University Coursework Management System

Student Name: Michael McKibbin
Module: Software Development
Module Code: SWDE_IT803 - LY_ICSWD_B 2025/2026
Exercise 1: UCMS - Revised Exercise 1 - No Packages
Lecturer: Lusungu Mwasinga
Date: 26th October 2025

Table of Contents

1. Overview

This project implements the revised version of Exercise 1, the **University Coursework Management System (UCMS)** in Java.

Objective:

To apply and demonstrate **Object-Oriented Programming (OOP)** principles through the design and implementation of a University Coursework Management System in Java.

The system demonstrates core Object-Oriented Programming (OOP) principles — *encapsulation, inheritance, polymorphism, abstraction,* and *composition* — through three main components:

1. **Assessment Module** – an abstract base class `Assessment` and three concrete subclasses:
   - `FinalExamAssessment` ,
   - `TestAssessment` , and
   - `AssignmentAssessment` .

   Each assessment records details such as ID, student, module, lecturer, total marks, weight, assigned/due dates, and provides behaviours for displaying details, checking overdue status, and calculating weighted scores.

2. **Administration Module** – the `Admin` class extends `User` and includes multiple constructors demonstrating **constructor chaining**. Linking the no-argumant, partial, and full constructors, ensures that all initialisation follows the same logic path resulting in consistent object set-up.
   Manages in-memory collections of students, lecturers, courses, and course modules, supporting actions such as:
   - Adding/removing students and lecturers
   - Creating courses
   - Adding course modules to courses
   - Assigning course modules to lecturers

3. **Main Application** – Demonstrates realistic interactions between users, showing the relationships and operations among `Admin`, `Lecturer`, `Student`, `Course`, `CourseModule`, and various assessments.

2. Project Structure (No Packages)

Note: This version was refactored to remove sub-packages for simplicity. All classes now reside in the default project directory, but maintain their logical groupings as shown in the UML diagrams.

3. Key Features

The table below summarises the main object-oriented principles and design features demonstrated in the project.

Principle	Description
Encapsulation	All fields are private; getters/setters control access to data.
Inheritance	`User` is extended by `Student`, `Lecturer`, and `Admin`; `Assessment` is extended by concrete subclasses.
Polymorphism	A single `List<Assessment>` holds mixed assessment types; each subclass overrides `summary()` to display its own details.
Abstraction	`Assessment` is abstract and cannot be instantiated directly.
Composition	A `Course` contains multiple `CourseModule` objects and enrolled `Student`s; `CourseModule` references a `Lecturer`; `Admin` manages all entities.
Constructor Chaining	Demonstrated in the `Admin` class with multiple constructors calling `this(...)`.
Strong Typing with Enums	Prevents invalid data entry, improves code reliability, and ensures compile-time checking of assessment types.

4. UML Diagrams

The following UML diagrams and image files are stored in the `/docs/` folder. Each diagram illustrates a key part of the UCMS system, its relationships, and the application of OOP principles.

Diagram	Description
AssessmentClass.png	Focuses on the `Assessment hierarchy`, showing the abstract Assessment class and its three concrete subclasses (`FinalExamAssessment`, `TestAssessment`, and `AssignmentAssessment`), along with associations to `CourseModule`, `Student`, and `Lecturer`.
AdminRelationships.png	Displays the `Admin` class inheriting from `User` and its associations with `Student`, `Lecturer`, `Course`, and `CourseModule`, demonstrating management and composition relationships.
MainSimulation.png	Sequence diagram demonstrating how the `Main` class coordinates interactions among users and assessments, illustrating polymorphism in action.
ClassModelNoPackages.png	The complete UML class diagram showing all entities, inheritance hierarchies, and composition relationships in the no-packages version of the UCMS.
assessmentClass.puml	PlantUML source file for Figure 1 (Assessment Class Diagram).
ClassModelNoPackages.puml	PlantUML source file for the complete UCMS class model (Figure 4).
ClassModelNoPackages_Mermaid.png	Simplified Mermaid overview diagram showing class hierarchy and high-level structure.
packageUML.md	Markdown source for the Mermaid diagram used to generate a simplified class hierarchy

Figure 1 — Assessment Class

The Assessment Class Diagram illustrates the inheritance and composition relationships that define the assessment subsystem of the UCMS.
The abstract Assessment class provides a shared structure and behaviour for all assessment types, while its concrete subclasses (FinalExamAssessment, TestAssessment, and AssignmentAssessment) specialise that behaviour through unique attributes such as duration, number of questions, or submission requirements.
Associations to CourseModule, Student, and Lecturer demonstrate composition and contextual linkage—each assessment belongs to a module, is assigned by a lecturer, and is taken by a student. This design effectively demonstrates inheritance, abstraction, and composition in practice, ensuring extensibility for future assessment types.

