**CASE STUDY**

**UNIVERSITY MANAGEMENT SYSTEM**

Submitted by:

R. ROBLES

B.R. SOLIMAN

A. SANTOS

*Bachelor of Science in Computer Science Student*

2C

Submitted to:  
  
**Mr. Matt Wage Flordeliza***Instructor*

October 14, 2024

**Introduction**

A University Management System (UMS) is an all-in-one software solution aimed at handling the primary activities of a university, such as admission of students, maintaining faculty records, and managing courses. It facilitates student enrollment into classes, enables instructors to update lesson plans, submit and grade assignments, and evaluates students’ performance over a period of time. UMS presents a user-friendly design which enables educational institutions to make the process of administration easier, thus enhancing communication between students, lecturers and the administration. It does this by tracking students’ grades, managing their courses and updating their profiles which greatly promotes communication and transparency within the institution. This enables the normal record keeping of academic activities undertaken by students, grades assignment and monitoring of students’ performance to be less cumbersome.

This system is designed with an object-oriented programming (OOP) approach which allows room for flexibility and scaling up the system whenever there is a need. It employs the principles of inheritance, encapsulation, and polymorphism to ensure such an operation. For instance, in this system, the operations and internal structures pertaining to students, faculty members, and even courses are implemented once but can be used multiple times as needed hence improving the efficiency of the system. Keeping grades from potential unintended viewers is another aspect whereby encapsulation is important. Reliable access to this information is possible only to selected and duly authorized individuals. This system also improves system reliability especially where sensitive academic information is involved thereby enhancing system security.

**Case Analysis**

The University Management System (UMS) is a software package that is essential for educational institutions. It is created to effectively cover the management of students, faculty, and even subjects offered. UMS goes a step further by providing efficient and reliable operations such as enrolment of students, grading and management of courses offered. This is because this system was developed to solve the problems of the old administration which had in it many challenges like data corruption, too much paper work and limited interaction among clients as well as expansion problems. However, students, professors, administrative staff, information technology departments, and university authorities are some of the key parties that will ensure the successful implementation of the system. UMS offers management of students that allow students to register into courses and access the grades, manage courses where students can be added or removed with ease and a grading system where grades are given with ease but the data is protected using encapsulation. However, some issues like simply resistance to user adoption, architecture for data migration, project operation and support, and the need for tailored solutions may prevent the project from being carried out successfully. To avert these scenarios, a lot of assistance should be provided in the form of training sessions for users, changes should be made progressively in the system allowing for feedback, system support should not be discontinuous and security levels should be high. This way, it helps in maximizing operational capabilities, enhancing interaction, promoting data management, thus improving the quality of education offered to everyone. To its successful usage, what will occur is rational arrangement, education of users and help provided from time to time which in the end is a result of the success of the organization.

**Documentation**

The University Management System (UMS) was developed picturing the objectives and tangible outcomes that the University can set out to realize through teamwork and effective communication. Considering the principle of division of labor and the strengths of everyone's team members, we also understood the issues that wrangles many universities and developed a solution that optimizes such operations. This particular documentation stresses out the need for teamwork, defined responsibilities, and a cycle of development for developing an easy to use software. Teamwork being the center of our activity, we ensured that the UMS is appropriate for the use of the students and the teachers, thus enhancing their stay in the university

**Brainstorming**

On October 2, 2024, our team convened with the purpose of generating ideas for a University Management System. The first task outlined was how the system would be structured, as well as what features it would encompass based on its intended purpose. We began with the Student Class which contained relevant information such as the name, student ID, and the respective courses taken by the student. Methods such as enroll(course) and view\_grades() were included to enhance student interaction with the courses. Later, we turned our attention to the Course Class, which examined attributes such as course name, course ID, and students’ population. It was understood that methods such as add\_student(student) or remove\_student(student) would be essential for the effective management of classes by lecturers. About the Professor Class, we further discussed the benefits of deriving it from the Person class so as not to repeat ourselves. This will include professor’s ID and subjects that specific professor is responsible for, as well as functions like assigning a grade to a student and teaching a course to manage the workloads of the faculty. In the end, we thought about creating a Messenger chat for non-formal communication and updates, and conducting regular meetings every week to monitor our progress. All in all, this meeting provided a great start together with great objectives, which is the development of a University Management System that is beneficial to students, teachers as well as the University itself.

**Pictures**

FIGURE 1: During this stage, we present our suggestions. Creativity and strategy come together during brainstorming. Every thought we share, no matter how tiny, adds to the overall image as we work together.



FIGURE 2: After the ideas have been established, they need to be structured and arranged. Making sure that all required folders, structures, and basic configurations are correctly described is the focus of this stage, which is also about organization and accuracy.

