Date

Liskov Substitution:

Subclasses should be substitutable for of super classes.
If we substitute super class object with the sub class object, it should work.

Interface segregation:

Instead of having a hige interface, we should have single interface for one person.

* Separate interface.

Dependency Inversion:

Classes should depend on abstraction rather than concretion.
It will make the code generic.
To make classes flexible and changeable.

Design Pattern:

"A design pattern is an invariant solution to a recurring problem within a certain context."

Creational:
Deals with creation/instantiation of objects and classes

Structural: Deals with composition of objects and classes.

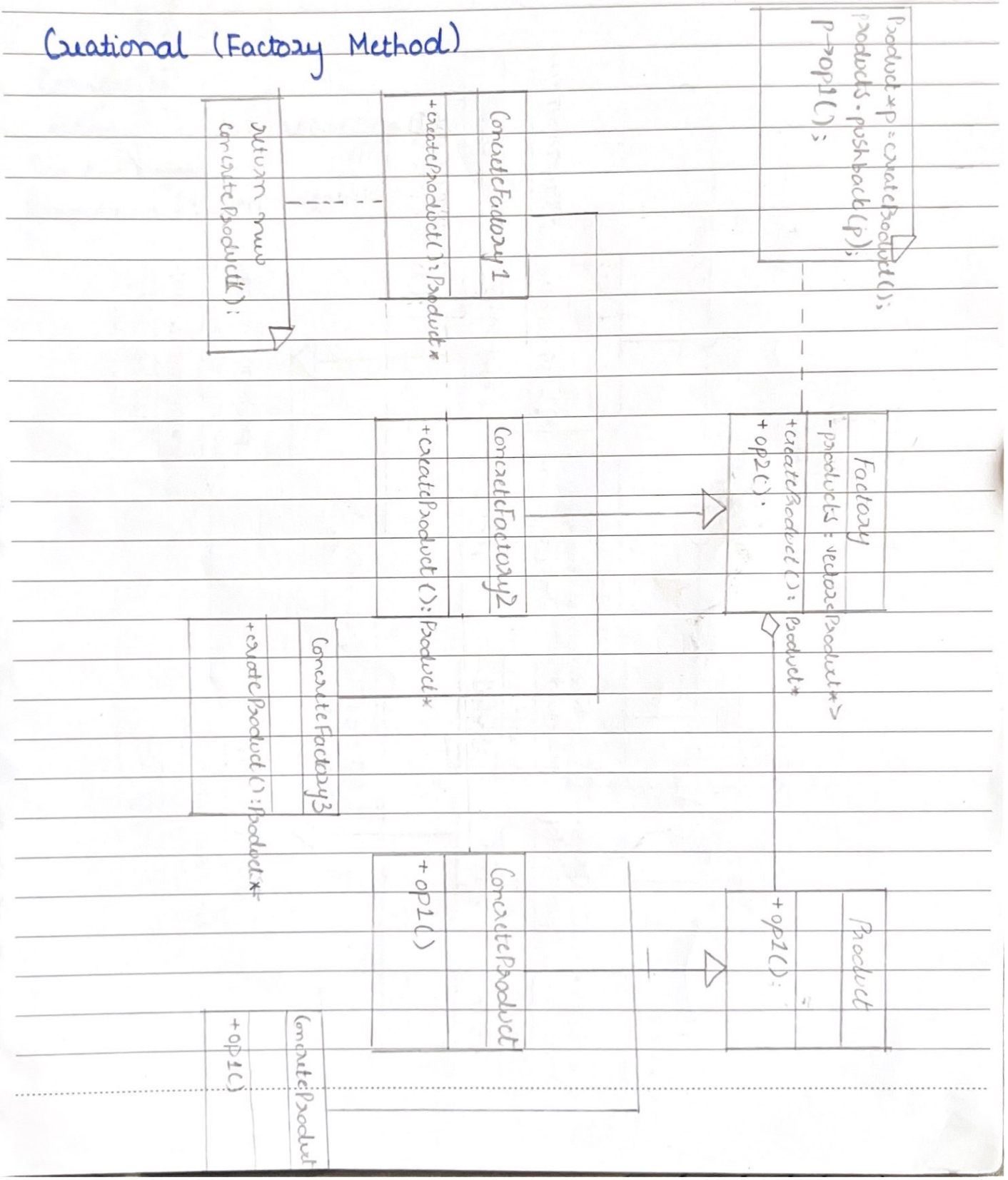
Behavioural: Deals with interaction of objects and classes.

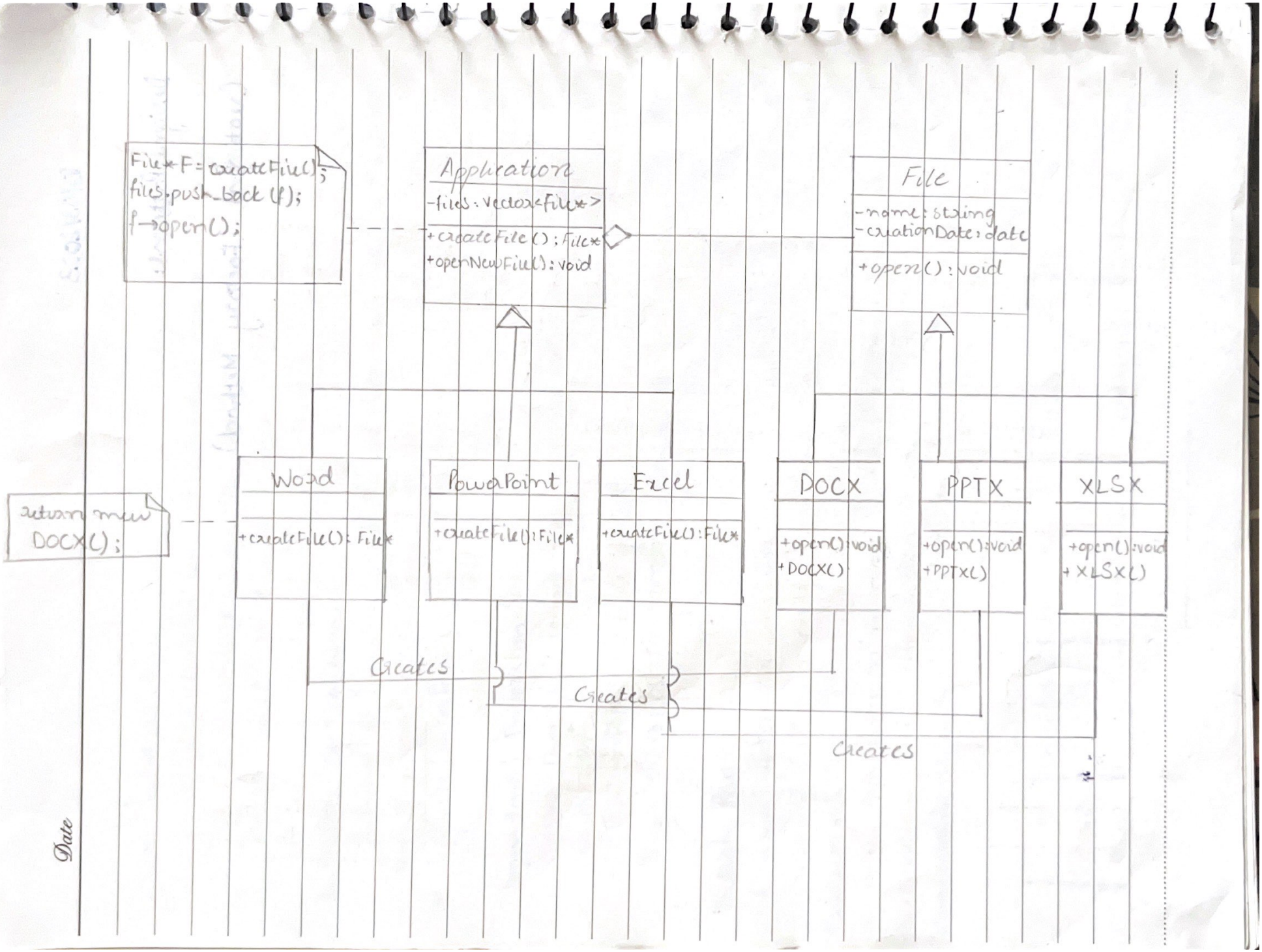
	creation	Structural	behavioural
Class	Factory		
Object		Composite	Observer.

