Static func. can be called before obj formation.

Singleton

A class with a single obj.

class SO{ ..... }

bool checkObj(){

```
class SO {
  static bool flag;
  SO() { if(flag==
         flag = 1
};
SO:: static flag = 0
```

```
class Main {
  private:
    Main() { }
    static Main * obj;
  public:
    static Main * getIns() {
        if (obj == null)
          obj = new Main();
        return obj; }
};  static Main* Main::obj = null;  (Initialize out of class)
void foo(){ Main* p1 = Main::getIns();
void bar(){ Main* p2 = Main::getIns();
```

State Design Offline

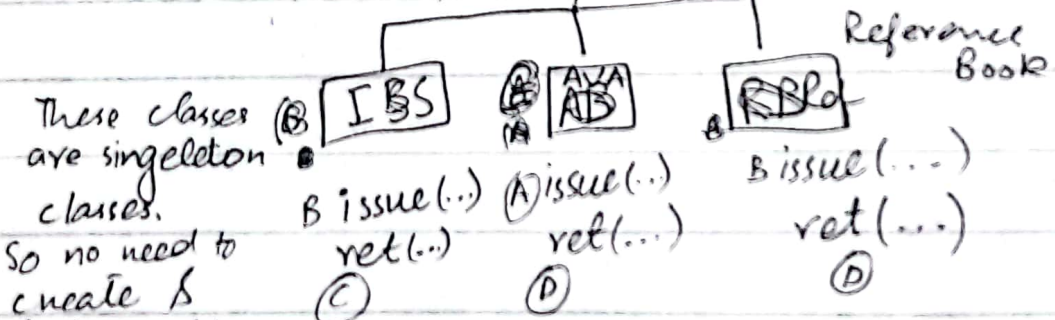
```
class Book{
  int state;
  bool issue (Mem * m)
  { if (state == AVA){
      -------
(A)     state = ISS;
        return True;
    }
(B) else return false;
  }
```

```
bool ret(){
  if(state == ISS)
  {  ----
     state = AVA
(C)  return true;
  }
  else
(D) return false;
}
```

Multiple books obj — 1 state obj

Abstract class

issue(...) = 0
ret() = 0



These classes are singeleton classes. So no need to create & delete obj.

```
int main(){
  Book bl(----);
  bl. setState(new AVa());
  bl. setState(new Iss());};
```

Refactor:

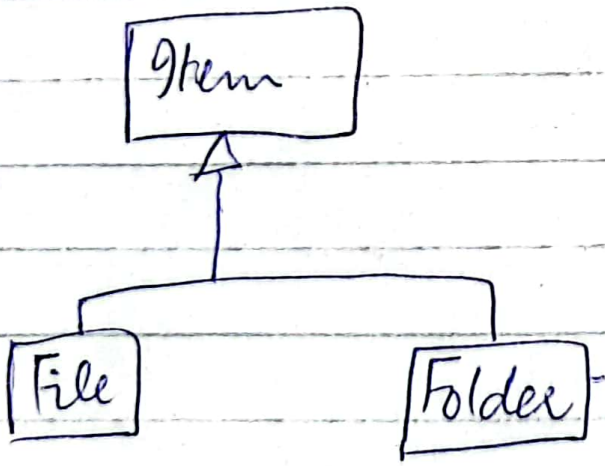
```
struct File{
    char * name;
    int size;
}
int comsiz(File *f){
    return f->size; }

struct Folder{
    char * name;
    File * a[m];
    Folder * b[n];
}
int comsize(Folder *f){
    int sum=0;
    for(i=1; i<m; i++){
        sum+=comsiz(f->a[i]);
    for(j=1; j<N; j++){
        sum+=comsiz(f->b[j]);
    return sum;
}
```

