General Introduction To Design Patterns

Note-Taking Application
Name: Kartikeya Saxena
Roll Number: 180101034

- **Introduction :-**
  The User Centric Software design is a dynamic process with lots and lots of iterations. It is a highly structured process with proper languages like **UML (Unified Modeling Language)** and **DFD (Data Flow Diagram)** to precisely describe all aspects of the software. But to effectively use these languages, we need to be creative. A good software design is not a straightforward process and requires past experience from trial-and-error. Software engineers could therefore get overwhelmed and lost in this design process. To help them save time and provide an organisation to the designing process, design patterns were introduced.

- **What are design patterns and why should we use them? :-**
  Design patterns help provide a compass for software engineers to navigate through the complicated task of designing systems. A design pattern is basically a collection of proven good design solutions to commonly occurring problems in software design.

- **Advantages and Disadvantages of using a design pattern :-**

  - **Advantages:**
    - Design patterns provide structure and organisation to the design making it more robust and flexible to future changes in it.
    - They make it easier and faster for software engineers to develop software designs.
    - They provide standard procedures to tackle problems thereby reducing the chances of errors in the design.
    - They help improve design readability by allowing the engineers to communicate and express their design properly and uniformly in standard terminology.

  - **Disadvantages:**
    - Design pattern attempts to regulate the accepted best practices but its structure is still not totally formal as it's still a little vague.
    - In practice, design patterns often result in inefficient design with high resource usage.

- **Types of design pattern :-**

  It broadly categorised into three types:
  - **Creational design patterns:** As the name suggests these design patterns handle the problem of creating objects (instances) from classes in a dynamic manner. It helps give more control over the stepwise creation process. It has the following well known patterns:

        - **Builder:** This design pattern uses a separate class called builder class to build complex objects made up of various simple objects by combining them in different ways and in a stepwise manner.
        - **Singleton:** This design pattern restricts the class to only a single instance. This is a very simple and elementary design pattern which defeats the purpose of classes itself that is why it is rarely used in practice.
        - **Factory Method:** This design pattern uses factory classes to build complex objects, leaving the creation details of the object (details like how to create, how many to create etc.) to the creator factory itself.

      Others include: Abstract factory, Prototype.

    - **Structural design patterns:** These design patterns mainly handle the combination of classes and objects in a flexible manner. Using these design patterns ensures that design changes in a single class does not alter the entire design structure making the design loosely coupled.
        - **Adapter:** This design pattern is extremely useful when multiple software engineers design separate modules. It uses an adapter class to make the classes in these modules compatible much like an electronic adapter.
        - **Bridge:** This design pattern separates the class abstraction from its implementation and provides a bridge between them. It can greatly lower the complexity of the design because of this general implementation.
        - **Decorator:** This design pattern allows changing (decoration) of certain instances by wrapping them around a decorator class giving them extra powers much like an upgrade without changing other instances.

      Others include: Composite, Facade, Flyweight, Private Class Data, Proxy, Filter.

    - **Behavioural design patterns**: These design patterns handle the communication between objects and classes. It has the following well known patterns:
        - **Iterator:** This allows traversal over collections without knowing about the internal structure of the collection by creating a separate iterator class which specifies the successor predecessor relationship among collection items.
        - **Template:** This helps to create a template for usage of class member functions in specific orders much like an actual template.
        - **Command:** This design pattern allows an object to act as a command from one class to another.

      Others include: Chain of responsibility, Interpreter, Mediator, Memento, Null Object, Observer, State, Strategy, Visitor.

- **Design Patterns useful for our note taking application design :-**
  1. Builder Design Pattern: I think that the Builder Design Pattern is quite useful for

creating various complex templates for flowcharts in the note editor. Builder Pattern is used to create complex objects by combining simple objects in different ways and in a stepwise manner using a builder class. It separates the construction process of a complex object from it. Various shapes could be combined in a stepwise manner to create complex flowcharts in the application. This pattern could give me more control for each step or subprocess in the construction process of the flowcharts. Same goes with shapes, a shape builder could allow me to create various kinds of complicated shapes in my notes application.

2. Iterator Design Pattern: I think that the Iterator Design pattern is quite useful for collections like notebook, subject and topic in the note taking application. The iterator method as the name suggests provides a general method of traversal over the collections without worrying about the internal structure of the collection items or the underlying data-structure used in the collection. Thus each of these collections could have separate representation in the form of lists or sets which could be easily changed if required. It would make my design more efficient as I could use the best data structure for notebook, subject and topic.

3. Bridge Design Pattern: I think that the Bridge Design Pattern is useful to segregate the note resource's implementation from their abstraction. A note contains various abstractions like image, shape, textbox etc. Each of which have a common implementation of transformations like rotation, expansion, contraction etc. Bridge pattern allows us to separate this transformer implementation from these abstractions thereby allowing each of them to evolve separately thereby reducing coupling. Hadn't this not been the case each of these transformations had to be implemented for each of these abstractions making the design more complex.

4. Filter/Criteria Design Pattern: This structural design pattern is quite rare but it is quite useful for searching. In this design pattern, various collections could be filtered based on various criteria making it easier to search things in collections. I think that this design pattern could be quite useful for note searching and sorting as it separates the process of filtering from the collection representation itself.

- **Conclusions :-**
  In the end design patterns are just the solutions of common design problems encountered by software engineers over and over again. They are just an extra set of tools in a software engineer's toolkit. It is still up to the software engineer to identify the design problem properly and select an appropriate design pattern.

- **References :-**

  - https://sourcemaking.com/design_patterns
  - https://www.tutorialspoint.com/design_pattern/index.htm
  - https://en.wikipedia.org/wiki/Software_design_pattern
  - https://howtodoinjava.com/gang-of-four-java-design-patterns/