Date 10/03/2023

Design Patterns:

Creational (Factory Method)




File* F = createFile();
files.push_back(f);
f->open();
Application
-files : vector<File*>
+createFile() : File*
+openNewFile() : void
File
-name : string
-creationDate : date
+open() : void
return new DOCX();
Word
+createFile() : File*
PowerPoint
+createFile() : File*
Excel
+createFile() : File*
DOCX
+open() : void
+DOCX()
PPTX
+open() : void
+PPTX()
XLSX
+open() : void
+XLSX()
Creates
Creates
Creates

Date

Date

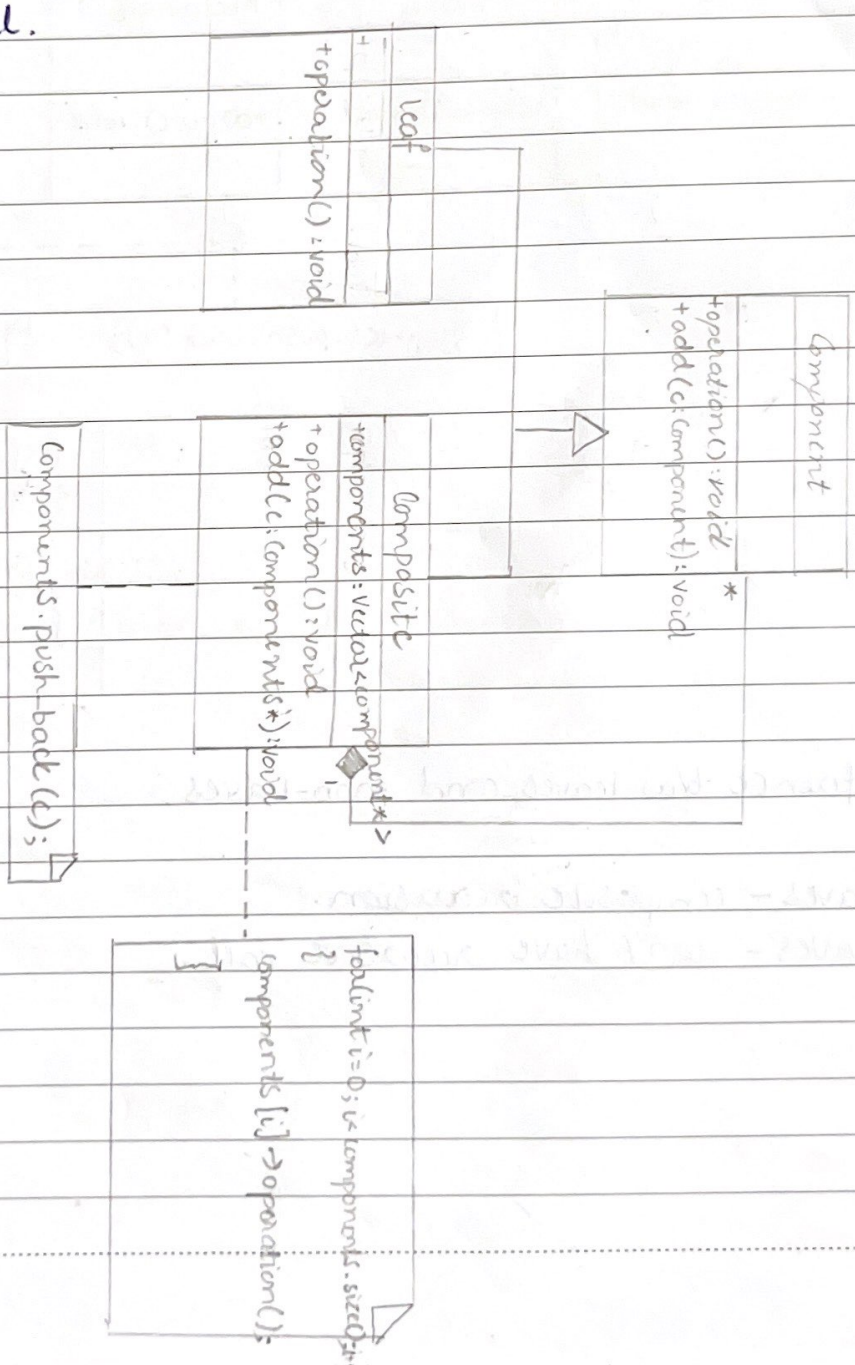
one extra level of indirection — essence of factory Method.

