Image Brightness Modification

Proiect Aplicații Web cu Suport Java

Proiect realizat de: Marin Andreea- Iulia

Universitatea Politehnică din București

An universitar: 2023-2024

Table of Contents

1. Introducere

Prezentarea generală a proiectului

Proiectul "Image Brightness Modification" propune dezvoltarea unei aplicații specializate în manipularea imaginilor digitale. Scopul principal al proiectului este furnizarea unui instrument flexibil și ușor de utilizat pentru ajustarea luminozității imaginilor, cu posibilitatea de a opera asupra unor formate populare, precum BMP.

BMP (Bitmap) este un format de fișier de imagine, utilizat în principal pentru stocarea de imagini pe sistemele de operare Windows. Acest format este cunoscut și sub denumirea de fișier bitmap sau DIB (Device-Independent Bitmap). Imaginile BMP sunt formate dintr-o grilă de pixeli, în care fiecare pixel stochează informații despre culoarea sa.

Principalele caracteristici ale formatului BMP includ:

- Fără Compresie: Fișierele BMP sunt adesea necompresate, ceea ce înseamnă că păstrează informațiile despre fiecare pixel fără a le reduce sau a le comprima. Aceasta face ca fișierele BMP să fie relativ mari în comparație cu formatele de fișiere comprimate.

- Suport pentru Culoare: BMP suportă diferite profunzimi de culoare, inclusiv alb-negru, scala de griuri și imagini color cu o gamă variată de culori.

- Structură Simplă: Fișierele BMP au o structură destul de simplă, cu un antet care conține informații despre dimensiunile imaginii, numărul de culori și altele. Această structură simplă face fișierele BMP relativ ușor de procesat.

- Compatibilitate cu Windows: BMP a fost inițial dezvoltat de Microsoft și este strâns legat de sistemele de operare Windows. Acest format este compatibil cu o varietate de programe și aplicații pe platforma Windows.

Deși BMP este un format robust și simplu, dimensiunile mari ale fișierelor și absența compresiei pot face ca alte formate, cum ar fi JPEG sau PNG, să fie preferate în anumite contexte, cum ar fi stocarea pe internet sau transferul rapid al imaginilor.

Obiective

- Ajustarea Luminozității: Dezvoltarea unui algoritm eficient pentru modificarea luminozității imaginilor, permițând utilizatorilor să obțină rezultate vizuale dorite.

- Interfață Utilizator Prietenoasă: Implementarea unei interfețe grafice intuitive, accesibilă pentru utilizatorii de toate nivelurile de experiență.

- Sincronizare și Procesare Paralelă: Utilizarea mecanismelor de sincronizare pentru gestionarea eficientă a proceselor paralele în cadrul aplicației, oferind performanțe ridicate în manipularea imaginilor.

- Utilizare a Tehnologiilor Java: Implementarea proiectului în limbajul de programare Java, folosind principii de programare orientată pe obiect și beneficiind de facilitățile oferite de platforma Java.

Motivație

Proiectul are la bază nevoia crescută de manipulare a imaginilor digitale în diverse contexte, precum prelucrarea de fotografii, grafică digitală sau aplicații de analiză de imagine. O aplicație specializată pentru ajustarea luminozității poate avea aplicabilitate extinsă în industrie, artă sau cercetare științifică.

De asemenea, dezvoltarea în limbajul Java vine în întâmpinarea cerințelor de portabilitate și performanță, oferind o soluție eficientă și accesibilă.

2. Cerințe de implementare

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 unui segment 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.

3. Descriere a Arhitecturii Structurale a Proiectului

Descrierea pachetelor și a claselor

1) Pachete

Pachetul image:

Conține clasele legate de manipularea imaginilor și procesarea acestora.

Pachetul test:

Include clasa principală Test, responsabilă pentru testarea funcționalității întregului sistem.

1) Clase și Ierarhii

Clase în interiorul pachetului image

- Interfata

Clasa Interfata este de fapt o interfață în Java, nu o clasă. Interfața este un tip de structură în Java care definește un set de metode abstracte (metode fără implementare) pe care orice clasă care implementează această interfață trebuie să le furnizeze. În cadrul proiectului, această interfață este folosită pentru a specifica metodele comune pe care toate clasele legate de manipularea imaginilor ar trebui să le implementeze.

