Week 8

L1 Software Architecture

- Software arch → way to organize your code, defines software elements (modules), rela^ns among them, & properties of both elements & rela^n.
- Client-server sys. → data is transacted in response to req
- Pipe & filter → data is passed from 1 comp. to comp., & transformed & filtered along the way.
- MVC → style where views of data are separated from manipula^ns of data.
  - Model → comp. which models the data eg. for service
  - View → GUI obj, presenta^n layer, visual represenita^n.
  - Controller → coordinates multiple views on the screen, & help users manipulate the model.
- P2P arch → distributed app. (diff. sys. form nodes, & share resources with each other), no centralised sys. that monitors all Transac^n of sys., node in the network make processing easier.
- Component → A well-defined functionality or behaviour sep. from other func. & behaviour.
- Connector → Code that transmits info. btw. components. (Responsible for regulating interac^ns btw. comp.
  - Protocols → set of pre-defined rules which describe how comp. should interact with each other.
- Design Pattern → descrip^n of communicating obj. & classes that are customized to solve a general design problem in a particular context.
- Design smell → dearming that your code maybe heading towards an anti-pattern.
- SOLID guidelines helps to avoid design smells.
- Refactoring → moving code btw. classes, create new classes, remove unnecessary classes.

M T W T F S S

Page No: YOUVA

Date:

12 Solid Principles - I

- Obj.-oriented principles for software design. Given by Robert Martin, makes code easier to understand, modular
- S → Single Responsibility
  ↳ every class must have only 1 responsibility / purpose
- O → Open-closed
  ↳ software entities should be open for extensions but closed for modificaⁿ. Extend a class behaviour (by sub-classes) w/o modifying existing classes.
- L → Liskov substitution
  ↳ applies to inheritance principles, derived classes should be substitutable by their base class.

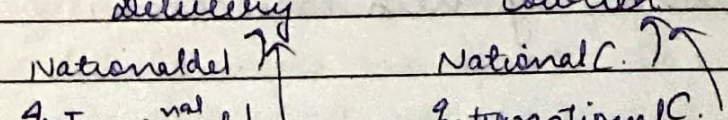
13 Solid Principles - II

- Interfaces ⇒ grp. of related methods with empty bodies, specify what interface should do, not how. Other classes can implement the interface by implementing all methods in the interface.
- I → interface segregaⁿ
  ↳ do not force any client to implement an interface which is irrelevant to them.
- D → dependancy inversion
  ↳ prefer abstracⁿ (interfaces) over implementaᵐ.

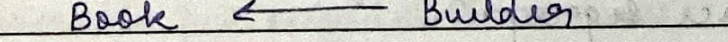
14 Creational Design Patterns

- Elements of design pattern
1. Problem – when to apply, what context
2. Soluⁿ – describe the elements that make up the design, their relaⁿ, etc. Template that can be applied.

3. Consequences – results, tradeoff, cost benefit
- types of design pattern –
1. creational – used during the process of obj. creaⁿ
2. structural – composiⁿ of classes or obj.
3. behavioural – characterize the ways in which classes or obj. interact & distribute responsibility.
- creational → Factory & Builder design pattern
- Factory ⇒
- functionality – need to deliver package using courier.
- replace direct obj. constructⁿ calls (using the new operator) with calls to a spl. factory method.

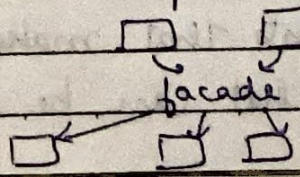


- Pros → single responsibility, open-closed
- Cons → code becomes complicated – a lot of new subclass needed to implement the pattern.
- Builder ⇒
- extract the obj. constructⁿ code out of its own class & move it to sep. objects cfd builders.



- Pros → construct obj. step-by-step, single-responsibility
- Cons → code becomes more complicated, new creation

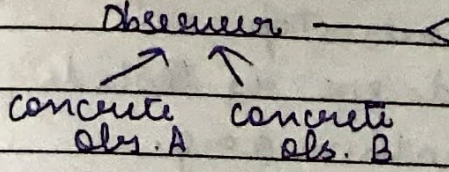
L5 Structural patterns
- Facade ⇒
- create several obj. + perform several steps, use existing complicated library funcⁿˢ
- provides a simple interface to a lib., or a complex set of class.



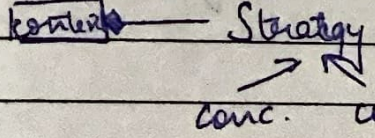
- pros ⇒ isolate code from other classes' complexity
- cons ⇒ tightly coupled to other obj., maintenance becomes more difficult.
- Adapter ⇒ Adaptee
  client → Adaptor
- pros ⇒ single responsibility, open-closed
- Cons ⇒ overall code complexity increases.

16 Behavioural patterns

- Iterator ⇒
- separate the behaviour of how elements are accessed into a sep. obj. cld an iterator.
- Java - interface - iterator (hasNext(), next())
- pros → single responsibility, open closed
- cons → can be an overkill for simple collections
- Observer ⇒
- subj. obj. maintains a list of observers, notifies them.



- Strategy ⇒
- extract diff. strategies into sep. classes, original class delegates the work to strategy obj.



- pros → isolate implementation details, open-closed
- cons → not req. if there are only a few algo which will be used.