# PROIECT AWJ

# Inversare de Imagini

Iancu George-Alexandru

332AB

2024

## 1. Introducere

Asa cum stim, avem imagini bmp pe 24 de biti, adica avem 3 canale de culoare, RGB, fiecare a cate 8 biti, deci un byte. Pe langa cele 3, mai avem un canal pentru transparenta. O sa vedem in cele ce urmeaza ca acest canal, numit alpha, nu o sa ne intereseza pe noi prea mult.

Asadar, in functie de aplicatia pe care o avem trebuie sa manipulam aceste canale, sau grupari ale acestor canale pentru a obtine efectul dorit.

## 2. Descriere

Aplicatia pe care am ales-o este de inversare a imaginii sau de negativizare a ei. O sa prezint in continuare un scurt exemplu:

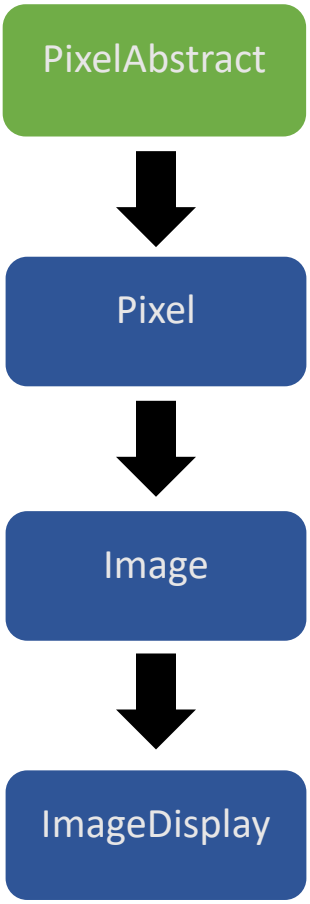
Partea de procesare este destul de simpla, intrucat trebuie sa trecem prin fiecare pixel si sa-l modificam dupa o regula simpla. In locul fiecarei culori din

fiecare pixel, punem diferenta pana la 255, care stim ca este valoarea maxima pe 8 biti.

## 3. Structura

Avem o structura complexa, intrucat am impartit-o in 3 parti.

1. Imaginea practica




### PixelAbstract:

Este clasa de baza a acestei parti. Definim in ea cele 4 atribute ale unui pixel si un atribut static care o sa tina minte cate viitoare obiecte de tip pixel am instantiat. Mai avem pe langa acestea un bloc de initializare si un bloc static de intializare, o metoda abstracta, care sa evidentieze abstractizarea clasei si gettere

si settere pentru toate cele 4 atribute. Atributele sunt private pentru a respecta incapsularea deci avem nevoie de get si set.

```java
public abstract class PixelAbstract {
    // Cele 4 atribute ale unui pixel, cele 3 culori ale sale si alpha, care este
    // transparenta
    // Sunt private pentru a respecta incapsularea
    private int red, green, blue, alpha;
    // Atribut pentru a tine minte cati pixeli am instantiat
    private static int noPixels;

    // Bloc de initializare
    {
        red = 0;
        green = 0;
        blue = 0;
        alpha = 0;
    }
    // Bloc static de initializare pentru variabila statica
    static {
        noPixels = 0;
    }

    // Aratam aici abstractizarea clasei
    public abstract void printPixel();

    // Aici vem gettere si settere individuale pentru fiecare atribut
    public int getRed() {
        return red;
    }
```