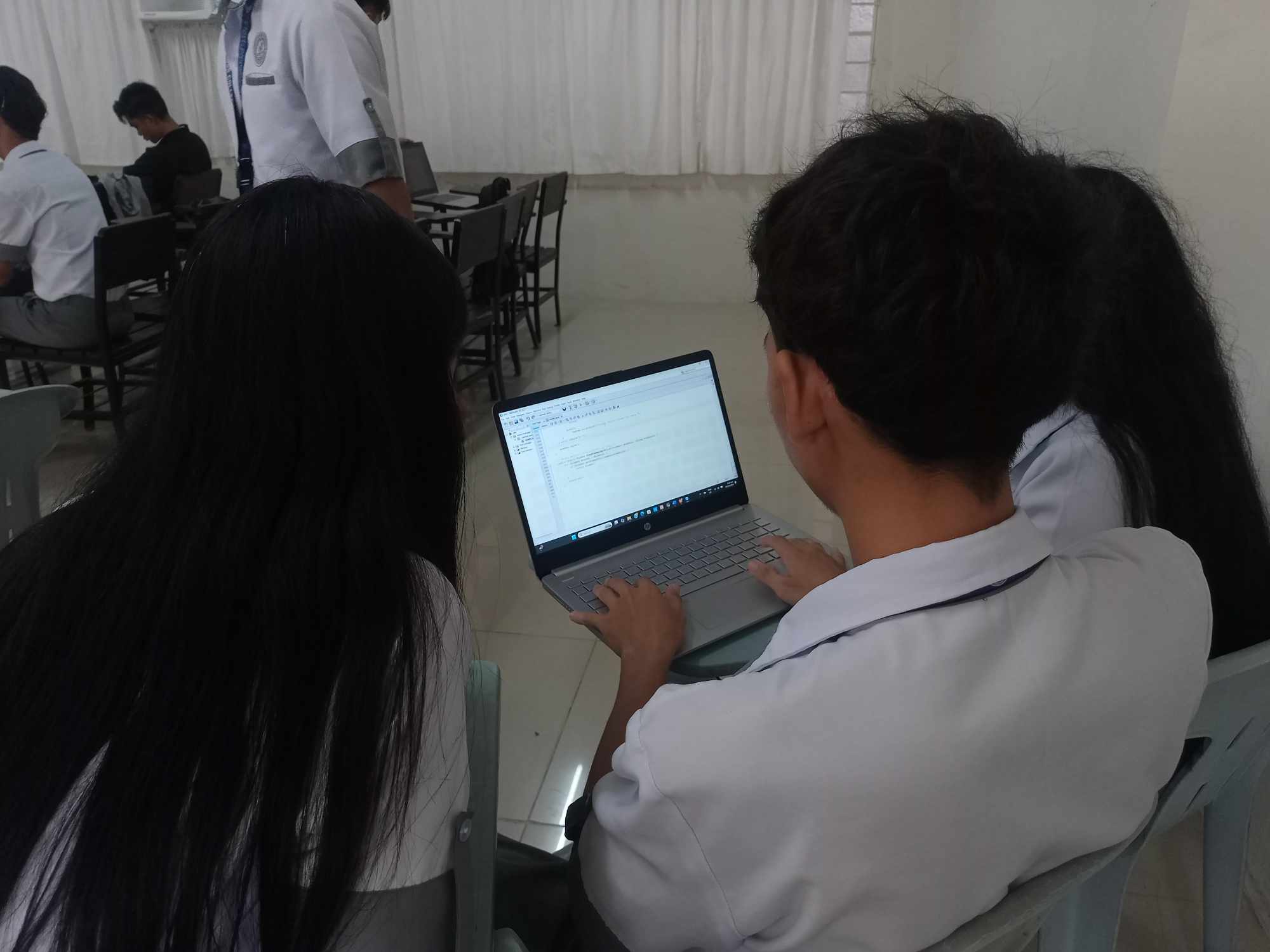


Figure 3: It's during the coding stage when our concepts become reality. Equipped with a well-defined plan, we begin developing the project's fundamental functionality, step-by-step.

