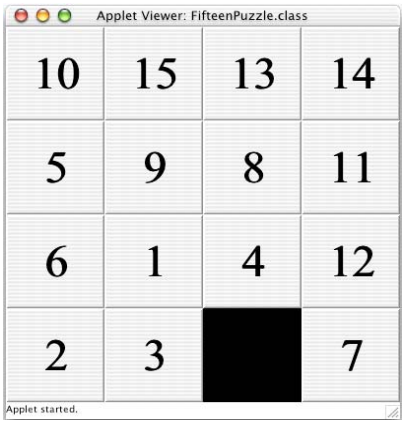
Ekstraopgave 4



Løsningen på opgaven udgøres af en applet, hvor de enkelte brikker er repræsenteret som knapper på brugergrænsefladen.

Det tomme felt er repræsenteret som en knap med sort baggrundsfarve. Til denne knap er knyttet teksten "16".

Et træk foretages ved at ombytte to brikkers tilknyttede tekst og baggrundsfarve.

En udgangsstilling bestemmes ved tilfældig blanding af knapperne, dog under hensyntagen til, at udgangsstillingen skal kunne løses. En simpel metode hertil er ud fra løsningsstillingen at foretage et antal ombytninger af det tomme felt med en af dets nabobrikker. Denne metode er implementeret i metoden `shuffle(int swaps)`.

En mere effektiv blandingsmetode er implementeret i metoden `shuffle()`. Ved denne metode placeres de 15 brikker tilfældigt på brættet. Derefter undersøges det, om den genererede stilling kan løses. Det kan den i halvdelen af tilfældene. I modsat fald ombyttes den sidste brik med den tredjesidste brik (på figuren brik "7" med brik "3".

Metoden til at afgøre, om en stilling kan løses, kan beskrives kort således.

- Bestem for hvert felt, hvor mange felter efter dette felt, der har et nummer, der er mindre end feltets eget nummer. Summen af disse resultater udgør antallet af *inversioner*.
- Farvelæg brættet som et skakbræt med skiftevis sorte og hvide felter. Start med et hvidt felt i øverste venstre hjørne.

Stillingen kan løses, hvis det tomme felt er på et hvidt felt, og antallet af inversioner er lige – eller, hvis det tomme felt er på et sort felt, og antallet af inversioner er ulige.

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;
import java.applet.*;

class Square extends JButton implements ActionListener {
    Square[] neighbor = new Square[4];

    Square(String label) {
        super(label);
        setFont(new Font("Serif", Font.BOLD, 48));
        setFocusPainted(false);
        if (label.equals("16"))
            setBackground(Color.black);
        setBorder(BorderFactory.createRaisedBevelBorder());
        addActionListener(this);
    }

    boolean isEmpty() {
        return getText().equals("16");
    }

    void swap(JButton b) {
        String tempText = getText();
        setText(b.getText());
        b.setText(tempText);
        Color tempColor = getBackground();
        setBackground(b.getBackground());
        b.setBackground(tempColor);
    }

    int getNumber() {
        return Integer.parseInt(getText());
    }

    public void actionPerformed(ActionEvent e) {
        for (int i = 0; i < 4; i++) {
            Square n = neighbor[i];
            if (n != null && n.isEmpty())) {
                swap(n);
                n.repaint();
                repaint();
                if (FifteenPuzzle.isSolved())
                    FifteenPuzzle.successSound.play();
                return;
           }
        }
        FifteenPuzzle.errorSound.play();
    }
}
```

```
public class FifteenPuzzle extends JApplet {
    static Square[] board = new Square[16];
    static AudioClip errorSound, successSound,
                     startSound, stopSound;

    public void init() {
        getContentPane().setLayout(new GridLayout(4, 4));
        for (int i = 0; i < 16; i++)
            board[i] = new Square(Integer.toString(i + 1));
        for (int i = 0; i < 16; i++) {
            if (i % 4 > 0) board[i].neighbor[0] = board[i - 1];
            if (i % 4 < 3) board[i].neighbor[1] = board[i + 1];
            if (i >= 4)    board[i].neighbor[2] = board[i - 4];
            if (i < 12)    board[i].neighbor[3] = board[i + 4];
            getContentPane().add(board[i]);
        }
        errorSound =   getAudioClip(getCodeBase(),
                                    "danger,danger...!.au");
        successSound = getAudioClip(getCodeBase(), "yahoo1.au");
        startSound =   getAudioClip(getCodeBase(), "hello.au");
        stopSound =    getAudioClip(getCodeBase(), "goodbye.au");
        setSize(400, 400);
        shuffle();
    }

    public void start() { startSound.play(); }

    public void stop() { stopSound.play(); }

    public static boolean isSolved() {
        for (int i = 0; i < 16; i++)
            if (board[i].getNumber() != i + 1)
                return false;
        return true;
    }

    void shuffle() {
        for (int i = 1; i < 16; i++) {
             int j = (int) (Math.random() * (i + 1));
             board[i].swap(board[j]);
         }
         if (!isSolvable())
             board[13].swap(board[15]);
    }

    boolean isSolvable() {
        int inversions = 0;
        for (int i = 0; i < 15; i++)
            for (int j = i + 1; j < 16; j++)
                if (board[i].getNumber() > board[j].getNumber())
                    inversions++;
        int i;
        for (i = 0; i < 16; i++)
            if (board[i].isEmpty())
                break;
        return (inversions % 2 == (i + i / 4) % 2);
    }
```

```
    void shuffle(int swaps) {
        Square emptySquare = null;
        for (int i = 0; i < 16; i++) {
            if (board[i].isEmpty()) {
                emptySquare = board[i];
                break;
            }
        }
        for (int t = 0; t < swaps; t++) {
            int k = 0;
            for (int i = 0; i < 4; i++)
                if (emptySquare.neighbor[i] != null)
                    k++;
            k = (int) (Math.random() * k);
            for (int i = 0; i < 4; i++) {
                if (emptySquare.neighbor[i] != null && k-- == 0) {
                    emptySquare.neighbor[i].swap(emptySquare);
                    break;
                }
            }
        }
    }
}
```