
Try a few moves:

8	bR	bN	bB	bQ	bK	bB	bN	bR
7	bP	bP	bP	bP	bP	bP	bP	bP
6								
5								
4								
3								
2	wP	wP	wP	wP	wP	wP	wP	wP
1	wR	wN	wB	wQ	wK	wB	wN	wR
	a	b	c	d	e	f	g	h

e2-e4

8	bR	bN	bB	bQ	bK	bB	bN	bR
7	bP	bP	bP	bP	bP	bP	bP	bP
6								
5								
4					wP			
3								
2	wP	wP	wP	wP		wP	wP	wP
1	wR	wN	wB	wQ	wK	wB	wN	wR
	a	b	c	d	e	f	g	h

b7-b6

8	bR	bN	bB	bQ	bK	bB	bN	bR
7	bP		bP	bP	bP	bP	bP	bP
6		bP						
5								
4					wP			
3								
2	wP	wP	wP	wP		wP	wP	wP
1	wR	wN	wB	wQ	wK	wB	wN	wR
	a	b	c	d	e	f	g	h

wNq1-f3

8	bR	bN	bB	bQ	bK	bB	bN	bR
---	----	----	----	----	----	----	----	----

7	bP		bP	bP	bP	bP	bP	bP
6		bP						
5								
4				wP				
3					wN			
2	wP	wP	wP	wP		wP	wP	wP
1	wR	wN	wB	wQ	wK	wB		wR
	a	b	c	d	e	f	g	h

bBc8-b7

8	bR	bN		bQ	bK	bB	bN	bR
7	bP	bB	bP	bP	bP	bP	bP	bP
6		bP						
5								
4					wP			
3						wN		
2	wP	wP	wP	wP		wP	wP	wP
1	wR	wN	wB	wQ	wK	wB		wR
	a	b	c	d	e	f	g	h

d2-d3

8	bR	bN		bQ	bK	bB	bN	bR
7	bP	bB	bP	bP	bP	bP	bP	bP
6		bP						
5								
4				wP				
3			wP		wN			
2	wP	wP	wP		wP	wP	wP	
1	wR	wN	wB	wQ	wK	wB		wR
	a	b	c	d	e	f	g	h

bBb7xe4

	bR	bN		bQ	bK	bB	bN	bR
8								
7	bP		bP	bP	bP	bP	bP	bP

```

6 |   |   | bP |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
5 |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
4 |   |   |   |   | bB |   |   |   |   |
+---+---+---+---+---+---+---+---+
3 |   |   |   |   | wP |   |   | wN |   |
+---+---+---+---+---+---+---+---+
2 | wP | wP | wP |   |   |   | wP | wP | wP |
+---+---+---+---+---+---+---+---+
1 | wR | wN | wB | wQ | wK | wB |   | wR |
+---+---+---+---+---+---+---+---+
  a   b   c   d   e   f   g   h

```

d3xe4

```

+---+---+---+---+---+---+---+---+
8 | bR | bN |   |   | bQ | bK | bB | bN | bR |
+---+---+---+---+---+---+---+---+
7 | bP |   |   | bP | bP | bP | bP | bP | bP |
+---+---+---+---+---+---+---+---+
6 |   | bP |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
5 |   |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
4 |   |   |   |   |   | wP |   |   |   |
+---+---+---+---+---+---+---+---+
3 |   |   |   |   |   |   | wN |   |   |
+---+---+---+---+---+---+---+---+
2 | wP | wP | wP |   |   |   | wP | wP | wP |
+---+---+---+---+---+---+---+---+
1 | wR | wN | wB | wQ | wK | wB |   | wR |
+---+---+---+---+---+---+---+---+
  a   b   c   d   e   f   g   h

```

Reset board

```

+---+---+---+---+---+---+---+---+
8 | bR | bN | bB | bQ | bK | bB | bN | bR |
+---+---+---+---+---+---+---+---+
7 | bP | bP | bP | bP | bP | bP | bP | bP |
+---+---+---+---+---+---+---+---+
6 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
5 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
4 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
3 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
2 | wP | wP | wP | wP | wP | wP | wP | wP |
+---+---+---+---+---+---+---+---+
1 | wR | wN | wB | wQ | wK | wB | wN | wR |
+---+---+---+---+---+---+---+---+
  a   b   c   d   e   f   g   h