Login

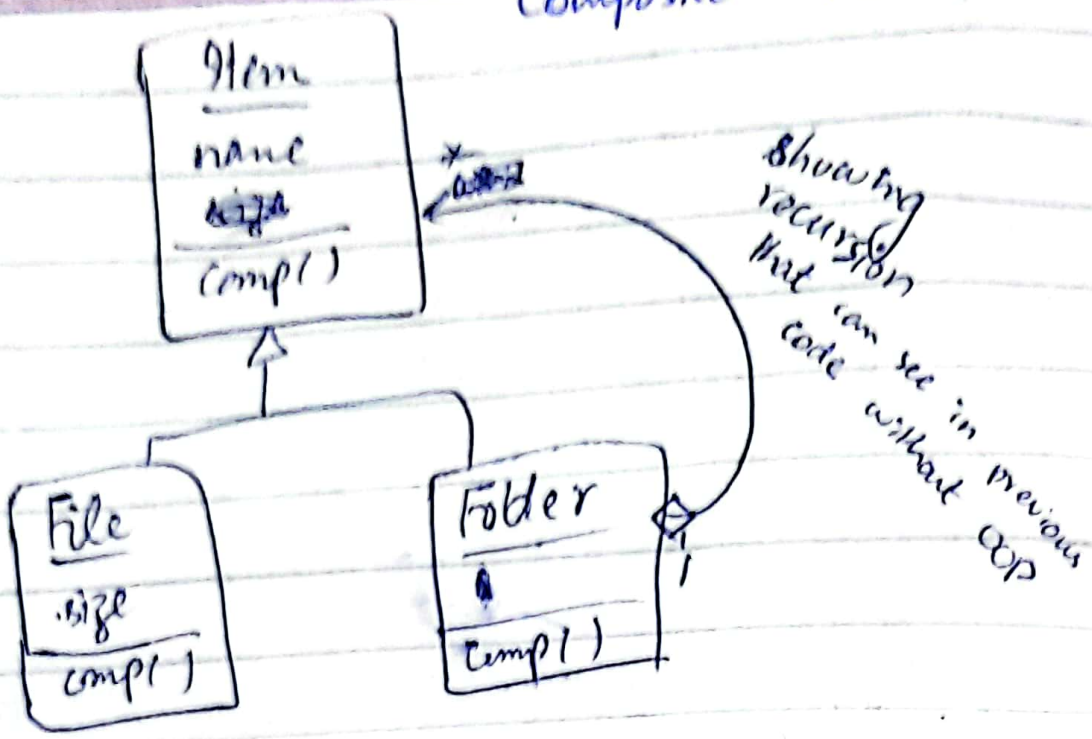
S → b
B → b

OOP used:

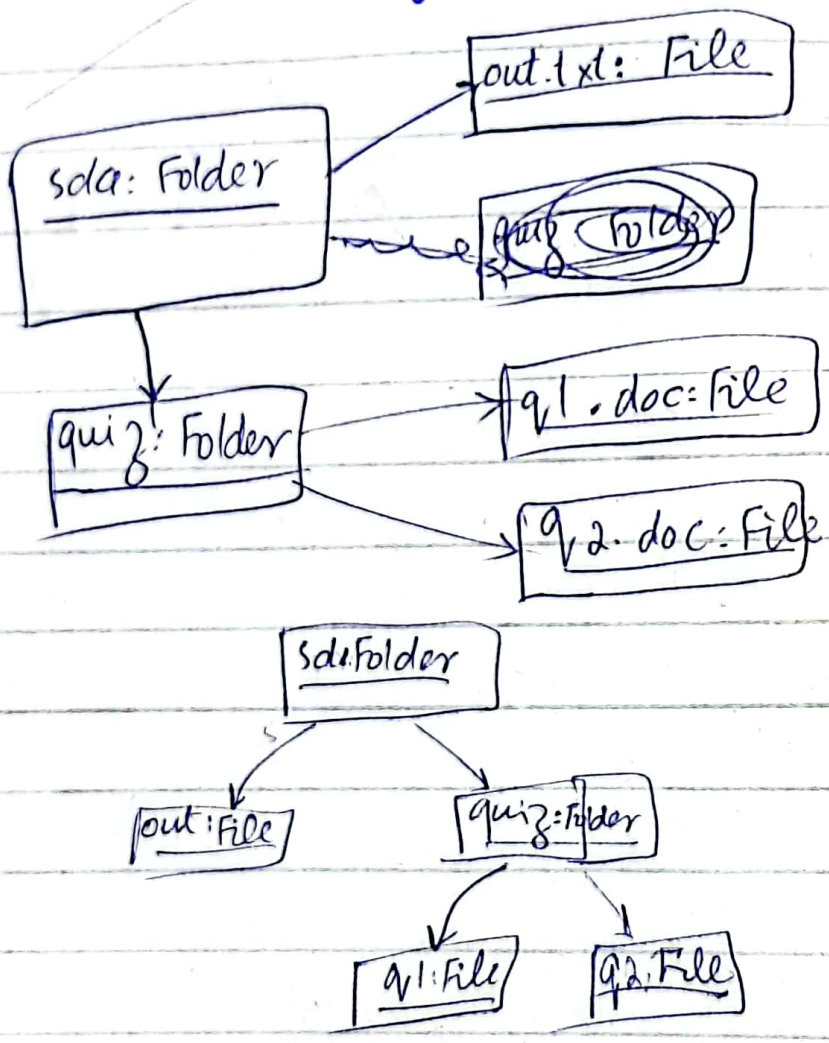
```
class Item{
    char * name;
    virtual int comsize() = 0;
};
class File: public Item{
    int size;
    int comsize(){ return size;}
};
class Folder: public Item{
    Item * a[N];
    int comsize(){
        int sum = 0;
        for(i=0; i<N; i++){
        sum += a[i] -> compSize();
    return sum; }
```



