Saveetha School of Engineering

Saveetha Institute of Medical and Technical Sciences

Department of Computer Science Engineering

Assignment on topic: Unit 1

CO2: Develop object-oriented programs using Java classes and objects

How would you design the Student management class and implement the described functionality for this student management system? Include the code and expected output based on the given scenario.

Scenario:

You are working as a software developer for a university that wants to build an **online student management system**. Administrators should be able to manage student records, view details of enrolled students, and update information such as grades and courses.

You need to design a class StudentRecord that can represent individual student records with the following details:

- **studentName**: Name of the student.
- **studentID**: A unique identifier for the student.
- **enrolledCourses**: A list of courses the student is currently enrolled in.
- **grades**: A dictionary to store grades for each course.
- **isActive**: Boolean to track whether the student is currently active or inactive.

Tasks:

1. Define the StudentRecord class:

- Create attributes to store the following information:
    - studentName: Name of the student.
    - studentID: A unique identifier for the student.
    - enrolledCourses: A list of courses the student is enrolled in.
    - grades: A dictionary to store grades for each course.
    - isActive: Boolean to track whether the student is currently active or inactive.

2. Implement the following methods:

- **displayStudentInfo()**: Display the student's name, ID, enrolled courses, and active status.
- **addCourse(courseName)**: Add a new course to the student’s enrolled courses.
- **updateGrade(courseName, grade)**: Update the grade for a specific course. If the course does not exist, inform the user.
- **deactivateStudent()**: Mark the student as inactive.
- **reactivateStudent()**: Mark the student as active.

3. Simulate student record management:

- Create two objects representing two different students, each with unique IDs, names, and enrolled courses.
- Display the details of each student.
- Add a new course to one of the student’s records and update the grades for their courses.
- Deactivate one of the students and display their updated details.
- Attempt to add a grade for a course that the student is not enrolled in, testing the system's response.

Steps :

- Display the details of both students.
- Enroll the first student in a new course.
- Update the grades for the first student’s courses.
- Deactivate the first student and display their updated details.
- Attempt to add a grade for a course that the second student is not enrolled in to validate error handling.

Test Scenarios:

Scenario 1:

1. **Student Name**: John Doe
    - **ID**: S101
    - **Enrolled Courses**: ["Mathematics", "Science"]
    - **Grades**: {"Mathematics": "A", "Science": "B"}
    - **Status**: Active

2. **Student Name**: Jane Smith
    - **ID**: S102
    - **Enrolled Courses**: ["History", "English"]
    - **Grades**: {"History": "B", "English": "A"}
    - **Status**: Active

Deliverables:

1. The system should correctly display the details of both students.
2. A new course should be successfully added to the first student's record.
3. Grades for courses should be updated correctly.
4. The system should accurately reflect the deactivation of the first student.
5. Attempting to add a grade for a non-enrolled course should produce an error message.

Grading Rubrics

Criterion	Needs Improvement (0-2)	Satisfactory (3-5)	Good (6-8)	Excellent (9-10)
Class Design	- Missing class definition.	- Basic class structure with some attributes but incomplete.	- Class structure is mostly correct with minor naming convention issues.	- Complete class design with all required attributes.
	- Missing or incorrect attributes.	- Lacks some essential attributes.	- Proper use of encapsulation and correct attributes.	- Excellent use of encapsulation, clear naming conventions.
Constructor Implementation	- Missing or incomplete constructor.	- Constructor present but some attributes are not initialized correctly.	- Constructor initializes most attributes, with minor issues like defaults.	- Properly initializes all attributes. Clear and logical parameters.

Object Creation and Testing	- Missing or incorrect student record creation.	- Objects are created but testing is incomplete or incorrect.	- Objects are created and tested with minor edge cases missed.	- Objects are correctly created and thoroughly tested, including edge cases.
Method Implementation	- Missing or incomplete methods.	- Basic methods implemented but lack robustness or error handling.	- Most methods are implemented with minor logical issues.	- All required methods implemented, handling edge cases effectively.
Code Quality and Testing	- Poor code quality with many errors.	- Code works but lacks consistency and has untested parts.	- Code is clean and follows best practices but misses some edge cases.	- High-quality, well-documented code. Fully tested, handles all edge cases.
Submission Deadline	- Frequently misses deadlines.	- Submits work within a short grace period.	- Deadlines are met in most cases with minor delays.	- Always meets or beats submission deadlines.

SAVEETHA INSTITUTE OF MEDICAL AND TECHNICAL SCIENCES

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CSA09 – PROGRAMMING IN JAVA

ASSIGNMENT REPORT

REGISTER NUMBER: 192365009

NAME: V.S.SUPRAJA

SUBMSSION DATE:

STUDENT MANAGEMNET SYSTEM

Objective:

The objective of this assignment is to design and implement a basic **student management system**. The system should allow administrators to view student details, manage course enrollments, update grades, and toggle the active status of a student. This functionality will be achieved through the design of a StudentRecord class with relevant attributes and methods.

Design and Implementation:

The solution was designed based on the requirements provided in the scenario. The key aspects of the implementation are as follows:

Class Design:

- The class StudentRecord was created to represent a student record.
- **Attributes include:**
    - **studentName**: The name of the student.
    - **studentID**: The unique identifier for the student.
    - **enrolledCourses**: A list of courses the student is currently enrolled in.
    - **grades**: A dictionary to store grades for each course.
    - **isActive**: A boolean value indicating if the student is active or inactive.

