Strategy pattern

main class → client

i) code that changes <> code that remains same.

DP code এর যে part গুলোতে change হয় সেগুলো আলাদা করে encapsulate করতে হবে যাতে পরে কোনো change আসলে unchanged part এ কোনো affect না পরে,

→ Scalibility তে benefit পাওয়া যায়

→ Modularity এর কারণে Testing এ benefit পাওয়া যায়,

Context : Code এর যে part change হয়না, calculator, player, car, Duck
Behaviour/strategy: ..... " " করবার change হয়, operation, run/quibble, horn, fly

ii) Change ==> code in interface/abstraction. not in implementation

Interface defines signature of it's object

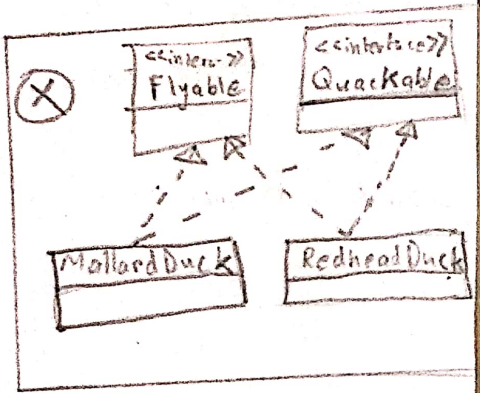
DP → setter() method দিয়ে dynamically behaviour change করা যায়,

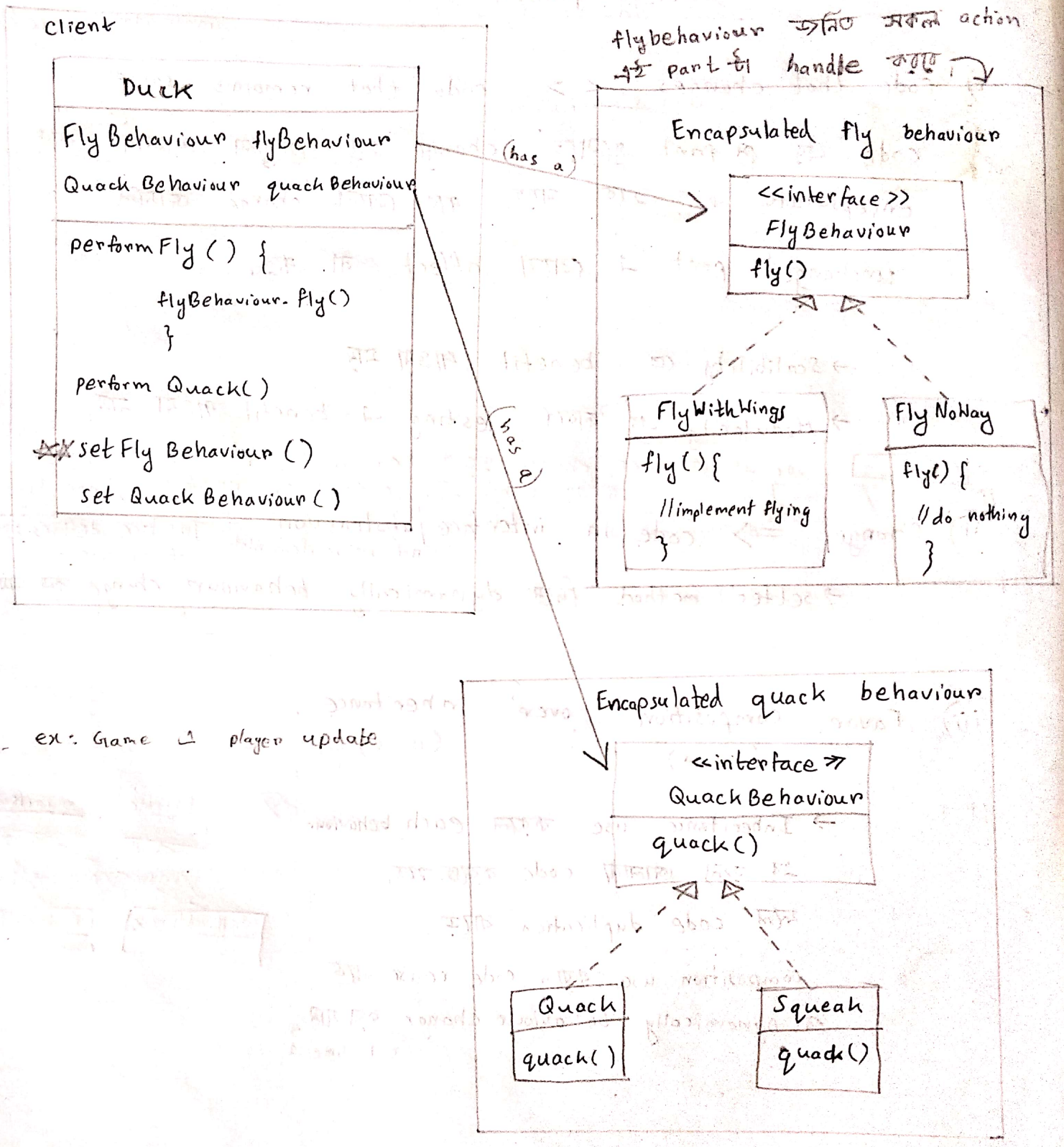
iii) Favor composition (has a) over inheritance (is a).

DP → Inheritance use করলে each behaviour এর জন্য আলাদা code করতে হবে, ফলে code duplication বাড়ে.

Composition use করলে code reuse বাড়ে এবং Dynamically behaviour change করা যায়

run time




client
Duck
FlyBehaviour flyBehaviour
QuackBehaviour quackBehaviour
performFly() {
flyBehaviour.fly()
}
performQuack()
setFlyBehaviour()
setQuackBehaviour()
(has a)
(has a)
flybehaviour এনিত সকল action
এই part টা handle করবে
Encapsulated fly behaviour
<<interface>>
FlyBehaviour
fly()
FlyWithWings
fly(){
//implement flying
}
FlyNoWay
fly(){
//do nothing
}
Encapsulated quack behaviour
<<interface>>
QuackBehaviour
quack()
Quack
quack()
Squeak
quack()
ex: Game → player update

Observer pattern

i) Subject এবং Observer এর মধ্যে one-to-many dependency থাকে, সেখানে subject এর state change হলে তার উপর dependant observer গুলো automatically update হয়।

ii) Strive for loosely coupled designes between subject and observer.

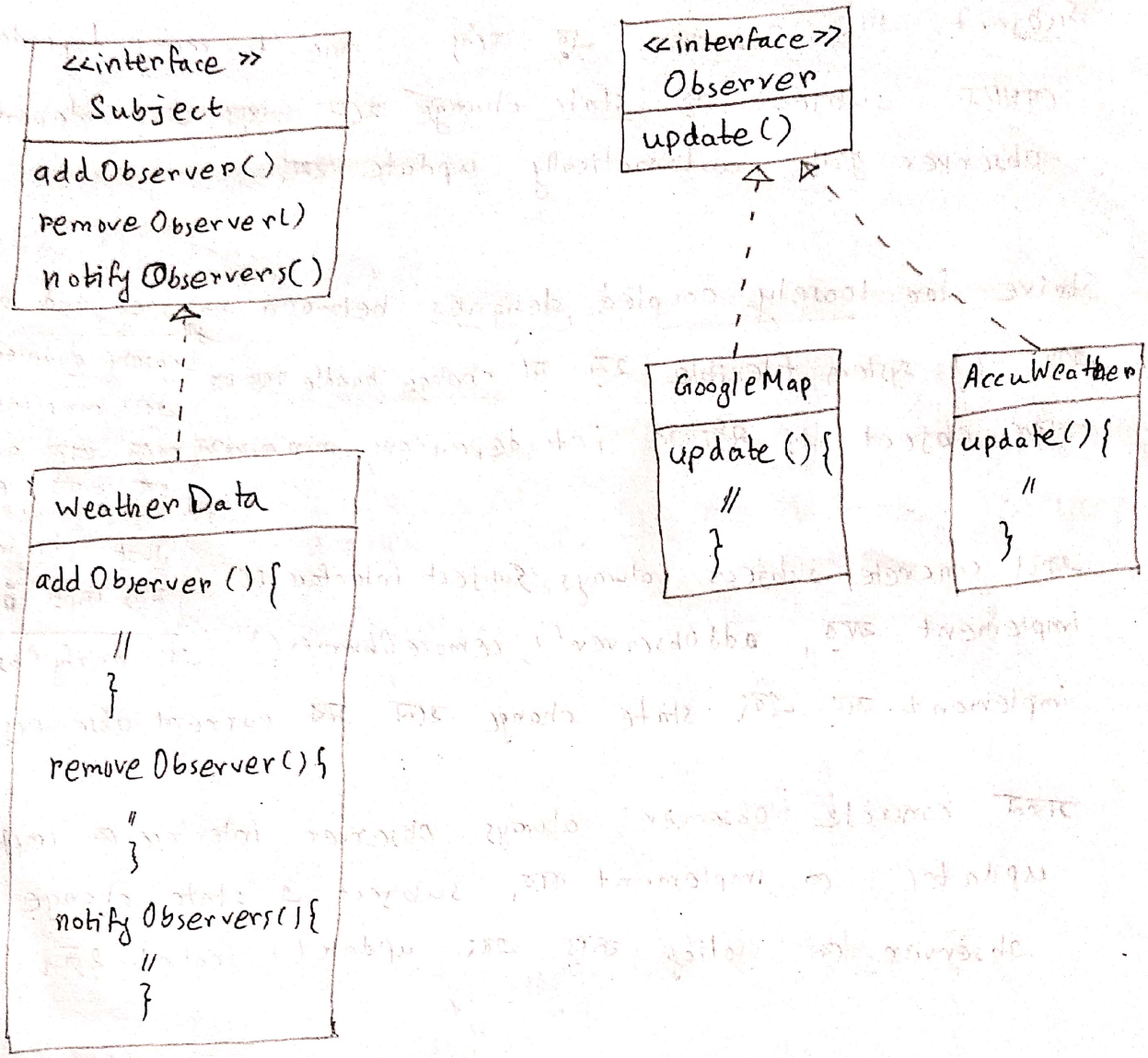
এতে OO system flexible হয় যা change handle করতে পারে, কারণ object এর মধ্যকার interdependency minimized হয়।