```

Test En Passant

```

+---+---+---+---+---+---+---+---+
8 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
7 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
6 |   |   |   |   |   |   |   |   |

```

```

+---+---+---+---+---+---+---+---+
5 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
4 | wP | bP |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
3 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
2 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
1 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
  a   b   c   d   e   f   g   h

```

b4xa3

```

+---+---+---+---+---+---+---+---+
8 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
7 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
6 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
5 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
4 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
3 | bP |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
2 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
1 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
  a   b   c   d   e   f   g   h

```

Test promotion

```

+---+---+---+---+---+---+---+---+
8 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
7 |   | wP |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
6 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
5 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
4 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
3 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
2 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
1 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
  a   b   c   d   e   f   g   h

```

b7-b8wQ

```

+---+---+---+---+---+---+---+---+
8 |   | wQ |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
7 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
6 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+
5 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+---+

```

The figure shows a grid of points. The horizontal axis is labeled with letters 'a' through 'h' at the bottom. The vertical axis is labeled with numbers '1' through '4' on the left. The grid consists of 8 columns and 4 rows of points, each marked with a '+' symbol. Dashed lines connect the points horizontally and vertically, forming a grid of squares.

Test Castle: 0-0

8	bR				bK			bR
7								
6								
5								
4								
3								
2								
1	wR				wK			wR
	a	b	c	d	e	f	g	h

0-0

8	bR				bK			bR
7								
6								
5								
4								
3								
2								
1	wR					wR	wK	
	a	b	c	d	e	f	g	h

```
Can white castle: false
Can black castle: true
```

0-0

[illegible]

```
Can white castle: false
```

```
Can black castle: false
```

Test Castle: 0-0-0

8	bR				bK			bR
7								
6								
5								
4								
3								
2								
1	wR				wK			wR
	a	b	c	d	e	f	g	h

0-0-0

8	bR				bK			bR
7								
6								
5								
4								
3								
2								
1			wK	wR				wR
	a	b	c	d	e	f	g	h

```
Can white castle: false
```

```
Can black castle: true
```

0-0-0

+

8			bK	bR				bR
7								
6								
5								
4								
3								
2								
1		wK	wR					wR
	a	b	c	d	e	f	g	h

```
Can white castle: false
Can black castle: false
```

Test move with king

8	bR				bK			bR
7								
6								
5								
4								
3								
2								
1	wR				wK			wR
	a	b	c	d	e	f	g	h

wKe1-e2

8	bR				bK			bR
7								
6								
5								
4								
3								
2					wK			
1	wR							wR
	a	b	c	d	e	f	g	h

```
Can white castle: false
```

```
Can black castle: true
```

Test move with rooks

8	bR				bK			bR
7								
6								
5								
4								
3								
2								
1	wR				wK			wR
	a	b	c	d	e	f	g	h

wRa1xa8

8	wR				bK			bR
7								
6								
5								
4								
3								
2								
1					wK			wR
	a	b	c	d	e	f	g	h

```
Can white castle: true
Can black castle: true
```

bKe8-f8

8	wR					bK		bR
7								
6								
5								
4								
3								
2								
1					wK			wR

main-output.txt

Montag, 13. November 2017 09:15

```

+---+---+---+---+---+---+---+
| a | b | c | d | e | f | g | h |
+---+---+---+---+---+---+---+

```

```

Can white castle: true
Can black castle: false

```

wRh1-h4

```

+---+---+---+---+---+---+---+
8 | wR |   |   |   |   | bK |   | bR |
+---+---+---+---+---+---+---+
7 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
6 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
5 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
4 |   |   |   |   |   |   |   | wR |
+---+---+---+---+---+---+---+
3 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
2 |   |   |   |   |   |   |   |   |
+---+---+---+---+---+---+---+
1 |   |   |   |   | wK |   |   |   |
+---+---+---+---+---+---+---+
| a | b | c | d | e | f | g | h |

