src\ImageEditorPanel.java

```java
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.io.File;
import javax.imageio.ImageIO;
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Dimension;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.JPanel;

@SuppressWarnings("serial")

public class ImageEditorPanel extends JPanel implements KeyListener {

    Color[][] pixels;

    public ImageEditorPanel() {
        BufferedImage imageIn = null;
        try {
            imageIn = ImageIO.read(new File("City.jpg"));
        } catch (IOException e) {
            System.out.println(e);
            System.exit(1);
        }
        pixels = makeColorArray(imageIn);
        setPreferredSize(new Dimension(pixels[0].length, pixels.length));
        setBackground(Color.BLACK);
        addKeyListener(this);
    }

    public void paintComponent(Graphics g) {
        for (int row = 0; row < pixels.length; row++) {
            for (int col = 0; col < pixels[0].length; col++) {
                g.setColor(pixels[row][col]);
                g.fillRect(col, row, 1, 1);
            }
        }
    }

    public Color[][] makeColorArray(BufferedImage image) {
        int width = image.getWidth();
        int height = image.getHeight();
        Color[][] result = new Color[height][width];
        for (int row = 0; row < height; row++) {
            for (int col = 0; col < width; col++) {
                Color c = new Color(image.getRGB(col, row), true);
                result[row][col] = c;
            }
        }
        return result;
    }

    public Color[][] horizontalFlip(Color[][] oldArr) {
        Color[][] newArr = new Color[oldArr.length][oldArr[0].length];
        for (int r = 0; r < oldArr.length; r++) {
            for (int c = 0; c < oldArr[r].length; c++) {
                newArr[r][newArr[r].length - c - 1] = oldArr[r][c];
            }
        }
        return newArr;
    }

    public Color[][] verticalFlip(Color[][] oldArr) {
        Color[][] newArr = new Color[oldArr.length][oldArr[0].length];
        for (int r = 0; r < oldArr.length; r++) {
            for (int c = 0; c < oldArr[r].length; c++) {
                newArr[newArr.length - r - 1][c] = oldArr[r][c];
            }
        }
        return newArr;
    }

    public Color[][] grayscale(Color[][] oldArr) {
        final int NUM_COLORS = 3;
        Color[][] newArr = new Color[oldArr.length][oldArr[0].length];
        for (int r = 0; r < oldArr.length; r++) {
            for (int c = 0; c < oldArr[r].length; c++) {
                Color col = oldArr[r][c];
                double red = col.getRed();
                double blue = col.getBlue();
                double green = col.getGreen();
                int gray = (int) ((red + blue + green) / NUM_COLORS);
                Color grayColor = new Color(gray, gray, gray);
                newArr[r][c] = grayColor;
            }
        }
        return newArr;
    }

    public Color[][] blur(Color[][] oldArr) {
        int radius = 7;
        int total = 0;
        Color[][] newArr = new Color[oldArr.length][oldArr[0].length];
        for (int r = 0; r < oldArr.length; r++) {
            for (int c = 0; c < oldArr[r].length; c++) {
                int redTotal = 0;
                int greenTotal = 0;
                int blueTotal = 0;
                for (int i = r - radius; i <= r + radius; i++) {
                    for (int j = c - radius; j <= c + radius; j++) {
                        if ((i < oldArr.length) && (i > 0) && (j < oldArr[r].length) && (j >
0)) {
                            Color col = oldArr[i][j];
                            redTotal = redTotal + col.getRed();
                            greenTotal = greenTotal + col.getGreen();
                            blueTotal = blueTotal + col.getBlue();
                            total++;
                        }

                    }
                }
                redTotal = (int) (redTotal / total);
                greenTotal = (int) (greenTotal / total);
                blueTotal = (int) (blueTotal / total);
                Color newColor = new Color(redTotal, greenTotal, blueTotal);
                newArr[r][c] = newColor;
                total = 0;
            }
        }
        return newArr;
    }

    public Color[][] contrast(Color[][] oldArr) {
        final int DIVIDER = 127;
        final double POSITIVE_SHIFT = 1.3;
        final double NEGATIVE_SHIFT = 0.7;
        final int COLOR_MAX = 255;
        Color[][] newArr = new Color[oldArr.length][oldArr[0].length];
        for (int r = 0; r < oldArr.length; r++) {
            for (int c = 0; c < oldArr[r].length; c++) {
                Color col = oldArr[r][c];
                int red = col.getRed();
                int blue = col.getBlue();
                int green = col.getGreen();
                if (red >= DIVIDER) {
