IMAGE CONTRAST MODIFICATION

DE BLAGA RAZVAN MIHAI

333AA

Tema de proiect aleasa este schimbarea contrastului unei imagini , o tema care m-a ajutat sa inteleg bazele sincronizarii firelor de executie independente si asincrone prin intermediul scenariului producator/consumator , implementarea metodei prin care doua fire de executie pot comunica si anume comuniarea prin fluxuri de tip “pipe” precum si procesarea si prelucrarea unei imagini de tip BMP-RGB .

Ce reprezinta contrastul unei imagini ?

Contrastul este o proprietate intrinsecă a unei imagini care cuantifică diferența de luminozitate între părțile luminoase și întunecate ale unei imagini .

Contrastul caracterizează distribuția luminii unei imagini. Vizual, poate fi interpretat ca o răspândire a histogramei de luminozitate a imaginii.

Cazuri speciale :

- Pentru contrast zero, imaginea observată este complet gri
- Pentru un contrast maxim, fiecare pixel din imagine este negru sau alb

Modificarile de contrast si de brightness sunt transformari de genul :

f(x)=a*x + b unde a este coeficientul de contrast , b controleaza brightness-ul si x este o componenta de culoare(R,G or B) de la pixelul curent .

Pentru o evidentiere mai buna a contrastului si brightnessului formula poate fi scrisa astfel :

f(x)=α(x−128)+128+b

Formula contrast :

Sunt multe posibilitati de a defini formula contrastului . Unele include culoare , altele nu . In general , formulele contrastului reprezinta un raport de tipul :

$$\frac{\text{Luminance difference}}{\text{Average luminance}}.$$

Motivul din spatele acestui lucru este că o diferență mică este neglijabilă dacă luminanța medie este mare, în timp ce aceeași diferență mică contează dacă luminanța medie este scăzută.

Descrierea implementarii :

Implementarea alesa de mine consta in respectarea cerintelor temei :

1. Imaginea sursa este BMP (fisier) – 24bit BMP – RGB
2. Pentru procesare se folosesc doar algoritmi si/ sau secvente de cod low-level (nu se accepta utilizare de metode de procesare altele decat cele scrise in tema)
3. Include in totalitate conceptele POO – incapsulare, mostenire, polimorphism, abstractizare
4. Codul sursa respecta absolut toate "Coding standards". Codul sursa este comentat
5. Operatii de lucru cu fisiere
6. Operatii de intrare de la tastatura si prin parametri liniei de comanda pentru asignarea fisierelor de intrare, parametri / setarile / optiunile de executie si pentru asignarea fisierelor de iesire
7. Aplicatia trebuie sa fie multimodulara (impartirea in clase cu ierarhii – chiar cu cost in timp de procesare). Cel putin 3 niveluri de mostenire
8. Include varargs
9. Include constructori
10. Include cel putin un bloc de initializare si un bloc static de initializare
11. Include Interface (cu o clasa care o implementeaza)
12. Include Clase Abstracte cu metode abstracte si clase concrete care extind clasele abstracte
13. Include tratarea exceptiilor
14. Aplicatia contine 2 pachete: Pachetul 1 sa contina aplicatia de test, pachetul 2 sa contina restul claselor

15. Aplicatia contine Producer-Consumer cu urmatoarele cerinte:
    a. un nou thread este alocat citirii din fisier a imaginii sursa – Producer Thread. Intra in Not Runnable dupa citirea a fiecarui sfert (1/4) de informatie
    b. un nou thread (Consumer Thread) este alocat consumului informatiei furnizate de Producer Thread. Se utilizeaza "multithread communication" (notify).
    c. Se insereaza output la consola si sleep (1000) pentru a evidentia etapele comunicarii.
    d. Se folosesc elementele de sincronizare pentru protectia la o eventuala interferenta cu alte posibile threaduri
    e. Dupa terminarea consumului intregii informatii de imagine sursa, se incepe procesarea