```

```

Can white castle: false
Can black castle: false

```

Listing 1: source/Figure.java

```

1 package org.chpr.chess.objects;
import org.chpr.chess.IBoard;
5 import java.util.ArrayList;
import java.util.List;

public class Figure {
    public static final int PAWN = 1;
10 public static final int ROOK = 2;
    public static final int KNIGHT = 3;
    public static final int BISHOP = 4;
    public static final int QUEEN = 5;
    public static final int KING = 6;

15 public static final String PAWN_STRING = "P";
    public static final String ROOK_STRING = "R";
    public static final String KNIGHT_STRING = "N";
    public static final String BISHOP_STRING = "B";
20 public static final String QUEEN_STRING = "Q";
    public static final String KING_STRING = "K";

    public static final int WHITE_OFFSET = 0;
    public static final int BLACK_OFFSET = 10;

25 public static final String WHITE_STRING = "w";
    public static final String BLACK_STRING = "b";

    public static final int WHITE = 0;
30 public static final int BLACK = 1;

    public static final String[] ARR_TYPE_STRING = {PAWN_STRING, ROOK_STRING,
        KNIGHT_STRING, BISHOP_STRING, QUEEN_STRING, KING_STRING};

    public static int getType(int figureIndex) {
35 return figureIndex % 10;
    }

    public static int getColor(int figureIndex) {
40 return figureIndex / 10;
    }

    public static String toString(int figureIndex) {
        return toString(getColor(figureIndex), getType(figureIndex));
    }

45 public static String toString(int color, int type) {
    if (type == 0) {
        return "░░";
    }
    String ret = "";
50 if (color == WHITE)
        ret = ret.concat(WHITE_STRING);
    else if (color == BLACK)
        ret = ret.concat(BLACK_STRING);
55 ret = ret.concat(ARR_TYPE_STRING[type - 1]);
    return ret;
}

60 public static short fromString(String str) {
    short ret = 0;
    if (str.startsWith(WHITE_STRING)) {
        ret += WHITE_OFFSET;
    }
}

```

```

        str = str.substring(1);
    }
    else if (str.startsWith(BLACK_STRING)) {
        ret += BLACK_OFFSET;
        str = str.substring(1);
    }
    for (int i = 0; i < ARR_TYPE_STRING.length; i++)
        if (str.equals(ARR_TYPE_STRING[i]))
            return (short)(ret + i + 1);
    return -1;
}

static public List<Move> getValidMoves(IBoard board, int col, int row) {
    //TODO: Implement in exercise 2
    return null;
}

static private boolean isValidDestination(IBoard board, int color, int col, int row)
{
    if (col < 0 || col > 7)
        return false;
    if (row < 0 || row > 7)
        return false;
    short[][] figures = board.getFigures();
    short fig = figures[col][row];
    if (color == WHITE && fig > 0 && fig < BLACK_OFFSET)
        return false;
    if (color == BLACK && fig > BLACK_OFFSET)
        return false;
    return true;
}

static private boolean isFree(IBoard board, int col, int row) {
    if (col < 0 || col > 7)
        return false;
    if (row < 0 || row > 7)
        return false;
    short[][] figures = board.getFigures();
    short fig = figures[col][row];
    if (fig == 0)
        return true;
    return false;
}
}

```

Listing 2: source/Main.java

```

1 package org.chpr;

import org.chpr.chess.Board;
import org.chpr.chess.objects.Figure;
5 import org.chpr.chess.objects.Move;

public class Main {
    private static Board board;
    private static int currentColor;

10    public static void main(String[] args) {
        currentColor = Figure.WHITE;
        board = new Board();

15        testFewMoves();
        testEnPassant();
        testPromotion();
    }
}