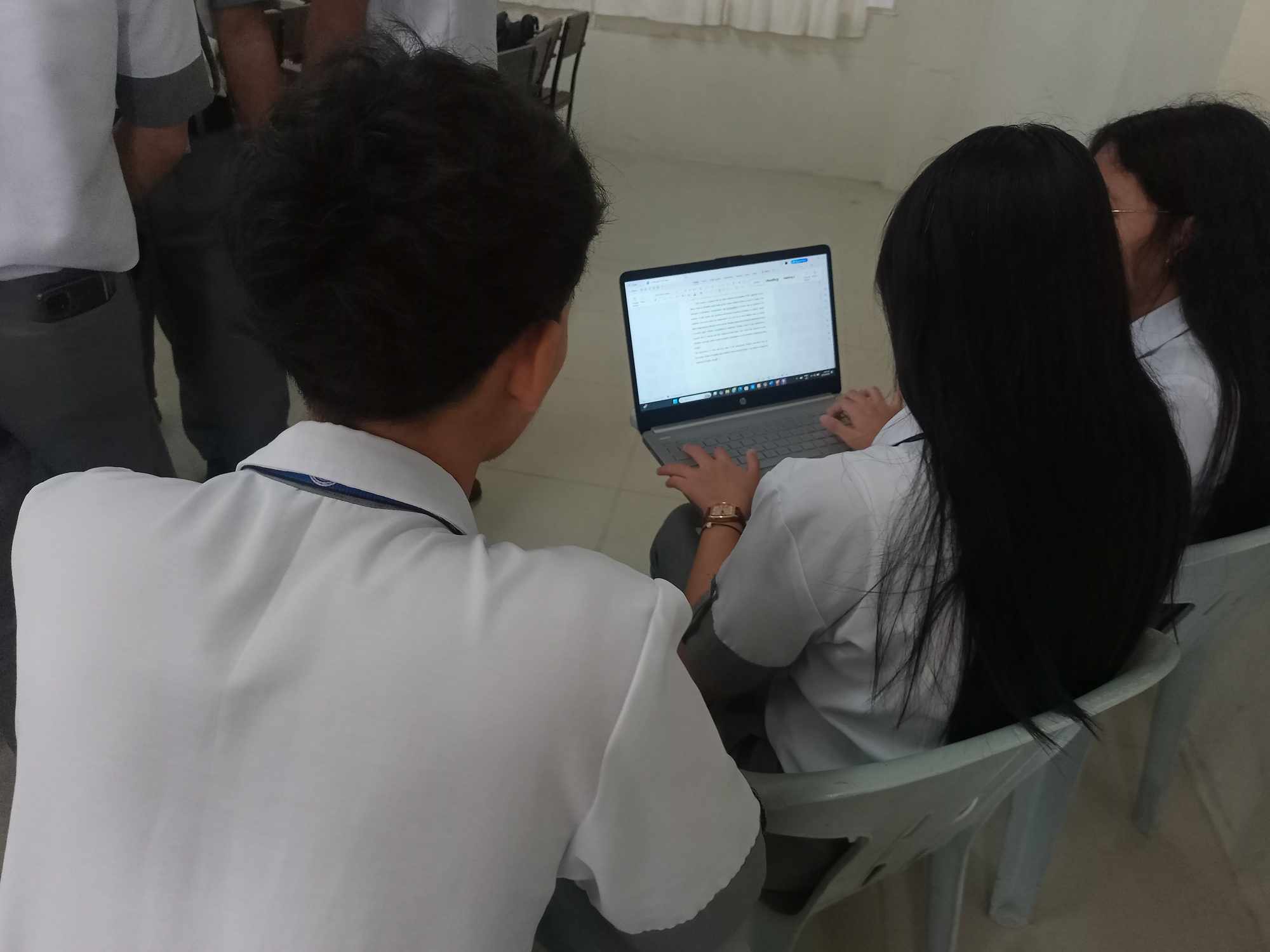


Figure 4 : During this stage, we provide thorough documentation that describes the use, organization, and goal of the code.



Figure 5: At last, we go on to the testing phase, when we confirm that every component of the project works as planned. Performance checks, debugging, and a lot of testing are required for this.



Figure 6: In closing, we take stock of the journey, evaluating the successes and identifying areas in need of development. The process is completed with an effective, thoroughly tested, and documented project.

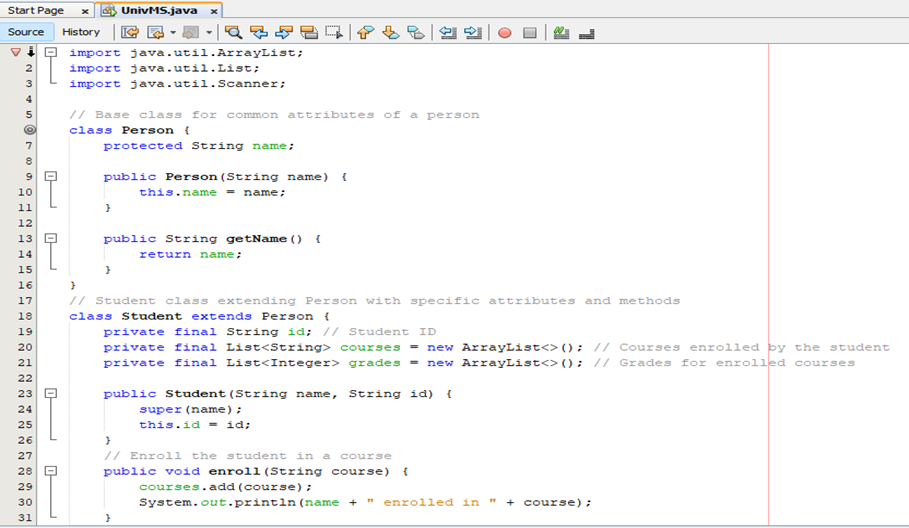
**Solution**

This approach implements a good university management system using object oriented programming concepts in Java with notable focus on encapsulation and inheritance. The primary classes such as student, course and professor are designed in a way where they hide all the attributes and functions related to them. The student class has basic attributes like name, student\_id, courses\_enrolled as well as methods that help enroll into courses and check grades. The course class contains important details about the course such as course\_name, course\_code and provides methods to add or delete students from the course when needed. The professor class has been made in such a way that it can inherit a common ancestor class which possesses some common attributes of staffs in the university. Therefore, the Professor class can benefit from inheritance and have access to those attributes as well as add its own special features such as grading students and supervising particular courses. Professor\_id is one of those important attributes that is used for distinguishing every other professor.

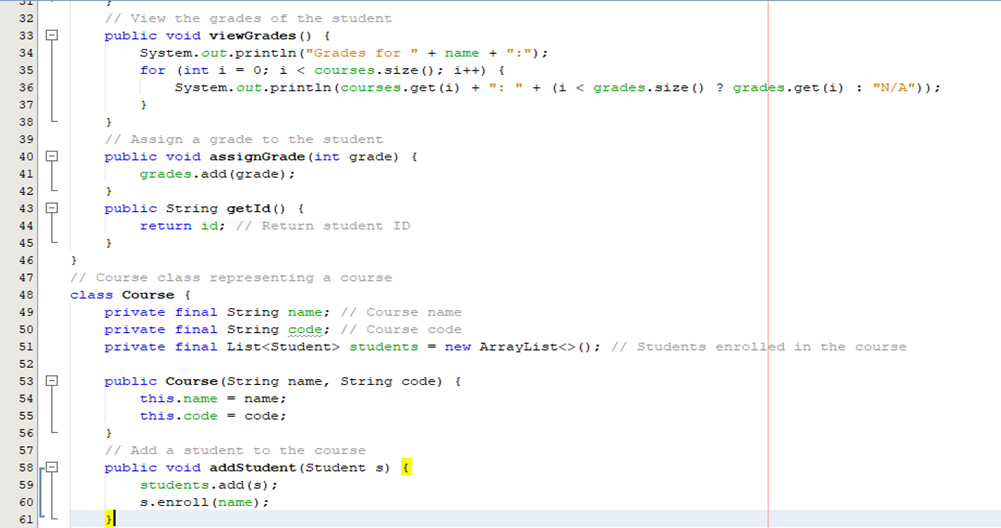
In order to boost user experience, the solution also provides for input handling through the use of the Scanner class from Java. This allows for on-the-fly creation of objects with respect to students, courses and professors, hence making the system engaging. For instance, when the system is creating a professor object, it will ask for the name and the professor\_id from the user. The constructor of the Professor class uses invokes the constructor of the base class to initialize all the inherited properties. In conclusion, such approaches result in an effective as well as a flexible and user-centric University management system.