DP

Loosely coupled:
একটা component এ change করলে অন্য component এ খুব কমই effect পড়ে, Just 1টা interface এর মধ্য দিয়ে interact করে।

একটা concrete subject always Subject interface কে implement করে, addObserver(), removeObserver() এবং notifyObservers() implement করে এবং state change হলে সব current observers কে update করে।

সকল concrete observer always Observer interface কে implement করে, update() কে implement করে, subject এ state change হলে observer কে notify করে এবং update() called হয়।


<<interface>>
Subject
addObserver()
removeObserver()
notifyObservers()
<<interface>>
Observer
update()
GoogleMap
update(){
//
}
AccuWeather
update(){
//
}
WeatherData
addObserver(){
//
}
removeObserver(){
//
}
notifyObservers(){
//
}

Decorator pattern

object কে wrap করে.

i) Decorator দিয়ে dynamically object এর behaviour extend করি.

→ Inheritance use করলে শুধুমাত্র compile time এ statically behaviour determine করা যায়, কিন্তু composition use করলে run time এ (main function) decorator দিয়ে behaviour extend করা যায়,

তাই আমরা composition use করি এবং এতে flexibility বাড়ে,

open-close principle

ii) Classes should be open for extension, but closed for modification of existing code.

→ নতুন functionality add করার ক্ষেত্রে, আমরা existing code alter না করে, নতুন code লিখে functionality add করি,

এর ফলে pre-existing tested code এ bug বা side effect কম হয়,

iii)

```
abstract BaseItem
    getDescription()
    cost()

Item extends BaseItem
    getDescription()
    cost()

abstract BaseDecorator extends BaseItem
    BaseItem ob

Decorator extends BaseDecorator
    getDescription()
    cost()
```

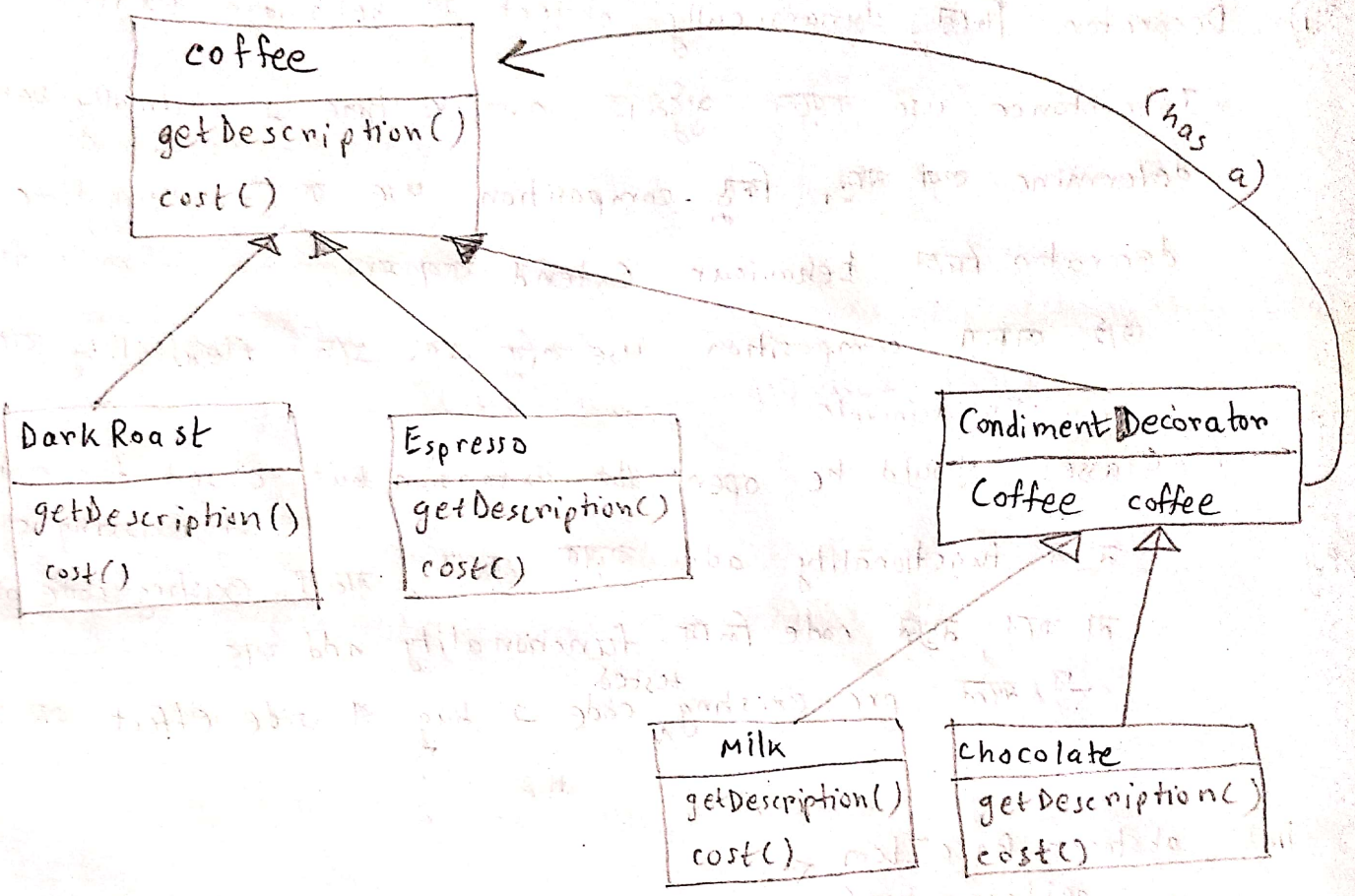
(has a)

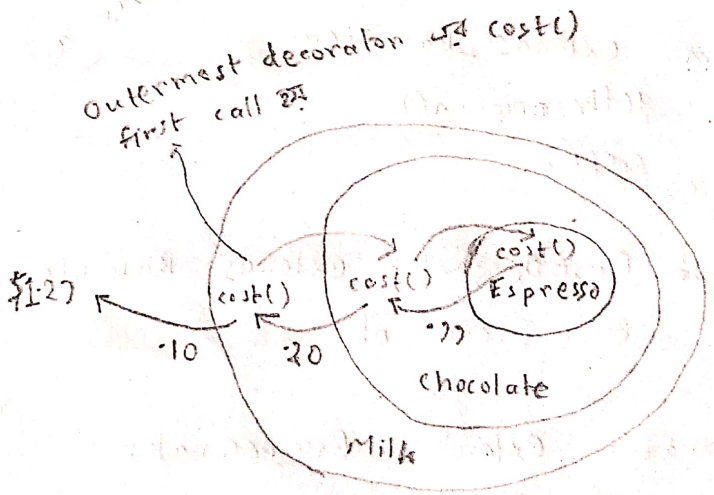
// Decorator এবং Decorator দিয়ে item decorate করে তারাই same interface implement করে,

// Base Decorator এ BaseItem এর একটা instance থাকে, এটা decorator যে item কে decorate করবে তাকে point করে

Coffee coffee = new Espresso()

coffee = new Chocolate(coffee)



