Tic Tac Toe LLD Practice

Requirement Gathering

1. Game will be between 2 players or multiplayer?
2. Game will be player between HUMANS or AI can also play with HUMAN?
3. Board size will be 3*3 or dynamic? Board will be square or can be rectagle as well?
4. Game play symbole will be X & O or can be any symbol? Predefined symbole or selected by player?
5. Winning case? Row Col Diagonal Anti-Diagonal - anyone filled with same symbole?
6. Do I need to take any input from user of can I hardcode for LLD?

Flow

1. Game starts with 2 players: player_1 and player_2
2. Player_1 / Player_2 choose a coorninate(x,y) to place his symbole(X, O)
    2.1. Validate coorninate(x,y) is valid: empty, in board
    2.2. check if curr player won after his last move: check row, col, diagonal, anti-diagonal
    2.3. if curr player won, [end the game - curr player won]
    2.4. check if no place left in the board to play next move -> [end the game - TIE]

Note: Game goes infinite till either one player won or no place left in board to play(TIE)

Entity

1. Enums: PlayerType, GameStatus, Symbole
2. Player (Factory)
    1.1. Human
    1.2. AI
2. Game Board
    2.1. TicTacToe Game Board (Singleton)
3. Game Controller: Has players and game board instance, plays the game

Design Pattern

1. Singlton - to have single game board instance - MUST HAVE
2. Factory - Create instance of player - Human / AI - MUST HAVE
3. Observer - Notify player for there move - CAN HAVE

Code Structure: https://onecompiler.com/java/43qf52bu4

```
enum GameStatus {
   START,
   IN_PROGRESS,
   PAUSE,
   TIE,
   STOP;
}

enum PlayerType {
   HUMAN,
   AI;
}

enum PlayerSymbol {
   X,
   O;
}

abstract class Player {
   private String name;
   private PlayerSymbol playerSymbol;

   Player(String name, PlayerSymbol playerSymbol) {
      // constructor to create a player instance
   }

   abstract boolean playMove(Board board, int row, int col);
}

class HumanPlayer extends Player {
   HumanPlayer(String name, PlayerSymbol playerSymbol) {
      supar(name, playerSymbol);
```

```
        supar(name, playerSymbol);
    }

    boolean playMove(Board board, int row, int col) {
        board[row][col] = this.playerSymbol;
    }
}

class AIPlayer extends Player {
    AIPlayer(String name, PlayerSymbol playerSymbol) {
        supar(name, playerSymbol);
    }

    boolean playMove(Board board, int row, int col) {
        board[row][col] = this.playerSymbol;
    }
}

class PlayerFactory {
    public static getPlayerOfType(PlayerType type, String name, PlayerSymbol symbol) {
        switch (type) {
            case HUMAN -> return new HumanPlayer(name, symbol);
            case AI -> return new AIPlayer(name, symbol);
        }
    }
}

class Board {
    private PlayerSymbol[][] board;
    private int size;

    private Board(int size) {

    }

    public static Board getBoardInstance(int size) {
        if (board == null) {
            this.size = size;
            board = new PlayerSymbol[size][size];
        }
        return board;
    }

    private boolean isValidBoardPosition(int row, int col) {
        // check if row and col are in between 0 and size
    }

    private boolean isValidMove(int row, int col) {
        // check if choosen board position is empty to place symbole
        return isValidBoardPosition(row, col) && this.board[row][col] == null;
    }

    public boolean placeSymbol(int row, int col, PlayerSymbol symbol) {
        if (isValidMove(row, col)) {
            this.board[row][col] = symbol;
            return true;
        }
        return false;
    }

    public void printBoard() {
        // prints the curr game board
    }

    public boolean isBoardFull(Board board) {
        // check the matrix board, if all are non-null return true
    }
}

interface BoardGames {
    void play();
}

class TicTacToe implements BoardGames {
    private Board board;
    private Deque<Player> players;
    private GameStatus status;
    private Scanner scanner;

    TicTacToe() {
        scanner = new Scanner(System.in);
        // get size from user
        this.board = new Board(size);
        // get playerType from user
        // get name and symbol
        Player player1 = PlayerFactory.getPlayerOfType(playerType, name, symbol);
        Player player2 = PlayerFactory.getPlayerOfType(playerType, name, symbol);
        players.add(player1);
        players.add(player2);
        status = GameStatus.START;
    }

    void play() {
        status = GameStatus.IN_PROGRESS;

        while(status == GameStatus.IN_PROGRESS) {
            this.board.printBoard();

            if (this.board.isBoardFull()) {
                status = GameStatus.TIE;
                break;
            }

            Player currPlayer = players.removeFirst();
            // Get ROW and COL value from the user
            boolean moveComplete = this.board.placeSymbol(ROW, COL, currPlayer.playerSymbol);

            if (!moveComplete) {
                // curr player need to play again - choose another ROW & COL
                // currently he player an invalid move
                players.addfirst(currPlayer);
                continue;
            }

            if (checkIfCurrPlayerWon(row, col, currPlayer.playerSymbol)) {
                // currPlayer won - show message
                status = GameStatus.END;
                break;
            }

            players.addlast(currPlayer);
        }
    }

    private boolean checkIfCurrPlayerWon(int row, int col, PlayerSymbol playerSymbol) {
        // check if row has all block having playerSymbol
        // check if col has all block having playerSymbol
        // if (row == col) check diagonal and anti-digonal for all block having playerSymbol
    }
}
```

ChatGPT Review

1. Constructor `Board(size)` is private — but used directly in `TicTacToe`. Needs refactoring.
2. Need to have `private static Board instance;` in Board and this instance would be used everywhere
3. checkIfCurrPlayerWon(...) logic ideally belongs to `Board`, not `TicTacToe`
4. Player class methord playMove() is incorrect
    - 4.1 playMove() can't directly WRITE on board
    - 4.2. playMove() should call board.placeSymbol()