În esență, această interfață definește cinci metode:

```java
package image;

import java.awt.image.BufferedImage;

//Interfata pentru definirea metodelor comune pentru gestionarea imaginilor
public interface Interfata {
 // Returneaza calea imaginii
 String getPath();

 // Seteaza calea imaginii
 void setPath(String path);

 // Returneaza imaginea
 BufferedImage getImg();

 // Seteaza imaginea
 void setImg(BufferedImage img);

 // Ajusteaza luminozitatea imaginii
 void adjustBrightness(int factor);
}
```

getPath(): Returnează calea imaginii.

setPath(String path): Setează calea imaginii pe baza unui șir de caractere dat.

getImg(): Returnează imaginea sub formă de obiect BufferedImage.

setImg(BufferedImage img): Setează imaginea cu un obiect BufferedImage dat.

adjustBrightness(int factor): Ajustează luminozitatea imaginii pe baza unui factor dat.

Prin intermediul acestei interfețe, toate clasele care doresc să manipuleze imagini în cadrul proiectului trebuie să ofere implementări pentru aceste metode. Astfel, interfața stabilește un contract pentru funcționalitățile comune în manipularea imaginilor.

- AbstractImage

Clasa AbstractImage este o clasă abstractă care servește ca un șablon comun pentru toate clasele de imagini din cadrul proiectului. Această clasă abstractă implementează interfața Interfata și furnizează o implementare de bază pentru unele dintre metodele acesteia.

```java
package image;

import java.awt.image.BufferedImage;

// Clasa abstracta folosita ca template pentru clasa Image
public abstract class AbstractImage implements Interfata {

    private BufferedImage img;

    // Setter - scrie in obiect imaginea sursa
    public void setImg(BufferedImage img) {
        this.img = img;
    }

    // Getter - returneaza imaginea stocata
    public BufferedImage getImg() {
        return img;
    }

    // Metoda pentru ajustarea luminozitatii
    public abstract void adjustBrightness(int factor);

}
```

Principalele aspecte ale clasei AbstractImage sunt:

img (BufferedImage): Această variabilă de instanță reprezintă imaginea propriu-zisă și este utilizată pentru a stoca imaginea în cadrul obiectelor derivate.

setImg(BufferedImage img): Această metodă furnizează o modalitate de a seta imaginea în obiectul AbstractImage. Este utilizată pentru a încărca o imagine în obiect.

getImg(): BufferedImage Această metodă furnizează o modalitate de a obține imaginea stocată în obiectul AbstractImage.

adjustBrightness(int factor): Aceasta este o metodă abstractă pe care clasele derivate trebuie să o implementeze. Metoda este responsabilă pentru ajustarea luminozității imaginii pe baza unui factor dat.

Fiind o clasă abstractă, AbstractImage nu poate fi instantiată direct. În schimb, ea furnizează o bază comună pentru clasele concrete care vor fi derivate din ea, cum ar fi BrightnessModification.

❖ Image

Clasa Image reprezintă o implementare concretă a clasei abstracte AbstractImage și implementează interfața Interfata. Această clasă are rolul de a reprezenta o imagine și de a oferi metode pentru manipularea și gestionarea acesteia. Iată implementarea clasei Image:

```java
package image;

public class Image extends AbstractImage {

    private String path;

    // Bloc de initializare de instanta
    {
        path = "rezultat.bmp";
    }

    // Constructor implicit
    public Image() {
        super(); // Apelam constructorul clasei parinte
    }

    // Constructor cu path dat
    public Image(String path) {
        super(); // Apelam constructorul clasei parinte
        this.path = path;
    }

    // Setare path catre imagine
    public void setPath(String path) {
        this.path = path;
    }

    // Returnare path al imaginii
    public String getPath() {
        return path;
    }
```

Principalele aspecte ale clasei Image sunt:

path (String): Reprezintă calea către imagine. Această variabilă este inițializată cu o valoare implicită în blocul de inițializare de instanță.