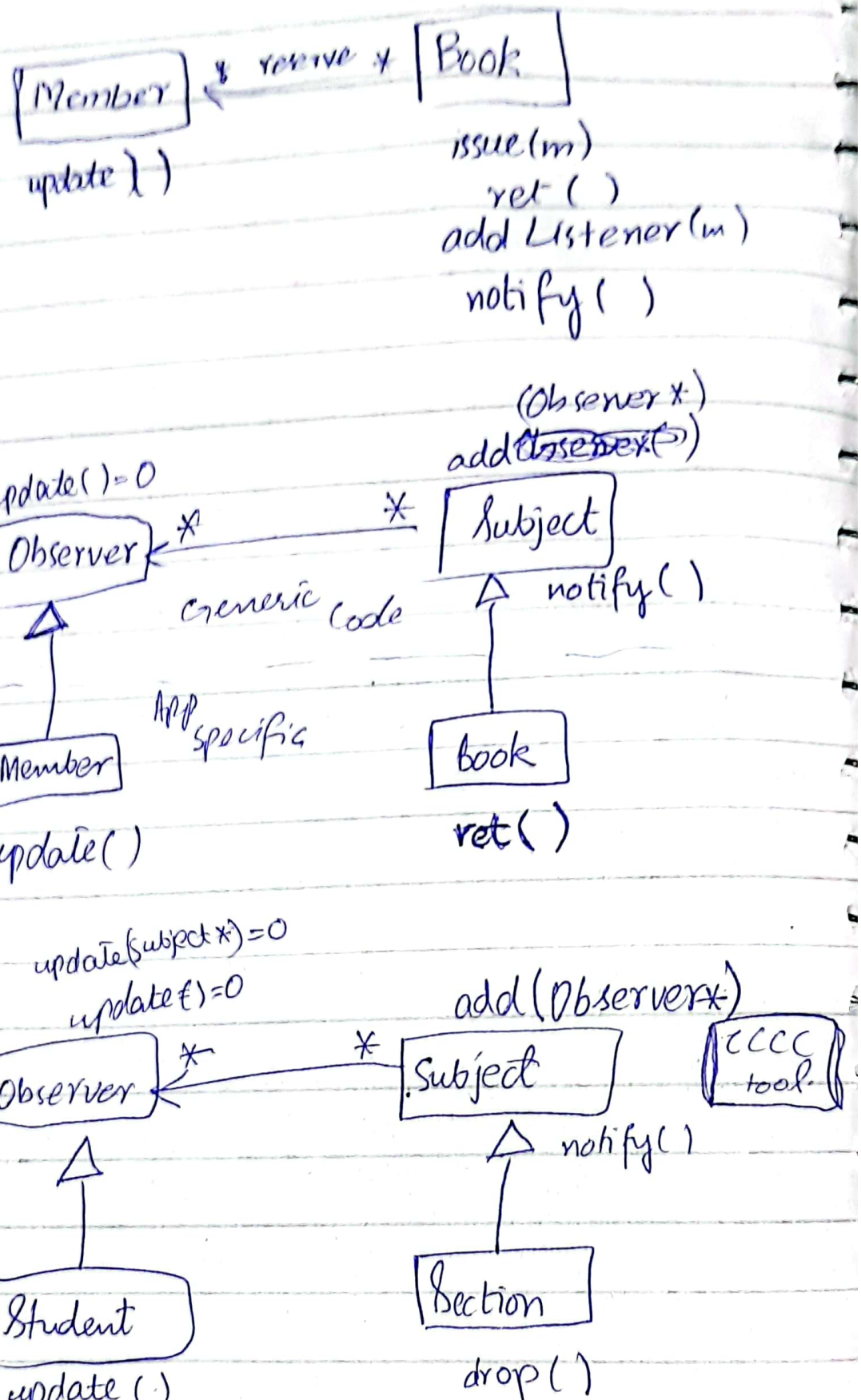
Composite Pattern

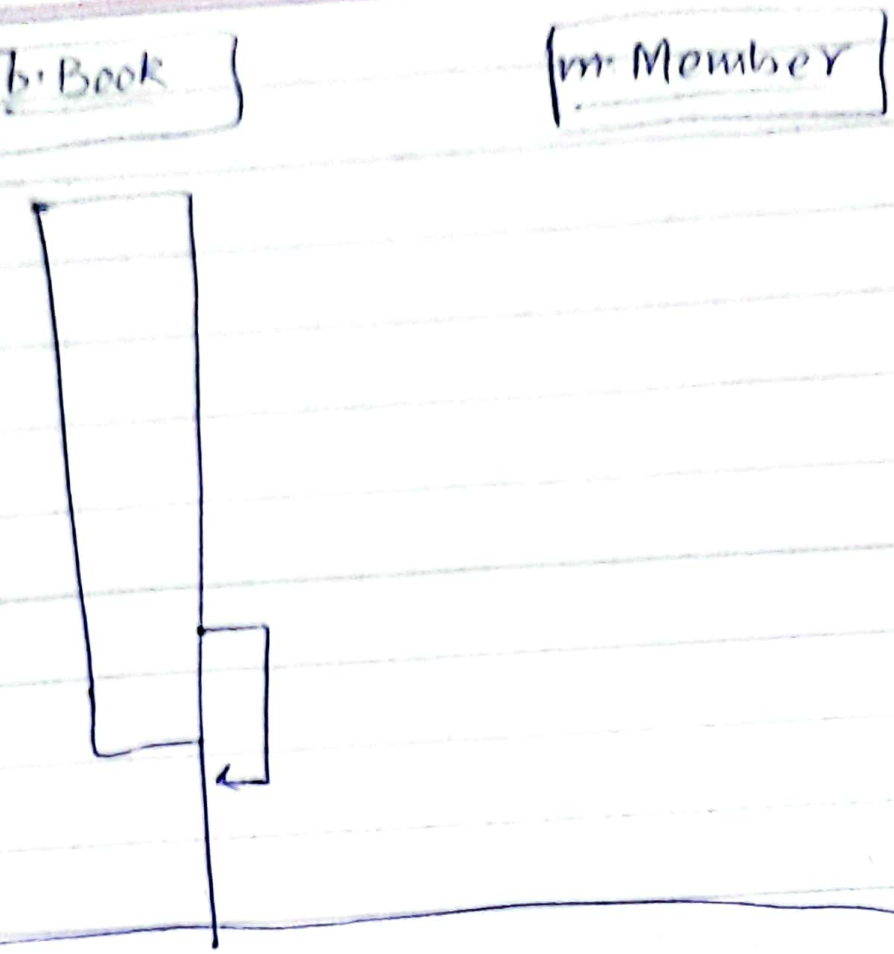


