**CHAPTER THREE**

3.1 Low Level (Detail) Design (Class Diagram)

Low-Level Design (LLD), also known as Detail Design, focuses on the internal structure and implementation details of the system components identified during the High-Level Design. For a Kindergarten Management System, the LLD class diagram will delve into the attributes (data) and operations (behavior) of each class, their relationships, and constraints.

3.2 Components Of Class Diagram

**\* Classes: Represented as rectangles, each divided into three compartments: class name, attributes, and methods.**

**\* Attributes: Define the properties of the class using data types like int, string, date, etc.**

**\* Methods: Operations or functions that the class can perform. These reflect business logic (e.g., login, uploadDocs, markAttendance).**

**\* Relationships:**

**\* Association: Direct connection between two classes (e.g., a User can send multiple Notifications).**

**\* Aggregation: Represented by a hollow diamond, indicating a whole-part relationship (e.g., Parent has one or more ChildProfiles).**

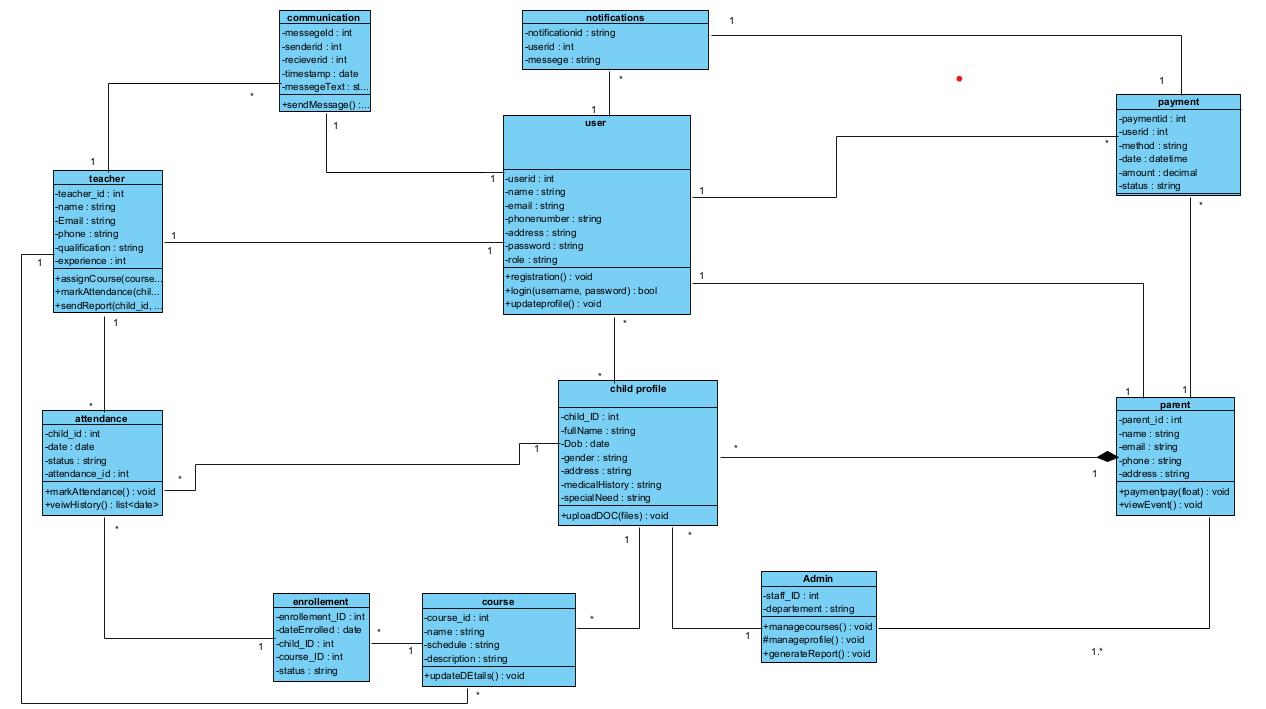
**\* Multiplicity: Defines how many instances of a class can be associated with another (e.g., 1..\\* means one to many).**

**\* Visibility Notations:**

**+ Public**

**- Private**

3.3 Examples Of Class Diagram

****

**Figure 3.1 Class Diagram For Kindergarten Management System**

**Tools and Steps to Draw High Level Sequence Diagram**

* **Tool used:visual paradigm**

**Steps to Draw Low Level(Detail) Design (Class Diagram)**

· **Identify the Classes:**

* Look at the image and identify the rectangles. Each rectangle represents a class.
* List the classes present in the diagram:
  + User
  + Communication
  + Teacher
  + Course
  + Enrollment
  + Attendance
  + ChildProfile
  + Parent
  + Payment
  + Admin
  + Notification

· **Draw the Classes:**

* In your chosen tool, create a class element (usually a rectangle) for each class in the list.
* Arrange the classes in a way that makes the diagram easy to read. A common approach is to place related classes near each other.

· **Add Attributes:**

* Inside each class rectangle, there are sections. The top section contains the class name. The middle section lists the attributes (data) that the class holds.
* For each class, add the attributes shown in the image, including their data types (e.g., userId: int, name: string).
  + **User:** userId: int, name: string, username: string, email: string, phoneNumber: string, address: string, password: string, role: string
  + **Communication:** messageId: int, senderId: int, receiverId: int, timestamp: date, messageText: string
  + **Teacher:** teacher\_id: int, name: string, email: string, phone: string, qualification: string, experience: string
  + **Course:** course\_id: int, name: string, schedule: string, description: string
  + **Enrollment:** enrollment\_id: int, dateEnrolled: date, child\_id: int, course\_id: int, status: string
  + **Attendance:** attendance\_id: int, child\_id: int, teacher\_id: int, date: date, status: string, attendance\_id: int
  + **ChildProfile:** child\_id: int, fullName: string, dob: date, gender: string, address: string, medicalHistory: string, specialNeeds: string
  + **Parent:** parent\_id: int, name: string, email: string, phone: string, address: string
  + **Payment:** paymentId: int, parentId: int, invoiceId: string, paymentDate: datetime, amount: decimal, status: string
  + **Admin:** staff\_ID: int, department: string
  + **Notification:** notificationId: int, message: string, timestamp: datetime, userId: int

· **Add Operations (Methods):**

* The bottom section of each class rectangle lists the operations (actions) that the class can perform.
* For each class, add the operations shown in the image, including their parameters and return types (if any)
  + **User:** registration(): void, loginUsername(username, password): bool, updateProfile(): void
  + **Teacher:** assignCourse(courseId): void, markAttendance(child\_id, date, status): void, sendReportCard(child\_id): void
  + **ChildProfile:** uploadDOC(files): void
  + **Parent:** viewEvents(): void
  + **Admin:** manageUsers(): void, manageCourses(): void, manageProfile(): void, generateReport(): void

· **Draw Relationships (Associations):**

* The lines connecting the classes represent relationships between them.
* Draw lines between the classes as they appear in the image.

· **Add Multiplicity:**

* At each end of a relationship line, there are symbols or numbers that indicate multiplicity. This shows how many instances of one class are related to instances of another class.
* Add the multiplicity symbols as they are in the image:
  + 1 - One
  + \* - Many (zero or more)
  + 1..\* - One or more