Constructori: Clasa dispune de doi constructori - unul implicit și unul care primește calea către imagine. Ambii constructori apelează constructorul clasei părinte (super()) pentru a inițializa proprietățile moștenite.

setPath(String): Metodă pentru setarea caii către imagine.

getPath(): String: Metodă pentru obținerea căii imaginii.

adjustBrightness(int): Metodă suprascrisă din clasa părinte (AbstractImage). În clasa Image, această metodă poate fi implementată specific pentru ajustarea luminozității a imaginilor. În exemplul dat, metoda este goală și așteaptă a fi implementată în clasele derivate.

Clasa Image servește drept bază pentru alte clase concrete de imagini și pune la dispoziție metoda abstractă adjustBrightness, care poate fi implementată diferit în funcție de nevoile claselor derivate.

- BrightnessModification

Clasa BrightnessModification este o clasă concretă care extinde clasa abstractă AbstractImage. Aceasta adaugă funcționalități specifice pentru ajustarea luminozității imaginilor. În plus, clasa implementează metodele specifice pentru citirea, scrierea și afișarea imaginilor procesate.

```
1  package image;
2
3  import java.awt.Color;
4  import java.awt.Desktop;
5  import java.awt.image.BufferedImage;
6  import java.io.File;
7  import java.io.IOException;
8  import javax.imageio.ImageIO;
9
10 // Clasa BrightnessModification extinde clasa Image
11 public class BrightnessModification extends Image {
12     // Calea de iesire pentru imagine
13     private String outputPath;
14
15     // Constructorul implicit, seteaza calea imaginii sursa
16     public BrightnessModification() {
17         setPath("in.bmp");
18     }
19
20     // Metoda pentru scrierea imaginii in fisier
21     public void write() throws IOException {
22         ImageIO.write(getImg(), "bmp", new File(getPath()));
23     }
24
25     // Setter pentru calea de iesire
26     public void setOutputPath(String outputPath) {
27         this.outputPath = outputPath;
28     }
29
30     // Getter pentru calea de iesire
31     public String getOutputPath() {
```

Codul întreg se poate găsi în fișierul sursă al proiectului.

Principalele aspecte ale clasei BrightnessModification sunt:

outputPath (String): Calea de ieșire pentru imaginea procesată.

write(): void: Metodă pentru scrierea imaginii într-un fișier.

setOutputPath(String outputPath): void: Metodă pentru setarea căii de ieșire.

read(): void și read(String path): void: Metode pentru citirea imaginii de la căi specificate.

writeOutput(): void: Metodă pentru scrierea imaginii procesate într-un fișier la calea specificată.

displayImage(): void: Metodă pentru afișarea imaginii procesate.

adjustBrightness(int factor): void: Suprascrierea metodei de ajustare a luminozității, care modifică culorile pixelilor imaginii pe baza unui factor dat.

- ConsumerThread

Clasa ConsumerThread implementează un thread consumator în cadrul unui sistem de tip Producer-Consumer. Acest thread este responsabil pentru procesarea imaginii furnizate de către ProducerThread după ce a primit notificarea că imaginea este pregătită pentru procesare. Mai jos este implementarea clasei ConsumerThread:

```java
package image;
// In clasa ConsumerThread
import java.io.IOException;

// Clasa pentru definirea unui thread consumator
public class ConsumerThread implements Runnable {
    // Referinta catre obiectul partajat intre thread-uri
    private final SharedData sharedData;

    // Constructor care primeste obiectul partajat
    public ConsumerThread(SharedData sharedData) {
        this.sharedData = sharedData;
    }

    // Metoda care va fi executata de catre thread-ul consumator
    @Override
    public void run() {
        try {
            // Asteptam notificarea ca imaginea este pregatita pentru procesare
            sharedData.waitImageProcessed();

            // Procesam imaginea
            System.out.println("Consumer Thread: Procesare imagine...");
            long startTime = System.nanoTime();
            sharedData.getImageProcessor().adjustBrightness(sharedData.getFactor());
            long processingTime = System.nanoTime() - startTime;
            System.out.println("Timp procesare:   " + processingTime + " nanosecunde");
            System.out.println("Consumer Thread: Imagine procesata.");

            // Scriem imaginea procesata in fisier
            sharedData.getImageProcessor().writeOutput();
```

Principalele aspecte ale clasei ConsumerThread sunt:

sharedData (SharedData): Este o referință către obiectul SharedData care conține datele partajate între thread-uri.

run(): void: Este metoda care va fi executată de către thread-ul consumator. În cadrul acestei metode, thread-ul așteaptă notificarea că imaginea este pregătită pentru procesare (waitImageProcessed). După ce primește notificarea, thread-ul efectuează procesarea imaginii, măsoară timpul de procesare și afișează informații relevante. Ulterior, imaginea procesată este scrisă în fișier, iar producătorului i se transmite notificarea că procesarea este terminată (setImageReady).

