**Course Enrollment and Grade Management System Documentation**

**Purpose**

The Course Enrollment and Grade Management System is designed to assist universities in managing student enrollment in courses, assigning grades, and calculating overall grades for students. The system provides a modular and object-oriented approach to organize code, encapsulate data and behavior within classes, and effectively utilize static methods and variables to track enrollment and grade-related information.

**Classes**

**1. Student Class:**

**Purpose:**

* Represents a student with information such as name, ID, and enrolled courses.

**Instance Variables:**

* **name**: String – It saves name of student.
* **studentID**: String - stores the unique identifier of the student.
* **enrolledCourses**: List<Course> - stores the list of courses in which the students are enrolled.

**Methods**:

* **enrollInCourse(Course course)**: Adds the provided course to the list of enrolled courses for the student.
* **assignGrade(Course course, int grade)**: Assigns a grade for the student in the specified course.

**2. Course Class:**

**Purpose**:

* Represents a course with information such as course code, name, maximum capacity, and the number of enrolled students.

**Instance Variables:**

* **courseCode**: String - stores the unique identifier of the course.
* **courseName**: String - stores the name of the course.
* **maxCapacity**: int - stores the maximum number of students that could enroll in tcourse.
* **enrolledStudents**: int - stores the current number of students enrolled in the course.

**Static Variables:**

* **totalEnrolledStudents**: int - static variable to keep track of total number of enrolled students across all instances of the class.

**Methods**:

* **incrementEnrolledStudents()**: Increments the count of enrolled students for the course.
* **getTotalEnrolledStudents()**: Returns the total number of enrolled students across all courses.

**3. CourseManagement Class:**

**Purpose**:

* Serves as a management class for handling courses, student enrollments, grades, and overall grade calculations.

**Static Variables:**

* **courses**: List<Course> - static variable to store the list of courses.
* **studentGrades**: Map<Student, Map<Course, Integer>> - static variable to store grades for each student in each course.

**Methods**:

* **addCourse(String courseCode, String courseName, int maxCapacity)**: Add a new course to old the list of courses.
* **enrollStudent(Student student, Course course)**: Add a student in the course and updates the total enrolled students count.
* **assignGrade(Student student, Course course, int grade)**: Assigns a grade for a student in a course and updates the studentGrades map.
* **calculateOverallGrade(Student student)**: Calculates the overall grade for a student based on individual grades.

**4. Course\_Enrollment\_and\_Grade\_Management\_System Class:**

**Purpose**:

* Provides an interactive command-line interface for administrators to interact with the Course Enrollment and Grade Management System.

**Methods**:

* **main(String[] args)**: Displays a menu with options to add a new course, enroll students, assign grades, calculate overall course grades, and exit.

**Static Methods and Variables**

* **Purpose:** Static methods and variables are utilized to maintain shared data across multiple instances of the Course and Student classes, allowing the system to keep track of enrollment and grade-related information.

**In Course Class:**

* **totalEnrolledStudents** (static variable): Keeps track of total number of enrolled students across all instances of the class.

**In CourseManagement Class:**

* **courses** (static variable): Maintains a shared list of courses accessible across all instances of the CourseManagement class.
* **studentGrades** (static variable): Manages grades for each student in each course, accessible and modifiable across all instances of the CourseManagement class.

**Running the Program**

1. **Interact with the Administrator Interface:**
   * Follow the on-screen instructions to choose options from the menu (1-5).
   * Input data as requested (e.g., course information, student information, grades).
   * The program will display success messages or error messages based on the operations performed.
2. **Exit the Program:**
   * To exit the program, choose option 5 from the menu.

**Program:**

import java.util.List;

import java.util.Scanner;

public class Course\_Enrollment\_and\_Grade\_Management\_System {

    private static final Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {

        while (true) {

            displayMenu();

            int choice = getUserChoice();

            switch (choice) {

                case 1:

                    addCourse();

                    break;

                case 2:

                    enrollStudent();

                    break;

                case 3:

                    assignGrade();

                    break;

                case 4:

                    calculateOverallGrade();

                    break;

                case 5:

                    System.out.println("\nExiting the program. Goodbye!\n");

                    System.exit(0);

                default:

                    System.out.println("Bad choice. Please try again.\n");

            }

        }

    }

    private static void displayMenu() {

        System.out.println(

                "\n\n\n##########\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    === Administrator Interface ===                   \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*##########\n");

        System.out.println(

                "##########\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    1. Add a new course                               \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*##########");

        System.out.println(

                "##########\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    2. Enroll a student                               \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*##########");

        System.out.println(

                "##########\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    3. Assign a grade                                 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*##########");

        System.out.println(

                "##########\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    4. Calculate overall course grade for a student   \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*##########");

