Agenda

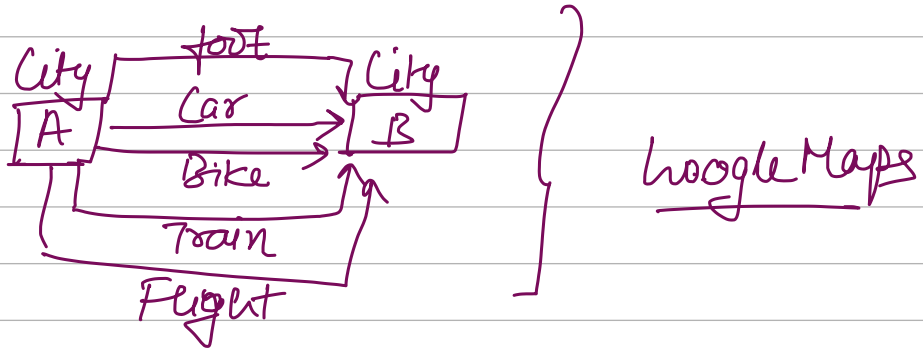
Start @ 9:05 PM

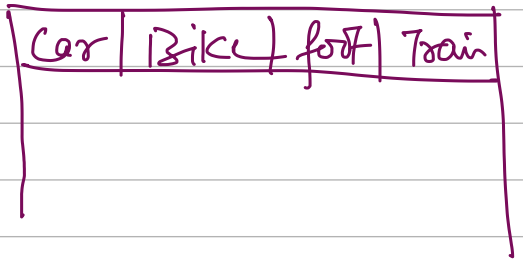
1) Behavioral design patterns
   a) Strategy design Pattern
   b) Observer design pattern

Behavioural Design Patterns

↳ Based on actions / methods / function / Behaviors

Strategy Design Pattern





When we search for a path from point A to point B in google maps, there are multiple paths possible based on the modes of transport.

Google Map

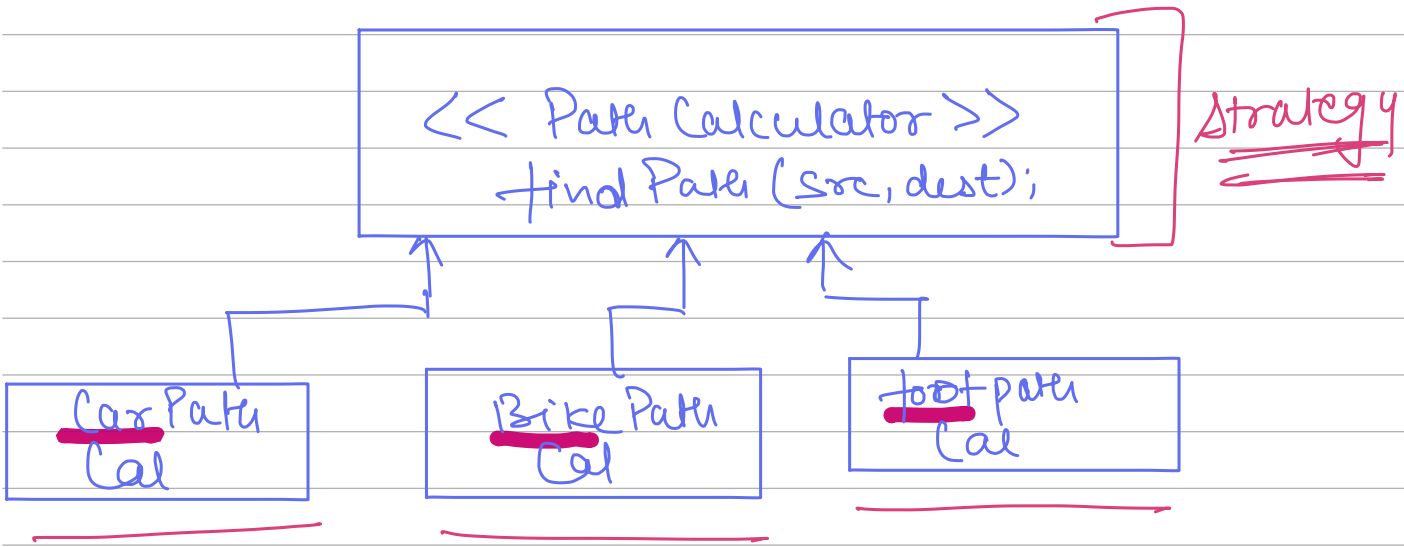
```
findPath(A, B, mode)
{ if (mode == Car)
  {
  }
  else if(mode == Bike)
  {
  }
  else if(mode = walk)
  {
  }
}
```

Voilating
OCP
SRP

⇒ When there are multiple ways (Algo) of doing something often we see the voilation of SRP & OCP because of multiple if-else condition

⇒ Every way of finding path is independent of each other

⇒ Rather than implementing these ways in one method, implement them in separate classes.