## Pixel:

Mosteneste clasa de mai sus si aici avem toate metodele si constructorii de care avem nevoie. Am implementat aici si metodele de mai sus care erau abstracte si o metoda cu varargs, care este statica si care printeaza un numar variabil de pixeli.

```java
public class Pixel extends PixelAbstract {
    // Constructor fara parametri, care seteaza totul pe 0
    public Pixel() {
        setPixel(0, 0, 0, 0);
        setNoPixels(getNoPixels() + 1);
    }

    // Constructorul cu parametri, care apeleza functia de setare a fiecarui atribut
    public Pixel(int red, int green, int blue, int alpha) {
        setPixel(red, green, blue, alpha);
        setNoPixels(getNoPixels() + 1);
    }

    // Functie pentru setare a atributelor, deci un setter mai general
    public void setPixel(int red, int green, int blue, int alpha) {
        setRed(red);
        setGreen(green);
        setBlue(blue);
        setAlpha(alpha);
    }

    // Un getter care returneaza un vector de dimensiune 4, cu toate cele 4 atribute
    public int[] getPixel() {
        int[] vector = new int[3];
        vector[0] = getRed();
        vector[1] = getGreen();
        vector[2] = getBlue();
        vector[3] = getAlpha();
```

```
    vector[1] = getGreen();
    vector[2] = getBlue();
    vector[3] = getAlpha();

    return vector;
}

// Functie pentru printarea celor 4 atribute
public void printPixel() {
    System.out.println(
            "Red: " + getRed() + "\nGreen : " + getGreen() + "\nBlue: " + getBlue() + "\nalpha: " + getAlpha());
}

// Functie pentru inversarea culorii unui pixel
public void invertPixel() {
    this.setPixel(255 - this.getRed(), 255 - this.getGreen(), 255 - this.getBlue(), this.getAlpha());
}

// Functie static de prinatare a mai multor pixeli, care voloseste varargs
public static void printPixels(Pixel... args) {
    int i = 1;
    for (Pixel pixel : args) {
        System.out.println("Pixelul: " + i);
        i++;
        pixel.printPixel();
    }
}
```

## Image:

Avem aceasta clasa cu care lucram cel mai mult unde stocam o imagine propriu-zisa cu ajutorul clasei BufferedImage, dar stocam si o matrice de pixeli pentru a o putea prelucra mult mai usor. Avem constructori in care salvam imagini din fisier, unde avem nevoie de try-catch pentru ca pot aparea erori la citire, precum lipsa unui fisier.

```
public Image(String fileName, boolean matrix) {
    long startTime = System.currentTimeMillis();
    try {
        // Citim imaginea efectiva cu o functie ajutatoare
        BufferedImage imageAux = ImageIO.read(new File(fileName));
        // Aici avem cazul in care facem automat matricea
        if (matrix == true)
            setImageMatrix(imageAux);
        // si casul in care nu salvam si matricea
        else
            setImage(imageAux);

    } catch (IOException e) {
        // Printez exceptia care poate aparea
        System.out.println("Exceptie : " + e.getMessage());
    } finally {
    }
    long endTime = System.currentTimeMillis();
    System.out.println("Citirea din fisierul sursa a durat:  " + (endTime - startTime) + " milliseconds");
}
```

Exista doua modalitati de setare a imaginii, prima in care se seteaza automat si matricea si a doua in care nu. Stocam, in acelasi timp, si timpul de executie.

O alta functie importanta este functia de inversare a imagnii, pe care nu o folosim in acest proiect in mod direct intrucat o sa realizam inversarea in cele 3 clase importante, dar algoritmul este acelasi.

```
public void invertImage() {
    for (int i = 0; i < this.width; i++) {
        for (int j = 0; j < this.height; j++) {
            // Parcurg fiecare pixel si ii inversez fiecare canal de culoare si la
            // transparenta nu umblu
            this.imageMatrix[i][j].setRed(255 - imageMatrix[i][j].getRed());
            this.imageMatrix[i][j].setGreen(255 - imageMatrix[i][j].getGreen());
            this.imageMatrix[i][j].setBlue(255 - imageMatrix[i][j].getBlue());

            // int rgb = (this.imageMatrix[i][j].getRed() << 24) |
            // (this.imageMatrix[i][j].getGreen() << 16 ) |
            // (this.imageMatrix[i][j].getGreen() << 8 ) |
            // (this.imageMatrix[i][j].getAlpha());
            // Folosesc aici o functie ajutatoare pentru a seta culoarea inapoi in imaginea
            // mea efectiva, dar puteam sa o fac si manual, asa cum e mai sus
            int rgb = new Color(this.imageMatrix[i][j].getRed(), this.imageMatrix[i][j].getGreen(),
                    this.imageMatrix[i][j].getBlue()).getRGB();

            this.image.setRGB(i, j, rgb);
        }
    }
}
```

Avem nevoie si de o functie de salvare a imaginii intr-un anumit fisier.

```
public void saveImage(String fileName) {
    long startTime = System.currentTimeMillis();
    try {
        // Creez un nou fisier cu numele dat
        File outputfile = new File(fileName);
        // Salvez noua imagine
        ImageIO.write(this.image, "jpg", outputfile);
    } catch (IOException e) {
        // Printez exceptia care poate aparea
        System.out.println("Exceptie : " + e.getMessage());
    } finally {
    }
    long endTime = System.currentTimeMillis();
    System.out.println("Scrierea in fisierul destinatie a durat:  " + (endTime - startTime) + " milliseconds");
}
```

