+91 2697265
??

★ Creational Patterns:
1. Factory Pattern
2. Singleton Pattern. (only one object)

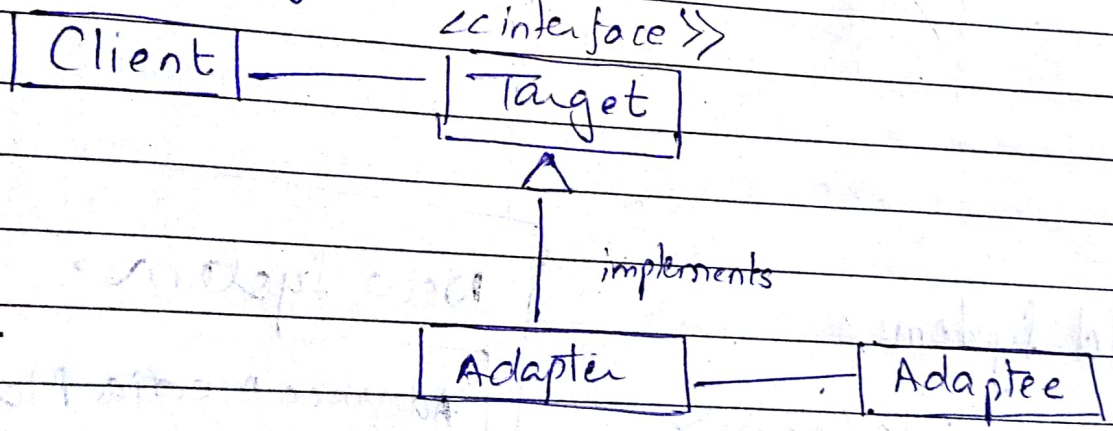
(22/02)

★ Structural Pattern

1. Adapter Pattern:

i) → Definition: Adapter pattern converts interface of one class into the interface which client expects.

ii) → Class Diagram:



Target: It is the interface which defines the actual interface used by the client

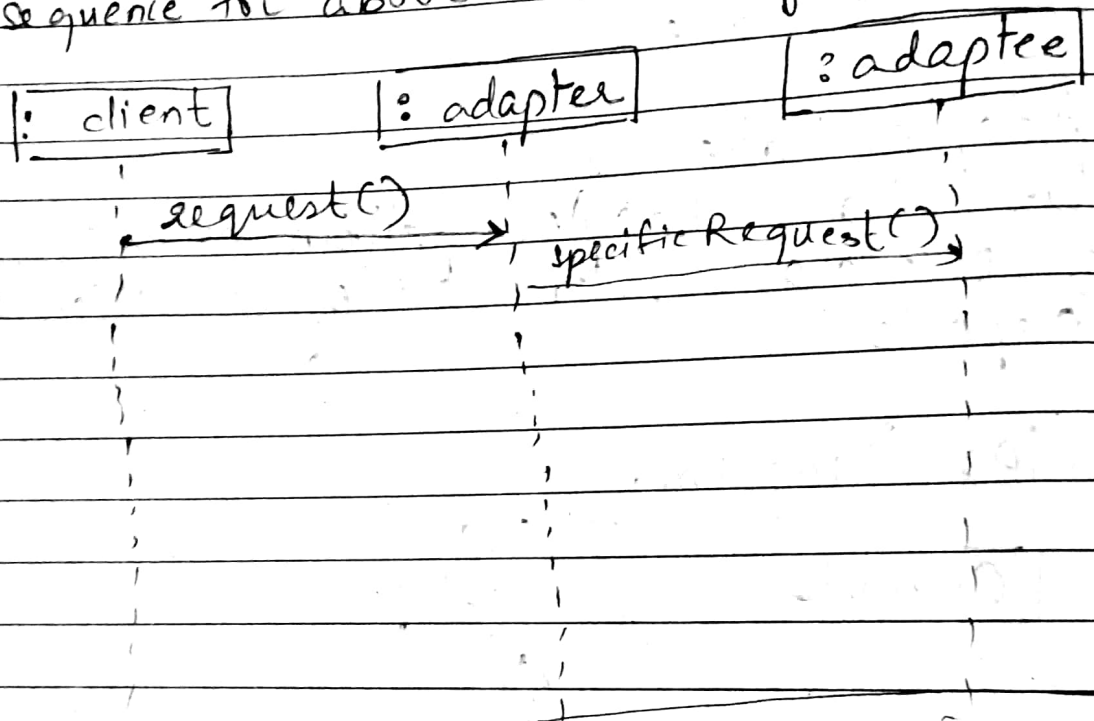
Adapter: Adapter is the class which handles the request from the client to the adaptee.