Composite

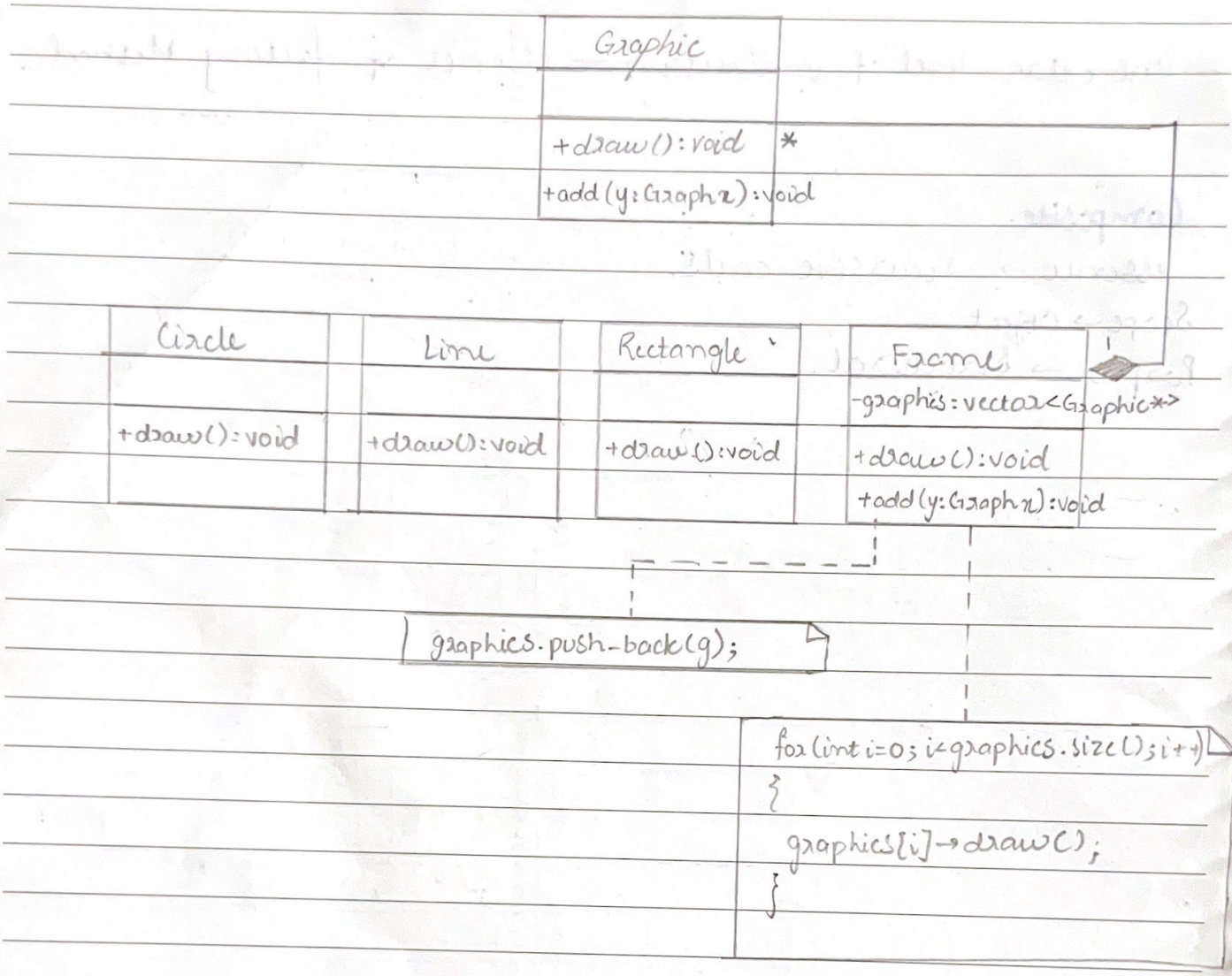
essence → recursive calls.

Scope → Object

Purpose → Structural.



Date



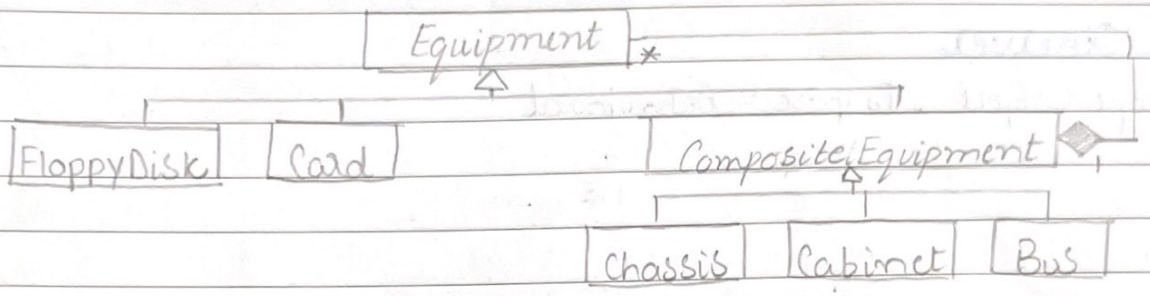
difference b/w leaves and non-leaves.

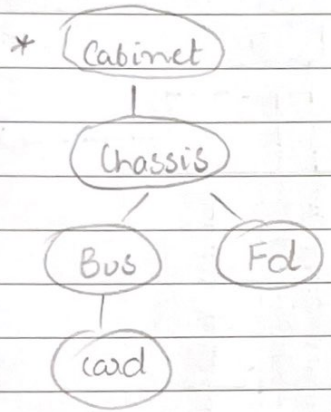
nonleaves - composite recursion.

leaves - don't have recursive call.

Gang of 4 book

Date





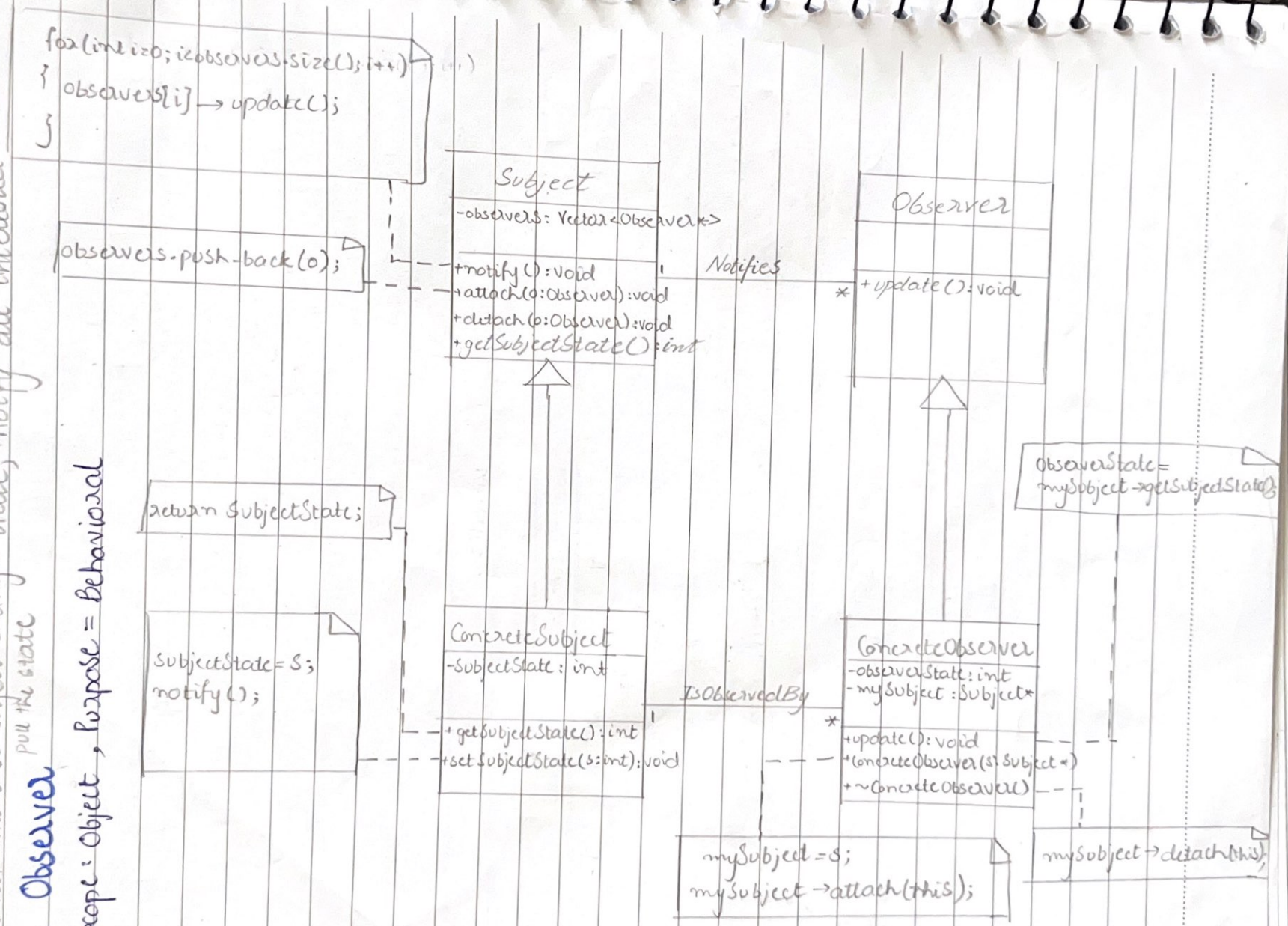
* Decorator. - wed April 5
less than 10 slides. (title slide included)

