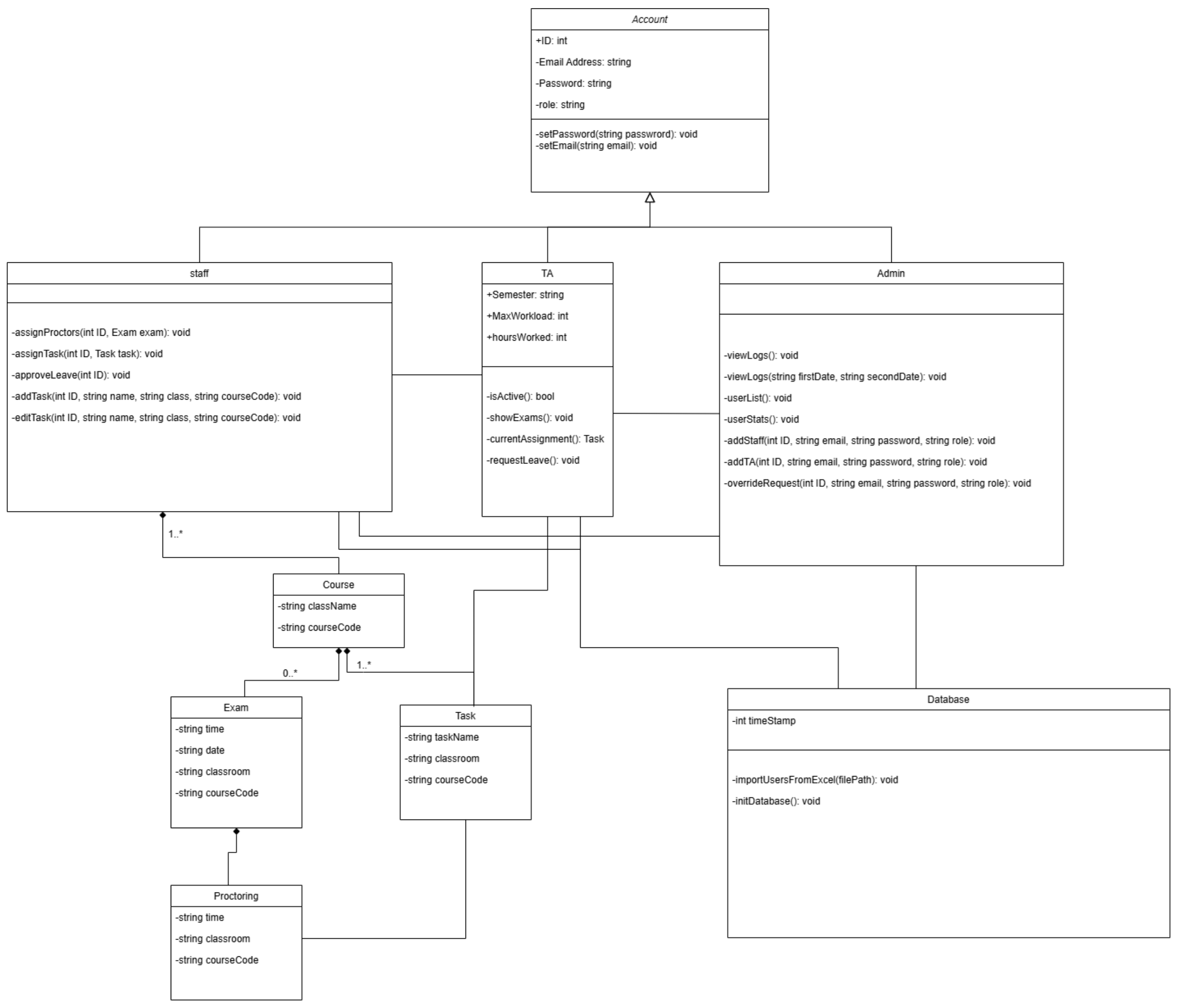
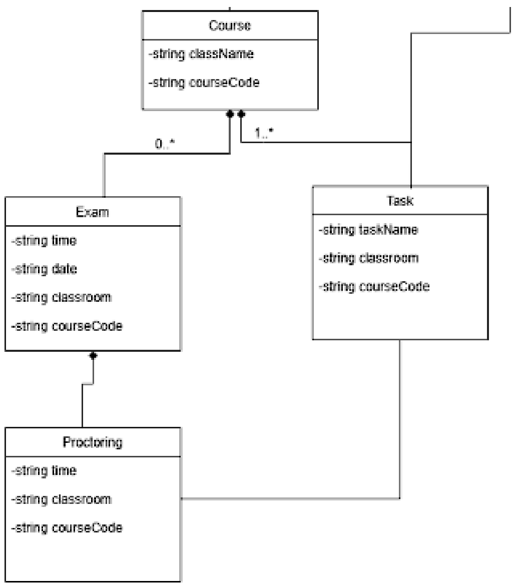
Class Diagram


Account
+ID: int
-Email Address: string
-Password: string
-role: string
-setPassword(string passwrord): void
-setEmail(string email): void
staff
-assignProctors(int ID, Exam exam): void
-assignTask(int ID, Task task): void
-approveLeave(int ID): void
-addTask(int ID, string name, string class, string courseCode): void
-editTask(int ID, string name, string class, string courseCode): void
TA
+Semester: string
+MaxWorkload: int
+hoursWorked: int
-isActive(): bool
-showExams(): void
-currentAssignment(): Task
-requestLeave(): void
Admin
-viewLogs(): void
-viewLogs(string firstDate, string secondDate): void
-userList(): void
-userStats(): void
-addStaff(int ID, string email, string password, string role): void
-addTA(int ID, string email, string password, string role): void
-overrideRequest(int ID, string email, string password, string role): void
1..*
Course
-string className
-string courseCode
0..*
1..*
Exam
-string time
-string date
-string classroom
-string courseCode
Task
-string taskName
-string classroom
-string courseCode
Database
-int timeStamp
-importUsersFromExcel(filePath): void
-initDatabase(): void
Proctoring
-string time
-string classroom
-string courseCode