adapter implements target

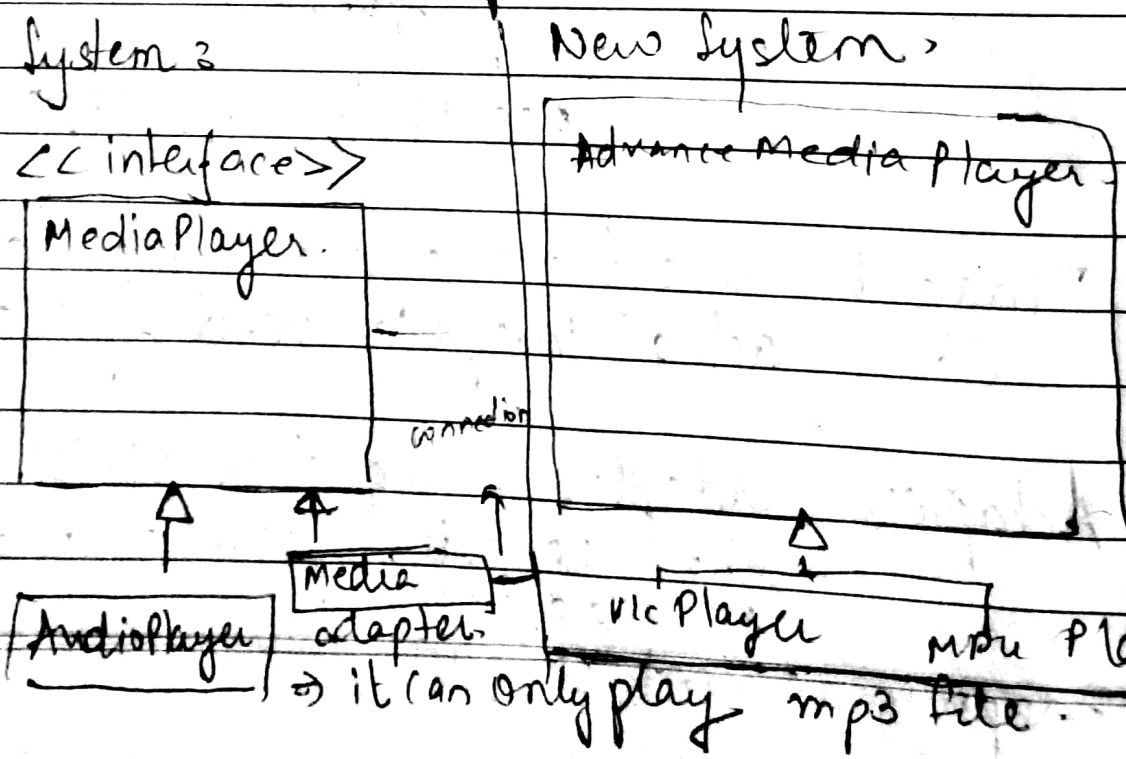
Adaptee: It is the existing interface which needs to adapt so that clients can communicate with it.

★ Sequence for above class diagram.

eg:



Old system:



⇒ it can only play mp3 file.

Client will call the mediaplayer class.
→ Hete Hence to play .vlc & .mp4 files mediaadapter is used.

Methods:
mediaadapter ⇒ +play()
audioplayer ⇒ +play()
mediaplayer ⇒ +play()

adv. mediaplayer ⇒ playvlc(), playmp4().
vlcplayer ⇒ playvlc, playmp4.
~~mediaadapter~~ ⇒
mp4player ⇒ playvlc, playmp4.

Sequence:
→ Calling constructor of audioplayer with filename & filetype
→ If it is mp3 simply call play() method.
→ If not, it will create mediaadapter object which again checks file type and then appropriately executes playvlc or playmp4 method.

→ Types of adapter patterns:

1. Object adapter
2. class Adapter.

(1.) client — Adapter — Adaptee

(2.) client — Adapter — adaptee 1
— adaptee 2.