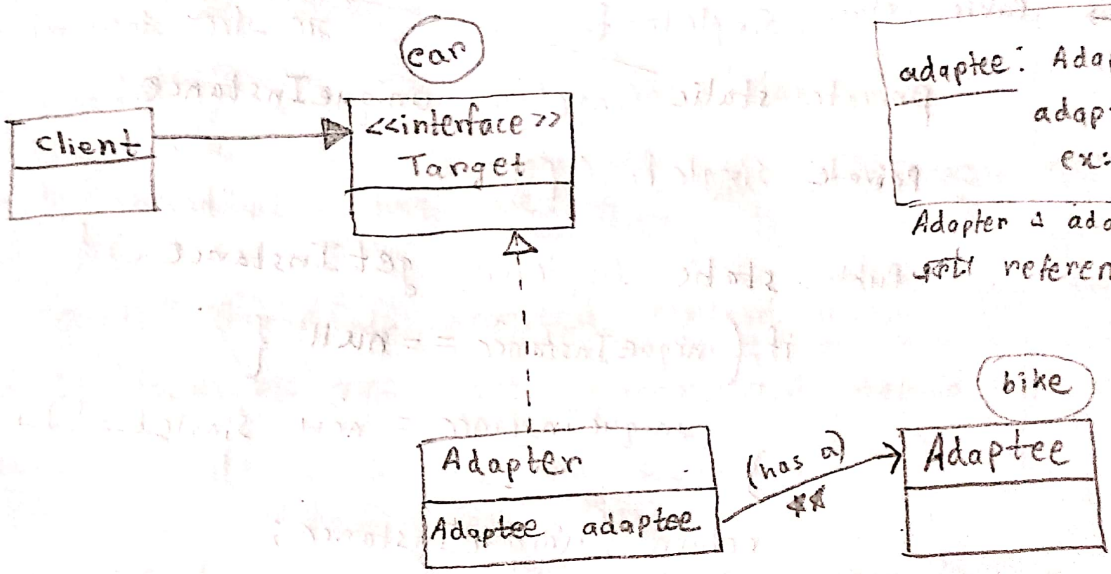


Adapter pattern

i) Converts the interface of a class into another interface that (one or more) the clients expect.

দুইটা class এর interface incompatible হলে adapter দিয়ে তাদের একসাথে কাজ করানো যায়।

ii)



adaptee: Adapter যাকে adapt করে, ex: bike

Adapter এ adaptee এর একটা reference থাকে।

Client এর প্রয়োজন হচ্ছে Target interface (car), But আমাদের কাছে আছে Adaptee (bike)

Adapter, bike এর interface টাকে car এর সাথে adapt করাবে,

Adapter, Car interface কে implement করে এবং Car এর method call গুলো নিয়ে bike এর equivalent method গুলো call দিবে।

iii) Adapter class change গুলোকে encapsulate করে।

অর্থাৎ, Car class এবং bike class এ কোনো হাত দিতে হয় না।

Singleton pattern

i)
→ Singleton pattern ensure করে যে একটা class এর at a time শুধু একটা instance ই থাকবে এবং তাকে access করার জন্য একটা global point থাকবে (static getInstance() method).

ex: Gpu driver object, Register settings object

ii)
→

```
Pubic class Singleton {
    private static Singleton uniqueInstance;
    Private Singleton() { }
    Public static Singleton getInstance() {
        if (uniqueInstance == null) {
            uniqueInstance = new Singleton();
        }
        return uniqueInstance;
    }
}
```

reference variable টা static, যাতে এটার static method থেকে access করা যায়।

Private constructor — যাতে class টাকে বাইরে থেকে instantiate না করা যায়।

Static uniqueInstance — Singleton class এর একমাত্র instance টা hold করার জন্য।

Static getInstance() — class টা instantiate করার একমাত্র way।

এটা lazy instantiation কারণ instance টা কখনো প্রয়োজন না হলে এটা create ও হয় না।

iii)

→ Multithreading system এ multiple thread getInstance() method কে access করে multiple object create করে ফেলতে পারে,

(1ম Thread null condition satisfy করে instance create করতে যে সময় নেয় ঐ সময়ে আরেকটা Thread এসে null condition satisfy করে ফেলতে পারে, এর ফলে 2টা object create হয়ে যায়)

Solution:

i) synchronized keyword use করে, (getInstance(), fill(),...)

→ multiple thread at a time synchronized method গুলোকে access করতে পারবে না,

But কোনো Thread synchronized method access করলে সম্পূর্ণ class টা lock করে ফেলে, তাই synchronized method optimized না, synchronized block use করা better.

একবার Singleton initialize হয়ে গেলে synchronization এর আর প্রয়োজন পড়ে না,

Thread গুলা always যাতে Latest value টার access পায়

** ii) synchronized block use করে double-checked locking.

Volatile keyword ensure করে যে multiple Thread uniqueInstance কে correctly handle করবে,

```
public class Singleton{
    private volatile static Singleton uniqueInstance;
    private singleton() { }
    public static Singleton getInstance () {
        if(uniqueInstance == null) {
            synchronized (Singleton.class) {
                if (uniqueInstance == null) {
                    uniqueInstance = new Singleton();
                }
            }
        }
        return uniqueInstance;
    }
}
```

1st Null condition ensure করে যে, একবার uniqueInstance create হওয়ার পরে আর synchronized block এ ঢুকবে না। efficient করে।

2nd Null condition ensure করে যে, first time synchronized block এ ঢুকার পর যাতে multiple Thread, object create করে না ফলে।

* Synchronized block use করার জন্য JVM 5 প্রয়োজন হয়,

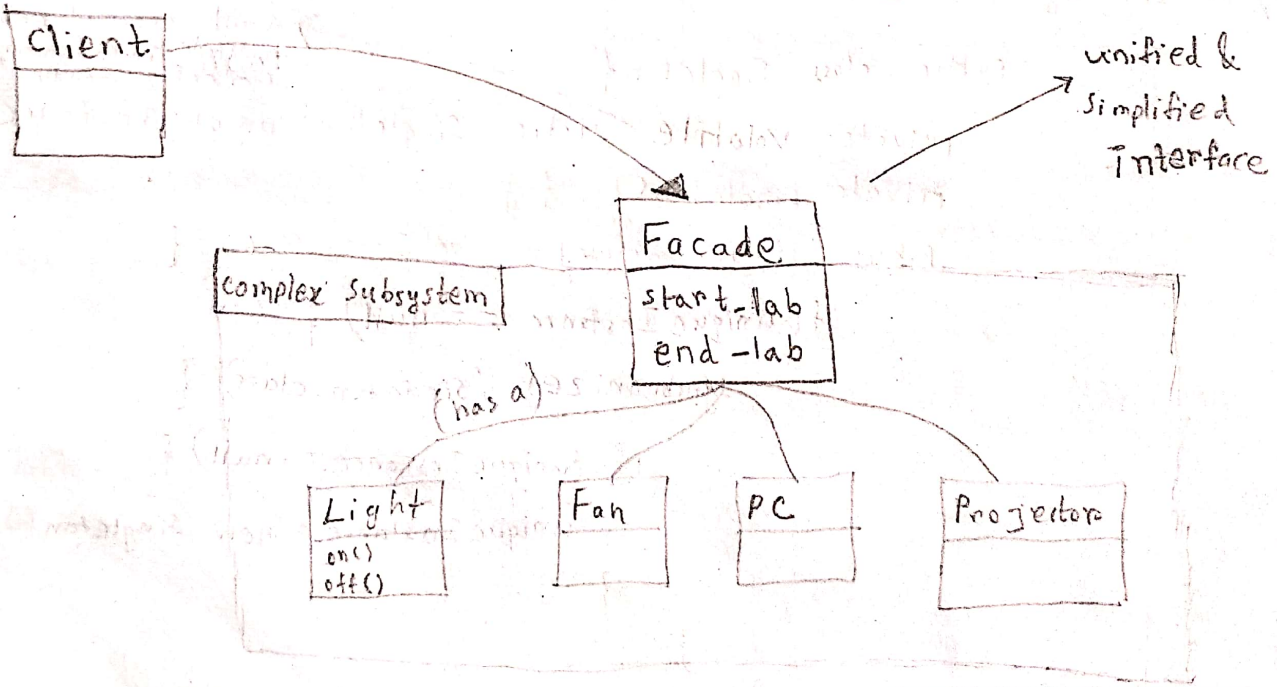
iii) Eager instantiation: Private static Singleton uniqueInstance = new Singleton();

Facade pattern

i) Facade pattern provides a unified and simplified interface to a complex subsystem. (Lab, Home theatre)

ii) Facade decouples a client from a subsystem.
→ Subsystem এর component এ কোনো change (PC upgrade) আসলে, client code change করতে হয় না, কারণ client যে code করা হয়েছে facade এর জন্য, subsystem এর জন্য না,
But Facade আর subsystem আগের tightly coupled,

iii) Principle of least knowledge
→ class গুলোর মধ্যকার dependency (Projector with PC) বেশি হলে system টি fragile হয়। System এর এক part এ change করলে অন্য part এ affect ফেলে, ফলে maintenance cost এবং complexity বাড়ে, তাই Facade manages all those subsystem components.



