223         }
224         ~ return null;
225     }
226 }

```

How to run the program

To run the program, you need to have Java installed on your computer. You can then compile the program using the command `javac CourseManagement.java` and run it using the command `java CourseManagement`.

Interacting with the administrator interface

The program will display an administrator menu, and you can interact with the program by entering the corresponding numbers for each option and providing the necessary information when prompted.

Add Course: You will be prompted to enter the name, code, and maximum capacity of the course. The course will then be added to the course list.

Enroll Student: You will first be prompted to choose whether you want to enroll a new student or an existing student. If you choose to enroll a new student, you will be prompted to enter the student's name and ID, and then the course code for the course you want to enroll them in. If you choose to enroll an existing student, you will be prompted to enter the student's ID and the course code.

Assign Grade: You will be prompted to enter the student's ID and the course code for the course you want to assign a grade for. Then, you will be prompted to enter the grade.

Display Courses: This will display information about all courses, including the course name, code, maximum capacity, and the students enrolled in the course.

Display Students: This will display information about all students, including their name, ID, and the courses they are enrolled in along with their grades.

Exit: This will terminate the program.