Date

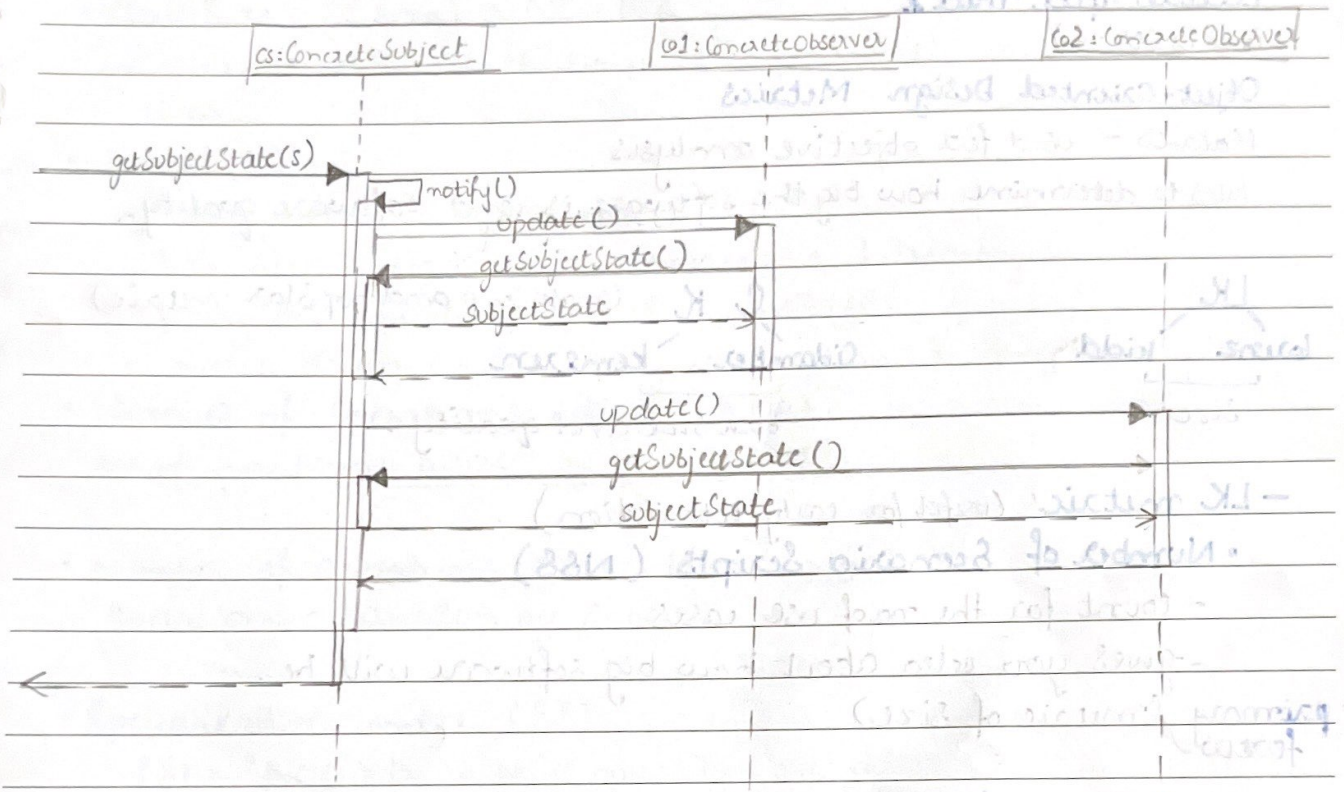
Essence – whenever subject change state, notify all interested observers

3. Observer (Pull the state)

Scope: Object, Purpose = Behavioral



Date


cs:ConcreteSubject
co1:ConcreteObserver
co2:ConcreteObserver
getSubjectState(s)
notify()
update()
getSubjectState()
subjectState
update()
getSubjectState()
subjectState

Date 12/04/2023

Lecture after mid 2

Book 3 (Ch: 6)

Object-Oriented Design Metrics

Matrics – used for objective analysis

helps to determine how big the software is and software quality.

LK
Lorenz Kidd
size

C K (most imp and popular metric)
Cidamber Kemerer
both size and quality.

- LK metric (useful for early estimation)
  - Number of Scenario Scripts (NSS)
    - count for the no. of use cases.
    - gives you idea about how big software will be.

primary focus → (metric of size)

  - Number of Subsystems (NSuS)
    - The no. of packages.
  - Number of key classes (NKC)
    key classes (domain class)
  - Number of supporting classes (NSC)
    no. of supporting / UI classes
  - Average no. of supporting classes per key class = NSC / NKC (How ACD becomes DCD)
  - No. of attributes (NA)
    applicable for single class. count inherited attributes
  - No. of operations (NO)
    consider inherited operations as well.

Date

- **Class Size : (CSize)** = NO + NA
  - estimate how much time is required to implement a class.
- **Depth (D) :**
  - class specific
  - how deep a class is in inheritance hierarchy.
  - proxy for how complicated a class is.
  - move down in hierarchy → the class becomes complex.
- **Number of Operations added (NOA)**
  no. of operations added by a subclass.
- **Number of Operations Overridden (NOO)**
  Operations overridden by subclass.
- **Specialization index (SI)**

$$\uparrow SI = \frac{\uparrow NOO \times D}{\downarrow NO} = \frac{\text{No. of operations overidden} \times \text{depth}}{\text{total no. of operations.}}$$

Date 14/04/2023

- CK metrics

- Weighted Methods per Class (WMC) = $\sum_{i=1}^{n} C_i = n$

  n = total no. of methods in class.
  each method is assigned complexity C.
  So it is sum of complexity for all classes.
  - WMC doesnot considered inherited operations.
    WMC = NO - inhaited operation.