Google Maps

```
findPath(src, dest, mode)

PC = PCF.getPathCalForMode(mode);

PC.findPath();
```

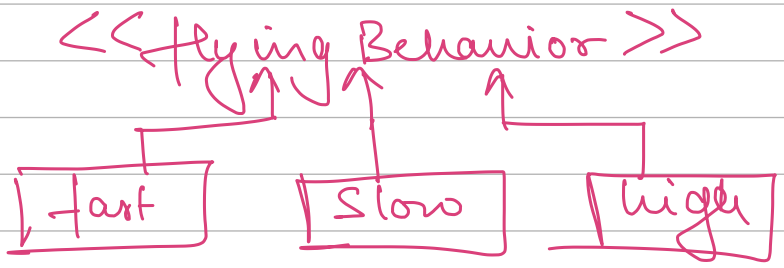
PathCalculatorFactory

```
static getPathCalForMode(String Mode)
{ if (mode == bike)
      return new bikePC()
  if (mode == car)
      return new carPC()
```

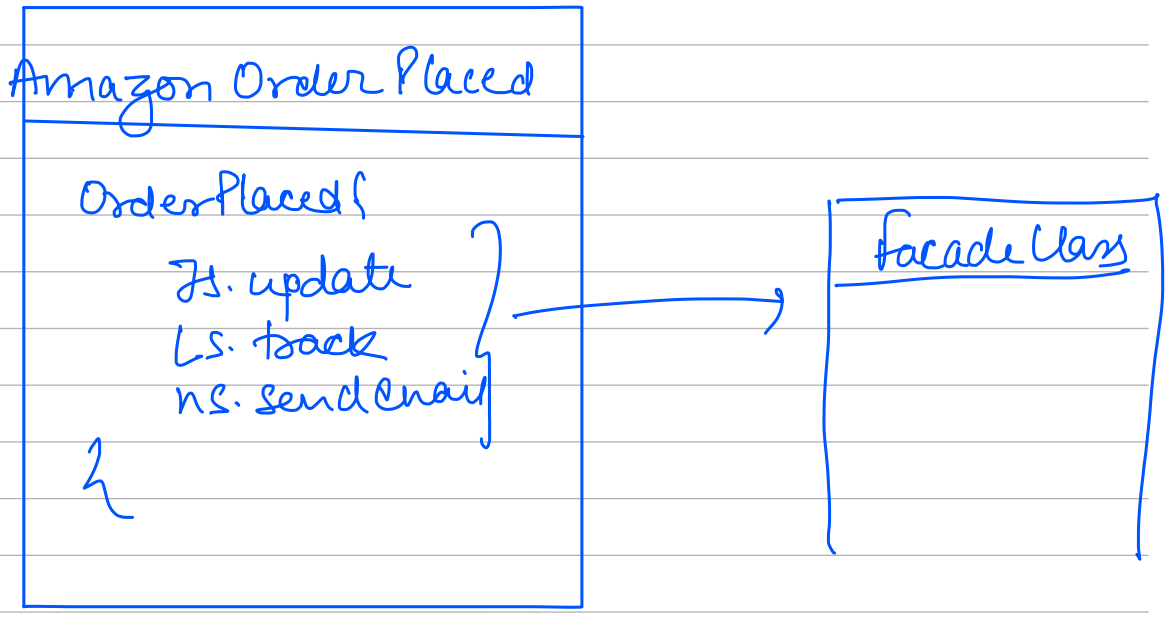
SRP
OCP

Strategy

Multiple ways / Algos to do something



Observer design Pattern

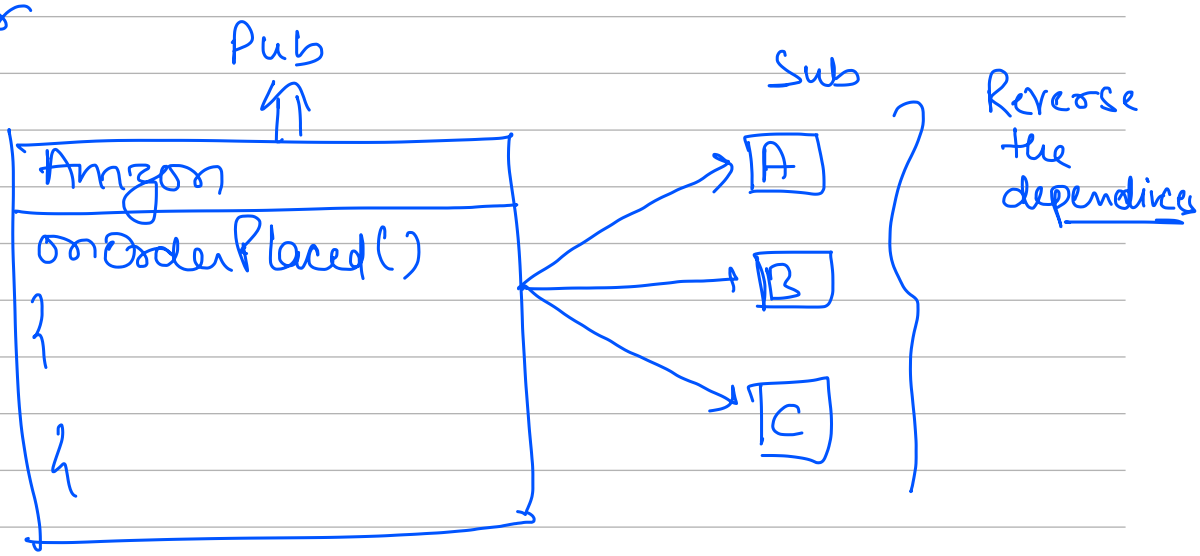


Problem statement

⇒ When an event happens, we might need to add/remove functionalities, that need a code change & thus recompilation of App"

⇒ We should be able to add/remove the functionalities at runtime without any code change.

