AGENDA

-> design T·T·T

(R) ★] APPROACHING SCHEMA DESIGN:

# HOW TO FIND CARDINALITY -

1:1

1:M / M:1

M:N

1.> for all classes in class diagram - create table

2.> for each class, Add Primitive attributes in table

Eg: BookMyshow - ticket

```
class ticket {
    id
    time_of_booking
    List<seat>
    user_id
}
```

s1.> create table

s2.> Add
id / time / user-id

3.> for every Relation - find cardinality
based on cardinality - create new tables

## *] CODE / GOOD PRACTISE :

-> (I) Judges by code

-> Project structure
-> Atleast something must RUN

Always code by functionality.

Note:

Instead of models → controllers → services (x)

Go ahead with :

All Modes → Requirements

(classes in your class diagram)

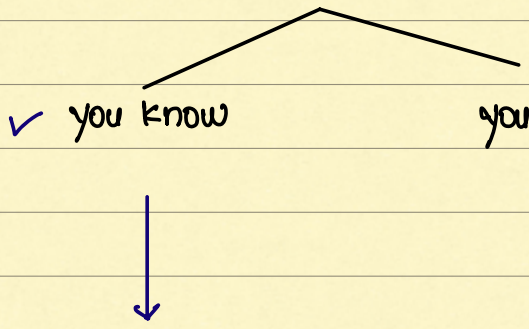
*] TERMINOLOGIES:

Models / controllers / service

Models → All classes in class diagram
Controllers → Entry point for Requests
service → does all hardwork
Repo → classes that Query d/b. (DAO)

*) DESIGN TTT

S1.> Overview




Ask:

(i) confirm with Interviewer [what your understanding]

(ii) what type of design:
- Entity design
- Interactive system
- web-API system

NOw, you have idea about expectations

S2.) Gather Requirements:

→ Give suggestions with Rational.

Eg:

"Board size is 3x3" —— x

↳ Q.) will board size be 3x3 OR can be variable ?

# EXAMPLES OF REQUIREMENTS:

i) size of board can vary
ii) # of Players : (N-1)
iii) Every Player has own symbol
iv) Every Player can select their symbol
[UNIQUE]
v) who Plays first:
⟶ Randomized list & Play in order

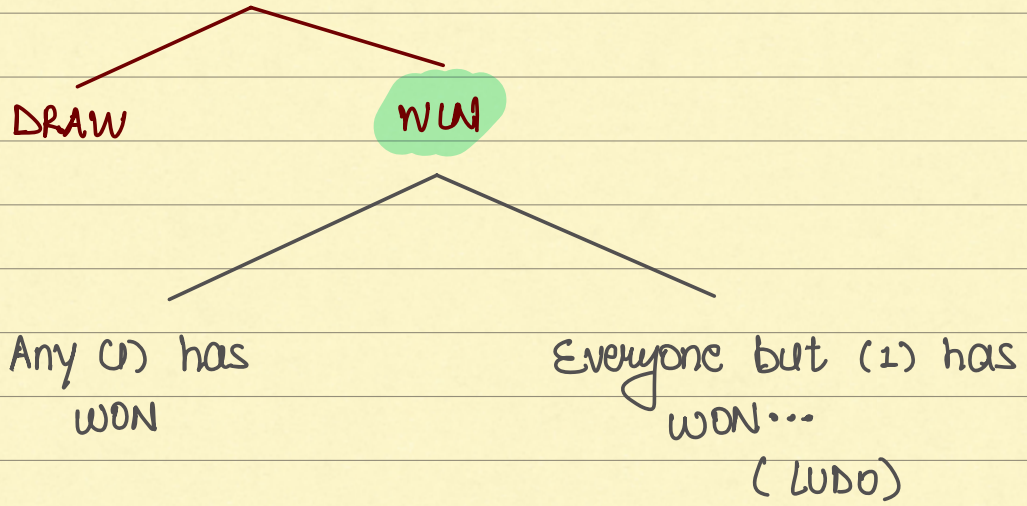
eg: A, B, C, D $\xrightarrow{\text{Random}}$ B, C, D, A

vi) Can Game have bots ?

Edge case — can all Players be bots ?

vii) Bot difficulty level — Easy / Med / Hard

viii) How/ when Game would start ?
ix) How can Game End ?


DRAW
WIN
Any (1) has WON
Everyone but (1) has WON...
(LUDO)

# MULTIPLE WAYS A USER CAN WIN:

can be Multiple ways.

ROW
COL
DIAG

→ This is decided at Game start

x.> Blocked cells [Not considered]
xi> Player can UNDO
↳ ANY NO'S of UNDO.

xii> Re-watch Game (store all Moves)

# THINGS TO NOTE:

Any Game: size of board / # Players / bots /
win-criteria / UNDO / Leaderboard /
Game start / Game end

S3.) Usecase diagram — Required only for web.

