classmate
Date
Page

(2) Structural Design Pattern

- Structural design patterns are design patterns that ease the design by identifying a simple way to realize relationships between entities
- Flexible interconnecting modules which can work together in a larger system.
- Structural patterns describe how classes and objects form larger structures

Structural design pattern

- Adapter
- Bridge
- Composite
- Decorator
- Facade

- Flyweight
- proxy.

1. Adapter

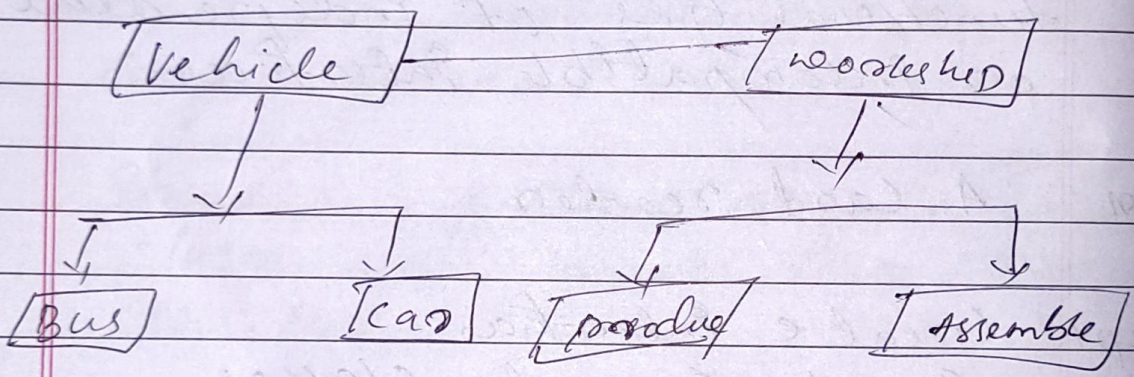
* Adapter pattern works as a bridge between two incompatible interfaces.
* This pattern involves a single class which is responsible to join functionalities of independent or incompatible interfaces.

ex. A card reader

Step 1 Create Interface
2 Create Concrete classes
3 Create adapter class
4 Create Concrete Class
5 Use the audio player to play different types of audio formats.

2. Bridge

- Bridge is used when we need to decouple an abstraction from its implementation so that the two can vary independently.
- This pattern decouples implementation class and abstract class by providing a bridge structure between them.

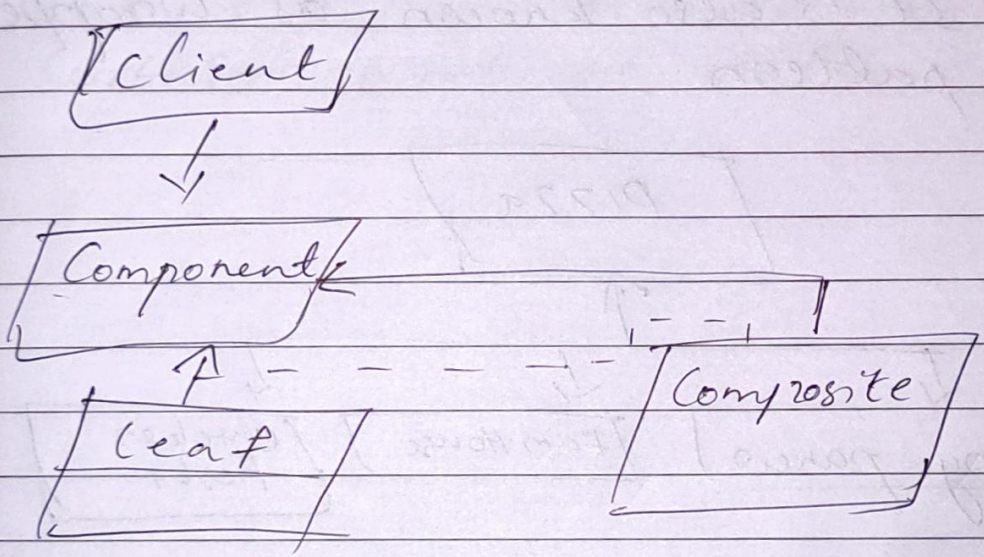


3 Composite

- Compose objects into tree structure to represent part-whole hierarchy
- Allows you to treat individual object and compositions object

