Software Architecture:-

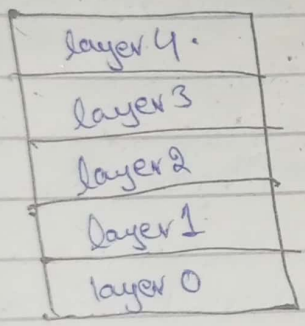
Software Structure.

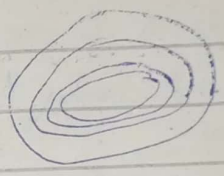
- Structure of a program comprised by its major constituent, their responsibilies & properties & Relationships & interaction b/w them

Architectural Design.

i-e In a shopping mall there are sensors who sensee any fire happening & after that they sprinkle water.

Layered style Architecture: (Major)





- partitioned into layers or groups.
- Use services of the layer below. & provide services to the above layers.

Uses & Invokes:-

Invokes:-

Module A invokes or call Module B (A triggers execution of B).

Uses:-

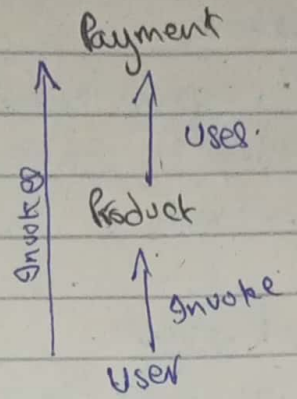
A uses B if B has correct version that must be present for A to execute correctly.

Note that:-

Uses	Invoke.
✓	✗
✗	✓
✓	✓
✗	✗.

↳ Exist in System but has no relation with anyone.

Example.



Layer Constraints:-

Static structure:-

layer are independent Not use any layer but invoke.

Dynamic:-

Upper layer use only 1 below layer than in its strict.

Upper layer uses more than 1 layer than it is related.

Forming layers:-

1 level of abstraction:-
i.e Network communication layers.

2 Virtual Machine:-
i.e OS, interpreters.

3 Information hiding, decoupling etc.
i.e User Interface layers, virtual device layers.

Layer Style Advantage.

Coupling. Modules depend on each other, slow processes.

Cohesion:-

- Disadvantages:- order of
- Debugging , layer dependency on each other

Shared Data Style:-

- Data store at one place & Modules are connected to ~~each~~ that store.
- change in store may cause change in every module.
- Server has high load.

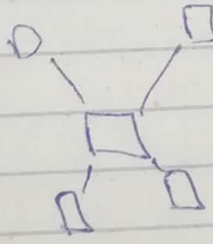
Definition:-

Two variants:-

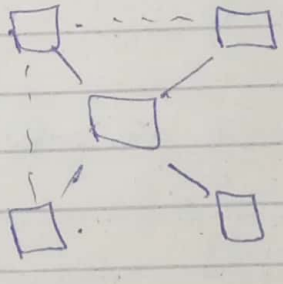
- change in shared data ~~activ~~ activates accessors when the stores change Blackboard style:-
- functionality to client is Repository.

Server not control component but component use server:-

Thin client fat Server.



fat client:- thin server:-



Advantages:-

Disadvantages

Event-Driven Style:-

- System not having more functionality
- Sensors Detect Eg Event occur.
- Input / Out put

Implicit Invocation:
Not having a proper system that is run by any user.

Stylistic Variation:-
like traffic warden uses cameras who take pictures of every car.
- Synchronously
- Asynchronously.

- Advantages
Easy to add Eg Remove components:-

Disadvantages:-
More Events less functionality

Model-View-Controller ~~Syst~~ style

Pipe & filter Style.
- Amazon, firewall

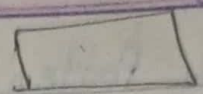
Pipe: Execution

filter: change or pass from one to another

lexical Analyzer:
int x, y Means-of anything.-
Syntax Error caught by it.

Acyclic Graph.

Semantic Analyzer:-



Car Eg vehicle

filter

pipe

Semantic meaning ⇩ same.