Arat aici si o metoda care seteaza matricea de pixeli, prin alocarea de memeorie.

```
public void setImageMatrix(BufferedImage image) {
    // Citim imaginea efectiva cu o functie ajutatoare
    this.image = image;
    // Salvam cele doua dimensiuni si alocam spatiu
    this.width = this.image.getWidth();
    this.height = this.image.getHeight();
    // Puteam sa apelez direct constructorul de mai sus, dar nu este prima
    // instructiune din acest constructor
    this.imageMatrix = new Pixel[this.width][this.height];
    for (int i = 0; i < this.width; i++) {
        for (int j = 0; j < this.height; j++) { // Alocam memeorie pentru fiecare pixel din matrice
            this.imageMatrix[i][j] = new Pixel();
            // Folosim functie ajutatoare pentru a obtine culoarea din
            // imagine
            Color mycolor = new Color(this.image.getRGB(i, j));
            // Setam cele 4 atribute cu ajutorul setterelor pentru ca sunt private
            this.imageMatrix[i][j].setRed(mycolor.getRed());
            this.imageMatrix[i][j].setGreen(mycolor.getGreen());
            this.imageMatrix[i][j].setBlue(mycolor.getBlue());
            this.imageMatrix[i][j].setAlpha(mycolor.getAlpha());
        }
    }
}
```

## ImageDisplay:

Am folosit aceasta clasa doar pentru a printa pe ecran o imagine, fie dintr-un anumit fisier, fie direct, cu ajutorul unui obiect al clasei BufferedImage. Instantiez un obiect din aceasta clasa si este printat pe ecran un frame cu imaginea respectiva. Am supradefinit aici doi constructori pentru cele doua cazuri precizate mai sus.

```
import javax.swing.ImageIcon;

public class ImageDisplay extends Image {
    // Aici mostenim clasa Image, pentru ca vrem sa afisam imaginea efectiva pe
    // display

    // Avem un constructor, care apeleaza constructorul din
    // clasa parinte, deci imi printeaza imaginea direct in aceasta clasa, imaginea
    // care este intr-un anumit fisier dat prin numele fileName
    // Aici nu am nevoie de exceptii pentru citirea din fisier pentru ca o am deja
    // in clasa parinte
    public ImageDisplay(String fileName, String title) {
        // Creez imagine din fisierul dat
        super(fileName, true);
        // Partea grafica de afisare a imaginiii
        ImageIcon icon = new ImageIcon(this.getImage());
        JFrame frame = new JFrame(title);
        frame.setLayout(new FlowLayout());
        frame.setSize(this.getWidth(), this.getHeight());
        JLabel lbl = new JLabel();
        lbl.setIcon(icon);
        frame.add(lbl);
        frame.setVisible(true);
    }

    public ImageDisplay(Image image, String title) {
        ImageIcon icon = new ImageIcon(image.getImage());
        JFrame frame = new JFrame(title);
```

2. Producer-Consumer-WriterResult

## Buffer:

Vrem sa sincronizam cele doua thread-uri producator si consumator, deci avem nevoie de o resursa comuna care sa gestioneze accesul celor doua. Bufferul are un pixel si inca un flag pentru valabilitatea sa.

```
public class Buffer {
    // Avem aici buffer-ul pentru cele doua clase care au acces comun la aceasta
    // resursa
    // Resursa comunca este un pixel
    private Pixel pixel;
    // Am nevoie de un flag pentru a arata faptul ca pot pune sau scoate din el
    private boolean available = false;

    public synchronized Pixel get() {
        while (!available) {
            try {
                wait();
                // Asteapta producatorul sa puna o valoare
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        available = false;
        notifyAll();
        return pixel;
    }

    public synchronized void put(Pixel pixel) {
        while (available) {
            try {
                wait();
                // Asteapta consumatorul sa preia valoarea
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        this.pixel = pixel;
        available = true;
        notifyAll();
    }
```