Decorator wraps an object to extend behaviour
Adapter " " " to change its interface
Facade " " " to provide simplified interface

iv) Open-closed principle violate করে,
→ subsystem এ component add করলে বা change করলে existing code (facade) এ হাত দিতে হয়,

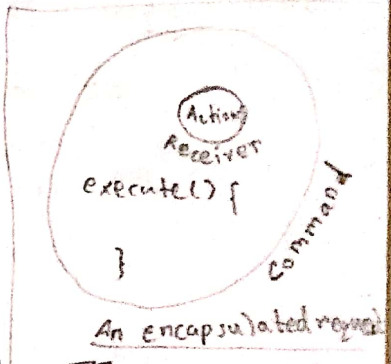
v) Single responsibility violate করে না, কারণ Facade নিজে কাজ করে না, কাজ করায়,

Command Pattern

→ একটা Home Automation System, Programmable remote থাকবে যেখানে each slot এ ইচ্ছেমতো command set করা যাবে (run time এ)
Remote এর command এর মাধ্যমে device গুলো বিভিন্ন action perform করবে,
light, fan light on

☆☆ Remote, vendor class (device controller) এর খুটিনাটি জানতে পারবে না এবং future এ নতুন vendor class আসলে existing code modify করা যাবে না,
এজন্য command pattern RemoteControl class এবং Vendor class কে decouple করে,

request encapsulate command হিসেবে



An encapsulated request

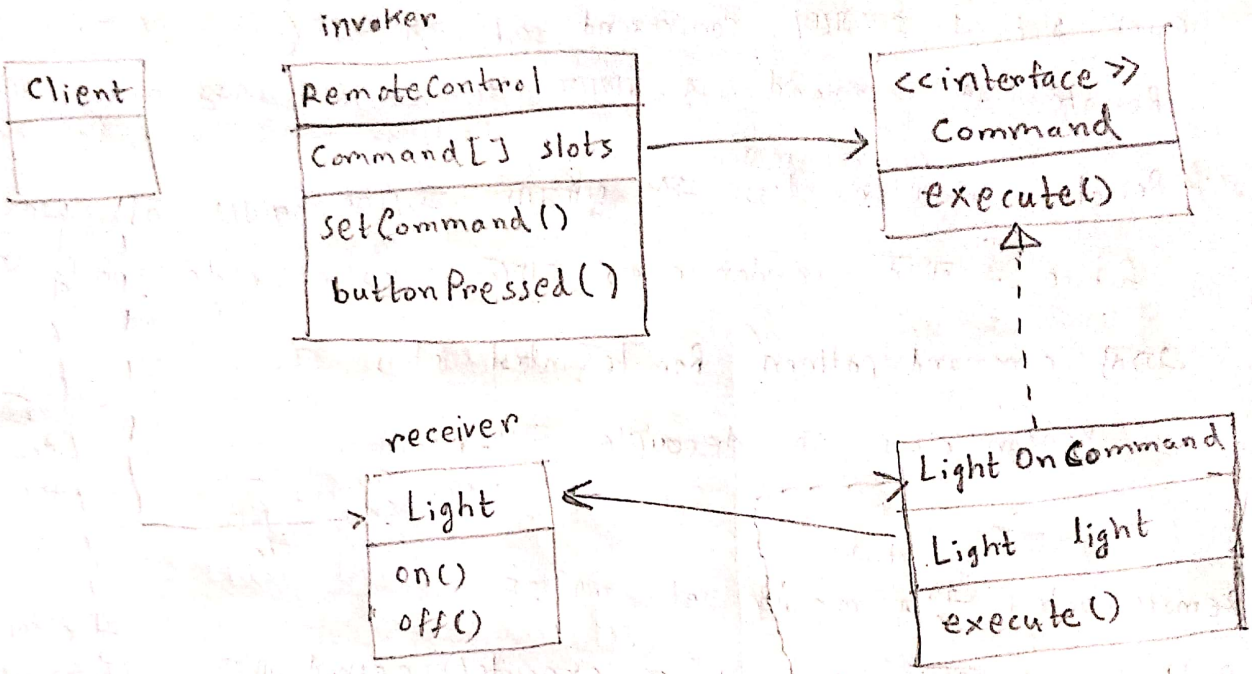
slot এ

→ RemoteControl এর ^ Command set করা হয়,
ButtonPressed হলে command এর execute() method call হয়,
execute() method, vendor class এর method call করে
বিভিন্ন Action perform করে,

ফলে remote এর slot এ Different type of request যাওয়া যায়, Ac, light, fan on

Customer	client	client
Waiter	Invoker	RemoteControl
order	Command	Command
Cook	Receiver	Light, Fan

→ Facade pattern open-closed principle violate করলেও Command pattern তা মেনে চলে,


invoker
Client
RemoteControl
Command[] slots
setCommand()
buttonPressed()
<<interface>>
Command
execute()
receiver
Light
on()
off()
LightOnCommand
Light light
execute()

Prototype pattern

→ Prototype design pattern use করে existing কোনো object কে state সহ clone করা যায়,

Object creation costly হলে (যখন database operation প্রয়োজন হয়) Prototype pattern use করে object clone() করা better. (Game এ tree, building, villain)

→ naive approach: ঐ class এর new object create করে original object এর field value new object এ copy করা যায়,
But এক্ষেত্রে private field গুলো বাইরে থেকে access করা যায় না,

→ prototype pattern:

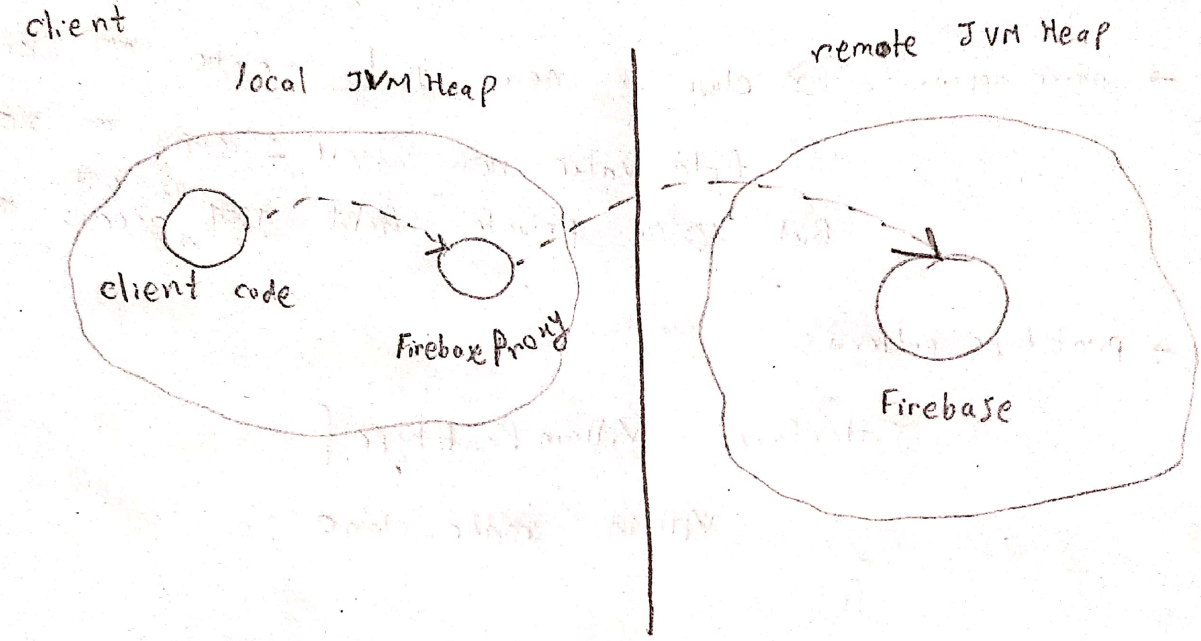
```
Interface   Villain Prototype {
        Villain   make_clone();
    }

abstract  class  Villain  implements Villain Prototype {
        abstract  void  attackProtagonist();
    }

Class   VillainType1   extends  Villain {
 private  int  hp, mp;
  VillainType1 (  )  {.}
  void  attackProtagonist() { . }
  Public  Villain  make_clone() {
          return  new  VillainType1 (this.hp, this.mp);
    }
}
```

→ Java তে clonable কে extend করে clone করা যায়,

Proxy Pattern

→ Proxy Pattern remote object এর access control করার জন্য surrogate বা placeholder provide করে।

→ Firebase, youtube video cache, Cloud Database



→RMI use করে Proxy pattern implement করা হয়।

MVC Pattern → Compound pattern

→ MVC pattern code কে functional segment এ divide করে, Development এ সহজ করে।

model: Model maintains all the data and it's state.

