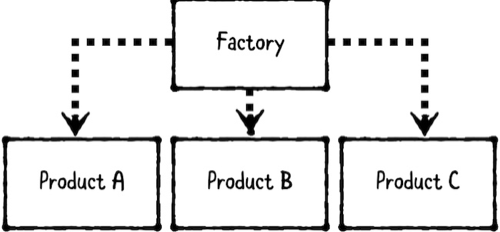
Singleton

The singleton pattern guarantees that only one instance of a class is instantiated. There are times that you want to make sure only one instance of a class is instantiated and that your application only uses that instance. That's the primary and only goal of the singleton pattern.

Factory

The factory pattern is a creational pattern that provides a way to make objects without

exposing creation logic.



Use the factory pattern whenever you want to separate out product creation logic,

instead of having consumers create products directly.

A factory is very useful when you have a group of related products, such as polymorphic

subclasses or several objects that implement the same protocol. For example, you can

use a factory to inspect a network response and turn it into a concrete model subtype.

Facade

Like a building. this design pattern hides all the complexities of the building and displays a friendly face. Facade can be recognized in a class that has a simple interface, but delegates most of the work to other classes. Usually, facades manage the full life cycle of objects they use.

The advantages of façade can be summarized as follow:

* Makes complex APIs easier to read, understand, and use, because their complexity is

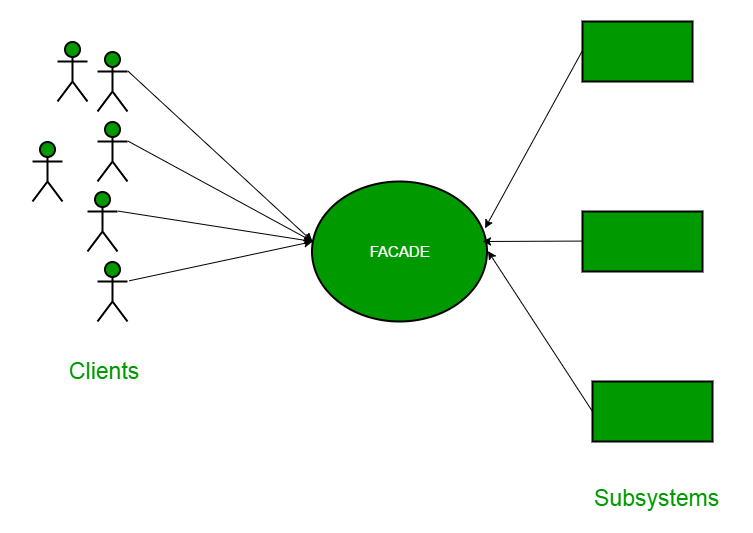
wrapped up in simpler methods.

* Reduces dependencies on the internals of those APIs, which allows you to change them

freely as long as the end result does the same job.

* Lets you wrap a bad architecture in something that works better based on experience, or a

complex architecture in something simpler that does the job for most people.



Decorator

Decorator is a structural pattern that allows adding new behaviors to objects dynamically by placing them inside special wrapper objects. Decorators offer a flexible way to add responsibility to individual objects. Unlike inheritance, decorated objects are not limited by their parent classes. Putting in other terms, a client has control over how and when to decorate the component.

There are few characters of decorator pattern:

* The decorators have the same super-type or conformance as the object they decorate.
* More than one decorator can be used to comprise an object.
* Once the decorator has the same super-type or conformance as its decorated object (decoratee), it is possible to pass a decorated object in place of the original object.
* The decorator adds its own behavior before or after delegating the decoratee its work.
* The objects can be decorated at any moment, which makes it possible to decorate them dynamically at run-time, with as many decorators as needed.

Delegation

A delegate is one object that is asked to respond to events that happen to another object, or to

guide its behavior. At its core, delegation makes it easy to customize behavior of objects while also minimizing coupling. Delegation design pattern is widely used in iOS applications.

Adapter

Adapter is a structural design pattern that allows objects with incompatible interfaces to work together. In other words, it transforms the interface of an object to adapt it to a different object.

An adapter wraps an object, therefore concealing it completely from another object. For example, you could wrap an object that handles meters with an adapter that converts data into feet. The adapter design pattern can be used when you want to use a third-party class but its interface doesn’t match the rest of your application’s code; or when you need to use several existing subclasses but they lack particular functionality and, on top of that, you can’t extend the superclass.