        System.out.println(

                "##########\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    5. Exit                                           \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*##########");

    }

    private static int getUserChoice() {

        System.out.print("##########\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    Enter your choice (1-5):        ");

        while (!scanner.hasNextInt()) {

            System.out.println(

                    "##########\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*    Invalid! Please enter any number.      \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*##########");

            scanner.next(); // consume the invalid input

        }

        return scanner.nextInt();

    }

    private static void addCourse() {

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

        String courseCode = scanner.next();

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

        String courseName = scanner.next();

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

        while (!scanner.hasNextInt()) {

            System.out.println("Invalid! Please enter any number.");

            scanner.next();

        }

        int maxCapacity = scanner.nextInt();

        CourseManagement.addCourse(courseCode, courseName, maxCapacity);

        System.out.println("Course added successfully!");

    }

    private static void enrollStudent() {

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

        String studentName = scanner.next();

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

        int studentID = scanner.nextInt();

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

        String courseCode = scanner.next();

        Student student = new Student(studentName, studentID);

        Course course = findCourseByCode(courseCode);

        if (course != null) {

            CourseManagement.enrollStudent(student, course);

            System.out.println("Student registered successfully!");

        } else {

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

        }

    }

    private static void assignGrade() {

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

        int studentID = scanner.nextInt();

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

        String courseCode = scanner.next();

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

        while (!scanner.hasNextInt()) {

            System.out.println("Invalid! Please enter any number.");

            scanner.next();

        }

        int grade = scanner.nextInt();

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

        for (Course c : CourseManagement.getCourses()) {

            System.out.println("Course Code: " + c.getCourseCode() + ", Course Name: " + c.getCourseName());

        }

        System.out.println("\nAvailable Students:");

        for (Course c : CourseManagement.getCourses()) {

            for (Student s : c.getEnrolledStudents()) {

                System.out.println("Student ID: " + s.getStdID() + ", Student Name: " + s.getstdName());

            }

        }

        Student student = findStudentById(studentID);

        Course course = findCourseByCode(courseCode);

        if (student != null && course != null) {

            CourseManagement.assignGrade(student, course, grade);

            System.out.println("Student grade updated successfully!");

        } else {

            System.out.println("Error: Student or course not found.");

        }

    }

    private static void calculateOverallGrade() {

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

        int studentID = scanner.nextInt();

        Student student = findStudentById(studentID);

        if (student != null) {

            int overallGrade = CourseManagement.calculateOverallGrade(student);

            System.out.println("Overall grade for student " + student.getstdName() + " is: " + overallGrade);

        } else {

            System.out.println("Error: Student not found.");

        }

    }

    private static Course findCourseByCode(String courseCode) {

        for (Course course : CourseManagement.getCourses()) {

            if (course.getCourseCode().equals(courseCode)) {

                return course;

            }

        }

        return null;

    }

    private static Student findStudentById(int studentId) {

        for (Course course : CourseManagement.getCourses()) {

            List<Student> enrolledStudents = course.getEnrolledStudents();

            if (enrolledStudents != null) {

                for (Student student : enrolledStudents) {

                    if (student.getStdID() == studentId) {

                        return student;

                    }

                }

            }

        }

        return null;

    }

}

import java.util.ArrayList;

import java.util.List;

class CourseManagement {

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

    public static List<Course> getCourses() {

        return courses;

    }

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

        Course newCourse = new Course(courseCode, courseName, maxCapacity);

        courses.add(newCourse);

    }

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

        if (courses.contains(course) && course.getEnrolledStudents().size() < course.getMaxCapacity()) {

            student.enrollInCourse(course);

            course.setEnrolledStudents(student);

        } else {

            System.out.println("Error: : It seems that course is full");

        }

    }

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

        if (courses.contains(course) && student.getEnrolledCourses().contains(course)) {

            student.assignGrade(course, grade);

        } else {

            System.out.println("Error: Invalid course or student.");

        }

    }

    public static int calculateOverallGrade(Student student) {

        int totalGrades = 0;

        int numberOfCourses = student.getEnrolledCourses().size();

        for (Course course : student.getEnrolledCourses()) {

            List<Integer> grades = course.getGrades();

            if (!grades.isEmpty()) {

                totalGrades += grades.get(0);

            }

        }

        return numberOfCourses > 0 ? totalGrades / numberOfCourses : 0;

    }

}

import java.util.ArrayList;

import java.util.List;

class Course {

    private String courseCode;

    private String courseName;

    private int maxCapacity;

    private List<Student> enrolledStudents;

    private List<Integer> grades;

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

        this.courseCode = courseCode;

        this.courseName = courseName;

        this.maxCapacity = maxCapacity;

        this.enrolledStudents = new ArrayList<>();

        this.grades = new ArrayList<>();