```

```

        testCastle();
    }

    private static void testFewMoves() {
        printBoard("Try a few moves:");
        executeAndPrint("e2-e4");
        executeAndPrint("b7-b6");
        executeAndPrint("Ng1-f3");
        executeAndPrint("Bc8-b7");
        executeAndPrint("d2-d3");
        executeAndPrint("Bb7xe4");
        executeAndPrint("d3xe4");

        resetBoard(true);
        System.out.println("-----\n");
    }

    private static void testEnPassant() {
        short[][] figures = new short[8][8];
        figures[0][3] = Figure.PAWN + Figure.WHITE_OFFSET;
        figures[1][3] = Figure.PAWN + Figure.BLACK_OFFSET;
        board.setFigures(figures);
        currentColor = Figure.BLACK;

        printBoard("Test En Passant");
        executeAndPrint("b4xa3");
        resetBoard(false);
    }

    private static void testPromotion() {
        short[][] figures = new short[8][8];
        figures[1][6] = Figure.PAWN + Figure.WHITE_OFFSET;
        board.setFigures(figures);
        currentColor = Figure.WHITE;

        printBoard("Test promotion");
        executeAndPrint("b7-b8Q");
        resetBoard(false);
        System.out.println("-----\n");
    }

    private static void testCastle() {
        setUpCastle();
        printBoard("Test Castle: 0-0");
        executeAndPrint("0-0");
        printCastleStatus();
        executeAndPrint("0-0");
        printCastleStatus();

        setUpCastle();
        printBoard("Test Castle: 0-0-0");
        executeAndPrint("0-0-0");
        printCastleStatus();
        executeAndPrint("0-0-0");
        printCastleStatus();

        setUpCastle();
        printBoard("Test move with king");
        executeAndPrint("Ke1-e2");
        printCastleStatus();

        setUpCastle();
        printBoard("Test move with rooks");
        executeAndPrint("Ra1xa8");
    }
}

```

```

85     printCastleStatus();
        executeAndPrint("Ke8-f8");
        printCastleStatus();
        executeAndPrint("Rh1-h4");
        printCastleStatus();

        resetBoard(false);
90     System.out.println("-----\n");
    }

    private static void setUpCastle() {
        resetBoard(false);
95     short[][] figures = new short[8][8];
        figures[0][0] = Figure.ROOK + Figure.WHITE_OFFSET;
        figures[7][0] = Figure.ROOK + Figure.WHITE_OFFSET;
        figures[4][0] = Figure.KING + Figure.WHITE_OFFSET;

        figures[0][7] = Figure.ROOK + Figure.BLACK_OFFSET;
        figures[7][7] = Figure.ROOK + Figure.BLACK_OFFSET;
        figures[4][7] = Figure.KING + Figure.BLACK_OFFSET;

        board.setFigures(figures);
105    currentColor = Figure.WHITE;
    }

    private static void executeAndPrint(String moveString) {
        Move m = Move.Import(moveString, board, currentColor);
110    board.executeMove(m);
        System.out.println(m);
        System.out.println(board);
        System.out.println("\n");

115    currentColor = (currentColor == Figure.WHITE ? Figure.BLACK : Figure.WHITE);
    }

    private static void printBoard(String title) {
        System.out.println(title);
120    printBoard();
    }

    private static void printBoard() {
        System.out.println(board);
125    System.out.println("\n");
    }

    private static void printCastleStatus() {
        System.out.println("Can_White_Castle:_" + board.canWhiteCastle());
130    System.out.println("Can_Black_Castle:_" + board.canBlackCastle());
        System.out.println("\n");
    }

    private static void resetBoard(boolean print) {
135    board.reset();
        currentColor = Figure.WHITE;
        if (print) {
            System.out.println("Reset_Board");
            System.out.println(board);
            System.out.println("\n");
140    }
    }
}

```

Listing 3: source/Move.java

```

1 // e2-e4 pawn moves from e2 to e4
  // e2xf3 pawn moves from e2 to f3 and hits
  // Ra1-a8 rook moves from a1 to a8
  // a7xb8N pawn hits b8 and gets promoted to knight
5 // a7-a8Q pawn moves to a8 and gets promoted to queen
  // 0-0 king side castle
  // 0-0-0 queen side castle
  //

10 package org.chpr.chess.objects;

import org.chpr.chess.IBoard;

import java.util.List;

15 public class Move {

    private int color;
    private final int type;
20    private final int sourceCol;
    private final int sourceRow;
    private final int destCol;
    private final int destRow;
    private int fig;
25    private IBoard board;
    private boolean hit;
    private boolean prom;

    public Move(IBoard board, int color, int type, int sourceCol, int sourceRow, int
        destCol, int destRow, boolean newType) {
30    this.color = color;
        this.type = type;
        this.sourceCol = sourceCol;
        this.sourceRow = sourceRow;
        this.destCol = destCol;
        this.destRow = destRow;
        this.board = board;
        this.fig = type + (color == Figure.WHITE ? Figure.WHITE_OFFSET :
            Figure.BLACK_OFFSET);
        this.hit = false; // will be handled by setHit()
        this.prom = newType;
40    }

    public IBoard getBoard() {
        return board;
    }

45    public int getFigureIndex() {
        return fig;
    }

    public int getColor() {
50    return color;
    }

    public int getType() {
55    return type;
    }

    public int getSourceCol() {
        return sourceCol;
60    }

    public int getSourceRow() {

