

# Image Smoothing (Convolution Mask)

## Aplicații Web cu Suport Java

**Student: Codescu Elisabeta Maria**

**Grupa: 331AC**

**Anul Universitar 2022-2023, Facultatea de Automatică și Calculatoare, UPB**

### Introducere și Descrierea Aplicației Cerute

Proiectul are ca temă Image Smoothing with Convolution Mask și realizează blurarea oricărei imagini de tip 24bit bmp introdusă ca input, generând o nouă imagine, tot de tip 24bit bmp. Acest proces de blurare a fost realizat în întregime în Eclipse IDE și are la bază utilizarea metodei convolution mask.

### Partea Teoretică

Procesul de blurare implementat în aplicație se folosește de matricea de convoluție, numită și kernel (denumire utilizată de altfel în cadrul aplicației). Acest kernel este, de fapt, o matrice de dimensiuni restrânse. Utilizările acestei matrici nu se limitează însă doar la metoda de blurare exemplificată în acest proiect: ea poate fi folosită pentru multiple alte prelucrări de imagini, precum sharpening, edge detection etc. După generarea kernelului, are loc convoluția între kernel și imaginea suport.

Cunoaștem expresia algebrică a unei convoluții, adaptată pentru determinarea kernelului:

$$g(x, y) = \omega * f(x, y) = \sum_{dx=-a}^a \sum_{dy=-b}^b \omega(dx, dy) f(x - dx, y - dy),$$

unde  $g(x,y)$  este imaginea căreia i-a fost aplicat filtrul kernel,  $\omega$  este filtrul kernel, iar  $f(x,y)$  este imaginea originală de prelucrat.

Procesul de convoluție constă în adunarea fiecărui pixel din imagine la pixelii învecinați, ponderați cu ajutorul kernelului. Mai în detaliu, putem considera drept exemplu două matrici de dimensiune 3x3, prima dintre ele fiind kernelul, iar cea de-a doua fiind imaginea de prelucrat. Convoluția are loc prin inversarea atât a rândurilor, cât și a coloanelor matricii kernel și multiplicarea și însumarea intrărilor la nivel local.

Elementul de pe rândul 2 și coloana 2 din matricea imaginii prelucrate va avea următoarea formulă de determinare:

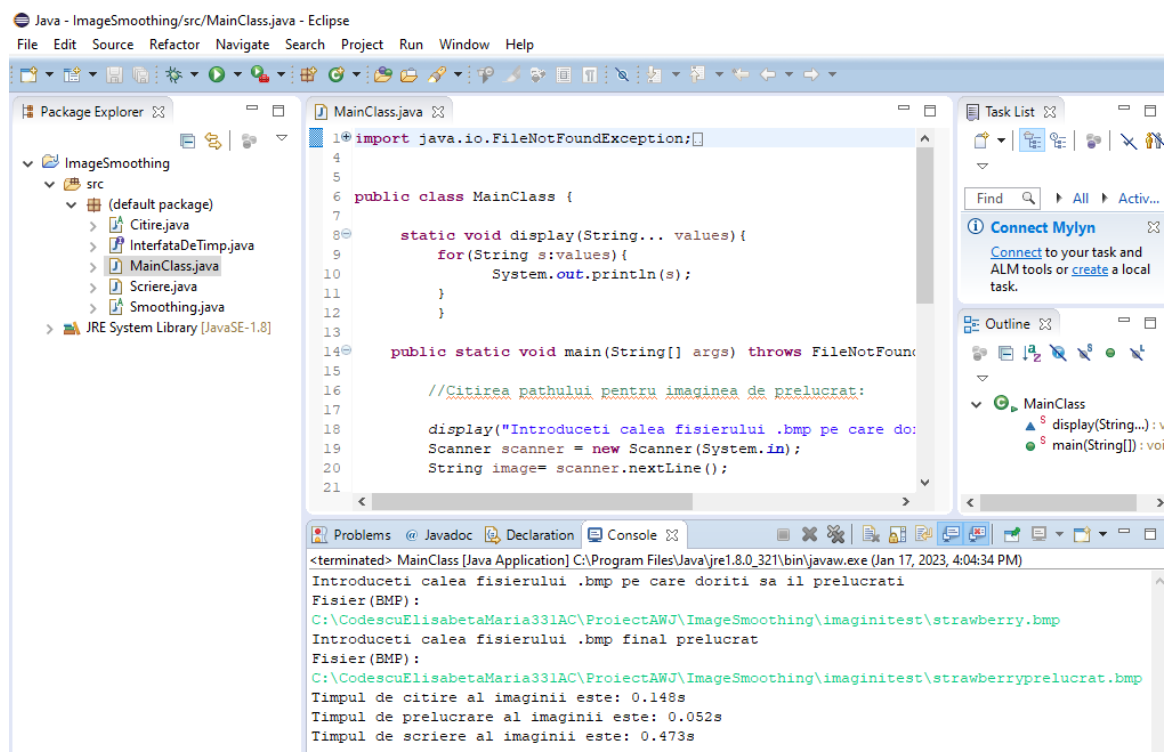
$$\left( \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} * \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \right) [2, 2] = (i \cdot 1) + (h \cdot 2) + (g \cdot 3) + (f \cdot 4) + (e \cdot 5) + (d \cdot 6) + (c \cdot 7) + (b \cdot 8) + (a \cdot 9).$$

