Design Patterns (LLD-Focused) Cheat Sheet

CREATIONAL PATTERNS

1) Singleton

Problem: Need exactly one instance (config, DB pool).

Idea: Private ctor + global access + thread-safe init.

Code:

```
public final class Config {
  private static volatile Config INSTANCE;
  private Config() {}
  public static Config getInstance() {
    if (INSTANCE == null) {
      synchronized (Config.class) {
        if (INSTANCE == null) INSTANCE = new Config();
      }
    }
    return INSTANCE;
  }
}
```

2) Factory Method

Problem: Let subclasses decide which product to create.

Idea: Define create() in base, override in concrete factories.

3) Abstract Factory

Problem: Create families of related objects together.

Example: UI kit (Dark/Light).

4) Builder

Problem: Construct complex objects step-by-step.

Example: Building a Loan Schedule row.

5) Prototype

Problem: Clone existing objects (cheap copies).

STRUCTURAL PATTERNS

6) Adapter

Problem: Incompatible interfaces; make legacy fit new.

7) Decorator

Problem: Add behavior dynamically without subclass explosion.

8) Facade

Problem: Hide complex subsystem behind a simple API.

9) Proxy

Problem: Control access - lazy load, caching, security.

10) Composite

Problem: Treat part-whole uniformly (trees).

BEHAVIORAL PATTERNS

11) Strategy

Problem: Swap algorithms at runtime.

12) Observer (Publish-Subscribe)

Problem: Notify dependents on state change.

13) Command

Problem: Encapsulate requests as objects - undo, queue, log.

14) Chain of Responsibility

Problem: Pass request along handlers till one handles it.

15) Template Method

Problem: Fixed algorithm skeleton with overridable steps.

16) State

Problem: Behavior changes with internal state.

HOW TO PICK PATTERNS IN LLD INTERVIEWS

- Start from use-cases & constraints.
- Show class roles & interactions.
- Call out trade-offs: coupling, complexity, testability.
- Demonstrate evolution: new Strategy + Factory with no changes to callers.

Quick combos:

- Strategy + Factory
- Decorator + Facade
- Observer + Command
- State + Chain