## Producator:

Aceasta clasa preia informatii din imaginea care ii este data la initializare si apoi trimite pixel cu pixel valorile catre consumator. Avem, asadar, un constructor unde atribuim valori atributelor image si, evident, o valoarea care este buffer-ul comun. Trebuie sa folosim un BufferedImage pentru a evita lucrul cu partea

neinteresanta a fisierelor, header-ul si alte informatii care nu tin chiar de procesarea efectiva. Extragem, asadar, pixelul de la fiecare pozitie si il trimitem in buffer. Mai avem 4 cazuri pentru a evidentia segmentarea imaginii si pentru a tine evidenta parcursului. Clasa parinte este clasa thread, in mod evident.

```java
public void run() {
    // System.out.println("Producatorul ruleaza!");
    // El imi pune doar pixelii individuali
    // Afiseaza, de asemenea, etapele sale, cele patru sferturi ale imaginii
    Pixel pixelAux = new Pixel();
    for (int i = 0; i < image.getWidth(); i++) {
        for (int j = 0; j < image.getHeight(); j++) {
            // Folosim functie ajutatoare pentru a obtine culoarea din imagine
            Color mycolor = new Color(image.getImage().getRGB(i, j));
            // Setam cele 4 atribute cu ajutorul setterelor pentru ca sunt private
            pixelAux.setRed(mycolor.getRed());
            pixelAux.setGreen(mycolor.getGreen());
            pixelAux.setBlue(mycolor.getBlue());
            pixelAux.setAlpha(mycolor.getAlpha());

            // Fac aici cele patru cazuri pentru cele patru sferturi ale imaginii
            // si fac si sleep-ul pentru a permite citirea de catre consumator
            if (i == image.getWidth() / 4 && j == 0) {
                System.out.println("Producatorul pune primul sfert!\n");
                try {
                    sleep((int) (1000));
                } catch (InterruptedException e) {
                }
            } else if (i == 2 * image.getWidth() / 4 && j == 0) {
                System.out.println("\nProducatorul pune al doilea sfert!\n");
                try {
                    sleep((int) (1000));
                } catch (InterruptedException e) {
                }
            } else if (i == 3 * image.getWidth() / 4 && j == 0) {
                System.out.println("\nProducatorul pune al treilea sfert!\n");
                try {
                    sleep((int) (1000));
                } catch (InterruptedException e) {
                }
            } else if (i == image.getWidth() - 1 && j == image.getHeight() - 1) {
                System.out.println("\nProducatorul pune toata imaginea!\n");
                try {
                    sleep((int) (1000));
                } catch (InterruptedException e) {
                }
            }

            buffer.put(pixelAux);
            // pixelAux.printPixel();
        }
```

Consumator:

Aceasta clasa este putin mai complicata, deoarece chiar daca este similara clasei producator, face legatura printr-un pipe cu clasa WiterResult.

```
public void run() {
    long startTime = System.currentTimeMillis();
    Pixel pixelAux = new Pixel();
    int byteHelp = 0;
    if (action == 0) {
        for (int i = 0; i < image.getWidth(); i++) {
            for (int j = 0; j < image.getHeight(); j++) {
                // Citesc pixelul si setez pixelul
                pixelAux = buffer.get();
                image.imageMatrix[i][j].setPixel(pixelAux.getRed(), pixelAux.getGreen(), pixelAux.getBlue(),
                        pixelAux.getAlpha());
                for (int k = 0; k < 4; k++) {
                    switch (k) {
                    case 0:
                        byteHelp = image.imageMatrix[i][j].getRed();
                        break;
                    case 1:
                        byteHelp = image.imageMatrix[i][j].getGreen();
                        break;
                    case 2:
                        byteHelp = image.imageMatrix[i][j].getBlue();
                        break;
                    case 3:
                        byteHelp = image.imageMatrix[i][j].getAlpha();
                        break;
                    }
                    try {
                        out.writeInt(255 - byteHelp);
                    } catch (IOException e) {
                        e.printStackTrace();
                    }
                    try {
                        sleep((int) (0));
                    } catch (InterruptedException e) {
                    }

                }

                if (i == image.getWidth() / 4 && j == 0) {
                    System.out.println("Consumatorul primeste primul sfert!");

                } else if (i == 2 * image.getWidth() / 4 && j == 0) {
                    System.out.println("Consumatorul primeste al doilea sfert!");

                } else if (i == 3 * image.getWidth() / 4 && j == 0) {
                    System.out.println("Consumatorul primeste al treilea sfert!");
```

