Report Lab 3 - 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 a Piece 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**

* You most use polymorphism
* You must factor out all common behavior and attributes in a new class
* You shall not use interfaces
* You must do first your algorithm, then the class diagram, and the pseudo code of the two most complex methods.

1. **High-level approach (algorithm)**
   1. The program receives a file containing the information of the pieces
   2. Line by line, the program reads the file.
   3. A piece object is created with the information, according to the name of the piece, the constructor of that type of piece is called
   4. The object gets stored in a piece object
   5. When the file is done reading, the array of pieces is returned
   6. The program asks the user to input the new position
   7. It iterates through the Piece array
   8. Checks if the new position is valid for the piece
   9. If it is, then it prints the move
   10. After the whole array is checked, the program asks the user if he wants to exit or test another position.
2. **2 Pseudo codes for major functions**

**//creates the piece according to its type / name**

**MakePiece(piece[])**

1. BEGIN
2. INIT Piece pieceObj = new Piece();
3. IF (piece.length == 4)
4. SWITCH (piece[0])
5. CASE "PAWN": pieceObj = new Pawn(piece)
6. break
7. CASE "KNIGHT": pieceObj = new Knight(piece)
8. break;
9. CASE "PAWN": pieceObj = new Pawn(piece)
10. break
11. CASE "KNIGHT": pieceObj = new Knight(piece)
12. break;
13. CASE "PAWN": pieceObj = new Pawn(piece)
14. break
15. CASE "KNIGHT": pieceObj = new Knight(piece)
16. break;
17. DEFAULT:
18. PRINT "WRONG PIECE"
19. END SWITCH
20. END IF
21. RETURN pieceObj
22. END

**//checks that the pawn moved at most once, or at most two only if its original position was the original position and only moved forward, that it moved forward or diagonally once. For black pieces its “backward”.**

**PAWN**

**validate(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