Execute concurrently.

- linear Execution:-

Disadvantages:-

Error handling is Difficult.

Model View & Controller sytle.

Data ↓ form in C# that we can see ↓ we click button Eg we went to the function. that function is the controller of the view:-

Front End

BL Business logic.

Back end.

DL

Data logic.

attached to the Data.

Data is independent, Used in View

Disadvantage.

MVC-Behavior:-

all the work that we done in SDA lab C#:-

View & Controller are often hard to separate.

- Slow processing when connected to Database

Hybrid Architectures:-

Software Design Pattern.

A pattern is an outline of a reuseable Solution to a general problem encountered in a particular context

A pattern should have:-

context, problem, forces, Solution, Anti-Solution, related, patterns, references:-

i.e.

like static variable has 1 space in RAM & that is share or access across multiple classes to reduce space problem.

context:-

Paragraph, Description of problem

Problem:-

Question type · short form of context with ?

Forces:-

What kind of constraints we have, which I have to take care of during the solution.

Solution:-

Solution is a class Diagram + Description.

Anti-Solution:- anti-pattern

common Mistakes made by programmers:-

Solve the problem & can't take care of the forces:- (also class diagram)

related pattern:- optional. related Solution:-.

Solution of a problem, with an alternate solution, solve the problem, can take care of the forces.

• Best one is Solution, other are related Pattern.

References:-

A pattern, who is the author of the pattern or in which year do they publish.

Time:-

change in context also change the pattern.

* **Abstration- occurance ~~procontext~~- pattern- context-**

- In a library, books have same title, Actor, publisher etc. but bar_code is different
- A flight has same destination, leaving station but has different date & passengers:-

Problem:-

how can we represent such set of occurances in class diagram.

Forces:-

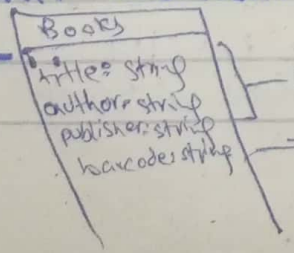
Duplication & Flexibility.

- less efficiency of program
- take care of problem in such a way that common data should not be in multiple classes.

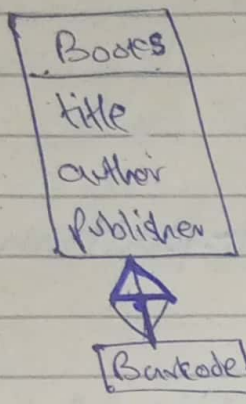
- easily add & remove occurance.
- previous one will not be disturbed

Anti-Solution:-

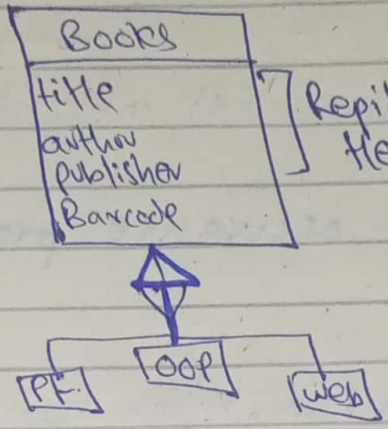
1-



2:- Common information is in parent class & different is in inherited class:-

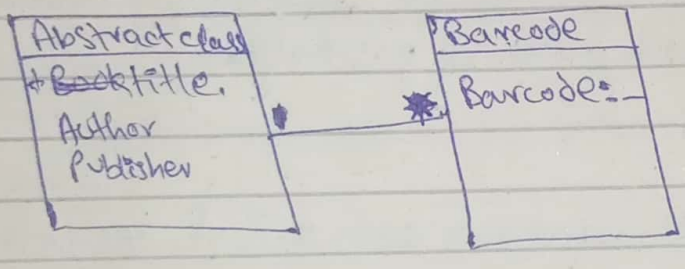


3:-



Repitition + Flexibility problem.

