Assignment 9
Identification and Implementation of GOF Pattern

AIM - Apply any four GOF patterns to refine Design Model for a given problem description, using effective UML 2 diagrams and implement them with a suitable object-oriented language.

PROBLEM STATEMENT -

1) Identification and Implementation of GOF pattern
2) Apply any two GOF patterns to refine the Design Model for a given problem description using effective UML 2 diagrams and implement them with a suitable object-oriented language.

OBJECTIVE -

1) To Study GOF patterns.
2) To identify the applicability of GOF in the system.
3) Implement a system with GOF pattern.

THEORY -

The GoF Design Patterns are broken into three categories:
Creational Patterns for the creation of objects, Structural Patterns to provide a relationship between objects and finally, Behavioral Patterns to help define how objects interact.

1) Creational Design Patterns

a) Abstract Factory - Allows the creation of objects without specifying their concrete type.
b) Builder - Uses to create complex objects.
c) Factory Method - Creates objects without specifying the exact class to create.
d) Prototype - Creates a new object from an existing object.
e) Singleton - Ensures only one instance of an object is created.

2) Structural Design Patterns

a) Adapter - Allows for two incompatible classes to work together by wrapping an interface around one of the existing classes.
b) Bridge - Decouples an abstraction so two classes can vary independently.
c) Composite - Takes a group of objects into a single object.
d) Decorator - Allows for an object's behaviour to be extended dynamically at run time.
e) Facade - Provides a simple interface to a more complex underlying object.

f) Flyweight - Reduces the cost of complex object models.
g) Proxy - Provides a placeholder interface to an underlying object to control access, reduce cost, or reduce complexity.

3) Behavior Design Patterns

a) Chain of Responsibility - Delegates command to a chain of processing objects.
b) Command - Creates objects which encapsulate actions and parameters.
c) Interpreter - Implements a specialized language.
d) Iterator - Accesses the elements of an object sequentially without exposing its underlying representation.
e) Mediator - Allows loose coupling between classes by being the only class that has detailed knowledge of their methods.
f) Memento - Provides the ability to restore an object to its previous state.
g) Observer - Is a publish/subscribe pattern which allows a number of observer objects to see an event.
h) State - Allows an object to alter its behaviour when its internal state changes.
i) Strategy - Allows one of a family of algorithms to be selected on-the-fly at run-time.
j) Template Method - Defines the skeleton of an algorithm as an abstract class, allowing its subclasses to provide concrete behaviour.
k) Visitor - Separates an algorithm from an object structure by moving the hierarchy of methods into one object.

