EU3 - Extra Exercise 3

ID1018

October 22, 2014

Inheritance, class hierarchies and polymorphism

The representation of chess pieces

A) The chess-board and the pieces

A chess-board consists of 8 by 8 squares.

A particular square on the chess-board is identified by its row and column. The rows can be designated by the letters a, b, c, d, e, f, g, and h, and the columns with the digits 1, 2, 3, 4, 5, 6, 7, and 8. In this notation the square a4 is found in row a and column 4. A square, which from now on will be called a *field*, is a container for a chess piece. A chess piece can be placed on the field, or be removed from it. A field is also a carrier of information: it can be marked or unmarked.

A number of chess pieces belong to the chess-board. A piece can enter the board on a certain field, or be removed from the board. When it is on the board, the piece can mark all the fields it can reach in a single move. The piece can also remove the markings. A piece can find out if its on the board or not.

The pieces are distinguished by their colour (white or black) and their name. The names of the pieces are Pawn, Rook, Knight, Bishop, Queen, and King.

A model of a chess-board and the pieces is to be created.

A model of a chess-board and pieces — incomplete

```
public class Chessboard
{
    public static class Field
```

```
{
    private char    row;
    private byte    column;
    private Chesspiece    piece = null;
    private boolean    marked = false;

    public Field (char row, byte column) {}

    public void put (Chesspiece piece) {}

    public Chesspiece take () {}

    public void mark () {}

    public void unmark ()  {}

    public String toString ()
    {
        String    s = (marked)? "xx" : "--";
        return (piece == null)? s : piece.toString ();
    }
}


public static final int    NUMBER_OF_ROWS = 8;
public static final int    NUMBER_OF_COLUMNS = 8;

public static final int    FIRST_ROW = 'a';
public static final int    FIRST_COLUMN = 1;

private Field[][]    fields;

public Chessboard ()
{
    fields = new Field[NUMBER_OF_ROWS][NUMBER_OF_COLUMNS];
    char    row = 0;
    byte    column = 0;
    for (int r = 0; r < NUMBER_OF_ROWS; r++)
    {
        row = (char) (FIRST_ROW + r);
        column = FIRST_COLUMN;
        for (int c = 0; c < NUMBER_OF_COLUMNS; c++)
        {
            fields[r][c] = new Field (row, column);
            column++;
        }
    }
}

public String toString ()  {}

public boolean isValidField (char row, byte column)  {}
```

```
public abstract class Chesspiece
{
    private char    color;
    // w - white, b - black

    private char    name;
    // K - King, Q - Queen, R - Rook, B - Bishop, N - Knight,
    // P     Pawn

    protected char    row = 0;
    protected byte    column = -1;

    protected Chesspiece (char color, char name)  {}

    public String toString ()
    {
        return "" + color + name;
    }

    public boolean isOnBoard ()
    {
        return Chessboard.this.isValidField (row, column);
    }

    public void moveTo (char row, byte column)
    throws NotValidFieldException
    {
        if (!Chessboard.this.isValidField (row, column))
               throw new NotValidFieldException
               ("bad field: " + row + column );

        this.row = row;
        this.column = column;

        int    r = row - FIRST_ROW;
        int    c = column - FIRST_COLUMN;
        Chessboard.this.fields[r][c].put (this);
    }

    public void moveOut ()  {}

    public abstract void markReachableFields ();

    public abstract void unmarkReachableFields ();
}


public class Pawn extends Chesspiece
{
    public Pawn (char color, char name)
    {
        super (color, name);
    }
```

```
        public void markReachableFields ()
        {
            byte    col = (byte) (column + 1);
            if (Chessboard.this.isValidField (row, col))
            {
                int    r = row - FIRST_ROW;
                int    c = col - FIRST_COLUMN;
                Chessboard.this.fields[r][c].mark ();
            }
        }

        public void unmarkReachableFields ()
        {
            byte    col = (byte) (column + 1);
            if (Chessboard.this.isValidField (row, col))
            {
                int    r = row - FIRST_ROW;
                int    c = col - FIRST_COLUMN;
                Chessboard.this.fields[r][c].unmark ();
            }
        }
    }


    public class Rook extends Chesspiece  {}

    public class Knight extends Chesspiece  {}

    public class Bishop extends Chesspiece  {}

    public class Queen extends Chesspiece  {}

    public class King extends Chesspiece  {}
}
```

Exercises on the chess-board and pieces

1. Complete the definition class `Chessboard` and all classes inside it. Create the exception class `NotValidFieldException`. Describe the classes and their members.

2. Why is the class `Field` a nested class? Can it be created outside of `Chessboard`?

3. Create a simple test program where an object of class `Chessboard` and several objects of the classes that represent pieces are created and used.

B) The presentation of the pieces

The program `ReachableFieldsOnChessboard` creates a chess-board and a number of pieces. It is done like this:

```
Chessboard    chessBoard = new Chessboard ();
System.out.println (chessBoard + "\n");

Chessboard.Chesspiece[]  pieces = new Chessboard.Chesspiece[6];
pieces[0] = chessBoard.new Pawn ('w', 'P');
pieces[1] = chessBoard.new Rook ('b', 'R');
pieces[2] = chessBoard.new Queen ('w', 'Q');
pieces[3] = chessBoard.new Bishop ('w', 'B');
pieces[4] = chessBoard.new King ('b', 'K');
pieces[5] = chessBoard.new Knight ('w', 'N');
```

Then the pieces introduce themselves. A piece enters the chess-board (on a random field) and marks all the fields it can reach in a single move. The piece waits a moment and then removes the markings. Finally the piece steps off the chess-board to make room for the next piece.

The chess-board is shown in each presentation. When a white knight enters the board and marks its reachable fields the board may look like this:

```
   1  2  3  4  5  6  7  8
a  -- xx -- -- -- xx -- --
b  -- -- -- wN -- -- -- --
c  -- xx -- -- -- xx -- --
d  -- -- xx -- xx -- -- --
e  -- -- -- -- -- -- -- --
f  -- -- -- -- -- -- -- --
g  -- -- -- -- -- -- -- --
h  -- -- -- -- -- -- -- --
```

Exercises on the piece presentations

1. Create and test the program `ReachableFieldsOnChessboard`.

2. The pieces are stored in a common vector despite being of different types. How is this possible? What would the program look like without this possibility?

3. Despite differences in their behaviour, the pieces can present themselves in a common way — in a loop. Why is this possible? Is there any alternative to this?