Documentation

A) Printing Board

The main problem of the printing board for me the logic behind the board. First time I didn't know how to make the chess board. Then I found out that I can use matrix for the board. I opened a 8x8 matrix named Square[][] squares variable named squares. The main challenge here finding the way to construct the board so then I started to write main columns and rows with symbols. Then for placing the chars I used switch() function it was so easy to construct the board with these steps. For initializing board we need to use for loop for each square so I opened a for loop inside a another for loop looping all of the board and creating squares. Then I manually added the pieces to the board. The first I created my board it was rectangular. So I re-organized my board later finished the all of the codes.

B) Defining board and Square Class

The squares class includes rows and columns with these rows and columns we create our matrix in ChessBoard class like Square[][] squares; Our board will include 64 squares so the inside of the Square will be Square[8][8].

C) Implementing methods of Board and Square Classes

```
public boolean isWhitePlaying()
```

- Defining the which player will start and when the player moves the jumping the next player we are using this method for this.
- Returns whitePlaying variable if its true white will play if its not black will play.
- Returned white playing variable.

```
public boolean isGameEnded()
```

- Game will end when other player runs out the pieces. This method counts the pieces and in return if there is no pieces left from the enemy other side wins.
- Return Black wins or White wins printed outlines.

- Opened two counter for black and white pieces. Then I opened two for loops to check all of the squares in the board I used if statement to find the remaining pieces and added one by one in the counter. When one of the counters reach zero then program will print the winners name.

```
public int charToInt(char to)
```

- Defining the chars to integer numbers. This method places the right side and left side the numbers for the main board.
- char to: changing the char with to statement, switching them.
- Used switch function with cases. Used random chars with our 0-7 integers for board.
- Returns unnecessary number to itself.

```
public void nextPlayer()
```

- When player makes a move the next the player will play.
- Turns white player to black.
- It is a void method returns nothing.

```
private void initializeBoard()
```

- Initializing the board with rows and columns supported by two loops every square now has columns and rows appointed to them which includes manually settled locations of pieces. For pawns used a different loop.
- Constructed loop inside a loop that every row and column created the new squares inside our board. Inside matrix manually replaced special pieces. Pawns replaced with a for loop that takes same row but different columns.
- It is a void method returns nothing.

```
private boolean isValidPosition(int row, int col) {
```

- Checking the users input that inside borders of the board.
- int row: rows inside the board that keeps the information of the piece locations. Int col: columns inside the board that keeps the information of the piece locations.

- Returns rows and columns are whether bigger than the size of the board if they are bigger than the board this boolean will turn false if they are not it will return true.
- Basic return statement with and statements and some basic math done.

```
public Square[] getSquaresBetween(Square location,Square targetLocation)
```

- Returns the squares between given squares if they are at the same row or same column or same diagonal
- s1: the beginning square ; s2: final square
- Array of squares as the same order from s1 square to s2 square not including s1 and s2, null is returned if no squares are in between
- First, whether the given squares are at the same row or same column or same diagonal is checked. If this is the case, based on the distances of squares an array is created. Arrays element are assigned with the squares by iterating from s1 Square to s2 Square.

```
public String toString()
```

- Building the main board for visuality.
- Recursion function returns itself
- None parameter
- Creating new string builder and using loops with using string builder that builds the entire board with symbols and the chars appending to it.

```
public Square(int row, int column)
```

- Main constructor for Square class.

```
public boolean isEmpty()
```

- Checking the piece if its null or not null
- Returns the piece variable.

```
public void putNewQueen(int color)
```

```
public void putNewQueen(int color)
```

- If the pawn comes to the last row it becomes a brand new queen.
- int color: color of the piece whether its black or white.
- Creating a new queen with Queen class to the pawn.

```
public void clear()
```

- Making piece null.
- Whenever a piece eaten it will become a null piece with this method.

```
public boolean isAtLastRow(int color)
```

- Cheking the piece is at last row or not if it is needed.
- int color: color of the piece whether its black or white.
- Returns it's the last row or not the last row.
- It is needed for the when a pawn reaches the last row and wanted to become a queen.

```
public String toString()
```

- Turning in to visuality of a square if it's a blank square it will stay as a blank square but if it is not it will be passed trough a Piece classes toString() method.
- Return if it is empty square then will print a blank space if its not there will be a pieces first letter.

```
public boolean isAtSameColumn(Square otherSquare)
```

- Method wrote for check that the two squares in the same column or not.
- Square otherSquare: the other square that we wanted to control.
- Return other squares column.

```
public boolean isAtSameRow(Square otherSquare)
```

- Method wrote for check that the two squares in the same column or not.
- Square otherSquare: the other square that we wanted to control.
- Return other squares row.

```
public boolean isNeighborColumn(Square otherSquare)
```

- Method wrote for check that the two squares is neighbor column or not.
- Return column distance is equal to one or not.
- Square otherSquare: the other square that we wanted to check.

```
public boolean isInDiagonalWith(Square otherSquare)
```

- Method that checks that the two squares is in the same diagonal or not.
- Returns if row diff and column diff is equal or not.
- Square otherSquare: the other square that we wanted to check.
- Taking row and column differences of the first square and the second square then we will check if they are diagonal or not.

```
public int getRowDistance(Square otherSquare)
```

- Calculating the row difference between our first square and second square.
- Square otherSquare: the other square for calculate the value between the rows.
- Returns the row difference between squares.

```
public int getColDistance(Square otherSquare)
```

- Calculating the column difference between our first square and second square.
- Square otherSquare: the other square for calculate the value between the columns.
- Returns the column difference between squares.

