Pattern:

A Pattern is a solution to a problem in a context

- The **context** is the situation in which pattern applies. this should be recurring situation
- The **problem** refers to the goal you are trying to achieve in this context, but it also refers to any constraints that occur in the context.
- The **solution** is what your are after: a general design that anyone can apply which resolves goal and set of constraints

> "If you find yourself in a context with a problem that has goal that it affected by a set of constraints, then you can apply design that resolves the goal and constraints and leads to a solution."

Patterns are proven object-oriented experience. Patterns are not invented the are discovered. Patterns provide a shared language that can maximize the value of your communication with other developers

1. Strategy Pattern:

The Strategy Pattern defines family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from the clients that use it.

2. Observer Pattern:

The Observer Pattern defines a one-to-many dependency between objects so that when one object changes state all dependent are notified and update automatically.

3. Decorator Pattern:

The Decorator Pattern attaches additional responsibilities to an object dynamically. Decorator provide flexible alternative to subclassing for extending functionality.

4. Factory Method Pattern:

The Factory Method Pattern defined an interface for creating an object, but lets subclass decide which class to instantiate, Factory method lets a class differ instantiation to subclass.

5. Abstract Factory Pattern:

The Abstract Factory Pattern provides an interface for creating families of related or dependent objects without specifying their concrete class

6. Singleton Pattern:

The Singleton Pattern ensures a class has only one instance, and provides a global point of access to it.

7. Command Pattern:

The command pattern encapsulates a request as an object, there by letting you parameterize other objects with different requests, queue or log requests, and support undoable operations.

8. Adapter Pattern:

The Adapter Pattern converts the interface of a class into another interface the client expects. Adapter lets class work together that couldn't otherwise because of incompatible interface.

9. Facade Pattern:

The Facade Pattern provides a unified interface to a set of interface in a subsystem, Facade defines a higher-level interface that makes the subsystem easier to use.

10. Template Method Pattern:

The Template Method Pattern defined the skeleton of the algorithm in a method, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of algorithms without changing the algorithm's structure.

11. Iterator Pattern:

The Iterator Pattern provides a way to access the elements of an aggregate object sequentially without exposing its underlying representation.

12. Composite Pattern:

The Composite Pattern allows you to compose objects into tree structure to represent part-while hierarchies. Composite lets clients treat individual objects and compositions of objects uniformly.

13. State Pattern

The State Pattern allows an object to alter its behavior when its internal stage changes. The object will appear to change its class.

14. Proxy Pattern

The Proxy Pattern provides a surrogate or placeholder for another object to control access to it.

15. Compound Pattern

The Compound Pattern combines two or more patterns into solution that solves recurring or general problem.

16. Bridge Pattern

The Bridge Pattern allows you to vary the implementation and abstraction by placing the separate class hierarchies.

17. Builder Pattern