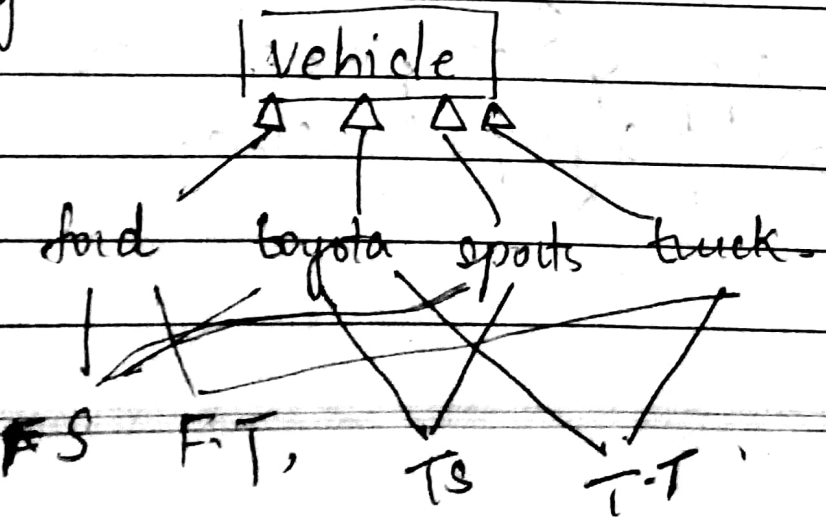
iii) when to use adapter pattern?

| NewApp | Adapter | Legacy |

→ adapter acts as a wrapper between new application & legacy systems

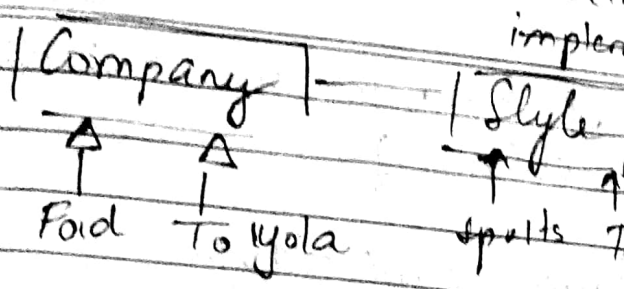
→ For component based development.

2. Bridge Pattern.



implementation abstraction

abstraction implemen

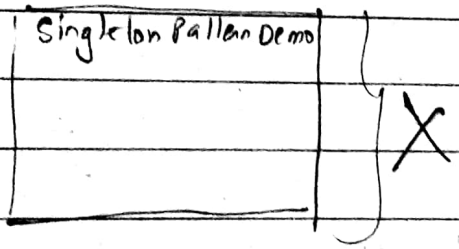


⇒ Bridge pattern decouples class abstraction from its implementation so that each can vary independently.

23/02/18

★ Singleton Pattern:

→ creating a single object programatically ?



★ Prototype Pattern:

→ when creation of object directly is costly a clone object can be created when the behaviour of the objects is exactly similar to each other.

