Converting Color Image to Gray-Scale Image
Weighted method (luminosity method)

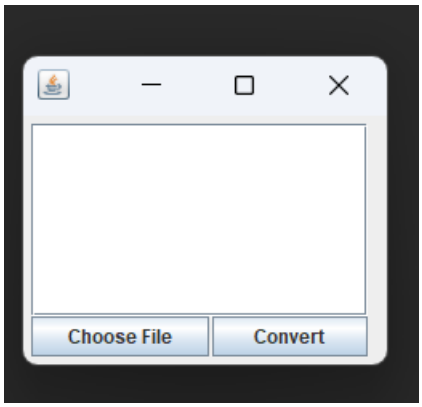
Blaga Razvan Mihai

333AA

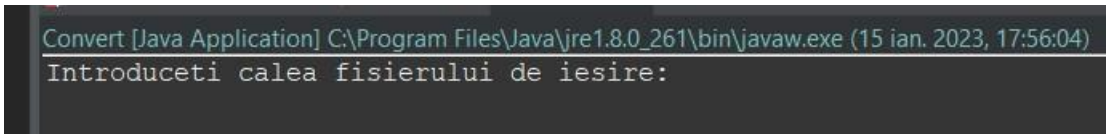
Tema de procesare pentru proiectul meu este“.Converting Color Image to Gray-Scale Image – Weighted method (luminosity method)” , o tema interesanta care m-a ajutat sa inteleg cum se proceseaza o imagine.

Aplicatia are ca scop convertirea unei imaginii color in gray-scale, cu o metoda specifica si anume: weighted method.

La rularea aplicatiei se deschide o fereastra de tipul „choose file” unde trebuie sa alegem imaginea pe care dorim sa o convertim. Dupa aceea, vom primi un mesaj de la consola infromandu-ne ca trebuie sa specificam calea si denumirea pentru noua imagine. Dupa acest pas se incepe efectiv procesarea.



Mai intai se apasa pe Choose File pentru a se selecta imaginea dorita, dupa aceea se apasa butonul convert.



Aici se va introduce calea fisierului de iesire, fara extensia acestuia.

Dupa ce s-a introdus calea fisierului de iesire, se va incepe executia.

Main-ul este in clasa Convert, unde se vor instantia toate clasele necesare acestei aplicatii.

```
public class Convert{

    public static void main(String[] args) {
        ChooseFile choose = new ChooseFile();
        choose.setVisible(true);
        //ReadFileName readIn = new ReadFileName();
        ReadFileName readOut = new ReadFileName();
        //readIn.Read("de intrare"); //citeste numele fisierului de intrare
        readOut.Read("de iesire"); //citeste numele fisierului de iesire
        //OriginalImage img_orig = new OriginalImage(readIn.getFileName());//instantiez clasa pentru imaginea originala
        //String a = choose.getNume();
        OriginalImage img_orig = new OriginalImage(choose.getNume());
        Buffer b = new Buffer();
        ProducerThread prod = new ProducerThread("ProducerImage",img_orig.getImg(),b,img_orig.getImg().getHeight(),img_
        //ConsumerThread consum = new ConsumerThread("consumer",img_orig.getWidth(),img_orig.getHeight(),b,img_orig.get
        prod.start();
        //consum.start();
        try {
            PipedOutputStream pipeOut = new PipedOutputStream();
            PipedInputStream pipeIn = new PipedInputStream(pipeOut);
            DataOutputStream out = new DataOutputStream(pipeOut);
            DataInputStream in = new DataInputStream(pipeIn);
            ConsumerThread c1 = new ConsumerThread("ConsumerImage",img_orig.getWidth(),img_orig.getHeight(),b,img_orig.
            WriteResult wr = new WriteResult(in,img_orig.getImg().getHeight(),img_orig.getImg().getWidth(),readOut.getF
            c1.start();
            wr.start();
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
```

Am lasat comentat partea de inserare a imaginii de la tastatura, prin scriere in consola, in caz ca se doreste tastarea numelui, si nu cautarea in folder prin fereastra choose file.

ReadFileName este clasa ce va contine numele fisierului si ajuta la citirea imaginii.

Dupa citirea imaginii de output, se instantiaza clasa OriginalImage ce va contine imaginea originala cu dimensiunile ei.