Observer



We want to do multiple things if an event happens

↓

Multiple things wants to get executed when an event happens

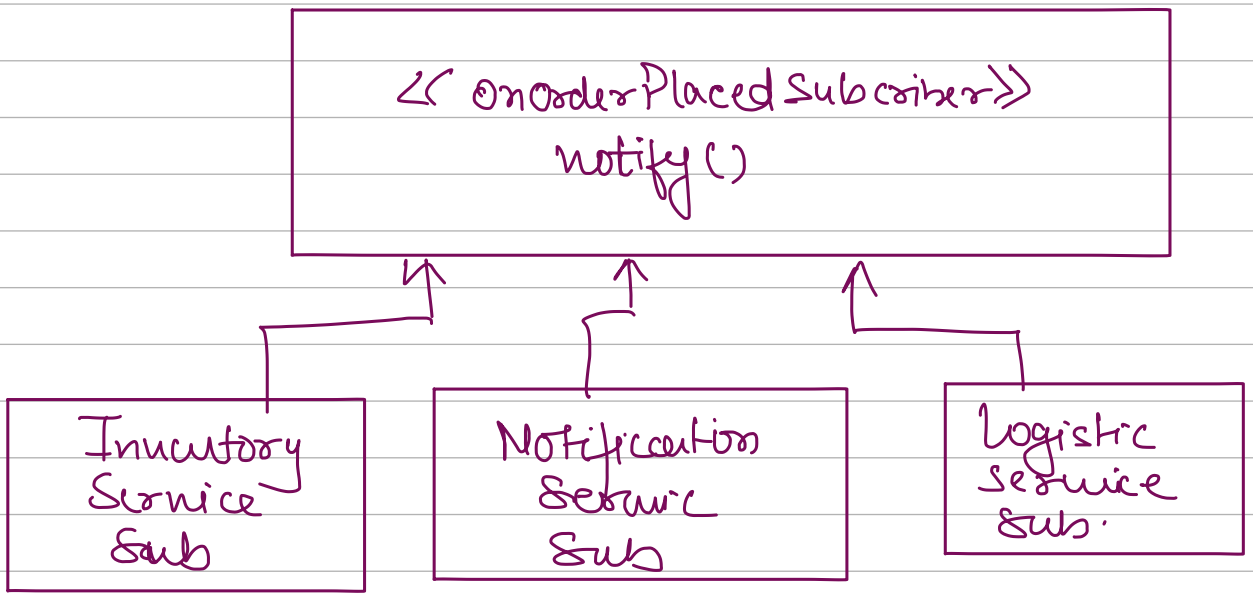
Pub-Sub Pattern

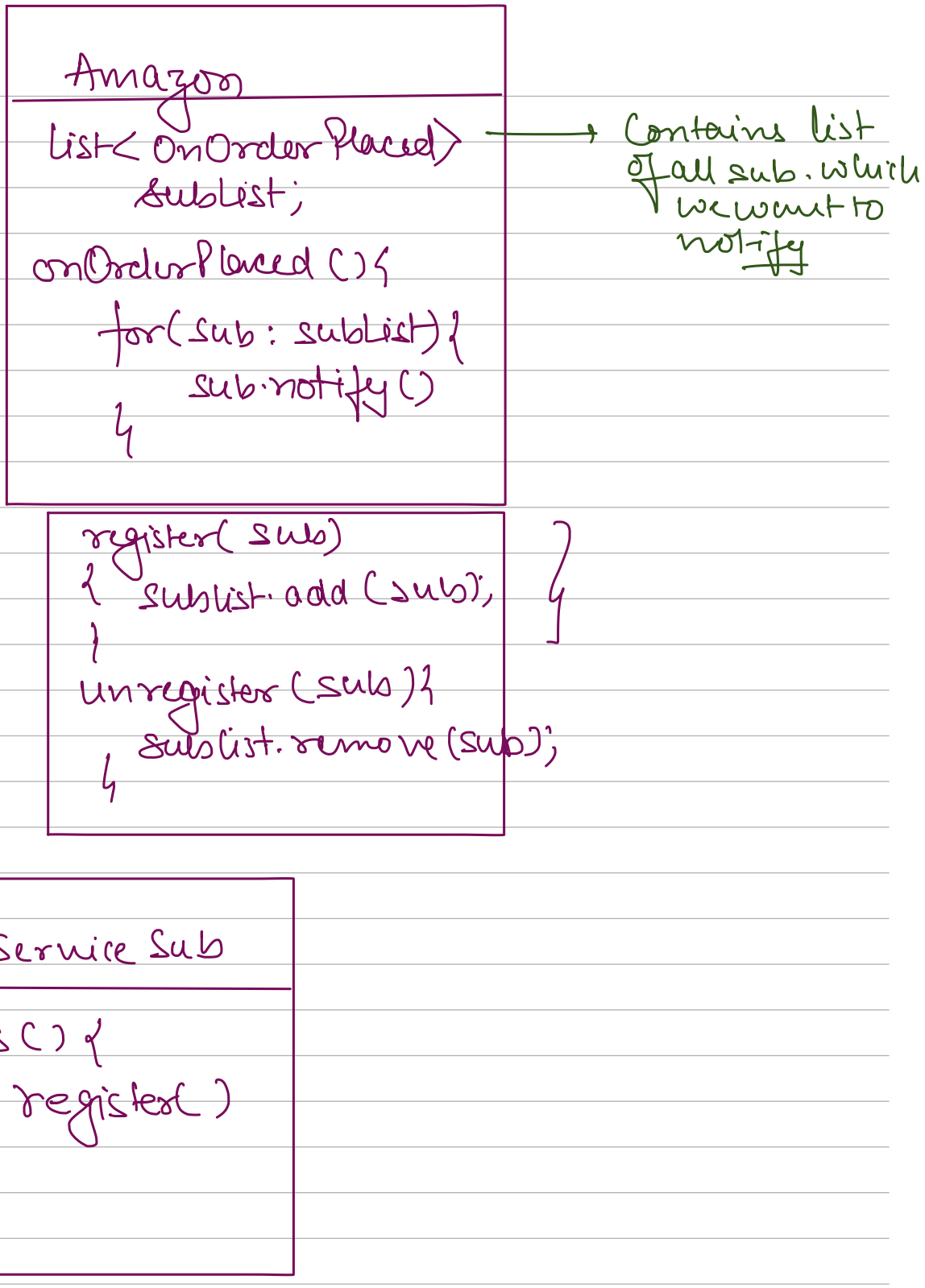
How to implement

1) Create a publisher. Pub needs to have a method that allows subscriber to subcribe

2) For each subscriber create a separate class

3) Make all subscriber classes implement an interface




Amazon
List<OnOrderPlaced> subList;
onOrderPlaced(){
for(sub : subList){
sub.notify()
}
}
Contains list of all sub. which we want to notify
register(Sub)
{ subList.add(sub);
}
unregister(Sub){
subList.remove(sub);
}
InServiceSub
ISS(){
register()
}