**Strategy pattern** defines a family of algorithms, encapsulates each one, makes them interchangeable. Strategy lets algorithm vary independently from clients that use it.

*Encapsulates interchangeable behaviors and uses delegation to decide which one to use.*

1. Identify the aspects of your application that vary and separate them from what stays the same
2. Program to an interface, not an implementation
3. Favor composition over inheritance

**Observer Pattern** defines a one-to-many dependency between a set of objects so that one object changes state, all other depending objects are notified and updated automatically.

*Allows objects to be notified when state changes*

1. Strive for loosely coupled design between objects that interact

**Decorator Pattern** attaches additional responsibility to an object dynamically, it provides flexible alternative to subclassing for extending functionality.

*Wraps an object to provide new behavior*

1. Classes should be open for extension, but closed for modification.

**Factory method pattern** defines an interface for creating an object, but let’s subclasses decide which class to instantiate. Factory method lets a class defer instantiation to subclasses

*Subclasses decide which concrete class to create*

1. Depend upon abstraction; do not depend upon concrete classes.

**Abstract factory** **pattern** provides an interface for creating families of related or dependent objects without specifying their concrete classes.

*Allows a client to create families of objects without specifying their concrete classes*

**Singleton pattern** ensures a class has only one instance, and provides a global point of access.

*Ensures one and only object is create*  
**Command pattern** encapsulate a request as an object thereby letting you parameterize other objects with different requests, queue or log requests, and support undoable options.

*Encapsulate a request as an object*

## Bridge pattern

## Builder pattern

**Facade pattern** provides a unified interface to a set of interface in a subsystem. Facade defines a higher level of interface that makes the subsystem easier to use.

*Simplifies the interface of a set of classes*

**Adapter pattern** convertsthe interface of a class into another interface the client expects. Adapters let classes to work together that couldn’t otherwise because of incompatible interfaces.

*Wraps an object and provides different interface to it*

**Template pattern** defines the skeleton of the algorithm in a method, deferring some steps to subclasses. Template method let subclasses to redefine certain steps of the algorithm without changing the algorithms structure.

*Subclasses decide how to implement steps in an algorithm.*

1. Don’t call us, we will call you

**Iterator pattern** provides the way to access the elements of an aggregate object sequentially without exposing its underlying representation.

*Provides a way to traverse a collection of objects without exposing its implementation*

1. A class should have only one reason to change

**Composite pattern** allows you to compose objects into tree structures to represent part-whole hierarchies. Composite lets client to treat individual objects and composition of objects uniformly.

*Client treats collection of objects and individual objects uniformly*

**State pattern** allows an object to alter its behavior when it’s internal state changes. The object will appear to change its class.

*Encapsulate state behavior and uses delegation to switch between behavior.*

**Proxy pattern** provides a surrogate or placeholder for another object to control access to it.

*Wraps an object to control access to it*