МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
КАФЕДРА «ЭВМ и системы»

ОТЧЁТ
по лабораторной работе № 2
Увеличение и уменьшение цифровых изображений

Листов **8**

Выполнил студент группы Э-56
Козей Д. А.

Проверил Дубицкий А. В.

Брест 2020

Цель работы: Изучить методы увеличения и уменьшения цифровых изображений и применить полученные знания на практике.

Задание: написать программу способную производить увеличение/уменьшение исходного изображения в нецелое число раз методом билинейной интерполяции. Программа должна сохранять полученное изображение в виде файла формата BMP. С помощью программы уменьшить исходное изображение в 1,2; 3; 7; 21 раз. Полученные изображения затем восстановить до исходного размера и сравнить результаты с исходным изображением

Код программы:

```
package com.company;

import java.io.File;
import java.lang.System;

public class Main {
    static final String PATH_TO_FILE = "D:\\labs\\doggie.
      bmp";
    private static final float[] coefficients = new float
      [] {1.2f, 3f, 7f, 21f};

    public static void main(String[] args) {
        File output = new File(PATH_TO_FILE);
        if(output.exists()) {
            BilinearInterpolation task = new
              BilinearInterpolation(output);
            for (float koef: coefficients) {
                File newFile = task.compress(koef);
                task.resize(koef, newFile);
            }
        } else {
            System.out.println("Enter: correct image path
              ");
        }
```

```
    }
}
```

```
package com.company;

import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.lang.System;

public class BilinearInterpolation {
    private BufferedImage sourcePicture;

    public BilinearInterpolation(File image){
        try {
            sourcePicture = ImageIO.read(image);
        }catch (IOException e){
            System.out.println(e.getMessage());
        }
    }

    public File compress(double koef) {
        int outputWidth = (int) (sourcePicture.getWidth()
          / koef);
        int outputHeight = (int) (sourcePicture.getHeight
          () / koef);
        BufferedImage outputPicture = new BufferedImage(
          outputWidth, outputHeight, BufferedImage.
          TYPE_INT_RGB);
        bilinearInterpolation(outputWidth, outputHeight,
          outputPicture, null);
        try {
```

```
            String coefString;
            if (koef < 10) {
                coefString = String.valueOf(koef).
                  substring(0, 3).replace('.', '_');
            }else {
                coefString = String.valueOf(koef).
                  substring(0, 4).replace('.', '_');
            }
            String pathToNewFile = Main.PATH_TO_FILE.
              substring(0, 18) + coefString + ".bmp";
            File newFile = new File(pathToNewFile);
            if (newFile.exists()) {
                ImageIO.write(outputPicture,"bmp", newFile
                  );
                return newFile;
            }else {
                if (newFile.createNewFile()) {
                    ImageIO.write(outputPicture, "BMP",
                      newFile);
                    return newFile;
                }else {
                    System.out.println("File creation
                      error");
                }
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
        return null;
    }

    public void resize(double koef, File output) {
        BufferedImage newPicture = null;
```

```
        try {
            newPicture = ImageIO.read(output);
        } catch (IOException e) {
            e.printStackTrace();
        }
        if (newPicture != null) {
            int outputWidth = (int) ( newPicture.getWidth
               () * koef);
            int outputHeight = (int) (newPicture.getHeight
               () * koef);
            BufferedImage outputPicture = new
               BufferedImage(outputWidth, outputHeight,
               BufferedImage.TYPE_INT_RGB);
            bilinearInterpolation(outputWidth,
               outputHeight, outputPicture, newPicture);
            try {
                ImageIO.write(outputPicture, "bmp", output
                   );
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    private void bilinearInterpolation(Integer outputWidth
       , Integer outputHeight,
                                       BufferedImage
                                          outputPicture
                                          ,
                                          BufferedImage
```

```
                                                    newPicture
                                                    ) {
BufferedImage picture;
if (newPicture == null) {
    picture = sourcePicture;
}else  {
    picture = newPicture;
}
for (int x=0; x<outputWidth; ++x) {
    for(int y = 0; y < outputHeight; ++y){
        float gx = ((float) x) / outputWidth *  (
          picture.getWidth() - 1);
        float gy = ((float) y) / outputHeight * (
          picture.getHeight() - 1);
        int gxi = (int) gx;
        int gyi = (int) gy;
        int rgb = 0;
        int c00 = picture.getRGB(gxi, gyi);
        int c10 = picture.getRGB(gxi + 1, gyi);
        int c01 = picture.getRGB(gxi, gyi + 1);
        int c11 = picture.getRGB(gxi + 1, gyi + 1)
          ;
        for (int k = 0; k <= 2; ++k) {
            float b00 = get(c00, k);
            float b10 = get(c10, k);
            float b01 = get(c01, k);
            float b11 = get(c11, k);
            int ble = ((int) blerp(b00, b10, b01,
              b11, gx - gxi, gy - gyi)) << (8 * k)
              ;
            rgb = rgb | ble;
        }
        outputPicture.setRGB(x, y, rgb);
```

