INTRODUCTION OF DESIGN PATTERNS

Design Patterns came into existence when "Christopher Alexander" discovered that using some 'design constructs' repeatedly lead to a particular effect in the design which was actually the desired outcome. In context of Software Engineering, the solution to a problem which occurs often within a given software design is a Design Pattern. They are Reusable and Reusing these Design Patterns is much more efficient than developing new solutions from scratch as they greatly reduces the error probability.

Design Patterns can be thought of as connections between classes and objects with some defined actions which leads to a solution as a whole. They are like pre-defined templates which solve a problem and is applicable to different application scenarios. Since they are relationships between classes and objects that makes design patterns independent of the programming language used and are merely general strategies to solve a common problem. They might not be needed always in a design project but can be used wherever their purpose is filled i.e. wherever a designer encounters a problem which can be solved by using design patterns.

TYPES OF DESIGN PATTERNS

Design Patterns solve a problem and based on the type of problem they solve, these are divided into 3 classification categories:

1. Creational Design Patterns – These patterns are responsible for class instantiation (with effective inheritance) and object creation in a controlled manner e.g. Abstract Factory, Factory Method, Builder, Object Pool, Singleton and Prototype.

    - Singleton - This pattern ensures that there is only one instance that can exist by the instantiation of this class.
    - Factory Method – This pattern deals with the problem of creating objects without the need of specifying object's exact class.

2. Structural Design Patterns – These patterns are responsible for creating new structures by organizing different objects and classes which leads to a new functionality e.g. Bridge, Adapter, Composite, Facade, Decorator, Proxy, Flyweight and Private Class Data.

    - Flyweight – As the name suggests, it is a pattern which helps in minimizing an object's memory usage by sharing its data with similar objects.
    - Adapter – This pattern allows an interface of existing class to be used as another interface.

3. Behavioural Design Patterns – These patterns are concerned with the communication between class's objects e.g. Chain of responsibility, Command, Interpreter, Iterator, Mediator, Memento, Null Object, State, Strategy, Template Method and Visitor.

    - Iterator – In this pattern, an "iterator" is used go through and access a container's elements and decouples algorithms from containers.
    - Null Object – This patters describes the use and behaviour of an object that is defined as neutral (NULL).

There are all together more than 23 design patterns which exists and have their respective use in designing.

ADVANTAGES OF DESIGN PATTERNS

1. Design Patterns are tested solutions and they provide us with a way to solve software development related issues with a proven solution.

2. The modules which are developed using these Design Patterns are highly cohesive and have minimal coupling which are the key characteristics of a good software design.

3. Since they are templates, the communication between Designers about the high-level design of the application becomes much efficient and improves code readability.

4. Reusing Design Patterns leads to lesser issues which prevents major problems in the design at a later stage.

5. They are general solutions to the design problems and provide clarity to the design and makes it more flexible & reusable unlike absolute solutions.

DESIGN PATTERNS USED IN PROJECT GIVEN

(HUMAN DIGESTIVE SYSTEM VR APPLICATION)

1) **(Singleton)** The Digestive System 3D Model is a single initialized model and only one instance of it can exist in the 3D environment therefore it's a singleton design pattern.

2) **(State)** When we use functions like rotate and simulation mode, the state of the 3D model changes and it becomes dynamic hence state design pattern is applicable. Also State objects are often Singleton.

3) **(Composite)** The 3D Model consists of multiple organs and user can interact with each organ separately and hence due to decoupled organization, it is a composite design pattern.

4) **(Memento)** When switching into dynamic and static simulation, there is need to store the state of the 3D model and we can achieve that by memento design pattern as it captures an object's internal state and allows the object to restore it later.

5) **(Lazy Initialization)** When user is logging in to the VR Application, the 3D Model and its properties are not required to load before successful sign-in and therefore lazy initialization is used where the 3D Environment and Objects are only initialized when required.