Lecture34:- Let's Build the Snake & Ladder game

Requirements

- size of boards should be scalable.
- There are standard game rules and should be further extensible.
- There can be game setup strategy like Random setup, custom setup, standard setup etc.
- Notification (in-app).

UML Diagram Details

① we will be using top-down approach i.e:- starting with main class then creating supportive class as needed.

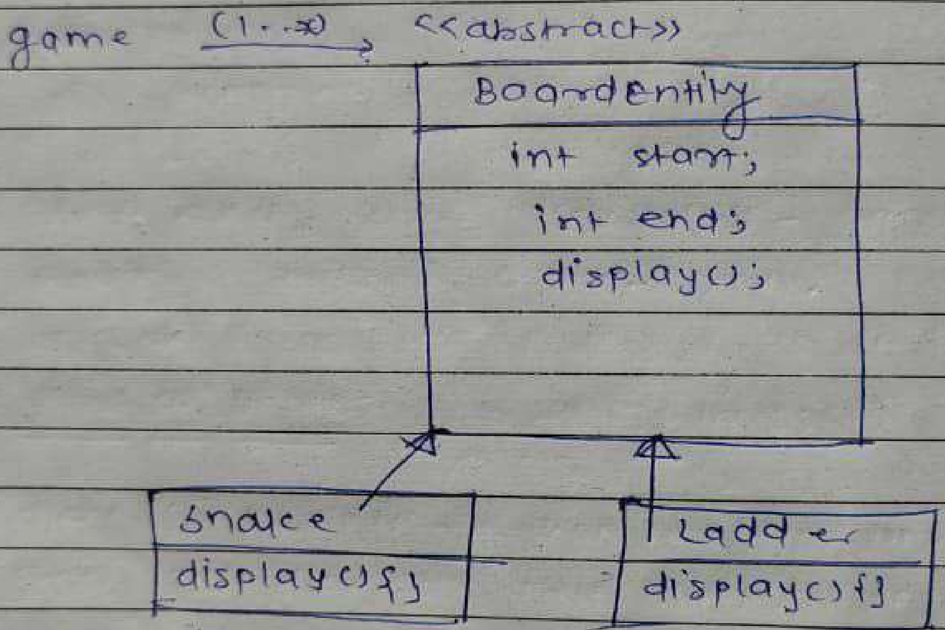
② If for creating cells in our brand we are thinking / of 2D vector then it's no need As we can store it in 1D vector as no will be consistant but issue of snake & ladder ? then map <snake, ladder> will be perfect.

③ create the orchestrator class - Game and for playing game we will need board then create Board class.

④ Now in Board class first requirement is board scalablity. we will take size from client.

Page No.
Date

(5) Now we have to display snake and ladder so we can use two seprate vector to do so but instead we can see use parent class BoardEntity inherit and override by snake and ladder class.
snake move →. Head - tail.
Ladder move tail - Head.



(6) Now in our Board class we use map as discussed.
- map <int, BoardEntity> mp;
storing head and tail of index of snake & ladder.

(7) why not using ID? because we have stored size and we know it's continuous value (1 -- size) so we no need to give extra space.

⑧ Methods in Board class:-

① addEntity (BoardEntity b) {}

- Takes BoardEntity and add in our vector and entity would know it's start and end point.

② canAddEntity (pos);

- check wheather we can add entity or not. if we add the entity the check ~~the~~ there will be snake / ladder.

③ display()

- display full board.

④ setboard (setop. starts){}

- It call diff type of strategy (requirement random, custom & standard as type pass.

⑨ Let's create setopstrategy of board and it's concreate classes / strategies using strategy pattern.

⑩ Let's disscuss concreate strategies

① standard strategy:- It's one which we play normally of (1-100)size and fixed place of snake & ladder

