UML diagrams

class diagram

use-case diagram

Association → Aggregation, Composition

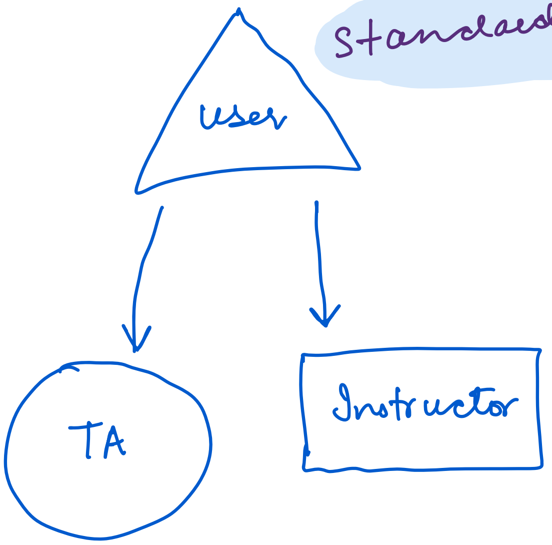
words → text, email } misunderstanding, ambiguity

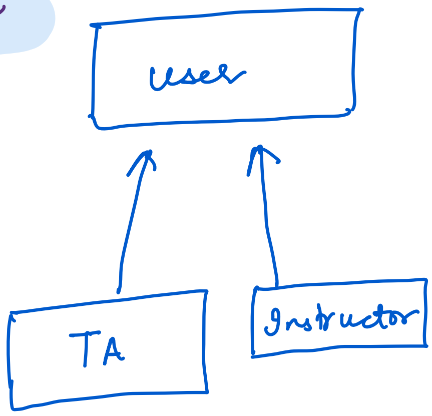
Picture → 1000 words

↓

flowcharts / diagrams / memes

Standardisation





UML diagrams :-

ASCII
↑

Unified Modelling language

Structural

codeBase is structured

Behavioural

- how your system works & interacts
- what all features you have.