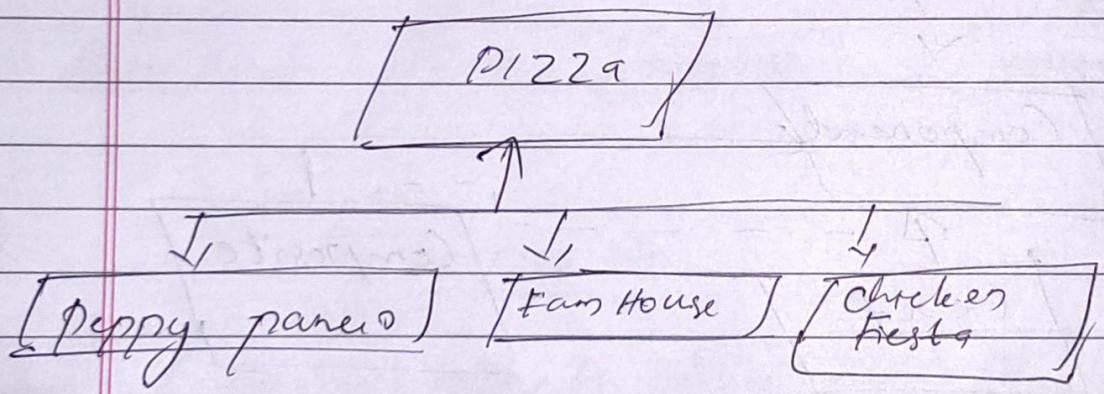
uniformly

- This is applicable when we want to create an object and represent in tree structure.
- Break it down into simple elements and containers.
- Create a leaf class to represent simple elements. A program may have multiple different leaf classes.

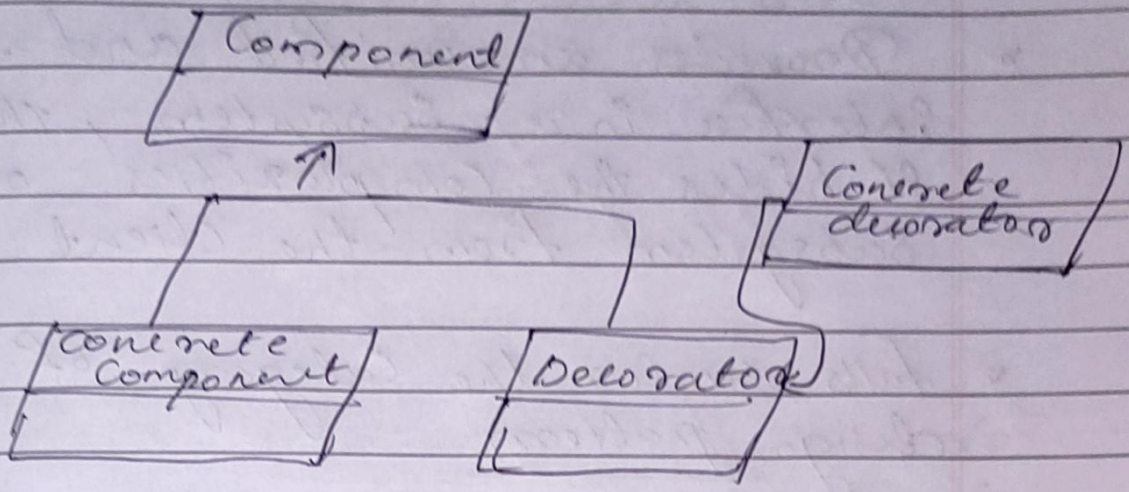


4 # Decorator

* Decorator pattern allows a user to add new functionality to an existing object without altering its structure during dynamically.
* This pattern creates a decorator class which wraps the original class and provides additional functionality.
* More flexible than inheritance
* It is also known as wrapper pattern.