▷ যেমন ORM দিয়ে DB এর table কে class হিসেবে access করা যায়,
▷ শুধুমাত্র Model ই Database কে access করতে পারে,

View: Client end এ model এর data কীভাবে represent হবে, তা view ঠিক করে,

▷ Frontend

Controller: View, controller এর মাধ্যমে Model কে access করে,

→ Client, view এর সাথে interact করে,

View তে কোনো action perform করলে তা controller handle করে,
↳ Strategy pattern

Controller ই action perform এর জন্য প্রয়োজন অনুসারে model এর state change করে,

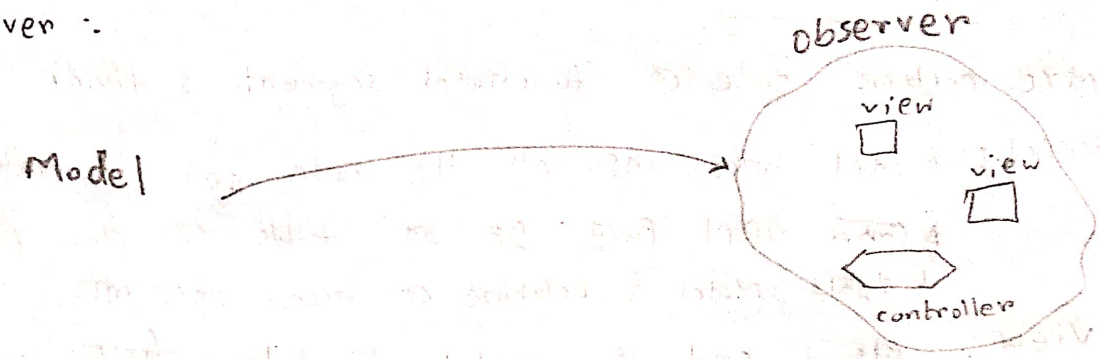
Model এর state change হলে সে view কে notify করে
↳ Observer pattern

→ Strategy:

Controller হচ্ছে view এর behaviour, Different behaviour পাওয়ার জন্য View controller এর different implementation use করে,

User action গুলো controller perform করে, ফলে View এবং Model decoupled থাকে,

→ Observer :



Model এর state change হলে observer (view, controller) কে notify করবে এবং তারা update হবে.

→ Adapter :

Adapter use করে অন্য একটা model কে adapt করে existing view এবং controller এর সাথে কাজ করানো যায়,

DJ Beat ← Adapter — Heart Beat

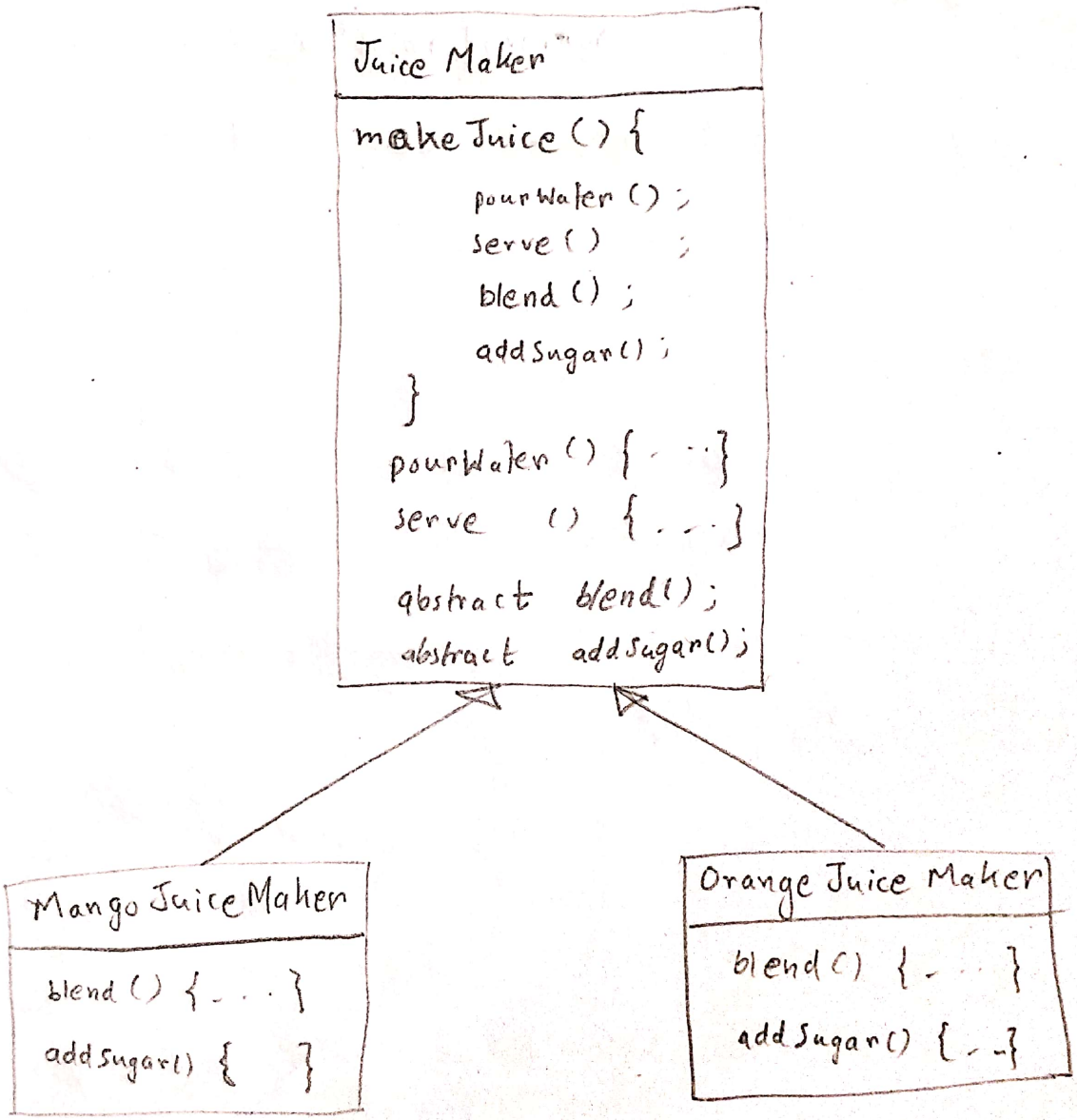
Target interface Adaptee

Code reuse বাড়ে.

Template pattern

→ Parent class এ Template method (algorithm) define করা থাকে, Template method এর common method গুলো parent class এই define করা থাকে, বাকি method গুলো subclass এ define করা হয়, Code reuse বাড়ে।

→



SOLID

→ makes software design understandable, flexible, maintable, scalable

Parallel development, Loose coupled (High cohesion), modular, testable, code duplication ↓, code reuse ↑

Dynamic.

Single responsibility

(method, class)

→ Every module should have one and only one responsibility and thus only have one reason to change.

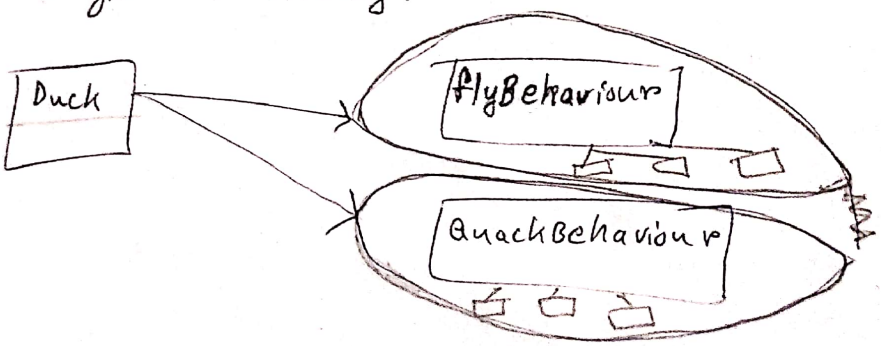
- এই নিয়ম module গুলো Loosely coupled হয়

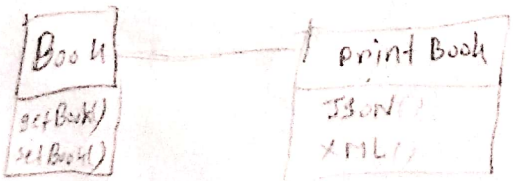
→ method এ single responsibility.

setInformation (String Name, String Autor) ✗

setName (String Name) ; setAuthor (String Author) ✓

class এ single Responsipibility.