class diagram

package

component diagram

use-case diagram

sequence

activity

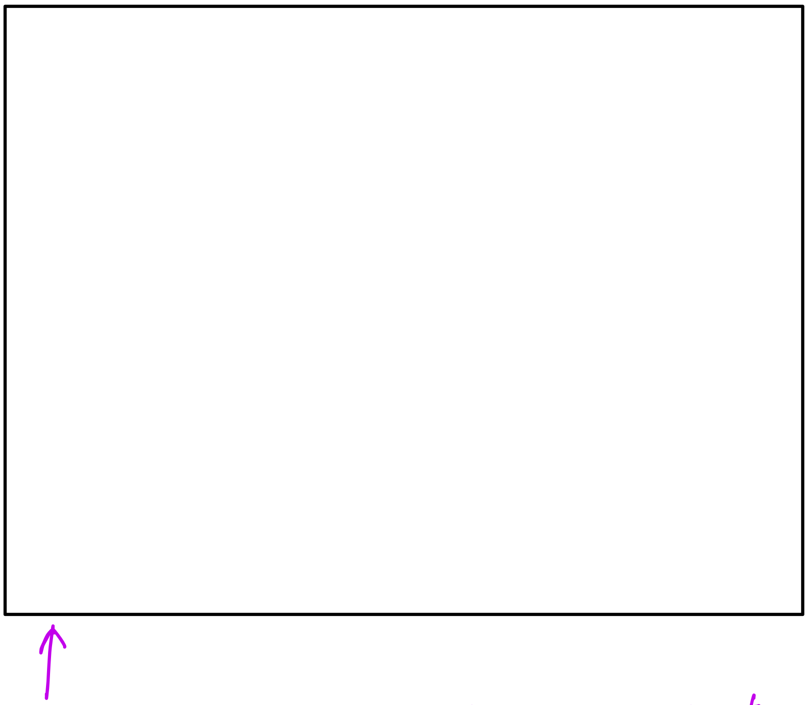
state

use - case diagrams

- diff features / functionalities / actions that your system supports

- who are the users / actors

I system Boundary

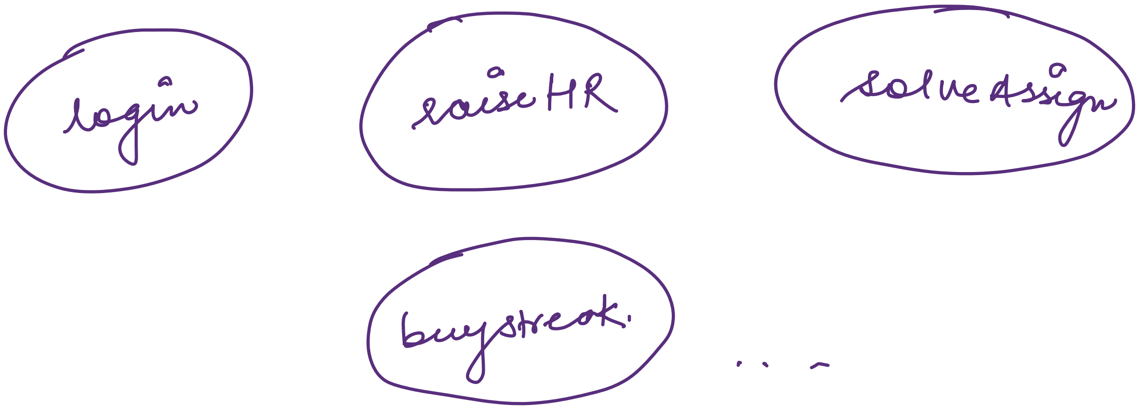


represents scope of your system

It doesn't include outsourced features.

II use-cases

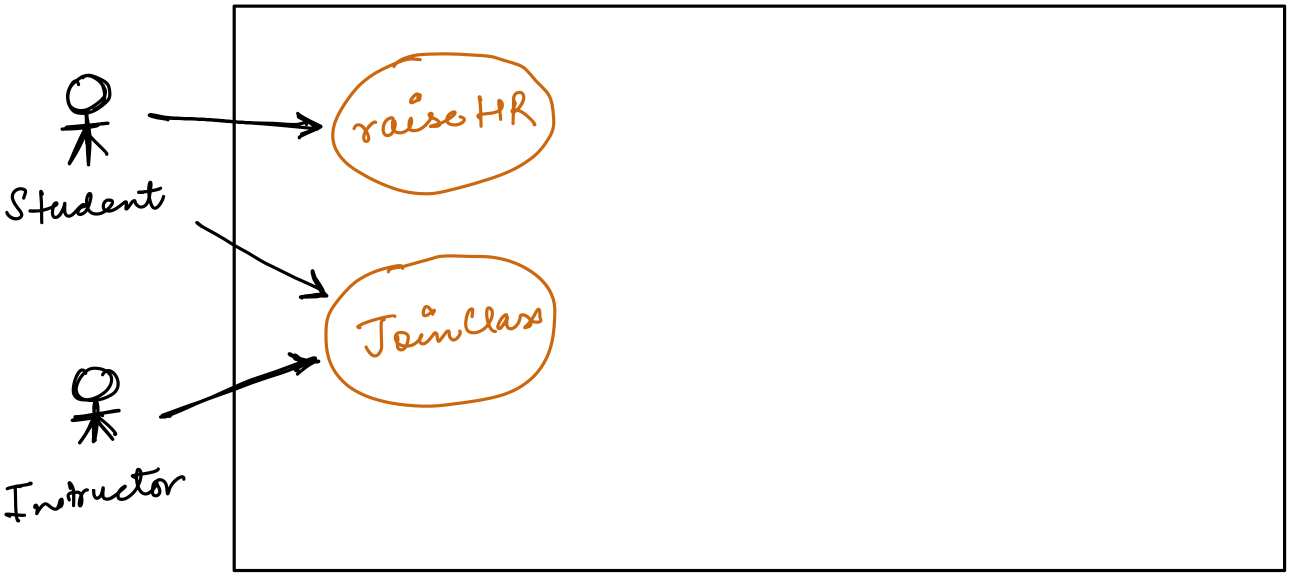
- features/ actions
- oval
- must always be a verb



. . .