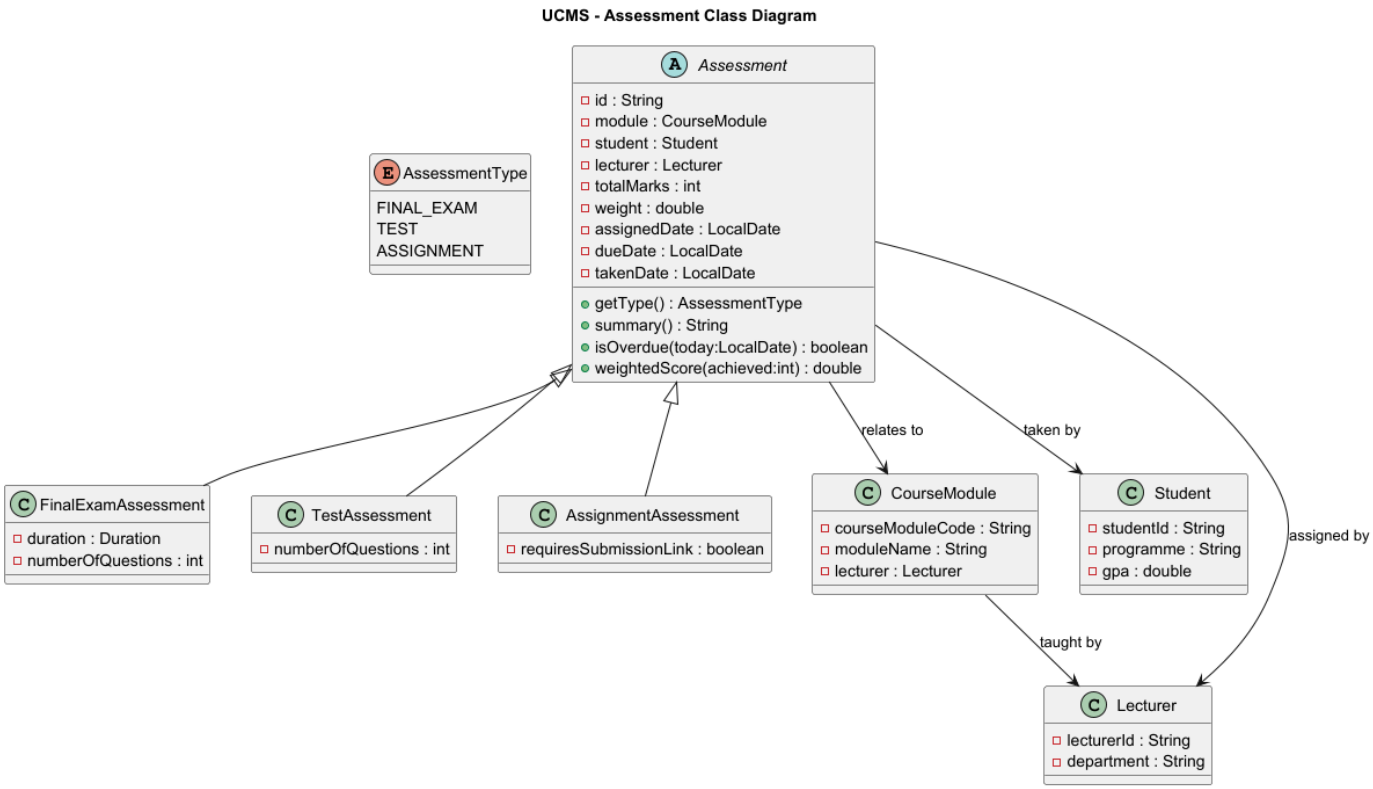

Figure 1: Assessment module class hierarchy showing the abstract `Assessment` *class and its three concrete subclasses (* `FinalExamAssessment`, `TestAssessment`, *and* `AssignmentAssessment`*) with associations to* `CourseModule`, `Student`, *and* `Lecturer`.