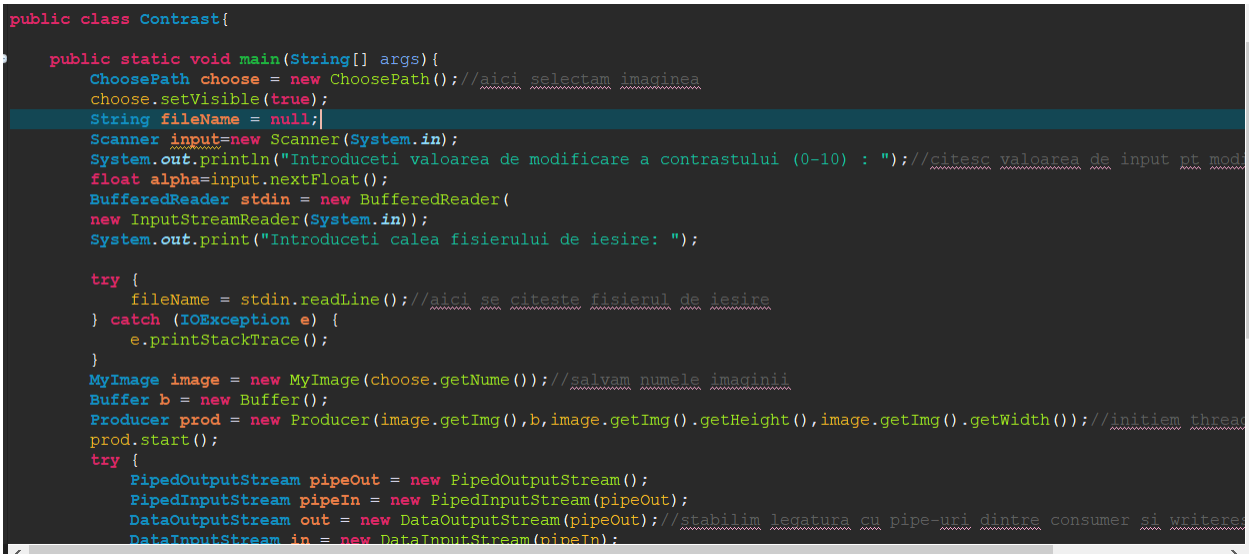
16. Aplicatia contine comunicatie prin Pipes cu urmatoarele cerinte
    a. Consumer utilizeaza un Pipe pentru a transmite imaginea procesata catre un obiect de tipul WriterResult
    b. Transmiterea prin pipe se face partitionand informatia in 4 segmente.
    c. La transmiterea fiecarui segment segment se trimite la consola un mesaj
    d. La receptia fiecarui segment segment se trimite la consola un mesaj
    e. Rezutatul se depune intr-un fisier

Astfel , in implementare am o interfata , o clasa abstracta care implementeaza aceea interfata , din clasa abstracta rezulta trei niveluri de mostenire . De asemenea , am si firele de executie Producer si Consumer care sunt sincronizate printr-un buffer precum si WriteResult care comunica cu Consumer printr-un ‘pipe’ .

Descrierea aplicatiei implementate :

Aplicatia implementata de mine are 2 pachete si anume : packWork si packTest .

In packTest se afla main-ul aplicatiei si anume :

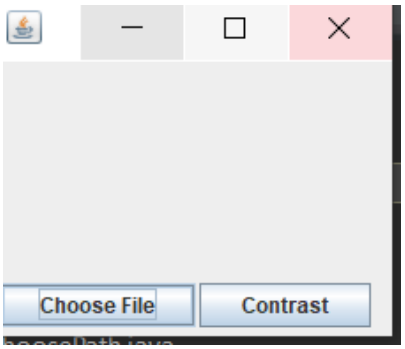


```
public class Contrast{

    public static void main(String[] args){
        ChoosePath choose = new ChoosePath();//aici selectam imaginea
        choose.setVisible(true);
        String fileName = null;
        Scanner input=new Scanner(System.in);
        System.out.println("Introduceti valoarea de modificare a contrastului (0-10) : ");//citesc valoarea de input pt modi
        float alpha=input.nextFloat();
        BufferedReader stdin = new BufferedReader(
        new InputStreamReader(System.in));
        System.out.print("Introduceti calea fisierului de iesire: ");

        try {
            fileName = stdin.readLine();//aici se citeste fisierul de iesire
        } catch (IOException e) {
            e.printStackTrace();
        }
        MyImage image = new MyImage(choose.getNume());//salvam numele imaginii
        Buffer b = new Buffer();
        Producer prod = new Producer(image.getImg(),b,image.getImg().getHeight(),image.getImg().getWidth());//initiem threa
        prod.start();
        try {
            PipedOutputStream pipeOut = new PipedOutputStream();
            PipedInputStream pipeIn = new PipedInputStream(pipeOut);
            DataOutputStream out = new DataOutputStream(pipeOut);//stabilim legatura cu pipe-uri dintre consumer si writere
            DataInputStream in = new DataInputStream(pipeIn);
```

Aici aleg locatia imaginii de intrare prin clasa ChoosePath(o explic mai jos) , inputul cu cat sa modelam contrastul imaginii precum si numele fisierului de output . Mai departe declar imaginea cu care voi lucra din clasa MyImage , bufferul b si thread-ul producer .