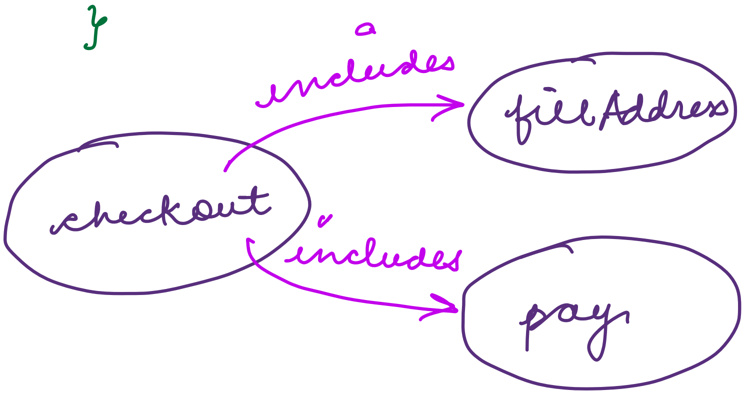
III Actors

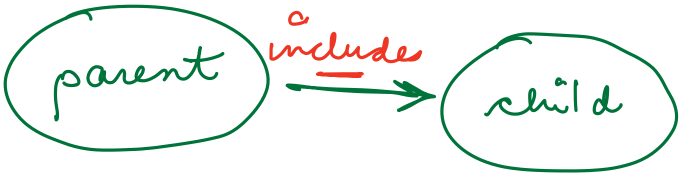
- people who use a particular use-case
- must be noun
- stick diagram



IV Includes

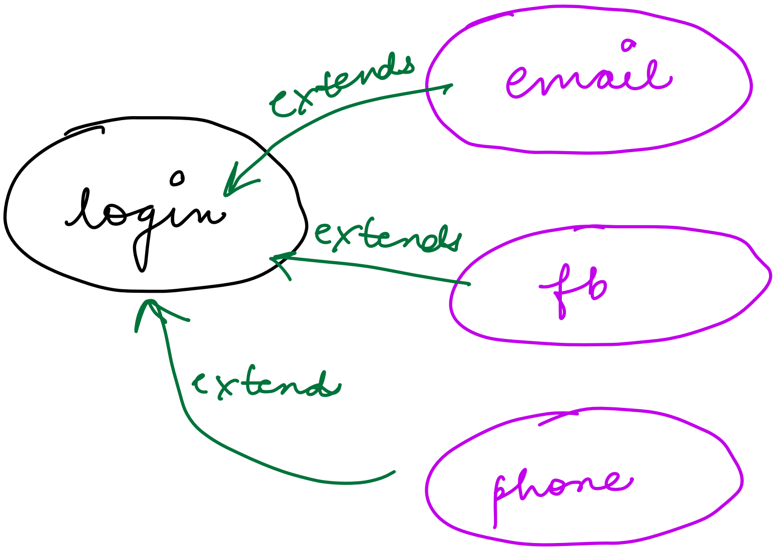
```
checkout(){
    fillAddress();
    pay();

}
```

