**Design Patterns Cheat Sheet**

**Template Method** (Behavioral)

Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure. (.NET example 5, Spring: JdbcTemplate)

**Factory Method** (Creational)

Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses. (.NET example 6)

**Strategy** (Behavioral)

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it. (.NET example 7, Java: Collections.sort(lst, Comparator<T>))

**Builder** (Creational)

Separate the construction of a complex object from its representation so that the same construction process can create different representations. (.NET/Java: StringBuilder)

**Abstract Factory** (Creational)

Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

**Decorator** (Structural)

Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality. (.NET/Java: Streams)

**Composite** (Structural)

Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly (ex: FileSystem with Directories/Files)

**Visitor** (Behavioral)

Represent an operation to be performed on the elements of an object structure. Visitor lets you define a new operation without changing the classes of elements on which it operates. (.NET: Roslyn)

**Chain Of Responsibility** (Behavioral)

Avoid coupling the sender of a request to its receiver by giving more than one object the chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it (JavaScript: DOM events)

**Mediator** (Behavioral)

Define an object that encapsulates how a set of objects interact. Mediator promotes loose coupling by keeping objects from referring to each other explicitly, and it lets you vary their interaction independently.

**Memento** (Behavioral)

Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

**Observer** (Behavioral)

Define a one-to-many dependency between objects so that when one object changes, all its dependents are notified and updated automatically. (.NET example 2.1, Spring: ApplicationContext's events)

**State** (Behavioral)

Allow an object to alter its behavior when its internal state changes. The object will appear to change its class. (ex: An IoT device)

**Adapter** (Structural)

Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.

## Bridge (Structural)

Decouple an abstraction from its implementation so that the two can vary independently.