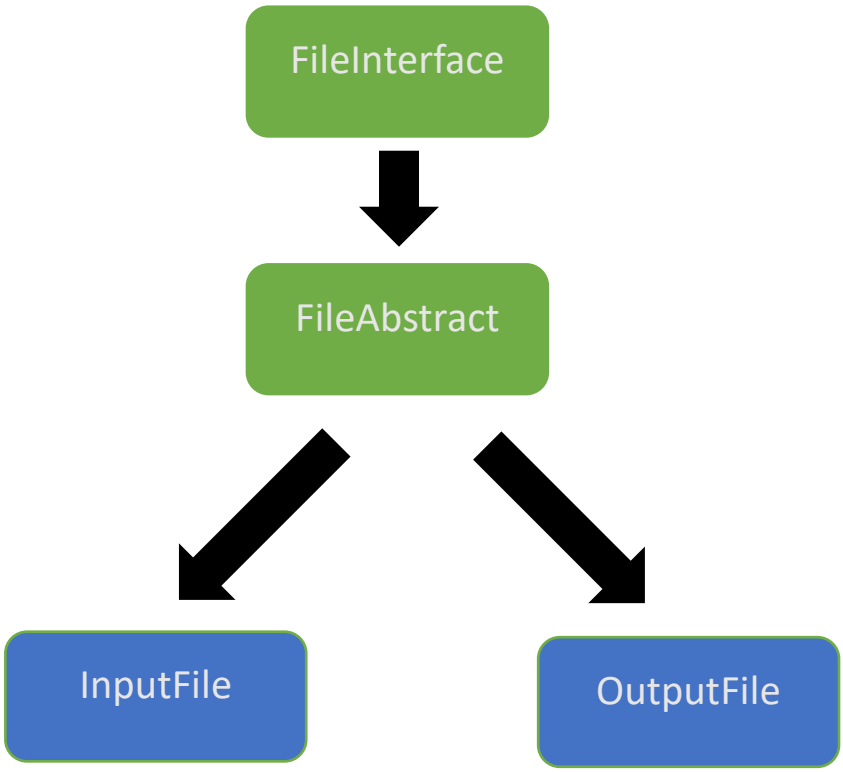
Asa cum se vede, punem in pipe valoarea pentru un canal, direct inversata. Deci facem procesarea imaginii cumva, in interiorul pipe-ului. In rest, este foarte similara cu clasa anterioara.

WriterResult:

Avem aici o alta clasa foarte similara cu cele doua de mai sus. Ideea este aici ca luam byte-ul, il punem in locul care trebuie si apoi salvam si vizualizam imaginea.

```
public void run() {
    System.out.println("Writer ruleaza!");
    int byteHelp = 0;
    for (int i = 0; i < image.getWidth(); i++) {
        for (int j = 0; j < image.getHeight(); j++) {
            for (int k = 0; k < 4; k++) {

                try {
                    byteHelp = in.readInt();
                } catch (IOException e) {
                    e.printStackTrace();
                }

                switch (k) {
                case 0:
                    image.imageMatrix[i][j].setRed(byteHelp);
                    // System.out.println("Producatorul a pus :\t" + i);
                    break;
                case 1:
                    image.imageMatrix[i][j].setGreen(byteHelp);
                    break;
                case 2:
                    image.imageMatrix[i][j].setBlue(byteHelp);
                    break;
                case 3:
                    image.imageMatrix[i][j].setAlpha(byteHelp);
                    break;
                }

                if (i == image.getWidth() - 1 && j == image.getHeight() - 1 && k == 3) {
                    System.out.println("Am pus toata informatia prin pipe-uri!");
                    imgDisp = new ImageDisplay(image, "Imaginea din WriterResult");
                    image.saveImage(fileName);
                }
            }
            int rgb = new Color(image.imageMatrix[i][j].getRed(), image.imageMatrix[i][j].getGreen(),
                    image.imageMatrix[i][j].getBlue()).getRGB();

            image.getImage().setRGB(i, j, rgb);

            if (i == image.getWidth() / 4 && j == 0) {
                System.out.println("WriterRezult primeste primul sfert!");

            } else if (i == 2 * image.getWidth() / 4 && j == 0) {
                System.out.println("WriterRezult primeste al doilea sfert!");

            } else if (i == 3 * image.getWidth() / 4 && j == 0) {
                System.out.println("WriterRezult primeste al treilea sfert!");
            }
        }
```

3. Gestionarea fisierelor




## FileInterface:

O interfata simpla unde setez ce metode vreau sa am in clasele care se extind din aceasta.

```
public interface FileInterface {
    // Interfata pentru cele doua tipuri de fisiere
    public String getFileName();

    public void setFileName(String fileName);

    public void displayFileName();
}
```