**Screenshot of Input and Output**

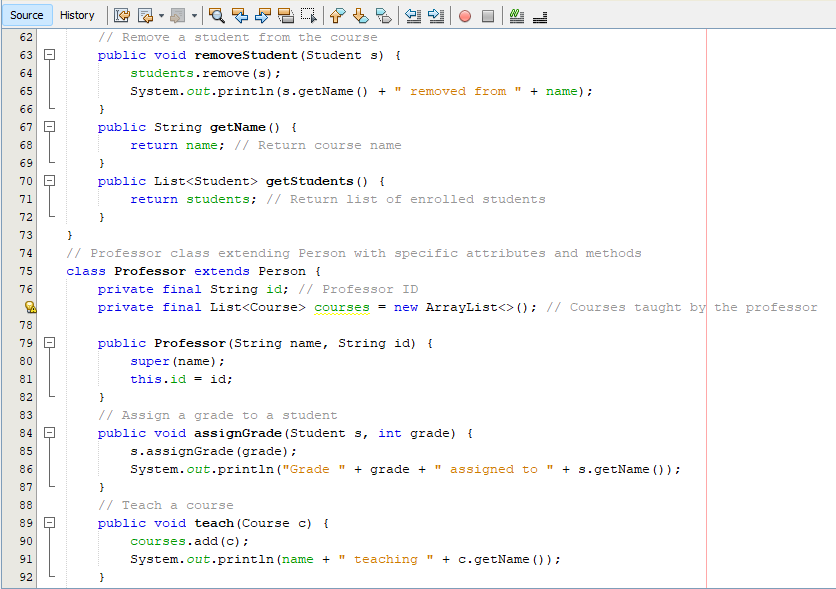
**INPUT**

****

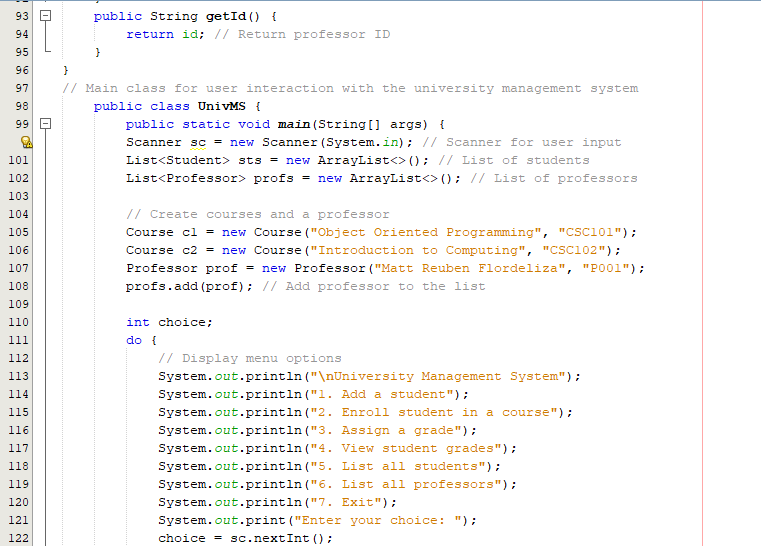
*In this input, The Person class is the parent (or base) class in the system. It stores general information that applies to any person, whether they are a professor, student, or other staff members. The Student class will represent a student in the system. Each student can be enrolled in multiple courses, and they have grades associated with each course.*

****

*In this input, the Course class represents a university course. It has attributes to store course details like the name and code.*

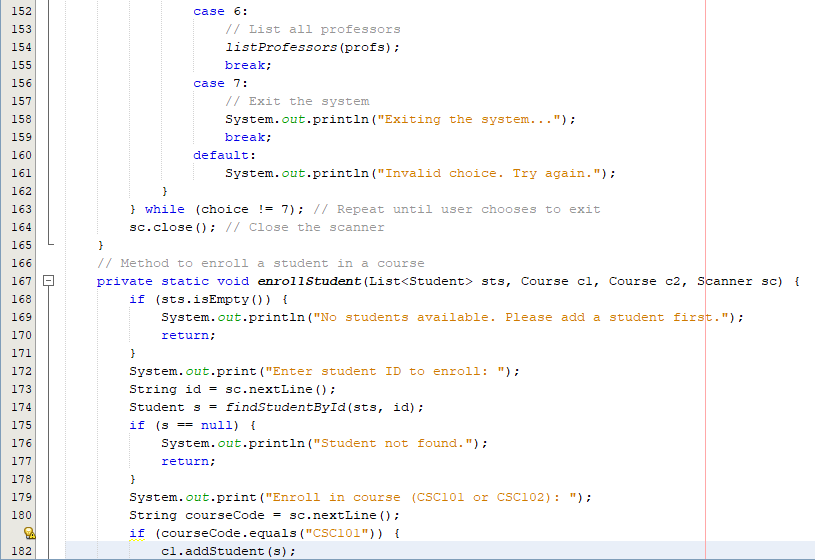
****

*In this input, the Professor class extends the Person class. It inherits the attributes and methods from Person but also adds additional features specific to a professor, like professorId and the ability to teach courses.*