## Descrierea funcțională a aplicației

Pașii pe care îi parcurge aplicația sunt:

1. Calea introdusă din consolă ce corespunde imaginii ce se dorește a fi procesată este preluată
2. Este preluată, de asemenea, tot din consolă, și calea unde se dorește scrierea imaginii procesate
3. Este citită imaginea suport, în format 24bit BMP
4. Simultan cu citirea imaginii, se contorizează timpul necesar citirii acesteia și se stochează într-o variabilă
5. Are loc procesarea efectivă a imaginii și netezirea acesteia
6. Simultan cu procesul de blurare/netezire, se contorizează timpul necesar procesării imaginii
7. Este scrisă imaginea finală, la adresa introdusă la pasul 2
8. Simultan cu scrierea imaginii, se contorizează timpul de scriere
9. Se afișează toți cei trei timpi – de citire, procesare și scriere, și se încheie execuția aplicației.

În captura de ecran inclusă mai jos, se poate observa testarea aplicației cu scopul blurării unei poze cu un coș de capșuni:



The screenshot displays the Eclipse IDE interface. The Package Explorer on the left shows the project structure: ImageSmoothing (src) containing Citire.java, InterfataDeTimp.java, MainClass.java, Scriere.java, and Smoothing.java. The MainClass.java file is open in the editor, showing the following code:

```
1 import java.io.FileNotFoundException;
4
5
6 public class MainClass {
7
8     static void display(String... values) {
9         for (String s : values) {
10             System.out.println(s);
11         }
12     }
13
14     public static void main(String[] args) throws FileNotFoundException {
15
16         //Citirea pathului pentru imaginea de prelucrat:
17
18         display("Introduceti calea fisierului .bmp pe care doriti sa il prelucrati");
19         Scanner scanner = new Scanner(System.in);
20         String image = scanner.nextLine();
21     }
22 }
```

The Console window at the bottom shows the execution output:

```
<terminated> MainClass [Java Application] C:\Program Files\Java\jre1.8.0_321\bin\javaw.exe (Jan 17, 2023, 4:04:34 PM)
Introduceti calea fisierului .bmp pe care doriti sa il prelucrati
Fisier (BMP) :
C:\CodescuElisabetaMaria331AC\ProiectAWJ\ImageSmoothing\imagine\test\strawberry.bmp
Introduceti calea fisierului .bmp final prelucrat
Fisier (BMP) :
C:\CodescuElisabetaMaria331AC\ProiectAWJ\ImageSmoothing\imagine\test\strawberryprelucrat.bmp
Timpul de citire al imaginii este: 0.148s
Timpul de prelucrare al imaginii este: 0.052s
Timpul de scriere al imaginii este: 0.473s
```

Pentru referință, imaginea originală BMP este:



Iar aceasta este imaginea prelucrată, căreia i-a fost aplicat efectul de blur:



## Descrierea Arhitecturală a Aplicației

Aplicația implementată în proiect are 4 clase (MainClass, Citire, Scriere, Smoothing), precum și o interfață, numită InterfataDeTimp.

### Descrierea claselor

#### **MainClass**

În această clasă are loc preluarea datelor de input și output de la consolă/tastatură. Tot în MainClass, este instanțiată clasa Scriere și apelate metodele de scriere a imaginii în fișier și de afișare a celor 3 timpi (de citire, scriere și procesare).

În această clasă au fost utilizate argumente de tip `varargs`, în cadrul metodei de afișare și preluare de text în consolă.

### **Citare**

Această clasă este de tip public abstract și implementează interfața `InterfataDeTimp` pentru a putea calcula timpul necesar pentru citirea imaginii. Totodată, această clasă este moștenită de clasa `Scriere`. În cadrul acestei clase, are loc citirea imaginii alese pentru prelucrare, în scopul căreia a fost importată, de asemenea, clasa Java numită `java.awt.image.BufferedImage`, care ajută la prelucrarea fișierelor sub formă de imagini.