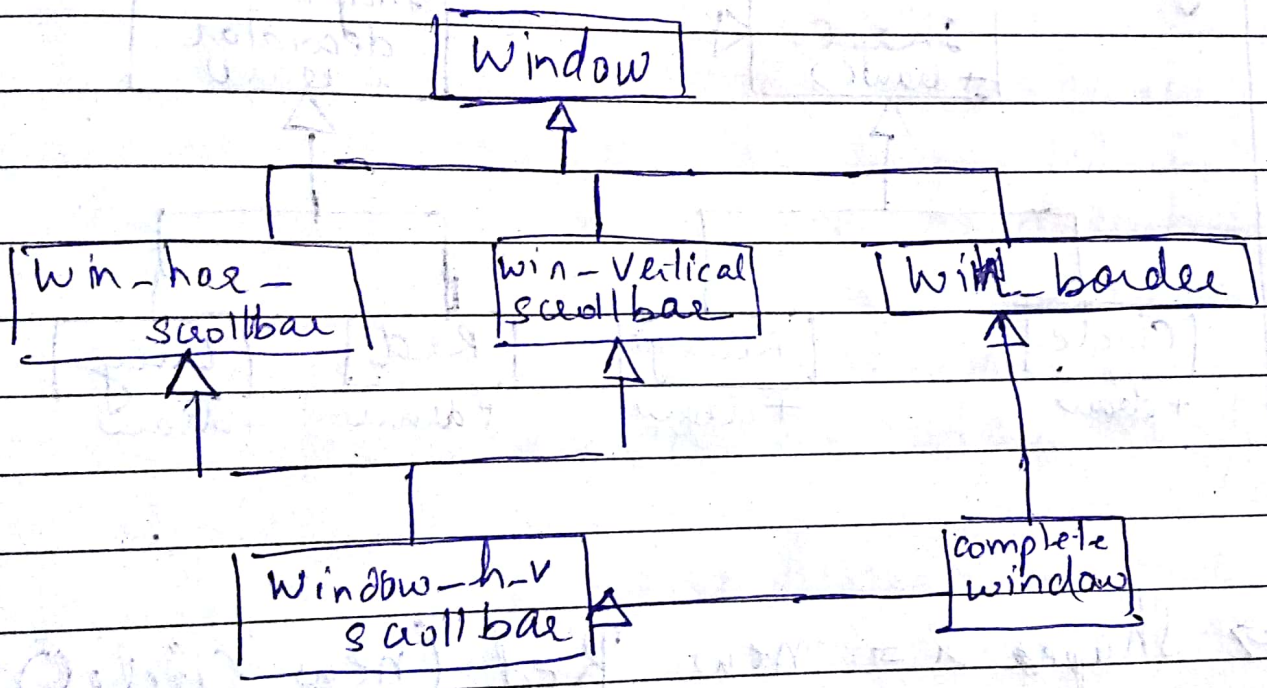
* Builder Pattern - (The last one..! :-:)

(2 lecs. 01/03/18 Missed).

* Decorator Pattern:

→ decorator pattern is used for attaching additional responsibility to an object dynamically.

we have a UI toolkit:



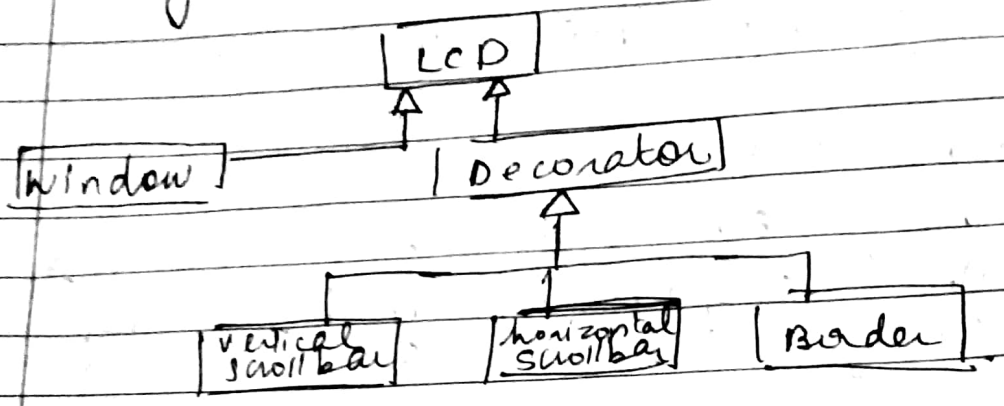
Issues: multiple inheritance.
it is a static design.

192.168.29.1
or
10)