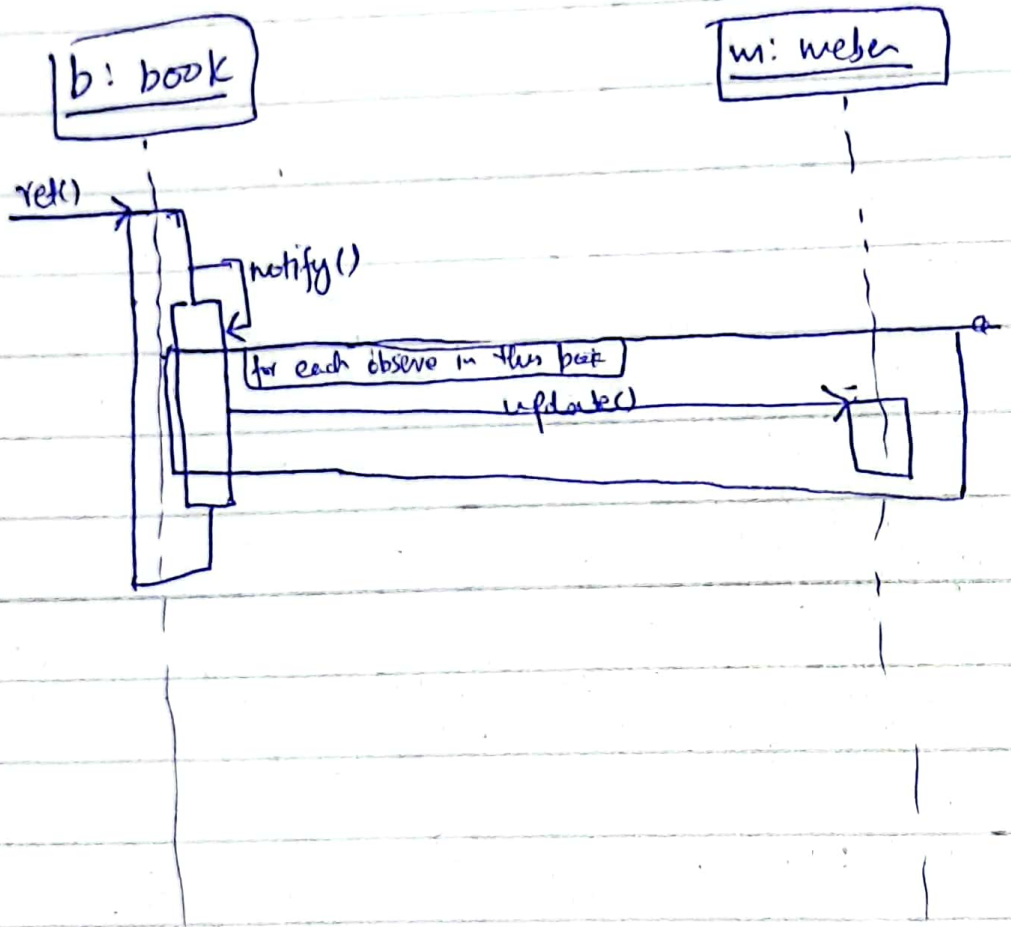
Object Diagram



Observer Design Pattern



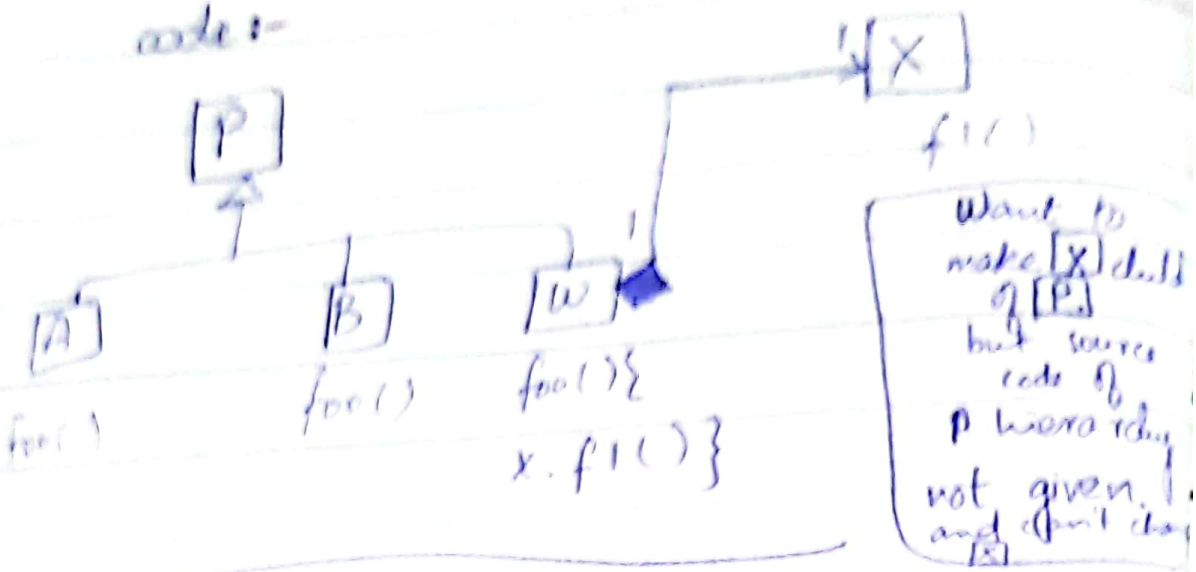

b: Book
