The program is designed only for white player

There are seven files in this project

Chess.java 🡪 This is main class that handles entire program. It generates possible positions of piece given by user.

Rook.java 🡪 This file returns array list of all the possible positions of rook after providing initial position.

King.java 🡪 This file returns array list of all the possible positions of king after providing initial position.

Queen.java 🡪 This file returns array list of all the possible positions of queen after providing initial position.

Pawn.java 🡪 This file returns array list of all the possible positions of pawn after providing initial position.

Bishop.java 🡪 This file returns array list of all the possible positions of bishop after providing initial position.

Knight.java 🡪 This file returns array list of all the possible positions of knight after providing initial position.

There are two global HashMaps in chess file

1. possiblePositions 🡪 This hashMap stores the list of all the possible moves for any given piece after initializing its start position.
2. occupiedPosition 🡪 When user selects the final position for any piece it will be stored in occupiedPositions

The programs works as follow:

1. User enters the piece name from Rook, King, Queen, Pawn, Bishop and Knight
2. User enter the current position of piece.
3. The program checks if the current positions is already occupied and if yes then asks again for user input
4. After entering the correct position, it calls the respective java file for that piece and gets all possible positions in posiiblePositions.
5. After getting positions it checks if value is already present in occuipiedPosition. If the value is present, then it will remove that value from possible positions.
6. The program will display possible positions in console.
7. User selects integer value for respective possible position. The selected possible position is added in occupied position and displayed on the console.
8. The program again asks for new piece if user wish to continue.

Pawn.java 🡪

For pawn possible positions program also considers the pawn capture positions.

The string input is spitted in two variables row and column.

If row is 2 pawn has 4 possible moves

If row is between 3 to 7 pawn has 3 possible

If row is 8 then no moves

Row and column values are incremented or decremented as per need to store possible positions

Rook.java 🡪

There are 4 for loops in this file

Row incremented from 1 to 8 Column is incremented from a to h to store possible positions

Bishop.java 🡪

There 2 for loops one to generate upper position from current position and one to generate below positions from current positions

Queen.java 🡪

Queen merges the for loops used in bishop and rook to generate all the possible positions

Knight.java 🡪

Row and column values are incremented or decremented as per knight moves in chess to store possible positions.

There are 8 possible values for knight for given position.

The program generate all 8 values for every position

Before storing the value in arraylist the program check if output generated satisfies the following condition

a< column <h

1 < row < 8

If condition is satisfied then only it will store the value in list

King.java 🡪

Row and column values are incremented or decremented as per king moves in chess to store possible positions.

There are 8 possible values for king for given position.

Before storing the value in arraylist the program check if output generated satisfies the following condition

a< column <h

1 < row < 8

If condition is satisfied, then only it will store the value in list

**Piece restrictions beyond filled position:**

For Rook, Bishop and Queen if there is any other piece in between possible moves then it cannot move beyond that position even if they are not occupied

Rook Logic 🡪

The program created 4 local variables col\_left, col\_right, row\_up and row\_down to store the nearest occupied row and column to restrict the movement of piece beyond that position

Bishop Logic 🡪

1. Created a local hashmap which stores the all possible generated moves.
2. First we check if hashmap contains positions that are already filled. If that’s the case then we perform the below operation
3. We have four diagonal directions where bishop can move therefore created four for loops for each direction.
4. The program removes the positions from arraylist beyond the filled position

Queen Login 🡪

As queen can move in both rook and bishop’s direction then program use same logic as bishop and rook to remove the possible positions beyond filled positions.