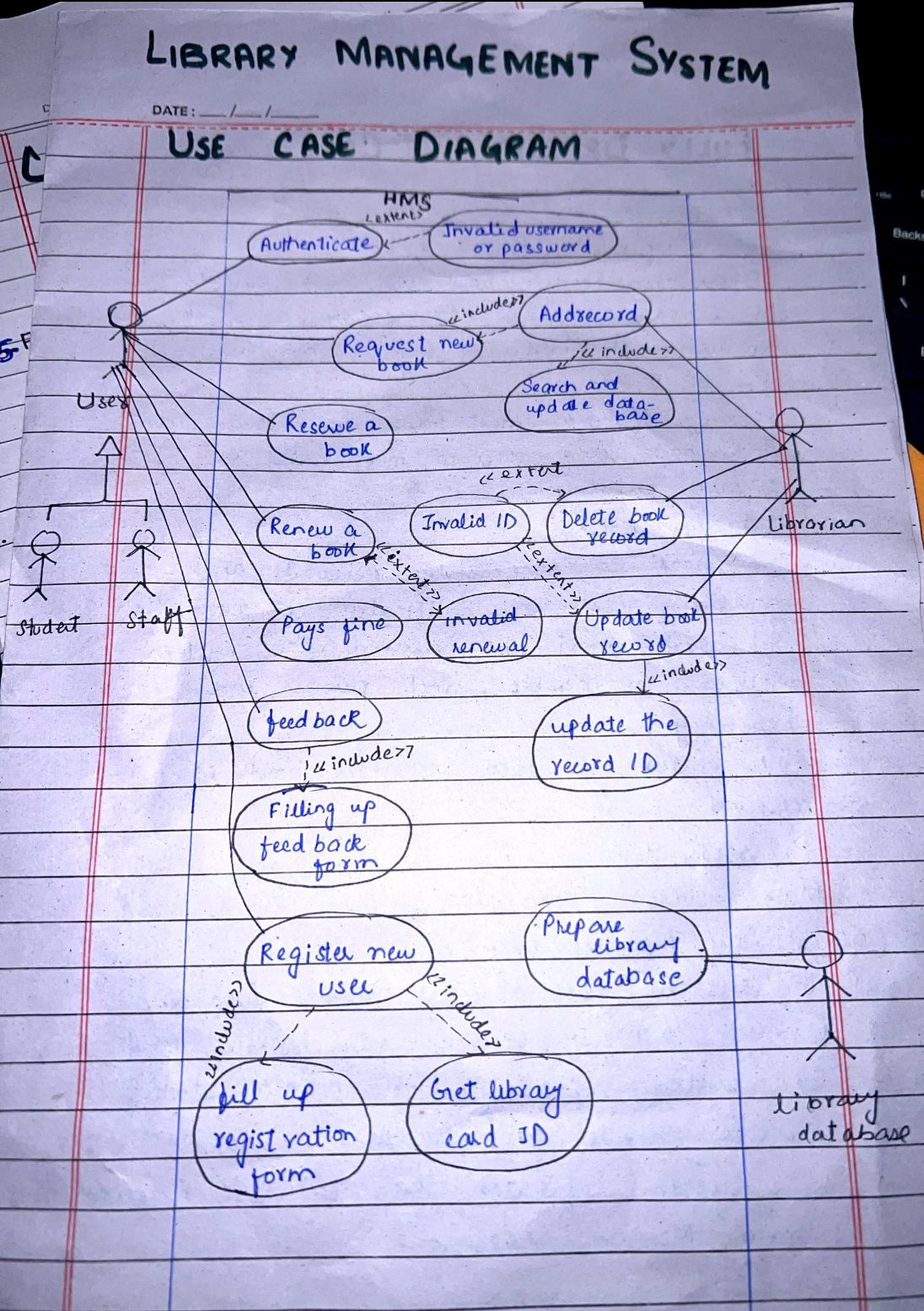
