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 java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import javax.swing.*;

public class ImageEditorPanel extends JPanel implements KeyListener {

    Color[][] pixels;
    final static int MAX_COLOR_VAL = 255;
    final int BLUR_RADIUS = 5;
    final double CONTRAST_FACTOR = 1.1;
    final double BLUE_FACTOR = 1.2;

    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);
        setFocusable(true);
        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() {
        // call your image-processing methods here OR call them from keyboard event
        // handling methods
        // pixels = flipHoriz(pixels);
        // pixels = flipVert(pixels);
        // pixels = grayscale(pixels);
        // pixels = vintage(pixels);
        // pixels = blur(pixels, BLUR_RADIUS);
        // pixels = blueTint(pixels, BLUE_FACTOR);
        // pixels = contrast(pixels, CONTRAST_FACTOR);
        repaint();
    }
```

```
55
56      public Color[][] makeColorArray(BufferedImage image) {
57          int width = image.getWidth();
58          int height = image.getHeight();
59          Color[][] result = new Color[height][width];
60
61          for (int row = 0; row < height; row++) {
62              for (int col = 0; col < width; col++) {
63                  Color c = new Color(image.getRGB(col, row), true);
64                  result[row][col] = c;
65              }
66          }
67          // System.out.println("Loaded image: width: " +width + " height: " + height);
68          return result;
69      }
70
71      public Color[][] flipHoriz(Color[][] oldImg) {
72          int rows = oldImg.length;
73          int cols = oldImg[0].length;
74          Color[][] horizImg = new Color[rows][cols];
75          for (int r = 0; r < rows; r++) {
76              for (int c = 0; c < cols; c++) {
77                  int flipCol = cols - 1 - c;
78                  horizImg[r][flipCol] = oldImg[r][c];
79              }
80          }
81          return horizImg;
82      }
83
84      public Color[][] flipVert(Color[][] oldImg) {
85          int rows = oldImg.length;
86          int cols = oldImg[0].length;
87          Color[][] vertImg = new Color[rows][cols];
88          for (int r = 0; r < rows; r++) {
89              int flipRow = rows - 1 - r;
90              for (int c = 0; c < cols; c++) {
91                  vertImg[flipRow][c] = oldImg[r][c];
92              }
93          }
94          return vertImg;
95      }
96
97      // applies a grayscale to the image.
98      public Color[][] grayscale(Color[][] oldImg) {
99          final int DIVISOR = 3;
100         int rows = oldImg.length;
101         int cols = oldImg[0].length;
102         Color[][] grayImg = new Color[rows][cols];
103         for (int r = 0; r < rows; r++) {
104             for (int c = 0; c < cols; c++) {
105                 Color pixel = oldImg[r][c];
106                 int red = pixel.getRed();
107                 int green = pixel.getGreen();
108                 int blue = pixel.getBlue();
109                 int grayValue = (red + green + blue) / DIVISOR;
110
111                 Color gray = new Color(grayValue, grayValue, grayValue);
```

```
                grayImg[r][c] = gray;
            }
        }
        return grayImg;
    }

    public Color[][] blueTint(Color[][] oldImg, double blueFactor) {
        int rows = oldImg.length;
        int cols = oldImg[0].length;
        Color[][] blueImg = new Color[rows][cols];

        // Iterate each pixel and apply blue tint
        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                Color pixel = oldImg[r][c];
                int newBlue = (int) (pixel.getBlue() * blueFactor);
                newBlue = Math.min(MAX_COLOR_VAL, Math.max(0, newBlue));
                blueImg[r][c] = new Color(pixel.getRed(), pixel.getGreen(), newBlue);
            }
        }
        return blueImg;
    }

    // applys a vintage effect to photo
    public static Color[][] vintage(Color[][] oldImg) {
        int rows = oldImg.length;
        int cols = oldImg[0].length;
        Color[][] vintageImg = new Color[rows][cols];

        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
                Color pixel = oldImg[r][c];
                int red = pixel.getRed();
                int green = pixel.getGreen();
                int blue = pixel.getBlue();
                // values from wikipedia
                int vintageRed = (int) ((0.393 * red) + (0.769 * green) + (0.189 * blue));
                int vintageGreen = (int) ((0.349 * red) + (0.686 * green) + (0.168 * blue));
                int vintageBlue = (int) ((0.272 * red) + (0.534 * green) + (0.131 * blue));
                // Check values are in color range
                vintageRed = Math.min(MAX_COLOR_VAL, Math.max(0, vintageRed));
                vintageGreen = Math.min(MAX_COLOR_VAL, Math.max(0, vintageGreen));
                vintageBlue = Math.min(MAX_COLOR_VAL, Math.max(0, vintageBlue));
                vintageImg[r][c] = new Color(vintageRed, vintageGreen, vintageBlue);
            }
        }
        return vintageImg;
    }

