Course Management System - Deliverable 1

1. Scenario

The Course Management System (CMS) is designed for educational institutions to manage courses, students, teachers, and assignments. Administrators can create user accounts, teachers can manage courses and assignments, and students can enroll in courses, view assignments, and manage their schedules. The system maintains academic records and facilitates interactions between these user types.

---

2. Design Paradigm (Functionalities to Demonstrate)

- **User Management**: Create different types of users (Admin, Teacher, Student)
- **Course Management**: Create, view, and manage courses
- **Assignment Management**: Create and track assignments
- **Enrollment System**: Students enroll in courses
- **Schedule Management**: Students can upload/download schedules
- **Grading System**: Teachers can grade assignments

---

3. Expected Output

Users will interact with a console-based application that allows:

- Administrators to create teachers and students
- Teachers to create courses and assignments
- Students to enroll in courses and manage their schedules
- All users to view relevant academic information
- Sorting and searching capabilities for courses and assignments

---

4. Class Hierarchies

Hierarchy 1: User System

- **User class** (abstract)
    - Administrator class
    - Teacher class
    - Student class

- **AcademicItem class** (abstract)
    - Course class
    - Assignment

User **(Abstract Base Class)** methods:

- **getRole()**: String (abstract) Returns the role of the user (Admin, Teacher, or student)

Administrator class methods:

- **createTeacher**(name: String, age: int): User, creates a Teacher object, assigns a unique ID, and adds it to the system.
- **createStudent(**name: String, age: int): User, same as previous method but for a Student object.
- **getRole()**: String, Returns "Administrator". Overrides the abstract method from User.

Teacher class methods:

- **getRole()**: String, Returns "Teacher". Overrides the abstract method from User.
- **createAssignment(title: String, description: String)**: Assignment, creates a new Assignment for the teacher's course and adds it to students' task lists.

Student class methods:

- **getRole()**: String, Returns "Student". Overrides the abstract method from User.
- **uploadSchedule(path: String)**: void, Reads a CSV file from path and updates the student's schedule (list of courses). Uses **TextIO**.
- **downloadSchedule(path: String)**: void, Writes the student's schedule to a CSV file at path. Uses **TextIO**

AcademicItem (Abstract Base Class) methods:

- **getType()**: String *(abstract), Returns the type of academic item ("Course" or "Assignment"). Polymorphic implemented by subclasses.*
- **compareTo(o: AcademicItem)**: int, Compares two AcademicItems by createdDate (for sorting). Implements Comparable<AcademicItem>.

Course class methods:

- **getType()**: String, Returns "Course". Overrides the abstract method from AcademicItem.
- **enrollStudent(student: Student)**: Boolean, Adds a student to the course's students list if not already enrolled. Returns true on success.

Assignment class methods:

- **getType()**: String, Returns "Assignment". Overrides the abstract method from AcademicItem.

CourseComparator subclass method:

compare(c1: Course, c2: Course): int

Compares courses by:
•ID (default)
• Title (alphabetical)
• Enrollment count (number of students). Used for custom sorting.

---

5. Interface Specification

CreateUser Interface:

- Implements the **createTeacher** & **createStudent** abstract methods

This interface ensures consistent user creation methods are implemented by the Administrator class.

---

6. Runtime Polymorphism

- The **compareTo** method in **AcademicItem** will demonstrate runtime polymorphism when sorting different academic items
- The **createTeacher** and **createStudent** methods in the Administrator class (implementing CreateUser) will show interface method polymorphism

---

7. TextIO Usage

The **student** class will use TextIO for:

- Reading schedule files in **uploadSchedule(**String path)
- Writing schedule files in **downloadSchedule**(String path)