```

```

    return sourceRow;
}

65 public int getDestCol() {
    return destCol;
}

70 public int getDestRow() {
    return destRow;
}

75 public void setColor(int color) {
    if (this.color != color) {
        if (color == Figure.WHITE)
            fig -= 10;
        else if (color == Figure.BLACK)
            fig += 10;
    }
    this.color = color;
}

85 public boolean isHit() {
    return hit;
}

public void setHit() {
    hit = true;
}

90 public boolean isProm() {
    return prom;
}

95 @Override
public String toString() {
    String ret = "";
    if (Figure.getType(fig) == Figure.KING) {
        if (sourceCol == 4 && destCol == 6)
            return "0-0";
        if (sourceCol == 4 && destCol == 2)
            return "0-0-0";
    }
    if (Figure.getType(fig) != Figure.PAWN && !prom) {
        ret = ret.concat(Figure.toString(fig));
    }
    char[] col = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'};
    ret = ret.concat(Character.toString(col[sourceCol]));
    ret = ret.concat(Integer.toString(sourceRow + 1).concat(hit ? "x" : "-"));
    ret = ret.concat(Character.toString(col[destCol]));
    ret = ret.concat(Integer.toString(destRow + 1));
    if (prom)
        ret = ret.concat(Figure.toString(fig));
115 // short[[]] b = board.getFigures();
// for (int column = 0; column < b.length; column++) {
//     for (int row = 0; row < b[0].length; row++) {
//         if (b[column][row] == Figure.KING + (color == Figure.WHITE ?
//         Figure.BLACK_OFFSET : Figure.WHITE_OFFSET)) {
120 //             // TODO check for check, need valid moves for this
//             }
//         }
//     }
    return ret;
125 }

```

```

@Override
public boolean equals(Object obj) {
    if (obj.getClass() != Move.class)
        return false;
    Move m = (Move)obj;
    if (color != m.getColor())
        return false;
    if (!board.equals(m.getBoard()))
        return false;
    if (sourceCol != m.getSourceCol())
        return false;
    if (sourceRow != m.getSourceRow())
        return false;
    if (destCol != m.getDestCol())
        return false;
    if (destRow != m.getDestRow())
        return false;
    if (fig != m.getFigureIndex())
        return false;
    // no need to check for hit, because when all is equal, then hit is also equal
    // (when setHit() is used correctly)
    // no need to check for color, its in figureIndex
    // no need to check for type, its in figureIndex (fig)
    // no need to check for promotion, because when all is equal, then promotion is
    // also equal
150 return true;
}

public static Move Import(String str, IBoard board, int color) {
    if (str.equals("0-0")) {
        int row = (color == Figure.WHITE ? 0 : 7);
        return new Move(board, color, Figure.KING, 4, row, 6, row, false);
    }
    if (str.equals("0-0-0")) {
        int row = (color == Figure.WHITE ? 0 : 7);
        return new Move(board, color, Figure.KING, 4, row, 2, row, false);
    }
    char[] col = {'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h'};
    int figureIndex = 0;
    if (color == Figure.BLACK)
        figureIndex += Figure.BLACK_OFFSET;
    // if str starts with a-h, then its a pawn
    for (char c : col) {
        if (str.startsWith(Character.toString(c))) {
            figureIndex += Figure.PAWN;
            // concat for equal position of cols, rows, ...
            str = "P".concat(str);
            break;
        }
    }
    if (str.startsWith(Figure.ROOK_STRING))
        figureIndex += Figure.ROOK;
    else if (str.startsWith(Figure.KNIGHT_STRING))
        figureIndex += Figure.KNIGHT;
    else if (str.startsWith(Figure.BISHOP_STRING))
        figureIndex += Figure.BISHOP;
    else if (str.startsWith(Figure.QUEEN_STRING))
        figureIndex += Figure.QUEEN;
    else if (str.startsWith(Figure.KING_STRING))
        figureIndex += Figure.KING;
185 int sourceCol = str.charAt(1) - 'a';
int sourceRow = Integer.parseInt(Character.toString(str.charAt(2))) - 1;
boolean hit = Character.toString(str.charAt(3)).equals("x") ? true : false;

