Online Course Registration System

1. Project Overview

Students are required to work on a structured software project that applies the core principles of object-oriented analysis, modeling, and design.

In this project, each team will develop a comprehensive analysis and design model for an **Online Course Registration System**.

The project will simulate the first stages of software development — from problem definition and requirements gathering to system modeling and design — but will not include implementation.

The outcome will be a set of professional documents and UML diagrams that reflect the complete understanding of the system's structure and behavior, as would be delivered to a software development team for implementation.

Each phase of the project builds upon the previous one, allowing teams to gradually refine their analysis, design, and implementation into a coherent and maintainable software solution.

2. Project Objective

By completing this project, students will be able to:

- Apply object-oriented principles (encapsulation, inheritance, polymorphism, abstraction) to analyze and design systems.
- Identify system actors, use cases, and relationships.
- Use UML modeling techniques to represent system structure and behavior.
- Transform analysis results into design-level specifications.
- Demonstrate good team collaboration, documentation, and version control practices.

3. System Description

The Online Course Registration System aims to facilitate course management and enrollment processes for students, instructors, and administrators within a university or educational platform

System Overview:

- Students can browse available courses, view schedules, and register for courses within allowed limits.
- Instructors can manage course details, view registered students, and approve special requests.
- Administrators can manage course offerings, academic terms, user accounts, and registration rules.
- The system should validate registration constraints (e.g., prerequisites, course capacity, schedule conflicts).
- Notifications or confirmations should be generated for successful or failed registrations.

4. Project Phases and Deliverables

The project will be divided into **four main phases**, each focusing on a key stage of system development.

Each phase builds on the previous one, culminating in a comprehensive System Design Report.

Phase 1 – Problem Definition & Requirements Analysis

Objective:

Understand the problem domain and specify what the system should accomplish.

Tasks:

1. Define the problem context and the goals of the Online Course Registration System.
2. Identify key users (actors) such as:
    - Student
    - Instructor
    - Administrator
3. Gather and classify requirements:
    - **Functional Requirements:** registering courses, viewing schedules, managing courses, etc.
    - **Non-functional Requirements**: usability, performance, reliability, security.
4. Identify system constraints and assumptions.
5. List the main use cases and describe each briefly.
6. Draw the **Use Case Diagram** showing actors and their interactions.

Deliverables:

Requirements Specification Document including:

- System overview and objectives
- Actors and their roles
- Functional and non-functional requirements
- System constraints and assumptions
- Use case list and descriptions
- UML **Use Case Diagram**

Expected Output:

A concise, well-structured report (PDF or DOCX) with the system's problem analysis and a clear use case diagram.

Phase 2 – System Analysis & Modeling

Objective:

Model how the system behaves and identify its internal structure.

Tasks:

1. Develop **detailed use case scenarios** (steps, preconditions, postconditions).
2. Identify conceptual classes (e.g., *Student*, *Course*, *Registration*, *Instructor*, *Semester*, *Schedule*).
3. Define relationships between classes (association, inheritance, aggregation, etc.).
4. Draw:
   - **Class Diagram (Analysis Level)** showing conceptual classes and relationships.
   - **Sequence Diagrams** for main use cases, such as "Register for Course" or "Add New Course."
   - **Activity Diagrams** to model workflows (e.g., "Course Registration Process").

Deliverables:

Analysis Model Report, including:

- Expanded use case descriptions
- Analysis-level **Class Diagram**
- **Sequence Diagrams** for at least 2–3 core use cases
- **Activity Diagrams** for main workflows
- Explanatory notes on modeling decisions

Expected Output:

A cohesive document containing UML diagrams that correctly represent the system's analysis model.

Phase 3 – System Design

Objective:

Translate the analysis model into a detailed object-oriented design ready for implementation.

Tasks:

1. Refine the **Class Diagram** into a **Design Class Diagram**:
   - Add attributes, data types, and methods for each class.
   - Specify relationships, multiplicities, and navigability.
2. Organize the system into packages or modules (e.g., *User Management*, *Course Management*, *Registration Management*).
3. Design **Component or Package Diagrams** to illustrate modular structure.
4. For objects with dynamic states (e.g., *Registration*), develop **State Machine Diagrams**.
5. (Optional) Create **UI mockups or wireframes** for main system screens (Student Dashboard, Course List, Registration Form, etc.).

Deliverables:

System Design Document, including:

- Refined **Design Class Diagram** (detailed)
- **Component or Package Diagram** showing modules
- **State Machine Diagram(s)** for key entities
- (Optional) **UI Mockups or Navigation Flow**
- Summary of design approach and justification of key design decisions.

Expected Output:

A detailed, logically consistent design report that clearly connects analysis models to design artifacts.

Phase 4 – Implementation

Objective: Translate the system design into an initial working software prototype that demonstrates the core logic of the Online Course Registration System.

