System Design & Design Patterns Sheet

> A concise, interview-focused sheet for mastering design patterns, LLD & HLD, and practice problems — modeled like a DSA sheet: theory + code + categorized practice questions (Easy / Medium / Hard) + study plan and templates.

Table of Contents

---

1) Quick Study Plan

- **30 days (Foundations)**: Learn pattern categories, study 8-10 patterns (Singleton, Factory, Builder, Adapter, Decorator, Strategy, Observer, Command). Implement each in Java and solve easy LLD problems.
- **60 days (Apply & Practice)**: Study remaining patterns, implement medium LLD problems (Parking Lot, Elevator, TicTacToe, Thread-safe Singleton, File System), start HLD basics (cache, load balancer, database partitioning).
- **90 days (Mastery & Mock Interviews)**: Hard LLD problems, design 8–10 HLD systems end-to-end (chat, video streaming, notification, ecommerce), mock interviews, and optimizations.

---

2) How to use this sheet

- For each pattern: read intent + problem + UML, type out the Java code snippet, then convert to a small micro-task (write tests or extend to thread-safety).
- For LLD problems: first design classes on paper, state invariants, write core methods, then optimize for concurrency and edge cases.
- Track progress: ✅when you can explain pattern in 2 mins, when you implemented it, ✅✅when you applied it in a real design.

---

3) Design Patterns — Overview & Cheatsheet

- **Creational**: Singleton, Factory Method, Abstract Factory, Builder, Prototype.
- **Structural**: Adapter, Bridge, Composite, Decorator, Facade, Flyweight, Proxy.
- **Behavioral**: Chain of Responsibility, Command, Interpreter, Iterator, Mediator, Memento, Observer, State, Strategy, Template Method, Visitor.

Cheats: If you see *change in object creation* → creational. If you see *change in object composition* → structural. If you see *change in behavior or communication* → behavioral.

---

4) Pattern Template (use in interviews)

1. **Name & Category**
2. **Intent (1 sentence)**
3. **Problem it solves**
4. **Structure / UML (brief)**
5. **Participants**
6. **Key idea**

7. **Simple Java snippet (core parts only)**
8. **When to use / Pros & Cons**
9. **Real-world example**

---

5) Detailed Patterns — (examples)

Singleton (Creational)

Intent: Ensure a class has only one instance and provide a global access point. **Problem:** Global shared resource with single instance (config manager, logger). **Java (thread-safe lazy, double-checked locking):**

```java
public class Singleton {
    private static volatile Singleton instance;
    private Singleton(){}
    public static Singleton getInstance(){
        if(instance == null){
            synchronized(Singleton.class){
                if(instance == null) instance = new Singleton();
            }
        }
        return instance;
    }
}
```

When to use: Shared immutable or thread-safe resources. **Avoid** for testability unless injectable.

Factory Method

Intent: Define an interface for creating an object, but let subclasses decide which class to instantiate. **Example Java skeleton:**

```java
interface Button { void render(); }
class WindowsButton implements Button { public void render(){ /*...*/ }}
class LinuxButton implements Button { public void render(){ /*...*/ }}
abstract class Dialog {
    abstract Button createButton();
    public void renderWindow(){ Button ok = createButton(); ok.render(); }
}
class WindowsDialog extends Dialog{ Button createButton(){ return new
WindowsButton(); }}
```

Builder

Intent: Build complex object step-by-step. **When:** Many constructor parameters or optional params.

```
class User {
  private final String name; private final int age; private final String
address;
  private User(Builder b){ name = b.name; age = b.age; address = b.address; }
  public static class Builder{
    private String name; private int age; private String address;
    public Builder name(String n){ this.name = n; return this; }
    public Builder age(int a){ this.age = a; return this; }
    public User build(){ return new User(this); }
  }
}
// Usage: User u = new User.Builder().name("S").age(25).build();
```

(For the full document: each pattern above has the same structure: intent, problem, UML, Java snippet, when to use, variations, pitfalls.)

