PMD report

Problems found

#	File	Line	Problem
1	Project\src\com\Chess\ChessMove.java	1	Package name contains upper case characters
2	Project\src\com\Chess\ChessMove.java	9	The class 'ChessMove' has a Modified Cyclomatic Complexity of 5 (Highest = 11).
3	Project\src\com\Chess\ChessMove.java	9	The class 'ChessMove' has a Standard Cyclomatic Complexity of 5 (Highest = 11).
4	Project\src\com\Chess\ChessMove.java	9	The class 'ChessMove' has a total cyclomatic complexity of 89 (highest 19).
5	Project\src\com\Chess\ChessMove.java	9	This class has too many methods, consider refactoring it.
6	Project\src\com\Chess\ChessMove.java	15	Field comments are required
7	Project\src\com\Chess\ChessMove.java	16	Avoid using implementation types like 'ArrayList'; use the interface instead
8	Project\src\com\Chess\ChessMove.java	16	Field comments are required
9	Project\src\com\Chess\ChessMove.java	17	Public method and constructor comments are required
10	Project\src\com\Chess\ChessMove.java	23	Public method and constructor comments are required
11	Project\src\com\Chess\ChessMove.java	23	The method 'readInput' has a Modified Cyclomatic Complexity of 11.
12	Project\src\com\Chess\ChessMove.java	23	The method 'readInput' has a Standard Cyclomatic Complexity of 11.
13	Project\src\com\Chess\ChessMove.java	23	The method 'readInput(String)' has a cognitive complexity of 17, current threshold is 15
14	Project\src\com\Chess\ChessMove.java	23	The method 'readInput(String)' has a cyclomatic complexity of 19.
15	Project\src\com\Chess\ChessMove.java	23	The method 'readInput(String)' has an NPath complexity of 592, current threshold is 200
16	Project\src\com\Chess\ChessMove.java	26	Avoid using Literals in Conditional Statements
17	Project\src\com\Chess\ChessMove.java	31	Potential violation of Law of Demeter (method chain calls)
18	Project\src\com\Chess\ChessMove.java	31	Potential violation of Law of Demeter (method chain calls)
19	Project\src\com\Chess\ChessMove.java	32	Avoid using Literals in Conditional Statements
20	Project\src\com\Chess\ChessMove.java	39	Found 'DD'-anomaly for variable 'bwhite' (lines '39'-'39').
21	Project\src\com\Chess\ChessMove.java	39	Found 'DD'-anomaly for variable 'bwhite' (lines '39'-'41').
22	Project\src\com\Chess\ChessMove.java	39	Found 'DU'-anomaly for variable 'bwhite' (lines '39'-'65').
23	Project\src\com\Chess\ChessMove.java	41	Found 'DD'-anomaly for variable 'bwhite' (lines '41'-'39').
24	Project\src\com\Chess\ChessMove.java	41	Found 'DU'-anomaly for variable 'bwhite' (lines '41'-'65').
25	Project\src\com\Chess\ChessMove.java	45	Avoid instantiating new objects inside loops
26	Project\src\com\Chess\ChessMove.java	52	Potential violation of Law of Demeter (method chain calls)

27	Project\src\com\Chess\ChessMove.java	52	Potential violation of Law of Demeter (method chain calls)
28	Project\src\com\Chess\ChessMove.java	53	Avoid using Literals in Conditional Statements
29	Project\src\com\Chess\ChessMove.java	56	Avoid using Literals in Conditional Statements
30	Project\src\com\Chess\ChessMove.java	67	Public method and constructor comments are required
31	Project\src\com\Chess\ChessMove.java	72	Public method and constructor comments are required
32	Project\src\com\Chess\ChessMove.java	100	Avoid using Literals in Conditional Statements
33	Project\src\com\Chess\ChessMove.java	102	A method should have only one exit point, and that should be the last statement in the method
34	Project\src\com\Chess\ChessMove.java	105	A method should have only one exit point, and that should be the last statement in the method
35	Project\src\com\Chess\ChessMove.java	108	A method should have only one exit point, and that should be the last statement in the method
36	Project\src\com\Chess\ChessMove.java	110	Avoid using Literals in Conditional Statements
37	Project\src\com\Chess\ChessMove.java	111	Avoid unnecessary if..then..else statements when returning booleans
38	Project\src\com\Chess\ChessMove.java	112	A method should have only one exit point, and that should be the last statement in the method
39	Project\src\com\Chess\ChessMove.java	115	A method should have only one exit point, and that should be the last statement in the method
40	Project\src\com\Chess\ChessMove.java	119	Avoid unnecessary if..then..else statements when returning booleans
41	Project\src\com\Chess\ChessMove.java	120	A method should have only one exit point, and that should be the last statement in the method