Dupa acest pas se incepe citirea pixelilor.

Se aloca un thread pentru citirea imaginii din fisier („ProducerThread”), acest thread Intra in Not Runnable dupa citirea a fiecarui sfert (1/4) de informatie. Se va utiliza “multithread communication” (notify), iar un nou thread(„ConsumerThread”) va fi alocat pentru consumul de informatie. Acesta va calcula noua valoare care trebuie stocata pe canalele r, g, b , valoare ce va ajuta la coversia in gray-scale. La fiecare sfert de informatie trimis, Producer va primi comanda „sleep(1000)” care il va pune in Not Runnable si se va insera output la consola pentru a evidentia etapele comunicarii.

```
public void run(){
    long start = System.currentTimeMillis();
    OriginalImage.printName();
    for (int y = 0; y < img.getHeight(); y++) {
    for (int x = 0; x <  img.getWidth(); x++) {
        if((y == (height-1)/4) && (x == (width-1)/4)){
            System.out.println(super.getName()+" a parcurs primul sfert");//la parcurgerea fiecarui sfert
            try {                                                          //output la consola si sleep(1000)

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

```
            }
        }
        else if((y == height-1) && (x == width-1)){
            System.out.println(super.getName()+" a parcurs al patrulea sfert");
            try {
                sleep(1000);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        buffer.put(img.getRGB(x, y));//se pune valorea pixelului in buffer pentru
                                     //a fi preluenat de consumer

    }
```

Consumer, la primirea fiecarui sfert va afisa acest lucru. Dupa ce s-a terminat consumul intregii imagini, va incepe procesarea acesteia.

```
public void run(){
    long start = System.currentTimeMillis();
    BufferedImage img2 = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
    int p;


    for (int y = 0; y < getHeight(); y++) {
    for (int x = 0; x < getWidth(); x++) {
        p = buffer.get();//se ia pixelul din buffer
        Color color = new Color(p, true);//pentru a stoca valoarea rgb a fiecarui pixel
        int red = color.getRed();// valoarea red a pixelului
        int green = color.getGreen();//valoarea green
        int blue = color.getBlue();//valoarea blue
        int value = (int) (red* 0.2126 + 0.7152 * green + 0.0722 * blue); //calculare valoare noua pentru r,g,b
        if((y == (height-1)/4) && (x == (width-1)/4)){//primeste primul sfert de informatie de la Producer
            System.out.println(super.getName()+" a trimis primul sfert catre WriteResult ");//si il trimitre catre
        }
        else if((y == (height-1)/2) && (x == (width-1)/2)){
            System.out.println(super.getName()+ " a trimis al doilea sfert catre WriteResult");
        }
        else if((y == (height-1)*3/4) && (x == (width-1)*3/4)){
            System.out.println(super.getName()+" a trimis al treilea sfert catre WriteResult");
        }
        else if((y == height-1) && (x == width-1)){
            System.out.println(super.getName()+" a trimis al patrulea sfert catre WriteResult");
        }
        try {
            out.writeInt(value);//trimite noua valoare a pixelului catre WriteResult prin pipe
            } catch (IOException e) {
            e.printStackTrace ();
```

Procesarea imaginii se va face in “Write Result”. In pipe-ul ConsumerThread-WriteResult, Consumer este thread-ul de tip “producer” iar thread-ul WriteResult este thread-ul “consumer” pentru acest caz. WriteResult insereaza output la consola dupa primirea fiecarui segment de informatie. Fiecare pixel primit il salveaza intr-o matrice, iar dupa ce a primit toti pixelii, se incepe procesarea imaginii gray-scale. Dupa scrierea fiecarui sfert in imaginea noua, output la consola ca a facut acest lucru.