    }

    public String getCourseCode() {

        return courseCode;

    }

    public String getCourseName() {

        return courseName;

    }

    public int getMaxCapacity() {

        return maxCapacity;

    }

    public List<Student> getEnrolledStudents() {

        return enrolledStudents;

    }

    public List<Integer> getGrades() {

        return grades;

    }

    public void setEnrolledStudents(Student student) {

        this.enrolledStudents.add(student);

    }

    public void assignGrade(Student student, int grade) {

        grades.add(grade);

    }

}

import java.util.ArrayList;

import java.util.List;

class Student {

    private String stdName;

    private int stdID;

    private List<Course> enrolledCourses;

    public Student(String stdName, int stdID) {

        this.stdName = stdName;

        this.stdID = stdID;

        this.enrolledCourses = new ArrayList<>();

    }

    public String getstdName() {

        return stdName;

    }

    public int getStdID() {

        return stdID;

    }

    public List<Course> getEnrolledCourses() {

        return enrolledCourses;

    }

    public void enrollInCourse(Course course) {

        if (!enrolledCourses.contains(course)) {

            enrolledCourses.add(course);

        } else {

            System.out.println("Student already enrolled in this course.");

        }

    }

    public void assignGrade(Course course, int grade) {

        if (enrolledCourses.contains(course)) {

            course.assignGrade(this, grade);

        } else {

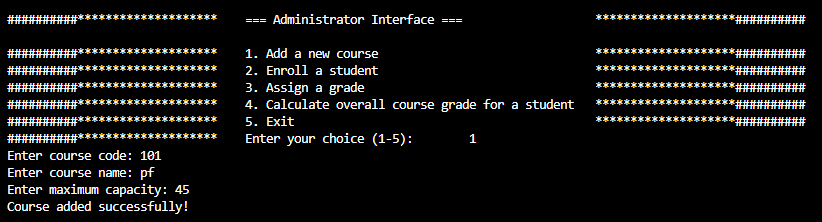
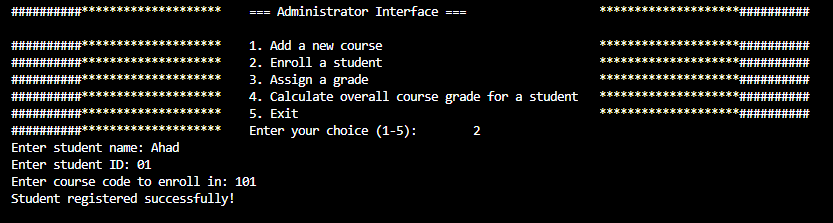
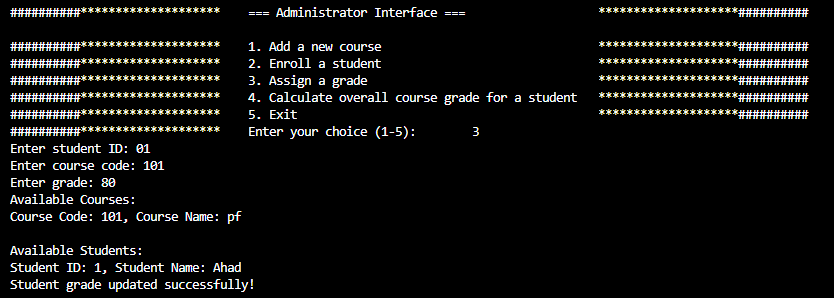
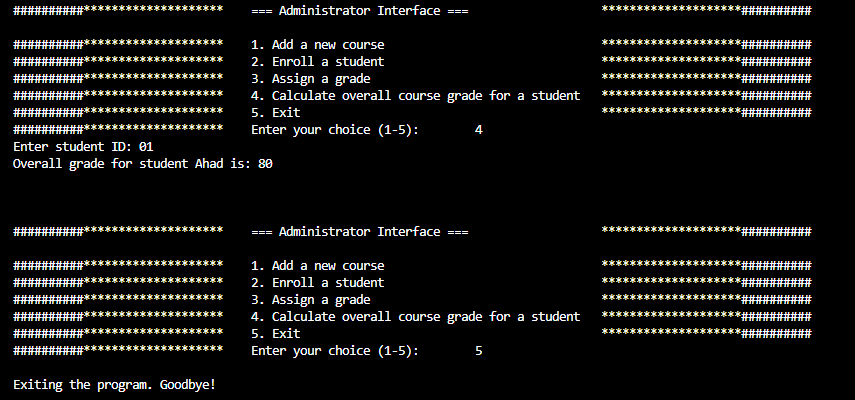
            System.out.println("Student not enrolled in this course.");

        }

    }

}

**Output Snapshots:**

  
  
  
  
  
  
  
  
  
**The End**