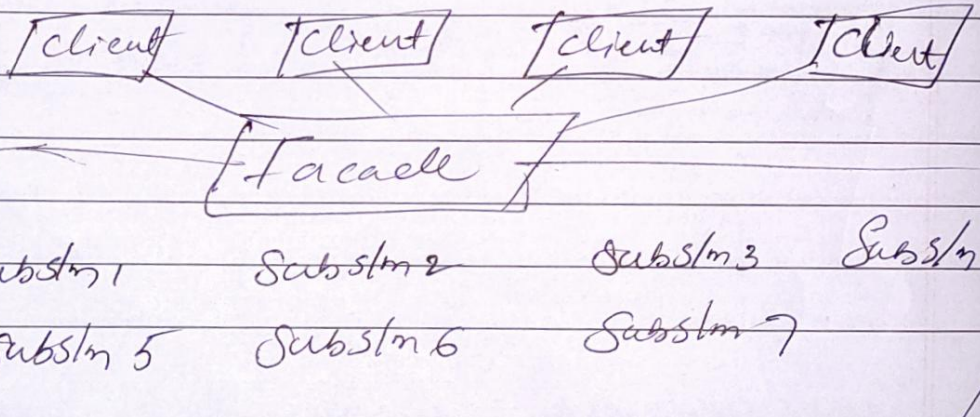
Structure



- The decorator pattern can be used to make it possible to extend the functionality of a certain object at runtime.

5 Facade

* Provides an unified and simplified interface in a subsystem, therefore it hides the complexities of the subsystem from the client.
* Falls under the category of structural design pattern
* Facade means face or mask
* We want to layer the subsystems. Use a facade to define an entry point to each subsystem level.
* It provide single simplifide interface with more general facilities of a sub system



* Facade knows which Sbm or which SubSbm classes are responsible for a request.

6 Flyweight

* Mostly used when we want to create large number of similar patterns
* Flyweight pattern is primarily used to reduce the number of objects created and to decrease memory requirements
* Intrinsic and Extrinsic states
Each flyweight object is divided into two pieces.
* The state-dependent (extrinsic) part and the state independent (intrinsic) part.
* Intrinsic state is shared in the

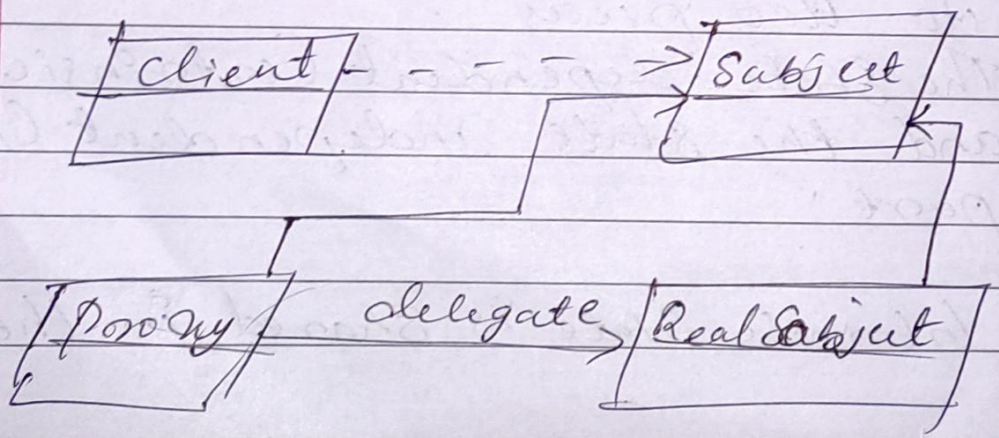
Flyweight object

- Extrinsic state is stored or computed by client objects, and passed to the flyweight when its operations are invoked.

7. Proxy

- In proxy pattern, a class represents functionality of another class.
- In proxy pattern, we create object having original object to interface its functionality to outer world

Structure



(3) Behavioral Design Pattern

* Behavioral design patterns are design patterns that identify common communication patterns between objects and realize these patterns.

