Software Design Patterns & Architecture Cheat Sheet

1. Creational Patterns

```
Factory Method:
  Interface for creating objects, subclasses decide which class to instantiate.

Singleton:
  Ensure a class has only one instance.
  Example (Python):
    class Singleton:
        _instance = None
        def __new__(cls):
            if not cls._instance:
                cls._instance = super().__new__(cls)
            return cls._instance

Builder:
  Separate construction of complex object from representation.

Prototype:
  Clone existing object instead of creating from scratch.
```

2. Structural Patterns

```
Adapter:
  Convert one interface to another the client expects.

Decorator:
  Add responsibilities to objects dynamically.

Facade:
  Simplify interaction with complex systems (wrap subsystems).

Composite:
  Treat group of objects as a single instance (tree structures).

Proxy:
  Provide a placeholder for another object to control access.
```

3. Behavioral Patterns

```
Observer:
  One-to-many dependency between objects (e.g., event listeners).

Strategy:
  Define family of algorithms, make them interchangeable.

Command:
  Encapsulate request as an object (queue, log, undo).
```

State:
  Alter behavior when internal state changes (finite state machines).

Mediator:
  Define object that handles communication between others (chat room).

4. Architectural Patterns

MVC (Model-View-Controller):
  Separates data (model), UI (view), logic (controller).

MVVM (Model-View-ViewModel):
  ViewModel binds model to view (used in modern frontend frameworks).

Microservices:
  Decouple large app into small, independent services.

Monolith:
  All components deployed as one single unit.

Event-Driven:
  Publish/subscribe or messaging based; reactive systems.

Hexagonal (Ports and Adapters):
  Core logic isolated from external interfaces via ports.

5. Modern Patterns & Practices

Domain-Driven Design (DDD):
  Model complex domains with bounded contexts and ubiquitous language.

CQRS:
  Separate read and write operations into different models.

Event Sourcing:
  Persist events rather than current state.

Circuit Breaker:
  Prevent calls to a failing service until it recovers.

Bulkhead:
  Isolate failures in parts of the system to prevent cascade.

Service Mesh:
  Infrastructure layer for microservice communication (e.g., Istio).

6. SOLID Principles

S - Single Responsibility:

A class should have one reason to change.

O - Open/Closed:
Open for extension, closed for modification.

L - Liskov Substitution:
Subtypes should be substitutable for base types.

I - Interface Segregation:
No client should depend on unused methods.

D - Dependency Inversion:
Depend on abstractions, not concrete implementations.

7. Anti-Patterns (What to Avoid)

- God Object: Class doing too much
- Spaghetti Code: Complex and tangled logic
- Golden Hammer: Using one tool for all problems
- Lava Flow: Unused legacy code that persists
- Copy-Paste Programming: Code duplication instead of reusability