42	Project\src\com\Chess\ChessMove.java	212	Found 'DD'-anomaly for variable 'bpos2' (lines '212'-'214').
43	Project\src\com\Chess\ChessMove.java	213	Avoid using Literals in Conditional Statements
44	Project\src\com\Chess\ChessMove.java	220	Found 'DD'-anomaly for variable 'bpos2' (lines '220'-'222').
45	Project\src\com\Chess\ChessMove.java	221	Avoid using Literals in Conditional Statements
46	Project\src\com\Chess\ChessMove.java	237	A method should have only one exit point, and that should be the last statement in the method
47	Project\src\com\Chess\ChessMove.java	240	A method should have only one exit point, and that should be the last statement in the method
48	Project\src\com\Chess\ChessMove.java	243	A method should have only one exit point, and that should be the last statement in the method
49	Project\src\com\Chess\ChessMove.java	246	Found 'DD'-anomaly for variable 'bsame' (lines '246'-'248').
50	Project\src\com\Chess\ChessMove.java	251	A method should have only one exit point, and that should be the last statement in the method
51	Project\src\com\Chess\ChessMove.java	255	Prefer StringBuilder (non-synchronized) or StringBuffer (synchronized) over += for concatenating strings
52	Project\src\com\Chess\ChessMove.java	263	A method should have only one exit point, and that should be the last statement in the method
53	Project\src\com\Chess\ChessMove.java	266	A method should have only one exit point, and that should be the last statement in the method
54	Project\src\com\Chess\ChessMove.java	269	A method should have only one exit point, and that should be the last statement in the method
55	Project\src\com\Chess\ChessMove.java	276	Prefer StringBuilder (non-synchronized) or StringBuffer (synchronized) over += for concatenating strings

56	Project\src\com\Chess\ChessMove.java	282	A method should have only one exit point, and that should be the last statement in the method
57	Project\src\com\Chess\ChessMove.java	285	A method should have only one exit point, and that should be the last statement in the method
58	Project\src\com\Chess\ChessMove.java	288	A method should have only one exit point, and that should be the last statement in the method
59	Project\src\com\Chess\ChessMove.java	292	Prefer StringBuilder (non-synchronized) or StringBuffer (synchronized) over += for concatenating strings
60	Project\src\com\Chess\ChessMove.java	297	Public method and constructor comments are required
61	Project\src\com\Chess\ChessMove.java	303	Class comments are required
62	Project\src\com\Chess\ChessMove.java	304	Field comments are required
63	Project\src\com\Chess\ChessMove.java	305	Field comments are required
64	Project\src\com\Chess\ChessMove.java	306	Field comments are required
65	Project\src\com\Chess\ChessMove.java	308	Field comments are required
66	Project\src\com\Chess\ChessMove.java	310	Public method and constructor comments are required
67	Project\src\com\Chess\ChessMove.java	312	Avoid using Literals in Conditional Statements
68	Project\src\com\Chess\ChessMove.java	322	Public method and constructor comments are required
69	Project\src\com\Chess\ChessMove.java	325	Prefer StringBuilder (non-synchronized) or StringBuffer (synchronized) over += for concatenating strings
70	Project\src\com\Chess\ChessMoveTest.java	1	Package name contains upper case characters
71	Project\src\com\Chess\ChessMoveTest.java	7	Class comments are required
72	Project\src\com\Chess\ChessMoveTest.java	7	The class 'ChessMoveTest' might be a test class, but it contains no test cases.
