Chess LLD

RULES OF GAME :

1. 16 pieces for each player
2. White always move first
3. 8 pawns , 1 king , 1 queen , 2 bishops , 2 rooks , 2 knights
4. Moves for each piece :
   1. King : only one step in any direction
   2. Queen : 'n' steps in any direction
   3. Rook : 'n' steps in vertical/horizontal direction
   4. Bishop : 'n' steps in diagonally
   5. Knight : moves in 'L' shape , two steps straight then one step perpendicularly.
   6. Pawn : only one step(two steps only allowed on first move) , but captures diagonally
5. Special Moves :
   1. Castling : Move involving king and rook
      1. King : moves two squares towards the rook.
      2. Rook : moves over the king to adjacent square.
   2. En Passant : A special pawn capture move
   3. Pawn Promotion : If a pawn reaches to the far end of board , it promotes to any piece (usually queen)
6. Winning the Game :
   1. Check : A king is under threat to be captured
   2. Checkmate : A king is in check and cannot escape - game over.
   3. Stalemate : A player has no legal moves , and the king is not in check - results in draw.

KEY REQUIREMENTS :

1. A 8x8 game board
2. 2 human players
3. Alternating turns between the players
4. Move validation to ensure no illegal moves
5. Detection of check , checkmate or draw

KEY COMPONENTS :

1. Piece : pieces of the game (king,queen,rook,...)
2. Board : 8x8 grid for the board game

3. Player : 2 players who will be making the valid moves on the pieces.

APPROACH :

1. Strategy Pattern : For Pieces Movement - implementing different movement strategies for every different type of piece.
2. Singleton Pattern : For Board - ensuring only one single instance for the chess board.
3. Factory Pattern : For Piece Creation - instantiating chess pieces ensuring the follow the piece interface
4. Manage Game State : use an enum to track the game state.

DESIGN :

Class - Game
```
Board board,
Player player1,
Player player2,
boolean isWhiteTurn,
ArrayList<Move> gameLog,
Status status
```
```
Game(Player player1,Player player2),
void Start()
void makeMove(Move move,Player player)
```

STATUS
```
ACTIVE,
BLACK_WIN,
WHITE_WIN
SAVED,
STALEMATE
```

Interface - BoardGame

Board
```
static Board instance,
Cell[][] board
```
```
Board(int rows)
static Board getInstance(int rows)
void initializeBoard(int rows)
void setPieceRow(int row,boolean isWhite)
void setPawnRow(int row,int rows,boolean isWhite)
```

Move
```
Cell startCell,
Cell endCell
```
```
Move(startCell,endCell)
boolean isValid()
Cell getStartCell()
Cell getEndCell()
```

Player
```
String name,
boolean isWhite
Player(name,isWhite)
```

Cell
```
int row,
int col,
Piece piece,
String label
```
```
Cell(row,col,piece)
Piece getPiece(piece)
void setPiece(piece)
```

PieceFactory
```
static Piece createPiece(String pieceType,boolean isWhite)
```

Piece
```
boolean isWhitePiece,
boolean isKilled,
MovementStrategy movementStrategy
```
```
Piece(isWhitePiece,movementStrategy)
boolean isWhite()
boolean isKilled()
void setKilled(isKilled)
boolean validMove(board,startCell,endCell)
```

Pawn
```
Pawn(boolean isWhite)
```

Rook
```
Rook(boolean isWhite)
```

King
```
King(boolean isWhite)
boolean validMove(board,startCell,endCell)
```

Knight
```
Knight(boolean isWhite)
```

Queen
```
Queen(boolean isWhite)
boolean validMove(board,startCell,endCell)
```

Bishop
```
Bishop(boolean isWhite)
boolean validMove(board,startCell,endCell)
```

KingMovementStrategy
```
boolean validMove(board,startCell,endCell)
```

QueenMovementStrategy
```
boolean validMove(board,startCell,endCell)
```

BishopMovementStrategy
```
boolean validMove(board,startCell,endCell)
```

MovementStrategy
```
boolean validMove(board,startCell,endCell)
```