****

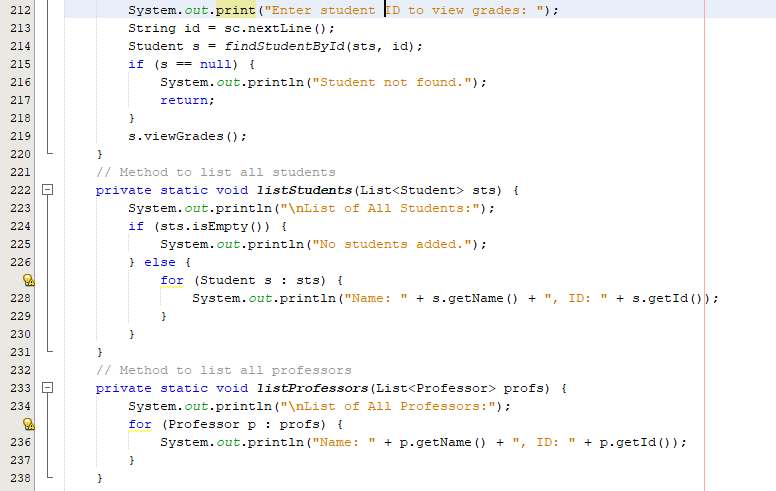
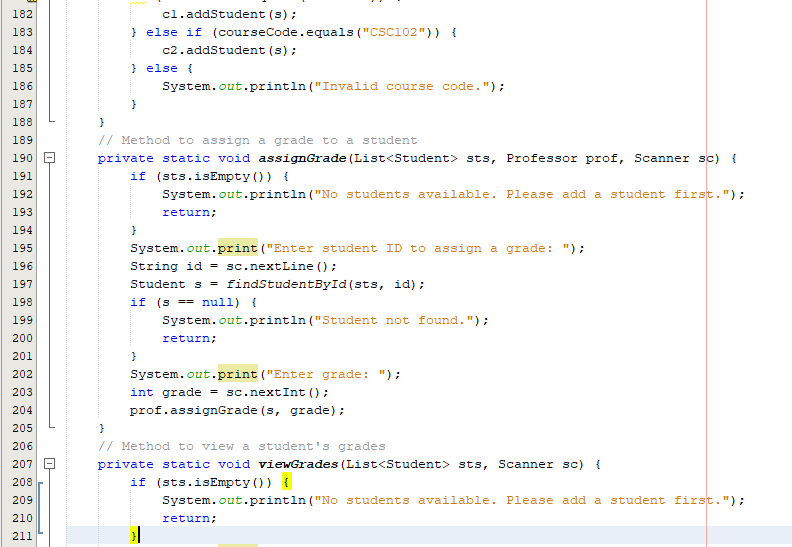
*In this input, The UnivMS class is a simple program that helps users interact with a university management system. When it starts, it creates a list for students and another for professors. It also sets up two courses and a professor.*

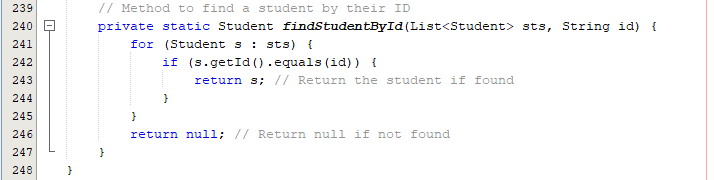
****

****

*In this input, if a user wants to add a student, they will be prompted to enter the student’s name and ID, which will then be added to the list. Similarly, if they want to enroll a student in a course, they need to provide the student’s ID and the course code. If they want to assign a grade or view*

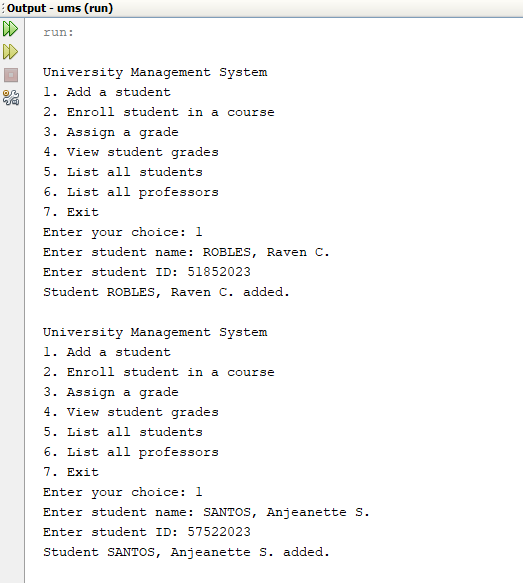
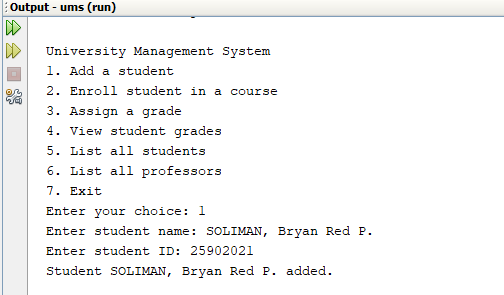
*grades, they follow a similar process.*