- Depth of Inheritence Tree (DIT)
  It is the same as LK's Depth metric

- Number of Children (NOC)
  Simply count child of classes (only immediate children not grand classes)
  if no child class value = 0.

- Coupling Between Objects (CBO)
  count all types of associations with the class.
  Ideally, the value should be low.
  ↑CBO = difficult to reuse.

- Response For a class (RFC)
  all methods of class + all methods called by those methods.

Date

- Lack of Cohesion of Methods (LCOM)

Set of Instance variable (non-static)
Set of Methods M (non-static)
$M = \{M_1, M_2, \ldots, M_n\}$

Each M excess subset of I

$M_i \rightarrow I_i$
$M_1 \rightarrow I_1$
$M_2 \rightarrow I_2$

Define two sets.
$P = \{(I_r, I_s) \mid I_r \cap I_s = \phi\}$ $r \neq s$

$Q = \{(I_r, I_s) \mid I_r \cap I_s \neq \phi\}$

For example:
let I_r = Roll No
I_s = CGPA
So it will be a part of P as nothing is common.

$|P| > |Q| \rightarrow$ size of Q
↙ size of P

$$LCOM \begin{cases} |P| - |Q|, & \text{if } |P| \geq |Q| \\ 0, & \text{otherwise.} \end{cases}$$

* High value of $|Q|$ = lower value of LCOM = higher cohesion.
* High value of LCOM is bad

Date

$|P| + |Q| = \frac{n(n-1)}{2}$ (total no. of possible pairs)

Example: * use constructors and destructors defined by users.

I = {pno, name, price, qty, tax, dis}.

M = {create_product (M_1), show_product (M_2), retpno (M_3), retprice (M_4), retname (M_5), retdis (M_6)}

I_1 = {pno, name, price, dis}. variables used by 1 method

I_2 = {pno, name, price, dis}. " " M_2

I_3 = {pno}. M_3

I_4 = {price} M_4

I_5 = {name} M_5

I_6 = {dis}. M_6.

$P = \{(I_x, I_y) \mid I_x \cap I_y = \phi\}$

$Q = \{(I_x, I_y) \mid I_x \cap I_y \neq \phi\}$ } $x \neq y$

n = no. of methods so n = 6.

Total pairs = $\frac{n(n-1)}{2} = \frac{6(5)}{2} = 15$

$P = \{(I_3, I_4), (I_3, I_5), (I_3, I_6), (I_4, I_5), (I_4, I_6), (I_5, I_6)\}$

$Q = \{(I_1, I_2), (I_1, I_3), (I_1, I_4), (I_1, I_5), (I_1, I_6), (I_2, I_3), (I_2, I_4), (I_2, I_5), (I_2, I_6)\}$

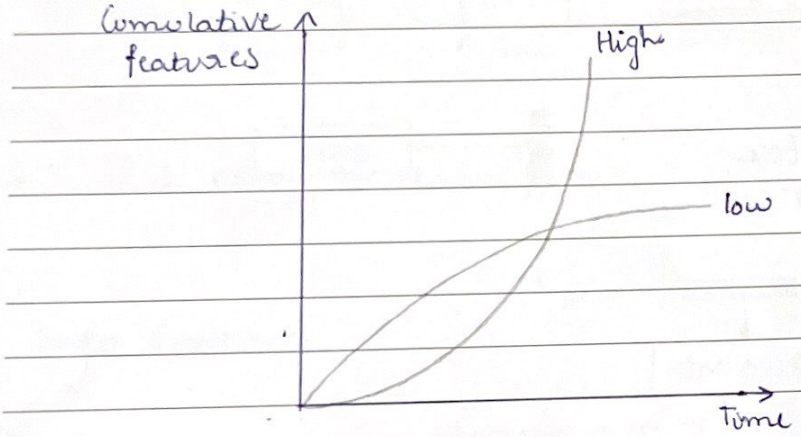
|P| = 6, |Q| = 9, if |P| > |Q| then LCOM = |P| − |Q|, otherwise = 0

So in this case 6 ≠ 9 so LCOM = 0 → class product.

Date

Architecture:

- high-level design — external / internal
- overall structure of software



High Quality Software → easier to add features.

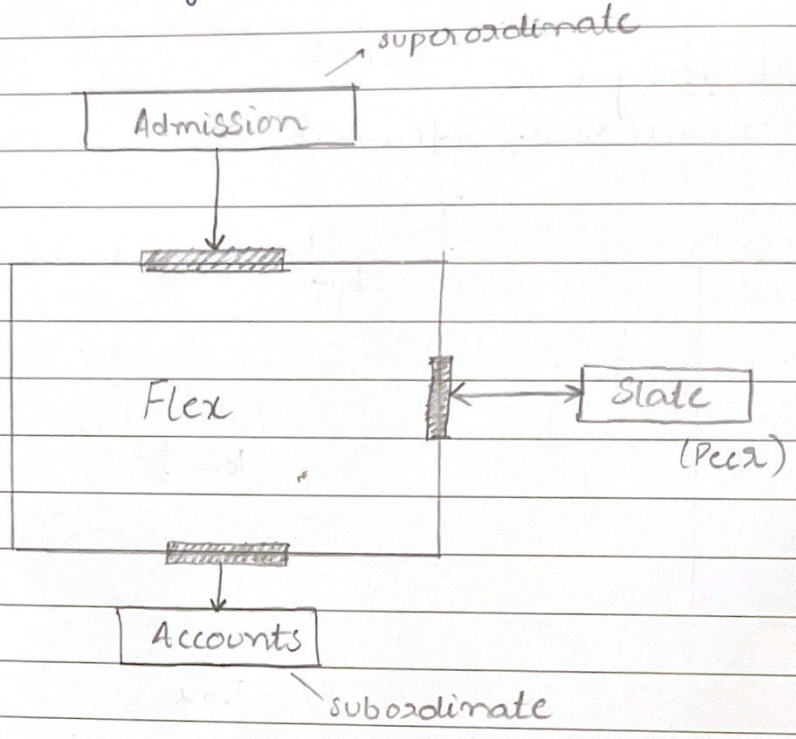
Design Stamina → high → good internal and external Quality.

Views

* Implementation view
* Deployment view
* Decomposition view - decompose systems into subsystems
* **Execution View** (our focus of study)
* Work assignment view.

Date

Architecture Context diagram:

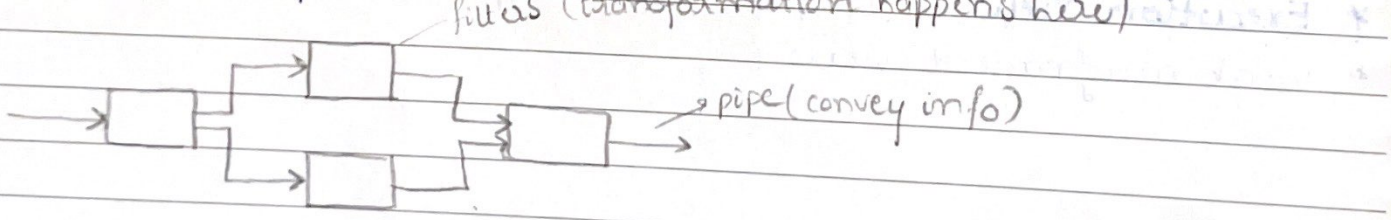


Architectural Styles and Patterns.