```



```

190     int destCol = str.charAt(4) - 'a';
    int destRow = Integer.parseInt(Character.toString(str.charAt(5))) - 1;
    // if str.length() is 7, then it must be a promotion with the additional
    // newTypeStr at the end
    if (str.length() == 7) {
        String newTypeStr = Character.toString(str.charAt(6));
        int newType = Figure.fromString(newTypeStr);
        Move m = new Move(board, color, newType, sourceCol, sourceRow, destCol, destRow,
195             true);
        if (hit)
            m.setHit();
        return m;
    }
    Move m = new Move(board, color, Figure.getType(figureIndex), sourceCol, sourceRow,
200     destCol, destRow, false);
    if (hit)
        m.setHit();
    return m;
}

205 public static boolean MovesListIncludesMove(List<Move> moves, Move move) {
    for (Move m : moves)
        if (move.equals(m))
            return true;
    return false;
}
210 }

```

Listing 4: source/IBoard.java

```

1 package org.chpr.chess;

import org.chpr.chess.objects.Move;

5 import java.util.List;

/**
 * Interface for public methods of our chess board
 */
10 public interface IBoard {
    /**
     * Return all figures that are on the board
     *
     * @return array of figures
     */
15 short[][] getFigures();

    /**
     * Set given figure at given position
     *
     * @param row the row where the figure should be positioned
     * @param column the column where the figure should be positioned
     * @param figure the figure that should be positioned
     */
25 void setFigure(int row, int column, short figure);

    /**
     * Reset the board, i.e. set board to start position
     */
30 void reset();

    /**
     * Copy board except with empty history
     *
     * @return incomplete copy of the board
     */
35

```

```

    /**
    IBoard cloneIncompletely();

    /**
    * Return list of all valid moves on the current board
    *
    * @return list of all valid moves
    */
    List<Move> getValidMoves();

45    /**
    * Return list of all valid moves for a given color on the current board
    *
    * @param color color for which valid moves should be returned
    * @return list of all valid moves for given color
    */
    List<Move> getValidMoves(int color);

55    /**
    * Return List of previously executed moves
    *
    * @return list of previously executed moves
    */
    List<Move> getHistory();

60    /**
    * Execute given move on current position
    *
    * @param move move to execute
    */
65 void executeMove(Move move);

    /**
    * Check if white can still castle
    *
    * @return true, if white can still castle
    */
70 boolean canWhiteCastle();

    /**
    * Check if black can still castle
    *
    * @return true, if black can still castle
    */
75 boolean canBlackCastle();

    /**
    * Check if color is in mat
    *
    * @param color color which should be checked for
    * @return true, if color is in mat
    */
85 boolean isMat(int color);
}

```

Listing 5: source/Board.java

```

1 package org.chpr.chess;

import org.chpr.chess.objects.Figure;
import org.chpr.chess.objects.Move;

5 import java.util.LinkedList;
import java.util.List;

