**Course Enrollment and Grade Management System**

Student Class

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

public class Student {

private String name;

private String id;

private ArrayList<Course> enrolledCourses;

private Map<Course, Double> grades;

public Student(String name, String id) {

this.name = name;

this.id = id;

this.enrolledCourses = new ArrayList<>();

this.grades = new HashMap<>();

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public ArrayList<Course> getEnrolledCourses() {

return enrolledCourses;

}

public void enrollCourse(Course course) {

if (course.getCurrentEnrollment() < course.getMaxCapacity()) {

enrolledCourses.add(course);

course.incrementEnrollment();

System.out.println("Enrolled in course: " + course.getName());

} else {

System.out.println("Cannot enroll in " + course.getName() + ": Capacity full.");

}

}

public void assignGrade(Course course, double grade) {

if (enrolledCourses.contains(course)) {

grades.put(course, grade);

System.out.println("Grade " + grade + " assigned for course: " + course.getName());

} else {

System.out.println("Cannot assign grade for course: " + course.getName() + " - Not enrolled.");

}

}

public double getGrade(Course course) {

return grades.getOrDefault(course, -1.0);

}

}

Course Class

public class Course {

private String courseCode;

private String name;

private int maxCapacity;

private int currentEnrollment;

private static int totalEnrolledStudents = 0;

public Course(String courseCode, String name, int maxCapacity) {

this.courseCode = courseCode;

this.name = name;

this.maxCapacity = maxCapacity;

this.currentEnrollment = 0;

}

public String getCourseCode() {

return courseCode;

}

public String getName() {

return name;

}

public int getMaxCapacity() {

return maxCapacity;

}

public int getCurrentEnrollment() {

return currentEnrollment;

}

public void incrementEnrollment() {

if (currentEnrollment < maxCapacity) {

currentEnrollment++;

totalEnrolledStudents++;

}

}

public static int getTotalEnrolledStudents() {

return totalEnrolledStudents;

}

}

CourseManagement Class

import java.util.ArrayList;

import java.util.HashMap;

import java.util.Map;

public class CourseManagement {

private static ArrayList<Course> courses = new ArrayList<>();

private static Map<Student, Map<Course, Double>> studentGrades = new HashMap<>();

public static void addCourse(String courseCode, String name, int maxCapacity) {

Course course = new Course(courseCode, name, maxCapacity);

courses.add(course);

System.out.println("Course added: " + name);

}

public static void enrollStudent(Student student, Course course) {

student.enrollCourse(course);

}

public static void assignGrade(Student student, Course course, double grade) {

student.assignGrade(course, grade);

if (!studentGrades.containsKey(student)) {

studentGrades.put(student, new HashMap<>());

}

studentGrades.get(student).put(course, grade);

}

public static double calculateOverallGrade(Student student) {

Map<Course, Double> grades = studentGrades.get(student);

if (grades == null || grades.isEmpty()) {

System.out.println("No grades available for student: " + student.getName());

return 0;

}

double totalGrades = 0;

for (double grade : grades.values()) {

totalGrades += grade;

}

return totalGrades / grades.size();

}

public static void listCourses() {

System.out.println("Available courses:");

for (Course course : courses) {

System.out.println(course.getName() + " (" + course.getCourseCode() + ")");

}

}

}

Administrator Interface

import java.util.Scanner;

public class AdministratorInterface {

public static void main(String[] args) {

Scanner scanner = new Scanner(System.in);

while (true) {

System.out.println("\nAdministrator Menu:");

System.out.println("1. Add a new course");

System.out.println("2. Enroll student in a course");

System.out.println("3. Assign grade to a student");

System.out.println("4. Calculate overall course grade for a student");

System.out.println("5. List all courses");

System.out.println("6. Exit");

System.out.print("Enter your choice: ");

int choice = scanner.nextInt();

scanner.nextLine(); // Consume newline

switch (choice) {

case 1:

System.out.print("Enter course code: ");

String courseCode = scanner.nextLine();

System.out.print("Enter course name: ");

String courseName = scanner.nextLine();

System.out.print("Enter maximum capacity: ");

int maxCapacity = scanner.nextInt();

CourseManagement.addCourse(courseCode, courseName, maxCapacity);

break;

case 2:

System.out.print("Enter student name: ");

String studentName = scanner.nextLine();

System.out.print("Enter student ID: ");

String studentId = scanner.nextLine();

Student student = new Student(studentName, studentId);

CourseManagement.listCourses();

System.out.print("Enter course code to enroll: ");

String enrollCourseCode = scanner.nextLine();

