```
package week6;

import java.awt.Color;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Random;
import java.util.Scanner;
import java.util.Stack;
import javax.imageio.ImageIO;

public class PixelGrid {

	private ArrayList<ArrayList<Color>> grid;
	private Stack<DeletedInfo> command;
	private Random random;

	/**
	 * this constructor stores color values of a buffered image
	 * from a png file and copies them into an arrayList of arrayList of colors
	 *
	 * @param filename
	 */
	public PixelGrid(String filename) {
		// allocate memory for grid

		readPixelGridFromFile(filename);
		this.random = new Random();
	}

	/**
	 * this constructor has a built in seed
	 * in order to test random with a passed in png
	 * @param filename
	 * @param seed
	 */
	public PixelGrid(String filename, int seed) {
		// allocate memory for grid

		readPixelGridFromFile(filename);
		this.random = new Random(seed);

	}

	/**
	 * this helper method makes the constructors less repetitive
	 * it stores color values of a buffered image from a png and copies
	 * it over to a grid
	 * @param filename
	 */
	private void readPixelGridFromFile(String filename) {
		this.grid = new ArrayList<ArrayList<Color>>();
		this.command = new Stack<DeletedInfo>();

		try (Scanner scan = new Scanner(new File(filename))) {
			File originalFile = new File(filename);
			BufferedImage oldImg = ImageIO.read(originalFile);

			for (int y = 0; y < oldImg.getHeight(); y++) { // rows

				ArrayList<Color> row = new ArrayList<Color>(); // makes a
new copy of row eveytime for loop
```

```
                                for (int x = 0; x < oldImg.getWidth(); x++) { // columns

                                        int pixel = oldImg.getRGB(x, y);
                                        // pixel is one value in grid

                                        // Creating a Color object from pixel value
                                        Color originalColor = new Color(pixel);
                                        row.add(originalColor);
                                }
                                grid.add(row);
                        }

                } catch (Exception ex) {
                        System.err.println("out of bounds exception");
                        ex.printStackTrace();
                }
        }

        /**
         * this is a constructor used for testing
         * it is given a grid and a seed(for random testing)
         * and makes a deep copy of and array list of array list of colors
         * @param givenGrid
         * @param seed
         */
        public PixelGrid(ArrayList<ArrayList<Color>> givenGrid, int seed) {

                this.grid = new ArrayList<ArrayList<Color>>();
                this.command = new Stack<DeletedInfo>();
                this.random = new Random(seed);

                for (int i = 0; i < givenGrid.size(); i++) {
                        ArrayList<Color> copy = new ArrayList<Color>();
                        for (int j = 0; j < givenGrid.get(i).size(); j++) {
                                copy.add(givenGrid.get(i).get(j));
                        }
                        grid.add(copy);
                }
        }

        /**
         * this method finds the column in the grid that
         * has the biggest sum of blue value
         * @return
         */
        public int findBluest() {

                int column = 0;
                int bluestCol = 0;

                for (int col = 0; col < grid.get(0).size(); col++) {
                        int blueSum = 0;
                        for (int row = 0; row < grid.size(); row++) {
                                blueSum += grid.get(row).get(col).getBlue();
                        }
                        if (blueSum > bluestCol) {
                                bluestCol = blueSum;
                                column = col;
                        }
                }
                return column;
        }

        /**
         * this method first saves the current command that the user inputted
```

```
	 * into a stack,
	 * then changes the color of a column passed in
	 * and changes the column to all blue or red pixels
	 * based on the boolean input
	 *
	 * @param col
	 * @param isBlue
	 */
	public void changeColor(int col, boolean isBlue) {

		if (!grid.isEmpty()) {
			ArrayList<Color> column = new ArrayList<Color>();

			for (int c = 0; c < grid.get(0).size(); c++) {
				for (int row = 0; row < grid.size(); row++) {
					column.add(grid.get(row).get(col));
				}
			}

			command.push(new DeletedInfo(col, column));

			if (isBlue) {
				for (int i = 0; i < grid.size(); i++) {
					this.grid.get(i).set(col, Color.BLUE);
				}
			} else {
				for (int i = 0; i < grid.size(); i++) {
					this.grid.get(i).set(col, Color.RED);
				}
			}

		}

	}

	/**
	 * this method feeds values of the pixelgrid into a buffered image
	 * then saves it to a particular name based on the boolean
	 * @param counter
	 * @param quit
	 */
	public void save(int counter, boolean quit) {

		BufferedImage newImg = new BufferedImage(grid.get(0).size(), grid.size(),
 BufferedImage.TYPE_INT_RGB);

		for (int x = 0; x < grid.size(); x++) {
			for (int y = 0; y < grid.get(0).size(); y++) {
				newImg.setRGB(y, x, grid.get(x).get(y).getRGB());
			}
		}

		if (quit == false) {
			File newFile = new File("temp_0" + counter + "x.png");
			try {
				ImageIO.write(newImg, "png", newFile);
			} catch (IOException e) {
				// TODO Auto-generated catch block
				e.printStackTrace();
			}
		} else {
			File newFile = new File("newIMG.png");
			try {
				ImageIO.write(newImg, "png", newFile);
			} catch (IOException e) {
```

```
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                }

        }

        /**
         * this method makes chooses a random int, which highlights
         * a random column in another function
         * @return
         */
        public int random() {

                int x = this.random.nextInt(grid.size() - 1);
                return x;
        }

        /**
         * this method pops the previous saved stack of commands
         */
        public void undoDelete() {

                if (!command.isEmpty()) {
                        DeletedInfo prevCommand = command.pop();
                        int indexToAddBack = prevCommand.getIndex();
                        ArrayList<Color> colorsToAddBack = prevCommand.getColumn();

                        int numRow = this.grid.size();

                        for (int r = 0; r < numRow; r++) {
                                this.grid.get(r).add(indexToAddBack,
colorsToAddBack.get(r));
                        }

                        System.out.println("there are no commands to undo");

                }
        }

        /**
         * this method removes a column from the grid
         * @param col
         */
        public void remove(int col) {

                for (int i = 0; i < grid.size(); i++) {
                        this.grid.get(i).remove(col);
                }
        }

        /*
         * this method produces a string representation
         * of a arraylist of arraylist of colors
         *
         * copied from integerGridLL
         */
        @Override
        public String toString() {
                StringBuilder printList = new StringBuilder();

                for (int i = 0; i < grid.size(); i++) {
                        String inner = grid.get(i).toString();
                        printList.append(inner);
                }
```

```
                return printList.toString();
        }

        /*
         * main function to test and print
         */
        public static void main(String[] args) {
                ArrayList<ArrayList<Color>> pg = new ArrayList<ArrayList<Color>>();
                ArrayList<Color> inner1 = new ArrayList<Color>();
                inner1.add(Color.BLUE);
                inner1.add(Color.GREEN);
                inner1.add(Color.GREEN);
                ArrayList<Color> inner2 = new ArrayList<Color>();
                inner2.add(Color.BLUE);
                inner2.add(Color.RED);
                inner2.add(Color.BLUE);
                ArrayList<Color> inner3 = new ArrayList<Color>();
                inner3.add(Color.GREEN);
                inner3.add(Color.RED);
                inner3.add(Color.GREEN);

                pg.add(inner1);
                pg.add(inner2);
                pg.add(inner3);

                System.out.println(pg.toString());

        }

}
```