```
}
public void run() {
    int [][] matrix = new int[height][width];//stocam intr-o matrice toti pixelii
    long start = System.currentTimeMillis();
    BufferedImage img2 = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);//creare noua imagine
    int value = 0;
    for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        try {
            value = in.readInt();//primire valoare pixel prin pipe
            int red = value;
            int green = value;
            int blue = value;
            Color color = new Color(red, green, blue);  //crrez noua culoare rgb pentru fiecare pixel
            matrix[y][x] = color.getRGB();
            if((y == (height-1)/4) && (x == (width-1)/4)){//output la consola dupa ce primeste segmente
                System.out.println(super.getName()+" a primit primul sfert de la Consumer ");//
            }
            else if((y == (height-1)/2) && (x == (width-1)/2)){
                System.out.println(super.getName()+ " a primit al doilea sfert de la Consumer");
            }
            else if((y == (height-1)*3/4) && (x == (width-1)*3/4)){
                System.out.println(super.getName()+" a primit al treilea sfert de la Consumer");
            }
            else if((y == height-1) && (x == width-1)){
                System.out.println(super.getName()+" a primit al patrulea sfert de la Consumer");
            }
        } catch (IOException e) {
            e.printStackTrace();
        }
}
}
```

```
}
}
        //dupa ce s-a primit toata informatia se incepe procesarea
        for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++){
            img2.setRGB(x, y, matrix[y][x]);
            if((y == (height-1)/4) && (x == (width-1)/4)){//output la consola dupa ce scrie cate un segment
                System.out.println(super.getName()+" a scris primul sfert de informatie");
            }
            else if((y == (height-1)/2) && (x == (width-1)/2)){
                System.out.println(super.getName()+ " a scris al doilea sfert de informatie");
            }
            else if((y == (height-1)*3/4) && (x == (width-1)*3/4)){
                System.out.println(super.getName()+" a scris al treilea sfert de informatie");
            }
            else if((y == height-1) && (x == width-1)){
                System.out.println(super.getName()+" a scris al patrulea sfert de informatie");
            }
        }
        }

try {

    //File f = new File("C:/@Facultate/"+fileName+".bmp");//se creeaza noul fisier, dupa nume trimis
    File f = new File(fileName+".bmp");//se creeaza noul fisier, dupa calea trimis
    ImageIO.write(img2, "bmp", f);//se scrie noua imagine
    System.out.println(super.getName()+" a scris imaginea");
}
catch (IOException e) {
    System.out.println(e);
}
```

Output la consola pentru fiecare segment:

```
ProducerImage a parcurs primul sfert
ConsumerImage a trimis primul sfert catre WriteResult
WriteResult a primit primul sfert de la Consumer
ProducerImage a parcurs al doilea sfert
ConsumerImage a trimis al doilea sfert catre WriteResult
WriteResult a primit al doilea sfert de la Consumer
ProducerImage a parcurs al treilea sfert
ConsumerImage a trimis al treilea sfert catre WriteResult
WriteResult a primit al treilea sfert de la Consumer
ProducerImage a parcurs al patrulea sfert
ConsumerImage a trimis al patrulea sfert catre WriteResult
ProducerImage a durat 28317 milisecunde
ConsumerImage a durat 28315 milisecunde
WriteResult a primit al patrulea sfert de la Consumer
WriteResult a scris primul sfert de informatie
WriteResult a scris al doilea sfert de informatie
WriteResult a scris al treilea sfert de informatie
WriteResult a scris al patrulea sfert de informatie
```

Aplicatia contine 2 pachete:

-packTest:aici este o singura clasa care contine aplicatia de test, si anume clasa Convert unde se instantiaza obiectele fiecarei clase.

-packWork continue restul claselor care au fost create pentru a implementa aplicatia si a atinge toate cerintele temei. AbstractImage este clasa abstracata:

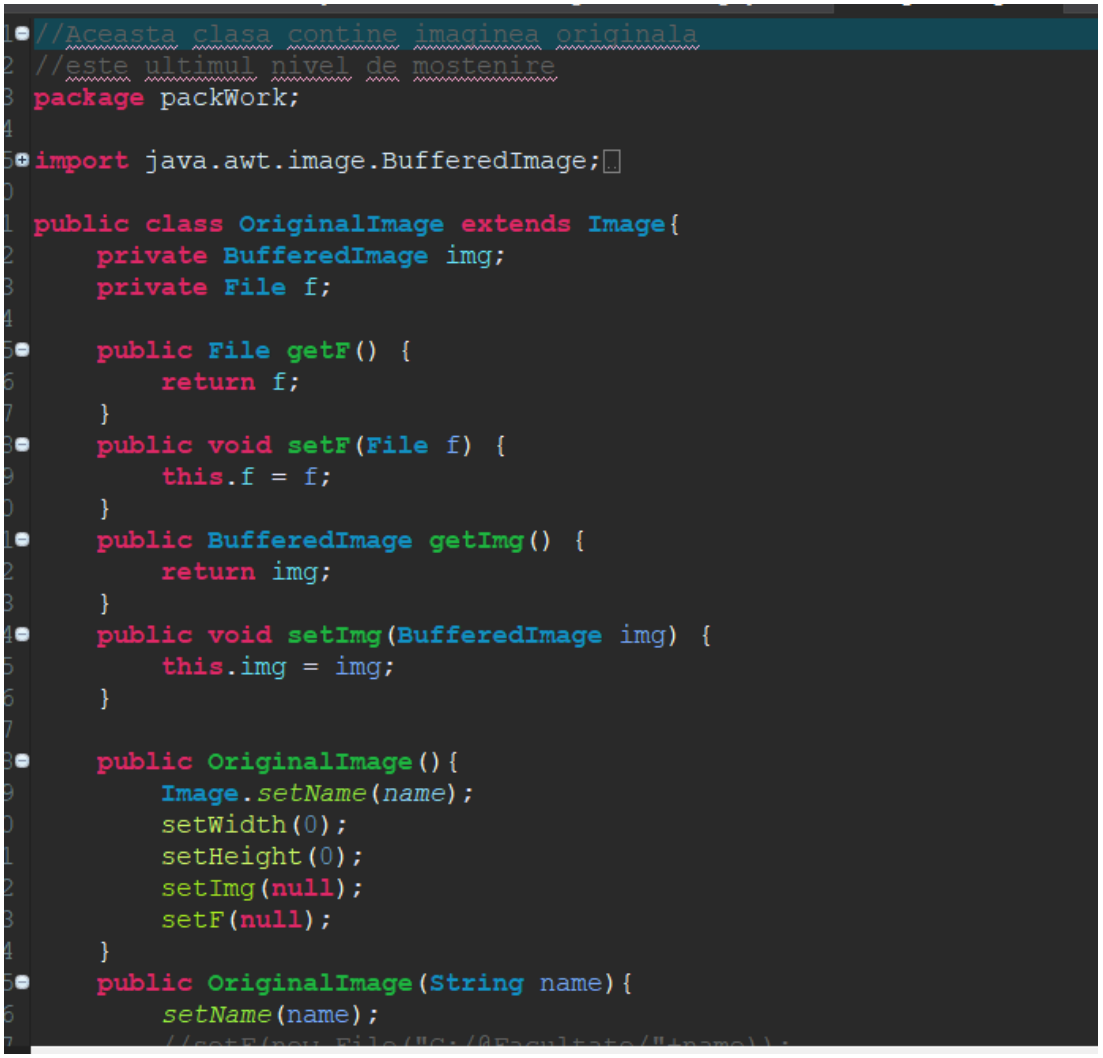
```java
//clasa abstracta
//din aceasta se mostenesc inca 2 clase
package packWork;

abstract public  class AbstractImage {
    protected static String name;

    public static String getName() {
        return name;
    }
    public static void setName(String name) {
        AbstractImage.name = name;
    }
    public void imageName(){//metoda implementata
        System.out.println("Image name: "+name);
    }
    abstract public void size();//metoda abstracta care va fi implementat in clasa care se
                                //va mosteni
}
```

din aceasta mostenindu-se clasa Image unde se va implementa metoda abstracata din clasa parinte:

```java
//prima clasa care se mosteneste din clasa abstracta
package packWork;

public class Image extends AbstractImage{
    private int width;
    private int 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 Image(){
        AbstractImage.setName("");
        width = 0;
        height = 0;
    }
    public Image(String name, int width, int height){
        setName(name);
        this.height = height;
        this.width = width;
    }


    @Override
```

```java
@Override
public void size() {//implementarea metodei care se mosteneste din clasa abstracta
    // TODO Auto-generated method stub
    System.out.println("Imaginea originala are width: "+getWidth()+", height: "+getHeight());
}
```

Din Image se mosteneste clasa OriginalImage care va contine imaginea originala cu marimile acesteia.