Solution:-



★ General Hierarchy Pattern:-

Context:-

VC
↓
Dean
↓
Chairman
↓
Professor
↓
Associate
↓
Assistant.

Roles occupy by VC.

→ Hierarchy end at this

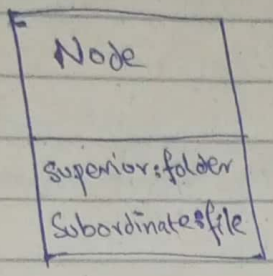
Problem:-

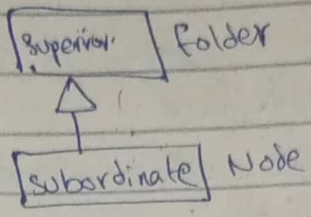
How can we show hierarchy in our class Diagrams in which some objects don't have subordinates?

Forces:-

- Flexibility → make multiple folders into the parent folder can't act any effect on parent.
- Common features:-

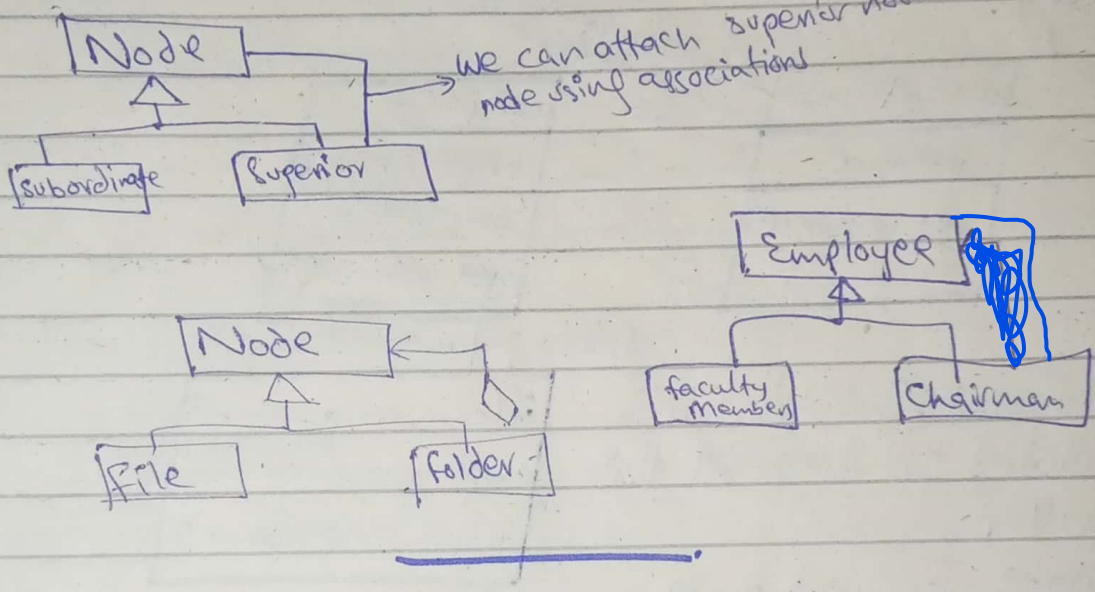
Anti-Solution:-





one folder has multiple folders
No Flexibility

Solution



3) Singleton Pattern:-

Context:-

University Ep company name is same for all student or Employees (university/company name is saved in class)

problem:

How do you make sure it is never possible to make more than one instance of a class:

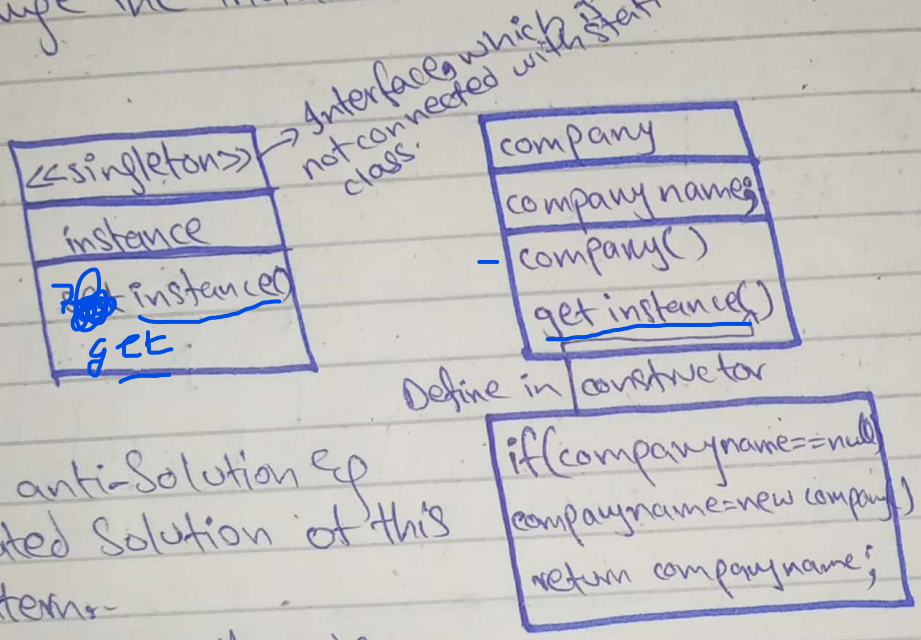
Forces:-

Singleton instance must be public Ep should not

be changed (Asseccible to all classes)

Solution:-

→ private class: instance.
→ public class static which creates & store the instance & return it.
→ Private constructor: Ensures that no other class change the instance.



No anti-Solution & Related Solution of this pattern:-
static keyword make a class at the backend also that stores data in variable.

4- Player Role Pattern:-

Context:-

A student can change his role from full time to Part time or from undergraduate to Post-graduate:-

Problem:-

How do you best present players & roles

where player can change roles?