****

****

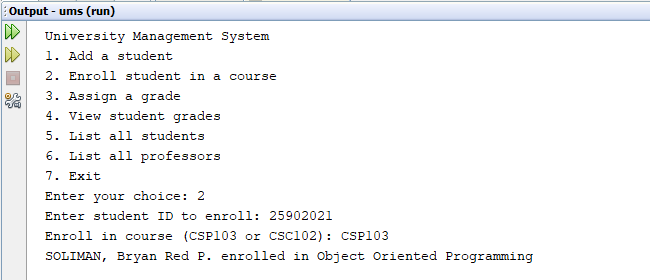
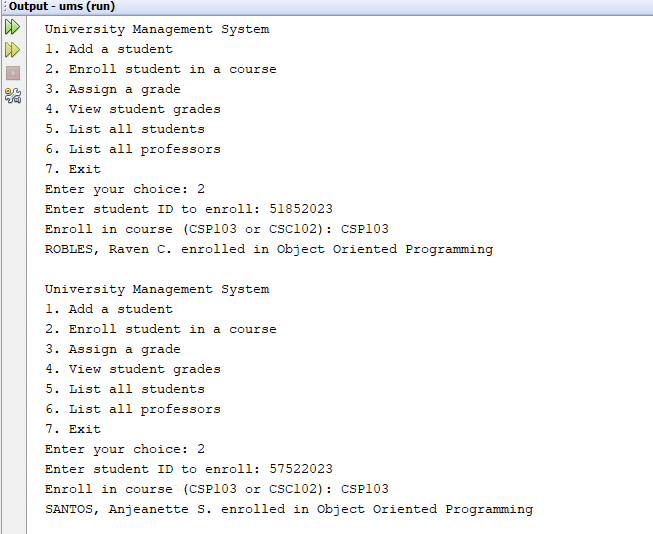
*The program also allows users to see a list of all students and professors. If a user tries to perform an action without any students in the system, they will receive a friendly reminder to add a student first. The program runs in a loop, letting users continue interacting until they decide to exit by choosing the exit option.*

**OUTPUT**

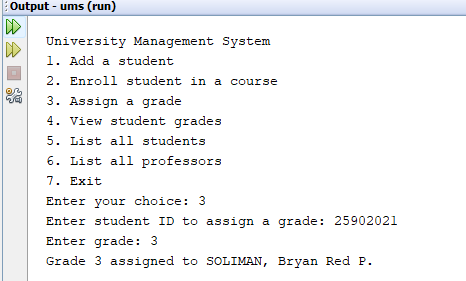
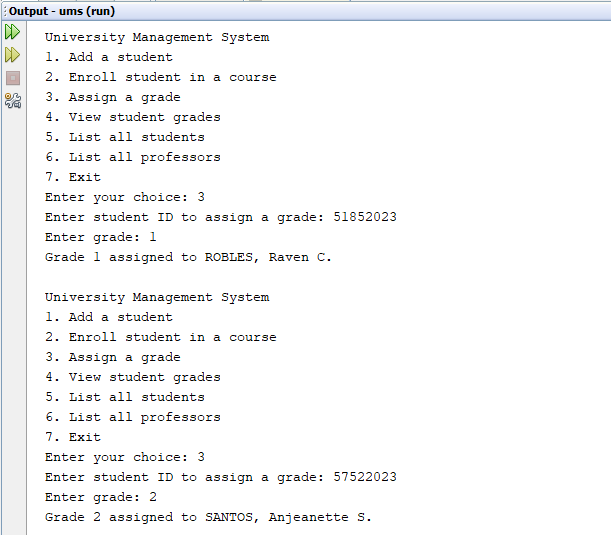


***Entering the Student's Name and Student ID***

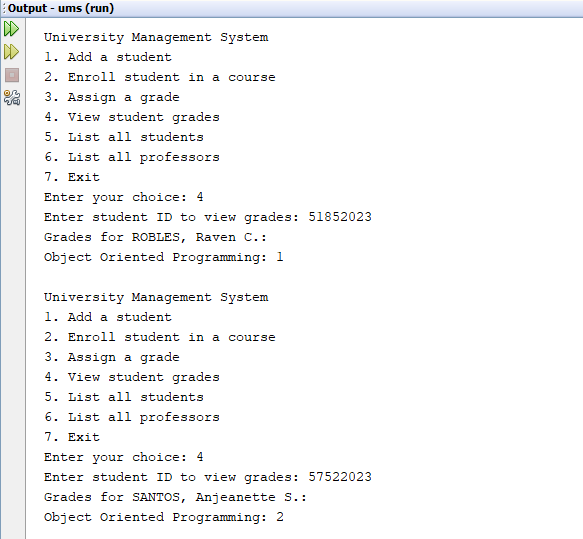
The University Management System lets users easily manage student information. When adding a student, the program prompts for their name and unique ID. For instance, if the user inputs "ROBLES, Raven C." with the ID "51852023," this information is stored in a Student object. The process can be repeated for other students, like "SANTOS, Anjeanette S." (ID "57522023") and "SOLIMAN, Bryan Red S." (ID "25902021"). This way, the system efficiently tracks all student names and IDs for easy management.

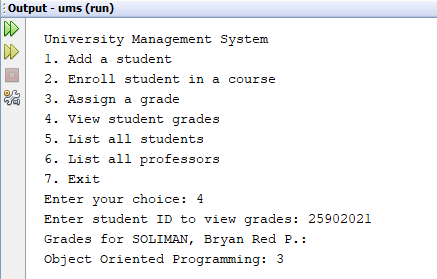
***Enrolling the Student in a Course***

In the University Management System, after adding students, users can enroll them in courses. For instance, when enrolling Raven with ID "51852023," the user selects "CSP103," and the system confirms, "ROBLES, Raven C. enrolled in Object Oriented Programming." This process continues for other students, like "SANTOS, Anjeanette S." (ID "57522023") and "SOLIMAN, Bryan Red P." (ID "25902021"), who are also enrolled in the same course. Each confirmation message indicates successful enrollment, with the program using an enroll() method to update the student's course records efficiently.