m: Member

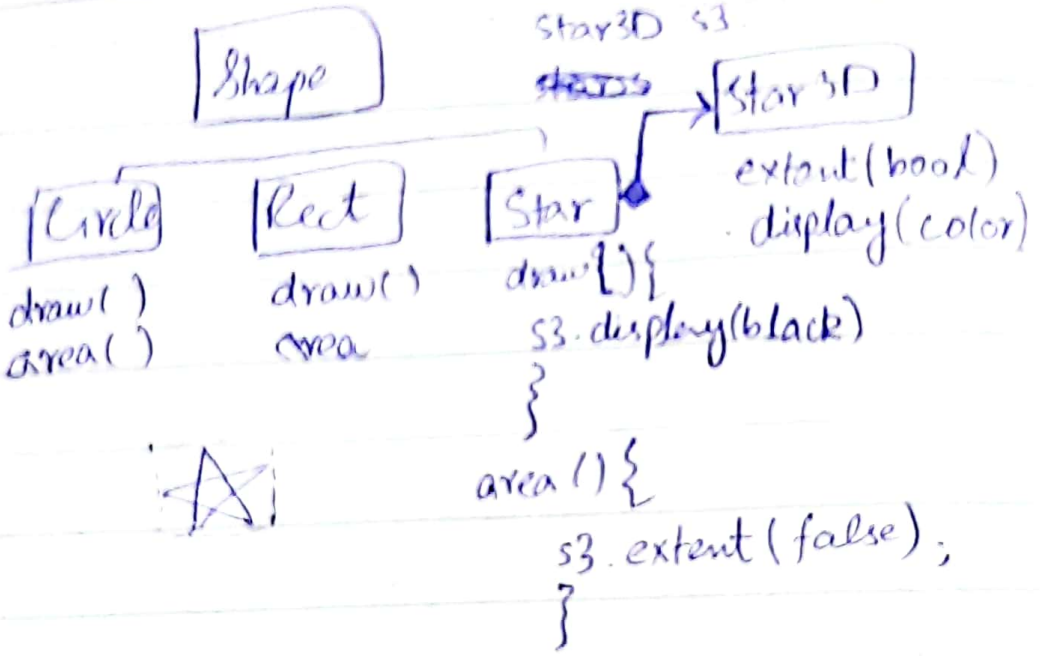

b: book
m: meber
ret()
notify()
for each observe in this book
update()