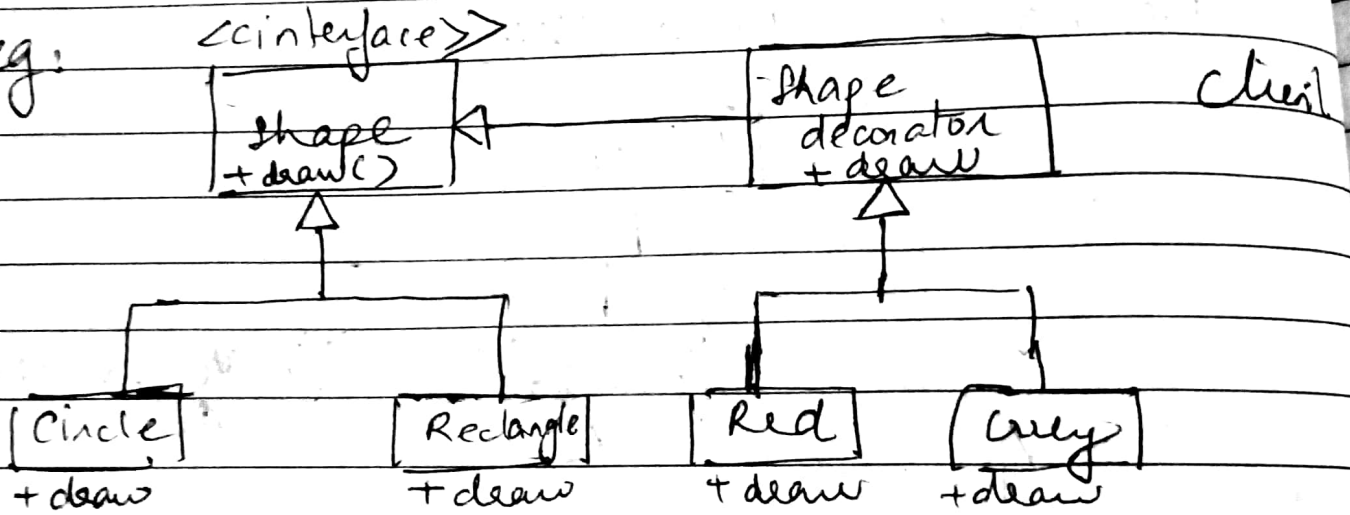
Page No.
Date / /

minimum
minimum

Using a decorator pattern:



Ahrd eg:



⇒ Shape s = new Red (new Circle());

⇒ ~~within~~ Code:

adapter is also a wrapper
decorator is also a wrapper

→ converts the interface.

→

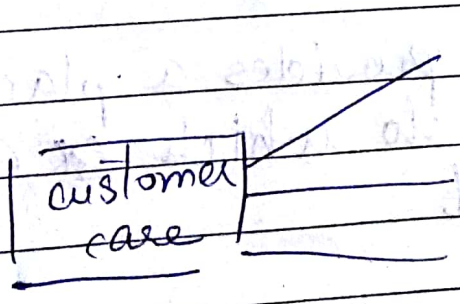
⇒ Adapter pattern provides different interface to its subject, while decorator pattern provides enhanced interface.

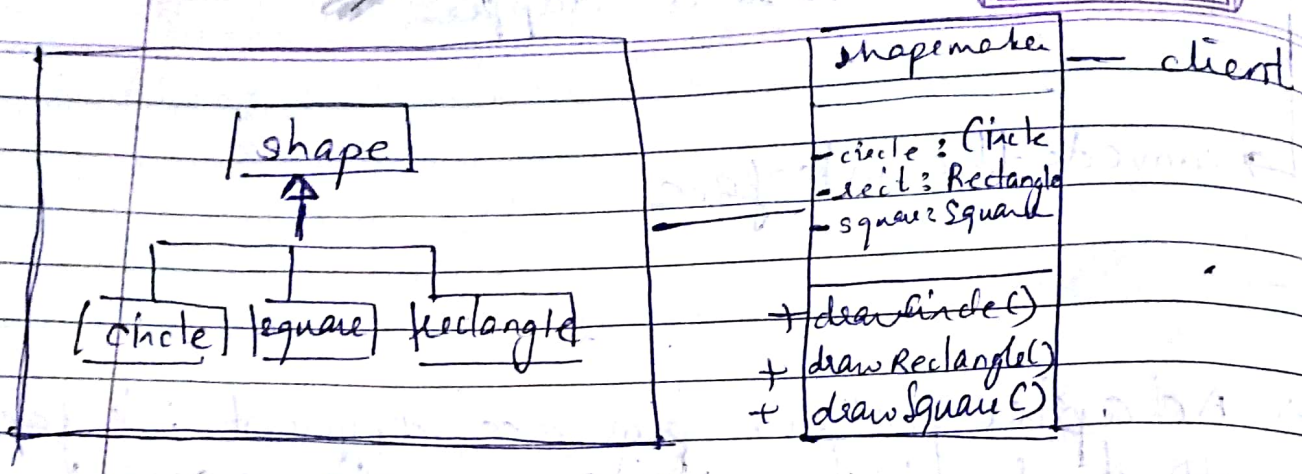
★ facade pattern : (masking, disguising).
→ Facade pattern provides high level interface to wrap complicated subsystems.