* By doing so, these patterns increase flexibility in carrying out this communication.

* Behavioral Patterns are those which are concerned with interaction between the objects.

features

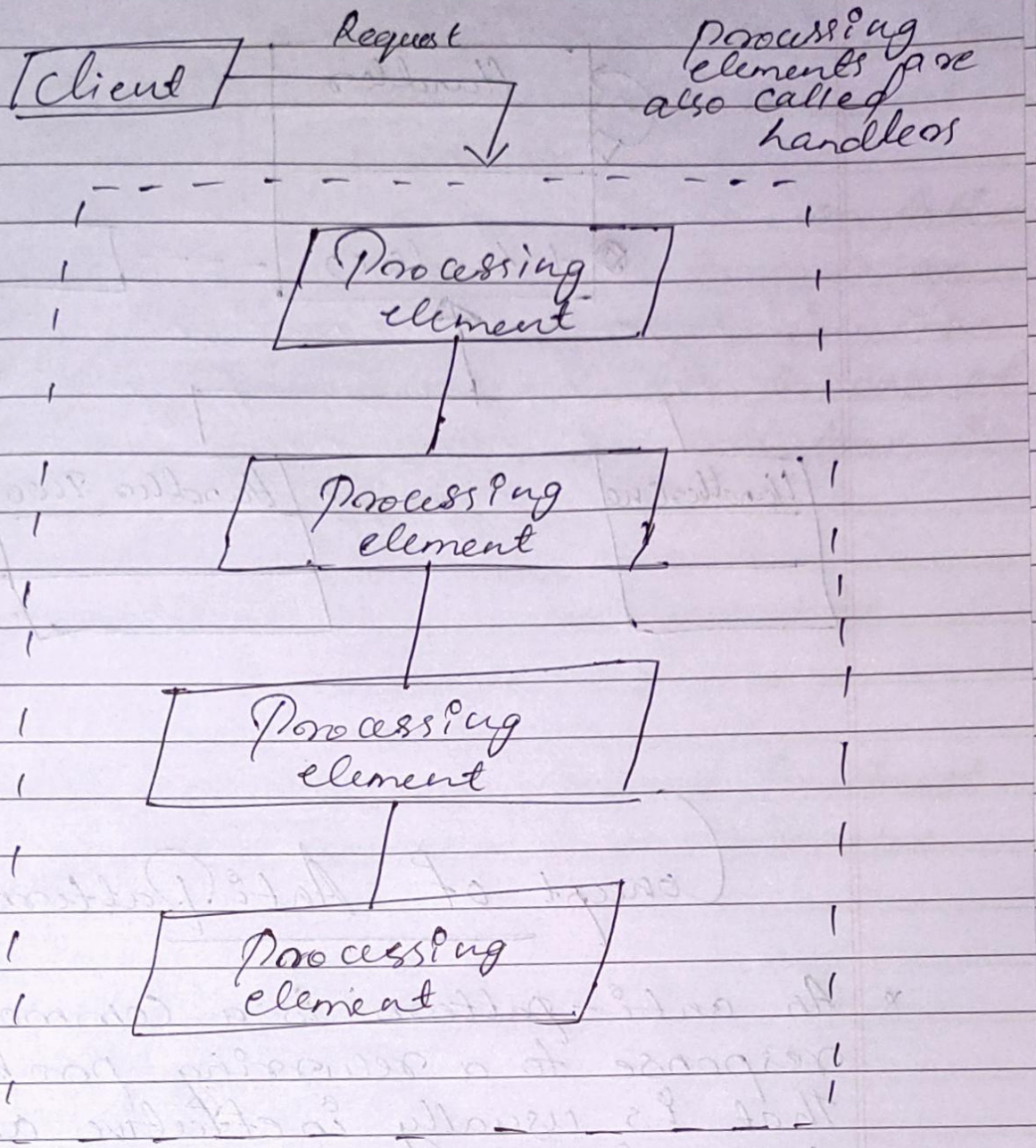
* Chain of responsibility
* Command
* Interpreter
* Iterator
* Mediator