---

6) LLD Interview Problems — Practice (Easy / Medium / Hard)

Easy

- Logger (Singleton + file rotation) — implement rotate
- LRU Cache (classic) — implement using LinkedHashMap or DoublyLinkedList + HashMap
- TicTacToe — Implement winner check, AI optional
- URL shortener (basic) — encode/decode
- Rate limiter (fixed window) — simple token bucket

Medium

- Parking Lot system — design classes, parking strategies, billing
- Elevator System — scheduling, concurrency
- File system (in-memory) — create/delete, ls, move, path handling
- Chat server (text only) — rooms, user sessions, message broadcast
- Bank account system — transactions, rollback, concurrency

Hard

- Distributed Lock Manager (Zookeeper style) — fairness, leases
- Messaging system with persistence & consumer groups (Kafka-like) — design partitions, retention
- Full URL shortener at scale (HLD) — partitioning, vanity URLs, analytics
- Design a social feed (newsfeed) — fanout vs pull, caching, ranking

- Design an autoscaling service for streaming ingestion

7) HLD Topics & Common Questions

- Load balancing strategies (round robin, least connections, IP hash)
- Caching: TTL, invalidation, cache-aside, write-through, write-back
- Databases: vertical vs horizontal scaling, sharding, replication (master-slave, multi-master), CAP theorem
- Queues & messaging: push vs pull, at-least-once vs exactly-once
- Storage: object store vs block store, CDN patterns
- Consistency models, consensus algorithms (Raft/Paxos high-level)
- API Gateway, Authentication (JWT vs OAuth2) patterns

8) System Design Checklist & Tradeoffs (short)

- Requirements: functional vs non-functional → list & prioritize
- Capacity estimation & load calculation (QPS, payload size, concurrency)
- Data modeling & partitioning plan
- Bottlenecks & single points of failure
- Choice of DB, cache, queue, CDN
- Monitoring & Observability (metrics, logs, tracing)
- Security & privacy considerations

9) How to approach an interview design question (script)

1. Clarify requirements (functional + non-functional)
2. Ask about scale & constraints (QPS, data growth, latency)
3. Sketch a high-level architecture (components)
4. Pick one or two components and deep-dive (APIs, data model, schemas)
5. Address bottlenecks and tradeoffs
6. Summarize and mention improvements/edge-cases

Use the STAR-like structure for answers: Situation → Task → Action → Result.

10) Example end-to-end walkthroughs (short index)

- Parking Lot (LLD) — includes classes: ParkingLot, Floor, Spot, Vehicle, ParkingStrategy, Ticket, Billing
- URL Shortener (HLD+LLD) — DB schema, hash function, collision handling, redirect flow
- Chat System (HLD) — Authentication, persistent storage, WebSockets, presence, scaling
- Notification Service — worker pool, retries, deduplication, rate limits

(Each walkthrough contains diagrams, class sketches, sample APIs and complexity analysis.)

11) Reference implementations & further reading

- *Design Patterns* — Erich Gamma et al.
- *Patterns of Enterprise Application Architecture* — Martin Fowler
- *System Design Interview — An insider's guide* — Alex Xu
- Public repos: GitHub pattern collections, microservices samples

12) Appendix

- UML mini-primer (class, sequence, component)
- Quick glossary (idempotent, eventual consistency, partition tolerance, CAP)
- Common pitfalls & anti-patterns

Next steps I can do for you (pick one)

- Convert this sheet into a printable PDF or Google Doc.
- Create a progressive checklist with daily tasks for 30/60/90 days.
- Generate 10 LLD problems with starter templates and test cases (Easy/Medium/Hard).
- Provide full Java implementations for selected patterns or problems.

Created for you. Modify or ask for expansions — e.g., full Java solutions for selected patterns, printable PDF, or interactive checklist.