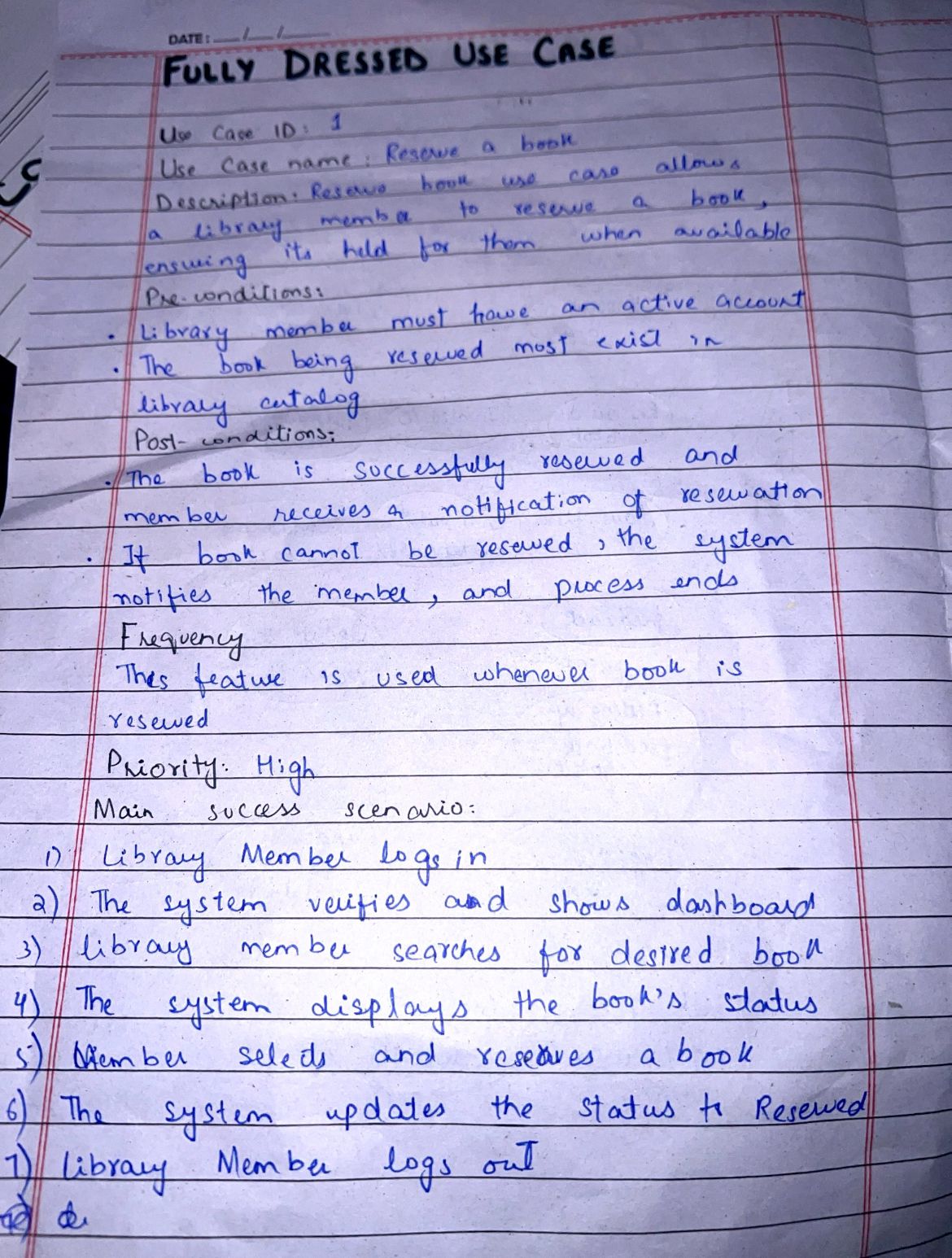
**LIBRARY MANAGEMENT SYSTEM**