② Random strategy:-

Here no. of snakes & ladders depend on difficulty suppose if easy - 70%. ladder

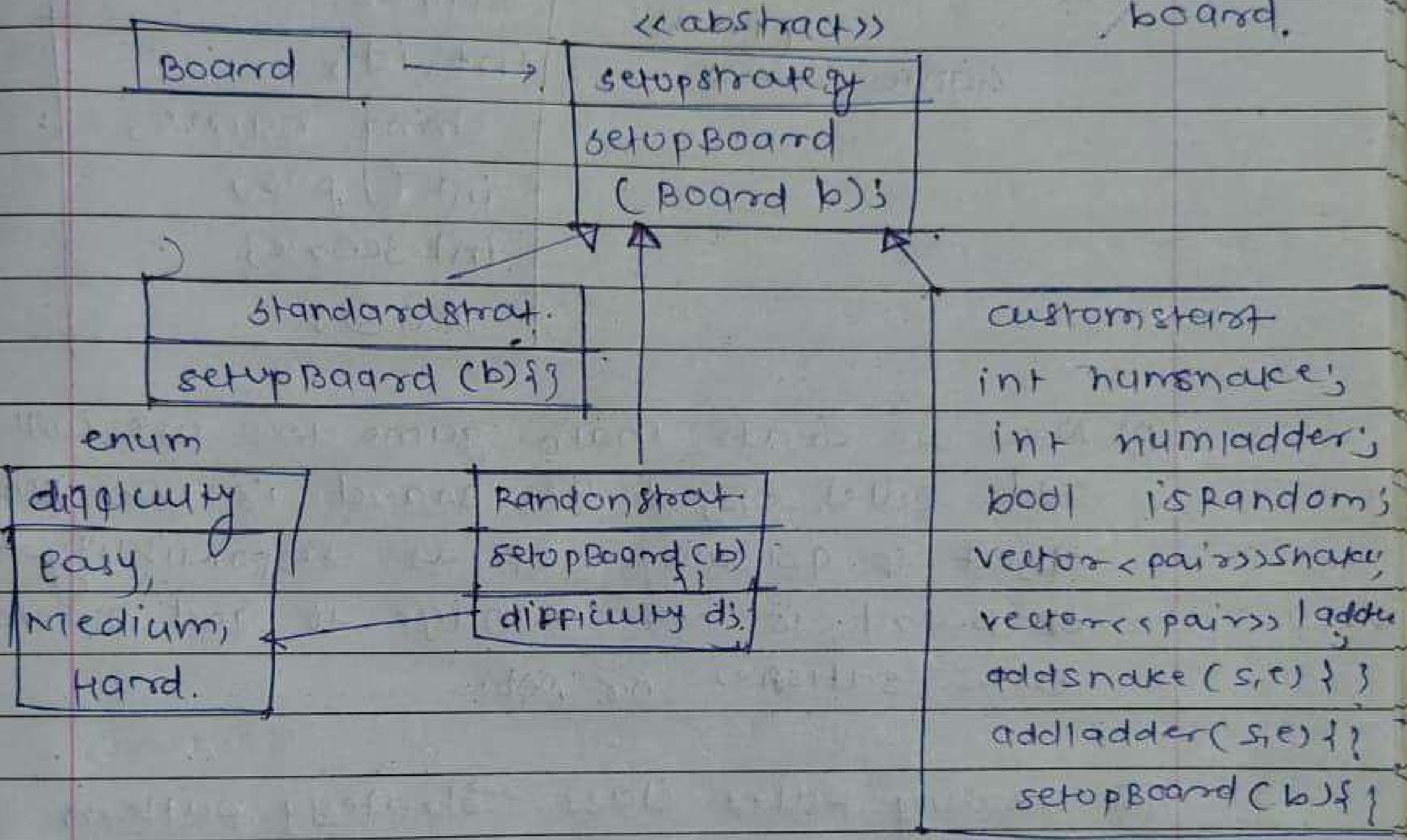
hard:- 70%. snake 30%. ladder

medium:- 50%. snake 50%. ladder.

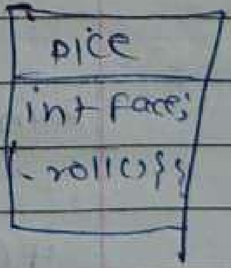
methods - ① addSnake (s, e) {} ② addLadder (s, e) {} ③ setupBoard (b) {}

start → s, end → e

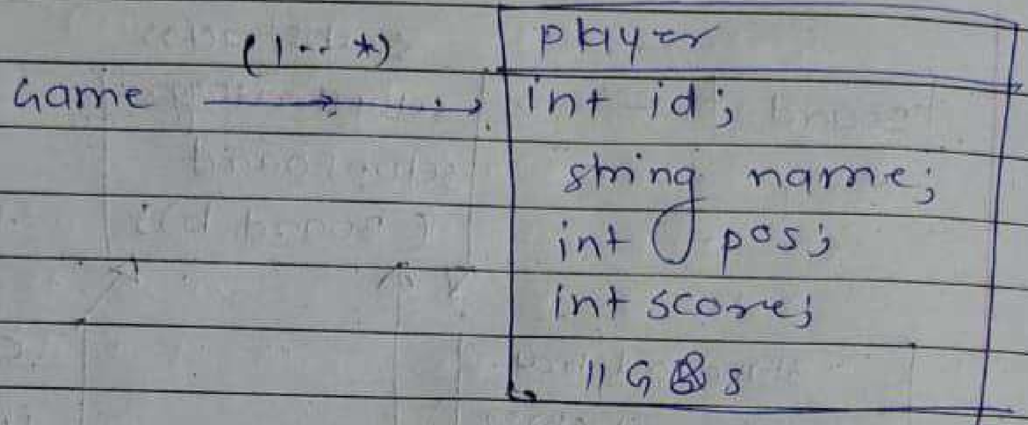
add snakes and ladder in our board.