Open closed principle

→ classes should be open for extension, but closed for modification. (of existing code)

নতুন functionality add করার ক্ষেত্রে আমরা existing code alter না করে, নতুন code লিখে functionality add করি,

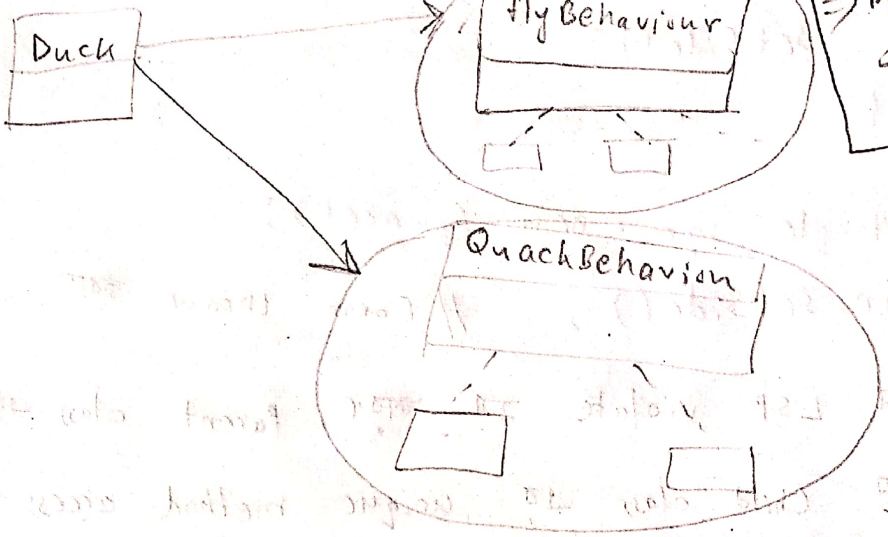
এর ফলে already tested module গুলো আর test করতে হয় না, just new functionality গুলো test করতে হয়।

⇒ Testing ↓ benefit ↑

⇒ Maintenance cost ↓

⇒ Dynamically run time ↓ Behaviour change করা যায়

→



Lishov Substitution principle

→ Parent Class এর reference দিয়ে Child Class এর object কে refer করলে code এর functionality তে যাতে কোনো সমস্যা না হয়,

→
```
Class Rectangle{
        setHeight();
        setWidth();
    }

class Square extends Rectangle {
        setSide();
    }

Rectangle sqr = new Square();
 sqr.setSide();  // Error throw করবে.
```

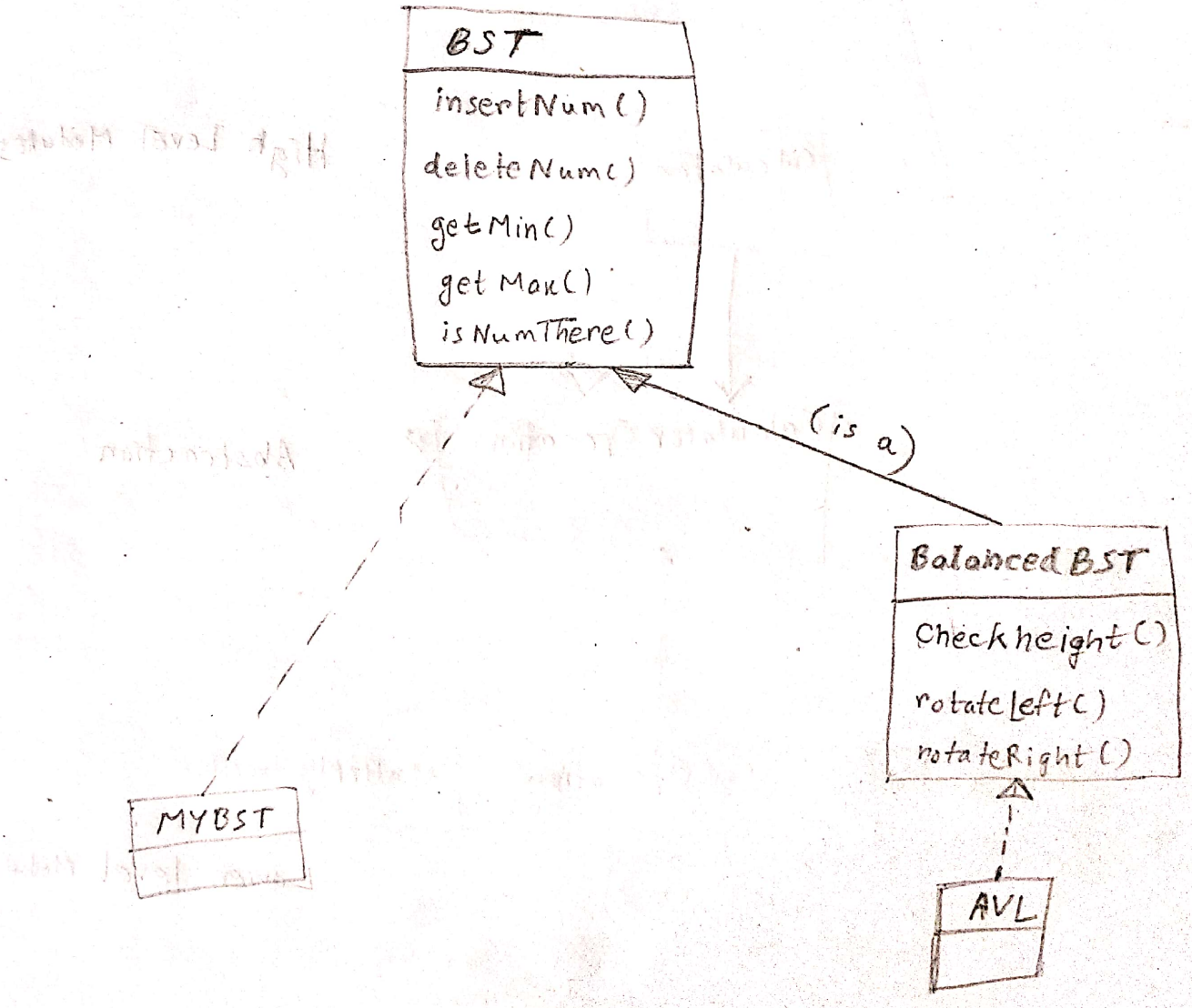
Bad design, এখানে LSP violate হয়, কারণ Parent class এর reference দিয়ে Child class এর unique method access করা যায় না,

→ Template (Rectangle) Parent class টা কে এমনভাবে design করতে হবে যাতে Child class (Square) এ নতুন কোনো functionality (setSide()) implement করতে না হয়,

✱→ LSP satisfy হলে automatically (OCP, DIP) satisfy হবে,

Interface Segregation

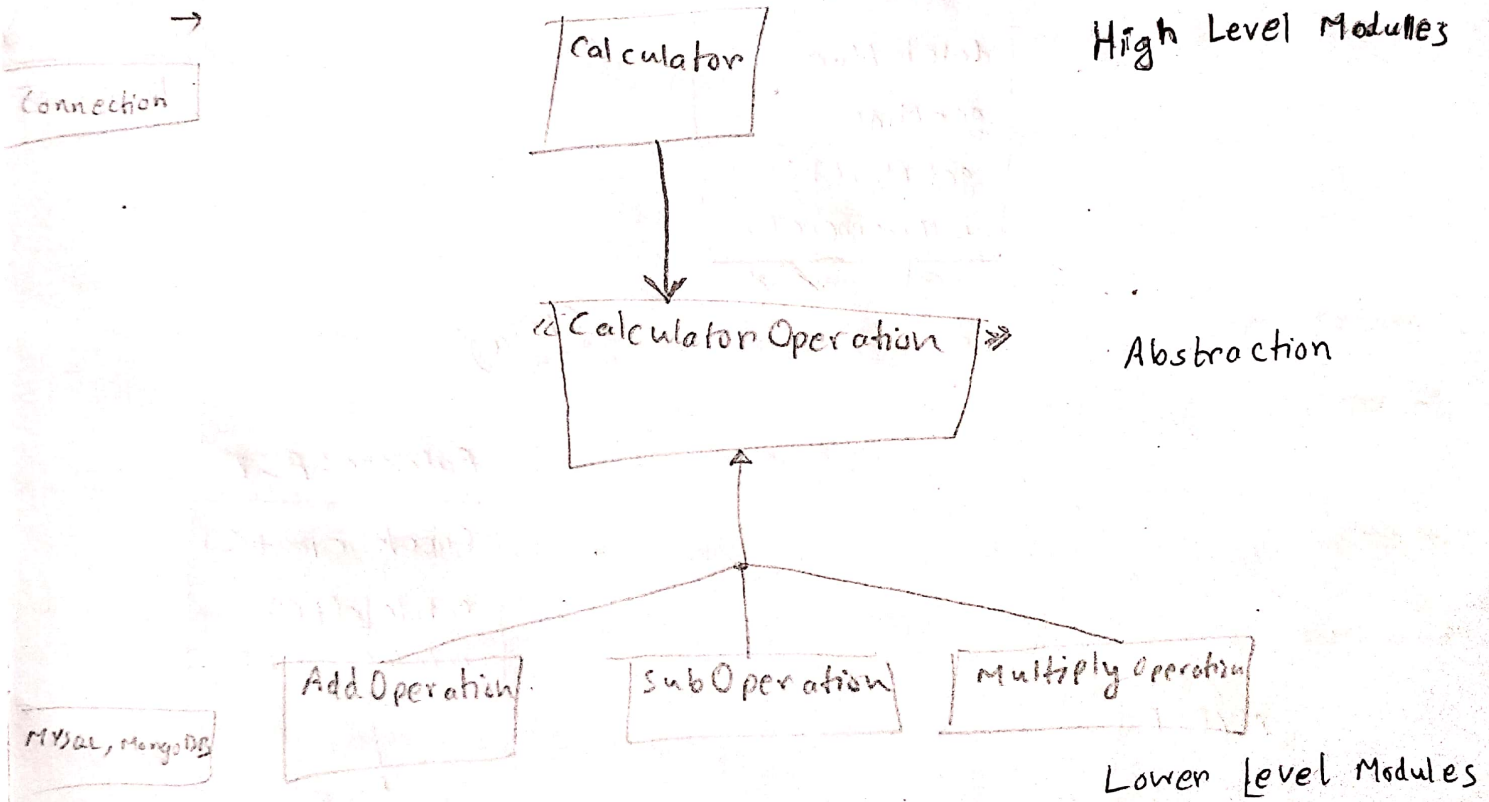
→ কোনো একটা interface কে যারা implement করে তাদেরকে যদি unnecessary functionalities implement করতে বাধ্য করা হয়, তখন বুঝতে হবে যে Interface টাকে ভেঙে Modular করা প্রয়োজন।