În clasa citire, avem constructorul de tip public `BufferedImage getImage()` și următoarele metode: public void `readImageFromFile` (în cadrul căreia se citește imaginea suport cu ajutorul unei metode numite `ImageIO.read` din biblioteca `java.awt.image.BufferedImage` importată mai sus, se tratează excepțiile și se calculează timpul de citire), public long `citireTimp`, precum și 2 clase de tip public abstract long denumite `smoothImageTimp` și `scriereTimp`. Acestor 3 clase din urmă li se aplică `override`.

### **Smoothing**

Această clasă este de tip public abstract și moștenește clasa `Citare`. În cadrul acestei clase are loc prelucrarea efectivă a imaginii alese.

Biblioteca `java.awt.image.BufferedImage` este esențială în cadrul acestei clase, întrucât am utilizat multiple clase ale acesteia: `java.awt.image.BufferedImage`, `java.awt.image.Kernel`, `java.awt.image.BufferedImageOp` și `java.awt.image.ConvolveOp`.

Prin clasa `Kernel` este definită matricea ce va fi utilizată pentru convoluție, prin clasa `BufferedImageOp` sunt descrise operațiile input/output aplicate obiectelor de tip `BufferedImage`, iar prin clasa `ConvolveOp` este efectuată operația de convoluție dintre kernel și imaginea originală.

Constructorul clasei `Smoothing` este public `Smoothing()`, iar metodele clasei `Smoothing` sunt public void `smooth()` (în care sunt utilizate clasele din biblioteca `java.awt.image.BufferedImage` pentru a putea fi modificată imaginea și totodată se calculează și timpul de procesare), public `BufferedImage getImageS()` prin care se returnează imaginea transformată, public long `smoothImageTimp()` căreia îi este aplicat `overriding`, și public long `scriereTimp()`.

### **Scriere**

Această clasă este tip public și moștenește clasa `Smoothing`. Se remarcă că a fost implementată moștenire pe 3 nivele (Clasa `Smoothing` moștenește clasa `Citare` și este moștenită la rândul său de către Clasa `Scriere`). În cadrul clasei `Scriere`, este scrisă imaginea finală procesată la destinația introdusă de la tastatură.

Constructorul clasei `Scriere` este public `Scriere()` iar metodele acesteia sunt: getterul public static `BufferedImage getImageW()` care returnează imaginea prelucrată, public `Scriere` (în care sunt folosite multiple metode moștenite de la clasele `Citare` și `Smoothing`), public void `writImageToFile` (în care se scrie în fișier folosind metoda `FileOutputStream`, sunt tratate excepțiile de tip

FileNotFoundException și IOException, și este calculat timpul de scriere), și public long scriereTimp() căreia îi este aplicat override.

### **Interface-ul InterfataDeTimp**

Această interfață este definită simplu, conținând doar 3 metode pentru calculul timpilor de citire, procesare și scriere, numite citireTimp, smoothImageTimp și scriereTimp. Aceste 3 metode sunt implementate de clasa Citire, apoi sunt moștenite de clasa Smoothing și respectiv de clasa Scriere, și li se aplică overriding.

### **Concluzii privitoare la eficiența prelucrării imaginilor**

Aplicația este, din punctul meu de vedere, destul de eficientă, fapt ce reiese din timpii calculați pentru citirea, scrierea și prelucrarea imaginilor. Aceștia sunt de ordinul jumătăților de secundă, uneori chiar mai puțin. Este posibil ca ierarhizarea pe 3 niveluri de moștenire să încetinească oarecum prelucrarea imaginii, ceea ce înseamnă că aceasta s-ar putea desfășura chiar mai rapid și eficient.

### **Bibliografia proiectului**

1. [https://www.w3schools.com/java/java\\_interface.asp](https://www.w3schools.com/java/java_interface.asp)
2. [https://www.w3schools.com/java/java\\_inheritance.asp](https://www.w3schools.com/java/java_inheritance.asp)
3. <https://docs.oracle.com/javase/7/docs/api/java/awt/image/BufferedImage.html>
4. [https://en.wikipedia.org/wiki/Kernel\\_\(image\\_processing\)](https://en.wikipedia.org/wiki/Kernel_(image_processing))
5. <https://www.tabnine.com/code/java/classes/java.awt.image.Kernel>
6. [https://www.tutorialspoint.com/java\\_dip/java\\_buffered\\_image.htm](https://www.tutorialspoint.com/java_dip/java_buffered_image.htm)
7. <https://docs.oracle.com/javase/7/docs/api/java/awt/image/class-use/BufferedImage.html>