                    red = (int) (red * POSITIVE_SHIFT);
                } else {
                    red = (int) (red * NEGATIVE_SHIFT);
                }
                if (red > COLOR_MAX) {
                    red = COLOR_MAX;
                }
                if (green >= DIVIDER) {
                    green = (int) (green * POSITIVE_SHIFT);
                } else {
                    green = (int) (green * NEGATIVE_SHIFT);
                }
                if (green > COLOR_MAX) {
                    green = COLOR_MAX;
                }
                if (blue >= DIVIDER) {
                    blue = (int) (blue * POSITIVE_SHIFT);
                } else {
                    blue = (int) (blue * NEGATIVE_SHIFT);
                }
                if (blue > COLOR_MAX) {
                    blue = COLOR_MAX;
                }
                Color grayColor = new Color(red, green, blue);
                newArr[r][c] = grayColor;
            }
        }
        return newArr;
    }

    public Color[][] posterize(Color[][] oldArr) {
        final Color col1 = new Color(62, 47, 91); // Deep Purple
        final Color col2 = new Color(242, 130, 0); // Orange
        final Color col3 = new Color(243, 239, 224); // Off White
        final Color col4 = new Color(230, 161, 215); // Light Purple
        Color[][] newArr = new Color[oldArr.length][oldArr[0].length];
        for (int r = 0; r < oldArr.length; r++) {
            for (int c = 0; c < oldArr[r].length; c++) {
                Color col = oldArr[r][c];
                int d1 = (int) (Math.sqrt(Math.pow((col.getRed() - col1.getRed()), 2)
                        + Math.pow((col.getGreen() - col1.getGreen()), 2)
                        + Math.pow((col.getBlue() - col1.getBlue()), 2)));
                int d2 = (int) (Math.sqrt(Math.pow((col.getRed() - col2.getRed()), 2)
                        + Math.pow((col.getGreen() - col2.getGreen()), 2)
                        + Math.pow((col.getBlue() - col2.getBlue()), 2)));
                int d3 = (int) (Math.sqrt(Math.pow((col.getRed() - col3.getRed()), 2)
                        + Math.pow((col.getGreen() - col3.getGreen()), 2)
                        + Math.pow((col.getBlue() - col3.getBlue()), 2)));
                int d4 = (int) (Math.sqrt(Math.pow((col.getRed() - col4.getRed()), 2)
                        + Math.pow((col.getGreen() - col4.getGreen()), 2)
                        + Math.pow((col.getBlue() - col4.getBlue()), 2)));
                int color = Math.min(Math.min(d1, d2), Math.min(d3, d4));
                if (color == d1) {
                    newArr[r][c] = col1;
                }
                if (color == d2) {
                    newArr[r][c] = col2;
                }
                if (color == d3) {
                    newArr[r][c] = col3;
                }
                if (color == d4) {
                    newArr[r][c] = col4;
                }
            }
        }
        return newArr;
    }

    public Color[][] vintage(Color[][] oldArr) {
        final int BRGHTNESS = 1;
        final double CONTRAST = 3.5;
        final int COLOR_MAX = 255;
        final int TINT = 50;
        Color[][] newArr = new Color[oldArr.length][oldArr[0].length];
        for (int r = 0; r < oldArr.length; r++) {
            for (int c = 0; c < oldArr[r].length; c++) {
                Color col = oldArr[r][c];
                int red = (int) CONTRAST * col.getRed() + BRGHTNESS + TINT;
                int green = (int) CONTRAST * col.getGreen() + BRGHTNESS;
                int blue = (int) CONTRAST * col.getBlue() + BRGHTNESS;
                if (red > COLOR_MAX) {
                    red = COLOR_MAX;
                }
                if (green > COLOR_MAX) {
                    green = COLOR_MAX;
                }
                if (blue > COLOR_MAX) {
                    blue = COLOR_MAX;
                }
                Color grayColor = new Color(red, green, blue);
                newArr[r][c] = grayColor;
            }
        }
        return newArr;
    }

    @Override
    public void keyTyped(KeyEvent e) {
        if (e.getKeyChar() == 'p') {
            pixels = posterize(pixels);
        }
        if (e.getKeyChar() == 'c') {
            pixels = contrast(pixels);
        }
        if (e.getKeyChar() == 'b') {
            pixels = blur(pixels);
        }
        if (e.getKeyChar() == 'h') {
            pixels = horizontalFlip(pixels);
        }
        if (e.getKeyChar() == 'j') {
            pixels = verticalFlip(pixels);
        }
        if (e.getKeyChar() == 'g') {
            pixels = grayscale(pixels);
        }
        if (e.getKeyChar() == 'v') {
            pixels = vintage(pixels);
        }
        repaint();
    }

    @Override
    public void keyPressed(KeyEvent e) {
        // unused
    }

    @Override
    public void keyReleased(KeyEvent e) {
        // unused
    }
}
```