    // changes contrast of the image
    public static Color[][] contrast(Color[][] oldImg, double factor) {
        int rows = oldImg.length;
        int cols = oldImg[0].length;
        Color[][] contrastImg = new Color[rows][cols];

        for (int r = 0; r < rows; r++) {
            for (int c = 0; c < cols; c++) {
```

```
                Color pixel = oldImg[r][c];
                // centers the original value by - half of color range
                // then multiplys by contrast factor
                // then adds back half of the color range
                int red = (int) (factor * (pixel.getRed() - MAX_COLOR_VAL / 2) +
MAX_COLOR_VAL / 2);
                int green = (int) (factor * (pixel.getGreen() - MAX_COLOR_VAL / 2) +
MAX_COLOR_VAL / 2);
                int blue = (int) (factor * (pixel.getBlue() - MAX_COLOR_VAL / 2) +
MAX_COLOR_VAL / 2);
                // makes sure new values are in rgb range
                red = Math.min(MAX_COLOR_VAL, Math.max(0, red));
                green = Math.min(MAX_COLOR_VAL, Math.max(0, green));
                blue = Math.min(MAX_COLOR_VAL, Math.max(0, blue));
                contrastImg[r][c] = new Color(red, green, blue);
            }
        }
        return contrastImg;
    }

    // adds blur effect to image
    // uses two helper methods
    public static Color[][] blur(Color[][] oldImg, int blurRadius) {
        int rows = oldImg.length;
        int cols = oldImg[0].length;
        Color[][] blurImg = new Color[rows][cols];
        for (int r = blurRadius; r < rows - blurRadius; r++) {
            for (int c = blurRadius; c < cols - blurRadius; c++) {
                Color[] neighbors = calcSquareNeighbors(oldImg, r, c, blurRadius);
                Color avgColor = calcAvgColor(neighbors);
                blurImg[r][c] = avgColor;
            }
        }
        return blurImg;
    }

    // helper method
    public static Color[] calcSquareNeighbors(Color[][] pixels, int centerRow, int centerCol,
int radius) {
        // Calc total number of pixels in the square neighborhood
        Color[] neighbors = new Color[(2 * radius + 1) * (2 * radius + 1)];
        int index = 0;
        // Negative sign for radius to do a full iteration over neighborhood of current
        // pixel
        for (int r = -radius; r <= radius; r++) {
            for (int c = -radius; c <= radius; c++) {
                neighbors[index++] = pixels[centerRow + r][centerCol + c];
            }
        }
        return neighbors;
    }

    // Helper method to calculate the average color of an array of colors
    // calculates the average color by using RGB values of each pixel
    public static Color calcAvgColor(Color[] colors) {
        int totalRed = 0;
        int totalGreen = 0;
        int totalBlue = 0;
```

```
223         for (Color pixel : colors) {
224             totalRed += pixel.getRed();
225             totalGreen += pixel.getGreen();
226             totalBlue += pixel.getBlue();
227         }
228         int size = colors.length;
229         int avgRed = totalRed / size;
230         int avgGreen = totalGreen / size;
231         int avgBlue = totalBlue / size;
232         avgRed = Math.min(MAX_COLOR_VAL, Math.max(0, avgRed));
233         avgGreen = Math.min(MAX_COLOR_VAL, Math.max(0, avgGreen));
234         avgBlue = Math.min(MAX_COLOR_VAL, Math.max(0, avgBlue));
235 
236         return new Color(avgRed, avgGreen, avgBlue);
237     }
238 
239     @Override
240     public void keyPressed(KeyEvent e) {
241         // the last else if is to reset the image back to original
242         // got the code from above in ImageEditorPanel
243         char keyChar = e.getKeyChar();
244         if (keyChar == 'h') {
245             pixels = flipHoriz(pixels);
246         } else if (keyChar == 'f') {
247             pixels = flipVert(pixels);
248         } else if (keyChar == 'g') {
249             pixels = grayscale(pixels);
250         } else if (keyChar == 'v') {
251             pixels = vintage(pixels);
252         } else if (keyChar == 'b') {
253             pixels = blur(pixels, BLUR_RADIUS);
254         } else if (keyChar == 'c') {
255             pixels = contrast(pixels, CONTRAST_FACTOR);
256         } else if (keyChar == 't') {
257             pixels = blueTint(pixels, BLUE_FACTOR);
258         } else if (keyChar == 'r') {
259             BufferedImage imageIn = null;
260             try {
261                 imageIn = ImageIO.read(new File("CITY.jpg"));
262             } catch (IOException ex) {
263                 System.out.println(ex);
264                 System.exit(1);
265             }
266             pixels = makeColorArray(imageIn);
267         }
268         repaint();
269     }
270 
271     @Override
272     public void keyTyped(KeyEvent e) {
273     }
274 
275     @Override
276     public void keyReleased(KeyEvent e) {
277     }
278 }
279 
```