S4.) class diagram:

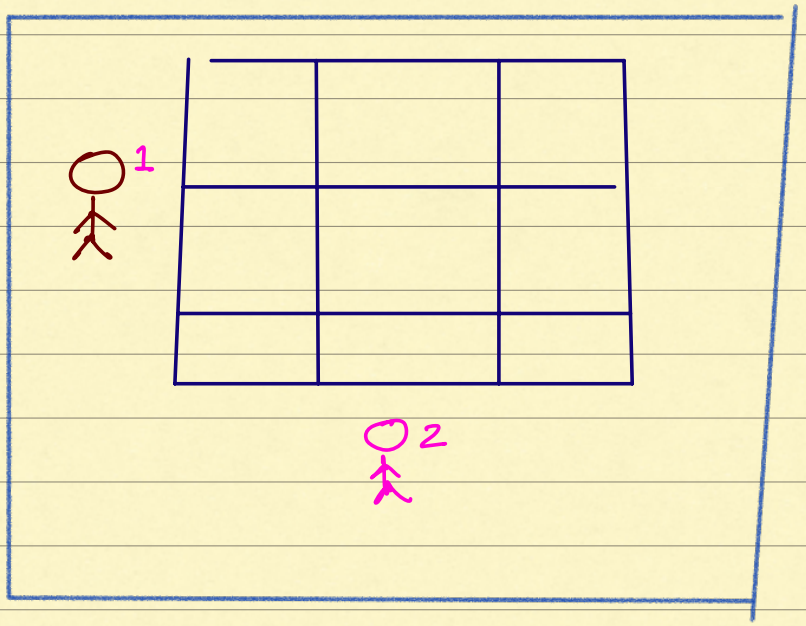
Q.) list all Entities

↓

WAYS:

① Visualization ✓

② NOUNS



## *] IMPLEMENTING UNDO :

Consider as - Global Move - anyone can do.

-> Every Move is stored in List<Move>

↓

UPON UNDO :

(1) way

s1.) Remove last element from List<Move>
s2.) update board / cell
s3.) Update turn

↓

IMPLEMENT :

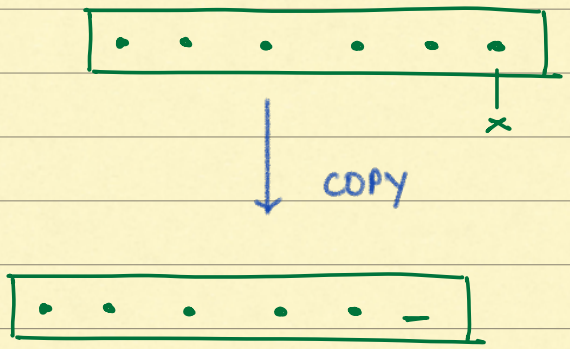
Chess Game — can be challenging

(2) better Approach for UNDO-

-> we have List<Move>

↓

UNDO : Re-do all moves from 1st to (N-1)




S1:> create New board

S2:> Redo All Moves except last

*) PROBLEMS

->

->

(3) ANOther better Approach:

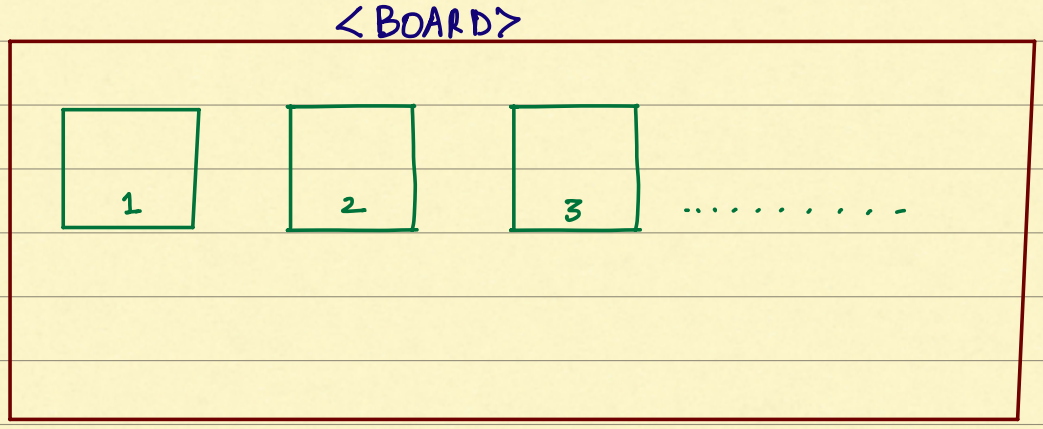
(time Machine)

-> we stored: list<Move> ✗

-> Instead → Board ———

list<board>

(create...)




Easy to Implement → you just need to Remove last index of <board> list.

→ Approach says: store 'Board' as well; after every move

ADVANTAGE:

1.) you know game board at every state
2.) Easy to Implement

DISADVANTAGE:

1.) Bad space complexity

SUMMARY:

* if reverting move is easy – use (A1)

Else

A2 OR A3

Tip → different ways to implement UNDO → can use

↓

design Pattern