In exceptia ce urmeaza fac comunicarea prin fluxul de tip “pipe” dintre Consumer si WriteResult unde thread-ul Consumer trimite pixelii modificati catre thread-ul WriteResult . De asemenea instantiez si doua thread-uri de tipul Consumer si WriteResult .

Clasa ChoosePath este o clasa ajutatoare de unde selectez locatia imaginii sursa din documentele calculatorului in folosinta . Fereastra ce se deschide este :

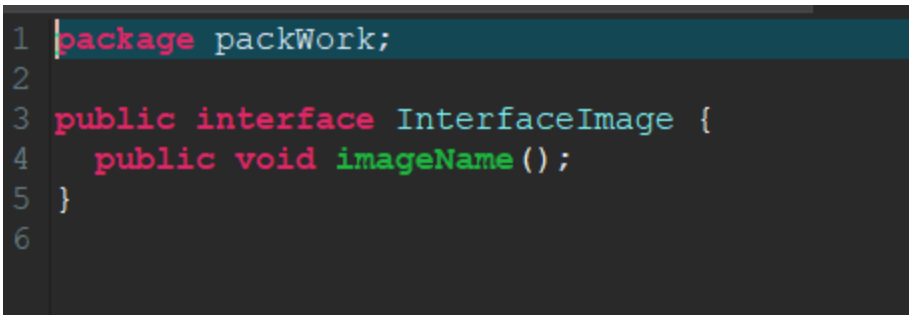


Aceasta are 2 butoane de folosinta si anume : ChooseFile care ma ajuta sa selectez locatia imaginii sursa si butonul Contrast care termina operatiunea .

```
JButton btnNewButton = new JButton("Choose File");
btnNewButton.setBounds(5, 125, 111, 25);
btnNewButton.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent arg0) {
    JFileChooser chooser = new JFileChooser();
    chooser.showOpenDialog(chooser);
    chooser.setVisible(true);
    setNume(chooser.getSelectedFile().toString());
    }
});
contentPane.setLayout(null);
contentPane.add(btnNewButton);


JButton btnConvert = new JButton("Contrast");
btnConvert.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        dispose();
    }
});
btnConvert.setBounds(118, 125, 97, 25);
contentPane.add(btnConvert);
```

In continuare voi prezenta cele 4(erau nevoie 3) niveluri de mostenire .

In primul rand am implementat interfata “InterfaceImage” pentru a ma ajuta sa construiesc clasele ce se deriva din ea .



```java
package packWork;

public interface InterfaceImage {
  public void imageName();
}

```

In continuare avem clasa abstracta care o sa implementeze interfata cu metoda abstracta dimensiuni() folosita pentru a obtine dimensiunile unei imagini . De asemenea am si implementat imageName() care este o implementare pentru metoda interfetei .

```java
package packWork;

abstract public  class AbstractClass implements InterfaceImage{//clasa abstracta care implementeaza interfata
    protected static String name;

    abstract public void dimensiuni();//metoda abstracta care va fi implementata in clasa mostenitoare

    public static String getName() {
        return name;
    }
    public static void setName(String name) {
        AbstractClass.name = name;
    }
    public void imageName(){//metoda de la interfata implementata
        System.out.println("Image name: "+name);
    }
}
```

Clasa “ParentImage” extinde clasa abstracta si este parintele clasei unde se afla imaginea “MyImage” . De asemenea implementeaza si metoda abstracta care se afla in clasa abstracta precum si alte metode necesare pentru implementare .

```java
public class ParentImage extends AbstractClass{//prima clasa care se mosteneste din clasa abstracta
    private int width;
    private int height;

    @Override
    public void dimensiuni() {//implementarea metodei din clasa abstracta
        // TODO Auto-generated method stub
        System.out.println("Imaginea originala are dimensiunile : "+width+" "+height);
    }

    public int getWidth() {
        return width;
    }
    public void setWidth(int width) {
        this.width = width;
    }
    public int getHeight() {
        return height;
    }
    public void setHeight(int height) {
        this.height = height;
    }
    public ParentImage(){
        AbstractClass.setName("");
        width = 0;
```