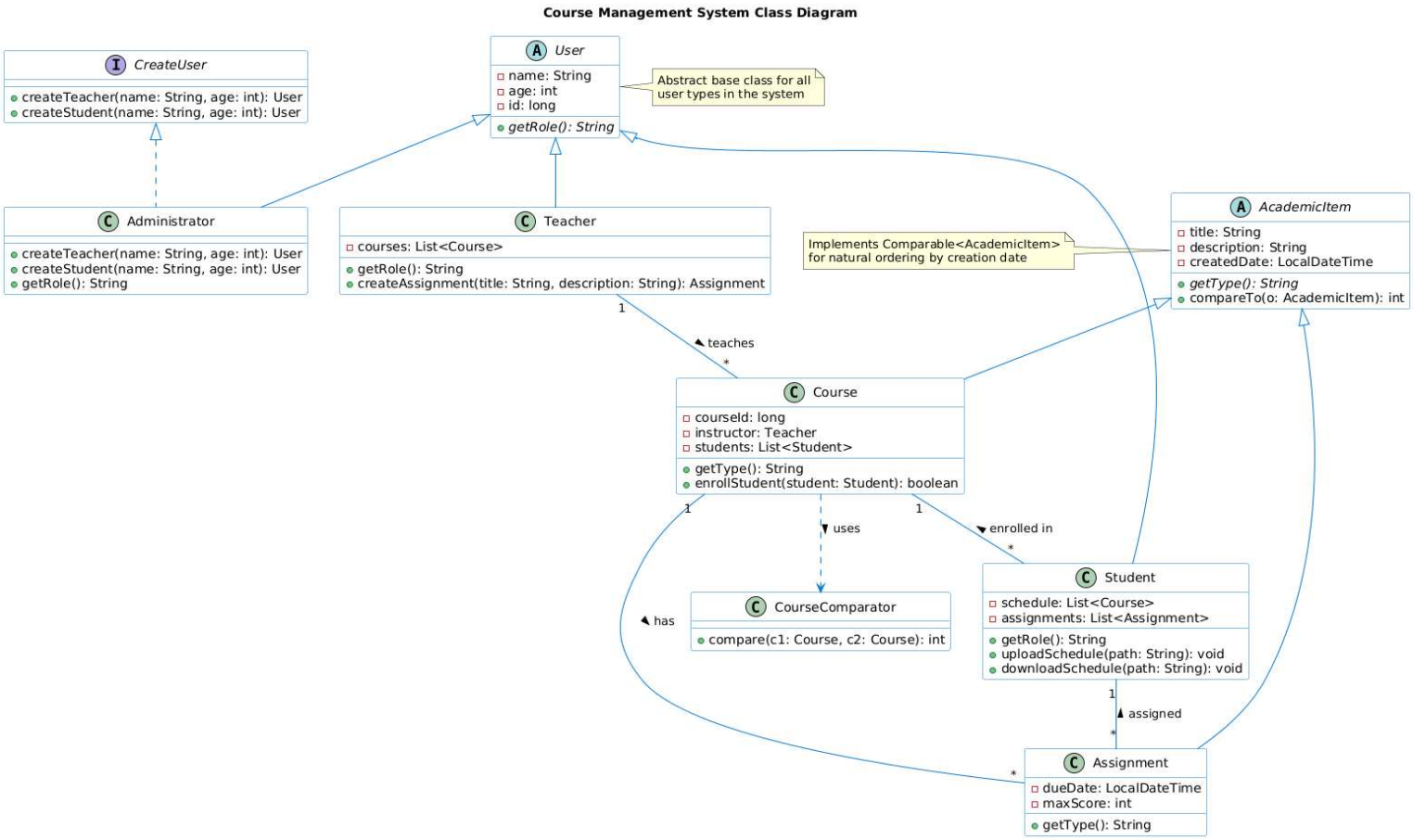
---

8. Comparable and Comparator

- **Comparable**: AcademicItem implements **Comparable<AcademicItem>** to sort by creation date
- **Comparator**: I'll create a **CourseComparator** class to sort courses by:
    - Course ID

- Title
- Number of enrolled students

9. Class Diagram:



10. Deliverable 2 Implementation (50%)

For the second deliverable, I will implement:

Classes:

- Administrator (full implementation)
- Student (with schedule methods)

- Course (basic structure)

Interface:

- Full implementation of CreateUser

Methods:

- **createTeacher()** and **createStudent()** in Administrator
- **uploadSchedule()** and **downloadSchedule()** in Student
- **compareTo()** in AcademicItem
- Basic course management methods in Course