73	Project\src\com\Chess\ChessMoveTest.java	7	This class has too many methods, consider refactoring it.
74	Project\src\com\Chess\ChessMoveTest.java	10	Field comments are required
75	Project\src\com\Chess\ChessMoveTest.java	12	Public method and constructor comments are required
76	Project\src\com\Chess\ChessMoveTest.java	23	Assigning an Object to null is a code smell. Consider refactoring.
77	Project\src\com\Chess\ChessMoveTest.java	31	Potential violation of Law of Demeter (object not created locally)
78	Project\src\com\Chess\ChessMoveTest.java	31	The String literal "Input should be correct" appears 5 times in this file; the first occurrence is on line 31
79	Project\src\com\Chess\ChessMoveTest.java	38	Potential violation of Law of Demeter (object not created locally)
80	Project\src\com\Chess\ChessMoveTest.java	45	Potential violation of Law of Demeter (object not created locally)
81	Project\src\com\Chess\ChessMoveTest.java	195	The String literal "Test Pawn movement" appears 4 times in this file; the first occurrence is on line 195

I can report some code improvment. For this 'Package name contains upper case characters', I can change package name as 'com.Chess' -> 'com.chess'
I can change all constant values set as final member varible, for example '3' as 'private final int LEGAL_LINE_COUNT = 3;'
Next part is Code Coverage report

Coverage Summary for Class: ChessMove (com.Chess)

Class	Method, %	Line, %
ChessMove	93.3% (14/15)	94.5% (154/163)
ChessMove$Piece	100% (2/2)	100% (10/10)
Total	94.1% (16/17)	94.8% (164/173)

```
1  package com.Chess;
2
3  import java.util.ArrayList;
4  import static java.lang.System.*;
5
6  /**
7   * compute all legal moves for a piece on a given chessboard
8   */
9  public class ChessMove {
10
11     /**
12      * chess board 2d array
13      */
14     private Piece[][] mPBoard;
15     private Piece mPiece;
16     private ArrayList<String> mLegalMoves;
17     public ChessMove(){
18
19         mPBoard = new Piece[8][8];
20         mLegalMoves = new ArrayList<>();
21     }
22
23     public void readInput(final String inputstr){
24
25         final String[] rowlst = inputstr.split("\n");
26         if(rowlst.length != 3){
27             throw new ArrayIndexOutOfBoundsException("Input should have three lines.");
28         }
29
30         for(int i = 0 ; i < 2 ; i++){
31             final String[] rowline = rowlst[i].trim().split(":");
32             if(rowline.length != 2){
33                 throw new IllegalArgumentException("Input line format is invalid.");
34             }
35             final String side = rowline[0];
36             if(!"BLACK".equals(side) && !"WHITE".equals(side)){
37                 throw new IllegalArgumentException("Input lines should start with 'BLACK' or 'WHITE'.");
38             }
39             boolean bwhite = true;
40             if("BLACK".equals(side)) {
41                 bwhite = false;
42             }
43             final String[] arrPiece = rowline[1].split(",");
44             for(final String piece : arrPiece){
45                 final Piece curPiece = new Piece(piece.trim(), bwhite);
46                 if(!isValidPiece(curPiece, ' ')){
47                     throw new IllegalArgumentException("Piece format of input line is invalid.");
48                 }
49                 mPBoard[curPiece.posX][curPiece.posY] = curPiece;
50             }
51         }
52         final String[] prow = rowlst[2].trim().split(":");
53         if(prow.length != 2){
54             throw new IllegalArgumentException("Line format of moving piece is invalid.");
55         }
56         if(!"PIECE TO MOVE".equals(prow[0].trim())){
57             throw new IllegalArgumentException("Line format of moving should start with 'PIECE TO MOVE'.");
58         }
59         mPiece = new Piece(prow[1].trim(), false);
60         if(!isValidPiece(mPiece, mPiece.name)){
61             throw new IllegalArgumentException("The name of moveing piece does not match with board.");
62         }
63         mPiece.bWhite = mPBoard[mPiece.posX][mPiece.posY].bWhite;
64
65     }