⑪ now, we create Dice class it should be scalable ∴ int faces → will show maximum no. on dice. and roll() method.

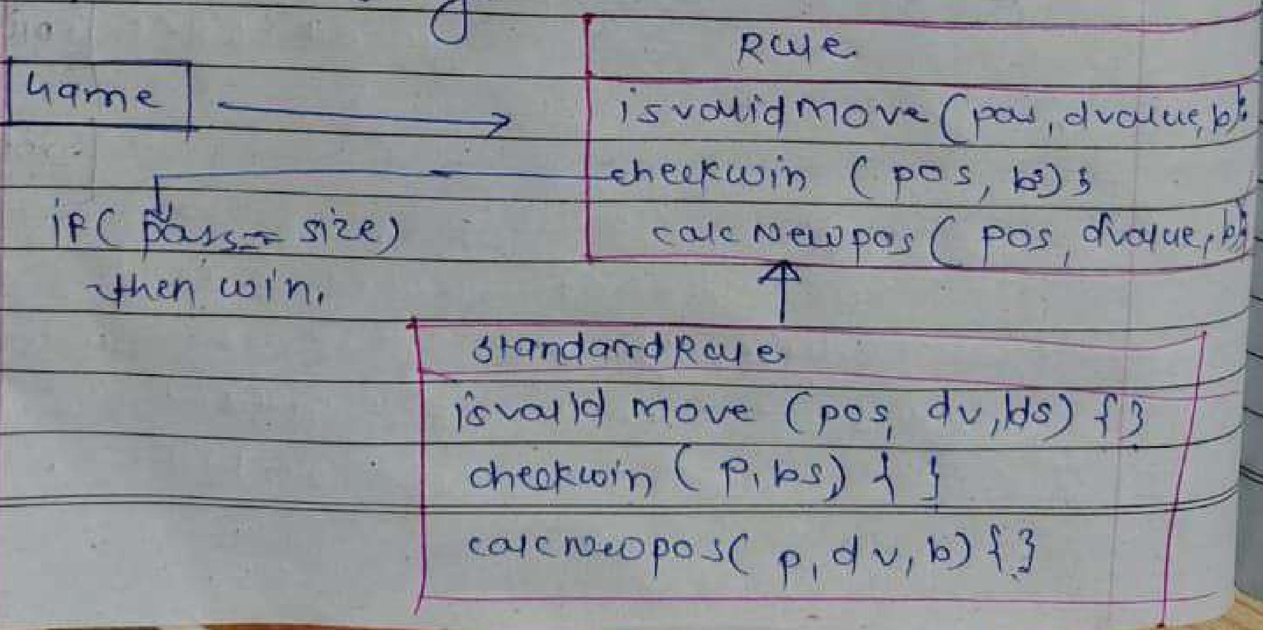


⑫ Now we create dequeue in our game class for players and then create player class.



⑬ Now we create main game class we will add rules object as Board is our dumb object if following SRP as responsibility of board is to manage it and no check rules satisfies or not.

⑭ Creating Rules class - Strategy pattern
Right now we are only using standard rule but if incase we need different rules so it should be added easily.



(15) Now last step is heirrarchy Notification for which we use IObserver and consoleNotifier

(16) we add more methods to game class.
① addObserver(ob);
② addplayer(p);
③ bool isGameover();
④ play();
⑤ notify().

(17) And finally our factory class - GameFactory having method createGame().

# Flow of Game :- while(!Gameover)
{}

① client calls play() → then we find currentplayer then roll dice after that checkValidmove → if yes then check for calNewPos as to check for snake & ladders.
② to check it, it will go to map entities in our board class and depending if it's there or not.
③ Then we will check win() condn if true then true and GameOver - true & loop terminates
④ If not then again find new player through dequeue and repeat step till someone wins.