Figure 2 — Main Simulation Diagram

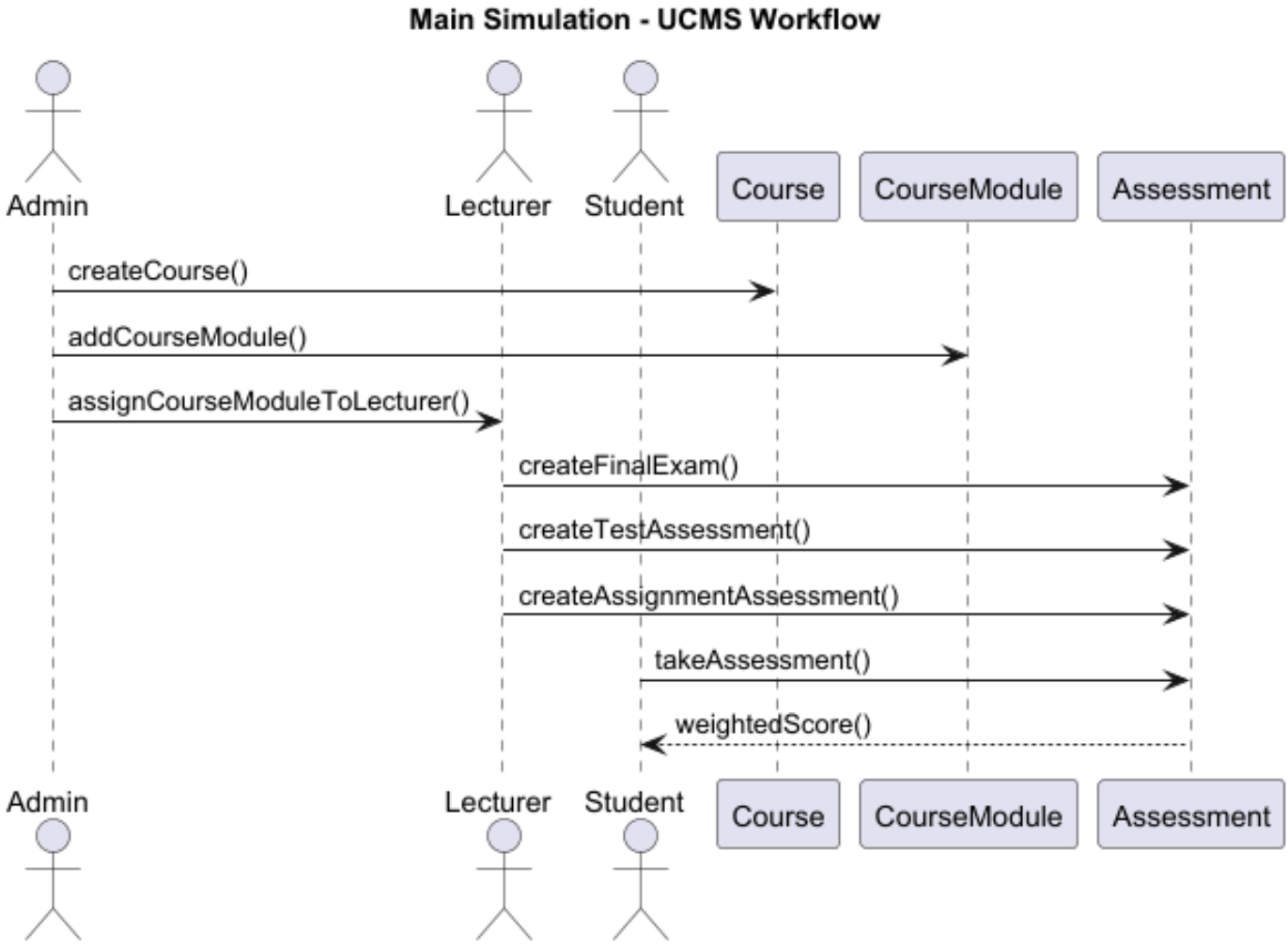

Figure 2: Main simulation sequence diagram demonstrating how `Main` coordinates user interactions and polymorphic assessment handling.

