Static ⟩

class → Eager Initialization

i) main class mutable & builder is immutable

└ Student ⇒ not true

ii) main class immutable & builder is mutable

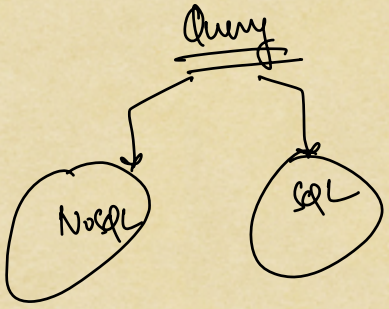
⇒ true

iii) build → object

⇒ true

iv) main class does not have any setters

⇒ true

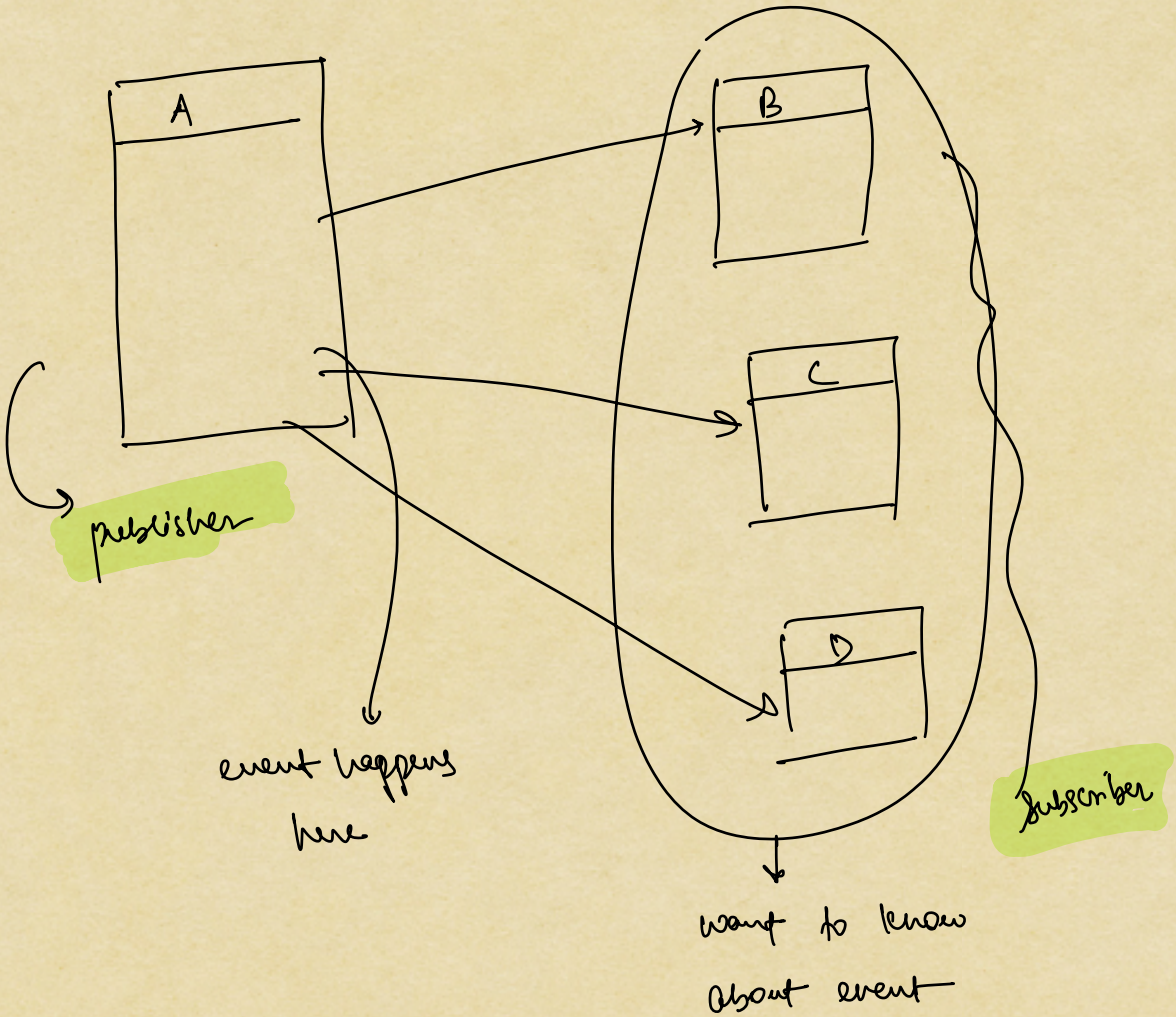


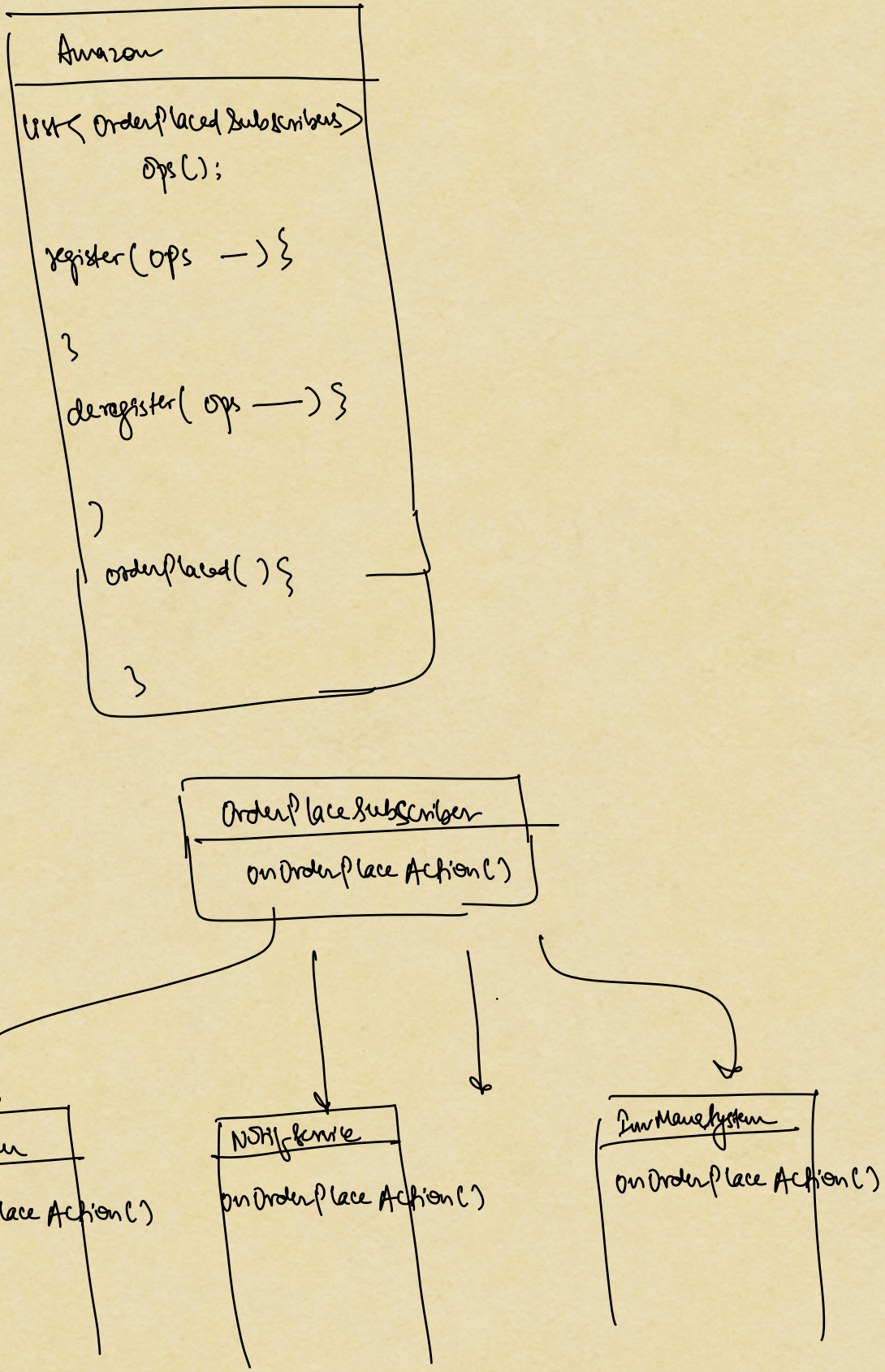
Query (createQuery())

Observer Design Pattern

* Whenever an event occurs, we might want to take some actions/tasks based on the occurrence of the event.

