Arv, klasshierarkier och polymorfism

Schackpjäsernas presentationer

A) Ett schackbräde och schackpjäserna

Ett schackbräde består av 8 x 8 fält.

Ett fält på ett schackbräde är bestämt med sin rad och sin kolumn. Raderna kan betecknas med bokstäverna a, b, c, d, e, f, g och h, och kolumnerna med siffrorna 1, 2, 3, 4, 5, 6, 7 och 8. Fältet a4 finns i så fall i raden a och kolumnen 4. Ett fällt är en behållare för en pjäs: en pjäs kan ställas på fältet, eller tas av det. Ett fält är också en informationsbärare: det går att markera det eller att ta bort markeringen.

Till ett schackbräde hör ett antal schackpjäser. En schackpjäs kan stiga på brädet, på ett visst fält, eller stiga av det. När den väl är på brädet, kan den markera alla de fält som den kan nå med ett enda steg. Den kan även ta bort markeringarna. En pjäs kan ta reda på om den är på brädet eller utanför den.

Pjäserna skiljer sig från varandra enligt sin färg – vit eller svart, och enligt sitt namn. En pjäs är antingen en bonde (eng. *Pawn*), ett torn (eng. *Rook*), en springare (eng. *Knight*), en löpare (eng. *Bishop*), en dam (eng. *Queen*) eller en kung (eng. *King*).

En modell av ett schackbräde och schackpjäserna ska skapas.

En modell av ett schackbräde och schackpjäserna – ej fullständig

```
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  {}
}
```

Uppgifter i samband med schackbrädet och schackpjäserna

1. Komplettera definitionsklassen `Chessboard` och alla klasser inuti den. Skapa undantagsklassen `NotValidFieldException`. Beskriv klasserna och deras medlemmar.

2. Varför skapas klassen `Field` som en nästlad klass? Kan den skapas utanför klassen `Chessboard`?

Varför skapas de klasser som representerar pjäserna som inre klasser? Kan de skapas utanför klassen `Chessboard`?

3. Skapa ett enkelt testprogram, där ett objekt av klassen `Chessboard` och flera objekt av de klasser som representerar pjäserna skapas och används.

B) Pjäsernas presentationer

Programmet `ReachableFieldsOnChessboard` skapar ett schackbräde och ett antal pjäser. Detta görs så här:

```
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');
```

Sedan presenterar sig var och en av pjäserna. En pjäs stiger på schackbrädet – på ett slumpmässigt fält, och markerar alla de fält som den kan nå med ett enda steg. Pjäsen väntar en stund, och därefter tar bort markeringarna. Till sist stiger pjäsen av, så att nästa pjäs kan komma och presentera sig.

Schackbrädet visas vid varje presentation. När en vit springare exempelvis stiger på och markerar de fält som den kan nå, kan schackbrädet se ut så här:

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

Uppgifter i samband med pjäsernas presentationer

1. Skapa och prova programmet `ReachableFieldsOnChessboard`.

2. Pjäserna kan lagras i en gemensam vektor, trots att de är av olika typer. Varför är det möjligt? Hur skulle programmet se ut utan denna möjlighet?

3. Trots skillnader i deras beteenden, kan pjäserna presentera sig på ett gemensamt sätt – i en loop. Varför är det möjligt? Finns det något alternativ till detta?