A. Strategy Design Patterns

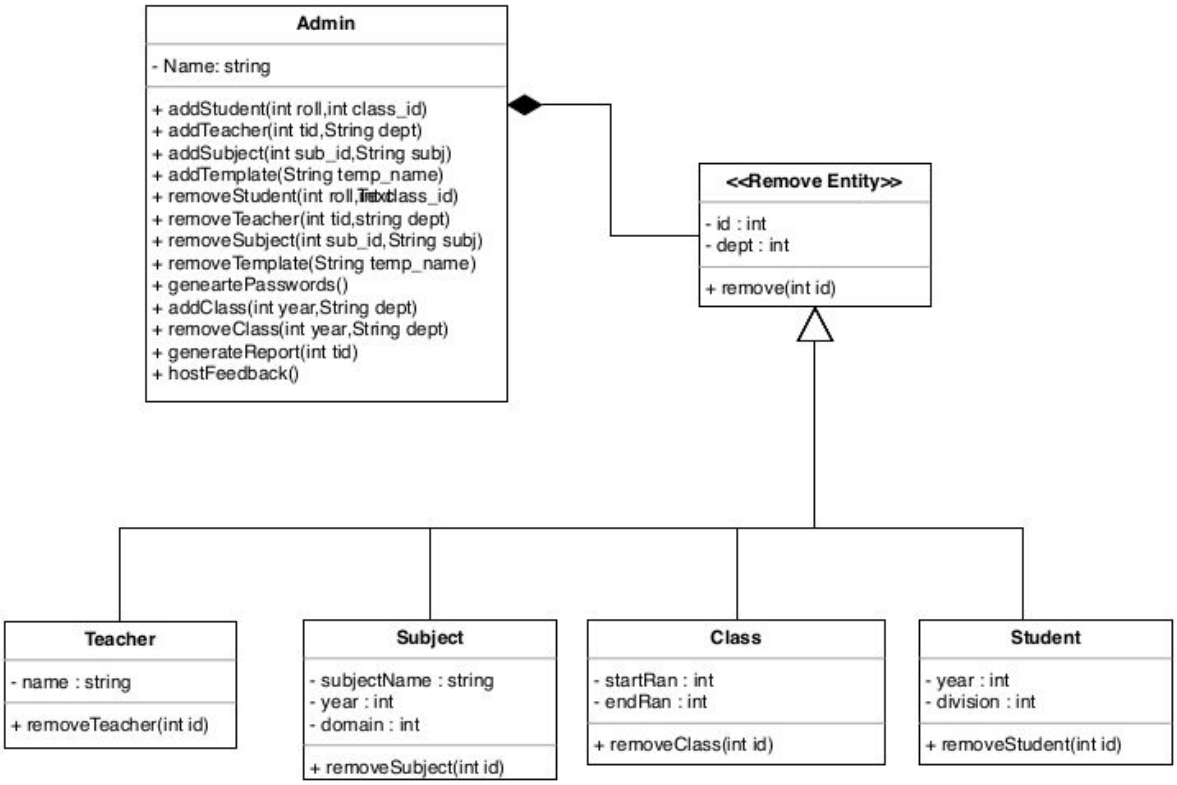
Defines a family of algorithms,

a) Encapsulates each algorithm, and makes the algorithms interchangeable within that family.
b) Strategy lets the algorithm vary independently from clients that use it.

Example :

For instance, a class that performs validation on incoming data may use a strategy pattern to select a validation algorithm based on the type of data, the source of the data, user choice, or other discriminating factors. These factors are not known for each case until run-time, and may require radically different validation to be performed. The validation strategies, encapsulated separately from the validating object, may be used by other validating objects in different areas of the system (or even different systems) without code duplication.

Strategy Design Pattern



In this project, we can use the strategy pattern to formulate multiple strategies for all entities. Admin class needs to delete the entities and so we need to implement several methods to delete the respective entities. We can use a single strategy interface to form a common strategy. This interface will be inherited by the subsequent four subclasses. Each of these classes will override the parent method. Then we can only create the object of a single subclass. Thus by calling appropriate subclasses we can make use of a strategy pattern to make our system more modular with respect to software engineering.

B. State Design Patterns

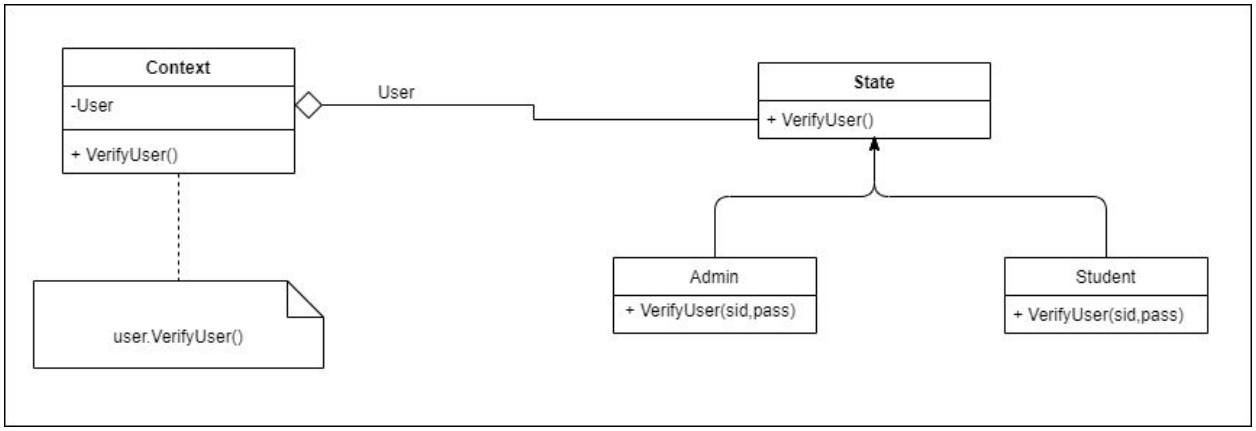
State pattern is one of the behavioral design patterns. State design pattern is used when an Object changes its behavior based on its internal state. The state pattern is a behavioral software design pattern that implements a state machine in an object-oriented way.
With the state pattern, a state machine is implemented by implementing each of this pattern is used in computer programming to encapsulate varying behavior for the same object based on its internal state.

This can be a cleaner way for an object to change its behavior at runtime without resorting to large monolithic/huge conditional statements and thus improve maintainability.

Example:

The State pattern allows an object to change its behavior when its internal state changes. This pattern can be observed in a vending machine.
Vending machines have states based on the inventory, amount of currency deposited, the ability to make change, the item selected, etc. When currency is deposited and a selection is made, a vending machine will either deliver a product and no change.

State Design Pattern



In this project, we can use the state design pattern to formulate multiple state for all entities. Admin class needs to verify the user and so does student to login into the system we need to implement this methods. Each of these classes will override the parent method (Account). Thus we can make use of a state design pattern to make our system more modular with respect to software engineering.

CONCLUSION -

Thus in this assignment we have successfully identified and implemented GOF patterns.