Anando Zaman

General Design Pattern Notes:

A pattern describes a problem which repeatedly occurs, and then describes the core of the soln to that problem.

It's basically a general solution or best practice to solve commonly recurring problems with specific context.

Classifications of patterns:

- Creational
- Structural
- Behavioural

Creational: Deals with object creation that serves as a solution for a problem

Examples include:

- Abstract Factory
- Object Pool
- Prototype
- Singleton

Structural: are concerned with how classes and objects are composed to form larger structures.

Examples include:

- Adapter
- Bridge
- Façade
- Proxy

Behavioral: describe interactions between objects and how they interact with each other. Used to make algorithm a class uses simply another parameter at runtime. Typically concerned with algorithms and assignment of responsibilities between objects.

Examples include:

- Command
- Iterator
- Observer
- State
- Strategy

Strategy Design Pattern(Behavioural): Defines a family of related algorithms, encapsulates each, and makes them interchangeable. So a set of algos that can be used interchangeably. Code receives runtime instructions specifying which to use, instead of only having one option. This can be great for times where maybe you want to sort an array for example. There are many sorting algorithms but if the array is small in size, it may be fine to use something like bubble sort or selection sort. However, if the array is large in size, then it may be better to use mergesort or quicksort due to the linearithmic time complexity caused by divide-and-conquer strategy to speed up performance. Being able to pick the algorithm dynamically like this at runtime can be quite useful. Captures abstraction in interface while burying implementation details in derived class.

Recall abstraction refers to having access to thing that are important(ie; important methods) to the public while hiding the rest that is NOT as useful.

MVC – Mode View Controller:

- Used commonly in video game design and mobile applications
- Separate computational elements(backend) from I/O.
- 3 components:
- Model encapsulates(hides info) system's data as well as operations on the data.
- View displays data from model components.
- Controller handles flow of activity/inputs. Used as an intermediate link between model and view.
- Controller may or may not depend on state of model. I like to think of it kind of like an intermediate API gateway that updated the View and model by having passing the information between them
- Real life example: Web applications. The web browser(UI) is the view, the web server is the controller while the database server is the model where the backend actions take place
- So Flask could be a web server, MySQL could be a database server, React Frontend with HTML/CSS could be the web-browser.

Proxy Pattern:

- Is kind of like an intermediary or middle man when direct access is not possible or is not the best approach.
- A client communicates with a representative or intermediate rather than the component itself. This represents the proxy. It does any pre and post processing as needed. For example; a bank cheque could be considered a proxy to you receiving your money as it is issued to your bank account rather than given to you directly as cash by your manager/boss.

Adapter:

- Allows incompatible classes to work together by converting interface of one class into another. For example; if we require JSON input for one method but another method outputs XML, we will need an adapter to parse XML to JSON.

Singleton:

- Limits creation of a class to only one object. This is beneficial when one object is needed to coordinate actions across the system. They are static classes or methods accessible anywhere without instantiation.

Factory method:

- Is a creational pattern.
- Creates instance of several derived classes.
- It produces objects without specifying exact class of the object to be created. Basically how you use the 'Object' or "T" keyword for generics as the data type is not defined specifically.
- If we specified the class name specifically instead, it could tightly couple the code.

Observer Design:

- When one object(the subject) changes state, all its dependents(observers) are notified typically by calling one of their methods. For example, if you follow someone on twitter, you are the observer. The person you are following is the subject. Observers are free to 'follow' other people(subjects) too.

Builder:

- Builds objects. Simplifies complex modular systems.
- A UML of composite or aggregate is usually produced.

State:

- Encapsulates the various states a machine can be in and allows an object to alter its behaviour when its initial state changes.