Forces:-

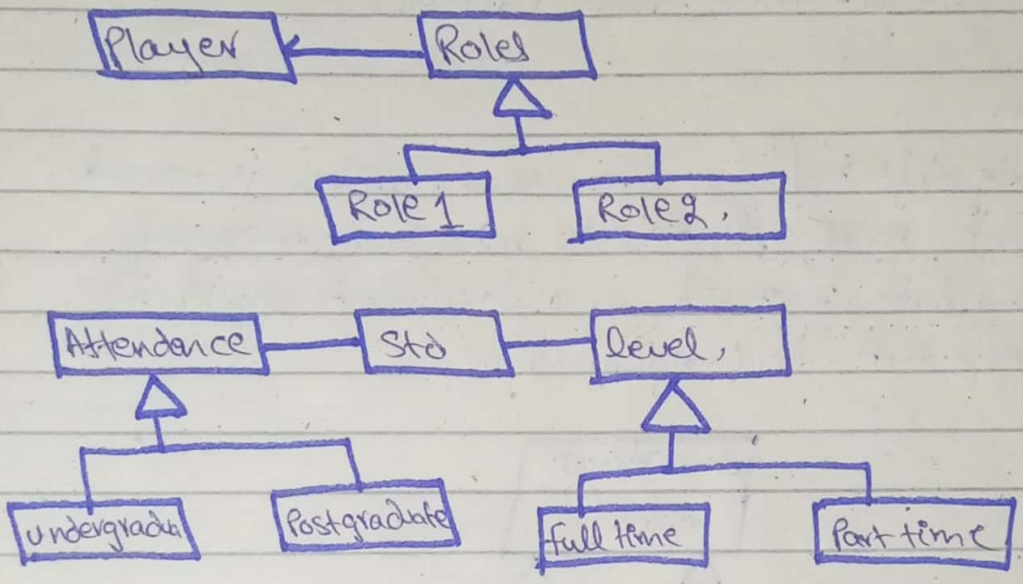
Encapsulation & multiple inheritance

Solution:-

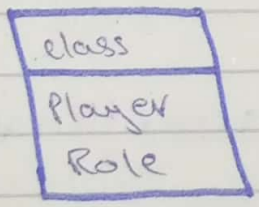
→ Make a role class:-

→ All roles will be subclasses of this role class:-

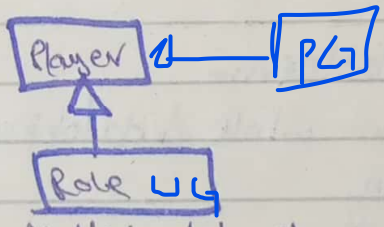
→ Attach roles class to player class with simple association:-



Anti-Solution:-

1. 

Encapsulation is neglected

2. 

Multiple inheritence occurred:-

5:- Proxy Pattern:-

Context:- you have huge Database & working with heavy weight classes which access this data but you cannot bring all data in main memory. And also you don't need whole data to work on

Problem:-

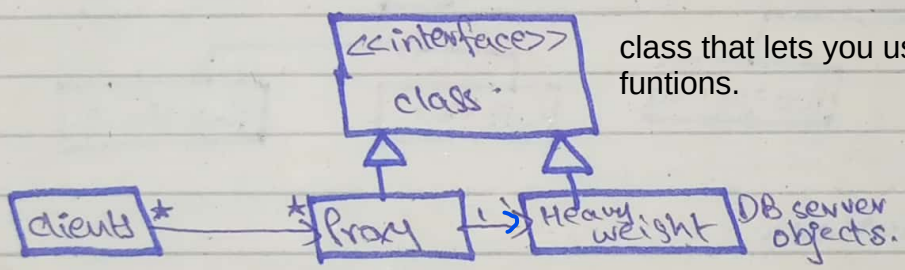
How do you reduce the need to load heavy weight object/data from DB/server

Forces:-

load only objects/data from server which is needed:-

→ Don't bring all objects into main memory:-

→ How data is stored and loaded it should be transparent to programmers:-



class that lets you use its funtions.

Anti-Pattern:-

load whole data/objects from DB/server to main memory:-

6:Delegation Pattern:-

Context:-

You are working in a class & realize text another class already have implementation of

the function, you are creating in first class.