* Listen to an event
* Option to be a listener or not should be present at runtime




Amazon
List<OrderPlacedSubscribers>
ops();
register(ops ->{
}
deregister(ops ->{
}
orderPlaced(){
}
OrderPlaceSubscriber
onOrderPlaceAction()
InvoiceGen
onOrderPlaceAction()
NotifyService
onOrderPlaceAction()
InvManagement
onOrderPlaceAction()

UML (Unified Modelling language)

* Standardisation on how to represent different SWE concepts in diagram

* 2 types of UML

i) Structural => how the codebase is structured

ii) Behavioural => how the system works
" " features work
different features supported

UML
→ Standardisation
→ easier to understand
→ less ambiguity
→ Increases visualisation

Structural	Behavioural
i) Class diagram	i) Activity Diagram
ii) Package diagram	ii) Use Case Diagram
iii) Object diagram	iii) Sequence Diagram
iv) Component diagram	

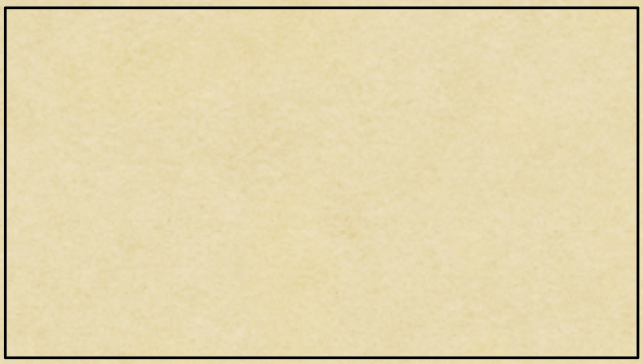
H/W

⇒ Use Case Diagram:

* Different features / functionalities that are supported by software system
* Users for these functionalities

⇒ 5 key words

1) System Boundary



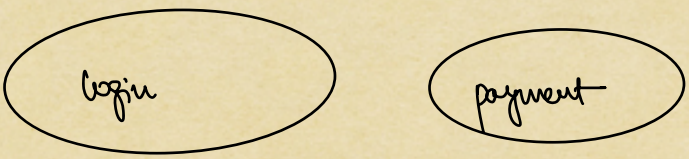
→ Represents scope of our current system