Architectural styles have more scope than patterns.

Architectural Styles:

- **Data Flow** (Pipe and filter)

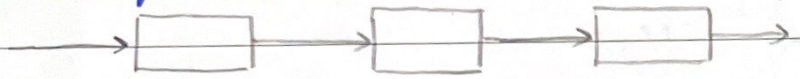


* many boxes and many pipes
* not very interactive

Date

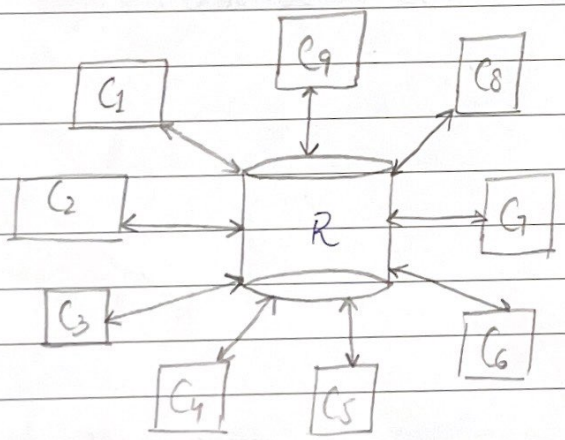
* mostly used in batch processing oninteractive systems
* Data processing applications used this.

Batch Sequential:



* Special case of data flow.

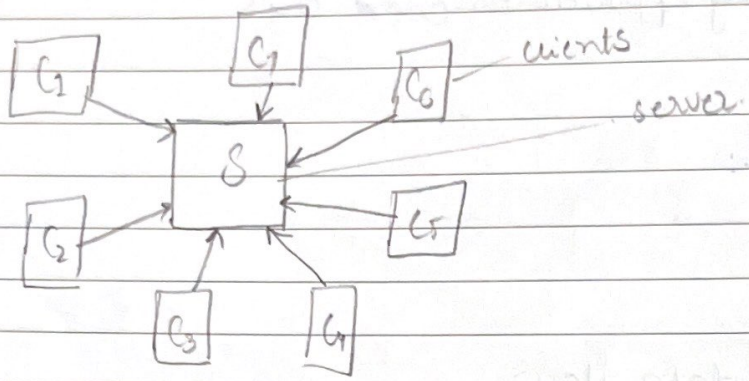
Data Centred



* The repositary can be active or passive.
* no data passing overhead.
* Drawback: if something happens to repositary, the system goes down.

Date

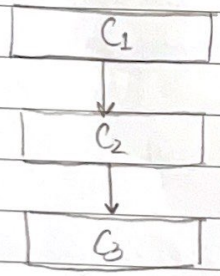
Client Server.



Peer-to-Peer (P2P)

* Every server that is a client is also a server.

Layered/Tiered

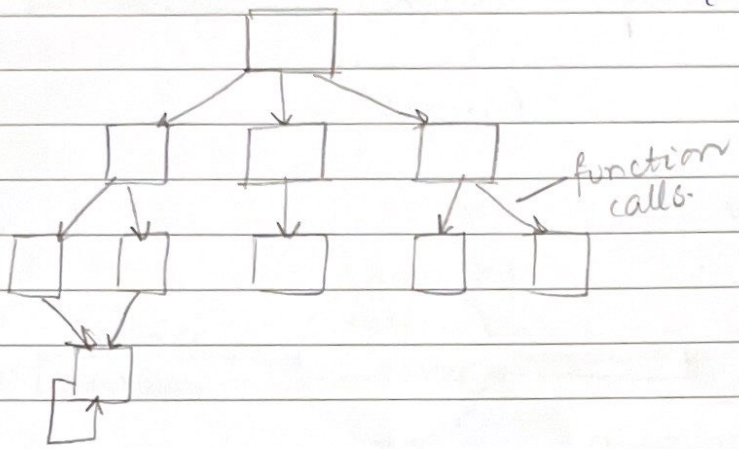


* Every layer acesses ther services of the layer below it.
* layer bridging is also allowed in impure version.
* achieving separation and independence

Date

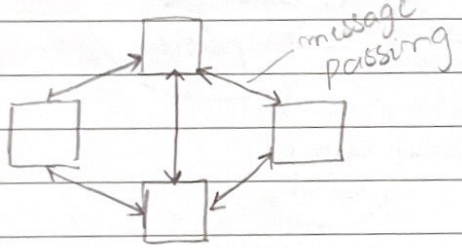
C & R (call and return)

* level of abstraction changes in call and return
* not consistent.



recursive calls are allowed.

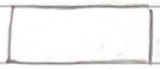
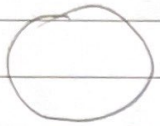
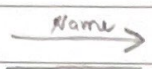
OOD (object oriented)



* DCD if all classes are considered as components

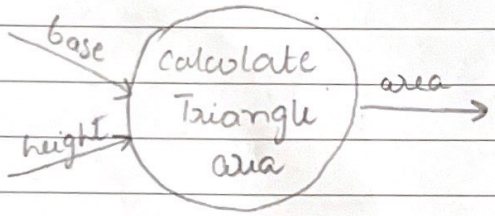
Date

DFD

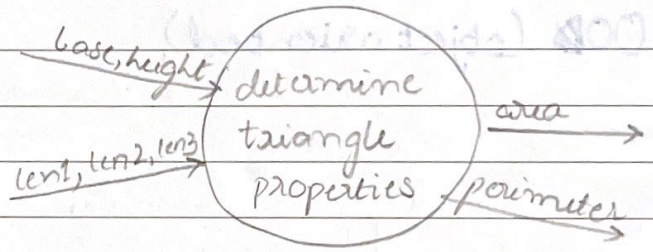
	External Entity.	(nouns)
	Process	(verbs)
	data flow	(nouns)
	data store	(nouns)

