“Gang of Four” Design Pattern Categories

Behavioral Design Patterns

Chain of Responsibility: avoids coupling the sender of a request to its receiver by giving more than one object a chance to handle a request. Chains the receiving objects and pass the request along the chain until an object handles it.
Command: encapsulates a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.
Interpreter: given a language, defines a representation for it grammar along with an interpreter that uses the representation to interpret sentences in the language.
Iterator: provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
Mediator: defines an object that encapsulates how a set of objects interact. **Mediator** promotes loose coupling by keeping objects from referring to each other explicitly, and lets you vary their interaction independently.
Memento: without violating encapsulation, captures and externalizes an object’s internal state so that the object can be restored to this state later.
Observer: defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
State: allows an object to alter its behavior when its internal state changes. The object will appear to change its class.
Strategy: defines a family of algorithms, encapsulates each one, and makes them interchangeable. **Strategy** lets the algorithm vary independently from clients that use it.
Template Method: defines 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.
Visitor: represents an operation to be performed on the elements of an object structure. **Visitor** lets you define a new operation without changing the classes of the elements on which it operates.

Creational Design Patterns

Abstract Factory: provides an interface for creating families of related or dependent objects without specifying their concrete classes.
Builder: separates the construction of a complex object from its representation so that the same construction process can create different representations. **Builder** allows an object to be constructed in pieces rather than all at once.
Factory Method: defines an interface for creating an object, but lets subclasses decide which class to instantiate. **Factory Method** lets a class defer instantiation to subclasses.
Prototype: specifies the kinds of objects to create using a prototypical instance, and creates new objects by copying this prototype.
Singleton: ensures a class only has one instance, and provides a global point of access to it.

Structural Design Patterns

Adapter: converts the interface of a class into another interface that clients expect. **Adapter** lets classes work together that couldn’t otherwise because of incompatible interfaces.
Bridge: decouples an abstraction from its implementation so that the two can vary independently.
Composite: composes objects into tree structures to represent part-whole hierarchies. **Composite** lets clients treat individual objects and compositions of objects uniformly.
Decorator: attaches additional responsibilities to an object dynamically. **Decorators** provide a flexible alternative to subclassing for extending functionality.
Façade: provides a unified interface to a set of interfaces in a subsystem. **Façade** defines a higher-level interface that makes the subsystem easier to use.
Flyweight: uses sharing to support large numbers of fine-grained objects efficiently.
Proxy: provides a surrogate or placeholder for another object to control access to it.

Flyweight: uses sharing to support large numbers of fine-grained objects efficiently.
Proxy: provides a surrogate or placeholder for another object to control access to it.