Date : 10.02.2023

SOLID Principles of OOPs:

S - Single Responsibility Principle

A class should have only 1 single reason to change

Basically,

one class should have only one function such that if we want to change one logic we could change only one class.

This makes the code more verbose, easy to understand and easy to maintain.

O - Open / close Principle

Open to Extension but closed for Modification

So,

Current code/class should not be modified rather make an interface and keep extending that for various use cases.

L - Liskov substitution Principle

If class B is a subtype of class A, then we should be able to replace object of A with B without breaking the behaviour of the program.

Subclass should extend the capability of parent class not narrow it down.

Date :

I - Interface segmented Principle

Interfaces should be such, that client should not implement unecessary functions they do not n

D- Dependency Inversion Principle

Class should depend on interface rather tha concrete class.

Example:

Mouse Interface
- wired
- Bluetooth

Keyboard Interface
- wired
- Bluetooth

In a class instead of having wired Keyboard or wired Mouse we use Interface of Keyboard and use anything inside that.

Advantages of SOLID Principles:

Helps us to write better code
- Avoid Duplicate Code
- Easy to maintain
- Easy to understand
- Flexible software
- Reduce complexity

GUI = Abstraction = Interface
API = Implementation = Platform

Date : 11-02-2023

Design Pattern

→ is a relation ≡ Inheritance
→ has a relation ≡ composition

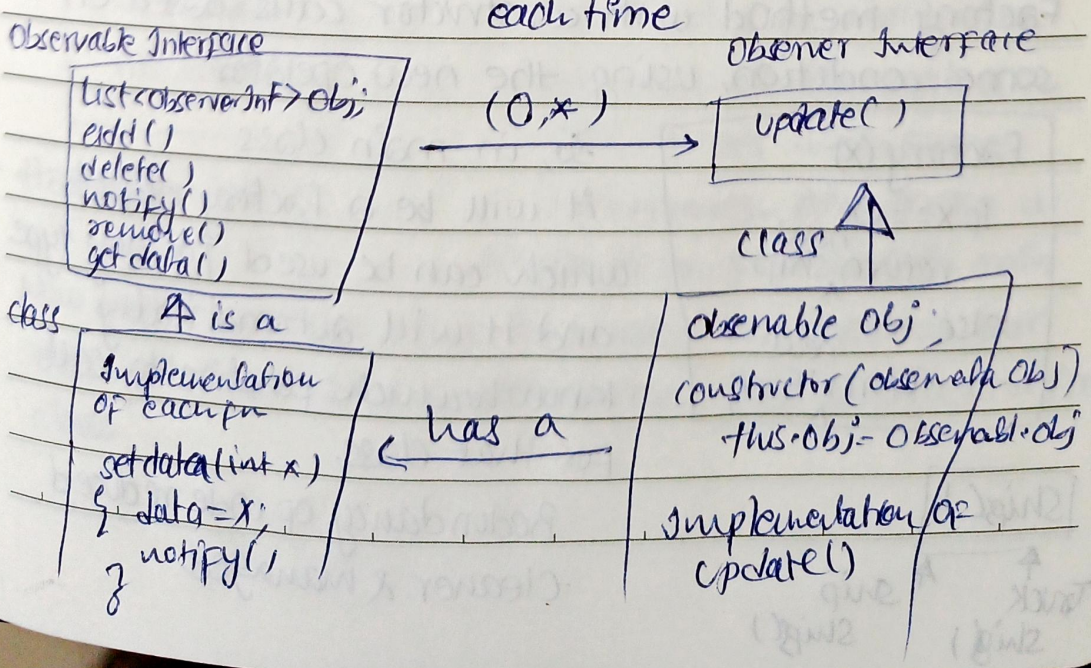
Strategy:

Instead of defining a function which does x and another function which does x+y which involves duplication of code, we make use of interface which has code of x and other class implement that interface as per their need.

Observer Design Pattern:

Observable — # Observer

S1 → S2 → S3 — updates to both each time — O_1 O_2



Date : 12·03·2023

Design Patterns

There are a total of 23 design patterns broadly divided into 3 categories:

1) Creational
2) Structural
3) Behavioral

☆ Creational Patterns:

1. Factory Method:

Create interface in superclass but allow subclasses to alter.

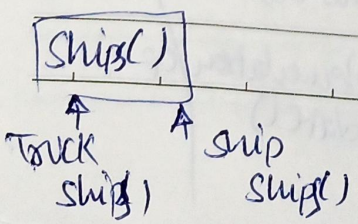
Ex: Take your shipping company of Trucks needs to be extended to Ships.

So, the way to go forward is, create a Factory method with constructor calls based on some condition, using the new operator.

```
Factory(x)
  if x == 0
    return new Ship
  else
    return new Truck
```

So, in main class it will be a Factory object which can be used for any type and it will automatically know which function to call for that class.

Redundancy of code reduced Cleaner & managed.



Date : 13·03·2023

2. Abstract Factory

Factory of Factories

Using the previous example let's say you've expanded to yatchs & buses also. Now you would like to separate land & water shipments. The way to do this would be make two methods like so:

```
DecideFactory(x)
  if (x == Land)
    return new LF
  else
    return new WF
```

```
FactoryInterface
  getVehicle()
```

```
WaterFactory
  return Ship
```

```
LandFactory
  return Truc
```

3. Builder

Step by step (but not Dynamic)

Take this class:

```
Ship()
  int make;
  int model;
  :
```

- has redundancy

You cannot create a dynamic constructor if fields are optional. So ~~cr~~ create a ShipBuilder with same attributes. and make a constructor for that in main

Now in the Builder class just use .set method for each attribute & call the same via a Director/Ma class

Date : 13-03-2023

4. Prototype

Copy existing objects without making your code dependent on their classes.

There is a .clone method in java which can clone an object but it's protected so your class needs to implement cloneable interface && you need to overwrite the clone method via shortcut.

But this will only get you shallow cloning which means if A is changed B is also changed i.e, two references to the same thing.

So, to remove this we need to change the return type of the clone function and write a for loop to copy each element this will make the new object treated as a unique one.

5. Singleton

Ensure a class has only one instance while providing a global access point to this instance.

Process to do:

Make constructor

write a getInstance method (since this method is static obj should be static)

call/create an instance through above method.

Example: A country will have only one Government

Date : 14.03.2023

☆ Structural Patterns:

1. Adapter:

Allows objects with incompatible interfaces to collaborate.

This basically creates a new method/class called the Adapter which takes input x and after some logic returns Y. Hence, through this incompatible X & Y can collaborate.

2. Bridge:

Lets you split a large class or a set of closely related classes into two separate hierarchies - abstraction & implementation - which can be developed independently of each other.

○ + □ Blue + Black

4 combinations & will increase

shape ◇ ——contains—→ color

Just 2 classes

Ultimately, keep the GUI (abstration) different from the API (implementation)

This way you can make changes to GUI without touching API and vice versa.

Date : 14·03·2023

3. Composite

lets you compose objects into tree structures and then work with these structures as if they were individual objects

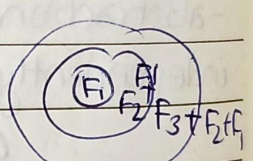
This pattern only makes sense when the ~~th~~core model is that of a tree.

Along with the interface we create another composite class which implements all the sub tree functions recursively until the leaf.

4. Decorator

lets you attach new behaviours to objects by placing these objects inside special wrapper objects that contain the behaviour.

If we don't have Decorator pattern, there would be Class Explosion.



A decorator class is implemented such that it has both has-a & is-a relationship.

5. Facade

Provides a simplified interface to a library, a framework or any other complex set of classes.

simply put hide system complexity from client.

This is same as process/exp layer between UI & sys layer. Something to note is you can use Facade of Facade

Date : 14.03.2023

6. Flyweight (also known as cache) # This is for redundant objects

lets you fit more objects into the available amount of RAM by sharing common parts of state between multiple objects instead of keeping all of the data in each object.

Properties which do not change are intrinsic and which change are extrinsic. Idea is to use extrinsic properties outside the ~~class~~ constructor in factory method as cache. & intrinsic properties do not have getter, setter functions.

7. Proxy

lets you provide a substitute or placeholder for another object. A proxy controls access to the original object, allowing you to perform something either before or after the request gets through to the original object.

This is similar to facade just that a proxy is only for specific class & has same response type as the main class. It cannot be made for more than 1 class. Although proxy of proxy is possible.

Date : 16.03.2023

☆ Behavioral Patterns:

1. Chain of Responsibility

Lets you pass requests along a chain of handlers. Upon receiving a request each handler decides to either process the request or to pass it to the next handler in chain.

We impement this in an abstract class with a protected parameter to call the next responsibility with conditions & add required chaggelic

2. Command

Copy and ctrl+C will have same logic but if requires different class then it's code redundancy. So, we make a command interface with invoker & receiver to be called as many times as we want.

3. Iterator

Lets you traverse elements of a collection without exposing its underlying representation (list/tree)

Already implemented in collection framework of Java. Basic idea is to have a common interface that can traverse any object & each object implements that interface

Date :

4. Mediator

lets you reduce chaotic dependencies between objects. The pattern restricts direct communications between the objects & forces them to ~~use~~ collaborate only via a mediator object. (Traffic light)

Add a mediator interface & its implementation User class will have a protected mediator object & functions will be called using that. Now its the job of the mediator to implement ~~to~~ logic to lets say handle ~~t~~raffic.

5. Memento

lets you save and restore the previous state of an object without revealing the details of its implementation. (undo operation)

Let the object itself do the task of making snapshot as it would have access to private members also and store ~~it~~ memento interface A caretaker is made who just has access to metadata. And undo is processed this way.

6. Observer ~~Pat~~ & 7. Strategy

Turn back 4 pages.

Date : 17.03.2023

Behavioral Patterns

8. State

lets an object alter its behaviour when its interval state changes. It appears as if the object changed its class.

For managing multiple states using if & else maybe cumbersome. So we make multiple state classes as per need implementing the same interface and asks the main function to replace active state

9. Template Method

Defines the skeleton of an algorithm in the superclass but lets subclasses override specific steps of the algorithm without changing its structure

10. Visitor (Double dispatch) - Two functions, accept & call so it checks both params & functions

lets you separate algorithms from the objects on which they operate.

Say Restaurant, Shop & Laundry want to send notifications. Instead of having multiple notify methods in each class, we have a Notification Interface which has all the methods & is implemented. All concrete classes just have to add a method to call the notify method.