src\ImageEditorPanel.java

```java
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.io.File;
import javax.imageio.ImageIO;
import java.awt.*;
import javax.swing.*;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

public class ImageEditorPanel extends JPanel implements KeyListener {
    boolean quit = false;

    Color[][] pixels;

    public ImageEditorPanel() {
        BufferedImage imageIn = null;
        try {
            // the image should be in the main project folder, not in \src or \bin
            imageIn = ImageIO.read(new File("Barack.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) {
        // paints the array pixels onto the screen
        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 void run() {
        while(!quit) {
            repaint();
        }
       pixels = flipHorizontal(pixels);
        repaint();
    }

    public Color[][] contrast(Color[][] inputArr) {
        final int MIDDLE_NUM = 127;
        final double CONTRAST = 0.5;
        int newRed = 0;
        int newGreen = 0;
        int newBlue = 0;
        Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
        for(int row = 0; row < inputArr.length; row++) {
            for(int col = 0; col < inputArr[0].length; col++) {
                Color c = inputArr[row][col];
                int red = c.getRed();
                int green = c.getGreen();
                int blue = c.getBlue();
                if(red <= MIDDLE_NUM) {
                    newRed = (int)(red * CONTRAST);
                } else {
                    newRed = (int)(red + (255 - red) * (CONTRAST));
                }
                if(green <= MIDDLE_NUM) {
                    newGreen = (int)(green * CONTRAST);
                } else {
                    newGreen =(int)(green + (255 - green) * (CONTRAST));
                }
                if(blue <= MIDDLE_NUM) {
                    newBlue = (int)(blue * CONTRAST);
                } else {
                    newBlue = (int)(blue + (255 - blue) * (CONTRAST));
                }
                outputArr[row][col] = new Color(newRed,newGreen,newBlue);
            }
        }
        return outputArr;
    }

    public Color[][] postarizeFilter(Color[][] inputArr) {
        Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
        Color[] palette = {
            new Color(114,219,139),
            new Color(104,151,227),
            new Color(217,195,230),
            new Color(26,4,2),
        };
        for(int row = 0; row < inputArr.length; row++) {
            for(int col = 0; col < inputArr[0].length; col++) {
                Color c = inputArr[row][col];
                Color newC = getNearestColor(c,palette);
                outputArr[row][col] = newC;
            }
        } return outputArr;
    }

```

```java
    public Color getNearestColor(Color c,Color[] palette) {
        Color nearest = null;
        double min = Integer.MAX_VALUE;
        for(Color p: palette) {
            int dRed = c.getRed() - p.getRed();
            int dGreen = c.getGreen() - p.getGreen();
            int dBlue = c.getBlue() - p.getBlue();
            double loss = Math.sqrt(Math.pow(dRed,2) + Math.pow(dGreen,2) + Math.pow(dBlue,2));
            if (loss < min) {
                min = loss;
                nearest = p;
            }
        }
        return nearest;
    }

    public Color[][] colorNeg(Color[][] inputArr) {
        Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
        for(int row = 0; row < inputArr.length; row++) {
            for(int col = 0; col < inputArr[0].length; col++) {
                Color c = inputArr[row][col];
                int red = 255 - c.getRed();
                int green = 255 - c.getGreen();
                int blue = 255 - c.getBlue();
                Color newC = new Color(red, green, blue);
                outputArr[row][col] = newC;
            }
        }
        return outputArr;
    }

    public Color[][] grayScale(Color[][] inputArr) {
        Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
        for(int row = 0; row < inputArr.length; row++) {
            for(int col = 0; col < inputArr[0].length; col++) {
                Color c = inputArr[row][col];
                int red = c.getRed();
                int green = c.getGreen();
                int blue = c.getBlue();
                int average = (red + green + blue) / 3;
                Color newC = new Color(average, average, average);