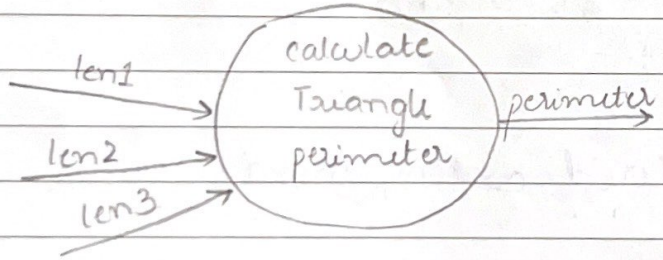
at level 0 → process name noun

Ln+1



Ln

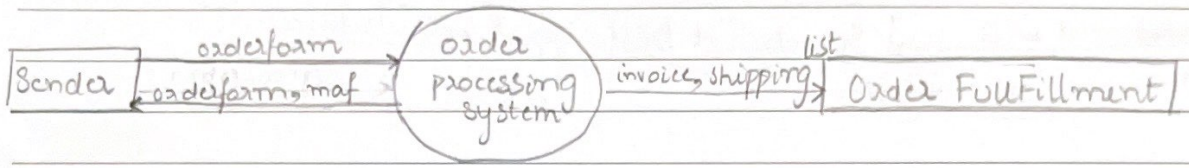




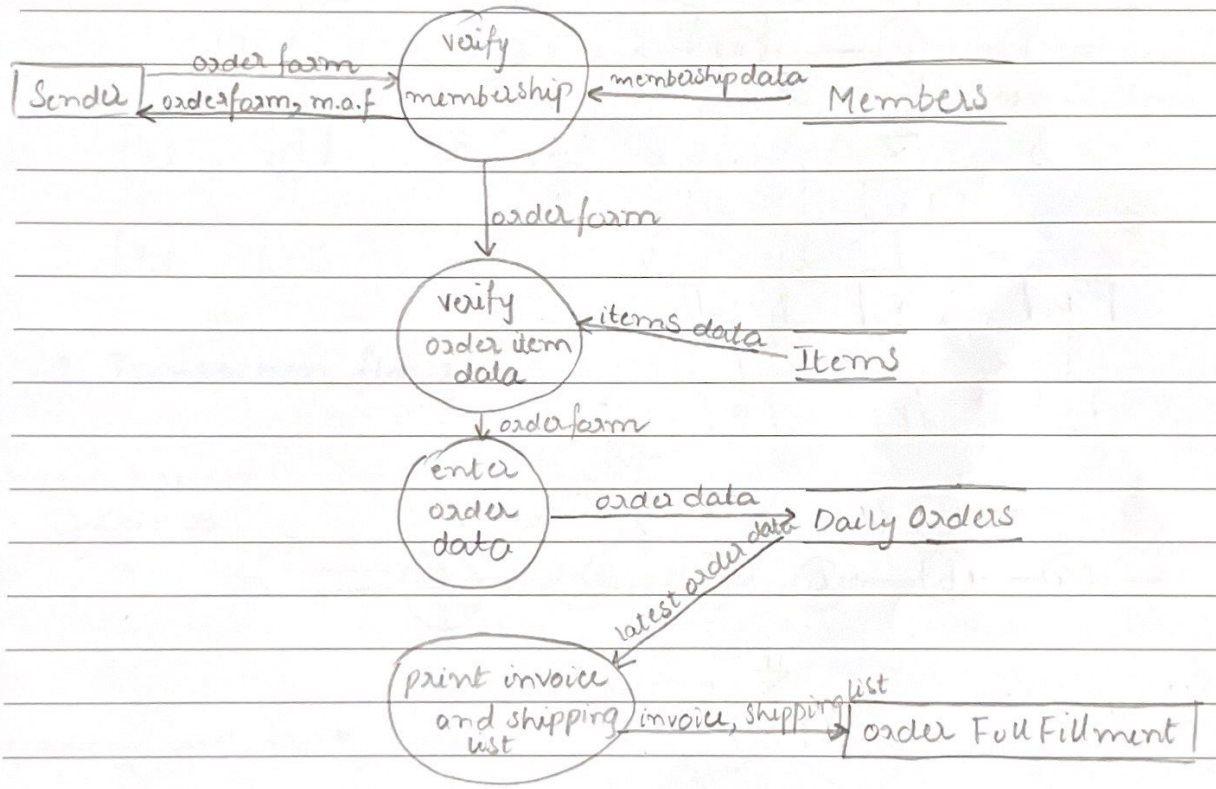
Date

DFD level 0

* in level 0 no data store



DFD L1

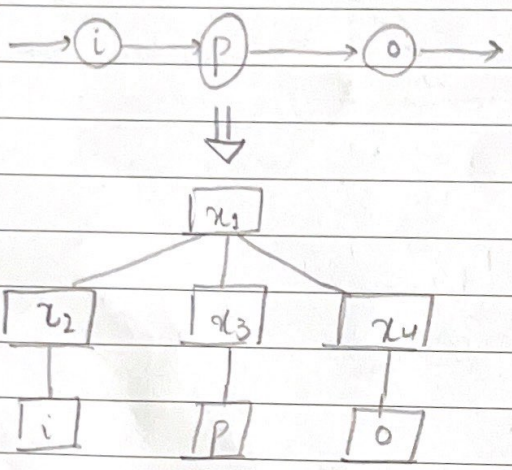


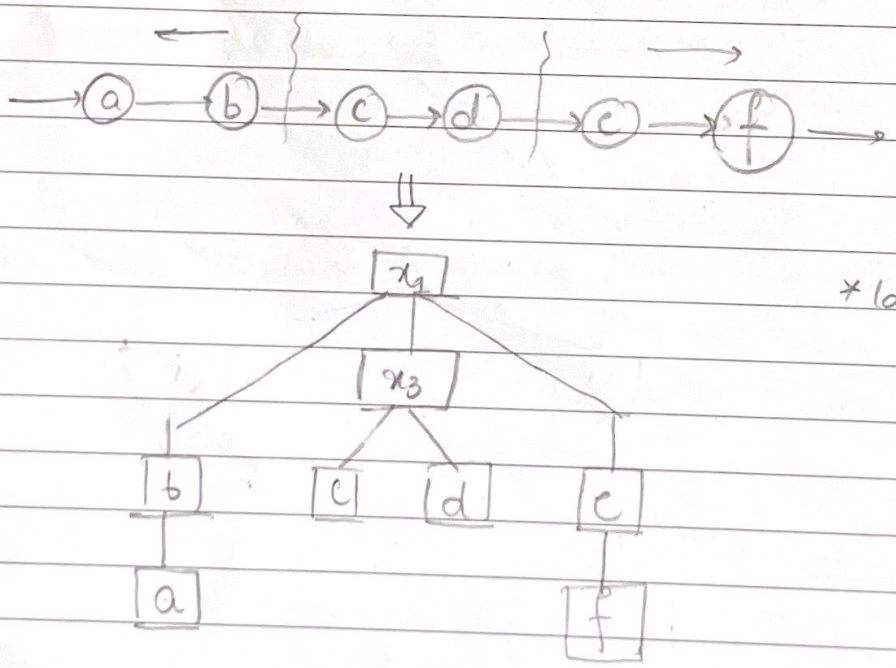
Date

Lowest level DFD → Structured Design → C&R Architecture

Information flow Type:

1. Transform Flow:

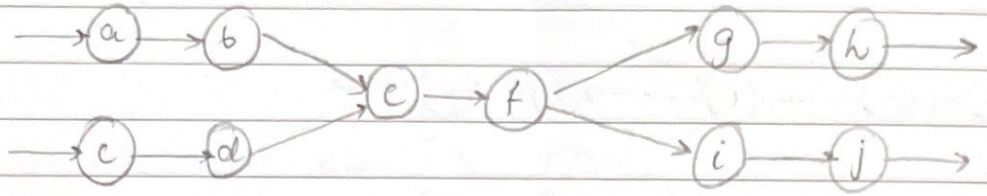




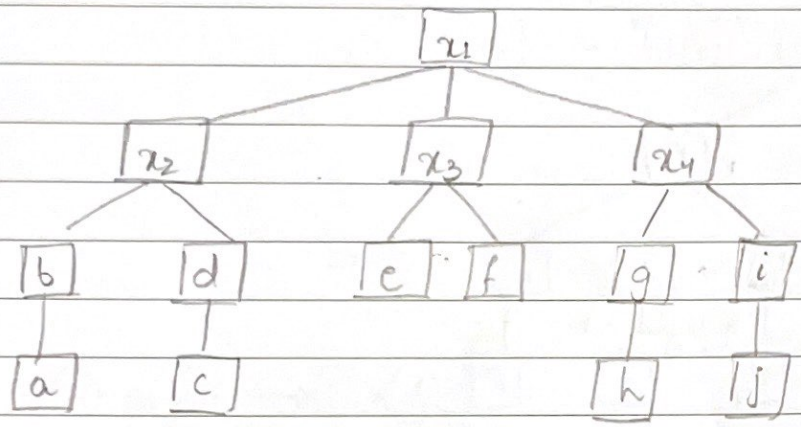
proued outsid
boundry.

* last step optimizati

Date

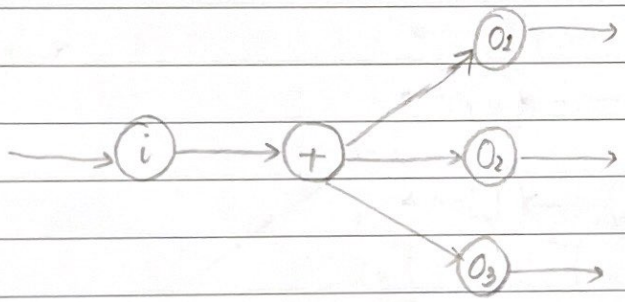


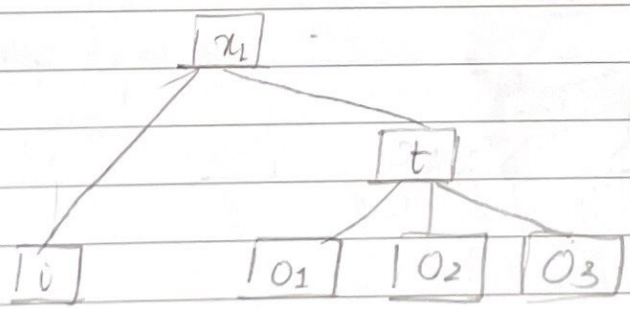
lowest level DFD



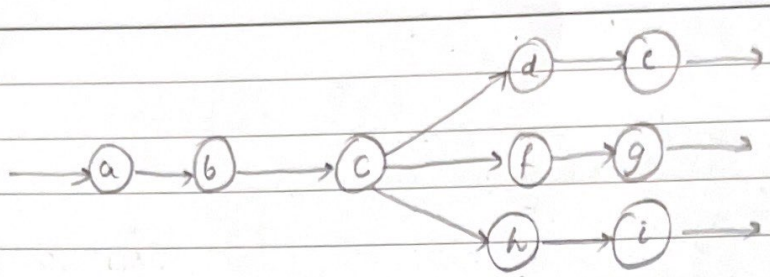
no optimization can be done.

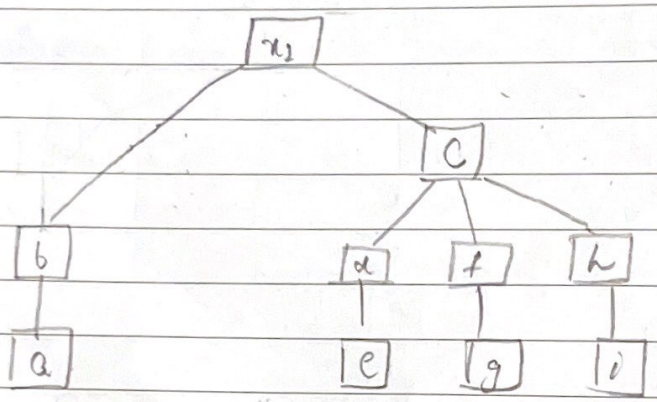
2. Transaction flow:



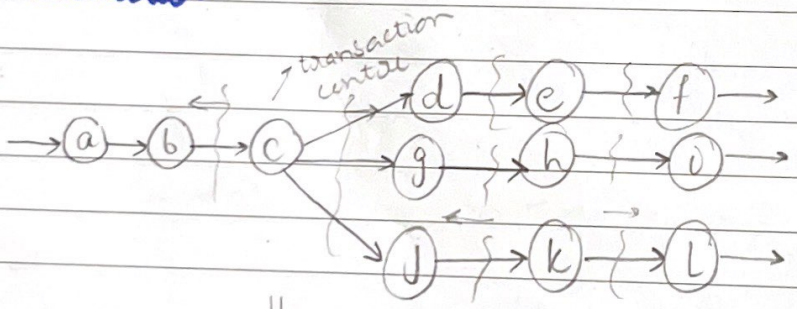


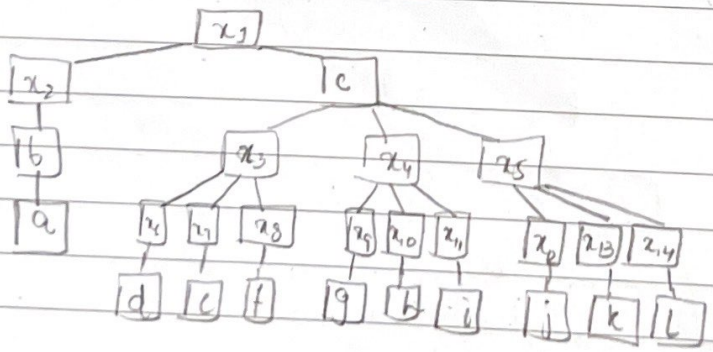
Date





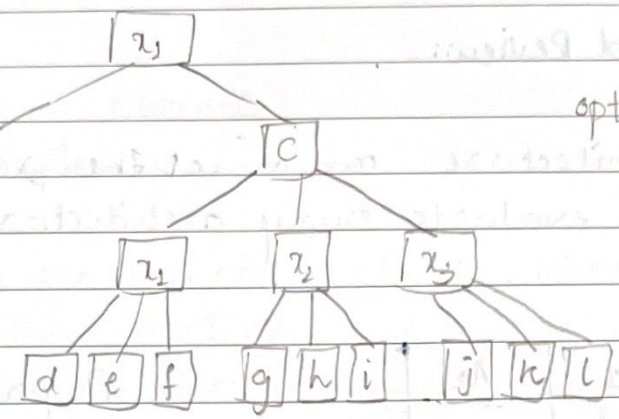
Scenario:





yet to be optimized.

Date



optimized architecture.