Report Lab 1 - Verification of pieces using functional decomposition

1. **Requirements**

The program must read a file that contains chess pieces and its data: type, color, position x and position y. It must store these pieces in an object or structure and store them in an array. The program asks the user for a new position and checks if the pieces could move to this new position. The program should write “Piece\_Name at X, Y can move to X+1, Y” or “Piece\_Name at X, Y can NOT move to X+1, Y”.

1. **Constraints**

Only one class can be created for the pieces, not one for each.

Cannot use inheritance.

The places for the x position are A-H and for the y position 1-8 because a chess board is 8 by 8. There are six different pieces: Pawn, Rook, Knight, Bishop, Queen, and King., with its own movement rules.

The movement rules for each chess piece follows:

* King can move exactly one square horizontally, vertically, or diagonally.
* Queen can move any number of vacant squares diagonally, horizontally, or vertically.
* Tower can move any number of vacant squares vertically or horizontally.
* Bishop can move any number of vacant squares in any diagonal direction.
* Knight can move one square along any rank or file and then at an angle. The knight’s movement can also be viewed as an “L” or “7″ laid out at any horizontal or vertical angle.
* Pawns can move forward one square, if that square is unoccupied. If it has not yet moved, the pawn has the option of moving two squares forward provided both squares in front of the pawn are unoccupied. A pawn cannot move backward.

1. **High-level approach (algorithm)**
   1. Open file to read
   2. Iterate every line of the file
   3. Create a piece object with the information on the line (corresponding to type, color, and position x and y)
   4. Add the piece object to an array of Pieces
   5. Print the pieces that were read
   6. Prompt the user for a new position to move to
   7. Check if it is a valid position in the chess board, not outside boundaries
   8. Look for the type of the piece to call the appropriate method to check if the move is valid
   9. Check if the move is valid for each piece in the array
   10. Print if the piece could move to the new position for each piece in the array
   11. Ask the user if he wants to test another piece or exit, if he does, return to step 6
2. **Pseudo code for each major function**

**PAWN(color, posX, posY, origX, origY)**

* 1. BEGIN
  2. INIT valid = true
  3. IF(origx != posX && Math.abs(origX-posX) != 1 || (Math.abs(origX-posX) == 1 && Math.abs(posY-origY) > 1))
  4. valid = false
  5. IF (color.equals("white"))
  6. IF (posY-origY > 2 || (origY != 2 && posY-origY > 1) || posY-origY <= 0)
  7. valid = false
  8. IF(color.equals("black"))
  9. IF (origY-posY > 2 || (origY != 7 && origY-posY > 1) || origY-posY <= 0)
  10. valid = false
  11. RETURN valid
  12. END

**KNIGHT(color, posX, posY, origX, origY)**

1. BEGIN
2. INITIALIZE valid
3. valid = (Math.abs(posY-origY) == 2 && Math.abs(origX-posX) == 1)||(Math.abs(posY-origY) == 1 && Math.abs(origX-posX) == 2)
4. RETUTN valid
5. END

**BISHOP(color, posX, posY, origX, origY)**

1. BEGIN
2. INITIALIZE valid
3. valid = Math.abs(posY-origY) == Math.abs(origX-posX) && Math.abs(posY-origY)>0
4. RETURN valid
5. END

**ROOK(color, posX, posY, origX, origY)**

1. BEGIN
2. INITIALIZE valid
3. valid = (posY == origY) != (posX == origX)
4. RETURN valid
5. END

**QUEEN(color, posX, posY, origX, origY)**

1. BEGIN
2. INITIALIZE valid
3. valid = (Math.abs(posY-origY) == Math.abs(origX-posX) && Math.abs(posY-origY)>0) || (posY == origY) != (posX == origX)
4. RETURN valid
5. END

**KING(color, posX, posY, origX, origY)**

1. BEGIN
2. INITIALIZE valid
3. valid = Math.abs(posY-origY) <= 1 && Math.abs(origX-posX) <= 1 && Math.abs(posY-origY)+ Math.abs(origX-posX)>0
4. RETURN valid
5. END
6. **List the major parts of your solution (class or methods), show them in a diagram, show their relations**
7. **MAIN LAB CLASS**
   1. **readPieces – reads file into pieces array**
   2. **countLines – counts the lines in the file**
   3. **makePiece – creates a piece from a string of information**
   4. **validate – checks the new positions for all pieces in the array**
   5. **validatePawn – checks moves for a pawn**
   6. **validateKnight – checks moves for a knight**
   7. **validateBishop – checks moves for a bishop**
   8. **validateRook – checks moves for a rook**
   9. **validateQueen – checks moves for a queen**
   10. **validateKing – checks moves for a king**
   11. **main – calls the above methods to execute the program**
8. **PIECE CLASS**
   1. **Attributes**
      1. **type**
      2. **color**
      3. **posX**
      4. **posY**
   2. **Constructors**
      1. **Piece default**
      2. **Piece with parameters**
   3. **Getters**
      1. **getType**
      2. **getColor**
      3. **getPosX**
      4. **getPosY**
   4. **toString**

**Piece**

type   
color  
posX  
posY

Piece()  
Piece(String, String, Char, int)  
getType()  
getColor()  
getPosX()  
getPosY()  
toString()

**Lab1**

scnr  
board  
test

readPieces(String)  
countLines(String)  
makePiece(String[])  
validate(Piece[], char, int)  
validatePawn(String, char, char, int, int)  
validateKnight (String, char, char, int, int)  
validateBishop(String, char, char, int, int)  
validateRook (String, char, char, int, int)  
validateQueen (String, char, char, int, int)  
validateKing (String, char, char, int, int)  
main()