## FileAbstract:

O clasa abtracta unde imi declar atributul si imi implementez metodele necesare pentru urmatoarea etapa si utilizez interfata de mai sus.

```
public abstract class FileAbstract implements FileInterface {
    // Clasa abstracta pentru cele doua tipuri de fisiere
    // Declar aici atributul pentru numele fisierului
    private String fileName = null;

    // Implementez doar doua functii pentru a arata abstractiza
    public String getFileName() {
        return this.fileName;
    }

    public void setFileName(String fileName) {
        this.fileName = fileName;
    }

    public abstract void displayFileName();
}
```

## InputFile:

Aici avem o clasa care are doar un constructor care se foloseste de un chooser specific Java care imi returneaza numele unui fisier pe care eu il setez in mod vizual cu ajutorul unui obiect extern.

```
public class InputFile extends FileAbstract {
    // Clasa pentru fisierul de intrare cu interfata grafica pentru selectare
    // Avem nevoie doar de un constructor
    public InputFile() {
        // Setez primul director care este afisat, care este cel curent
        String userDirLocation = System.getProperty("user.dir");
        File userDir = new File(userDirLocation);
        // default to user directory
        // JFileChooser fileChooser = new JFileChooser(userDir);
        JFileChooser chooser = new JFileChooser(userDir);
        chooser.setFileSelectionMode(JFileChooser.FILES_ONLY);
        chooser.showSaveDialog(null);
        // Setez numele fisierului de intrare dupa numele fisierului pe care l-am ales
        // eu
        setFileName(chooser.getSelectedFile().getName());
    }

    // Suprascriere functie
    public void displayFileName() {
        System.out.println(getFileName());
    }
}
```

## OutputFile:

Aici avem clasa de output care nu face altceva decat sa mi stabileasca numele pentru fisierul de output unde o sa fie salvata imaginea, deci am doua cazuri, fie ii dau o denumire automat, in functie de numele fisierului de intrare, fie utilizatorul poate seta un nume custom.

```
public OutputFile(String inputFileName) {
    // Numele automat pe care il sugereaza programul
    setFileName(inputFileName.replace(".bmp", "") + "_inverted.JPG");
    // Afisare pentru stabilirea deciziei
    System.out.println("Save the file as: " + getFileName() + "?");

    // Citire de la tastatura pentru stabilirea deciziei
    // Aruncare de exceptie pentru eroare de citire
    try (Scanner myScanner = new Scanner(System.in)) {
        String answear = "T";

        // Cat timp valoarea citita de la tastatura nu e 'Y' sau 'N', citeste in
        // continuare pana ajunge la una din cele doua
        while (answear.compareTo("Y") != 0 && answear.compareTo("N") != 0) {
            System.out.println("Y for yes\nN for no!");
            answear = myScanner.nextLine(); // Read user input
            // Transform in litera mare pentru a ma ajuta la comparatie
            answear = answear.toUpperCase();
            switch (answear) {
            case "Y":
                // Daca totul e ok, ies, deoarece numele e deja salvat
                break;
            case "N":
                System.out.println("Enter output file: ");
                // Citesc de la tastatura
                setFileName(myScanner.nextLine());
                System.out.println("Output file is: " + getFileName());
                break;
            default:
                System.out.println("Please enter Y or N!");
            }
            System.out.println(answear);
        }
    }
}
```

## 4. Performante

Nu pot masura chiar toti timpii intr-un mod foarte corect, deoarece la una dintre etape se asteapta input-ul utilizatorului, dar avem performante foarte bune de sub 100 de ms pentru operatiile care lucreaza direct, fara wait si undeva la 4412 pentru procesul de citire de catre producator, care are 4 delay-uri de 1000 de ms, deci, per total, avem performante bune.

## 5. Concluzii

Am reusit sa prelucram in acest proiect imagini de tipul bmp si sa le salvam intr-un fisier nou prin intermediul mai multor tehnici Java. Am utilizat thread-uri sincronizate, dar si comunicare prin pipe-uri, pe langa conceptele specificie POO de incapsulare, supradefinire, suprascriere, precum si clase astracte si interfete.

## 6. Bibliografie


- Curs facultate
- https://www.javatpoint.com/varargs
- https://stackoverflow.com/questions/180158/how-do-i-time-a-methods-execution-in-java
- https://beam.apache.org/releases/javadoc/current/org/apache/beam/sdk/io/gcp/bigquery/WriteResult.html
- https://stackoverflow.com/questions/14245170/comunication-using-pipe-java-multithreading
- https://www.w3schools.com/java/java_abstract.asp