Adapter pattern / wrapper pattern

How to plug a class into an inheritance hierarchy without changing?

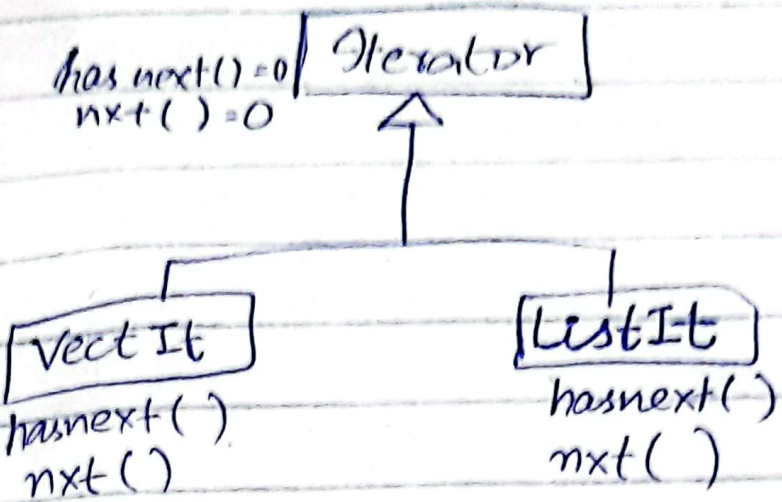
code :-





Vector [ ]

friend class Iterator{ };
int data;



class VectIt{

nxt(){

class ListIt{

nxt(){
this → data;
}
hasnext(){

class Vector{
int * a[N];
int max;
int no;
friend VectInt;

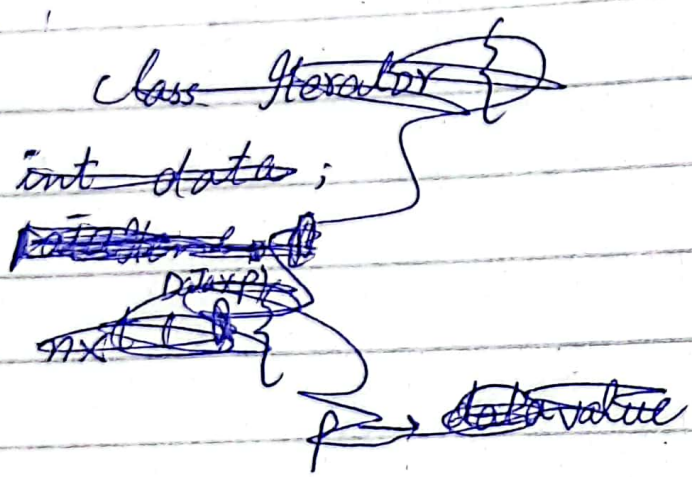
28/11 SDA

```
int sum(Vector * vec)
{  int s=0;
   for (int i=0; i<vec->no; ++i){
      s = s+(vec->a[i]);
   return s; }
```

For every data structure we should have to write separate sum func.

} Write code sum that works for each data structure

```
int sum( Iterator* it){
   int s=0;
   while (it->hasNext()){
      s = s+it->nxt();
   return s; }
```



```
class VectIt : public Iterator {
    Vector * vec;
    int i;
VectIt ( Vector * v ) {
        vec = v;
        i=0; }
bool hasNext(){
        return ( i< vec->no);
        }
int {nxt(){
        vec->a[i++];
        }
};
List{
    Node * head;
    friend ListIt;
```