Design Patterns Assignment

1. Singleton (Creational)

Singleton is a creational design pattern, which ensures that only one object of its kind exists and provides a single point of access to it for any other code.

WTUI:

- When a class in your program should have just a single instance available to all clients
- To decouple the logic of creation of an object from its usage. It helps reduce complexities by creating objects in a controlled manner with already mentioned tried & tested solutions.

2. Factory Method (Creational)

The Factory Method is a creational design pattern that just abstracts the creation of an object. Any instance returned by the factory would be of type Interface, in swift protocol, which any factory candidate class must have implemented.

WTUI:

- When you want the type of the object created to be determined at runtime.
- When you want the codebase more flexible to add or remove new types

3. Facade (Structural)

The Facade pattern is a structural design pattern that provides a simple interface to a library, framework, or complex system of classes. The Facade pattern provides a simple interface for working with complex subsystems containing lots of classes. The Facade pattern offers a simplified interface with limited functionality that you can extend by using a complex subsystem directly. This simplified interface provides only the features a client needs while concealing all others.

WTUI:

- When you want to provide a simple or unified interface to a complex system
- When you need to decompose a subsystem into separate layers

4. Decorator (Structural)

The Decorator pattern is a structural design pattern that allows you to dynamically attach new functionalities to an object by wrapping them in useful wrappers. You place a target object inside another wrapper object that triggers the basic behavior of the target object and adds its own behavior to the result. Both objects share the same interface, so it doesn't matter for a user which of the objects they interact with, either: clean or wrapped. You can use several wrappers simultaneously and get the combined behavior of all these wrappers.

WTUI:

- When you want to add responsibilities to objects dynamically and conceal those objects from the code that uses them
- When it's impossible to extend responsibilities of an object through inheritance

5. Builder (Creational)

The Builder pattern is a creational design pattern that allows you to create complex objects from simple objects step by step. This design pattern helps you use the same code for creating different object views. The Builder design pattern calls for separating the construction of an object from its own class. The construction of this object is instead assigned to special objects called builders and split into multiple steps. To create an object, you successively call builder methods, and you don't need to go through all the steps, only those required for creating an object with a particular configuration.

WTUI:

- When you want to avoid using a telescopic constructor, when a constructor has too many parameters, it gets difficult to read and manage
- When your code needs to create different views of a specific object
- When you need to compose complex objects

6. Template Method (Behavioral)

The Template Method pattern is a behavioral design pattern that defines a skeleton for an algorithm and delegates responsibility for some steps to subclasses. This pattern allows subclasses to redefine certain steps of an algorithm without changing its overall structure. This design pattern splits an algorithm into a sequence of steps, describes these steps in separate methods, and calls them consecutively with the help of a single template method.

WTUI:

- When subclasses need to extend a basic algorithm without modifying its structure
- When you have several classes responsible for quite similar actions, meaning that whenever you modify one class, you need to change the other classes