⇒ advantages of facade pattern:
It helps to decouple the subsystem from multiple clients.

dis: → facade can limit the functions provided to the clients.

eg:





shapemakers' constructor initialises 3 objects

→ facade defines a simpler interface for clients, while adapter uses existing interface to define new interface.

⇒ When to use:
- → DOA
- → when we want to hide complexity

★ Proxy Pattern:

→ Proxy pattern provides a placeholder for the actual object to which access should be controlled.

hmmm

Atm holds a proxy for an account ob

eg: login forms.
bank cheque is proxy for the money in account.

Virtual-Proxy is a placeholder for expensive objects. (images).
(used in web browsers).

Remote Proxy ⇒ eg: ATM.
It defines local representative of an object which resides at different space.

State Pattern

manager. setSucessor(VP)
VP. SetSucessor(Director).

★ OOSE Design Principles.

20/3/18

★ Proxy Patterns:

3. Protective Proxy:
eg: Proxy Server.

→ Protective proxy should be used when we need to control the acess to a sensitive master object.

4. Smart Proxy:
It is used in:
1. Count the no. of references to the real objects.
2. loading of persistent object when it is referenced.
3. Checking the lock on the real object.

⇒ When to use proxy patterns ?

1. When object is external to the system. (ex: atm).
2. Objects are created on demand.
3. Access control.
4. Add functionality at run time.

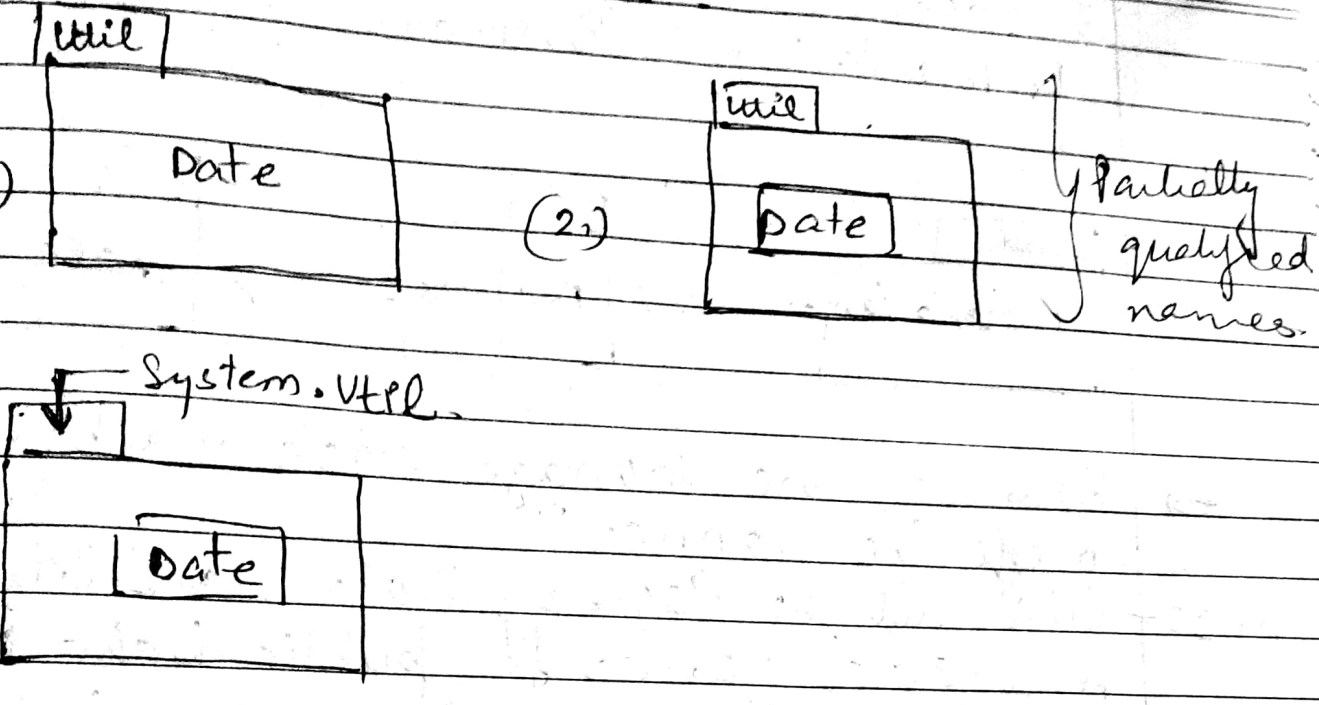
1. Adapter.
2. Bridge.
3. Composite: (employee can contain an array of employees)
4. Decolator
5. Facade.
6. Proxy

★ Package Diagram:

→ It is a grouping construct used for grouping of any UML element into higher level unit.

