FCDS
Programming II

University Management System
(Alexandria University)

Project Overview

Design and implement a University Management System for Alexandria University that simulates core university operations. This project will allow you to apply Object-Oriented Programming concepts learned in your Java courses to create a practical management system for students, courses, faculty, and administrative functions.

System Users

- **Students**: Registered students who can manage their academic activities
- **Faculty**: Professors and teaching assistants who manage courses and grades
- **Admin Staff**: Administrative personnel who handle university operations
- **System Administrator**: Technical administrator with full system access

Core Features

1. User Authentication and Profile Management

Features:

- Login system with username and password
- Different interfaces based on user role
- Basic profile management for all users
- Password reset functionality

Requirements:

- Passwords must contain at least 6 characters
- Store user credentials securely in files
- Display appropriate error messages for login failures
- Allow users to update basic profile information

Sample Scenario:

- A student enters ID "S2023001" and password "student123"
- System validates credentials against stored data
- Upon success, system displays the student dashboard
- If unsuccessful, system shows an error message

2. Student Management Features:

- Student registration and enrollment
- Course registration and withdrawal
- View grades and GPA calculation
- Academic record management

Student Data:

- Student ID, name, contact information, admission date
- Enrolled courses and grades
- Attendance records
- Academic status (Active, On Probation, Graduated)

Sample Scenario:

- Student selects "Register for Courses" option
- System displays available courses for the current semester

- Student selects courses to register (up to maximum allowed credits)
- System validates prerequisites and scheduling conflicts
- System confirms registration and displays updated schedule

3. Course Management Features:

- Course creation and modification
- Course scheduling
- Assignment and grade management
- Course materials distribution

Course Data:

- Course ID, title, description, credit hours
- Prerequisites
- Faculty assigned
- Maximum enrollment capacity
- Schedule (days, times, location)

Sample Scenario:

- Faculty member selects "Manage Course" option
- Faculty chooses a course they teach
- System displays course details and roster of enrolled students
- Faculty can add assignments, enter grades, or update course information
- System saves changes and notifies affected students if necessary

4. Faculty and Department Management Features:

- Faculty profile management
- Department structure
- Course assignment to faculty
- Faculty performance tracking

Faculty Data:

- Faculty ID, name, contact information, department
- Expertise and qualifications
- Courses taught
- Office hours and location

Sample Scenario:

- Admin selects "Assign Faculty to Course" option
- System displays list of courses and available faculty
- Admin selects course and assigns appropriate faculty
- System validates faculty workload and scheduling
- System confirms assignment and notifies faculty member

Object-Oriented Design Requirements

The system must be designed following these object-oriented principles:

1. Inheritance

- **Implementation Requirements**:
  - Create a base User class with common user attributes
  - Extend User with Student, Faculty, AdminStaff, and SystemAdmin subclasses
  - Implement a Course hierarchy for different course types (e.g., Lecture, Lab, Seminar)
  - Create a base Department class that can be extended for specific departments

- **Example**:
  - User defines common login behavior and profile management
  - Student extends User with student-specific attributes and methods
  - Faculty extends User with teaching-related capabilities
  - Different course types implement specialized behavior for attendance, grading, etc.

2. Encapsulation

- **Implementation Requirements**:
  - Make all class attributes private
  - Provide public getter and setter methods with appropriate validation
  - Hide implementation details from other classes
  - Create proper constructors for object initialization
- **Example**:
  - Student grades are private, only accessible through approved methods
  - Course enrollment checks capacity limits internally
  - Faculty workload is calculated through encapsulated methods
  - Student GPA calculation algorithm is hidden within the Student class

3. Polymorphism

- **Implementation Requirements**:
  - Create methods with the same name but different implementations in subclasses
  - Use parent class references to work with different object types
  - Implement interface methods differently based on concrete classes
- **Example**:
  - calculatePayroll() works differently for Faculty vs. AdminStaff
  - generateReport() produces different outputs for various user types
  - Course grading behavior varies by course type through method overriding

