**What is a Design Pattern**

Design patterns can be structural, behavioral, or creational. A design pattern is a reusable answer to a common reoccurring problem. Design patterns have many benefits, but the 3 most important ones are that they have been proven solutions, can easily be reused, and that patterns can be expressive. Now even though I said that patterns are solutions, they are not always the exact solution to a problem. They don't solve all problems they are good however for supporting software designers. Some of the advantages of design patterns are:

* The fact that patterns are reusable they help to prevent minor issues which could lead to major problems in development
* Patterns are generalized therefore they do not have to be associated with a specific problem
* Some patterns can even decrease the size of the file due to the fact that they can help to prevent repetition
* Patterns help to make communication more effective

**Types of Design Patterns**

**Constructor Pattern**: is a method that is used to initialize a new created specific types of object once memory has been allocated for it. Objects will be able to accept arguments which can be used to set values.

**Modules Pattern:** this pattern is used to help keep units of code cleanly and separated organized. It can be done by using object literal notation, AMD modules, CommonJS Modules, or ECMAScript Harmony modules. Designed to create classes that are either private or public

**The Revealing Module Pattern:** This is an updated version of the Modules Pattern. It allows us to be able to define all of our functions and variables in the private scope and return and anonymous object that points to private functions.

**Singleton Pattern:** this pattern restricts the instantiation of a class to a single object. It makes it so an object does not have to be created every time an event occurs. It serves as a shared namespace so that the implementation code is isolated from the global namespace.

**The Observer Pattern:** this pattern is where an object maintains a list of objects that depend on it and this will notify those objects when there is any change in state. For this pattern there is a Subject, Observer, ConcreteSubject, and ConcreteObserver. The subject is what maintains the list and the observer updates.

**Mediator Pattern:** this is an object that coordinates interactions, both logic and behavior, between multiple objects. It decides when to call certain objects based on actions and inputs.

**Prototype Pattern:** we can create objects that can act as protoypes for other objects. It will be used as a blueprint for each object that the constructor creates. Any object that is created by the constructor will have the same properties. Allows for ease of implementing inheritance, as well as possibly boosting performance.

**Command Pattern:** this pattern encapsulates method invocation, requests, or operations and condense them into a single object. It also allows for the decoupling of objects which gives us more flexibility when swapping classes.

**Facade Pattern:** this pattern helps to conceal the overall complexitiy of the code. It provides us with a higher lever interface for a larger body of code. It is a structural pattern.

**Factory Pattern:** this is a type of creational pattern that has to deal with the notation of objects. It is useful in the creation of objects, especially if the process for creating that object is complex. It provides a generic interface for creating these objects so that we can specify the type of factory object we want to create.

**Mixin Pattern:** these help with the functionality of class so that the process of inheritance is easier.

**Decorator Pattern:** this is a structural design pattern that promotes code re-use. It is kind of similar to a Mixin. The allow us to dynamically change existing classes.

**Flyweight:** this is another structural pattern that is used to optimization of code that is repetitive. It wants to minimize the amount of memory that is used by sharing as much data as possible. You just need to take several similar objects or data constructs and then placing this data into a single external object.

**My Patterns**

**Constructor and Decorator Patterns**

The constructor creates a function named Patient that has three parameters, fName, lName, and Dob. Next I created two instances of this function that is then decorated with a setAge and setWeight function for both of the objects. Next all the information is then printed to the screen .

**Facade and Flyweight Patterns**

The facade pattern has the flyweight pattern embedded in the sick: and healthy: functions. Here a Patient function is created that has a fName and lName for the parameters. When the facade is called it out it runs the args through a conditional statement and then returns the patient first and last name and their health status based on which conditions are satisfied.

**Mediator and Command Patterns**

A function called Jason is created and within this function there is a variable called schooler that has requestInfo inside of it. There are two events inside, complete and save. The commands then execute both the save and complete with the give data.