- **Group of Four Design Patterns**

  Design Patterns are a software engineering concept describing recurring solutions to common problems in software design.

  The authors Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides are often referred to as the GoF, or Gang of Four.

  The site explores the capabilities and pitfalls of object-oriented programming and describes several software design patterns.

  Part of the appeal of design patterns is that they can be used uniformly over many different languages and syntaxes.

  The basic structure stays the same; only the details change

  Creational Patterns

  1.**Abstract Factory**: Creates an instance of several families of classes. Provide an interface for creating families of related or dependent objects without specifying their concrete classes.

  2.**Builder**: Separates object construction from its representation. Separate the construction of a complex object from its representation so that the same construction processes can create different representations.

  3.**Factory Method**: Creates an instance of several derived classes. Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

  4.**Prototype**: A fully initialized instance to be copied or cloned. Specify the kinds of objects to create using a prototypical instance, and create new objects by copying this prototype.

  5.**Singleton:** A class of which only a single instance can exist. Ensure a class only has one instance, and provide a global point of access to it.

  Structural Patterns

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

  2.**Bridge**: Separates an object's interface from its implementation. Decouple an abstraction from its implementation so that the two can vary independently.

  3.**Composite:** A tree structure of simple and composite objects. Compose objects into tree structures to represent part-whole hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

4.**Decorator**: Add responsibilities to objects dynamically. Attach additional responsibilities to an object dynamically. Decorators provide a flexible alternative to subclassing for extending functionality.

5.**Facade:** A single class that represents an entire subsystem. Provide a unified interface to a set of interfaces in a system. Facade defines a higher-level interface that makes the subsystem easier to use.

6.**Flyweight:** A fine-grained instance used for efficient sharing. Use sharing to support large numbers of fine-grained objects efficiently. A flyweight is a shared object that can be used in multiple contexts simultaneously. The flyweight acts as an independent object in each context — it's indistinguishable from an instance of the object that's not shared.

7.**Proxy:** An object representing another object. Provide a surrogate or placeholder for another object to control access to it.

Behavioral Patterns

1.**Chain of Resp**. : A way of passing a request between a chain of objects. Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request. Chain the receiving objects and pass the request along the chain until an object handles it.

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

3.**Interpreter**: A way to include language elements in a program. Given a language, define a representation for its grammar along with an interpreter that uses the representation to interpret sentences in the language.

4.**Iterator**: Sequentially access the elements of a collection. Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

5.**Mediator**: Defines simplified communication between classes. 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.

6.**Memento**: Capture and restore an object's internal state. Without violating encapsulation, capture and externalize an object's internal state so that the object can be restored to this state later.

7.**Observer**: A way of notifying change to a number of classes. Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.

8.**State**: Alter an object's behavior when its state changes. Allow an object to alter its behavior when its internal state changes. The object will appear to change its class.

9.**Strategy**: Encapsulates an algorithm inside a class. Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

10.**Template:** Defer the exact steps of an algorithm to a subclass. 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.

11.**Visitor:** Defines a new operation to a class without change. 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 the elements on which it operates.

- **Introduction to Enterprise Patterns**
- **Windows Production .NET Debugging**

  Debugging tools for Windows

  WinDBG

  Tools and utilities

  In addition to the debuggers, Debugging Tools for Windows includes a set of tools that are useful for debugging.

  - Getting Started with Windows Debugging
  - Debugging Resources
  - Debugger Operation
  - Debugging Techniques
  - Symbols
  - Crash Dump Analysis
  - Bug Checks (Blue Screens)
  - Debugger Reference