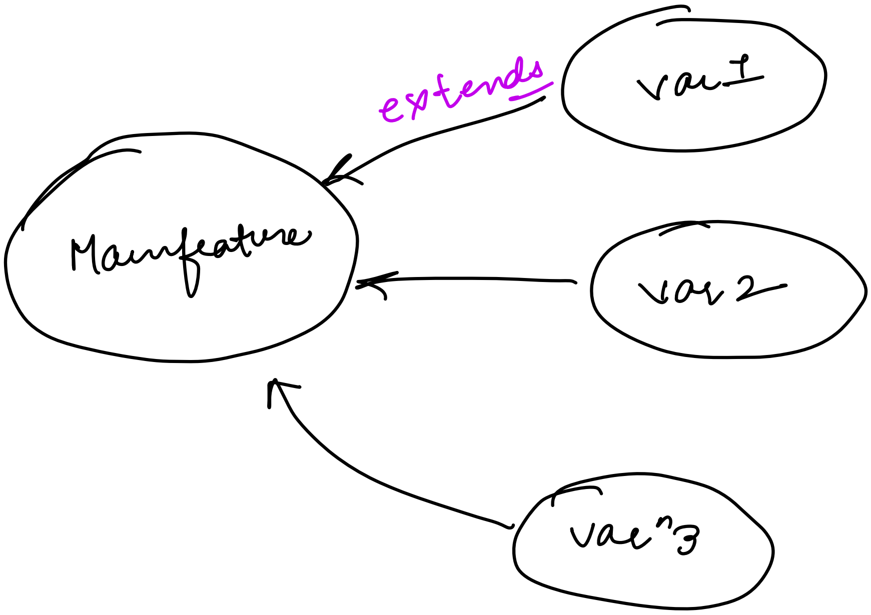


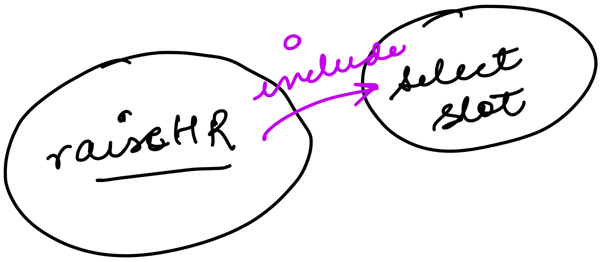


V extends

↳ If a use-case has multiple ways of doing it







- Draw a use-case diagram

Scaler

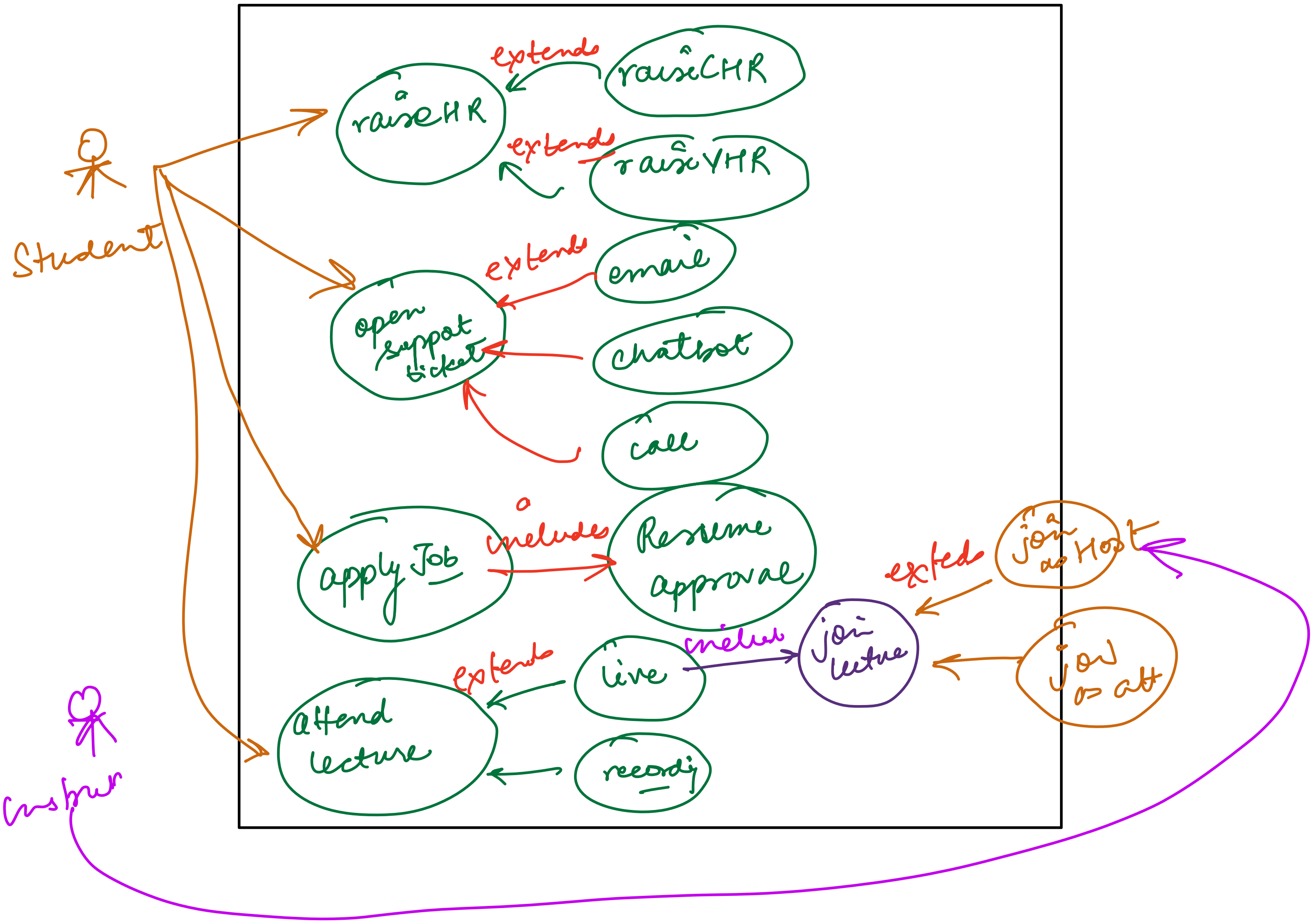
I 5 use cases

II atleast 2 actors

III 1 use-case : includes

IV 1 use-case : extends.

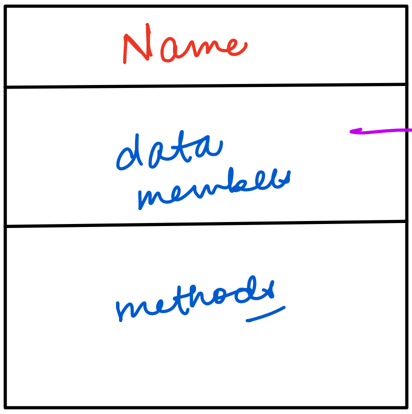
8 : 05 am



class diagram

- diff entities
  - classes
  - interface
  - enums
  - abstract
- representing relationship b/w entities
  - inheritance