66
67     public void clearBoard(){
68         mPBoard = new Piece[8][8];
69         mLegalMoves = new ArrayList<>();
70     }
71
72     public String movePiece(){
73         switch (mPiece.name) {
74             case 'K':
75                 moveKing();
76                 break;
77             case 'Q':
78                 moveQueen();
79                 break;
80             case 'R':
81                 moveRook();
82                 break;
83             case 'B':
84                 moveBishop();
85                 break;
86             case 'N':
```

```
                    moveKnight();
                    break;
                case 'P':
                    movePawn();
                    break;
                default:
                    break;
            }
            String result = "LEGAL MOVES FOR " + mPiece.getPiece() + ": ";
            result += String.join(", ", mLegalMoves);
            return result;
        }
        private boolean isValidPiece(final Piece curPiece, final char place){
            if(!"KQRBNP".contains(String.valueOf(curPiece.name))){
                return false;
            }
            if(curPiece.posX < 0 || curPiece.posX > 7) {
                return false;
            }
            if(curPiece.posY < 0 || curPiece.posY > 7) {
                return false;
            }
            if(place == ' '){
                if(mPBoard[curPiece.posX][curPiece.posY] == null) {
                    return true;
                }
                else {
                    return false;
                }
            }
            else{
                if(mPBoard[curPiece.posX][curPiece.posY].name == place) {
                    return true;
                }
                else {
                    return false;
                }
            }

        }

        private void moveKing(){
            for(int i = -1; i < 2; i++){
                for(int j = -1; j < 2 ; j++){
                    final int posx = i + mPiece.posX;
                    final int posy = j + mPiece.posY;
                    addMove(posx, posy);
                }
            }
        }

        private void moveRook(){
            for(int i = mPiece.posX - 1; i >= 0 ; i--){
                if(!addMove(i, mPiece.posY)) {
                    break;
                }
            }
            for(int i = mPiece.posX + 1; i < 8 ; i++){
                if(!addMove(i, mPiece.posY)) {
                    break;
                }
            }
            for(int i = mPiece.posY - 1; i >= 0 ; i--){
                if(!addMove(mPiece.posX, i)) {
                    break;
                }
            }
            for(int i = mPiece.posY + 1; i < 8 ; i++){
                if(!addMove(mPiece.posX, i)) {
                    break;
                }
            }
        }

        private void moveBishop(){
            for(int i = 1 ; i < 8 ; i++){
                final int posx = mPiece.posX - i;
                final int posy = mPiece.posY - i;
                if(!addMove(posx, posy)) {
                    break;
                }
            }
```

```
        for(int i = 1 ; i < 8 ; i++){
            final int posx = mPiece.posX - i;
            final int posy = mPiece.posY + i;
            if(!addMove(posx, posy)) {
                break;
            }
        }
        for(int i = 1 ; i < 8 ; i++){
            final int posx = mPiece.posX + i;
            final int posy = mPiece.posY + i;
            if(!addMove(posx, posy)) {
                break;
            }
        }
        for(int i = 1 ; i < 8 ; i++){
            final int posx = mPiece.posX + i;
            final int posy = mPiece.posY - i;
            if(!addMove(posx, posy)) {
                break;
            }
        }
    }
    private void moveQueen(){
        moveRook();
        moveBishop();
    }
    private void moveKnight(){
        addMove(mPiece.posX - 1, mPiece.posY - 2);
        addMove(mPiece.posX + 1, mPiece.posY - 2);
        addMove(mPiece.posX + 2, mPiece.posY - 1);
        addMove(mPiece.posX - 2, mPiece.posY - 1);