Tasks:

1. Set up the Development Environment
   - Use Java (as instructed).
   - Follow object-oriented principles and ensure clear separation of responsibilities.
   - Initialize a GitHub repository for your team project.
2. Implement Core Classes
   - Implement the main entities defined in the design model:

     Example: Student, Instructor, Course, Registration, Semester, Schedule.
   - Apply inheritance where applicable (e.g., User → Student, Instructor, Administrator).
   - Implement relationships and associations using attributes or references (e.g., List<Course> inside Student).
3. Develop Core Functionalities
   - Student functions:
     i) Browse and list available courses.
     ii) Register for a course (validate prerequisites, check capacity, detect time conflicts).
     iii) View registered courses.
   - Instructor functions:
     i) Approve or reject student registration requests.
     ii) View list of students in each course.
   - Administrator functions:
     i) Add/edit/remove courses and users.
     ii) Manage registration rules (max load, schedule limits, etc.).
4. Simulate User Interaction
   - Build a console-based interface (menu-driven system) to simulate user actions.
5. Documentation
   - Add inline code comments and a short README.md explaining:
     i) System overview
     ii) Setup instructions
     iii) Team members and their contributions

Deliverables:

- Implementation Report (Phase 4) including:
    - Overview of implemented features
    - Screenshots of console output (e.g., registration process)
    - Explanation of how the code maps to design diagrams
    - Class structure summary
- Code added to the GitHub Repositories and its link is added to the report
- Updated Final Consolidated Report (including Phases 1–4)

Expected Output:

A minimal yet functional console-based system demonstrating the core logic of course registration and management, aligned with the previous design and UML diagrams.

5. Final Submission Package

At the end of Phase 4, each team must submit a **Final Consolidated Report** that includes:

- All three phases (Requirements → Analysis → Design).
- All UML diagrams (numbered, labeled, and captioned).
- A professional cover page with project title, team members, section, and course name.
- Submission in **PDF format**, plus the source UML files (.mdj, .vpp, .drawio, etc.).
- GitHub repository containing all deliverables.

6. Submission Guidelines

- All documents must be prepared digitally — **no handwritten submissions**.
- UML diagrams must be drawn using professional tools: **StarUML**, **Visual Paradigm**, **Lucidchart**, or **draw.io**.
- Each diagram must include a **title**, **legend (if needed)**, and **team identifier**.
- Each phase has a submission deadline (to be announced in class).
- Late submissions without prior approval may result in grade deductions.
- Teams should ensure consistent naming across all diagrams and documents.

7. Notes and Recommendations

- Start early: defining use cases and actors clearly will make later phases much easier.
- Maintain **consistency between all diagrams** (e.g., class names should match across diagrams).
- Be professional: write reports in a formal tone with organized sections and proper captions.
- Use **version control (GitHub)** to track team contributions and ensure collaboration.
- Regular feedback sessions will be scheduled after each phase to guide improvements.

Guidelines for Delivering Each Phase

Phase	Deliverable Format	Main Files	Submission
Phase 1 – Requirements	PDF/DOCX	Phase1_Requirements.docx / .pdf, UseCaseDiagram.drawio	Upload and GitHub commit
Phase 2 – Analysis	PDF/DOCX	Phase2_Analysis.docx, ClassDiagram_Analysis.drawio, SequenceDiagram_RegisterCourse.drawio, ActivityDiagram_Registration.drawio	Upload and GitHub commit
Phase 3 – Design	PDF/DOCX	Phase3_Design.docx, DesignClassDiagram.drawio, ComponentDiagram.drawio, StateMachineDiagram_Registration.drawio	Upload and GitHub commit
Phase 4 – Implementation	PDF/DOCX + Source Code	Phase4_Implementation.docx, /src/ folder, README.md, sample data	Upload and GitHub commit
Final Package	Consolidated PDF + UML source + Code	FinalReport_TeamX.pdf, all .drawio/.mdj files, /src/ folder	Upload and GitHub commit

File and Folder Naming Convention

Use the following standardized structure: TeamX → X means team number found in teams sheet

```
OOAD_Project_TeamX/
│
├── Phase1_Requirements/
│   ├── Phase1_Requirements_TeamX.docx
│   ├── UseCaseDiagram_TeamX.drawio
│
├── Phase2_Analysis/
│   ├── Phase2_Analysis_TeamX.docx
│   ├── ClassDiagram_Analysis_TeamX.drawio
│   ├── SequenceDiagram_RegisterCourse_TeamX.drawio
│   └── ActivityDiagram_CourseRegistration_TeamX.drawio
│
├── Phase3_Design/
│   ├── Phase3_Design_TeamX.docx
│   ├── DesignClassDiagram_TeamX.drawio
│   ├── ComponentDiagram_TeamX.drawio
│   └── StateMachineDiagram_Registration_TeamX.drawio
│
├── Phase4_Implementation/
│   ├── Phase4_Implementation_TeamX.docx
│   ├── README.md
│   └── src/
│       ├── Main.java
│       ├── Student.java
│       ├── Course.java
│       ├── Registration.java
│       └── ...
└── FinalReport_TeamX.pdf
```