UML element: usecase, class, activity, state also collab & sequence can be grouped.

eg: System.Util.Date;

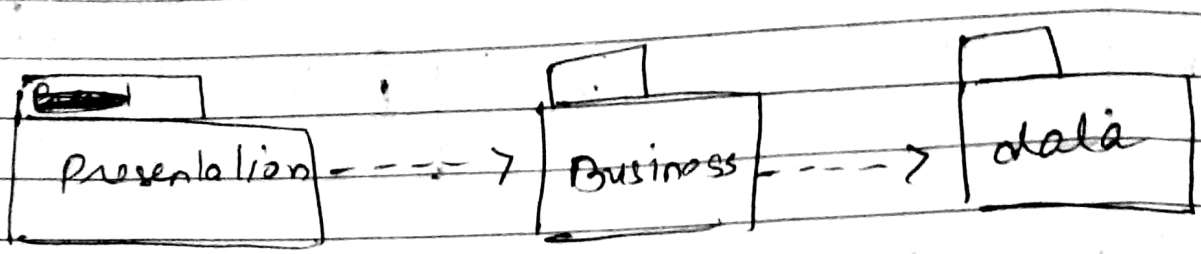


→ Types of notation:

1. Content list:
2. Content Box.
3. Fully qualified package.
4. Nested package.
5. Fully qualified

⇒ Package dependency: Package A depends on package B, if A contains a class which depends on one class in B.

3 classes:
1. Presentation 2. Business 3. Data.



⇒ How can we manage class visibility in effective manner?

Ans:- Give all the classes private visibility. Add extra classes for public behaviour & delegate the request to the private classes.

⇒ Which classes should belong to which package?

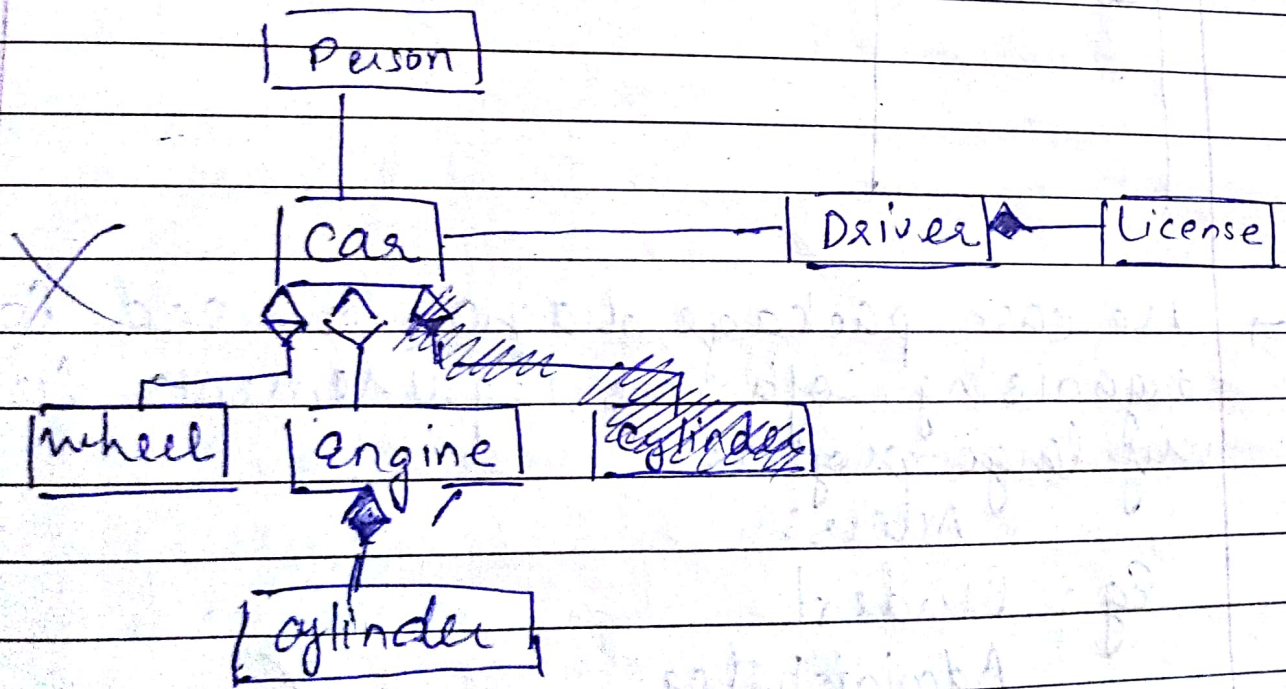
→ Common Closure Principle:
Classes within a package should require changing for similar reasons.

