U.N.S.T. POLITEHNICA București
Facultatea de Automatică si Calculatoare
Departamentul de Automatică şi Ingineria Sistemelor

Documentatie - Proiect Java
Procesare de imagini

- Converting Color Image to
Gray-Scale Image – Average method -

Studentă: Chiriță Andreea
Grupă: 332AB

Cuprins

1. Introducere

În cadrul acestui proiect, am dezvoltat o aplicație Java dedicată conversiei eficiente a imaginilor color în imagini grayscale folosind metoda mediei.

2. Descrierea aplicatiei cerute

Scopul Aplicației:

Această aplicație are ca scop conversia unei imagini color la o versiune în tonuri de gri, utilizând metoda mediei.

Funcționalități Principale:

- Conversia de imagini color la imagini grayscale folosind metoda mediei.
- Interacțiune simplă cu utilizatorul pentru specificarea căilor imaginilor de intrare și de ieșire.
- Monitorizarea timpului de execuție pentru evaluarea performanțelor algoritmului.
- Implementarea modelului de execuție producător-consumator pentru gestionarea eficientă a pixelilor imaginii.

3. Partea teoretica

Metoda Mediei pentru Imagine Grayscale

Metoda medie reprezintă o tehnica de conversie a unei imagini color in una în tonuri de gri.

Această metodă se bazează pe principiul fundamental al imaginilor color, care sunt compuse din trei canale principale: roșu (R), verde (G), și albastru (B), adesea cunoscute sub denumirea de modelul RGB.

În cadrul acestei metode, pentru fiecare pixel al imaginii color, se calculează valoarea medie a celor trei canale de culoare. Procesul constă în adunarea valorilor corespunzătoare ale canalelor R, G și B, iar rezultatul obținut este împărțit la 3.

$$\text{Valoare Grayscale} = \frac{R+G+B}{3}$$

In cod:

```
// Metoda pentru conversia pixelului la nuanță de gri
public void convertToGray() {
    // Calculează valoarea medie a componentelor roșie, verde și albastră
    int grayValue = (getRed() + getGreen() + getBlue()) / 3;

    // Setează toate componentele la valoarea medie, obținând astfel o nuanță de gri
    setRed(grayValue);
    setGreen(grayValue);
    setBlue(grayValue);
}
```

4. Descrierea structurala – arhitecturala

Pentru a face aplicația multimodulară și pentru a îndeplini cerințele de ierarhie și niveluri multiple de moștenire, vom organiza proiectul în clase și pachete distincte.

Nivele de mostenire:

Color (Interfață)

- ✓ Este interfața de bază pentru componente de culoare.

RgbColorComponent (Clasă)

- ✓ Implementează: Color.
- ✓ Definește comportamentul pentru componente RGB (roșu, verde, albastru) și furnizează metode pentru manipularea acestora.

GrayScaleConverterAbstract (Clasă Abstractă)

- ✓ Moștenește: RgbColorComponent.
- ✓ Clasă abstractă care adaugă o metodă abstractă pentru conversia la tonuri de gri.

GrayScalePixel (Clasă)

- ✓ Moștenește:GrayScaleConverterAbstract,
- ✓ Implementează conversia unui pixel la tonuri de gri și conține constructori pentru inițializarea pixelilor cu componente RGB sau cu un singur număr întreg.

Image (Clasă)

- ✓ Moștenește: Nicio clasă.
- ✓ Reprezintă o imagine și gestionează matricea de pixeli. Implementează metode pentru manipularea și setarea pixelilor, precum și pentru gestionarea sincronizată a producătorului și consumatorului.

Producer (Clasă)

- ✓ Moștenește: Thread.
- ✓ Extinde clasa Thread pentru a permite rularea pe un fir de execuție separat. Se ocupă de producerea pixelilor

Consumer (Clasă)

- ✓ Moștenește: Thread.
- ✓ Extinde clasa Thread pentru a permite rularea pe un fir de execuție separat. Se ocupă de consumarea pixelilor, conversia lor la tonuri de gri și salvarea rezultatului în fișiere separate.

5. Descrierea implementarii

packWork (Pachetul Pentru clase)

- ✓ ***Clasa Image***: Această clasă este responsabilă pentru gestionarea imaginilor și pixelilor.

  Variabile de clasă:

  - ***pixelMatrix:*** O matrice de obiecte GrayScalePixel reprezentând pixelii imaginii.
  - ***width și height:*** Dimensiunile imaginii.
  - ***completedPixelsCount:*** Numărul de pixeli care au fost procesați complet.
  - ***isAvailable:*** Un indicator care arată dacă sunt disponibili pixeli pentru consum.
  - ***imageCount:*** Un număr total de imagini create (variabilă statică).

  Constructori:

  - ***Image():*** Inițializează o imagine cu dimensiuni implicite de 1x1.
  - ***Image(int width, int height):*** Inițializează o imagine cu dimensiunile specificate.

  Blocuri de Inițializare:

  - ***Blocul static*** este executat atunci când clasa este încărcată și afișează un mesaj despre crearea imaginii.
  - ***Blocul de instanță*** este executat la crearea fiecărui obiect de tip Image și afișează un mesaj despre crearea unui obiect de imagine.

Metode principale:

- ***getPixel(int i, int j) :*** Această metodă primește coordonatele unui pixel și returnează obiectul GrayScalePixel asociat acestor coordonate din matricea imaginii. Dacă coordonatele sunt în afara limitelor imaginii, metoda "aruncă" o excepție de tip ArrayIndexOutOfBoundsException.
- ***setPixel(int i, int j, GrayScalePixel p):*** Această metodă primește coordonatele unui pixel și un obiect GrayScalePixel și setează acel pixel la coordonatele specificate în matricea imaginii. Dacă coordonatele sunt în afara limitelor imaginii, metoda "aruncă" o excepție de tip ArrayIndexOutOfBoundsException.
- ***getWidth():*** Returnează lățimea imaginii
- ***getHeight():*** Returnează inaltimea imaginii
- ***setSize(int width, int height):*** Această metodă primește noi dimensiuni pentru imagine și reinițializează matricea de pixeli conform noilor dimensiuni.
- ***Metode sincronizate pentru producător și consumator:*** Aceste metode sunt sincronizate pentru a asigura o gestionare corectă și sigură a matricei și a numărului de pixeli produși și consumați. Aceste metode sunt utilizate în implementarea modelului producător-consumator.
  - ***setProducedPixel(int i, int j, GrayScalePixel p):*** Această metodă este apelată de producer pentru a seta pixelul la coordonatele specificate și a actualiza numărul de pixeli produși.
  - ***getConsumedPixel(int i, int j):*** Această metodă este apelată de consumator pentru a obține pixelul de la coordonatele specificate și a actualiza numărul de pixeli consumați.

- ***checkBounds(int i, int j):*** Această metodă verifică dacă coordonatele specificate sunt în limitele imaginii. Dacă nu sunt, metoda "aruncă" o excepție de tip ArrayIndexOutOfBoundsException.

- ***waitForAvailability (boolean expected):*** Această metodă așteaptă până când imaginea devine disponibilă sau indisponibilă, în funcție de valoarea așteptată (expected).
- ***handleProducer(int i, int j, GrayScalePixel p, String role) :*** Gestionează producția unui pixel.
- ***handleConsumer(int i, int j, String role):*** Gestionează consumul unui pixel.
- ***handleCompletion(String role, int i, int j) :*** Gestionează finalizarea unei linii de pixeli.
- ***updatePixelAndCount(int i, int j, GrayScalePixel p, boolean isProducer, String role):*** Actualizează matricea și numărul de pixeli în funcție de tipul de operație (producție sau consum).

✓ ***Interfața Color***: Specificarea metodelor pentru gestionarea culorilor.

- ***setRed(int r) :*** Setează componenta roșie a culorii.
- ***getRed() :*** Obține valoarea componentei roșii a culorii.
- ***setGreen(int g):*** Setează componenta verde a culorii.
- ***getGreen():*** Obține valoarea componentei verzi a culorii.
- ***setBlue(int b)***: Setează componenta albastră a culorii.
- ***getBlue():*** Obține valoarea componentei albastre a culorii.

✓ ***Clasa RgbColorComponent***: Această clasă implementează interfața Color și definește comportamentul pentru componente RGB (roșu, verde, albastru).

Variabile de instanță:

- ***red, green, blue***: Componentele RGB.

Metode:

- ***setRed(int red) :*** Setează valoarea componentei roșii, Valoarea este validată folosind metoda privată validateColorValue.
- ***set Green(int green) :*** Setează valoarea componentei verzi, Valoarea este validată folosind metoda privată validateColorValue.
- ***setBlue(int blue) :*** Setează valoarea componentei albastre, Valoarea este validată folosind metoda privată validateColorValue.
- ***getRed(): int :*** Returnează valoarea componentei roșii.
- ***getGreen(): int :*** Returnează valoarea componentei verzi
- ***getBlue(): int :*** Returnează valoarea componentei albastre.
- ***validateColorValue(int value): int:***
  - Validează o valoare pentru o componentă de culoare (roșu, verde, albastru).
  - Dacă valoarea este mai mică decât 0, returnează 0.
  - Dacă valoarea este mai mare decât 255, returnează 255.
  - În caz contrar, returnează valoarea inițială.

✓ ***Clasa GrayScaleConverterAbstract :*** Clasă abstractă care adaugă o metodă abstractă pentru conversia la tonuri de gri.

Metoda abstracta:

- ***public abstract void convertToGray() :*** metodă abstractă pentru conversia la tonuri de gri.

✓ ***Clasa GrayScalePixel:*** Reprezentarea unui pixel în tonuri de gri.

Constructori:

- ***public GrayScalePixel(int red, int green, int blue):*** Inițializează un pixel cu componente RGB specific folosind metodele implementate in "RgbColorComponent"
- ***public GrayScalePixel(int pixel):*** Inițializează un pixel pe baza unui număr întreg care reprezintă valorile RGB combinate.

Metode:

- ***public void convertToGray():*** Convertește pixelul la nuanță de gri prin calcularea valorii medii a componentelor și setarea acestora la valoarea medie.Implementarea metodei abstracte din clasa pe care o extinde : Clasa GrayScaleConverterAbstract ;

✓ ***Clasa Producer :***

Clasa Producer extinde clasa Thread pentru a permite rularea pe un fir de execuție separat. Aceasta se ocupă de procesul de producere a pixelilor dintr-un fișier imagine și introducerea lor într-un obiect de tip Image.

Variabile de Instanță:

- ***Image img :*** Referință către obiectul Image care va fi populat cu pixeli din imaginea sursă.
- ***String filePath:*** Calea către fișierul imagine sursă.
- ***BufferedImage fileImg:*** Obiect BufferedImage pentru citirea imaginii sursă din fișier.
- ***File file :*** Reprezentarea fișierului imagine sursă.

Constructor:

- ***public Producer(Image img, String filePath):***
  - Inițializează obiectul Producer cu o referință la obiectul Image și calea către fișierul imagine sursă.
  - Adaugă extensia BMP la calea fișierului.
  - Inițializează imaginea din fișier.

Bloc de initializare de instanta:

- Executat la crearea fiecărui obiect Producer.
- Afișează un mesaj de consolă pentru a indica crearea unui obiect Producer.

Metode:

- ***public void run()***
  - ***:***Metodă care va fi executată atunci când firul de execuție al Producer este lansat.
  - Măsoară timpul de început și sfârșit al procesării imaginii.
  - Afișează durata procesării imaginii în milisecunde.

- ***private void initializeImageFromFile():***
  - Metodă privată pentru inițializarea imaginii din fișier.
  - Încearcă să citească imaginea sursă într-un obiect BufferedImage.
  - Setează dimensiunile imaginii în obiectul Image.

- ***private void processImagePixels():***
  - Metodă privată pentru procesarea pixelilor imaginii.
  - Iterează prin fiecare pixel al imaginii și îl setează în obiectul Image.
  - Introduce o pauză simulată între citirea segmentelor de informație.

✓ *Clasa Consumer:*

Clasa Consumer extinde clasa Thread și este responsabilă pentru procesul de consumare a pixelilor dintr-un obiect Image, convertirea acestora la tonuri de gri și salvarea rezultatului sub formă de imagine grayscale într-un fișier.

Variabile de Instanță:

- ***static int counter = 1 :*** Contor pentru a număra consumatorii și a crea nume unice pentru fișierele rezultate.
- ***Image img:*** Referință către obiectul Image din care se vor consuma pixeli.
- ***String pathout:*** Calea către directorul unde vor fi salvate fișierele rezultate.

Constructor:

- ***public Consumer(Image img, String pathout):*** Inițializează obiectul Consumer cu o referință la obiectul Image și calea către directorul de ieșire.

Bloc de Inițializare de Instanță:

- Executat la crearea fiecărui obiect Consumer.
- Afișează un mesaj de consolă pentru a indica crearea unui obiect Consumer.

Metode:

- ***public void run():***
  - Metodă care va fi executată atunci când firul de execuție al Consumer este lansat.
  - Generează un nume unic pentru fișierul rezultat.
  - Construiește calea către fișierul rezultat.
  - Creează un obiect BufferedImage pentru a salva rezultatul.
  - Iterează prin fiecare pixel al imaginii de intrare, îl convertește la tonuri de gri și actualizează imaginea rezultat.
  - Salvează imaginea rezultat într-un fișier BMP.

- Afișează durata procesării și calea fișierului rezultat.
- Pentru a evita conflictele la accesarea variabilei counter între mai mulți consumatori, operațiunile care modifică acest contor sunt sincronizate. (*synchronized (Consumer.class)*)

packTest (Pachetul pentru testare)

- ***Clasa MyMain***: Clasa de test și interacțiune cu utilizatorul.

  Această clasă servește drept clasă principală (entry point) a programului și gestionează procesul de citire de la tastatură, creare a obiectelor Image, Producer, și Consumer, lansare a firelor de execuție, măsurare a timpului de execuție, și repetare a procesului pentru mai multe imagini

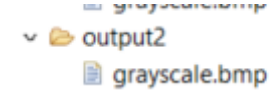
  - Se utilizează un obiect BufferedReader pentru citirea de la tastatură. System.in reprezintă intrarea standard (tastatura).
  - Există un bloc while care se execută atâta timp cât processAnotherImage este true.
  - În cadrul blocului try-catch, utilizatorul este solicitat să introducă calea către fișierul imaginii și calea către folderul unde va fi salvată noua imagine.
  - Se creează un obiect Image pentru a reprezenta imaginea.
  - Se măsoară timpul de execuție începând de la crearea obiectului Image.
  - Se creează și se pornesc firele de execuție Producer și Consumer.
  - Firele de execuție sunt pornite în fire de execuție distincte (start()).
  - Programul așteaptă terminarea firelor de execuție cu join().
  - Se măsoară timpul total de execuție.
  - Utilizatorul este întrebat dacă dorește să proceseze o altă imagine, iar processAnotherImage este actualizat în funcție de răspuns.
  - După încheierea procesului, obiectul BufferedReader este închis pentru a evita scurgeri de resurse.
  - Se afișează un mesaj la încheierea programului.

6. Evaluare Performante

```
Introduceti calea fisierului care contine poze pentru procesare, fara extensia fisierului: ./input1
Introduceti calea folderului unde se va salva noua imagine dupa aplicarea filtrului GrayScale: ./
Consumer consumed 23996000 pixels
Producer produced 24000000 pixels
Producer: Image processing took 8078 milliseconds
Consumer consumed 24000000 pixels
Consumer: Grayscale image creation took 8580 milliseconds. Result saved to: ./\output1\grayscale.bmp
Main Class: Total execution time: 8878 milliseconds
Doriti sa procesati o alta imagine? (Y/N):
```

Input1

```
Doriti sa procesati o alta imagine? (Y/N):
y
Introduceti calea fisierului care contine poze pentru procesare, fara extensia fisierului: ./input2
Introduceti calea folderului unde se va salva noua imagine dupa aplicarea filtrului GrayScale: ./
Consumer consumed 17910720 pixels
Producer produced 17915904 pixels
Producer: Image processing took 5197 milliseconds
Consumer consumed 17915904 pixels
Consumer: Grayscale image creation took 5367 milliseconds. Result saved to: ./\output2\grayscale.bmp
Main Class: Total execution time: 5770 milliseconds
Doriti sa procesati o alta imagine? (Y/N):
n
Programul s-a încheiat.
```



Input2

7. Cod Sursa

MyMain(packTest)

```
package packTest;

// Importarea de biblioteci
import java.io.BufferedReader; //Pentru citirea de la tastatura
import java.io.IOException; //Pentru gestionarea exceptiilor de intrare / iesire
import java.io.InputStreamReader; // Pentru conversia de la fluxul de intrare standard

// clasele din pachetul packWork
import packWork.Consumer;
import packWork.Image;
import packWork.Producer;

public class MyMain {
    public static void main(String...args) {
        // Creăm un obiect BufferedReader pentru citirea de la tastatură
        // BufferedReader este folosit pentru a citi linii de text de la tastatură. System.in reprezintă intrarea standard (tastatura).
        BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));

        // Variabila care controlează dacă se procesează o altă imagine
        boolean processAnotherImage = true;

        //Un bloc while care se execută atâta timp cât processAnotherImage este true.
        while (processAnotherImage) {
            //try-catch este utilizat pentru gestionarea excepțiilor de tip IOException care pot apărea în timpul citirii de la tastatură.
            try {
                // Aceste linii solicită utilizatorului să introducă calea către fișierul imaginii și calea către folderul unde va fi salvată noua imagine. Aceste valori sunt citite de la tastatură.

                // Solicităm calea către fișierul imaginii pentru procesare
                System.out.print("Introduceti calea fisierului care contine poze pentru procesare, fara extensia fisierului: ");
                String pathin = reader.readLine();

                // Solicităm calea către folderul unde se va salva noua imagine după aplicarea filtrului GrayScale
                System.out.print("Introduceti calea folderului unde se va salva noua imagine dupa aplicarea filtrului GrayScale: ");
                String pathout = reader.readLine();

                // Creăm un obiect Image pentru a reprezenta imaginea
                Image img = new Image();

                // Măsurăm timpul de execuție începând de aici
                long startTime = System.currentTimeMillis();

                // Creăm și pornim firele de execuție Producer și Consumer
                Producer P = new Producer(img, pathin);
                Consumer C = new Consumer(img, pathout);

                //Sunt porniți în fire de execuție distincte.
                P.start();
                C.start();

            // Așteptăm terminarea firelor de execuție
            try {
                //Metoda join() este folosită pentru a aștepta ca firul de execuție asupra căruia este apelată să se încheie.
                P.join();
                C.join();
            } //Acesta este blocul catch asociat cu eventuala "aruncare" a unei excepții de tip InterruptedException. Metoda join() poate "arunca" această excepție
            //dacă firul de execuție este întrerupt în timpul așteptării. În cazul în care apare această excepție, blocul catch afișează informații despre excepție folosind e.printStackTrace().
            catch (InterruptedException e) {
                e.printStackTrace();
            }

            // Măsurăm timpul total de execuție
            long endTime = System.currentTimeMillis();
            System.out.println("Main Class: Total execution time: " + (endTime - startTime) + " milliseconds");

            // Întrebăm utilizatorul dacă dorește să proceseze o altă imagine
            System.out.println("Doriti sa procesati o alta imagine? (Y/N): ");
            char response = reader.readLine().charAt(0);
            processAnotherImage = (response == 'Y' || response == 'y');
        } catch (IOException e) {
            //Dacă închiderea obiectului ar eșua și aruncă o excepție de tip IOException, acest bloc de cod capturează excepția
            //și afișează informații despre aceasta folosind e.printStackTrace().
            e.printStackTrace();
        }
    }

    try {
        // Închidem obiectul BufferedReader pentru a evita scurgeri de resurse
        reader.close();
        //Metoda close() este folosită pentru a închide resursele asociate cu obiectul reader. În acest caz, este folosită pentru a închide fluxul de intrare de la tastatură (System.in),
        //care a fost utilizat pentru citirea de la utilizator.
    } catch (IOException e) {
        e.printStackTrace();
    }

    // Afișăm un mesaj la încheierea programului
    System.out.println("Programul s-a încheiat.");
}
```

Color

```java
package packWork;

// Interfața `Color` este destinată să fie implementată de clasa RgbColorComponentAbstract.
public interface Color {

    // Metoda pentru a seta componenta roșie a culorii.
    void setRed(int r);

    // Metoda pentru a obține valoarea componentei roșii a culorii.
    int getRed();

    // Metoda pentru a seta componenta verde a culorii.
    void setGreen(int g);

    // Metoda pentru a obține valoarea componentei verzi a culorii.
    int getGreen();

    // Metoda pentru a seta componenta albastră a culorii.
    void setBlue(int b);

    // Metoda pentru a obține valoarea componentei albastre a culorii.
    int getBlue();
}
```

RgbColorComponent

```java
package packWork;

// Clasa  RgbColorComponent  implementeaza interfata Color si defineste comportamentul pentru componente RGB (rosu, verde, albastru).
public class RgbColorComponent implements Color {
    // Variabilele private pentru a stoca valorile rosii, verzi si albastre.
    private int red, green, blue;

    // Metoda pentru setarea valorii rosii cu validare.
    public void setRed(int red) {
        this.red = validateColorValue(red);
    }

    // Metoda pentru setarea valorii verzi cu validare.
    public void setGreen(int green) {
        this.green = validateColorValue(green);
    }

    // Metoda pentru setarea valorii albastre cu validare.
    public void setBlue(int blue) {
        this.blue = validateColorValue(blue);
    }

    // Metoda pentru obtinerea valorii rosii.
    public int getRed() {
        return red;
    }

    // Metoda pentru obtinerea valorii verzi.
    public int getGreen() {
        return green;
    }

    // Metoda pentru obtinerea valorii albastre.
    public int getBlue() {
        return blue;
    }

    // Metoda privata pentru validarea valorilor si returnarea unei valori valide.
    private int validateColorValue(int value) {
        // Verificam daca valoarea este mai mica decât 0.
        if (value < 0) {
            return 0;  // Returnam 0 dacă este mai mica decât 0.
        } else if (value > 255) {
            return 255;  // Returnam 255 daca este mai mare decat valoarea maxima permisa.
        } else {
            return value;  // Returnam valoarea initiala daca este in intervalul valid.
        }
    }
}
```

GrayScaleConverterAbstract

```java
package packWork;

// Declarăm clasa GrayScaleConverterAbstract ca fiind abstractă și ea moștenește funcționalitățile clasei RgbColorComponent.
public abstract class GrayScaleConverterAbstract extends RgbColorComponent {

    // Declaram o metoda abstracta care nu are implementare in aceasta clasa, dar va fi implementata in clasele derivate.
    // Aceasta metoda este responsabila pentru conversia pixelului la nuanta de gri.
    public abstract void convertToGray();
}
```

GrayScalePixel

```java
package packWork;

// Clasa Pixel reprezinta un pixel intr-o imagine RGB.

public class GrayScalePixel extends GrayScaleConverterAbstract {

    // Constructor cu componente RGB
    public GrayScalePixel(int red, int green, int blue) {
        // Seteaza componentele RGB ale pixelului folosind metodele setRed, setGreen si setBlue mostenite din clasa RgbColorComponent
        setRed(red);
        setGreen(green);
        setBlue(blue);
    }

    // Constructor cu pixel intreg
    public GrayScalePixel(int pixel) {
        //Acest constructor primeste un singur numar intreg pixel ca parametru,
        //reprezentand valorile RGB intr-un singur intreg.
        //Extrage apoi componentele individuale rosie, verde si albastra din intreg  folosind manipularea bitilor
        //si le seteaza folosind metodele setRed, setGreen si setBlue
        setRed((pixel >> 16) & 0xff);
        setGreen((pixel >> 8) & 0xff);
        setBlue(pixel & 0xff);
    }

    // Metoda pentru conversia pixelului la nuanta de gri pentru metoda abstracta din clasa pe care o extinde
    public void convertToGray() {
        // Calculeaza valoarea medie a componentelor rosie, verde si albastra
        int grayValue = (getRed() + getGreen() + getBlue()) / 3;

        // Seteaza toate componentele la valoarea medie, obtinand astfel o nuanta de gri
        setRed(grayValue);
        setGreen(grayValue);
        setBlue(grayValue);
    }
}
```

Producer:

```
// Clasa Producer extinde clasa Thread pentru a permite rularea pe un fir de execuție separat
public class Producer extends Thread {
    private Image img; // Referință către obiectul Image care va fi populat cu pixeli din imaginea sursă
    private String filePath; // Calea către fișierul imagine sursă
    private BufferedImage fileImg; // Obiect BufferedImage pentru citirea imaginii sursă din fișier
    private File file; // Reprezentarea fișierului imagine sursă

    // Constructor care primește un obiect Image și calea către fișierul imagine sursă
    public Producer(Image img, String filePath) {
        this.img = img;
        this.filePath = filePath + ".bmp"; // Adăugăm extensia BMP la calea fișierului
        initializeImageFromFile();
    }

    // Bloc de inițializare de instanță - se execută la crearea fiecărui obiect
    {
        System.out.println("Creating Producer object"); // Mesaj de consolă pentru a indica crearea unui obiect Producer
    }

    // Metodă care va fi executată atunci când firul de execuție al Producer este lansat
    public void run() {

        // Memorăm momentul de început al procesării imaginii
        long startTime = System.currentTimeMillis();

        // Apelăm metoda pentru procesarea pixelilor imaginii
        processImagePixels();

        // Memorăm momentul de final al procesării imaginii
        long endTime = System.currentTimeMillis();

        // Afișăm în consolă durata procesării imaginii
        System.out.println("Producer: Image processing took " + (endTime - startTime) + " milliseconds"); // Afișăm durata procesării
    }

    // Metodă privată pentru inițializarea imaginii din fișier
    private void initializeImageFromFile() {
        try {
            // Creăm un obiect de tip File pe baza căii către fișierul imaginii
            file = new File(this.filePath);

            // Citim imaginea sursă într-un obiect BufferedImage folosind ImageIO.read
            fileImg = ImageIO.read(file);

            // Setăm dimensiunile imaginii în obiectul Image
            img.setSize(fileImg.getWidth(), fileImg.getHeight());

        } catch (IOException e) {
            // În cazul în care apare o excepție (de exemplu, dacă nu se poate citi fișierul), afișăm un mesaj de eroare
            System.out.println("Error reading the image file: " + e.getMessage());
        }
    }
```

```
// Metodă privată pentru procesarea pixelilor imaginii
private void processImagePixels() {
    //lățimea și înălțimea imaginii folosind metodele getWidth() și getHeight() ale obiectului BufferedImage (fileImg).
    int width = fileImg.getWidth();
    int height = fileImg.getHeight();

    // Iterăm prin fiecare pixel al imaginii și îl setăm în obiectul Image
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            // Pentru fiecare pixel din imagine, creăm un obiect de tip GrayScalePixel pe baza valorii RGB a pixelului
            GrayScalePixel grayScalePixel = new GrayScalePixel(fileImg.getRGB(j, i));

            // Setăm pixelul în obiectul Image folosind metoda setProducedPixel
            img.setProducedPixel(i, j, grayScalePixel);
        }

        try {
            // Intră în starea Not Runnable după citirea unui segment de informație
            // (folosind sleep pentru a simula pauza)
            sleep(1000);
        } catch (InterruptedException e) {
            // În cazul în care apare o excepție de tip InterruptedException, afișăm o urmărire a stivei
            e.printStackTrace();
        }
    }
}
```

Consumer:

```java
// Această este declarația de început a fișierului și indică că clasa Consumer aparține pachetului packWork.
package packWork;

// Acestea sunt importurile necesare pentru a utiliza clase din Java Standard Library pentru manipularea imaginilor.
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;

// Clasa Consumer extinde clasa Thread pentru a permite rularea pe un fir de execuție separat
public class Consumer extends Thread {
    private static int counter = 1; // Contor pentru a număra consumatorii și a crea nume unice pentru fișierele rezultate
    private Image img; // Referință către obiectul Image din care se vor consuma pixeli
    private String pathout; // Calea către directorul unde vor fi salvate fișierele rezultate


    // Constructor care primește un obiect Image și calea către directorul de ieșire
    public Consumer(Image img, String pathout) {
        this.img = img;
        this.pathout = pathout;
    }

    // Bloc de inițializare de instanță - se execută la crearea fiecărui obiect
    {
        // Mesaj de consolă pentru a indica crearea unui obiect Consumer
        System.out.println("Consumer has been deployed"); // Mesaj de consolă pentru a indica crearea unui obiect Consumer
    }

    // Metodă care va fi executată atunci când firul de execuție al Consumer este lansat
    public void run() {
        int currentCounter;

        synchronized (Consumer.class) {
            // Incrementăm și memorăm valoarea contorului pentru a crea un nume unic pentru fișierul rezultat
            currentCounter = counter++;
        }

        // Construim calea către fișierul rezultat folosind numele unic generat
        String filePath = pathout + File.separator + "output" + currentCounter + File.separator + "grayscale.bmp";

        File f = null; // Reprezentarea fișierului rezultat
        BufferedImage fileImg = new BufferedImage(img.getWidth(), img.getHeight(), BufferedImage.TYPE_INT_RGB);

        // Memorăm momentul de început al procesării imaginii
        long startTime = System.currentTimeMillis();

        // Iterăm prin fiecare pixel al imaginii și îl convertim la tonuri de gri, salvând rezultatul într-un nou obiect Image
        for (int i = 0; i < img.getHeight(); i++) {
            for (int j = 0; j < img.getWidth(); j++) {
                // Consumăm pixelul din img
                GrayScalePixel consumedPixel = img.getConsumedPixel(i, j);

                // Convertim pixelul la tonuri de gri
                consumedPixel.convertToGray();


        // Construim un pixel RGB pe baza pixelului convertit
        int grayPixel = (consumedPixel.getRed() << 16) | (consumedPixel.getGreen() << 8) | consumedPixel.getBlue();

        // Actualizăm imaginea rezultat
        fileImg.setRGB(j, i, grayPixel);
    }
}


try {
    // Se creează un obiect File asociat căii specificate pentru fișierul rezultat.
    f = new File(filePath);

    // Salvăm imaginea rezultat în format BMP
    ImageIO.write(fileImg, "bmp", f); // Salvăm imaginea rezultat în format BMP

    // Memorăm momentul de final al procesării imaginii
    long endTime = System.currentTimeMillis();

    // Afișează în consolă durata totală a procesării imaginii și calea către fișierul rezultat
    System.out.println("Consumer: Grayscale image creation took " + (endTime - startTime) + " milliseconds. Result saved to: " + filePath);
} catch (IOException e) {
    // În cazul în care apare o excepție de tipul IOException în timpul operațiilor de scriere a imaginii în fișier, se capturează și se afișează un mesaj de eroare în consolă
    System.out.println(e);
}
```

Image

```java
package packWork;

public class Image {
    private GrayScalePixel[][] pixelMatrix; // Matricea de pixeli pentru imagine
    private int width, height, completedPixelsCount; // Dimensiunile imaginii, Numărul de pixeli consumați
    private boolean isAvailable; //Indicator dacă sunt disponibili pixeli pentru consum
    static private int imageCount = 1; //Numărul total de imagini create (variabilă statică)

    public Image() {
        this(1, 1); //Constructor : Inițializează o imagine cu dimensiuni implicite de 1x1.
    }

    // Constructor : Inițializează o imagine cu dimensiunile specificate.
    public Image(int width, int height) {
        this.width = width;
        this.height = height;
        this.pixelMatrix = new GrayScalePixel[height][width];
        this.isAvailable = false;
        imageCount++;
    }

    //Blocuri de Inițializare:
    static {
        //Blocul static este executat atunci când clasa este încărcată și afișează un mesaj despre crearea imaginii.
        imageCount++;
        System.out.println("Static block in Image class - Executed when the class is loaded");
        System.out.println("Created the " + imageCount + "-th image!");
    }

    {
        //Blocul de instanță este executat la crearea fiecărui obiect de tip Image și afișează un mesaj despre crearea unui obiect de imagine.
        System.out.println("Instance block in Image class - Executed when an instance is created");
    }

    public int getWidth() {
        return width; //Returnează lățimea imaginii
    }

    public int getHeight() {
        return height; //Returnează inaltimea imaginii
    }

    public GrayScalePixel getPixel(int i, int j) {
        checkBounds(i, j);
        return pixelMatrix[i][j]; //Returnează pixelul de la coordonatele specificate.
    }

    public void setPixel(int i, int j, GrayScalePixel p) {
        checkBounds(i, j);
        pixelMatrix[i][j] = p; //Setează pixelul la coordonatele specificate.
    }

    public void setSize(int width, int height) {
        pixelMatrix = new GrayScalePixel[height][width];
        this.width = width;
        this.height = height; //Setează noi dimensiuni pentru imagine.
    }

    // Metode Sincronizate pentru Producător și Consumator:

    //Așteaptă disponibilitatea și actualizează matricea și numărul de pixeli produși.
    public synchronized void setProducedPixel(int i, int j, GrayScalePixel p) {
        waitForAvailability(false);
        updatePixelAndCount(i, j, p, true, "Producer");
    }

    //Așteaptă disponibilitatea și actualizează matricea și numărul de pixeli consumați.
    public synchronized GrayScalePixel getConsumedPixel(int i, int j) {
        waitForAvailability(true);
        return updatePixelAndCount(i, j, null, false, "Consumer");
    }

    // Verifică dacă indicele pixelului este în limite.
    private void checkBounds(int i, int j) {
        if (i < 0 || i >= height || j < 0 || j >= width) {
            throw new ArrayIndexOutOfBoundsException("Error: Pixel index out of bounds!");
        }
    }


    // Metoda care așteaptă până când imaginea devine disponibilă sau indisponibilă, în funcție de valoarea așteptată (expected).
     private void waitForAvailability(boolean expected) {
         // Executăm o bucla while atâta timp cât starea de disponibilitate nu este cea așteptată
         while (isAvailable != expected) {
             try {
                 wait();  // Așteptăm notificarea, ceea ce poate schimba starea de disponibilitate
             } catch (InterruptedException e) {
                 e.printStackTrace();  // Tratarea excepției în caz de întrerupere așteptată
             }
         }
     }

    // Metoda care gestionează actualizarea matricei de pixeli și numărarea acestora pentru producător
    private void handleProducer(int i, int j, GrayScalePixel p, String role) {
        pixelMatrix[i][j] = p; // Actualizăm matricea de pixeli cu noul pixel
        completedPixelsCount++; // Incrementăm numărul de pixeli finalizați
        if (completedPixelsCount == width) { // Verificăm dacă toți pixelii pe linie au fost procesați
            handleCompletion(role, i, j); // Dacă da, gestionăm finalizarea procesului
        }
    }
```

```java
// Metoda care gestionează actualizarea matricei de pixeli și numărarea acestora pentru producător
private void handleProducer(int i, int j, GrayScalePixel p, String role) {
    pixelMatrix[i][j] = p; // Actualizăm matricea de pixeli cu noul pixel
    completedPixelsCount++; // Incrementăm numărul de pixeli finalizați
    if (completedPixelsCount == width) { // Verificăm dacă toți pixelii pe linie au fost procesați
        handleCompletion(role, i, j); // Dacă da, gestionăm finalizarea procesului
    }
}

// Metoda care gestionează finalizarea procesului pentru consumator
private void handleConsumer(int i, int j, String role) {
    completedPixelsCount--; // Decrementăm numărul de pixeli finalizați
    if (completedPixelsCount == 0) { // Verificăm dacă toți pixelii pe linie au fost consumați
        handleCompletion(role, i, j); // Dacă da, gestionăm finalizarea procesului
    }
}

// Metoda care gestionează finalizarea procesului atunci când toți pixelii de pe linie au fost procesați sau consumați
private void handleCompletion(String role, int i, int j) {
    isAvailable = !isAvailable; // Schimbăm starea de disponibilitate
    notifyAll(); // Notificăm toate firele de execuție așteptând în cadrul obiectului
    System.out.println(role + " " + (isAvailable ? "produced" : "consumed") + " " + (i * width + j + 1) + " pixels"); // Afișăm un mesaj despre finalizarea procesului
}

// Metoda care actualizează pixelul și numără pixelii finalizați, gestionând excepții în cazul depășirii indicilor
private GrayScalePixel updatePixelAndCount(int i, int j, GrayScalePixel p, boolean isProducer, String role) {
    try {
        if (isProducer) {
            handleProducer(i, j, p, role); // Dacă este producător, apelăm metoda de gestionare a producătorului
        } else {
            handleConsumer(i, j, role); // Dacă este consumator, apelăm metoda de gestionare a consumatorului
        }
        return pixelMatrix[i][j]; // Returnăm pixelul actualizat
    } catch (ArrayIndexOutOfBoundsException e) {
        e.printStackTrace();
        System.out.println("Error: Pixel index out of bounds!"); // Afișăm un mesaj de eroare în caz de depășire a indicilor matricei
        return null;
    }
}
```