Methods Implemented:

- **displayStudentInfo()**: This method displays the details of the student, including their name, ID, enrolled courses, grades, and active status.
- **addCourse(courseName)**: This method adds a new course to the student's record.
- **updateGrade(courseName, grade)**: This method updates the grade for a specific course. If the course does not exist, the method notifies the user.
- **deactivateStudent()**: This method sets the student's active status to inactive.
- **reactivateStudent()**: This method reactivates a previously inactive student.

Program Execution:

The program performs the following actions:

- Displaying the details of two student records.
- Adding a new course for the student "John Doe".
- Updating the grades for the courses of "John Doe".
- Deactivating "John Doe" and displaying the updated details.
- Attempting to add a grade for a course not enrolled by "Jane Smith" to test error handling.

Summary of Operations:

1. **Displaying Student Details**
    - Displays the initial details of both students, including their name, ID, enrolled courses, grades, and active status.
2. **Managing Courses**
    - Adds a new course for one of the students and confirms the update.
3. **Updating Grades**
    - Updates the grades for specific courses and verifies correctness.
4. **Changing Active Status**
    - Toggles the active status of a student and ensures the status is reflected accurately.
5. **Error Handling**
    - Tests the system's ability to handle updates to grades for courses a student is not enrolled in.

Code:

```
// Class definition for StudentRecord
class StudentRecord {
    private String studentName;
    private String studentID;
    private String[] enrolledCourses;
    private String[] grades;
    private boolean isActive;

    // Constructor to initialize student record
    public StudentRecord(String studentName, String studentID, String[] enrolledCourses) {
        this.studentName = studentName;
        this.studentID = studentID;
        this.enrolledCourses = enrolledCourses;
        this.grades = new String[enrolledCourses.length];
        this.isActive = true; // Initially, the student is active
    }

    // Method to display student information
    public void displayStudentInfo() {
        System.out.println("Student Name: " + studentName);
        System.out.println("Student ID: " + studentID);
        System.out.println("Active Status: " + (isActive ? "Active" : "Inactive"));
        System.out.println("Enrolled Courses and Grades:");
        for (int i = 0; i < enrolledCourses.length; i++) {
            System.out.println("- " + enrolledCourses[i] + ": " + (grades[i] == null ? "Not
Graded" : grades[i]));
        }
    }

    // Method to add a grade for a specific course
```

```
    public void updateGrade(String courseName, String grade) {
        for (int i = 0; i < enrolledCourses.length; i++) {
            if (enrolledCourses[i].equals(courseName)) {
                grades[i] = grade;
                System.out.println("Grade updated for course: " + courseName);
                return;
            }
        }
        System.out.println("Course not found: " + courseName);
    }

    // Method to deactivate the student
    public void deactivateStudent() {
        isActive = false;
        System.out.println("Student " + studentName + " has been deactivated.");
    }

    // Method to reactivate the student
    public void reactivateStudent() {
        isActive = true;
        System.out.println("Student " + studentName + " has been reactivated.");
    }
}

// Driver class for Student Management System
public class StudentManagementSystem {
    public static void main(String[] args) {
        // Create student records
        String[] courses1 = {"Mathematics", "Science"};
        String[] courses2 = {"History", "English"};
```

```
    StudentRecord student1 = new StudentRecord("John Doe", "S101", courses1);
    StudentRecord student2 = new StudentRecord("Jane Smith", "S102", courses2);

    // Display student info
    System.out.println("Student 1 Info:");
    student1.displayStudentInfo();
    System.out.println("\nStudent 2 Info:");
    student2.displayStudentInfo();

    // Update grade for a course for student 1
    System.out.println("\nUpdating grade for Student 1:");
    student1.updateGrade("Mathematics", "A");

    // Deactivate student 1
    System.out.println("\nDeactivating Student 1:");
    student1.deactivateStudent();

    // Attempt to update a grade for a non-enrolled course for student 2
    System.out.println("\nUpdating grade for a non-enrolled course for Student 2:");
    student2.updateGrade("Mathematics", "B");

    // Reactivate student 1
    System.out.println("\nReactivating Student 1:");
    student1.reactivateStudent();

    // Display updated student info
    System.out.println("\nUpdated Student 1 Info:");
    student1.displayStudentInfo();
}
```

}

Output:

```
Student 1 Info:
Student Name: John Doe
Student ID: S101
Active Status: Active
Enrolled Courses and Grades:
- Mathematics: Not Graded
- Science: Not Graded

Student 2 Info:
Student Name: Jane Smith
Student ID: S102
Active Status: Active
Enrolled Courses and Grades:
- History: Not Graded
- English: Not Graded

Updating grade for Student 1:
Grade updated for course: Mathematics

Deactivating Student 1:
Student John Doe has been deactivated.
```

```
Updating grade for a non-enrolled course for Student 2:
Course not found: Mathematics

Reactivating Student 1:
Student John Doe has been reactivated.

Updated Student 1 Info:
Student Name: John Doe
Student ID: S101
Active Status: Active
Enrolled Courses and Grades:
- Mathematics: A
- Science: Not Graded
```

Conclusion:

The student management system was successfully designed and implemented. It allows administrators to view student information, update grades, manage course enrollments, and toggle active statuses. The program handles non-enrolled courses gracefully and meets all outlined requirements.