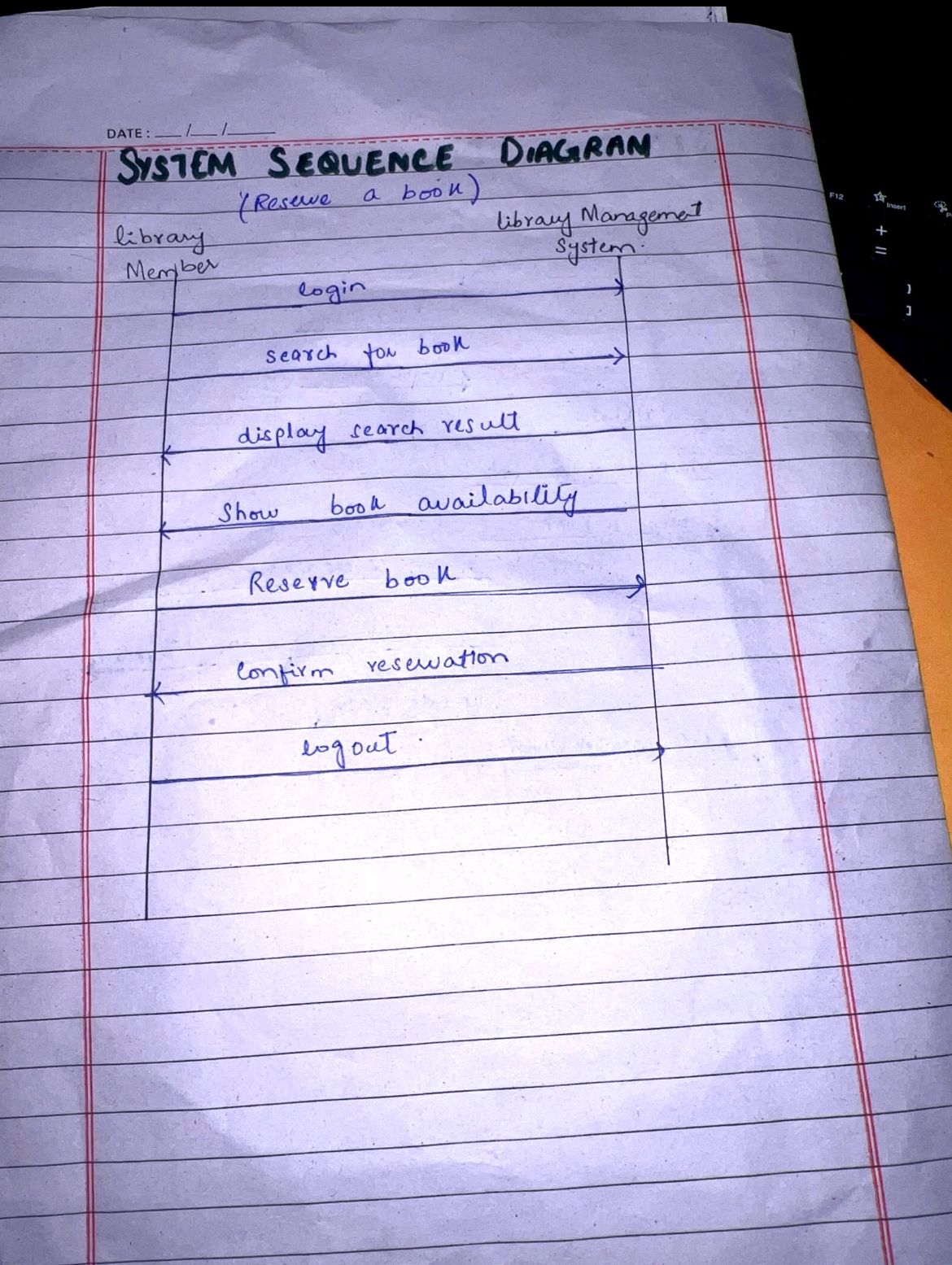
**Use case diagram**



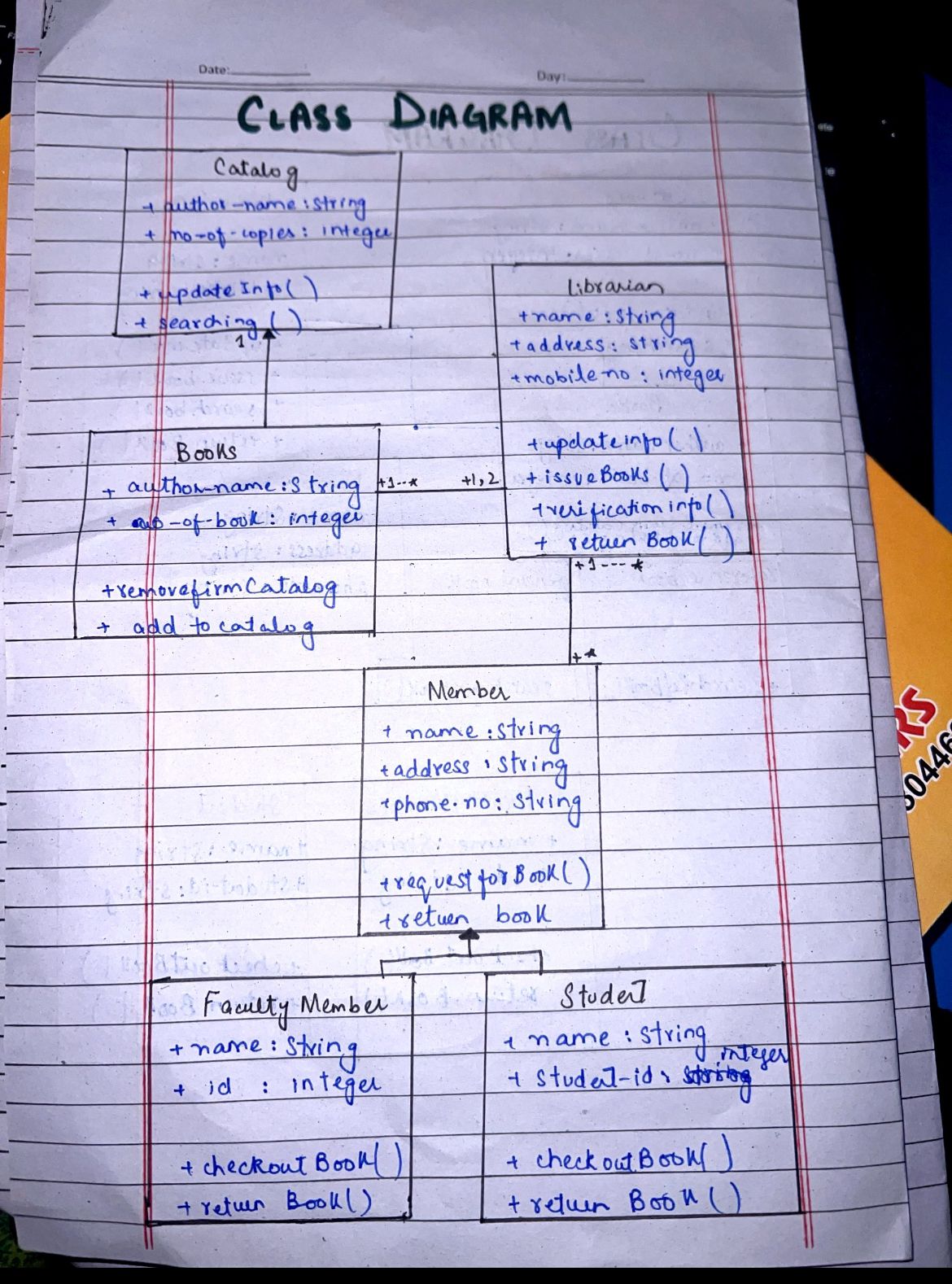
**Fully dressed use case**



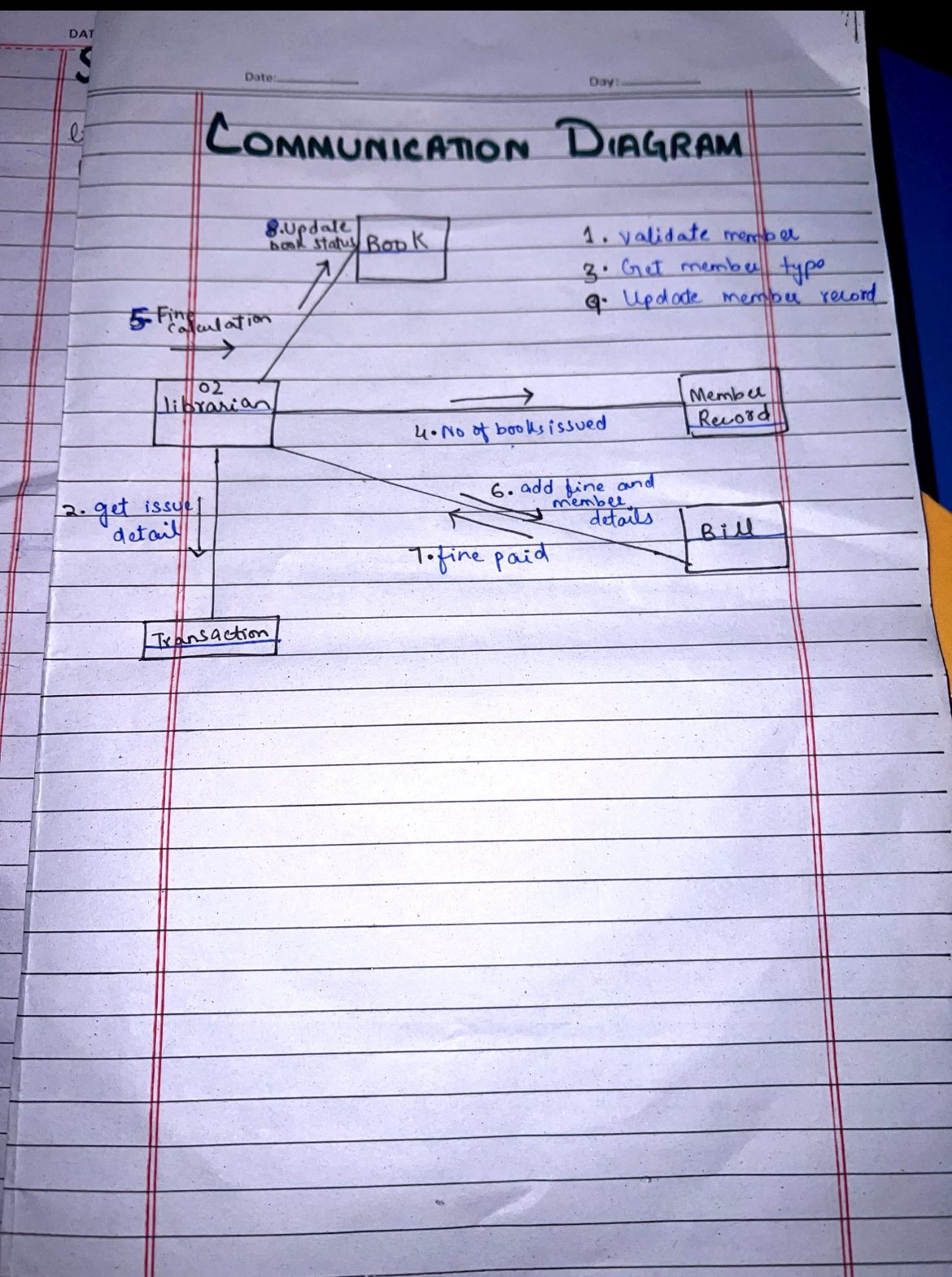
**System Sequence diagram**



**Class diagram**



**Communication diagram**



**Principles used in Communication diagram**

**1. Low Coupling:**

* **Definition**: Low coupling refers to minimizing dependencies between components, meaning changes in one part of the system shouldn't heavily affect others.
* **In the Diagram**: The components (Librarian, Book, Member Record, Bill, etc.) have specific, limited interactions with each other. For example, the **Librarian** interacts with the **Book** to update its status and with the **Member Record** to validate and update member information. Each interaction serves a clear purpose without creating excessive dependencies between these objects.

