1. **Printing the board:**

To print the board each Piece has a toString() method that returns a character that represents itself (P for Pawn for example). Then I’ve created 64 Square objects and some of them have Piece objects in them. To print the board I’ve made a method called printBoard() and the first thing that I printed is the coordinate character on the top; then printed – for separating rows. After that I’ve made a loop that prints the row number and all the Square objects. At the last row I’ve printed the coordinate characters again.

1. **Defining Board and Square Classes:**

The relation between Chessboard and Square objects is that the Chessboard object have 64 Square objects in an array and each square is constructed with a coordinate, and then the Piece of the squares are set. Each square is connected to the chessboard when constructing by putting “this” on the constructor where the board is specified.

1. **Implementing methods of Board and Square classes:**

**Board Methods:**

public Chessboard(){

This is the constructor for the chessboard class

It uses the initialize method down below.

public void printBoard(){

Prints the chessboard.

Doesn’t return anything.

First, I’ve created some strings for the coordinate chars and lines and printed them, after that I’ve made a loop that prints each Square.toString()

public void initialize(){,

Initializes the chessboard

Void

First of all, it creates 64 Square objects and puts them in an array, then it sets a Piece object in the corresponding Squares (such as setting white rook at A1)

public boolean isGameEnded(){

Checks to see if the game is ended if there are no pieces that exists from one color.

Returns the inverse of whiteFound or blackFound which means if there’s black and white then its false; if there is only white or black, it’s true.

It checks all of the squares in the array and if the squares have a white piece or black piece it will mark the found flags and returns them correspondingly.

public Piece getPieceAt(String location)

Returns the piece at the specified location.

Location: A string of a location of a Square

Piece object of the specified location is returned

First the column and row info are declared. Then if the row and column info is out of range and exception is thrown. At last, the piece is returned from the column and row.

public Square getSquareAt(String location){

Does the exact same thing as above but returns the Square instead.

public Square[] getRowSquaresBetween(Square s1, Square s2){

Returns the squares between given squares if they are at the same column.

S1: 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 each other.

A Square array is created based on the distances and the array is filled with squares that are between s1 and s2 with a loop.

public Square[] getColSquaresBetween(Square loc1,Square loc2)

Returns the squares between given squares if they are at the same row.

loc1: Beginning Square; loc2: 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 each other.

A Square array is created based on the distances and the array is filled with squares that are between s1 and s2 with a loop.

public Square[] getDiagSquaresBetween(Square loc1, Square loc2)

Returns the squares between given squares if they are at the same diagonal line.

loc1: Beginning Square ; loc2: 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 each other.

First if the squares get checked to see if they’re at same diagonal line. Then a Square array is created based on the distances and the array is filled with squares that are between s1 and s2 with a loop.

public void nextPlayer() {

Changes the player to the next color.

It inverses the Boolean whitePlaying from true to false or vice versa.

**Square Methods:**

public Square(int row,char column,Chessboard board){

This is constructor for Square class

It initializes row column and board variables.

public boolean isAtLastRow(int color){

Checks to see if a piece is at last row according to its color and location.

Color: the color of piece. 0 for white 1 for black (also set as a final variable in chessboard)

Returns True if the piece is white and its at 8th row or if the piece is black it’s at 1st row.

public int getRowDistance(Square s){

Returns the row distance between two square objects.

Square s: second Square object; this: first Square object

It returns the distance as an integer.

public int getColDistance(Square s){

Returns the column distance between two square objects.

Square s: second Square object; this: first Square object

It returns the distance as an integer.

public boolean isAtSameColumn(Square s){

Checks if both squares are at the same column.

Square s: second Square object; this: first Square object

Returns true if they are at same column.

Uses the accessor methods to compare the squares column locations.

public boolean isAtSameRow(Square s) {

Checks if both squares are at the same row.

Square s: second Square object; this: first Square object

Returns true if they are at same row.

Uses the accessor methods to compare the squares row locations.

public boolean isAtSameDiagonal(Square targetLocation) {

Checks if both squares at same diagonal.

Square targetLocation: second Square object; this: first Square object

If the absolute value of this.getRow() - targetLocation’s row is same as abs. value of this’s column -targetLocation’s column equal then it will return true.

public boolean isNeighborColumn(Square target){

Checks if both square are next to each other.

Square target: second Square object; this: first Square object

Returns true if abs value of (column of this) – (column of target) = 1

public boolean isEmpty(){

Checks to see if a square is empty or not.