Figure 3 — Admin Relationships

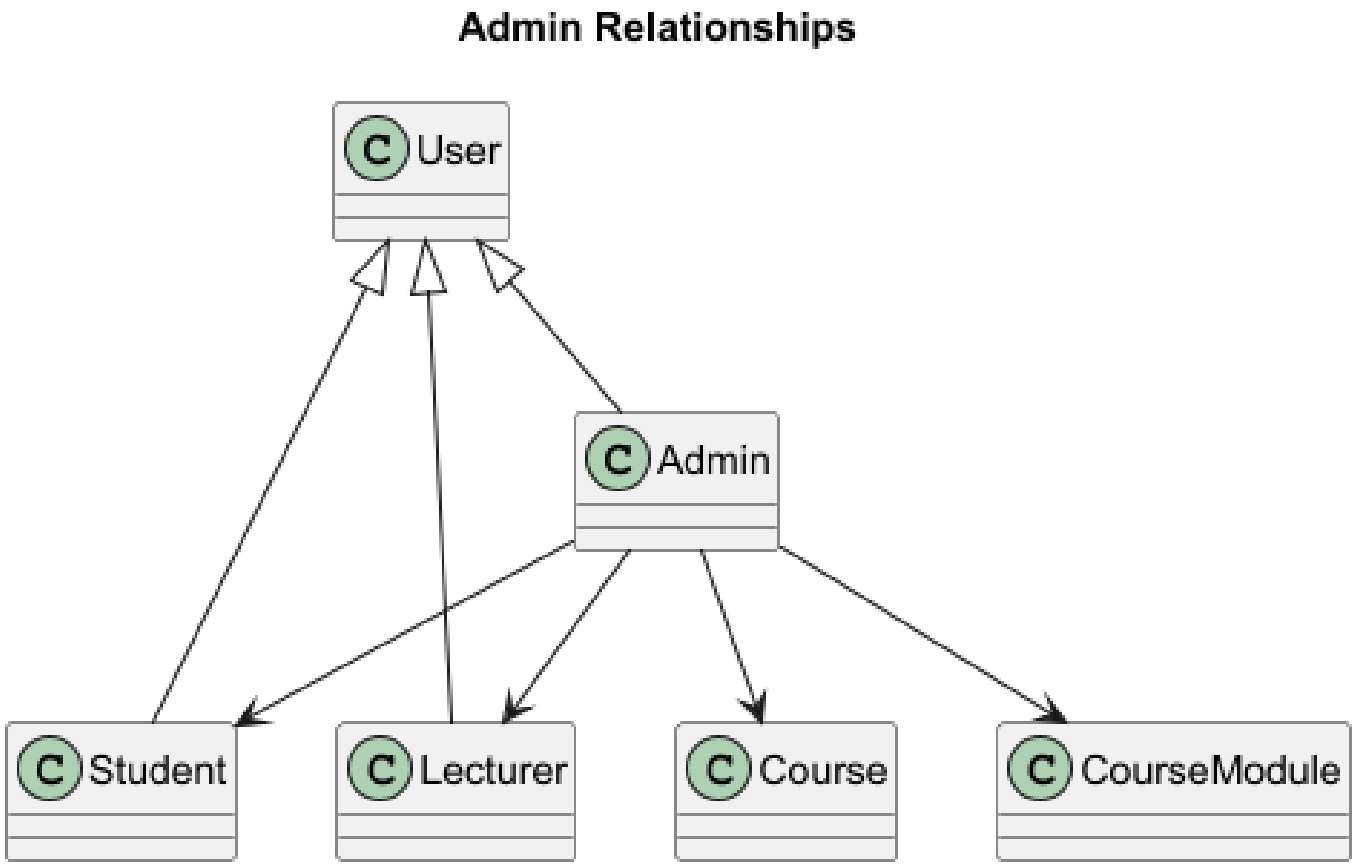

Figure 3: Admin class relationships with User, Student, Lecturer, and CourseModule.

Figure 4 — Class Model

The complete UCMS Class Model presents the full architecture of the system, integrating all major entities and their relationships.
It highlights the three primary inheritance hierarchies—User (extended by Admin, Lecturer, and Student), Assessment (extended by the specific assessment types), and the enum AssessmentType.
The diagram also shows how composition and aggregation connect the academic structures:

- A Course contains multiple CourseModule objects,
- each linked to a Lecturer and
- associated with enrolled Students.