8. Concluzii

În concluzie, proiectul demonstrează eficiența unei implementări paralele utilizând fire de execuție distincte pentru producerea și consumarea pixelilor unei imagini.

- ✓ ***Procesare Imagini Paralelă:*** Implementarea evidențiază abordarea procesării paralele a imaginilor, folosind producători și consumatori pentru a manipula pixelii imaginii în mod eficient pe fire de execuție separate.

- ✓ ***Sincronizare și Comunicare:*** Sincronizarea corectă a operațiilor asupra datelor partajate (pixelMatrix) și comunicarea între producător și consumator sunt gestionate eficient pentru a evita conflictele și pentru a menține coerența datelor

- ✓ ***Interacțiunea cu Utilizatorul:*** Utilizatorul interacționează prin introducerea căilor către fișierele de intrare și ieșire, permițând astfel procesarea mai multor imagini într-un mod consecutiv.

- ✓ ***Utilizarea Interfețelor și Claselor Abstracte:*** Se utilizează interfețe și clase abstracte pentru a defini și implementa comportamente specifice legate de culori, facilitând astfel modularitatea codului.

- ✓ ***Manipularea Excepțiilor:*** Gestionarea excepțiilor, în special cele legate de citirea și scrierea fișierelor de imagine, este inclusă în cod pentru a asigura o execuție robustă și pentru a furniza informații utile în caz de erori.

- ✓ ***Măsurarea Performanței:*** Se măsoară timpul de execuție al procesării imaginilor, oferind utilizatorului informații despre performanța programului.

- ✓ ***Structura OOP (Programare Orientată pe Obiecte):*** Implementarea respectă principiile OOP, utilizând clase, obiecte, moștenire, incapsulare și polimorfism pentru a organiza și structura codul într-un mod clar și modular.

9. Bibliografie

1. *Suport Curs AWJ.* (2023-2024).

2. Dyclassroom. *How to convert a color image into grayscale image in Java*. Preluat de pe "https://dyclassroom.com/image-processing-project/how-to-convert-a-color-image-into-grayscale-image-in-java".

3. geeksforgeeks. (n.d.). *Image Processing in Java – Colored Image to Grayscale Image Conversion*. Retrieved from "https://www.geeksforgeeks.org/image-processing-in-java-colored-image-to-grayscale-image-conversion/".