# JChess Project Description

Cojocaru Daniela

## Overview

This project implements a chess game simulation in Java, featuring core components such as players, a board, pieces, and game logic. It includes classes to represent the chessboard, game state, and chess pieces, adhering to object-oriented principles.

## Objectives

1. **Learn and Apply Object-Oriented Principles:** The project aims to enhance understanding and practical application of core OOP concepts like inheritance, polymorphism, encapsulation, and abstraction.
2. **Simulate a Chess Game:** Provide a functional representation of chess, allowing users to play a game with all basic rules implemented against a chess engine (Stockfish).
3. **Build a Foundation for Advanced Features:** Create a flexible design that can be extended with features like advanced rules (castling, en passant, promotion), AI opponents (using Stockfish), and an interactive graphical user interface (GUI) with Swing.
4. **Foster Problem-Solving and Algorithm Design Skills:** Develop logic to validate moves, detect game states (check, checkmate, stalemate, 50-Moves Rule), and manage player interaction.
5. **Encourage Code Modularity and Reusability:** Design components that are modular, testable, and reusable for potential expansion or integration into larger systems.

## Classes and Responsibilities

### 1. Player

**Package:** `elements`
Represents a chess player.
**Attributes:**

- `Game.Color playerColor`: The player's color (WHITE or BLACK).
- `ColorOption colorOption`: The player's color choice (WHITE, BLACK, or RANDOM).

**Methods:**

- `Player(ColorOption colorOption)`: Constructs a player with the specified color option. If the color option is RANDOM, the player is randomly assigned a color.
- `ColorOption getColorOption()`: Returns the player's color choice.
- `Player copy()`: Creates a copy of the current player with the same color and color option.

## 2. Square

**Package:** `elements`
Represents a single square on the chessboard.

**Attributes:**

- `Game.Color color`: The color of the square (WHITE or BLACK).
- `int rank`: The row index of the square.
- `char file`: The column index of the square.
- `boolean isEmpty`: Indicates whether the square is empty.
- `Piece piece`: The chess piece occupying the square (if any).

**Methods:**

- `Game.Color getColor()`: Returns the color of the square.
- `int getRank()`: Returns the rank (row) of the square.
- `char getFile()`: Returns the file (column) of the square.
- `boolean getIsEmpty()`: Returns whether the square is empty.
- `Piece getPiece()`: Returns the piece occupying the square, if any.
- `void setColor(Game.Color color)`: Sets the color of the square.
- `void setRank(int rank)`: Sets the rank (row) of the square.
- `void setFile(char file)`: Sets the file (column) of the square.
- `void setIsEmpty(boolean empty)`: Sets whether the square is empty.
- `void setPiece(Piece piece)`: Sets the piece occupying the square.

- `Square copy()`: Creates a copy of the current square, including all its properties and piece (if any).
- `String toString()`: Returns a string representation of the square's position (e.g., "a1", "h8").

## 3. Position

**Package:** `elements`
Represents the board state and position of pieces.

**Attributes:**

- `Square[][] board`: A 2D array of squares representing the board.
- `int positionNumber`: The position's identifier.
- `boolean whiteAllowedCastle`: Indicates if white can castle.
- `boolean blackAllowedCastle`: Indicates if black can castle.

**Methods:**

- `Position()`: Initializes the board and sets the starting position.
- `startPosition()`: Arranges pieces in their initial positions.
- `String toStringRank(int i, int j, Game game)`: Returns a string representation of the rank (row) and file (column) of a given square, including the piece occupying the square.
- `String toString(Game game)`: Returns a string representation of the entire board, displaying each square with its piece and color.

## 4. Move

**Package:** `elements`
Represents a move in a chess game, including details such as the start and end squares, the piece moved, move notation, and the resulting position after the move.

**Attributes:**

- `Square start`: The square from which the move starts.
- `Square end`: The square to which the move ends.
- `int moveNumber`: The number of this move in the sequence of the game.
- `Piece movedPiece`: The chess piece that is moved during this move.
- `String moveNotation`: The chess notation representing this move (e.g., "e4", "Nxf3").

