Compositional Design Principles

Principles for Flexible Design:

① *Program to an interface, not an implementation.*

② *Favor object composition over class inheritance.*

③ *Consider what should be variable in your design.*
(or: Encapsulate the behavior that varies.)

Design Pattern Index

Note: The page number refers to the page of the pattern overview box, not the first page of the chapter in which the pattern first appears.

ABSTRACT FACTORY: *Provide an interface for creating families of related or dependent objects without specifying their concrete classes.* Page 217.

ADAPTER: *Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.* Page 295.

BUILDER: *Separate the construction of a complex object from its representation so that the same construction process can create different representations.* Page 301.

COMMAND: *Encapsulate a request as an object, thereby letting you parameterize clients with different requests, queue or log requests, and support undoable operations.* Page 308.

COMPOSITE: *Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.* Page 322.

DECORATOR: *Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.* Page 289.

FACADE: *Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use.* Page 282.

ITERATOR: *Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.* Page 312.

MODEL-VIEW-CONTROLLER: *Define a loosely coupled design to form the architecture of graphical user interfaces having multiple windows and handling user input from mouse, keyboard, or other input sources.* Page 342.

NULL OBJECT: *Define a no-operation object to represent null.* Page 325.

OBSERVER: *Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.* Page 335.

PROXY: *Provide a surrogate or placeholder for another object to control access to it.* Page 317.

STATE: *Allow an object to alter its behavior when its internal state changes.* Page 185.

STRATEGY: *Define a family of business rules or algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithms vary independently from clients that use it.* Page 130.

TEMPLATE METHOD: *Define the skeleton of an algorithm in an operation, deferring some steps to subclasses or delegates. Template Method lets the behavior of certain steps of an algorithm be varied without changing the algorithm's structure.* Page 366.