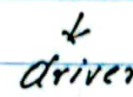
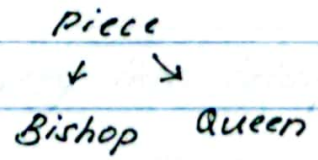
20th Sept

Relationships between classes

① is-a ⇒ Inheritance

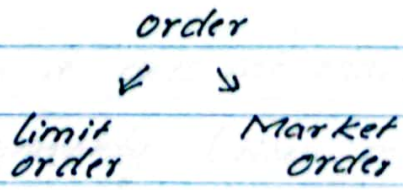
eg:


driver is-a user

eg:


Queen is-a piece

eg: Zerodha example



In certain cases, a type property might be enough. Need to have enough reasoning to use what you use - inheritance

② has-a relationship

- Order has a delivery address
- Car has-a engine

Can the subclass exist without parent class.

→ Restaurant —has-a→ Menu

Menu can't exist without restaurant
When restaurant is deleted, menu is also removed

→ Menu → food item

Every item belongs to the menu.

eg: Cafe 1 → Burger - different image / descript
Cafe 2 → Burger

So it does not make sense for food item to exist separately. (When menu is deleted → item deleted)

→ Types

has - a

Aggregation — Person - bag

Composition — ~~Part of Relation~~ Person - heart

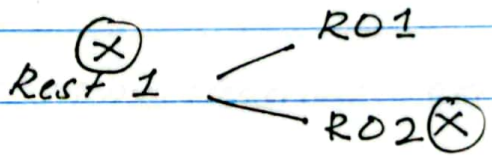
How to identify?

Composition — When entire life cycle of the object depends on the class its used.

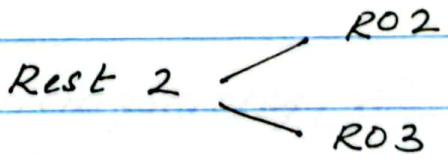
eg: User's shipping address.

Rest. owner can not be ~~aggregation~~ composition:

eg:



When restaurant 1 is removed, restaurant owner also gets deleted



All Many to Many relations ⇒ Aggregations

UML

① A ——▷ B Inheritance - A is a B (B is baseclass)

② A ◇—— B Aggregation - A has a B

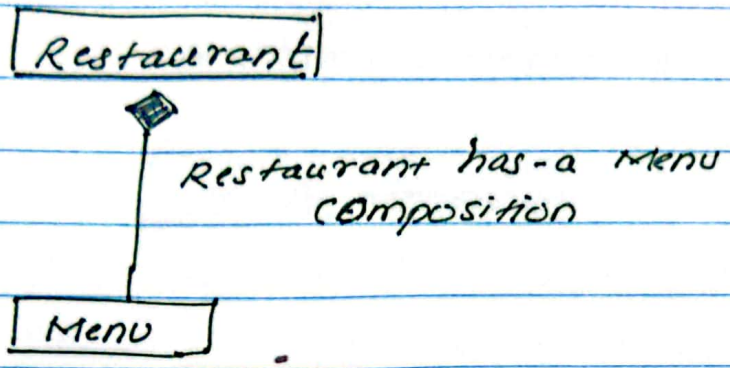
③ A ◆—— B Composition - A has a B (B can't exist without A)

④ A —— B Bidirectional - can call each other

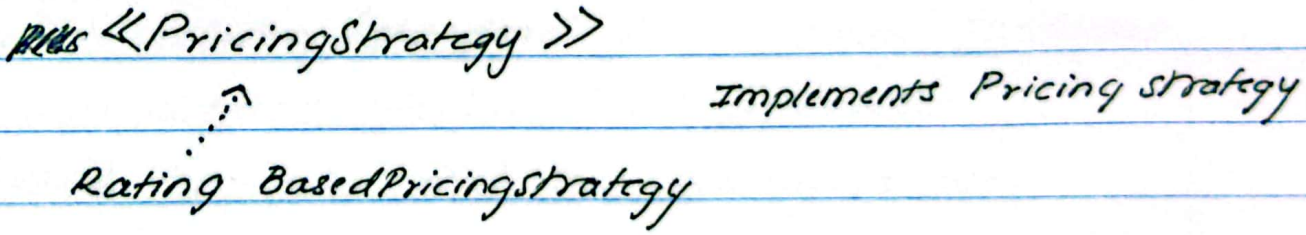
⑤ A ·····▷ B Implements - A implements B

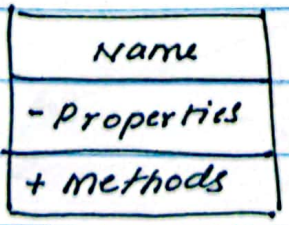
⑥ A ——→ B Unidirectional association

UML
Examples

① 

Lifecycle of Menu depends on restaurant

② 



3 sections on each class

Access modifier in ~~public~~ UML

- Private
+ Public
# protected

static → under lined in UML diagram
→ shared member of the class

Concepts to remember :

- KISS - Keep It Simple, Silly Stupid.
- YAGNI - You Aren't Gonna Need It
- DRY - Don't Repeat Yourself

Design Patterns

- Proven solution / standard for common problems.
- Design patterns follow SOLID principles.
- Tested templates that are ready to use.