Din clasa “ParentImage” se mosteneste clasa “MyImage” unde avem stocata imaginea sursa prin intermediul clasei “ChoosePath” . Avem si aici implementati constructori care sa ne ajute la rularea aplicatiei .

```java
package packWork;

import java.awt.Color;

public class MyImage extends ParentImage{//ultimul nivel de mostenire , aici avem imaginea
    private BufferedImage img;
    private File f;

    public File getF() {
        return f;
    }
    public void setF(File f) {
        this.f = f;
    }
    public BufferedImage getImg() {
        return img;
    }
    public void setImg(BufferedImage img) {
        this.img = img;
    }



    public MyImage(){
        ParentImage.setName(name);
        setWidth(0);
        setHeight(0);
```

Am terminat de incarcat imaginea sursa acum trebuie sa trecem la partea de implementare a thread-urilor .

Thread-ul Producer : rolul lui este de a lua fiecare pixel din imagine si de a-l pune in buffer pentru ca acele date sa fie prelucrate de Consumer . De asemenea , dupa fiecare sfert de parcurgere thread-ul intra in Not Runnable si se afiseaza un mesaj ca s-a parcurs un sfert .

```
public void run(){
    long start = System.currentTimeMillis();
    MyImage.printName();
    for (int i = 0; i < img.getHeight(); i++) {
    for (int j = 0; j <  img.getWidth(); j++) {
        if((i == (height-1)/4) && (j == (width-1)/4)){
            System.out.println("Producer a parcurs primul sfert");//la parcurgerea fiecarui sfert de informatie
                                                                  //thread-ul intra in Not Runnable
            try {

                sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        else if((i == (height-1)/2) && (j == (width-1)/2)){
            System.out.println("Producer a parcurs jumate");
            try {
                sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        else if((i == (height-1)*3/4) && (j == (width-1)*3/4)){
            System.out.println("Producer a parcurs al treilea sfert");
            try {
                sleep(1000);
```

Buffer-ul b : Buffer-ul b ne ajuta la sincronizarea threadurilor Producer si Cosnumer prin faptul ca cele doua threaduri nu acceseaza simultan buffer-ul si blocheaza obiectul Buffer cand este accesat de un fir de executie , asftel incat niciun alt fir de executie sa nu-l mai poata accesa . Cele doua fire de executie ajung sa coordoneze . Codul implementat pentru buffer este ca cel din curs asa ca nu mai e nevoie sa il explicitez .

Thread-ul Consumer : Threadul Consumer are rolul de a prelucra pixelii primiti de la Producer si de a realiza conversia necesara temei avute . In acest fir de executie are loc algoritmul de transformare al pixelilor pentru a ajusta contrastul imaginii . In prima faza obtin culoarea din pixelul de intrare si valorile ‘R’ , ‘G’ si ‘B’ . Noile valori vor fi egale cu produsul dintre constanta de contrast si valorile vechi . In final ii salvezi intr-o variabila noua rgb . Pixelii transformati sunt dupa trimisi prin “pipe” in firul de executie WriteResult pentru a fi scrisi in imaginea noua . Din nou , la fiecare sfert de pixeli trimisi se afiseaza un mesaj .

```
public void run(){
    long start = System.currentTimeMillis();
    int pixel;

    for (int i = 0; i < getHeight(); i++) {
    for (int j = 0; j < getWidth(); j++) {
        float alpha=getAlpha();
        pixel = buffer.get();//preluam pixelul din buffer
        Color color = new Color(pixel, true);//stocam valoarea rgb din fiecare pixel
        int red = color.getRed();// valoarea red
        int green = color.getGreen();//valoarea green
        int blue = color.getBlue();//valoarea blue
        int newred=(int) (alpha*red);//algoritmul de realizare a contrastului prin inmultirea fiecarei culori cu cons
        int newgreen=(int) (alpha*green);
        int newblue=(int) (alpha*blue);
        //int rgb=(int)(newred + newblue + newgreen);
        Color colorfinal=new Color(red,green,blue);
        int rgb = color.getRGB();
        if((i == (height-1)/4) && (j == (width-1)/4)){//primeste cate un sfert de informatie de la Producer si afiseaz
            System.out.println("Consumer a trimis primul sfert catre WriteResult ");//trimitem rezultatul catre Write
        }
        else if((i == (height-1)/2) && (j == (width-1)/2)){
            System.out.println("Consumer a trimis al doilea sfert catre WriteResult");
        }
        else if((i == (height-1)*3/4) && (j == (width-1)*3/4)){
            System.out.println("Consumer a trimis al treilea sfert catre WriteResult");
        }
        else if((i == height-1) && (j == width-1)){
            System.out.println("Consumer a trimis al patrulea sfert catre WriteResult");
```

Thread-ul WriteResult : Acest thread primeste prin pipe de la Consumer pixelii deja modificati si in prima faza are loc citirea lor si plasarea intr-un vector de pixeli dupa care are loc citirea vectorului si scrierea imaginii cu datele din noul vector . La fiecare sfert de citire de date si de scriere se afiseaza cate un mesaj .