Dependency Inversion

→ High level module গুলো directly low level module এর উপর depend করবে না, উভয়েই Abstraction এর উপর depend করবে,

→ Abstraction high level module এবং low level module কে decouple করে, ফলে low level module এ change হলে High level module এ effect পড়ে না,



SWE

Objective : Resource এবং effort optimize করে Good software develop করা।

কম খরচে

aspects of SWE :

- Software Specifications
  - Market Analysis
  - Requirement Engineering
  - Determining Features/Service
  - Feasibility check করা
- Software development :
  - Software Architecture design
  - Implementation
- Software validation
  - software Testing
  - Quality Assurance
- Software Evolution
  - Maintenance
  - Add new feature/service.

Essential attributes of a good Software :

- Maintainability | reusable, modular, Scalable.
- Acceptability | client have to accept the software.
- Dependability and security | System failure ↓ physical/economical damage হয়না।
  Malicious user ⊗
- Efficiency | Smart resource allocation, responsiveness, short processing time, memory utilization.

Software product Types :

- Generic Products | SWE farm/ developer has absolute authority over the product
  Market analysis করে requirement/feature determine করা হয়,
- Customized Products | Developer and customer has shared authority.
  " " " negotiate করে
  feature select করে,

General issues that affect most software

- Heterogeneity
- Business and Social change
- security and Trust

Issues of Professional responsibility

- Confidentiality
- Competence
- Intellectual property rights | client এর
- Computer misuse.

Software Process Models

High level abstract description of software development process.

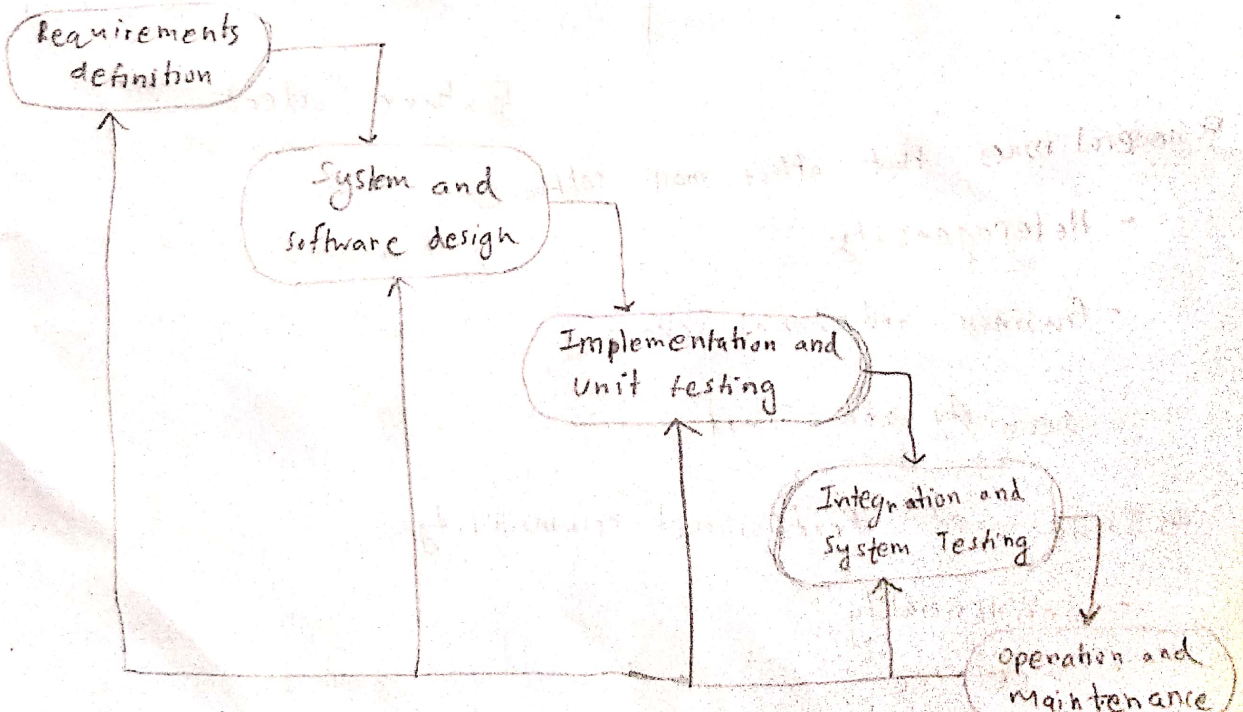
1) Waterfall Model RDIVM

Requirement → Design → Implementation → verification → Maintenance

pros :
- Process visibility
- task গুলো seperated
- Easy to manage

cons :
- One directional development process.
  previous stage revise করার কোনো way নাই,
  → can't cope with requirement change.

redesigned :



→ এবং previous stage শেষ হওয়ার আগে একটা stage শুরু হতে পারবে না, No parallel development
→ Intermediate stage গুলো থেকে আগের stage এ যাওয়া যায় না, Too formal
⇒ Modern software development এ stage গুলো overlap করে,

* safety/security critical software বুঝাতে Incremental development model (plane এর software একটাই final version হবে এবং এখানে কোনো bug থাকতে পারবে না) যুক্তিসম্মত নয়, এক্ষেত্রে Waterfall model use হয়,

* বর্তমানকালের সব software ই incremental development এর কোনো না কোনো version use করে develop করা হয়, (Agile method)

ii) Incremental development

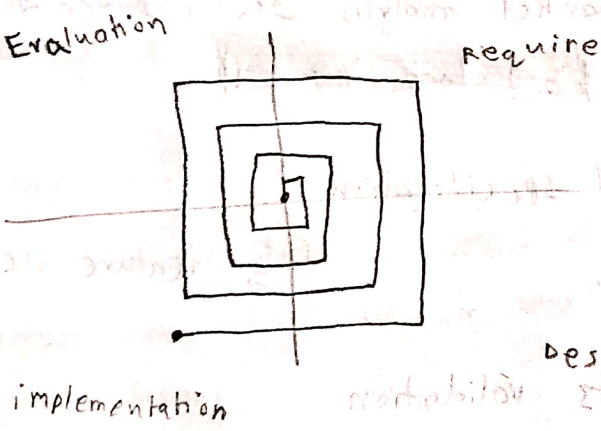
→ iterative process

Client থেকে requirement নেওয়ার পরে প্রথমে একটা ~~initial~~ protype বা initial version দিবে,

তারপর customer থেকে feedback নিয়ে feature add করে একটা Intermediate version দেখাবে,

এরকম বেশ কয়েকটা intermediate version দেখায়ে feedback নিয়ে, feature add করে, revise করে একটা final version দেখাবে

→



→ SUST online exam => Prototype, attendance feature, video call feature, final version.

Pros:
→ যে ধরনের software এ development এর সময় requirement বারবার change হয়, সেক্ষেত্রে ভালো suit করে
→ কম feature দিয়ে হলেও early deployment করা যায়
→ বারবার feedback নেয়ায় customer involved feel করে

Cons:
→ Process খুবো visible না,
→ Each version এর জন্য documentation করা cost effective না
→ Regular changes leads to messy code.

iii) Integration and Configuration.

(1) Software specification

→ software টাকে customer/user এর viewpoint থেকে describe করে,
user software থেকে কী কী feature পাবে,

→ Software specification ঠিকভাবে না করলে পরবর্তীতে system design এবং implementation এ problem হয়,

→ Requirement engineering আলাদা study আছে,

⇒ Requirement engineering :

- Requirement analysis and elicitation :
  - Market analysis এবং customer এর সাথে আলোচনা করি কিছু feature ঠিক করি

- Requirement specification :
  - " " কিছু feature select করি,

- Requirements validation :
  - selected requirement গুলো realistic কিনা check করি, feasibility study করি,

যে feature গুলো select করলাম এগুলো implement করার মতো enough ability আছে কিনা? (experience, resource, budget..)
এগুলো implement করলে customer use করতে পারবে কিনা? worth it কিনা?