Course enrollCourse = CourseManagement.courses.stream()

.filter(c -> c.getCourseCode().equals(enrollCourseCode))

.findFirst()

.orElse(null);

if (enrollCourse != null) {

CourseManagement.enrollStudent(student, enrollCourse);

} else {

System.out.println("Course not found.");

}

break;

case 3:

System.out.print("Enter student name: ");

String assignStudentName = scanner.nextLine();

System.out.print("Enter student ID: ");

String assignStudentId = scanner.nextLine();

Student assignStudent = new Student(assignStudentName, assignStudentId);

CourseManagement.listCourses();

System.out.print("Enter course code to assign grade: ");

String gradeCourseCode = scanner.nextLine();

Course gradeCourse = CourseManagement.courses.stream()

.filter(c -> c.getCourseCode().equals(gradeCourseCode))

.findFirst()

.orElse(null);

if (gradeCourse != null) {

System.out.print("Enter grade: ");

double grade = scanner.nextDouble();

CourseManagement.assignGrade(assignStudent, gradeCourse, grade);

} else {

System.out.println("Course not found.");

}

break;

case 4:

System.out.print("Enter student name: ");

String calcStudentName = scanner.nextLine();

System.out.print("Enter student ID: ");

String calcStudentId = scanner.nextLine();

Student calcStudent = new Student(calcStudentName, calcStudentId);

double overallGrade = CourseManagement.calculateOverallGrade(calcStudent);

System.out.println("Overall course grade for " + calcStudent.getName() + ": " + overallGrade);

break;

case 5:

CourseManagement.listCourses();

break;

case 6:

System.out.println("Exiting...");

scanner.close();

return;

default:

System.out.println("Invalid choice. Please try again.");

}

}

}

}

**Explanation**

Student Class:

Stores student information (name, ID, enrolled courses).

Methods to enroll in courses and assign grades.

Uses private instance variables and public getters/setters.

Course Class:

Stores course information (course code, name, maximum capacity).

Static variable to track total enrolled students.

Methods to get course information and increment enrollment count.

CourseManagement Class:

Static variables to store a list of courses and student grades.

Methods to add courses, enroll students, assign grades, and calculate overall grades.

Administrator Interface:

Command-line interface for administrators.

Menu options to add courses, enroll students, assign grades, and calculate overall grades.

Error handling for invalid inputs and full course capacity.

**Documentation**

Student Class:

Student(String name, String id): Constructor to initialize student name and ID.

enrollCourse(Course course): Enrolls the student in the given course.

assignGrade(Course course, double grade): Assigns a grade to the student for the given course.

Course Class:

Course(String courseCode, String name, int maxCapacity): Constructor to initialize course details.

static int getTotalEnrolledStudents(): Returns the total number of enrolled students across all courses.

CourseManagement Class:

`static void addCourse(String

**Output**

Here's how the Course Enrollment and Grade Management System looks like:

**Example Run of Administrator Interface**

Adding a New Course

Administrator Menu:

1. Add a new course

2. Enroll student in a course

3. Assign grade to a student

4. Calculate overall course grade for a student

5. List all courses

6. Exit

Enter your choice: 1

Enter course code: CS101

Enter course name: Introduction to Computer Science

Enter maximum capacity: 30

Course added: Introduction to Computer Science

Enrolling a Student in a Course

Administrator Menu:

1. Add a new course

2. Enroll student in a course

3. Assign grade to a student

4. Calculate overall course grade for a student

5. List all courses

6. Exit

Enter your choice: 2

Enter student name: Alice Johnson

Enter student ID: S001

Available courses:

Introduction to Computer Science (CS101)

Enter course code to enroll: CS101

Enrolled in course: Introduction to Computer Science

Assigning a Grade to a Student

Administrator Menu:

1. Add a new course

2. Enroll student in a course

3. Assign grade to a student

4. Calculate overall course grade for a student

5. List all courses

6. Exit

Enter your choice: 3

Enter student name: Alice Johnson

Enter student ID: S001

Available courses:

Introduction to Computer Science (CS101)

Enter course code to assign grade: CS101

Enter grade: 95

Grade 95.0 assigned for course: Introduction to Computer Science

Calculating Overall Course Grade for a Student

Administrator Menu:

1. Add a new course

2. Enroll student in a course

3. Assign grade to a student

4. Calculate overall course grade for a student

5. List all courses

6. Exit

Enter your choice: 4

Enter student name: Alice Johnson

Enter student ID: S001

Overall course grade for Alice Johnson: 95.0