**C:**

1. **Creating a piece** - complete, constructor - AA
2. **Move a piece**, validate movement - complete, moveTo method - AA
3. **Status for piece** - complete, get/set status for alive pieces. - AA
4. **Special movement rules** - complete, getRange method - AO

**B:**

1. **Implement abstraction** - complete - CC
2. **Set pieces method** **- unsure, need instructor**, this needs to be done in main
3. **Check if king is in check** - complete, kingCheck method - CC

**A:**

1. **Prevent the king from moving in to check** - complete, moveTo method - AA/CC
2. **King needs to be taken out of check** - **unsure, need instructor**
3. **En passant** - incomplete -
4. **Castling** - incomplete -

**Main() instructions:**

-each piece needs to be created in main using the constructors with correct parameters.

(x coordinate, y coordinate, piece type, isWhite)

Ie: Rook wRook1 = new Rook(0,0,"Rook",true);

-An Arraylist of pieces must be defined and populated in main, this is crucial for many of our functions. *Please also remove pieces from this list if they die* as their range and coordinates will affect the other pieces.

Ie:

ArrayList<Piece> pieces = new ArrayList<Piece>();

pieces.add(wRook1);

Piece overview

Variables

private int X; - x coordinate

private int Y;- y coordinate

private String type;- specifies type of piece

private Boolean isAlive;- true for alive, false for dead

private Boolean isWhite;- true for white, false for black

public ArrayList<int[]> range = new ArrayList<>();- an array list containing the pieces range, this is used to calculate valid moves and to see if the king is within striking distance.

Each of these variables also has corresponding get/set functions

**GetRange**:

Parameters - arraylist<Piece>

Description - this is different for each piece, this will step through each coordinate the piece can go to and add it to the range variable if applicable.

Output - updates the range for the piece given its current coordinates

**updateRange:**

Parameters - arraylist<Piece>

Description - for each piece in the array list, run getRange();

Output - updates all ranges for all pieces in the passed arraylist

**GetPiece**:

Parameters - int X, int Y, arraylist<Piece>

Description - checks the passed X and Y against each piece in the arraylist to see if there is already a piece there.

Output - 0 for none, 1 for a piece matching color, 2 for piece with opposing color

**King Check:**

Parameters - arraylist<Piece>

Description - checks the range of each piece in the passed array list to see id the king is in the range of a piece with an opposing color

Output - True if the king is in check, False if it is not.

**moveTo**: (this one is a bit complicated, call me if you have questions)

Parameters - int x, int y, king MyKing, arraylist<Piece>

Description -

save current coordinates to myCoords > update range > if passed x and y are in range, set x and y > update range > check to see if king is in check > if yes, restore coordinates (move piece back), else the function ends.

Output - nothing, just changes the pieces x and y coordinates to the ones passed as parameters if applicable.

**Print:**

Prints everything but the range, not used in code but made it for testing.