- `Position positionAfterMove`: The board position after this move is executed.

**Methods:**

- `Move(Square start, Square end, int moveNumber, Piece movedPiece, String moveNotation, Position positionAfterMove)`: Initializes a new move with the provided details.
- `Square getStart()`: Returns the starting square of the move.
- `Square getEnd()`: Returns the ending square of the move.
- `int getMoveNumber()`: Returns the move number in the game's sequence.
- `Piece getMovedPiece()`: Returns the piece involved in the move.
- `String getMoveNotation()`: Returns the move in standard chess notation.
- `Position getPositionAfterMove()`: Returns the board position after this move is executed.
- `String toString(Game game)`: Converts the move to a string, including the resulting board state, move number, and notation.
- `Move copy()`: Creates and returns a deep copy of the move, including the associated pieces and board position.

## 5. Piece (Abstract)

**Package:** `Project.src`
The base class for all chess pieces.

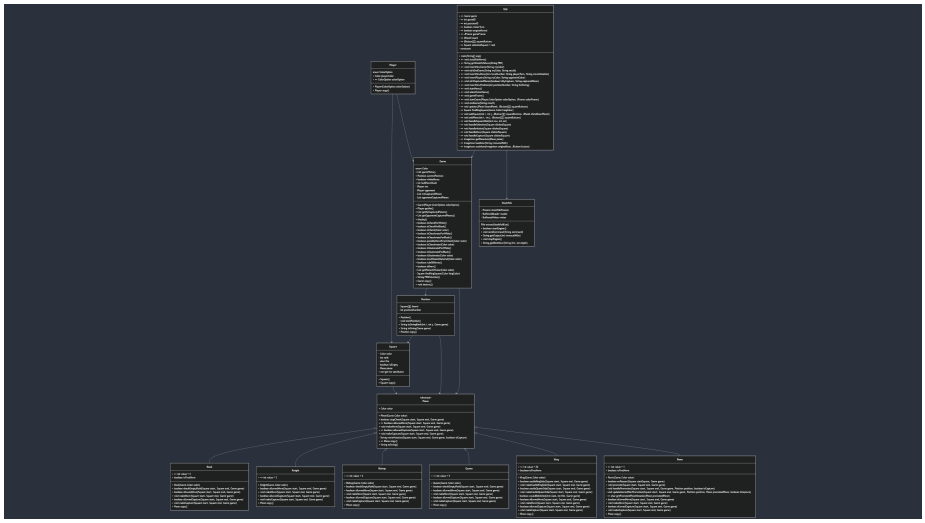
**Attributes:**

- `Game.Color color`: The piece's color (either white or black).
- Various constants for each piece type (e.g., `String WHITE_PAWN`).

**Methods:**

- `Piece(Game.Color color)`: Constructs a Piece with the specified color.
- `boolean allowedMove(Square start, Square end, Game game)`: Abstract method to determine if a move is allowed based on the piece's movement rules.
- `void makeMove(Square start, Square end, Game game)`: Makes a move from the start square to the end square if the move is valid and does not put the player in check.
- `String toString()`: Returns the string representation of the piece (e.g., "P" for white pawn, "r" for black rook).

## Class Diagram



## How to Run

1. Clone the repository.
2. Build the project using your preferred IDE.
3. Run the `App.main()` method to start the game.
4. Follow the console prompts to play.

## Future Improvements

- Develop a GUI to enhance the user experience, including panels for user icons, the user's and engine's captured pieces, game history (moves and their numbers), and a feature to display all possible squares where the selected piece can move.
- Improve the app's performance for faster gameplay.
- Develop an online multiplayer server where users can play against each other, with a rating system.
- Implement difficulty levels for the AI chess engine based on the depth of the Mini-Max algorithm, as used in Stockfish.

## Acknowledgments

This project is built with Java and adheres to standard object-oriented principles for modeling chess games.