In final , are loc scrierea si plasarea imaginii output pe locatia dorita cu ajutorul stringului nameFile .

```
public void run() {
    int [][] pixels = new int[height][width];//folosim o matrice pentru a stoca pixelii
    long start = System.currentTimeMillis();
    int input = 0;
    BufferedImage image2=new BufferedImage(width,height,BufferedImage.TYPE_INT_RGB);//aici se salveaza rezultatul
    for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        try {
            input = in.readInt();//primim fiecare pixel prin pipe
            Color color=new Color(input,true);//obtinem noua culoare rgb pentru fiecare pixel
            //int red = color.getRed();
            //int green = color.getGreen();
            //int blue = color.getBlue();
            //Color colorout = new Color(red, green, blue);
            pixels[i][j] = color.getRGB();
            if((i == (height-1)/4) && (j == (width-1)/4)){//parcurgem fiecare sfert si dam un mesaj
                System.out.println("WriteResult a primit primul sfert de la Consumer ");//
            }
            else if((i == (height-1)/2) && (j == (width-1)/2)){
                System.out.println("WriteResult a primit al doilea sfert de la Consumer");
            }
            else if((i == (height-1)*3/4) && (i == (width-1)*3/4)){
                System.out.println("WriteResult a primit al treilea sfert de la Consumer");
            }
            else if((i == height-1) && (j == width-1)){
                System.out.println("Writeresult a primit al patrulea sfert de la Consumer");
            }
        } catch (IOException e) {
```

De notat ca pentru fiecare thread afisez timpul de executie

Evaluare performante :

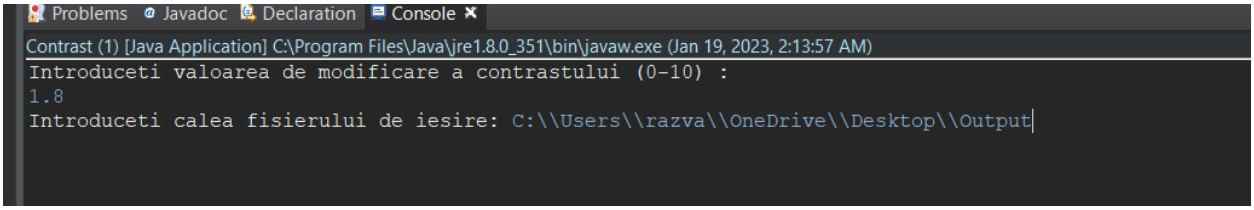
Pentru poza 1 am in input urmatoarele valori :

```
Contrast (1) [Java Application] C:\Program Files\Java\jre1.8.0_351\bin\javaw.exe (Jan 19, 2023, 2:09:49 AM)
Introduceti valoarea de modificare a contrastului (0-10) :
1.6
Introduceti calea fisierului de iesire: C:\\Users\\razva\\OneDrive\\Desktop\\Output
```

Poza inainte de executie :

Poza dupa executie salvata pe Desktop drept Output.bmp :

Pentru poza 2 am in input urmatoarele valori :



Poza inainte de executie :

Poza dupa executia aplicatiei :

Concluzii :

Algoritmul de modificare a contrastului prin prelucrarea pixelilor este un algoritm complicat , greu de inteles si destul de usor de gresit de aceea am intarziat mai mult cu trimiterea temei . Multe surse de pe internet folosesc functii predefinite in Java pentru calcularea contrastului cum am folosit si eu .

Bibliografie :

https://www.dfstudios.co.uk/articles/programming/image-programming-algorithms/image-processing-algorithms-part-5-contrast-adjustment/

https://en.wikipedia.org/wiki/Contrast_(vision)

https://docs.opencv.org/2.4/doc/tutorials/core/basic_linear_transform/basic_linear_transform.html

https://curs.upb.ro/2022/pluginfile.php/419617/mod_resource/content/1/Java09.pdf