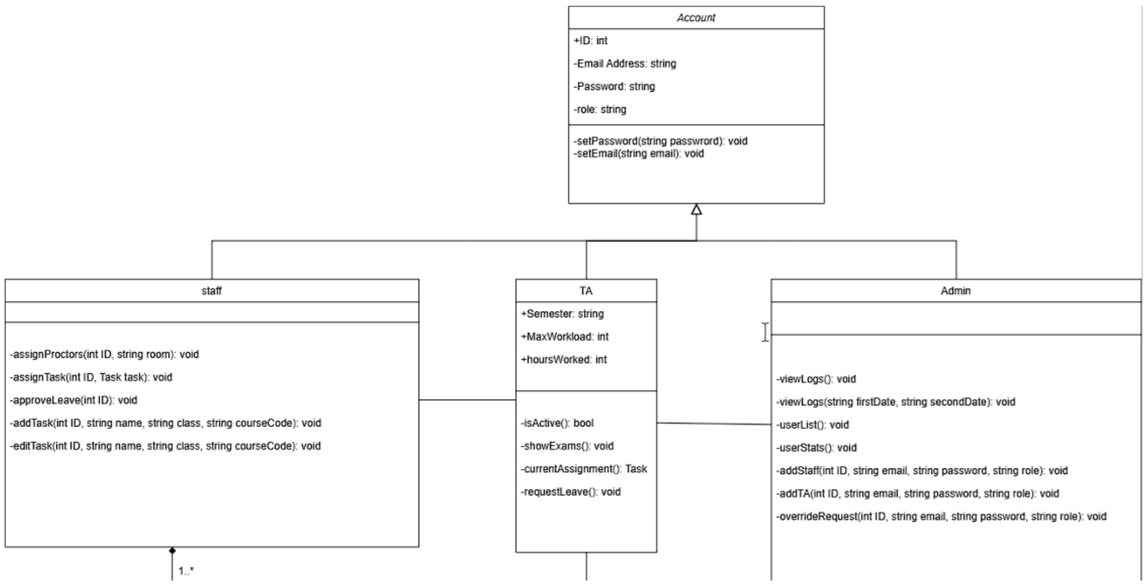
Software Design Patterns

1. Composition:



The composite design pattern is represented in the class diagram by the following classes:
Course: This is the class that holds the course information, both class name or course code. For example CS101 Algorithms and Programming I. It includes the Task class and the Exam class, which cannot exist without a course.
Tasks: This is a concrete class that represents a Task object and holds all information about a TA’s tasks including the name of the task, the classroom its in, and the course code it relates to. It is a leaf node in the composite hierarchy with no child objects. A TA cannot exist if courses didn’t exist.
Exam: This is a concrete class that represents an Exam object and holds all the information about an exam, including a time, date, classroom, and course code. An exam cannot exist without its parent, the course.
Proctoring: This is a concrete class that represents the proctoring task. It relates to the task class, since it is just a more specialized task class. Proctoring cannot exist without an exam class. It is a leaf node in the composite hierarchy with no child objects

2. Generalization (Inheritance)


Account
+ID: int
-Email Address: string
-Password: string
-role: string
-setPassword(string passwrord): void
-setEmail(string email): void
staff
-assignProctors(int ID, string room): void
-assignTask(int ID, Task task): void
-approveLeave(int ID): void
-addTask(int ID, string name, string class, string courseCode): void
-editTask(int ID, string name, string class, string courseCode): void
1..*
TA
+Semester: string
+MaxWorkload: int
+hoursWorked: int
-isActive(): bool
-showExams(): void
-currentAssignment(): Task
-requestLeave(): void
Admin
-viewLogs(): void
-viewLogs(string firstDate, string secondDate): void
-userList(): void
-userStats(): void
-addStaff(int ID, string email, string password, string role): void
-addTA(int ID, string email, string password, string role): void
-overrideRequest(int ID, string email, string password, string role): void

Account: This is the abstract superclass that generalizes common attributes and behaviors shared by all user types in the system. It holds general information such as ID, email address, password, and role. It also provides shared operations like setPassword and setEmail that are inherited by TA, staff, and admin subclasses. The Account class serves as the root of the generalization hierarchy.

TA (Teaching Assistant): This is a concrete subclass of Account that represents a user type with attributes and methods specific to TAs, such as Semester, MaxWorkload, and hoursWorked. TAs can check their active status, view their current assignment, request leave, and show their exam schedule. This class inherits the basic user structure from Account but extends it with TA-specific functionality.

Staff: This is another concrete subclass of Account, representing users who manage tasks and proctors. It includes methods such as assignProctors, assignTask, approveLeave, addTask, and editTask. This class inherits the basic user structure from Account but extends it with staff specific functionality.

Admin: This class is also a subclass of Account, representing administrative users with elevated privileges. Admins can perform actions such as viewing logs, accessing user statistics, and adding or overriding accounts. This class inherits the basic user structure from Account but extends it with admin specific functionality.