Administrative operations, represented by the Admin class, coordinate these relationships and demonstrate encapsulated management of data. The Main class acts as the entry point, orchestrating interactions and demonstrating polymorphism through the Assessment hierarchy.
Overall, the model conveys a cohesive object-oriented design that balances inheritance for reuse with composition for realistic domain modelling.

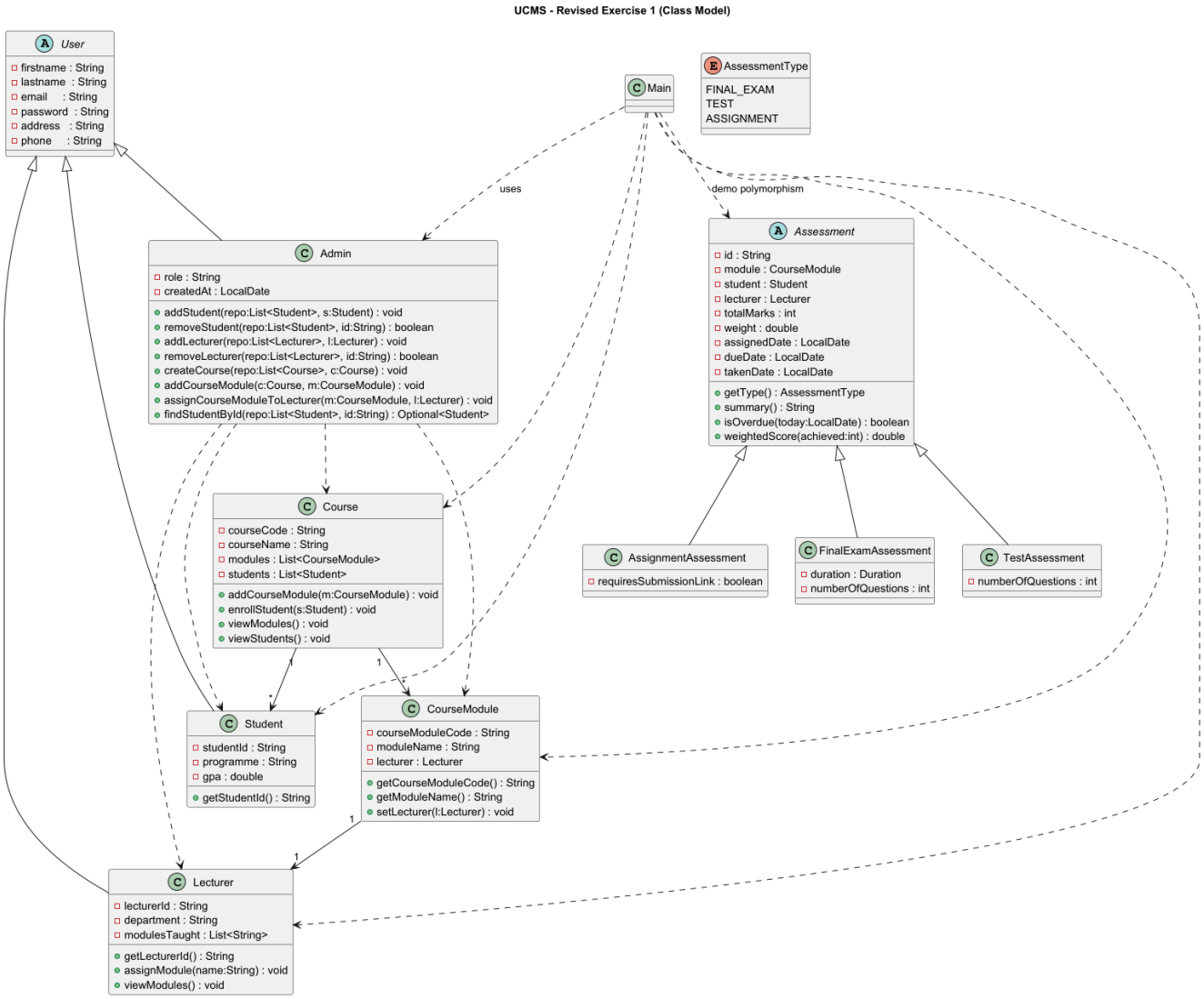

Figure 4: Detailed UML class Diagram.

5. Procedure (Running the Program)

1. Open the project

- Launch your preferred IDE (e.g. IntelliJ IDEA or Eclipse).
- Open the project folder or import the Maven project (pom.xml) to ensure all dependencies and source paths load correctly.
- Verify that the JDK 17 (or any Java 8 +) runtime is selected.