        addMove(mPiece.posX - 2, mPiece.posY + 1);
        addMove(mPiece.posX + 2, mPiece.posY + 1);
        addMove(mPiece.posX + 1, mPiece.posY + 2);
        addMove(mPiece.posX - 1, mPiece.posY + 2);
    }

    private void movePawn(){
        boolean bpos2 ;
        if(mPiece.bWhite){
            addMovePawnD(mPiece.posX - 1, mPiece.posY + 1);
            addMovePawnD(mPiece.posX + 1, mPiece.posY + 1);
            bpos2 = addMovePawnF(mPiece.posX, mPiece.posY + 1);
            if(mPiece.posY != 1) {
                bpos2 = false;
            }
        }
        else{
            addMovePawnD(mPiece.posX - 1, mPiece.posY - 1);
            addMovePawnD(mPiece.posX + 1, mPiece.posY - 1);
            bpos2 = addMovePawnF(mPiece.posX, mPiece.posY - 1);
            if(mPiece.posY != 6) {
                bpos2 = false;
            }
        }
        if(bpos2){
            if(mPiece.bWhite){
                addMovePawnF(mPiece.posX, mPiece.posY + 2);
            }
            else{
                addMovePawnF(mPiece.posX, mPiece.posY - 2);
            }
        }

    }
    private boolean addMove(final int posx, final int posy){
        if(posx < 0 || posx > 7) {
            return false;
        }
        if(posy < 0 || posy > 7) {
            return false;
        }
        if(posx == mPiece.posX && posy == mPiece.posY) {
            return false;
        }
        final boolean bnull = mPBoard[posx][posy] == null;
        boolean bsame = false;
        if(!bnull){
            bsame = mPBoard[posx][posy].bWhite == mPiece.bWhite;
        }
        if(bsame) {
            return false;
        }
        String strMove = "";
        strMove += String.valueOf((char)(posx + 'a'));
        strMove += String.valueOf((char)(posy + '1'));
```

```
            mLegalMoves.add(strMove);

            return bnull;
        }

        private void addMovePawnD(final int posx, final int posy){
            if(posx < 0 || posx > 7) {
                return ;
            }
            if(posy < 0 || posy > 7) {
                return ;
            }
            if(mPBoard[posx][posy] == null) {
                return;
            }
            if(mPBoard[posx][posy].bWhite == mPiece.bWhite) {
                return;
            }
            String strMove = "";
            strMove += String.valueOf((char)(posx + 'a'));
            strMove += String.valueOf((char)(posy + '1'));
            mLegalMoves.add(strMove);
        }

        private boolean addMovePawnF(final int posx, final int posy) {
            if(posx < 0 || posx > 7) {
                return false;
            }
            if(posy < 0 || posy > 7) {
                return false;
            }
            if(mPBoard[posx][posy] != null) {
                return false;
            }
            String strMove = "";
            strMove += String.valueOf((char)(posx + 'a'));
            strMove += String.valueOf((char)(posy + '1'));
            mLegalMoves.add(strMove);
            return true;
        }

        public static void main(final String[] args) {
            final ChessMove chess = new ChessMove();
            chess.readInput("WHITE: Rf1, Kg1, Pf2, Ph2, Pg3\nBLACK: Kb8, Ne8, Pa7, Pb7, Pc7, Ra5\nPIECE TO MOVE: Rf1");
            out.println(chess.movePiece());
        }

        private class Piece {
            public char name;
            public  int posX;
            public  int posY;

            public boolean bWhite;

            public Piece(final String piece, final boolean bWhite){

                if(piece.length() != 3) {
                    return;
                }
                name = piece.charAt(0);
                posX = piece.charAt(1) - 'a';
                posY = piece.charAt(2) - '1';
                this.bWhite = bWhite;

            }

            public String getPiece(){
                String str = String.valueOf(name);
                str += String.valueOf((char)(posX + 'a'));
                str += String.valueOf((char)(posY + '1'));
                return str;
            }
        }
    }
```