***Assigning a Grade to the Student***

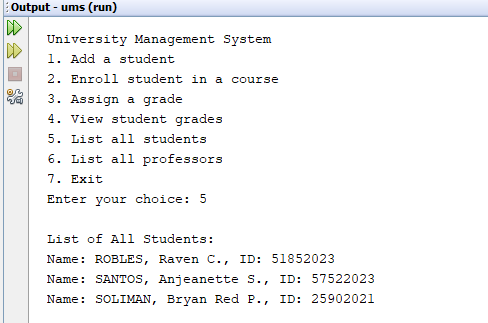
In the University Management System, users can assign grades to students after they’ve been added and enrolled. When assigning a grade, the user inputs the student’s ID and the desired grade. For instance, entering "51852023" for Raven results in "Grade 1 assigned to ROBLES, Raven C." This continues for other students: for "SANTOS, Anjeanette S." (ID "57522023"), the user assigns "Grade 2," and for "SOLIMAN, Bryan Red P." (ID "25902021"), they assign "Grade 3." Each confirmation message ensures the grades are recorded accurately.

****

****

***Viewing Student’s Grade***

In the University Management System, users can view student grades after assigning them. When a user selects the option to view grades, they enter the student's ID. For example, entering "51852023" for Raven displays "Grades for ROBLES, Raven C.: Object Oriented Programming: 1." This process is repeated for other students. For "SANTOS, Anjeanette S." (ID "57522023"), the output shows "Grades for SANTOS, Anjeanette S.: Object Oriented Programming: 2." Similarly, for "SOLIMAN, Bryan Red P." (ID "25902021"), it confirms "Grades for SOLIMAN, Bryan Red P.: Object Oriented Programming: 3." Each response clearly presents the assigned grades for easy reference.

****

***Displaying the List of Students***

In the University Management System, users can list all students by selecting the appropriate option. When the user chooses to view the student list, the system displays the following:

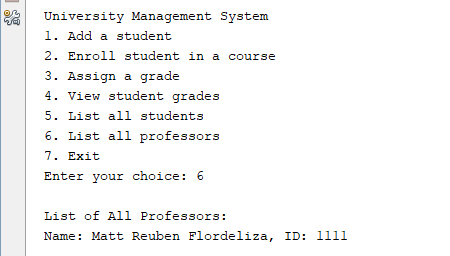
**List of All Students:**

**Name: ROBLES, Raven C., ID: 51852023**

**Name: SANTOS, Anjeanette S., ID: 57522023**

**Name: SOLIMAN, Bryan Red P., ID: 25902021**

This clear presentation allows users to see all registered students along with their IDs, making it easy to manage and reference their information.

****

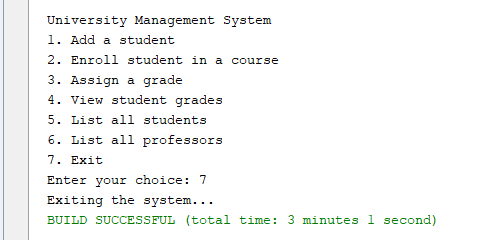
***Displaying the List of Professors***

In the University Management System, users can view the list of professors by selecting the corresponding option. When the user chooses to see the professors, the system displays:

List of All Professors:

Name: Matt Reuben Flordeliza, ID: 1111

This simple display lets users easily access the information about the registered professor along with their ID.

****

***Exiting the Program***

In the University Management System, users can exit the program by selecting the appropriate option. When the user chooses to exit, the system responds with:

**Exiting the system...**

This message confirms that the user has successfully logged out of the system, ending their session.

**Conclusion**

To sum up, the University Management System (UMS) helps to solve the problem of the majority of the universities- managing students, professors and courses. This follows a design which consists of three main classes, namely: Student, Course and Professor, enabling easy enrollment, course grading and management. The Student Class provides an option for easy enrollment and grade viewing which boosts user participation and responsibility. The Course Class takes care of course particulars such that students can easily enroll or drop classes where applicable. On the other hand, the Professor Class aids in the management of academic staff through features for giving grades and teaching subjects while utilizing the properties of a common parent class in order to cut down on code. The UMS development uses object-oriented programming methodology in java by making it possible to encapsulate any sensitive information and to apply inheritance for easier coding thus the system is adjustable as well as extensible. The use of the Scanner class also makes the system interactive as it enables students and professor objects to be created at runtime. The UMS, in addition, satisfies the operational requirements of the university at present and in addition provides the possibility for improvement in the future. This UMS, in any case, is a powerful instrument for streamlining routines, improving interactions and enhancing academic environment for all the university’s students as well as staff members.

