**Chess Game Documentation**

This documentation provides an overview of the implementation of a Chess game in Java, including the structure, key components, and features of the game.

**Project Overview**

The Chess Game project is a Java-based application that simulates a standard chess game. It allows two players to play chess with all standard rules, including piece movements, check, checkmate, and stalemate conditions.

**Project Structure**

1. **Main Class:** ChessGame
   * Entry point for the application.
   * Manages the overall game loop and user interactions.
2. **Core Classes**:
   * Board: Represents the chessboard as an 8x8 grid.
   * Piece: Abstract base class for all chess pieces.
   * King, Queen, Rook, Bishop, Knight, Pawn: Subclasses of Piece, implementing specific movement logic.
   * Player: Represents a player, tracking their pieces and moves.
   * Move: Represents a move made by a player, including source and destination squares.
3. **Utility Classes**:
   * Position: Encapsulates a specific square on the board (e.g., row and column).
   * GameUtils: Provides utility methods for validating moves, detecting check/checkmate, etc.
4. **Enums**:
   * PieceColor: Enum to represent the two sides, WHITE and BLACK.
   * GameStatus: Enum to represent the game status (ONGOING, CHECK, CHECKMATE, STALEMATE, etc.).

**Class Descriptions**

**1. Board**

* Attributes:
  + Piece[][] squares: 2D array representing the board.
* Methods:
  + initialize(): Sets up the board with all pieces in their starting positions.
  + getPiece(Position pos): Returns the piece at a given position.
  + movePiece(Position from, Position to): Moves a piece and updates the board.

**2. Piece (Abstract Class)**

* Attributes:
  + PieceColor color: Color of the piece (WHITE or BLACK).
  + Position position: Current position of the piece.
* Methods:
  + abstract boolean isValidMove(Position from, Position to, Board board): Validates a piece's movement.
  + List<Position> getPossibleMoves(Board board): Returns a list of valid moves for the piece.

**3. Subclasses of Piece**

* Each subclass (King, Queen, etc.) implements isValidMove() based on the rules of chess.
  + Example:
    - **King**: Can move one square in any direction.
    - **Rook**: Can move any number of squares vertically or horizontally.

**4. Player**

* Attributes:
  + String name: Name of the player.
  + PieceColor color: Color assigned to the player.
* Methods:
  + makeMove(Position from, Position to, Board board): Allows the player to make a move.

**5. Move**

* Attributes:
  + Position from: Starting position of the move.
  + Position to: Ending position of the move.
  + Piece capturedPiece: Captured piece, if any.
* Methods:
  + isCapture(): Returns true if the move results in capturing an opponent's piece.

**6. Position**

* Attributes:
  + int row, int col: Row and column of the position.
* Methods:
  + isValid(): Validates if the position is within the board.

**Game Flow**

1. **Initialization**:
   * Create the board and initialize pieces.
   * Create two players and assign colors.
2. **Game Loop**:
   * Alternate turns between players.
   * Display the board and prompt the current player for their move.
   * Validate the move and update the board.
   * Check for game-ending conditions (e.g., checkmate, stalemate).
3. **Game End**:
   * Announce the winner or declare a draw.

**Key Features**

1. **Piece Movement**:
   * Implements standard chess rules for each piece.
   * Validates moves, including capturing, castling, en passant, and pawn promotion.
2. **Check and Checkmate**:
   * Detects when a king is in check or checkmate.
3. **Stalemate**:
   * Detects stalemate conditions.
4. **Undo Move** (Optional):
   * Allows players to undo their last move.
5. **Graphical Interface** (Optional):
   * Can be implemented using JavaFX or Swing for a more interactive experience.

**Future Enhancements**

1. **AI Player:**
   * **Implement a better AI opponent using algorithms like Minimax. (to shorten the time period)**
2. **Animation:**
   * **Make Pick and Drop animation to the pieces.**
3. **Multiplayer Mode:**
   * **Enable online multiplayer using sockets or REST APIs.**
4. **Graphical User Interface (GUI):**
   * **Create a user-friendly interface using JavaFX or Swing.**