```

```

10 public class Board implements IBoard {
    static private int COLS = 8;
    static private int ROWS = 8;

    private short[][] figures;
    private boolean canWhiteCastleKingside = true;
    private boolean canWhiteCastleQueenside = true;
    private boolean canBlackCastleKingside = true;
    private boolean canBlackCastleQueenside = true;
    private List<Move> history;

    public Board() {
        this.reset();
    }

    @Override
    public short[][] getFigures() {
        return figures;
    }

    @Override
    public void setFigure(int row, int column, short figure) {
        figures[column][row] = figure;
    }

    /**
     * Set figures of board for better testing
     *
     * @param figures new position of board
     */
    public void setFigures(short[][] figures) {
        this.figures = figures;
    }

    @Override
    public void reset() {
        figures = new short[COLS][ROWS];
        // set white pieces
        figures[0][0] = Figure.ROOK + Figure.WHITE_OFFSET;
        figures[1][0] = Figure.KNIGHT + Figure.WHITE_OFFSET;
        figures[2][0] = Figure.BISHOP + Figure.WHITE_OFFSET;
        figures[3][0] = Figure.QUEEN + Figure.WHITE_OFFSET;
        figures[4][0] = Figure.KING + Figure.WHITE_OFFSET;
        figures[5][0] = Figure.BISHOP + Figure.WHITE_OFFSET;
        figures[6][0] = Figure.KNIGHT + Figure.WHITE_OFFSET;
        figures[7][0] = Figure.ROOK + Figure.WHITE_OFFSET;
        for (int col = 0; col < COLS; col++) {
            figures[col][1] = Figure.PAWN + Figure.WHITE_OFFSET;
        }
        // set black pieces
        int lastRow = ROWS - 1;
        figures[0][lastRow] = Figure.ROOK + Figure.BLACK_OFFSET;
        figures[1][lastRow] = Figure.KNIGHT + Figure.BLACK_OFFSET;
        figures[2][lastRow] = Figure.BISHOP + Figure.BLACK_OFFSET;
        figures[3][lastRow] = Figure.QUEEN + Figure.BLACK_OFFSET;
        figures[4][lastRow] = Figure.KING + Figure.BLACK_OFFSET;
        figures[5][lastRow] = Figure.BISHOP + Figure.BLACK_OFFSET;
        figures[6][lastRow] = Figure.KNIGHT + Figure.BLACK_OFFSET;
        figures[7][lastRow] = Figure.ROOK + Figure.BLACK_OFFSET;
        for (int col = 0; col < COLS; col++) {
            figures[col][lastRow - 1] = Figure.PAWN + Figure.BLACK_OFFSET;
        }
        // reset other properties

```

```

75     canWhiteCastleKingside = true;
    canWhiteCastleQueenside = true;
    canBlackCastleKingside = true;
    canBlackCastleQueenside = true;
    history = new LinkedList<>();
}

80 @Override
    public IBoard cloneIncompletely() {
        Board clonedBoard = new Board();
        clonedBoard.figures = figures;
        clonedBoard.canWhiteCastleKingside = canWhiteCastleKingside;
        clonedBoard.canWhiteCastleQueenside = canWhiteCastleQueenside;
        clonedBoard.canBlackCastleKingside = canBlackCastleKingside;
        clonedBoard.canBlackCastleQueenside = canBlackCastleQueenside;
        return clonedBoard;
    }

90 @Override
    public List<Move> getValidMoves() {
        //TODO: Implement in exercise 2
        return null;
    }

95 @Override
    public List<Move> getValidMoves(int color) {
        //TODO: Implement in exercise 2
        return null;
    }

100 @Override
    public List<Move> getHistory() {
        return history;
    }

    @Override
    public void executeMove(Move move) {
        int srcCol = move.getSourceCol();
        int srcRow = move.getSourceRow();
        int destCol = move.getDestCol();
        int destRow = move.getDestRow();
        short srcFigure = figures[srcCol][srcRow];
        short destFigure = figures[destCol][destRow];
        boolean whiteMove = move.getColor() == Figure.WHITE;

        figures[destCol][destRow] = srcFigure;
        figures[srcCol][srcRow] = 0;

        if (move.isProm()) {
            // promotion
            short newFigure = (short)(move.getType() + (whiteMove ? Figure.WHITE_OFFSET :
                Figure.BLACK_OFFSET));
            figures[destCol][destRow] = newFigure;
        }
        if (move.getType() == Figure.PAWN && move.isHit() && destFigure == 0) {
            // En passant
            figures[destCol][destRow + (whiteMove ? -1 : 1)] = 0;
        }
        if (move.getType() == Figure.KING) {
            if (Math.abs(destCol - srcCol) == 2) {
                // castle
                if (whiteMove) {
                    if (destCol == 2) {
                        figures[0][0] = 0;

```