**Reference**

<https://camudigitalcampus.com/guide/university-management-system>

<https://www.youtube.com/watch?v=9UYUjJI1dRU&list=PL_6klLfS1WqH8iVglMyh7FVniAvD9Ar-E>

<https://www.w3schools.com/java/java_oop.asp>

**Source Code**

import java.util.ArrayList;

import java.util.List;

import java.util.Scanner;

// Base class for common attributes of a person

class Person {

protected String name;

public Person(String name) {

this.name = name;

}

public String getName() {

return name;

}

}

// Student class extending Person with specific attributes and methods

class Student extends Person {

private final String id; // Student ID

private final List<String> courses = new ArrayList<>(); // Courses enrolled by the student

private final List<Integer> grades = new ArrayList<>(); // Grades for enrolled courses

public Student(String name, String id) {

super(name);

this.id = id;

}

// Enroll the student in a course

public void enroll(String course) {

courses.add(course);

System.out.println(name + " enrolled in " + course);

}

// View the grades of the student

public void viewGrades() {

System.out.println("Grades for " + name + ":");

for (int i = 0; i < courses.size(); i++) {

System.out.println(courses.get(i) + ": " + (i < grades.size() ? grades.get(i) : "N/A"));

}

}

// Assign a grade to the student

public void assignGrade(int grade) {

grades.add(grade);

}

public String getId() {

return id; // Return student ID

}

}

// Course class representing a course

class Course {

private final String name; // Course name

private final String code; // Course code

private final List<Student> students = new ArrayList<>(); // Students enrolled in the course

public Course(String name, String code) {

this.name = name;

this.code = code;

}

// Add a student to the course

public void addStudent(Student s) {

students.add(s);

s.enroll(name);

}

// Remove a student from the course

public void removeStudent(Student s) {

students.remove(s);

System.out.println(s.getName() + " removed from " + name);

}

public String getName() {

return name; // Return course name

}

public List<Student> getStudents() {

return students; // Return list of enrolled students

}

}

// Professor class extending Person with specific attributes and methods

class Professor extends Person {

private final String id; // Professor ID

private final List<Course> courses = new ArrayList<>(); // Courses taught by the professor

public Professor(String name, String id) {

super(name);

this.id = id;

}

// Assign a grade to a student

public void assignGrade(Student s, int grade) {

s.assignGrade(grade);

System.out.println("Grade " + grade + " assigned to " + s.getName());

}

// Teach a course

public void teach(Course c) {

courses.add(c);

System.out.println(name + " teaching " + c.getName());

}

public String getId() {

return id; // Return professor ID

}

}

// Main class for user interaction with the university management system

public class UnivMS {

public static void main(String[] args) {

Scanner sc = new Scanner(System.in); // Scanner for user input

List<Student> sts = new ArrayList<>(); // List of students

List<Professor> profs = new ArrayList<>(); // List of professors

// Create courses and a professor

Course c1 = new Course("Object Oriented Programming", "CSP103");

Course c2 = new Course("Introduction to Computing", "CSC102");

Professor prof = new Professor("Matt Reuben Flordeliza", "1111");

profs.add(prof); // Add professor to the list

int choice;

do {

// Display menu options

System.out.println("\nUniversity Management System");

System.out.println("1. Add a student");

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

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

System.out.println("4. View student grades");

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

System.out.println("6. List all professors");

System.out.println("7. Exit");

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

choice = sc.nextInt();

sc.nextLine(); // Consume newline

// Handle user choices

switch (choice) {

case 1:

// Add a new student

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

String sName = sc.nextLine();

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

String sId = sc.nextLine();

sts.add(new Student(sName, sId));

System.out.println("Student " + sName + " added.");

break;

case 2:

// Enroll student in a course

enrollStudent(sts, c1, c2, sc);

break;

case 3:

// Assign a grade to a student

assignGrade(sts, prof, sc);

break;

case 4:

// View grades of a student

viewGrades(sts, sc);

break;

case 5:

// List all students

listStudents(sts);

break;

case 6:

// List all professors

listProfessors(profs);

break;

case 7:

// Exit the system

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

break;

default:

System.out.println("Invalid choice. Try again.");

}

} while (choice != 7); // Repeat until user chooses to exit

sc.close(); // Close the scanner

}

// Method to enroll a student in a course

private static void enrollStudent(List<Student> sts, Course c1, Course c2, Scanner sc) {

if (sts.isEmpty()) {

System.out.println("No students available. Please add a student first.");

return;

}

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

String id = sc.nextLine();

Student s = findStudentById(sts, id);

if (s == null) {

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

return;

}

System.out.print("Enroll in course (CSP103 or CSC102): ");

String courseCode = sc.nextLine();

if (courseCode.equals("CSP103")) {

c1.addStudent(s);

} else if (courseCode.equals("CSC102")) {

c2.addStudent(s);

} else {

System.out.println("Invalid course code.");

}

}

// Method to assign a grade to a student

private static void assignGrade(List<Student> sts, Professor prof, Scanner sc) {

if (sts.isEmpty()) {

System.out.println("No students available. Please add a student first.");

return;

}

System.out.print("Enter student ID to assign a grade: ");

String id = sc.nextLine();

Student s = findStudentById(sts, id);

if (s == null) {

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

return;

}

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

int grade = sc.nextInt();

prof.assignGrade(s, grade);

}

// Method to view a student's grades

private static void viewGrades(List<Student> sts, Scanner sc) {

if (sts.isEmpty()) {

System.out.println("No students available. Please add a student first.");

return;

}

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

String id = sc.nextLine();

Student s = findStudentById(sts, id);

if (s == null) {

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

return;

}

s.viewGrades();

}

// Method to list all students

private static void listStudents(List<Student> sts) {

System.out.println("\nList of All Students:");

if (sts.isEmpty()) {

System.out.println("No students added.");

} else {

for (Student s : sts) {

System.out.println("Name: " + s.getName() + ", ID: " + s.getId());

}

}

}

// Method to list all professors

private static void listProfessors(List<Professor> profs) {

System.out.println("\nList of All Professors:");

for (Professor p : profs) {

System.out.println("Name: " + p.getName() + ", ID: " + p.getId());

}

}

// Method to find a student by their ID

private static Student findStudentById(List<Student> sts, String id) {

for (Student s : sts) {

if (s.getId().equals(id)) {

return s; // Return the student if found

}

}

return null; // Return null if not found

}

}