

Low Level Design (LLD) Teaching Roadmap

Phase 1: Foundations (Week 1-2)

Prerequisites

- **Object-Oriented Programming (OOP) Concepts**
 - Classes and Objects
 - Inheritance (IS-A relationship)
 - Polymorphism (method overriding and overloading)
 - Encapsulation (data hiding)
 - Abstraction (hiding implementation details)
- **Basic Data Structures**
 - Arrays and Lists
 - Hash Maps and Hash Sets
 - Stacks and Queues
 - Trees (basic understanding)
- **Why Design Matters**
 - Code maintainability
 - Scalability concerns
 - Readability and collaboration
 - Reducing technical debt

Phase 2: Core Design Principles - SOLID (Week 2-3)

S - Single Responsibility Principle (SRP)

- **Definition:** A class should have only one reason to change
- **Example:** Separate User class from UserValidator and UserRepository
- **Anti-pattern:** God classes that do everything

O - Open/Closed Principle (OCP)

- **Definition:** Classes should be open for extension but closed for modification
- **Example:** Using interfaces/abstract classes to add new functionality

- **Key concept:** Achieve through abstraction and polymorphism

L - Liskov Substitution Principle (LSP)

- **Definition:** Derived classes must be substitutable for their base classes
- **Example:** Square-Rectangle problem
- **Rule:** Child classes should strengthen, not weaken, the parent's contract

I - Interface Segregation Principle (ISP)

- **Definition:** Clients shouldn't be forced to depend on interfaces they don't use
- **Example:** Split fat interfaces into smaller, specific ones
- **Benefit:** Reduces coupling and improves flexibility

D - Dependency Inversion Principle (DIP)

- **Definition:** Depend on abstractions, not concretions
 - **Example:** Use interfaces instead of concrete classes
 - **Benefit:** Loose coupling, easier testing and maintenance
-

Phase 3: Design Patterns (Week 3-6)

Creational Patterns (Creating Objects)

1. Singleton Pattern

- **Purpose:** Ensure only one instance of a class exists
- **Use cases:** Database connections, logging, configuration managers
- **Implementation:** Private constructor, static instance, getInstance() method
- **Considerations:** Thread safety, lazy vs eager initialization

2. Factory Pattern

- **Purpose:** Create objects without specifying exact class
- **Use cases:** Shape factory, notification factory (Email, SMS, Push)
- **Benefit:** Centralizes object creation logic
- **Types:** Simple Factory, Factory Method, Abstract Factory

3. Builder Pattern

- **Purpose:** Construct complex objects step by step
- **Use cases:** Building objects with many optional parameters
- **Example:** Building a House, Car, or complex Query
- **Benefit:** Avoids telescoping constructors

Structural Patterns (Organizing Objects)

4. Adapter Pattern

- **Purpose:** Make incompatible interfaces work together
- **Use cases:** Third-party library integration, legacy code
- **Example:** Power adapter, media player adapter

5. Decorator Pattern

- **Purpose:** Add behavior to objects dynamically
- **Use cases:** Adding toppings to pizza, adding features to notifications
- **Benefit:** Alternative to subclassing

6. Facade Pattern

- **Purpose:** Provide simplified interface to complex subsystem
- **Use cases:** Home theater system, complex library APIs
- **Benefit:** Reduces complexity for clients

Behavioral Patterns (Object Communication)

7. Observer Pattern

- **Purpose:** Define one-to-many dependency between objects
- **Use cases:** Event handling, notification systems, MVC
- **Example:** YouTube subscriptions, stock market updates

8. Strategy Pattern

- **Purpose:** Define family of algorithms, make them interchangeable
- **Use cases:** Payment methods, sorting algorithms, navigation routes
- **Benefit:** Eliminates conditional statements

9. Command Pattern

- **Purpose:** Encapsulate requests as objects
- **Use cases:** Undo/Redo functionality, task scheduling, queuing
- **Example:** Remote control, text editor operations

10. State Pattern

- **Purpose:** Allow object to change behavior when internal state changes
 - **Use cases:** Order states, vending machine, TCP connection
 - **Example:** Document (Draft, Moderation, Published states)
-

Phase 4: Practice Problems (Week 6-8)

Beginner Level

1. Parking Lot System

- Vehicle types, parking spots, payment
- Focus: Basic OOP, encapsulation

2. Library Management System

- Books, members, borrowing, returning
- Focus: CRUD operations, relationships

3. ATM System

- Account operations, cash dispensing
- Focus: State pattern, transaction handling