D) Defining Piece Hierarchy

- Main class gathers all of the classes and uses them together with polymorphism all of the classes are connected with each other. All of the pieces in the board connected to the piece class which is a abstract class and the Piece class also in a aggregation with the Square class that has a composition with ChessBoard class like we can see in the class diagram that saved my life for this work.
- canMove, move and toString methods defined as a abstract method for using them later in piece classes that will extends the Piece class.
- Yes, there is a big code reuse in my implementation.

E) Implementing methods in Piece Hierarchy

- All of the pieces has a same constructor that has int color and Square location.

-King Class that extends Piece class

```
public boolean canMove(String to)
```

- This method decides the movement is valid or not and the borders that the king can move, king only can move the one of the neighbour square. If that square occupied by his own colour then he can not take the piece or jump through it but if that square holding a enemy piece than king can eat it easily.
- Returns validMove variable which decides the move can be valid or not (it contains true false statement)
- String to: Using to as a parameter helps us with getting the target location square for us.
- First we need to calculate the row and column distance between the two squares and then bordering our king's movement space with if statement can be say us that is a valid move or not a valid move for the black and white squares.

```
public void move(String to)
```

- Basic method for move the king from square to square. Same as the other piece classes will be executed if the canMove class is true. Setting location to target location.

- Using square target location variable and assigning the new location.
- String to: to is the target square as a parameter.

```
public String toString()
```

- Visuality of king for white and black side.
- Returns the colour as string if its white it will be K char else for black k char.

-Knight Class that extends Piece Class

```
public boolean canMove(String to){
```

- Determining the knight is valid to move or not and basic movement code for the knight like king.
- Returns validMove variable which decides the move can be valid or not (it contains true false statement)
- String to: Using to as a parameter helps us with getting the target location square for us.
- First of all calculating the movement of knight with column and row distances then using them inside a if statement for deciding the movement is valid or not also it prevents to eat his own colour pieces.

```
public void move(String to){
```

- Basic method for move the king from square to square. Same as the other piece classes will be executed if the canMove class is true. Setting location to target location.
- Using square target location variable and assigning the new location.
- String to: to is the target square as a parameter.

```
public String toString()
```

- Visuality of knight for white and black side.
- Returns the colour as string if its white it will be N char else for black n char.

-Bishop Class that extends Piece Class

```
public boolean canMove(String to)
```

- Finding out that the bishop is valid to move or not as other pieces it has a special case. The bishop can only move diagonal. In this method we check if the input is diagonal or not diagonal and if it is valid to move let it return true.
- Returns validMove variable which decides the move can be valid or not (it contains true false statement)
- String to: Using to as a parameter helps us with getting the target location square for us.
- Firstly we calculate the row and column distance again then we check with if statement are they equal to each other for diagonality if they are diagonal we check the target square if its null or other enemy piece in that square than the bishop can move to that square but if there is a teammate piece in that square it is not a valid move so it returns false.

```
public void move(String to)
```

- Basic method for move the king from square to square. Same as the other piece classes will be executed if the canMove class is true. Setting location to target location.
- Using square target location variable and assigning the new location.
- String to: to is the target square as a parameter.

```
public String toString()
```

- Visuality of bishop for white and black side.
- Returns the colour as string if its white it will be B char else for black B char.

-Queen Class that extends Piece Class

```
public boolean canMove(String to)
```

- Calculating the queen can make a valid move or can't do. The Queen can move diagonal, horizontal and vertical but can't jump through other pieces. So we need to calculate the movement between the squares and decide it's a valid move or not.
- Returns validMove variable which decides the move can be valid or not (it contains true false statement)
- String to: Using to as a parameter helps us with getting the target location square for us.
- First of all we calculate the row and col distance again then check they are equal or one of the distances must be zero to valid movements in a if statement. Then we calculate the squares between for a valid move with a method callet getSquaresBetween if target location of the piece is null or has a enemy piece we can say the queen can make a valid move but if the target location holds a same colour as the queen than we can say it is a invalid movement.

```
public void move(String to)
```

- Basic method for move the king from square to square. Same as the other piece classes will be executed if the canMove class is true. Setting location to target location.
- Using square target location variable and assigning the new location.
- String to: to is the target square as a parameter.

```
public String toString()
```

- Visuality of bishop for white and black side.
- Returns the colour as string if its white it will be Q char else for black q char.

-Rook Class that extends Piece Class

```
public boolean canMove(String to)
```

- Determine the rook can make a valid move or not. Rook can move only horizontally or vertically. Calculating the between squares of the target location and standing location of the rook. Can't jump through the pieces too.
- Returns validMove variable which decides the move can be valid or not (it contains true false statement)
- String to: Using to as a parameter helps us with getting the target location square for us.
- We need to calculate the row and column distances first. Then if row or column distance is zero in a if statement then we can check the squares between are null or not and also the target location. If the target location is null or a same colour as the rook it is a valid move but if it is not a null square and a same colour piece holds the square then rook can't move.

```
public void move(String to
```

- Basic method for move the king from square to square. Same as the other piece classes will be executed if the canMove class is true. Setting location to target location.
- Using square target location variable and assigning the new location.
- String to: to is the target square as a parameter.

```
public String toString()
```

- Visuality of bishop for white and black side.
- Returns the colour as string if its white it will be R char else for black r char.