LECTURE - 1

1. What is LLD? (Introduction)

LLD (Low-Level Design) is the bridge between your DSA knowledge and building real-world applications like Swiggy, Zomato, Ola, or Uber.

- **Context:** Knowing DSA (Data Structures & Algorithms) is not enough to build scalable software.
- **Goal:** To understand how to structure code so it can handle millions of users (Scalability) and is easy to work on (Maintainability).

Real World Analogy:

DSA solves **isolated problems** (like searching an element in an array using Binary Search). LLD builds the **entire application** around these solutions.

---

2. The Story: Anurag vs. Maurya (DSA vs. LLD)

Imagine two friends hired at a company called **"Quick Ride"** (similar to Ola/Uber) to build a ride-booking app.

Character Profile:

- **Anurag:** DSA Expert, but knows zero LLD.
- **Maurya:** Knows both DSA and LLD.

Task: Build the Quick Ride Application.

Anurag's Approach (Pure DSA Mindset)

Anurag immediately jumps to coding algorithms:

1. **Problem 1 (Route Finding):**
    - *Thought:* "I need to go from Source to Destination."
    - *Solution:* Treats the city as a **Graph** (Intersections = Nodes, Roads = Edges). Uses **Dijkstra's Algorithm** to find the shortest path.
2. **Problem 2 (Rider Assignment):**
    - *Thought:* "I need to find the nearest rider."
    - *Solution:* Maps every user to a **Priority Queue (Min-Heap)** of nearby riders. Popping from the heap gives the closest rider.

The Manager's Feedback (The Issue):

- "Where are the **Entities/Objects** (User, Rider)?"
- "How do they **relate** to each other?"
- "What about **Data Security**? (Hiding phone numbers)"
- "How will we handle **Notification & Payments**?"
- "Will this code handle **millions of users**?"

Maurya's Approach (LLD Mindset)
Maurya focuses on the **Structure** (Skeleton) first:

1. **Identify Objects:** User, Rider, Location, Notification, Payment.
2. **Relationships:** How does a User interact with a Rider?
3. **Security:** Ensuring User and Rider cannot see each other's phone numbers after the ride.
4. **Scalability:** Writing code that doesn't break when traffic increases.
5. **Integration:** *Only after* the structure is ready does he apply DSA (Graph/Heaps) to solve specific problems.

---

3. The 3 Pillars of LLD

When designing software, LLD focuses on three core principles:

Pillar	Meaning	Explanation
1. Scalability	Handling Growth	If users increase from 100 to 1 Million, the app should not crash. It deals with sustaining high traffic.
2. Maintainability	Easy to Update	Adding a new feature shouldn't break 4 old features. The code should be easy to **Debug** (find bugs).
3. Reusability	Plug & Play	Code should not be "Tightly Coupled". *Example:* A "Notification System" or "Rider Matching Algo" written for Quick Ride should be reusable in a Zomato clone with minimal changes.

---

4. What is NOT LLD? (HLD vs. LLD vs. DSA)

Many people confuse Low-Level Design with High-Level Design.

Comparison Table:

Feature	HLD (High-Level Design)	LLD (Low-Level Design)
Focus	**System Architecture**	**Code Structure**
Key Decisions	Tech Stack (Java vs Python), Database (SQL vs NoSQL), Server Scaling, Cost Optimization (AWS).	Class Diagrams, Object Relationships, Interfaces, OOP Principles.
Code Written	Almost Zero (Diagram heavy)	Actual Code Structure
Analogy	Designing the City Plan	Designing the House Blueprint

The Golden Rule:

> **"If DSA is the Brain of an application, LLD is the Skeleton."**

---

5. Interview Questions

Q1: Why isn't DSA enough to build an application?

Answer: DSA provides tools (algorithms) to solve specific, isolated problems (like sorting or searching). It doesn't tell you how to structure the code, handle database connections, ensure security, or make the system maintainable for other developers. That requires LLD.

Q2: What is "Tight Coupling" and why is it bad?

Answer: Tight coupling means your code components are highly dependent on each other. If you change one part, other parts break. LLD encourages **Loose Coupling** (Reusability), where components (like a Payment module) can be swapped or reused without breaking the app.

Q3: Difference between Debugging and Maintainability?

Answer: Debugging is the act of finding and fixing a bug. Maintainability is a property of the code that makes debugging *easy*. If code is maintainable, you can find bugs quickly and add features without creating new bugs.

Q4: Does LLD deal with Databases and Servers?

Answer: No, that is generally part of **HLD (High-Level Design)**. LLD deals with the classes, objects, and logic *within* the code that interacts with those databases, but the choice of database or server scaling strategy is an HLD decision.