Problem:-

How do you efficiently use functions that already exists in other class:-

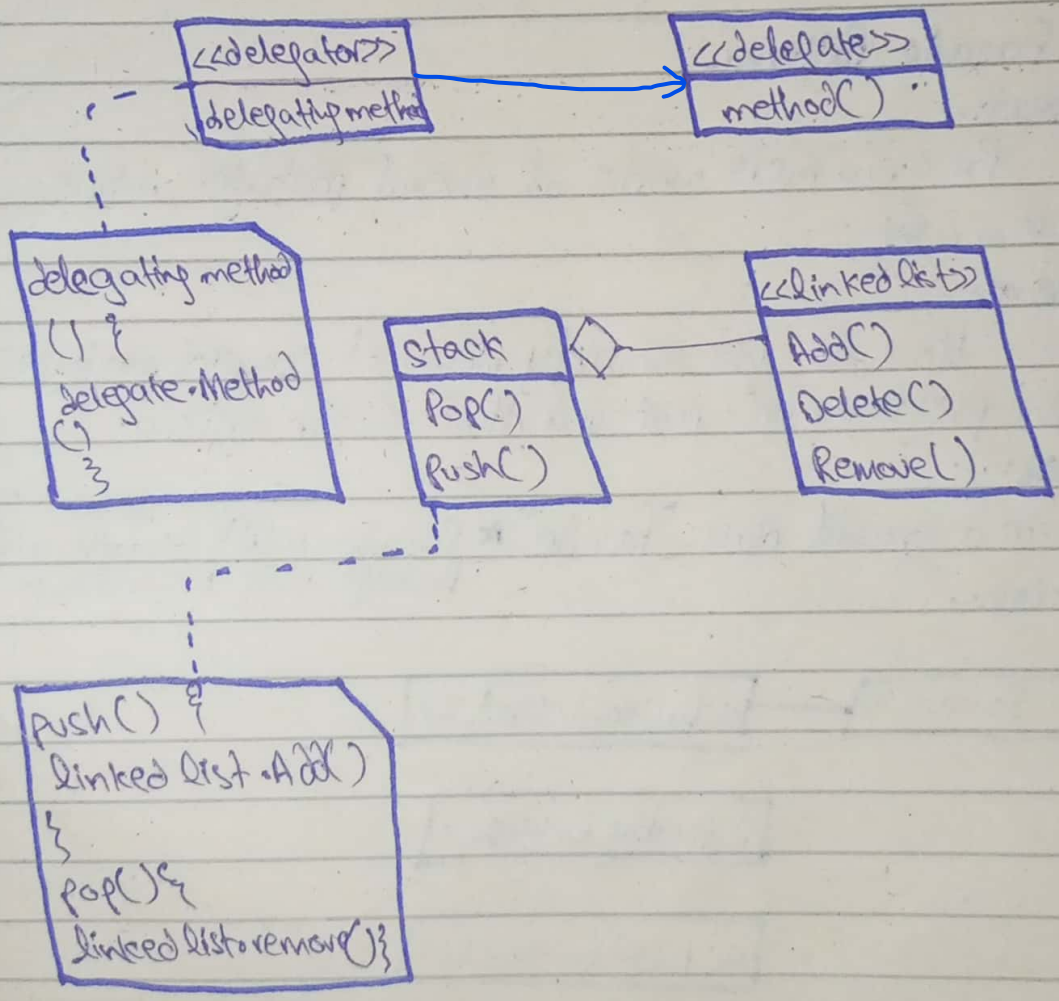
Forces:-

Minimize developments cost, complexity:-

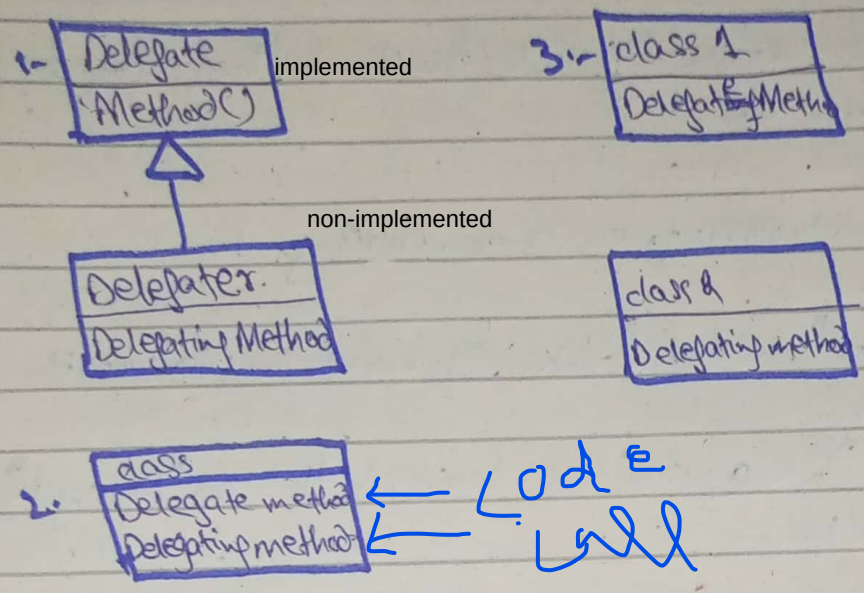
Solution:-

A class called "Delegate" already have implementation of a function:-

→ create class "Delegator" & connect this to "Delegate" using Association:-



Anti-Solution:-



7:- Facade Pattern:-

Context..

Programmers works of several packages while programming.

Problem:-

How do you simplify view of complex packages for programmers: must understood entire system.

Forces:. SOLID

Solution:..

* Create a special class "facade" * Facade will connected with all Package using Association.