② Software design and implementation

→ software কে Developer এর view point থেকে describe করে,

→ 4 activities, that may be part of the design process.

High level
- Architectural design : System এর overall structure, principal components and their relationships.
- Database design
- Interface design

low level
- Component selection and design. frontend এ password দিলে backend এটা DB এর সাথে মিলাচ্ছে check করে,

③ Software validation

- component testing / Unit testing:
  - Each component কে unit হিসেবে ধরে আলাদা আলাদা test করা
  - JUnit for Java

- System testing
  - এক একটা component এ একজন developen develop করে, সবগুলো component integrate করে overall system কে test করাই system testing.

- Customer testing
  - Customer feedback নিয়ে bug fix করা,

(4) Software Evolution

- Maintenance
- Bug fixing
- Add new feature
- Improvement

⇒ Software Evolution life cycle : OMEP

i) Operation

- software টা running phase এ আছে

ii) Maintenance

- Bug fixing

iii) Evolution

- new feature implement করা,
- Technical environment improve করা.

Javascript use করে dynamic করা
reactjs use করে frontend সুন্দর করা

iv) Phase out

- software withdrawn, কারো চাহিদা নাই, কারো কাজে আসতেছে না।

Coping with change: customer expectation meet করার জন্য software evolve করতে হয়,

→ Cost effective approaches to reduce the cost of rework.

- **Change Anticipation:** change টা guess করে নেয়া, future কী change প্রয়োজন হতে পারে,
- **Change Tolerance:** Git, Modular coding করতে হবে যাতে easily change করা যায়,

Prototyping vs Incremental Delivery

Prototyping: Demo version যেটা user কে final version সম্পর্কে একটা overall idea দিবে. helps with Requirement elicitation and Requirement validation.

Incremental Delivery: কিছু high priority feature full functional থাকতে হবে,

যেমন: online exam software & attendance system.

2 type এর software method থাকে।

i) Plan driven method

- Water fall model এই ধরণে separate stage থাকে, stage এর output গুলো আগে থেকে planned থাকে।
- too slow, unnecessary documentation এ অনেক time দিতে হয়।
- too formal
- water fall model

ii) Rapid Software development

- Plan driven method খুব slow হওয়ায় Business and Social change এর সাথে তাল মিলাতে পারে না, সেজন্য আসছে Rapid software development.
- too casual
- Agile method.

=> Plan based method — waterfall model
=> Agile method — iterative process — Rapid software development (Incremental delivery)

Agile method. (Adaptable)

- iterative approach follow করে।
- design এবং documentation এ অতো সময় না দিয়ে code এ focus করে।
- Working software quickly deploy করে।
- Requirement change এর সাথে quickly evolve করতে পারে।

- Plan-driven এ Requirement গুলো আগেই specified থাকে, contract এর মাধ্যমে
  Agile এ Requirement specification dynamic হয়।

Principles of Agile methods

i) Customer involvement

ii) Embrace change

iii) Incremental delivery

iv) Maintain simplicity

v) People, not process : Team এর Each developer কে decision making এ স্বাধীনতা দেয়া হবে।

Cons of Agile approach :

i) Contract না থাকায় পরে সমস্যা হতে পারে, ex. customer - সাথে চুক্তি

ii) change করলে multiple customer এর interest রক্ষা করা difficult

iii) Maintaining simplicity requires extra work.
- যেহেতু documentation এ কম জোর দেয়া হয়, তাই নতুন কোন developer আসলে code বুঝতে পারে না.

iv) Agile method expect করে small, tightly-integrated team থাকবে, তাই large system এ scaling করা difficult.

v) যেহেতু team member কে স্বাধীনতা দেয়া হয় তাই সে responsibility এর জন্য unsuited হলে ঝামেলা।

Inapplicability :
— large system, long projet : due to lack of documentation
— Safety critical system

Plan-driven আর Agile দুই method এই কিছু problem আছে,
Most project এই দুইটারই কিছু কিছু অংশ use করা হয়,

Extreme programming

→ Agile software development framework, Agile এ একটা structure
দেয়ার try করে,
→ Xp কিছু best practise introduce করে

i) User stories

Agile method এ Requirement specified থাকে না,
এর replacement হিসেবে থাকে user stories যেখানে
developer নিজেকে system user হিসেবে চিন্তা করে এবং
তার কী কী feature প্রয়োজন হতে তা client এর সাথে
আলোচনা করে user story আকারে note করে,
যদি customer ↓ feature না পেলে developer কে জানাতে পারে

ii) Refactoring

→ code এ কোনো ধরনের improvement করে code কে cleaner এবং modular করা,
code চিকিৎসকদের কাজ করলেও কোনো ধরনের improvement করা গেলে তা করে ফেলতে হবে,

code smelling: এটা কোনো bug নয় but system design এ deeper problem indicate করে, যেমন: comment, long method.
· comment থাকা মানে method এর নামটা self-explanatory না,
সেজন্য documentation প্রয়োজন হচ্ছে,

→ Refactoring করে code smelling দূর করতে হবে, code duplicacy কমাতে হবে,

iii) XP and change

change Anticipation করা tough,

XP and change বলে, change Anticipate না করে আমরা current code যতটুকু সম্ভব refactor করব clean এবং modular করে রাখি যাতে future change easily implement করা যায়,

iv) Test first development

- Implementation এ যাওয়ার আগে critical test case গুলো চিন্তা করে রাখতে হবে এবং code এ কোনো change করলে সবগুলো testcase আবার প্রথম থেকে run করতে হবে,

v) Pair Programming

দুজন programmer এক PC তে বসে code করবে

pros

→ informal review process হিসাবে কাজ করে যেহেতু দুইজনেই same code দেখে

→ knowledge spread করে, Junior, senior থেকে শিখতে পারে,

cons

- Productivity অর্ধেক হয়ে যায়,

SCRUM

daily meeting করে experience share করে

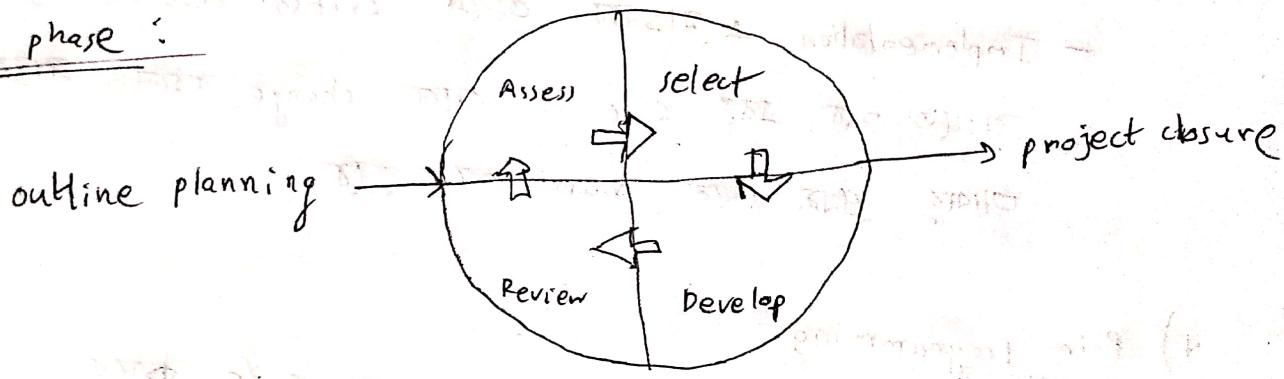
→ knowledge transfer হয়

→ ভালো কাজ করলে encourage করে

→ খারাপ " " guilt Feel হয়.

Team member বেশি হলে Leader না scrum করে।

⇒ 3 phase :



sprint cycle

⇒ Scrum Iterative development এ use হয়।

দুইটা consecutive version release এর মাঝে meeting হয় এটা 1 টা sprint cycle.

Scrum 1 sprint cycle এর একটা timeline থাকে যাতে নির্দিষ্ট সময় পরপর একটা version release হয়।

Domain requirement

কোনো একটা software এর কাজে হাত দেয়ার আগে ঐ software এর Domain related যতোটুকু সম্ভব knowledge gain করতে হবে, তারপর requirement analysis এ যেতে হবে,

Requirement engineering এর ৭ টি step:

Inception Elicitation Elaboration Negotiation Specification Validation

Requirements management → feature manage করা? deadline?

Functional requirement: কোনো একটা particular input বা particular situation এ system কীরকম behave করবে,

Non-Functional requirement

Product, organization, External requirements.

Functional requirement describes what the system should do (dynamic web) and Non-functional requirement :: how the system should do it. (JS)

Software measurement

→ Software কে measure করে project manager decision নেয় কয়জন manpower assign করবে, resource, deadline ...