→ doesn't include any 3rd party / outsourced system

→ should be a verb

2) Use Case

→ functionality / features / actions

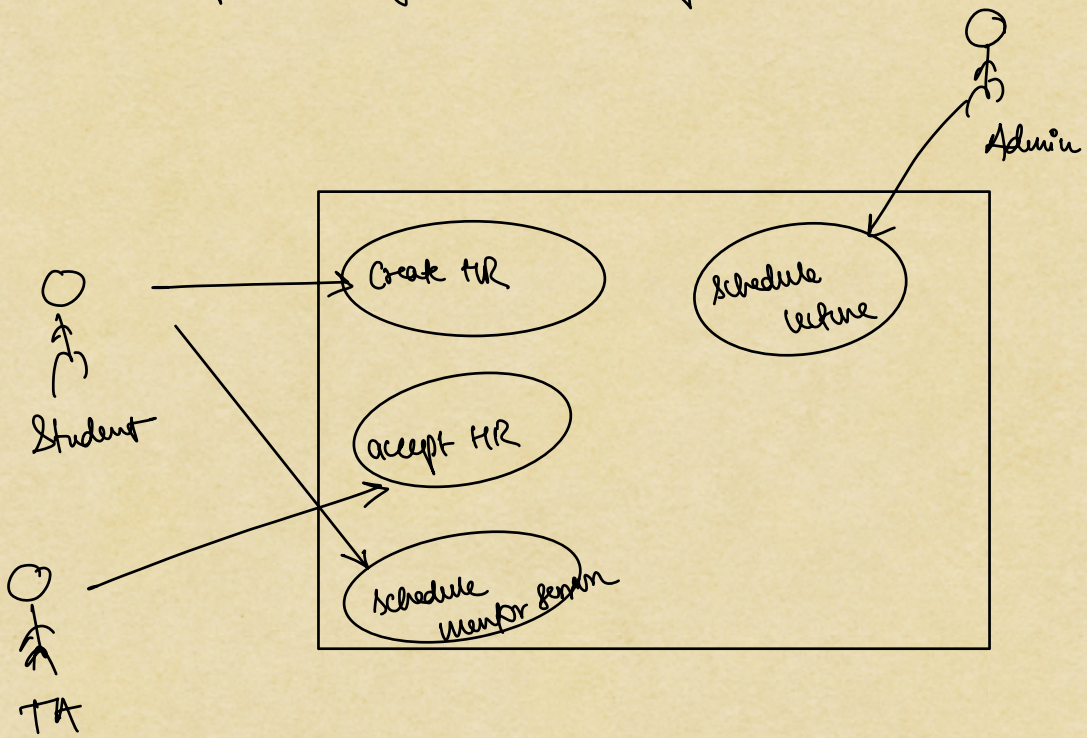


3) Actor

→ a person using a feature | use-case

→ should be a noun

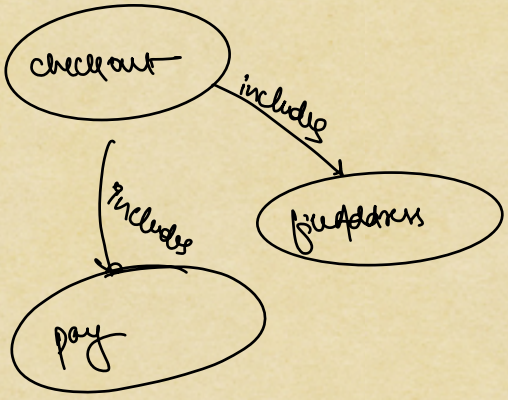
→ represent by a stick diagram



4) Includes

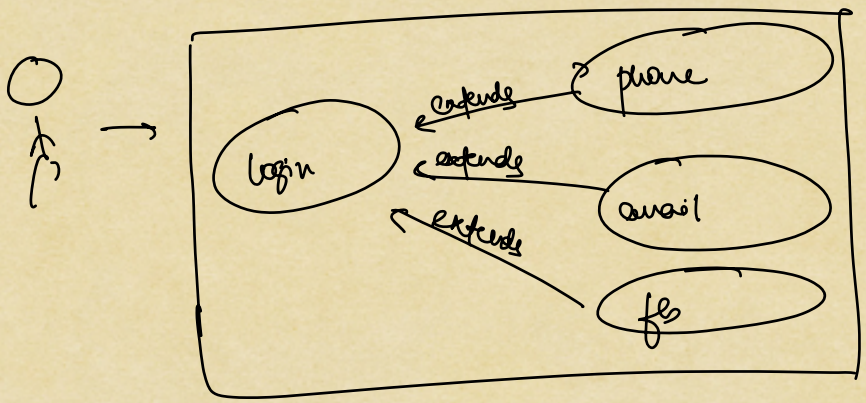
→ think of every features as a func

→ parent usecase includes the child usecase



⇒ Extends

If one feature has multiple variants :-



→ Class diagram

- attributes
- method

- Interfaces
- abstract
- aggregation & composition

↓

association

Static {

⇒ Static Initialise

}

Constructor for Static Variables

executed, when the class is loaded

```
class Student {

    List<Int> A;  <-- eager
    List<Int> B;  <-- lazy

    static {
        A = new ArrayList<>();
    }

    public void init(){
        B = new ArrayList<>();
    }
}

class Main {

    psvm() {
        Student s = new Student();
        s.init();
    }
}
```

Student {

Address {

City

}

}