**2. Polymorphism:**

* **Definition**: Polymorphism allows objects to be treated as instances of their parent class, enabling flexibility and extensibility.
* **In the Diagram**: Although the diagram doesn't explicitly show polymorphism, it can be assumed in the interactions. For instance, different types of members (e.g., student, faculty) might be handled in the **Member Record** system through polymorphic behaviors, where different types of users are validated using the same interface.

**3. Pure Fabrication:**

* **Definition**: Pure fabrication refers to creating classes or objects that don’t map directly to the real-world problem but are necessary for maintaining high cohesion and low coupling.
* **In the Diagram**: The **Bill** object could be an example of pure fabrication. It's not a real-world actor but a construct created to manage transactions such as fine payments, separate from the **Librarian** and **Transaction** components.

**4. High Cohesion:**

* **Definition**: High cohesion means that each component or class has a focused responsibility, performing a single task or a closely related set of tasks.
* **In the Diagram**: Each component performs a specific function:
  + **Librarian** handles fine calculations and book status updates.
  + **Book** manages validation, status updates, and member record updates.
  + **Bill** manages fine payments. Each component focuses on its specific responsibilities, demonstrating high cohesion.

**5. Controller:**

* **Definition**: The controller is responsible for handling system events and directing messages between different objects.
* **In the Diagram**: The **Librarian** acts as the controller, initiating interactions such as book validation, issuing books, and updating records. The librarian controls the flow of data between different system components (Book, Member Record, etc.).