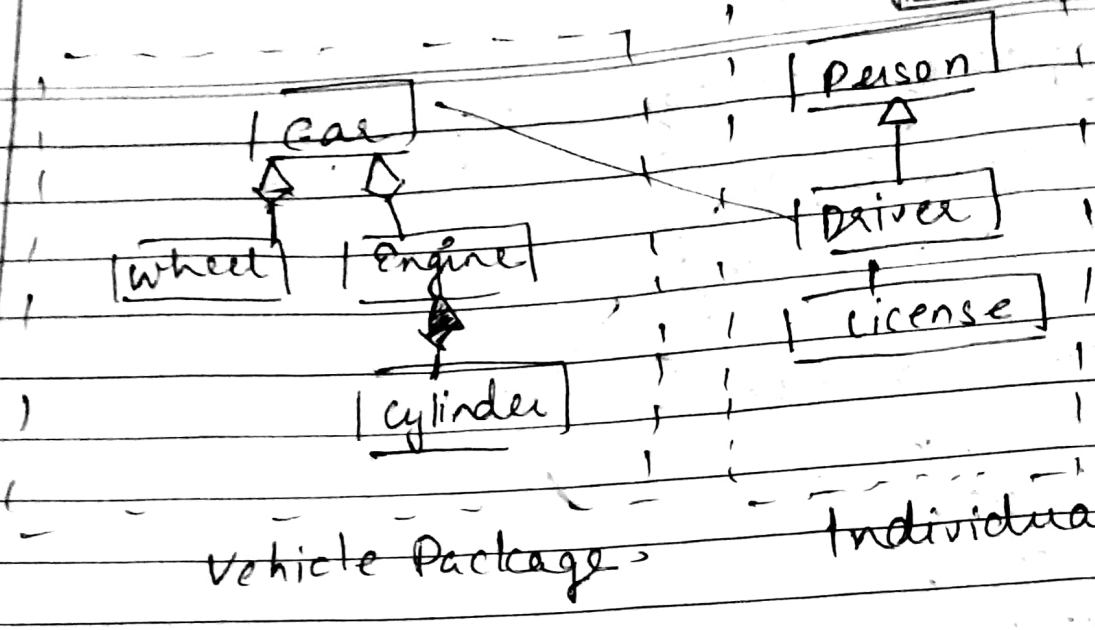
→ Common Reuse Principle:
Classes within the package should be reuse together.

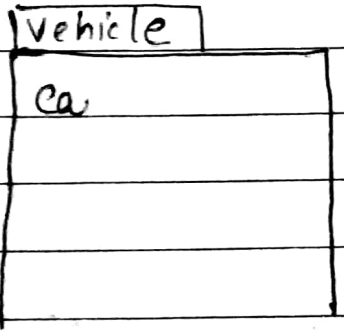
★ Types of package diagram:

1. Class Package.
2. Use case - Package.

Ex: Class names:
Car ✓
Cylinder ✓
Driver ✓
License ✓
Engine ✓
Person
Wheel ✓







⇒ Use case package diagram is used for organising software requirements for very large projects.

Actors:

Eg: Student
Administrator.

Use Cases: Enroll for registration 1
Pay fees 2

Register for seminar. 3

Manage accounts. } Admin. 4
Manage seminars. } 5

Packages: Seminar Mgmt: 3,5.
Account Mgmt: 2,4.
Registration Mgmt:

23/02 object oriented Software Testing: