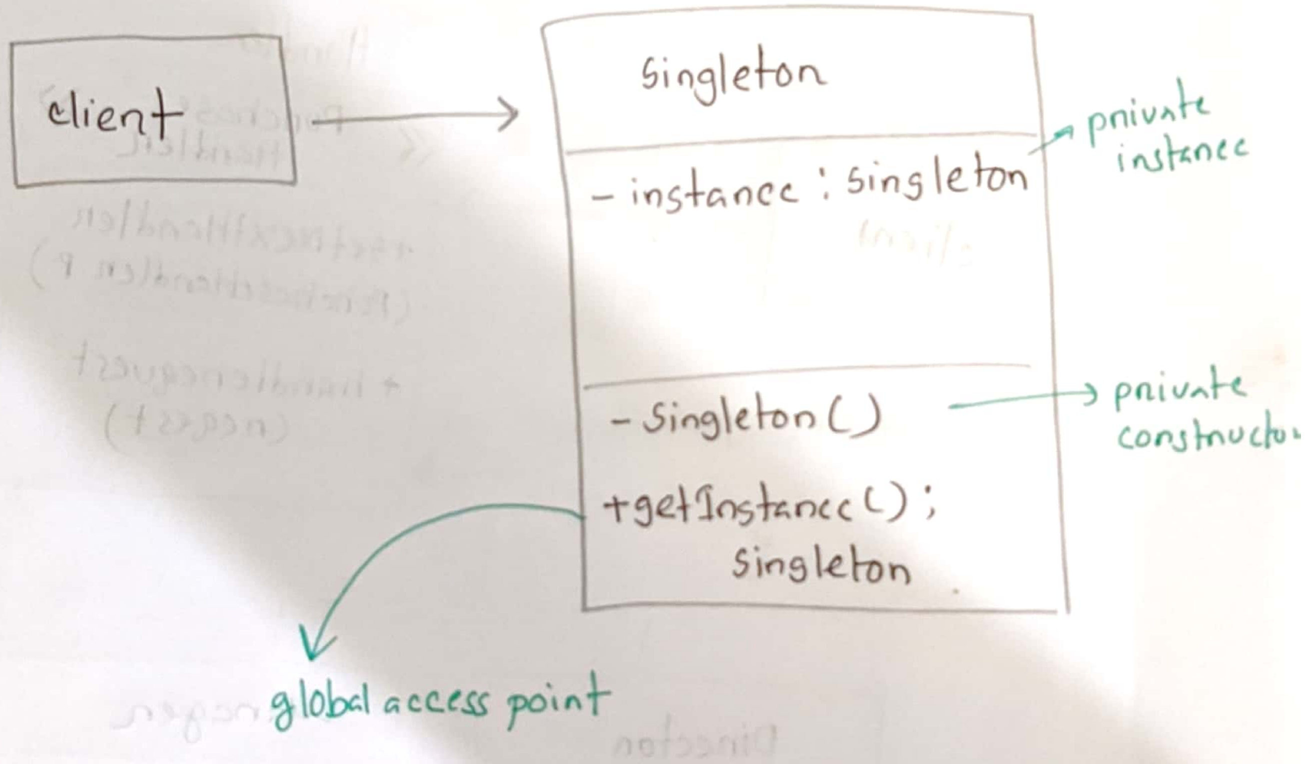
Singleton Design pattenn

Intent :- a class has only one instance and pnoviding a global access point to that instance

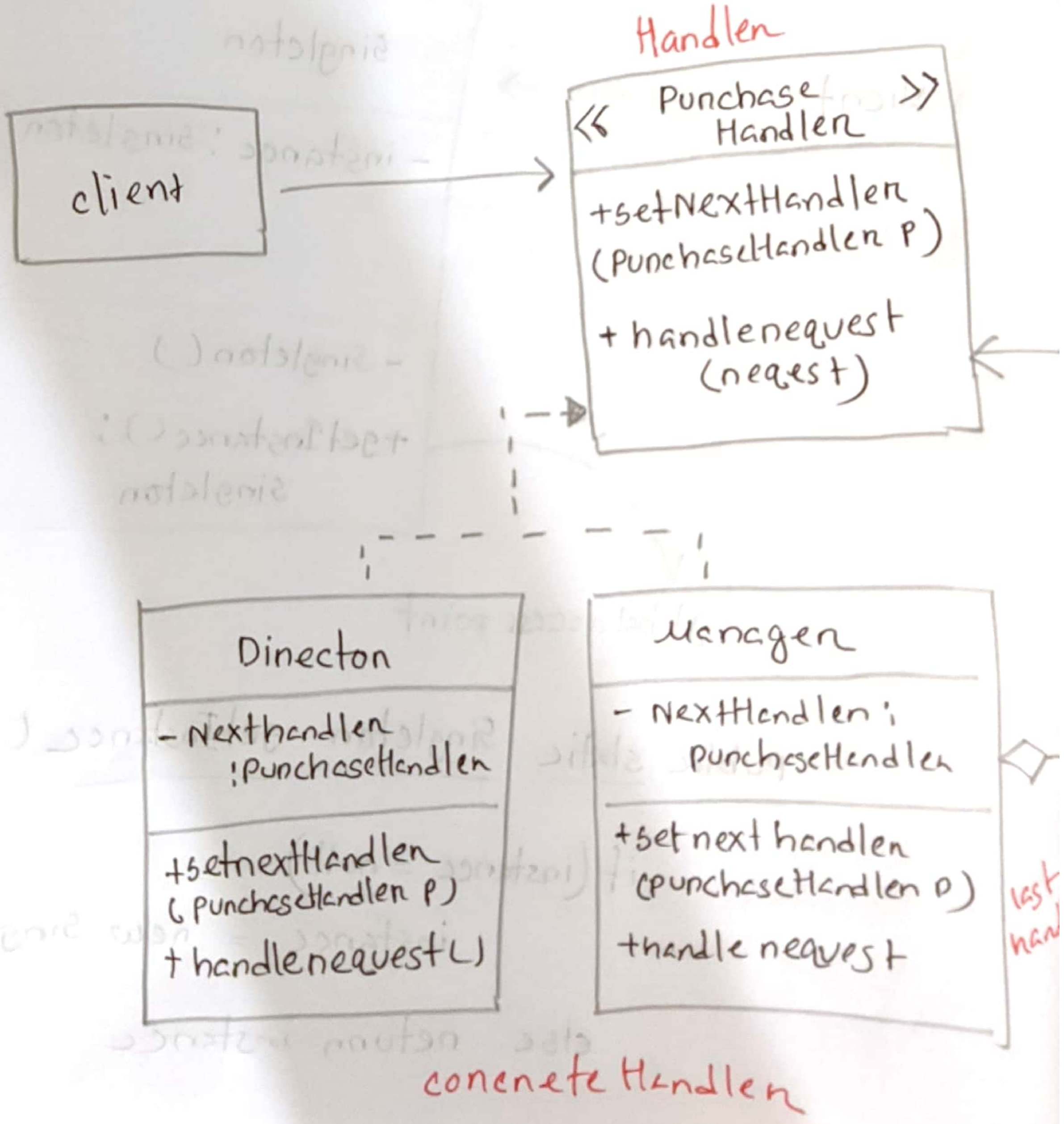


```
public static Singleton getInstance ( ) {
    if (instance = null)
        instance = new Singleton ( )

    else netunn instance
}
```

Chain of Responsibility

→ allow objects to pass nequest to a chain of potential handlens untill the nequest is handle on necched end of chain

→ each handlen how two option
① Handle the nequest
② pass it to next handlen



Handlen
<< Punchase Handlen >>
+setNextHandlen (PunchaseHandlen P)
+ handlenequest (neqest)
client
Dinecton
- Nexthandlen :PunchaseHandlen
+setnextHandlen (PunchaseHandlen P)
+ handlenequest()
Managen
- NextHandlen; PunchaseHandlen
+set next handlen (PunchaseHandlen P)
+handle nequest
last han
conenete Handlen

```
class Dinector implements punchaseHandler {
private Punchase Handlen handle ;

void setnextHandlen ( PunchaseHandlen handle )
        { this.handle = handle }

void handle_reqest ( nequest) {

        if (nequest)
                        { " got handled" }   handled it

        else  { "pass it next "
  pass it               handle. handleRequest(neqest) ;
              }
                                        }
```

— o —

```
#
        Punchase_Handlen   managen  =  new Managen
        punchaseHandlen    Dinecton  =  new Dinecton

        mangaen. setnexthandlen (Dinecton) ;
         managen. handleRegeuest(   ) ;
```

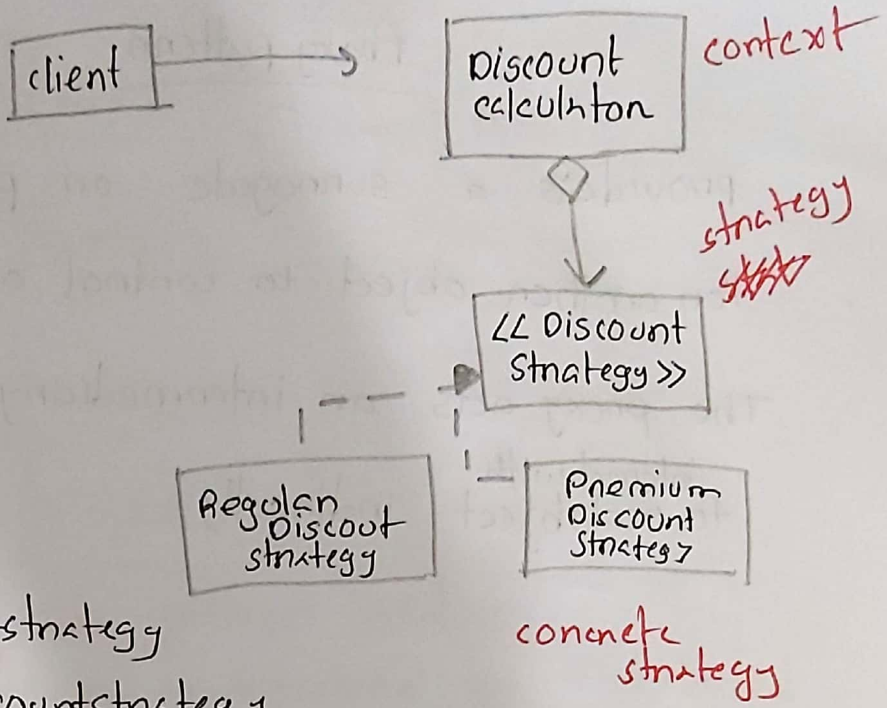
Pattenns that eliminate if else

① Chain of Responsibility :-

```
        private Handlen handle;

    void set.nextHandlen ( Handlen handle)
          { this.handle.= handle }

   void handleRequest ()
         { # handle
           # pass
           netunn handle. handleRequest()

         }
```

(nest follow last example)

② $\boxed{\text{strategy}}$

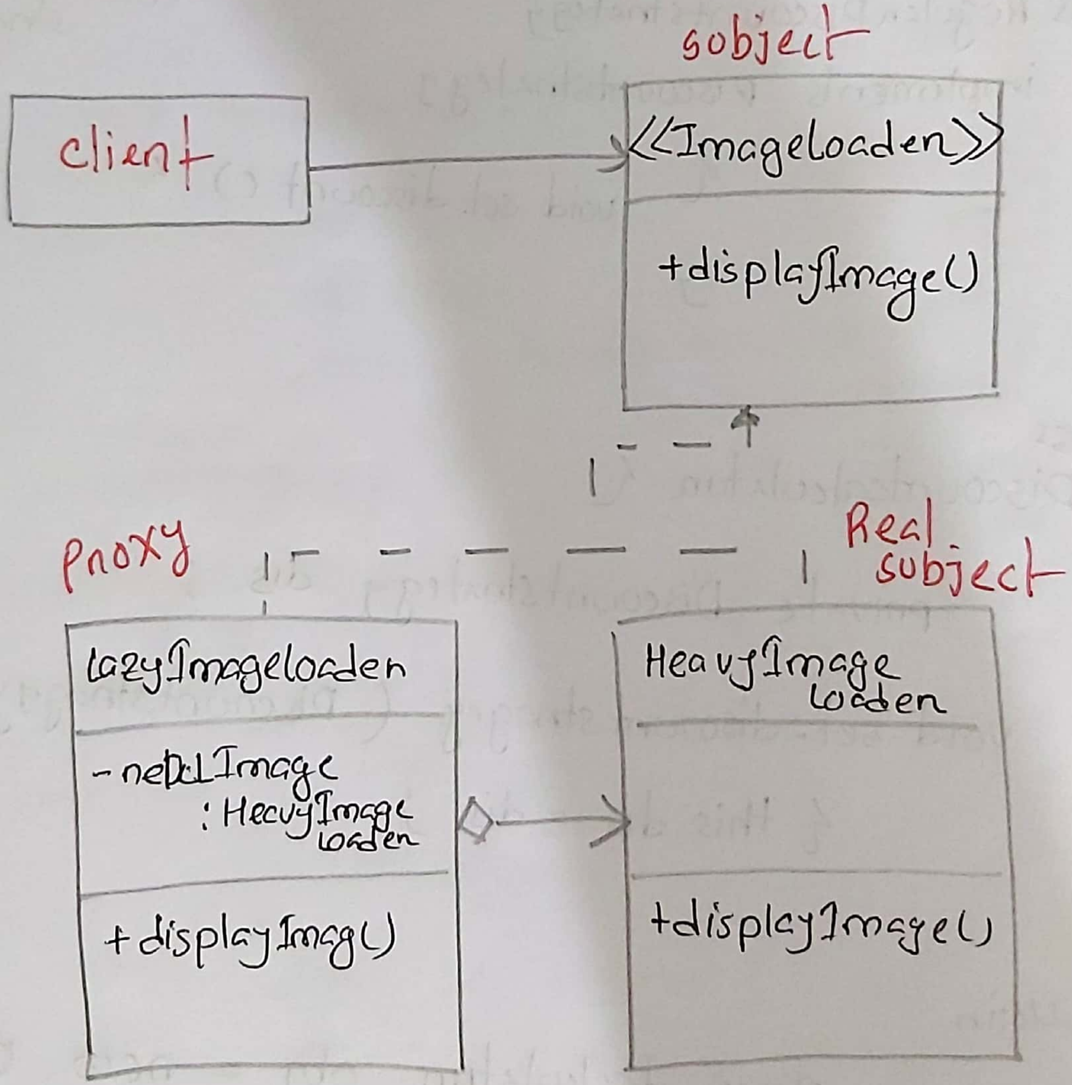


```
# intenface
    Discountstnategy {

  void setdiscount()

          }

# class RegulanDiscountstnategy
        implements Discountstnategy
                { void set-discount () { ... }
                }

    class
#  Discountcalculaton {

        pnivate Discountstnategy dis ;

     void set-discountstnagey ( Discountstnaggy dis)
           { this.dis = dis }

# Main

        Discountcalculaton obj = new Discount...

        obj.set-Discountstnategy(new RegulanDiscount
                                     stnategy() );
```

Proxy pattern

provides a surrogate or place holder for another object to control access of it.

The proxy acts an intermediary allowing client to interact with realobject indirectly



```
void display() {
      if(realImage=NULL)
           real Image=new HeavyImageLoader();
   ..realImage.displayImage          }
```

$$\begin{tikzcd} & \text{Proxy can be similar} \arrow[dl] \arrow[d] \arrow[r] & \text{Facade} \\ \text{Adapter} & \text{Decorator} & \end{tikzcd}$$

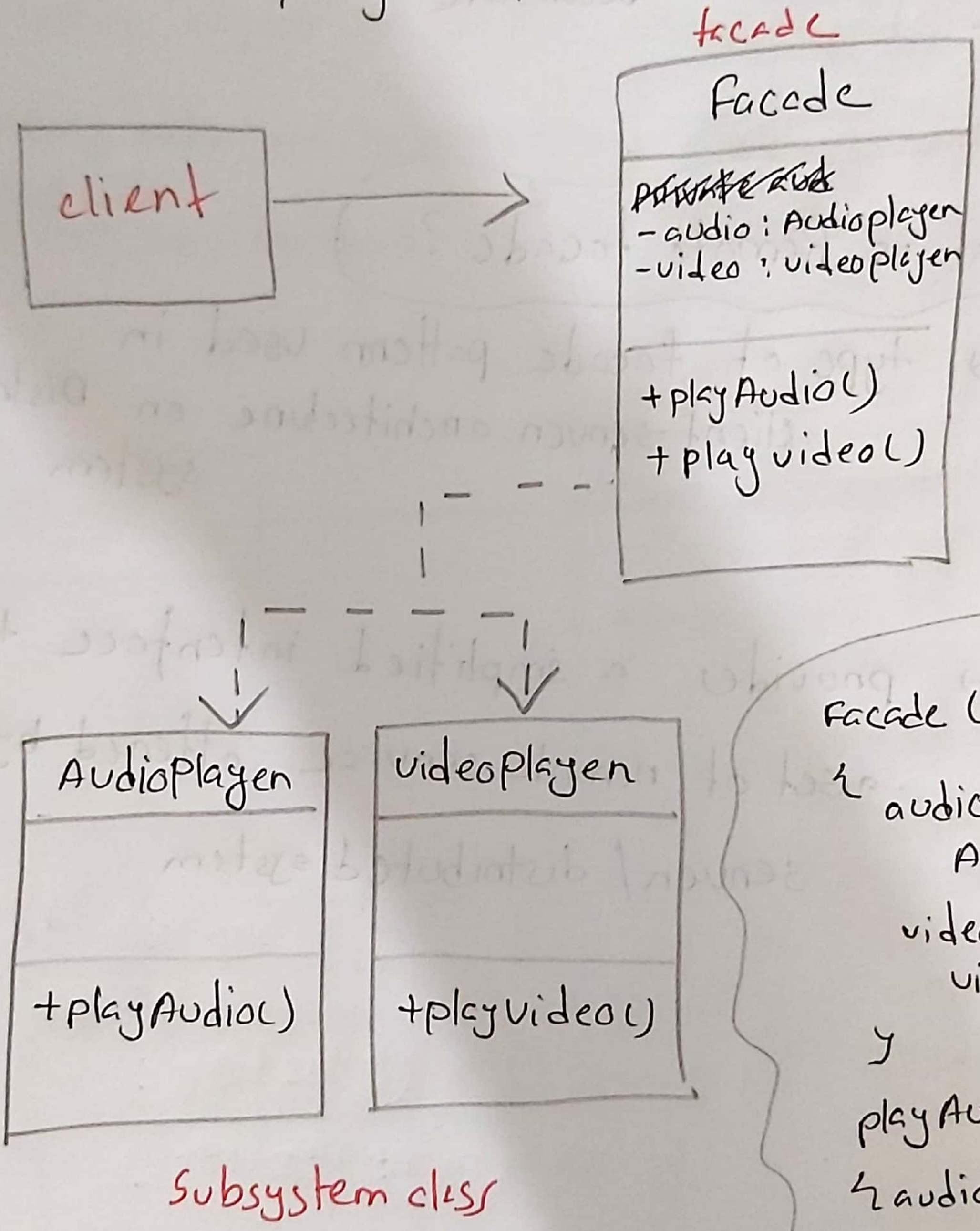
(*) what is Remote facade ?

→ type of facade pattern used in client-server architecture or Distributed system

→ provides a simplified interface to a set of remote service offered by a server/ distributed system

facade pattern

provides a simplified interface to a library / a framework / any complex set of classes.



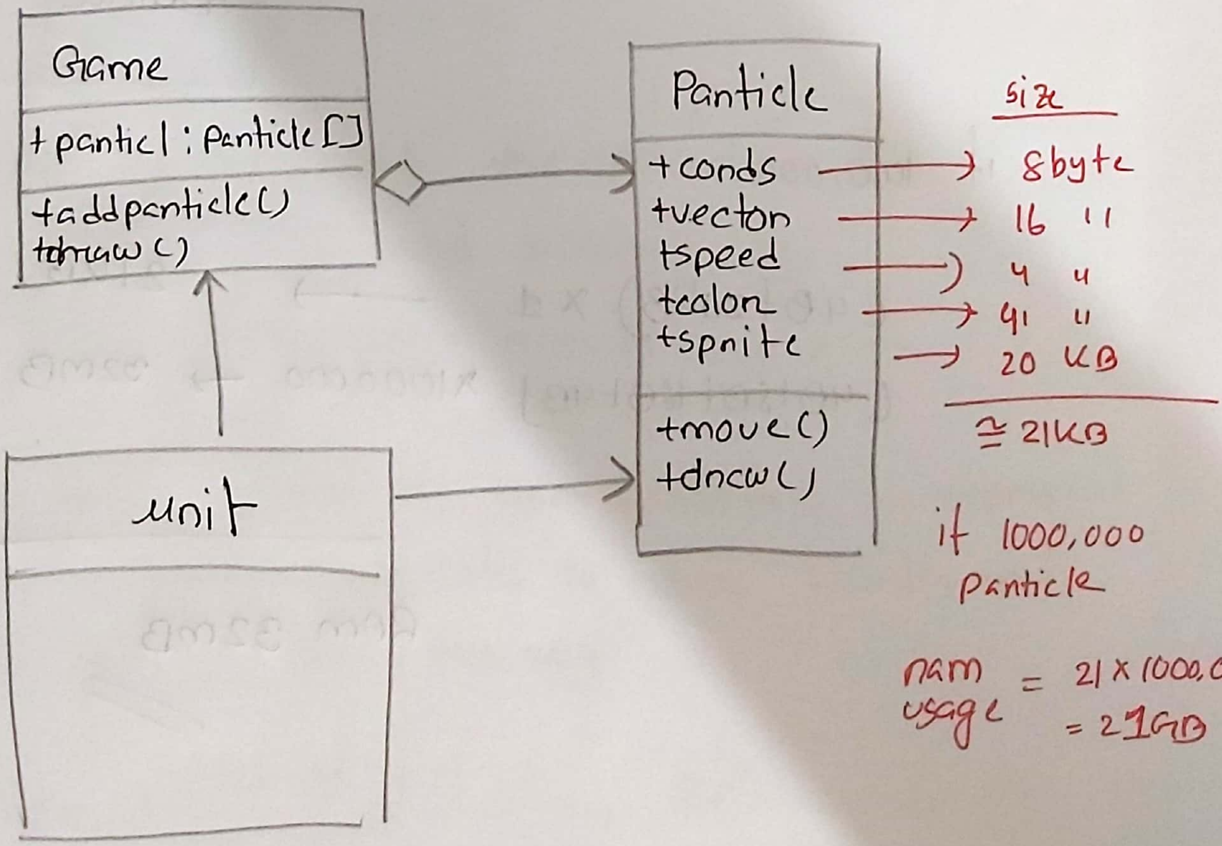
```
Facade ()
{  audio = new
        Audioplaye
   video = new
        videoplay
}
playAudio()
{audio.play
        Audio()
}
```

Flyweight

→ Reduce memory Requirement

is a structural design pattern that lets you fit more objects into the available amount of RAM by sharing common parts of state between multiple object instead of keeping all of the data in each object

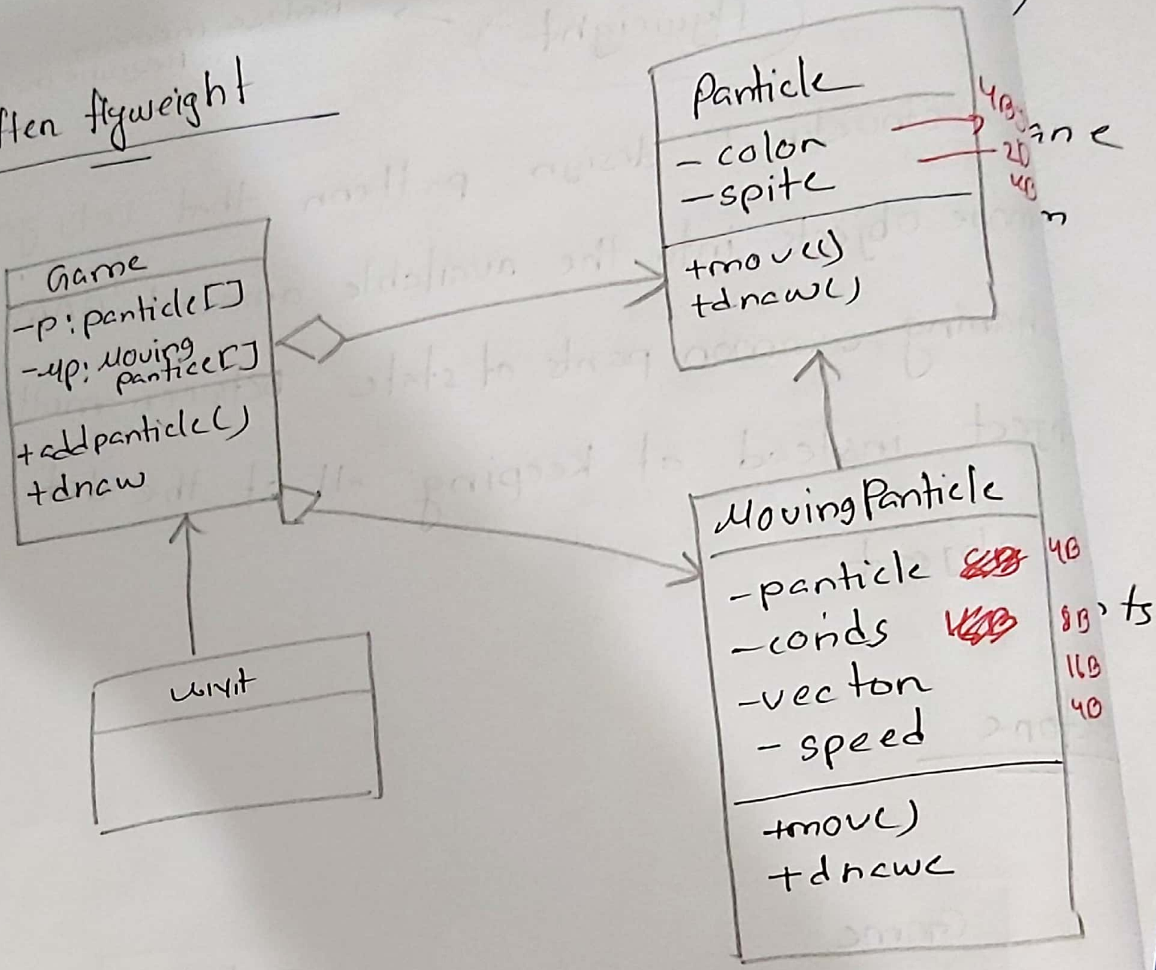
Before



if 1000,000 particle

ram usage = 21 x 1000,000 = 21GB

After flyweight



if 1000,000

$(4B+20kB) \times 1 \longrightarrow 21kB$

$(4B+8B+16B+4B) \times 1000000 \longrightarrow 32MB$

RAM 32MB

3 principle Layer

1. Presentation Layer
2. Application Layer (Business Logic Layer)
3. Data Layer

Service Layer

is a component of software architecture that act as an intermediary between presentation and data layer

—— o —

model → represents the application data and buisness Logic
managing - maintaining data

view → represent User Interface . represent the ~~user~~ data to user visually and Interactive way

Controller → intermediary between model-view
it recives user input from view and process it