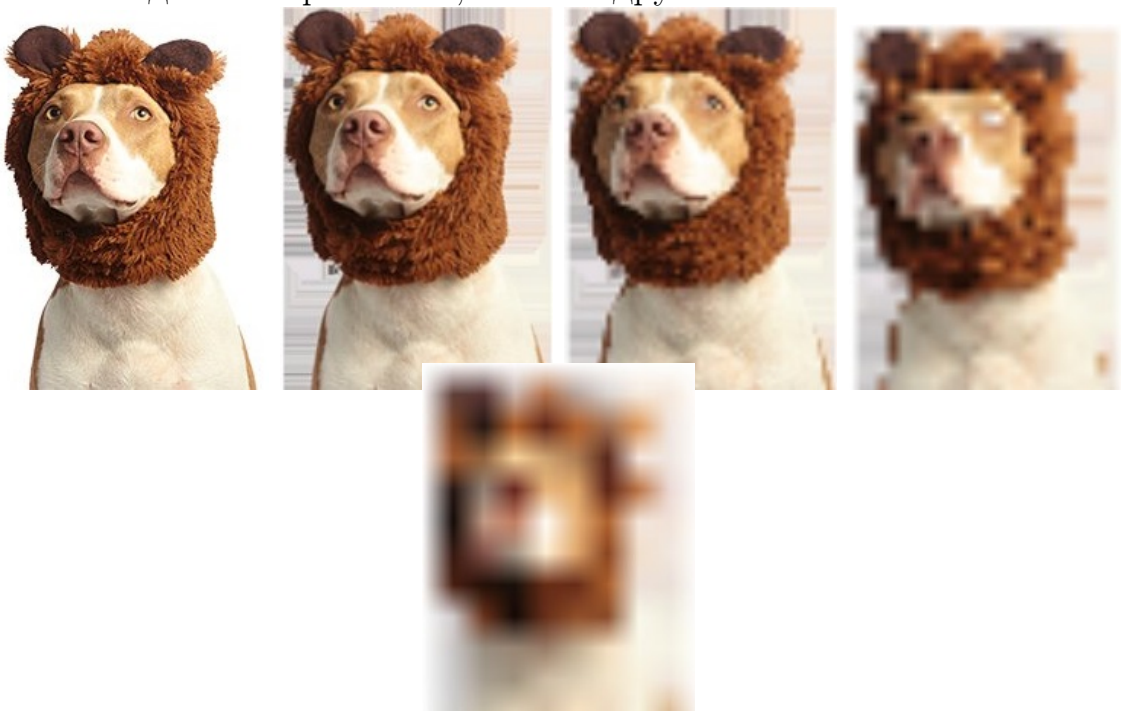
```
            }
        }
    }

    private int get(int self, int n) {
        return (self >> (n * 8)) & 0xFF;
    }

    private float lerp(float s, float e, float t) {
        return s + (e - s) * t;
    }

    private float blerp(final Float c00, float c10, float
      c01, float c11, float tx, float ty) {
        return lerp(lerp(c00, c10, tx), lerp(c01, c11, tx)
          , ty);
    }
}
```

В изображении увеличиваемом/уменьшенном с помощью метода билинейной интерполяции изображение размывается. Ключевая идея метода билинейной интерполяции заключается в том, чтобы провести обычную линейную интерполяцию сначала в одном направлении, затем в другом.



Вывод: Написали программу способную производить увеличение/уменьшение исходного изображения в нецелое число раз методом билинейной интерполяции.