4. Abstraction

- **Implementation Requirements**:
  - Create abstract classes for concepts that shouldn't be instantiated directly
  - Define interfaces for common behaviors
  - Hide complex implementation details behind simple method calls
- **Example**:
  - User as an abstract class
  - GradingSystem interface implemented by different grading strategies ○ Complex registration rules hidden behind a simple registerForCourse() method

5. Class Relationships

- **Implementation Requirements**:
  - Use composition to establish "has-a" relationships ○ Create proper associations between related classes
  - Implement aggregation for collecting related objects
- **Example**:
  - A Student has multiple CourseEnrollment objects ○ A Department has multiple Faculty members
  - A Course has a Faculty instructor and multiple Student enrollments ○ The University class serves as the central system coordinating all entities

Technical Requirements

Data Storage

File Requirements:

- Create separate text files for:
  - Users (users.txt) - store user credentials and roles
  - Students (students.txt) - store student details
  - Courses (courses.txt) - store course information
  - Enrollments (enrollments.txt) - store registration data
  - Departments (departments.txt) - store department information
- Use consistent formatting for easy reading/writing
- Implement proper file handling with try-catch blocks

UML Diagrams Class Diagram:

- Show all classes with attributes and methods
- Include inheritance relationships
- Show composition/aggregation relationships
- Display multiplicity on relationships

User Interface

- Create a console-based menu system
- Display different menus based on user role
- Provide clear feedback messages
- Format output for readability (tables, headers, etc.)

Bonus Features

1. Simple GUI

- Create a basic graphical interface using Java Swing
- Implement forms for student registration and course management
- Display academic information in organized panels
- Include navigation menus and dashboard views

2. Database Storage

- Replace file storage with a simple database (SQLite recommended)
- Create appropriate tables for all entities
- Implement proper data relationships
- Use prepared statements for database operations

Key Classes and Their Responsibilities 1. User (Abstract Class)

- **Responsibility**: Base class for all system users
- **Key Attributes**: userId, username, password, name, email, contactInfo
- **Key Methods**: login(), logout(), updateProfile()

2. Student (Extends User)

- **Responsibility**: Manages student academic information
- **Key Attributes**: studentId, admissionDate, academicStatus, enrolledCourses
- **Key Methods**: registerForCourse(), dropCourse(), viewGrades(), calculateGPA()

3. Faculty (Extends User)
- **Responsibility**: Manages teaching and course administration
- **Key Attributes**: facultyId, department, expertise, coursesTeaching
- **Key Methods**: assignGrades(), manageCourse(), setOfficeHours(), viewStudentRoster()

4. AdminStaff (Extends User)
- **Responsibility**: Handles administrative operations
- **Key Attributes**: staffId, department, role
- **Key Methods**: registerStudent(), createCourse(), assignFaculty(), generateReports()

5. SystemAdmin (Extends User)
- **Responsibility**: Manages system settings and user accounts
- **Key Attributes**: adminId, securityLevel
- **Key Methods**: createUser(), modifySystemSettings(), backupData(), managePermissions()

6. Course
- **Responsibility**: Stores course information and manages enrollment
- **Key Attributes**: courseId, title, description, creditHours, prerequisites, schedule, instructor, enrolledStudents
- **Key Methods**: addStudent(), removeStudent(), isPrerequisiteSatisfied(), getAvailableSeats()

7. Department
- **Responsibility**: Manages faculty and courses within a department
- **Key Attributes**: departmentId, name, faculty, offeredCourses
- **Key Methods**: addFaculty(), removeFaculty(), addCourse(), getFacultyList()

8. Enrollment
- **Responsibility**: Tracks student enrollment in courses
- **Key Attributes**: student, course, enrollmentDate, grade, status
- **Key Methods**: assignGrade(), getStatus(), withdraw()

9. University
- **Responsibility**: Central management of university operations
- **Key Attributes**: departments, users, courses, academicCalendar
- **Key Methods**: registerStudent(), hireFaculty(), createDepartment(), offerCourse()

10. FileManager
- **Responsibility**: Manages data persistence using files
- **Key Methods**: saveUsers(), loadUsers(), saveCourses(), loadCourses()