Returns true if the Piece == null

public String toString(){

Returns the piece.toString() if the piece is not null. If its null then it returns “ ”

public void putNewQueen(int color) {

puts a new queen at this square

Color: the color of piece. 0 for white 1 for black (also set as a final variable in chessboard)

public void clear() {

clears the piece in the square

1. **Defining Piece Hierarchy**

The Main class uses polymorphism to define only one loop for moving pieces around the board since all the Piece object have canMove and move methods.

The Piece class can be defined abstract and the canMove and move methods can be defined abstract.

There’s code reuse in a lot of methods such as, the getColor method, move method, canMove methods. For example, queen uses bishop’s canMove for diagonal and rook’s canMove for horizontal and vertical movements.

1. **Implementing methods in Piece Hierarchy**

**Piece class:**

public abstract boolean canMove(String loc);

Checks if the piece can move to the specified location

String loc: the location to check if the piece can move there

Returns true if the piece can move there.

This method has different implementations for different piece variants.

public void move(String to){

Moves the piece to the specified location

String loc: the location to move the piece

First it gets the specified locations square object and it sets the piece to that location and it clears the previous location.

public boolean canItMoveDiagonally(String to){

Checks to see if the piece can move diagonally to the desired position

String to: the location to check if the piece can move there

Returns true if it can move there

This function is used on bishop and queen pieces. First it checks if the locations are in the same diagonal line, then it checks the color of piece. After that it checks the desired location to see if its empty or there’s an opposing color piece is there. If that’s the case then it checks the squares between them to see if they’re empty. If that’s the case it will return true.

public boolean canItMoveHorizontally(String to){

Checks to see if the piece can move horizontally to the desired position

String to: the location to check if the piece can move there

Returns true if it can move there

This function is used on rook and queen pieces. First it checks if the locations are in the same row, then it checks the color of piece. After that it checks the desired location to see if its empty or there’s an opposing color piece is there. If that’s the case then it checks the squares between them to see if they’re empty. If that’s the case it will return true.

public boolean canItMoveVertically(String to){

Checks to see if the piece can move vertically to the desired position

String to: the location to check if the piece can move there

Returns true if it can move there

This function is used on rook and queen pieces. First it checks if the locations are in the same column, then it checks the color of piece. After that it checks the desired location to see if its empty or there’s an opposing color piece is there. If that’s the case then it checks the squares between them to see if they’re empty. If that’s the case it will return true.

**Bishop Class:**

public Bishop(int color, Square location) {

This is the constructor for the bishop, it defines the color of piece and the location.

public boolean canMove(String to){

this directly returns the canItMoveDiagonally() method above

public String toString() {

Returns B if its white, b if its black.

**King Class:**

public King(int color,Square location) {

This is the constructor for the king, it defines the color of piece and the location.

public boolean canMove(String loc) {

Checks if the king can move to the desired place.

String loc: the location to check

First it gets the column and row distance of the locations and if the distances are less and equal to 1 and if the place is empty or there is an opposing color piece there it will return true.

public String toString() {

Returns K if its white, k if its black.

**Knight Class:**

public Knight(int color, Square location) {

This is the constructor for the knight, it defines the color of piece and the location.

public boolean canMove(String loc) {

Checks if the knight can move to the desired place.

String loc: the location to check

If the desired locations column distance is 2 and row distance is 1 or vice versa and the desired location is empty or has an opposing color piece it will return true.

public String toString() {

Returns N if its white, n if its black.

**Pawn Class:**

public Pawn(int color, Square location) {

This is the constructor for the pawn, it defines the color of piece and the location.

public boolean canMove(String to) {

Checks if the pawn can move to the desired place.

String to: the location to check

First it checks if the target location is at the same column

Then if the row distance is 2 it will check if the pawn at its initial location. If its true then it will check the targetlocation and the square between initial and target is empty. If its false then it will only check the target location and it will return true if it’s the case.

Also, if the target is neighbor to initial position, it will check if the row distance is white and the target’s not empty and it has an opposing color piece there. If that’s the case it will return true.

public void move(String to) {

Pawn class overrides the move method in Piece class because if the pawn reaches last row, it will be promoted to queen. Other than that, it’s the same as Piece class.

public String toString() {

Returns P if its white, p if its black.

**Queen Class:**

public Queen(int color, Square location) {

This is the constructor for the queen, it defines the color of piece and the location.

public boolean canMove(String to) {

Returns true if either canItMoveDiag or canItMoveHorizontally or canItMoveVertically is true

public String toString() {

Returns Q if its white, q if its black.

**Rook Class:**

public Rook(int color, Square location) {

This is the constructor for the rook, it defines the color of piece and the location.

public boolean canMove(String to) {

Returns true if either canItMoveHorizontally or canItMoveVertically is true

public String toString() {

Returns R if its white, r if its black.