                outputArr[row][col] = newC;
            }
        }
        return outputArr;
    }

    //Multiple - pixel algorithm template
    public Color[][] multiPixelAlgo(Color[][] inputArr) {
        final int RADIUS = 3;
        Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
        for(int row = 0; row < inputArr.length; row++) {
            for(int col = 0; col < inputArr[0].length; col++) {
                int pixelNum = 0;
                int totalRed = 0;
                int totalGreen = 0;
                int totalBlue = 0;
                // initialize some variables
                // visit the neighbors centered cat row, cal
                for(int row2 = row - RADIUS; row2 <= row+ RADIUS; row2++) {
                    for(int col2 = col - RADIUS; col2 <= col + RADIUS; col2++) {
                        if(row2 >= 0 && row2 < inputArr.length && col2 >= 0 && col2 <
inputArr[0].length) {
                            //do some work with this neighbor
                            Color c = inputArr[row2][col2];
                            totalRed += c.getRed();
                            totalGreen += c.getGreen();
                            totalBlue += c.getBlue();
                            pixelNum++;
                        }
                    }
                }
                int avgRed = totalRed / pixelNum;
                int avgGreen = totalGreen / pixelNum;
                int avgBlue = totalBlue / pixelNum;
                Color newC = new Color(avgRed,avgGreen,avgBlue);
                outputArr[row][col] = newC;
            }
        }
        return outputArr;
    }

    //Single - pixel algorithm template
    public Color[][] singlePixelAlgo(Color[][] inputArr) {
        Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
        for(int row = 0; row < inputArr.length; row++) {
            for(int col = 0; col < inputArr[0].length; col++) {
                Color c = inputArr[row][inputArr[0].length - col - 1];
                //based on the values of Color c, create a new color
                outputArr[row][col] = c;
            }
        }
        return outputArr;
    }

    public Color[][] flipHorizontal(Color[][] inputArr) {
        Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
        for(int row = 0; row < inputArr.length; row++) {
            for(int col = 0; col < inputArr[0].length; col++) {
                Color c = inputArr[row][inputArr[0].length - col - 1];
                outputArr[row][col] = c;
            }
        }
        return outputArr;
    }

    public Color[][] flipVertical(Color[][] inputArr) {
        Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
        for(int row = 0; row < inputArr.length; row++) {
            for(int col = 0; col < inputArr[0].length; col++) {
                Color c = inputArr[inputArr.length - row - 1][col];
                outputArr[row][col] = c;
            }
        }
        return outputArr;
    }

    public Color[][] brighten(Color[][] inputArr) {
        Color[][] outputArr = new Color[inputArr.length][inputArr[0].length];
        for(int row = 0; row < inputArr.length; row++) {
            for(int col = 0; col < inputArr[0].length; col++) {
                Color c = inputArr[row][col];
                outputArr[row][col] = c.brighter();
            }
        }
        return outputArr;
    }

    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;
            }
        }
        // System.out.println("Loaded image: width: " +width + " height: " + height);
        return result;
    }

    public void keyPressed(KeyEvent e) {
        //unused
    }
    public void keyReleased(KeyEvent e) {
        //unused
    }
    public void keyTyped(KeyEvent e) {
        if (e.getKeyChar() == 'v') {
           pixels = flipVertical(pixels);
        }
        if (e.getKeyChar() == 'h') {
           pixels = flipHorizontal(pixels);
        }
        if (e.getKeyChar() == 'p') {
          pixels = postarizeFilter(pixels);
        }
        if (e.getKeyChar() == 'n') {
            pixels = colorNeg(pixels);
        }
        if (e.getKeyChar() == 'c') {
            pixels = contrast(pixels);
        }
        if (e.getKeyChar() == 'a') {
            pixels = brighten(pixels);
        }
        if (e.getKeyChar() == 'g') {
            pixels = grayScale(pixels);
        }
        if (e.getKeyChar() == 'b') {
            pixels = multiPixelAlgo(pixels);
        }
    }
}

```