Gestionarea excepțiilor: Se gestionează excepțiile de tip InterruptedException și IOException prin afișarea trace-ului stivei (stack trace) în cazul apariției acestora. Este important să tratăm excepțiile corespunzător în implementările reale.

- ProducerThread

Clasa ProducerThread implementează un thread producător în cadrul unui sistem de tip Producer-Consumer. Acest thread are responsabilitatea de a citi o imagine dintr-un fișier și de a o face disponibilă pentru procesare. Iată implementarea clasei ProducerThread:

```java
package image;
//Clasa pentru implementarea thread-ului producator
import java.io.IOException;

public class ProducerThread implements Runnable {
 // Referinta catre obiectul partajat
 private final SharedData sharedData;

 // Constructor
 public ProducerThread(SharedData sharedData) {
     this.sharedData = sharedData;
 }

 // Metoda run, care contine logica executata de thread-ul producator
 @Override
 public void run() {
     try {
         System.out.println("Producer Thread: Citire imagine...");
         sharedData.getImageProcessor().read();
         System.out.println("Producer Thread: Imagine citita.");

         sharedData.setImageReady(); // Notificam ca imaginea este gata pentru procesare
     } catch (IOException e) {
         e.printStackTrace();
     }
 }
}

```

Principalele aspecte ale clasei ProducerThread sunt:

sharedData (SharedData): Reprezintă o referință către obiectul SharedData, care gestionează datele partajate între thread-uri.

Constructor: Clasa dispune de un constructor care primește ca argument obiectul SharedData. Acest constructor este utilizat pentru a inițializa referința sharedData.

run(): Este metoda definitorie a interfeței Runnable și conține logica executată de thread-ul producător. În cadrul acestei metode, se apelează metoda read() a obiectului ImageProcessor pentru a citi imaginea și apoi se utilizează metoda setImageReady() pentru a notifica celelalte thread-uri că imaginea este pregătită pentru procesare.

Clasa ProducerThread face parte din arhitectura Producer-Consumer, în care rolul său este de a produce (în acest caz, citi) date (imaginile) și de a le pune la dispoziție pentru procesare de către thread-urile consumatoare.

❖ SharedData

Clasa SharedData este responsabilă pentru gestionarea datelor partajate între thread-uri în cadrul sistemului. Aceasta conține informații despre imaginea ce urmează a fi procesată, obiectul de procesare a imaginii (BrightnessModification), factorul de ajustare a luminozității și calea pentru salvarea imaginii procesate. De asemenea, oferă metode pentru a notifica și a aștepta thread-urile în cadrul modelului Producer-Consumer. Iată implementarea clasei SharedData:

```java
package image;

//Clasa pentru gestionarea datelor partajate intre thread-uri
import java.util.concurrent.locks.Condition;
import java.util.concurrent.locks.Lock;
import java.util.concurrent.locks.ReentrantLock;

public class SharedData {
 // Referinta catre obiectul de procesare a imaginii
 private final BrightnessModification imageProcessor;
 // Factorul de ajustare a luminozitatii
 private final int factor;
 // Calea pentru salvarea imaginii procesate
 private final String outputPath;
 // Flag pentru a indica daca imaginea este gata pentru procesare
 private boolean isImageReady = false;

 // Adaugam lock si conditie pentru sincronizare
 private final Lock lock = new ReentrantLock();
 private final Condition imageReadyCondition = lock.newCondition();

 // Constructor
 public SharedData(BrightnessModification imageProcessor, int factor, String outputPath) {
     this.imageProcessor = imageProcessor;
     this.factor = factor;
     this.outputPath = outputPath;

     // Setam calea de iesire in obiectul BrightnessModification
     imageProcessor.setOutputPath(outputPath);
 }
```

Principalele aspecte ale clasei SharedData sunt:

Obiecte Partajate:

imageProcessor (BrightnessModification): Reprezintă obiectul de procesare a imaginii.

factor (int): Reprezintă factorul de ajustare a luminozității.

outputPath (String): Reprezintă calea pentru salvarea imaginii procesate.

Sincronizare cu Lock și Condition:

Se utilizează un obiect Lock pentru a asigura sincronizarea în cadrul accesului concurent la datele partajate.

Se utilizează o condiție (imageReadyCondition) pentru a notifica și a aștepta thread-urile în cazul în care imaginea este gata pentru procesare.

Metode:

setImageReady(): Setează flag-ul pentru a indica că imaginea este pregătită pentru procesare și notifică alte thread-uri.

waitImageProcessed(): Așteaptă până când imaginea este procesată, utilizând condiția imageReadyCondition.

getImageProcessor(): Returnează obiectul de procesare a imaginii.

getFactor(): Returnează factorul de ajustare a luminozității.

getOutputPath(): Returnează calea pentru salvarea imaginii procesate.

Clase în interiorul pachetului test

❖ Test

Clasa Test din pachetul test reprezintă clasa principală a aplicației, responsabilă pentru testarea funcționalității întregului sistem. Aceasta conține metoda main, care servește ca punct de intrare în program. Iată implementarea clasei Test:

```java
// Importurile necesare pentru clasele din proiect
package test;

import image.BrightnessModification;
import image.SharedData;
import image.ProducerThread;
import image.ConsumerThread;
import java.io.IOException;
import java.util.Scanner;

// Clasa principala de test
public class Test {

    @SuppressWarnings("resource")
    // Metoda principala a aplicatiei
    public static void main(String[] args) throws IOException, InterruptedException {
        // Initializarea obiectului pentru procesarea imaginilor
        BrightnessModification imageProcessor = null;

        // Variabila pentru masurarea timpilor
        long startTime;

        // Variabile pentru calea de citire, calea de scriere si factorul de luminozitate
        String readPath = null;
        String outputPath = null;
        int factor = 0;

        // Verificare daca sunt suficienti parametrii in linia de comanda
        if (args.length < 3) {
            // Folosim input de la tastatura
            Scanner sc = new Scanner(System.in);
```

Principalele aspecte ale clasei Test sunt:

Punct de Intrare (main):

Metoda main servește ca punct de intrare în program.

Verifică dacă parametrii sunt furnizați fie de la linia de comandă, fie de la tastatură.

Manipularea Imaginilor:

Instantiază obiectul BrightnessModification pentru procesarea imaginilor.

Gestionarea Datelor Partajate:

Crează obiectul SharedData care va fi partajat între thread-uri.

Crearea și Pornirea Thread-urilor:

Inițiază și pornește thread-ul producător (ProducerThread).

Inițiază și pornește thread-ul consumator (ConsumerThread).

Așteptarea Terminării Thread-urilor:

Folosește join pentru a aștepta terminarea ambelor thread-uri.

Afisarea Imaginii:

Folosește metoda displayImage pentru a afișa imaginea procesată.

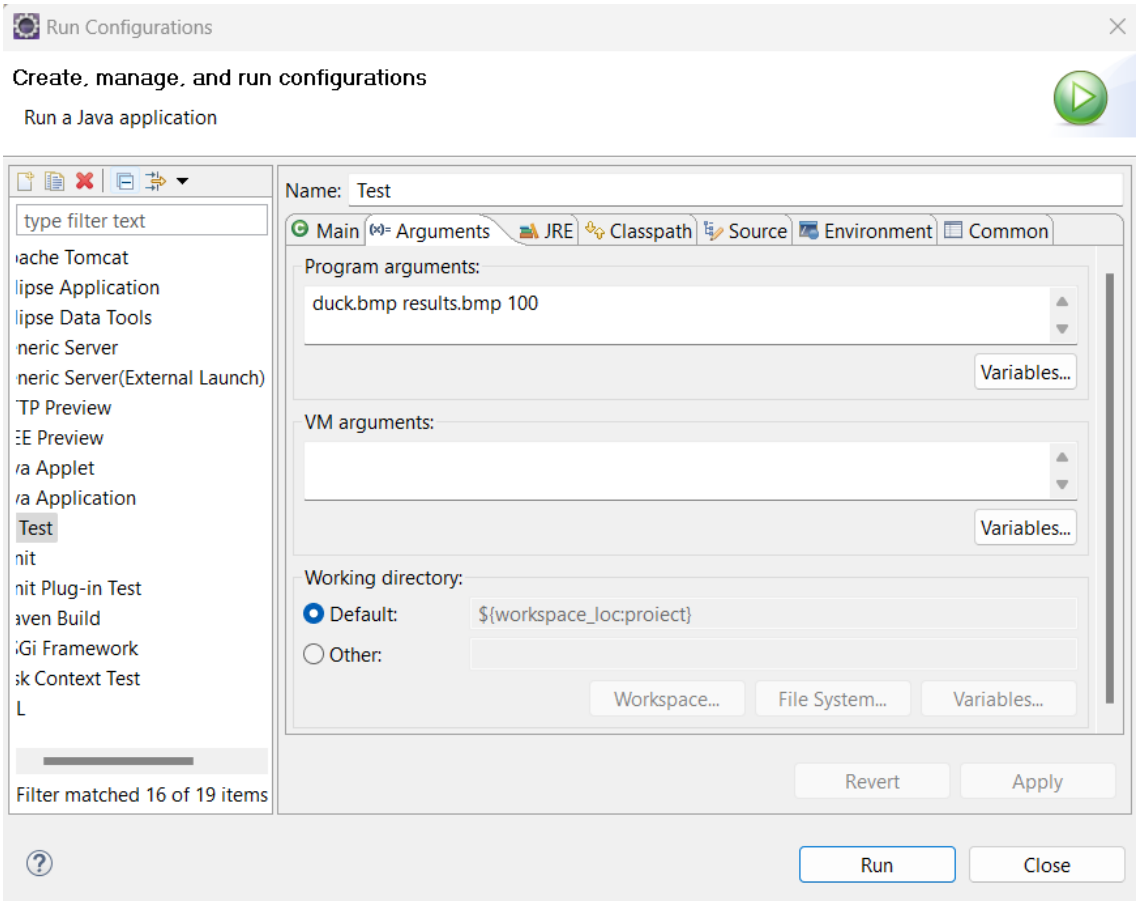
Descrierea generală a fluxului de date

Principalele aspecte ale clasei **Test** sunt:

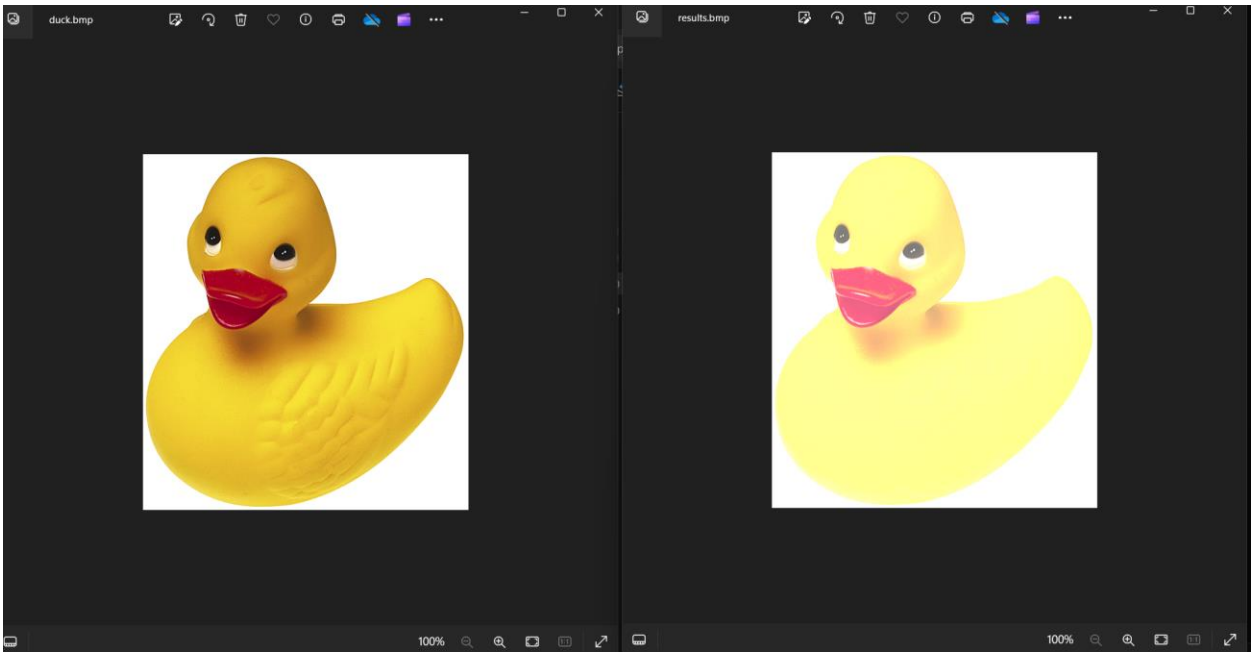
1. **Punct de Intrare (main):**
    - Metoda **main** servește ca punct de intrare în program.
    - Verifică dacă parametrii sunt furnizați fie de la linia de comandă, fie de la tastatură.
2. **Manipularea Imaginilor:**
    - Instantiază obiectul **BrightnessModification** pentru procesarea imaginilor.
3. **Gestionarea Datelor Partajate:**
    - Crează obiectul **SharedData** care va fi partajat între thread-uri.
4. **Crearea și Pornirea Thread-urilor:**
    - Inițiază și pornește thread-ul producător (**ProducerThread**).
    - Inițiază și pornește thread-ul consumator (**ConsumerThread**).
5. **Așteptarea Terminării Thread-urilor:**
    - Folosește **join** pentru a aștepta terminarea ambelor thread-uri.
6. **Afisarea Imaginii:**
    - Folosește metoda **displayImage** pentru a afișa imaginea procesată.