Intermediate Level

4. Chess Game

- Board, pieces, moves, game rules
- Focus: Strategy pattern, complex validation

5. Elevator System

- Multiple elevators, scheduling algorithm
- Focus: State pattern, optimization

6. Movie Ticket Booking System

- Shows, seats, booking, payment
- Focus: Concurrency, reservation handling

Advanced Level

7. Ride-Sharing Application (Uber/Ola)

- Riders, drivers, matching, pricing
- Focus: Multiple patterns, real-time updates

8. Hotel Booking System

- Rooms, reservations, pricing strategies
- Focus: Observer pattern, complex business logic

9. Online Shopping System (E-commerce)

- Products, cart, orders, inventory
 - Focus: Complete system design, scalability
-

Phase 5: System Components (Week 8-10)

Authentication & Authorization

- Login/Signup flows
- Token-based authentication (JWT)
- Role-based access control (RBAC)
- Session management

Rate Limiters

- Token bucket algorithm
- Sliding window
- Fixed window counter
- Use cases: API throttling, DDoS prevention

Caching Layers

- Cache eviction policies (LRU, LFU, FIFO)
- Cache invalidation strategies

- Write-through vs write-back
- Distributed caching

Database Schema Design

- Normalization and denormalization
 - Relationships (one-to-one, one-to-many, many-to-many)
 - Indexing strategies
 - Handling transactions
-

Teaching Tips & Best Practices

1. Show, Don't Tell Approach

- Write bad code together first
- Identify problems collaboratively
- Refactor and improve iteratively
- Makes concepts memorable

2. Pair Programming Sessions

- Work on same problem together
- Let your friend drive sometimes
- Guide through questions, not answers
- Build confidence through practice

3. Ask "Why" Questions

- "Why did you choose this approach?"
- "What happens if requirements change?"
- "How would you test this?"
- Develops critical design thinking

4. Real-World Examples

- Connect every concept to familiar apps
- Observer pattern → YouTube notifications

- Strategy pattern → Google Maps route options
- Singleton → App configuration
- Makes abstract concepts concrete

5. Code Reviews

- Review friend's designs together
- Focus on 1-2 improvements at a time
- Avoid overwhelming with too much feedback
- Celebrate good design decisions

6. Iterative Improvement

- Start with working solution
 - Apply one principle at a time
 - Show evolution, not perfection
 - Design is a journey, not destination
-

Key Evaluation Criteria for LLD

When reviewing designs, check:

- **Correctness:** Does it meet requirements?
 - **Extensibility:** Can new features be added easily?
 - **Maintainability:** Is code clean and understandable?
 - **Scalability:** Will it handle growth?
 - **Testability:** Can components be tested in isolation?
 - **SOLID compliance:** Are principles followed?
-

Common Mistakes to Avoid

1. **Over-engineering:** Don't use patterns unnecessarily
2. **Premature optimization:** Make it work first, optimize later
3. **Ignoring SOLID:** Leads to rigid, fragile code

4. **Poor naming:** Use clear, descriptive names
 5. **Tight coupling:** Makes changes difficult
 6. **Missing error handling:** Always consider edge cases
-

Resources for Practice

Practice Platforms

- LeetCode (Design section)
- InterviewBit (System Design)
- Educative.io courses
- GitHub repositories with LLD examples

Books (Optional)

- "Head First Design Patterns"
 - "Design Patterns: Elements of Reusable Object-Oriented Software"
 - "Clean Code" by Robert Martin
-

Weekly Progress Checklist

- Week 1-2: OOP concepts clear, can explain with examples
 - Week 2-3: Can identify SOLID violations in code
 - Week 3-4: Understands 3-4 design patterns deeply
 - Week 4-5: Can implement patterns from scratch
 - Week 5-6: Comfortable with remaining patterns
 - Week 6-7: Can design simple systems (parking lot, library)
 - Week 7-8: Can design medium complexity systems (chess, elevator)
 - Week 8-9: Can design complex systems with multiple components
 - Week 9-10: Can justify design decisions and handle what-if scenarios
-

Final Notes

Remember: LLD is learned by doing, not just reading. Encourage your friend to:

- Code every example themselves

- Break things and fix them
- Question every design decision
- Practice explaining their designs
- Review open-source code for inspiration

Good luck with teaching! The best teacher learns twice.