  - implementing interfaces
  - having another obj/ref as an attribute

class



→ public string name;

access modifier   name : data-type

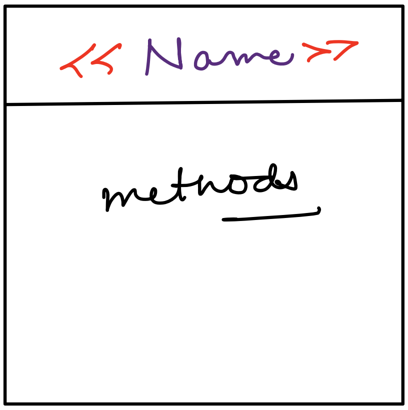
public +
private -
protected #

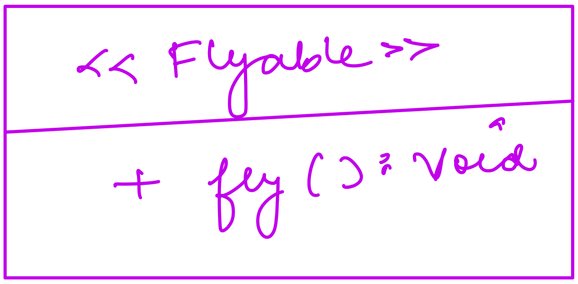
+ age : int
+ gradYear : int → static
- email : string
⋮

access modifier   name (dt1, dt2) : return Type

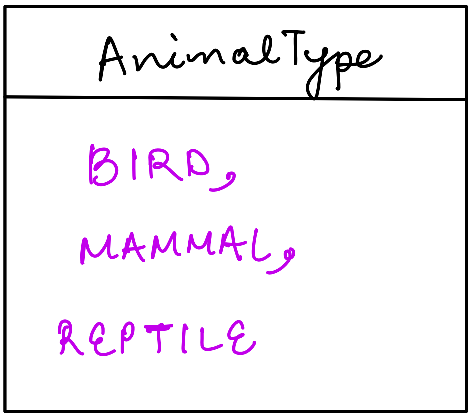
+ attendClass(string, string) : boolean

Interfaces



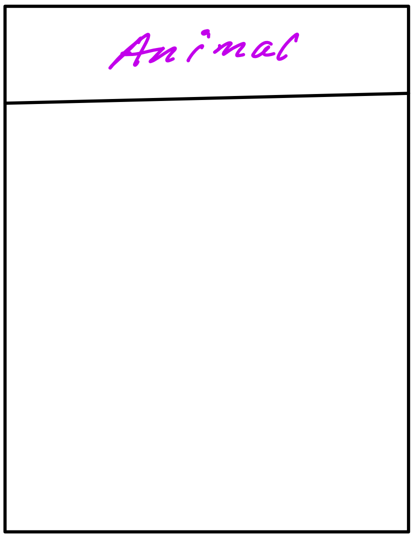


Enums:



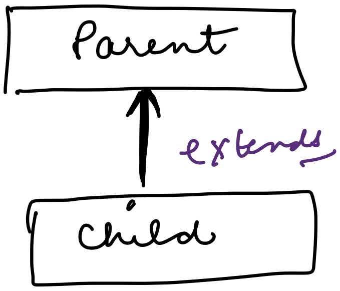
abstract

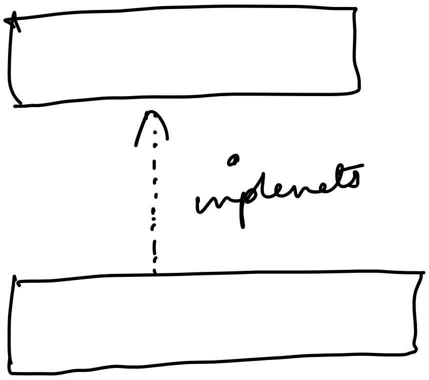
Italics



Abstract

Relationships





Association :

Having another class obj / ref as an attribute

Aggregation

- collecting
- weak
- contained entity has its own individual existence

Composition

- to create
- strong
- container & contained are highly dependent

```
Student {
    Batch batch;
}
```

```
Show {
    Movie movie;
}
```

Bullet

↓

10 types

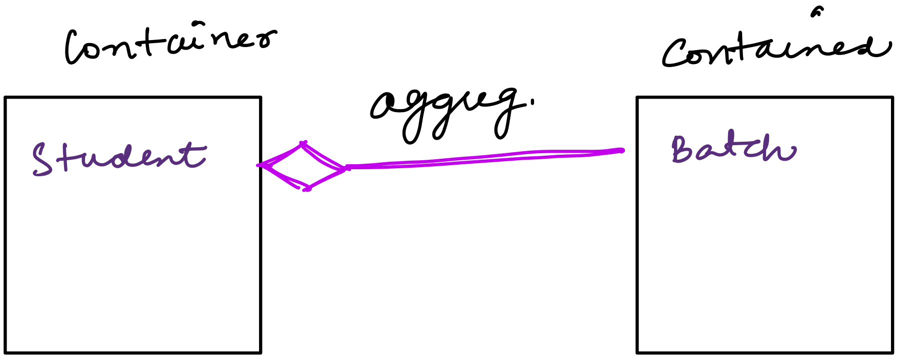
```
Order {          weak
    Customer customer;
strong
    List<OrderItem> orderItems;
}
```

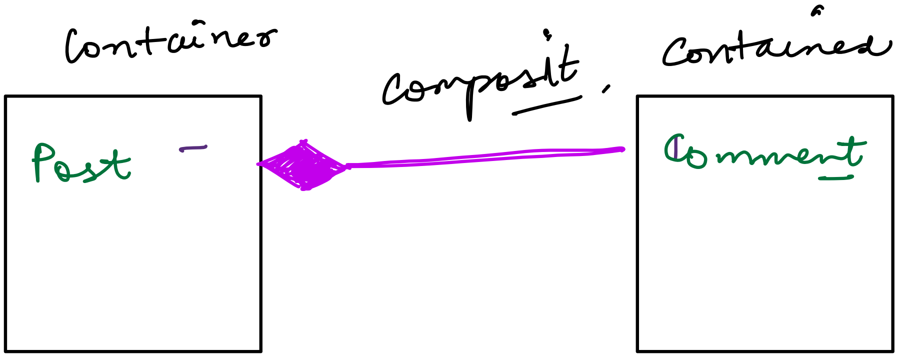
```
Post {
    List<Comment> comments;
}
```

```
WhatsAppGroup {
    List<User> users;
}
```

```
Flying Bullet {
    Bullet bullet;
}
```

```
OrderItem {
    Item item;
    String name;
    int count;
}
```


Container
Contained
aggreg.
Student
Batch


Container
composit
Contained
Post
Comment