4. Rezultate obținute

În urma introducerii următoarelor valori ale argumentelor liniei de comandă:



Se poate observa că mărirea luminozității a fost efectuată cu succes:



Și următorii timpi de citire/procesare:

```
Timp citire:    46890400 nanosecunde
Producer Thread: Citire imagine...
Producer Thread: Imagine citita.
Consumer Thread: Procesare imagine...
Timp procesare:   43191400 nanosecunde
Consumer Thread: Imagine procesata.
Calea fisierului care va fi deschis: C:\Users\andre\workspace\proiect\results.bmp
```

În lipsa argumentelor din linia de comanda, utilizatorul este rugat să introducă de la tastatură parametrii necesari.

5. Concluzii

Cerințele de implementare au fost îndeplinite, proiectul are o moștenire pe 4 niveluri(Interface->AbstractImage->Image->BrightnessModification), include concepte POO(abstractizare, moștenire,polimorfism, constructori etc.), implementează cu succes atât diminuarea luminozității(introducerea unei valori negative pentru factorul de modificare a luminozității) dar și de mărire (factor pozitiv) și folosește comunicarea Producer-Consumer pentru implementare.

6. Bibliografie

- Java Documentation: Oracle. "The Java™ Tutorials." https://docs.oracle.com/javase/tutorial/

- Programare Multithreading în Java:

Goetz, B. (2006). "Java Concurrency in Practice." Addison-Wesley.

- Manipularea Imaginilor în Java:

"How to Read and Write Image File in Java." JournalDev. https://www.journaldev.com/615/java-read-write-image-file

"Java - How to draw image with BufferedImage.drawImage." Stack Overflow. https://stackoverflow.com/questions/196890/java-how-to-draw-image-with-bufferedimage-drawimage

- Producer-Consumer Pattern:

"Producer-Consumer Problem." GeeksforGeeks. https://www.geeksforgeeks.org/producer-consumer-problem/

Goetz, B., Peierls, T., Bloch, J., Bowbeer, J., Holmes, D., & Lea, D. (2006). "Java Concurrency in Practice." Addison-Wesley.

- Lucrul cu Fișiere în Java:

"Java - File I/O." TutorialsPoint. https://www.tutorialspoint.com/java/java_files_io.htm

- Programare Orientată pe Obiect (POO):

Eckel, B. (2003). "Thinking in Java." Prentice Hall.

- Java Exception Handling:

"Java Exception Handling – try, catch, throw, throws, finally." JournalDev. https://www.journaldev.com/1733/java-exception-handling-tutorial