```java
//Aceasta clasa contine imaginea originala
//este ultimul nivel de mostenire
package packWork;

import java.awt.image.BufferedImage;

public class OriginalImage extends Image{
    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 OriginalImage(){
        Image.setName(name);
        setWidth(0);
        setHeight(0);
        setImg(null);
        setF(null);
    }
    public OriginalImage(String name){
        setName(name);
```

```
    public OriginalImage(){
        Image.setName(name);
        setWidth(0);
        setHeight(0);
        setImg(null);
        setF(null);
    }
    public OriginalImage(String name){
        setName(name);
        //setF(new File("C:/@Facultate/"+name));
        setF(new File(name));
        try {
            setImg(ImageIO.read(f));
            setWidth(img.getWidth());
            setHeight(img.getHeight());
        } catch (IOException e) {
            // TODO Auto-generated catch block
            e.printStackTrace();
        }
    }

    static{
        printName();
    }
    public static void printName(){
        System.out.println("Numele imaginii este "+name);
    }

}
```

ReadInterface este interfata ce continue metoda ReadInterface():

```
1  //interfata pentru clasa care citeste nume fisier
2  package packWork;
3
4  public interface ReadInterface {
5      public void Read(String scop);
6  }
7
```

metoda abstracta ce va fi implementat in clasa ReadFileName, clasa ce va continue numele fisierului de output:

```java
//Clasa pentru citire nume fisier
package packWork;

import java.awt.image.BufferedImage;

public class ReadFileName implements ReadInterface {//implementeaza o interfata
    private String fileName;
    private BufferedImage img;

    public  String getFileName() {
        return fileName;
    }

    public BufferedImage getImg() {
        return img;
    }

    public void setImg(BufferedImage img) {
        this.img = img;
    }

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

    public void Read(String scop){
        BufferedReader stdin = new BufferedReader(
        new InputStreamReader(System.in));
        System.out.print("Introduceti calea fisierului "+scop+":");

        try {
            fileName = stdin.readLine();
```

```java
        this.fileName = fileName;
    }

    public void Read(String scop){
        BufferedReader stdin = new BufferedReader(
        new InputStreamReader(System.in));
        System.out.print("Introduceti calea fisierului "+scop+":");

        try {
            fileName = stdin.readLine();
        } catch (IOException e) {
            e.printStackTrace();
        }


    }
}
```

numele fisierului de intrare fiind dat la inceputul rularii aplicatiei in fereastra choosefile. Cand se specifica numele fisierului de iesire, trebuie specificat si calea acestuia, scriind in consola.

Dupa ce se introduce numele fisierului de output, consola va afisa “Numele imaginii este null”, iar imediat dupa aceasta „Numele imaginii este” si numele imaginii. Aceasta se intampla din cauza blocului static ce contine metoda printName() si se afiseaza la rularea aplicatiei. Pentru fiecare thread se afiseaza timpul de executie in milisecunde.

```
Numele imaginii este null
Numele imaginii este C:\@Facultate\poza.bmp
```

Weighted method (luminosity method) este o versiune mai sofisticata a „average method”. De asemenea, face o medie a valorilor, dar formeaza o medie ponderata pentru a tine cont de perceptia umana. Prin multe repetitii ale experimentelor atent concepute, psihologii si-au dat seama cat de diferiti percepem luminanța sau rosu, verde si albastru. Ne-au oferit un set diferit de ponderi pentru media canalului nostru pentru a obtine luminanta totala. Formula pentru luminozitate este:

$$Z = 0,2126 \times R + 0,7152 \times G + 0,0722 \times B$$

Conform acestei ecuatii, rosul a contribuit cu 21%, verdele a contribuit cu 72%, ceea ce este mai mare în toate cele trei culori, iar albastrul a contribuit cu 7%.

Observam ca Producer isi termina primul executia, Consumer al doilea si WriteResult ultimul.

```
ProducerImage a durat 28317 milisecunde
ConsumerImage a durat 28315 milisecunde
```

```
WriteResult a durat 29375 milisecunde
```

Imagine originala:

Imagine convertita:

Bibliografie:

https://mmuratarat.github.io/2020-05-13/rgb_to_grayscale_formulas

Documentatie cod sursa:

https://stackoverflow.com

Cursuri Moodle