2. Run the main class

- Locate and run Main.java (package com.ucms.app or root folder if using the no-package version).
- The program can be run directly from the IDE's Run button or via the command line using:

```
java com.ucms.Main
```

3. Observe the console output The program demonstrates:

- Admin creation and constructor chaining
- Adding/removing students and lecturers
- Creating courses and assigning course modules
- Lecturer–student–assessment relationships
- Polymorphic assessment iteration and output
- Overdue status and weighted score calculations

Expected result: The console displays meaningful messages showing the entities created and the actions performed.

6. OOP Highlights

Principle	Example
Encapsulation	Private fields with getters/setters in `CourseModule` and `Assessment`.
Inheritance	`Admin`, `Lecturer`, `Student` extend `User`; `FinalExamAssessment`, `TestAssessment`, `AssignmentAssessment` extend `Assessment`.
Polymorphism	`for (Assessment a : assessments)` iterates through multiple types with overridden `summary()` methods.
Abstraction	`Assessment` defines shared structure and abstract behaviour for its subclasses.
Composition	`Course` "has-a" list of `CourseModule` and `Student`; `CourseModule` "has-a" `Lecturer`.
Constructor Chaining	`Admin` provides three constructors calling each other with `this(...)`.

7. Notes on Revisions

- **De-packaged this version** For simplicity, the project was de-packaged to a single root package.
- **Renamed** `Module` → `CourseModule` to avoid conflict with `java.lang.Module` and improve clarity.
- **Simplified "Quiz" and "Test"** into a single unified `TestAssessment` class. ('Quiz' is not mentioned in revised requirements).
- **Consolidated Admin operations** to manage data using in-memory `List<>` structures. (No db).
- **Main simulation** redesigned to show constructor chaining, encapsulation, and polymorphism with practical console outputs.

8. Technologies Used

- **Language:** Java 21 (Uses only *core Java8+ syntax*. Compatible with *Java 8, 11, 17, & 21*.)
- **Build IDE:** IntelliJ IDEA 2025.2
- **Build Tool:** Maven 3.13.0
- **Testing:** JUnit 5.12.1 for lightweight validation
- **Diagrams:** PlantUML / IntelliJ UML tool

9. Author

> **Name:** Michael McKibbin
> **Course:** B.Sc. (Hons) in Contemporary Software Development
> **Institution:** Atlantic Technological University (ATU)
> **Year:** 2025/2026
> **Exercise:** UCMS – Revised Exercise 1 - No Packages

10. Design Rationale

The revised UCMS design prioritises clarity, consistency, and maintainability, while demonstrating the core principles of encapsulation, inheritance, and polymorphism in a cohesive and extensible object-oriented structure.

The design was guided by the goals of Exercise 1: to produce a modular, well-structured system that models realistic academic relationships between users, courses, and assessments.

Renaming the original Module class to CourseModule eliminated a potential naming conflict with java.lang.Module and clarified the class's specific role within a Course. Although it was technically possible to retain the original name, using imports, doing so risked confusion between UCMS domain classes and Java system classes. The updated naming convention enhances readability and reinforces clear semantic intent throughout the codebase.

Administrative functionality was consolidated within a single Admin class, using in-memory collections (List<>) to manage entities such as students, lecturers, and courses. This approach reduces redundancy, simplifies maintenance, and provides a flexible foundation that could later be extended to persistent storage without structural changes.

Since references to 'Quiz' were removed in the revised specifications, the assessment hierarchy was streamlined from four classes to three — FinalExamAssessment, TestAssessment, and AssignmentAssessment — each extending a shared abstract superclass Assessment. These demonstrate inheritance and polymorphism while ensuring that common logic is defined once and reused across all assessment types.

Constructor chaining in the Admin class promotes code reuse and consistent object initialisation, while composition between Course, CourseModule, and Lecturer models "has-a" relationships that mirror real-world associations. Encapsulation is reinforced through the use of private fields and controlled accessors, preserving data integrity and supporting future scalability.

Overall, this iteration delivers a cleaner and more maintainable architecture with clear package separation, logical class interactions, and console output that effectively demonstrates the intended OOP behaviours. The design achieves the

educational objectives of the exercise by integrating theoretical principles into a practical and verifiable implementation.