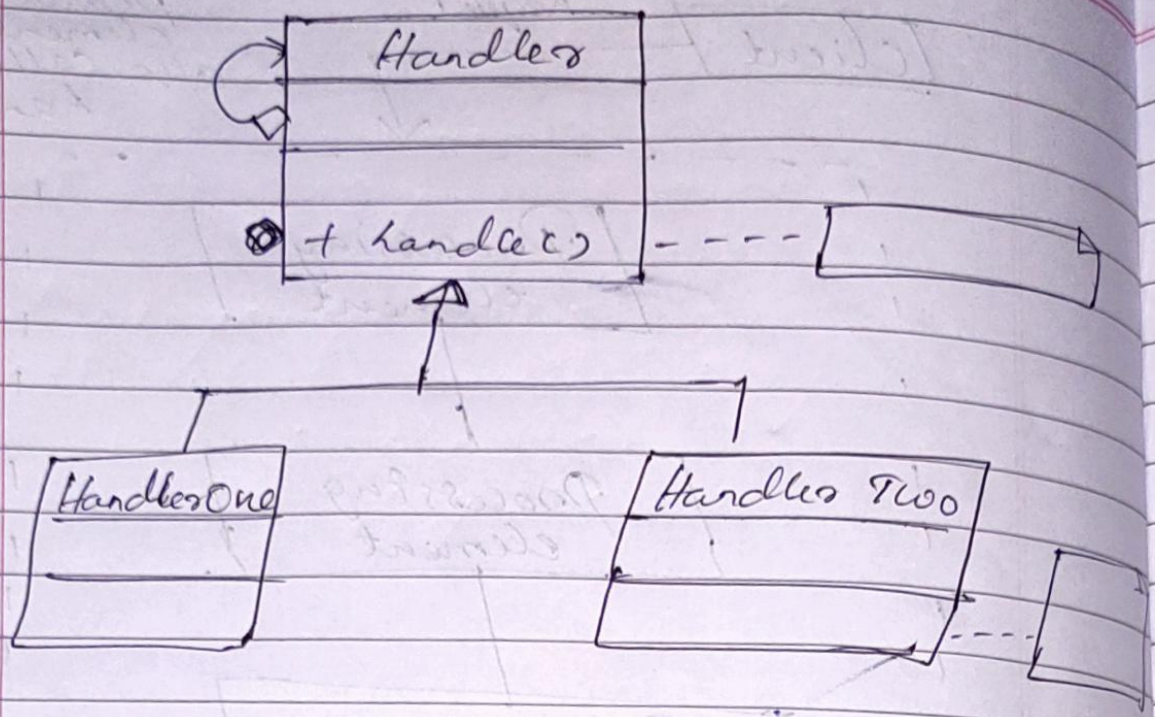
- Memento
- Null object
- Observer
- State
- Strategy
- Template method
- Visitor

Chain of responsibility

- Avoid coupling the sender of a request to its receiver by giving more than one object a chance to handle the request.
- Chain the receiving objects and pass the request along the chain until an object handles it.

- An object-oriented linked list with recursive traversal.





Concept of Anti Patterns

* An anti-pattern is a common response to a recurring problem that is usually ineffective and risks being highly counterproductive
* Antipatterns highlight the most common problems that face the S/W industry and provide the tools to enable you

to recognize these problems and to determine their underlying causes.

* When a pattern becomes an Anti-pattern, it is useful to have an approach for evolving the solution into a better one. This process of change, migration or evolution is called refactoring.

* In refactoring, we change one solution to another.

* Antipatterns are present in almost any organisation and software project.

* They can be categorised into three groups :-

↳ Development AntiPatterns :- Mainly concern the software developer "Situations encountered by the programmer when solving programming problems.

↳ Architectural Antipatterns: are important for the Software architect. Common Problems in S/w structure their consequences and solutions.

↳ Management Antipatterns: are relevant for the S/w project manager. Common problems and solutions due to the S/w organisation.