In **Low-Level Design (LLD)**, the focus is on **how** the system will be built. It involves designing individual modules/classes, their interactions, and implementing OOP principles, design patterns, and more.

Here's a **comprehensive list** of topics to study in LLD, grouped logically:

---

1. Object-Oriented Programming (OOP)

- **Principles of OOP**
    - Encapsulation
    - Abstraction
    - Inheritance
    - Polymorphism
- **SOLID Principles**
    - Single Responsibility Principle
    - Open/Closed Principle
    - Liskov Substitution Principle
    - Interface Segregation Principle
    - Dependency Inversion Principle

---

2. Class Design

- Defining classes, attributes, methods
- Constructors, destructors
- Access modifiers (public, private, protected)

- Relationships between classes:
    - Association
    - Aggregation
    - Composition
    - Inheritance (extends)
    - Implementation (implements interface)

---

3. Design Patterns (very important)

- **Creational Patterns**:
    - Singleton
    - Factory
    - Abstract Factory
    - Builder
    - Prototype
- **Structural Patterns**:
    - Adapter
    - Decorator
    - Proxy
    - Composite
    - Facade
- **Behavioral Patterns**:

- Strategy
- Observer
- Command
- State
- Chain of Responsibility
- Template Method

---

4. UML Diagrams (Visual Design)

- **Class Diagram**
- **Sequence Diagram**
- Use Case Diagram
- Activity Diagram
- Component Diagram

---

5. Designing Real-World Systems (Object Modeling)

Study how to **model real-world systems** using OOP:

- Parking Lot
- Library Management System
- Movie Booking System
- Ride Sharing (Uber/Ola)
- Food Delivery (Zomato/Swiggy)

- ATM System
- Splitwise
- Amazon / Flipkart cart systems

Focus on:

- Identifying key classes
- Defining responsibilities
- Interactions and communication (method calls)
- Managing dependencies

---

6. Code Principles

- DRY (Don’t Repeat Yourself)
- KISS (Keep It Simple, Stupid)
- YAGNI (You Aren’t Gonna Need It)
- Law of Demeter (Least knowledge principle)
- Composition over Inheritance

---

7. Interfaces vs Abstract Classes

- When to use interfaces
- When to use abstract classes
- Multiple inheritance via interfaces

---

8. Error Handling and Edge Cases

- Exception design
- Null object pattern
- Defensive coding practices

---

9. Testability

- Writing code that is easy to unit test
- Using dependency injection
- Mocking and stubbing dependencies

---

10. Scalability in Code Design (Readiness for HLD)

- Extensibility
- Modularity
- Loose coupling
- High cohesion

Would you like a **study plan** or a breakdown of **system examples with diagrams** next?