

**UNIVERSITATEA POLITEHNICA DIN BUCUREȘTI**

FACULTATEA DE AUTOMATICĂ ȘI CALCULATOARE

SPECIALIZARE: INGINERIA SISTEMELOR

**Gray Level Histogram of a Gray-Scale Image**

**Student:**

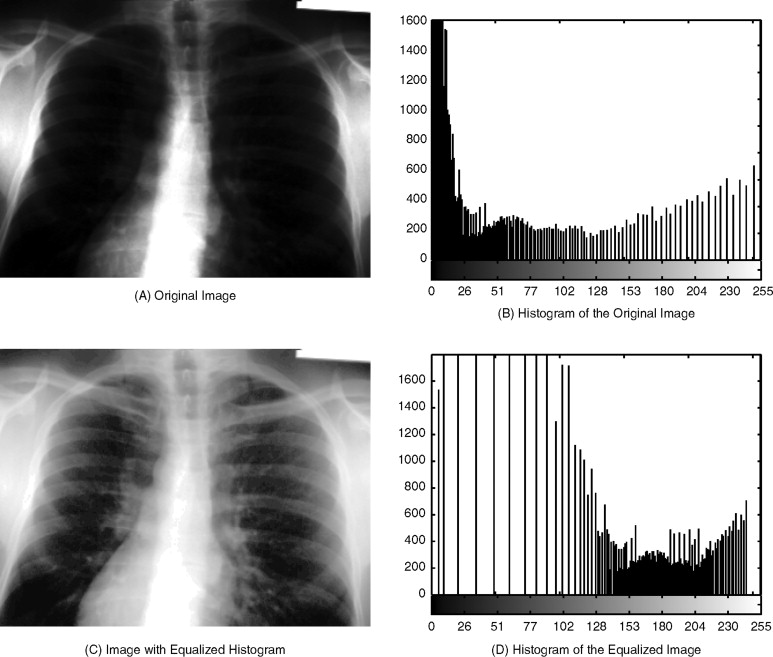
Badea Marilena-Dalia

**Grupa: 331AB**

1. **Introducere**

Tema constă în implementarea algoritmului Gray Level Histogram of a Gray-Scale Image, astfel imaginea rezultată trebuie să conțină 11 niveluri de gri, respectiv toți pixelii care au nivelul de gri +/- 5 față de nivelul de gri al maximului histogramei.

Prin **histogramă** înțelegem un grafic, reprezentând luminozitatea pe axa orizontală de la 0 la 255, pentru o scară de intensitate pe 8 biți, respectiv numărul de pixeli pe axa verticală.



[[1] Sursă imagine](https://medium.com/@rndayala/image-histograms-in-opencv-40ee5969a3b7)

1. **Descrierea aplicației**

Precum este specificat în cerințele de implementare ale proiectului, prima dată este încărcată o imagine RGB în format .bmp (24 bit).

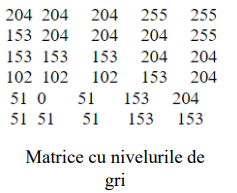
Pentru a aplica algoritmul asupra imaginii, trebuie urmați pașii:

1. O imagine colorată este transpusă într-o matrice cu valorile de gri ale pixelilor din imaginea procesată.
2. Într-un vector se salvează numărul de apariții al nivelului de gri din matrice, iar în funcție de valoarea maximă găsită în acest vector de apariții, se va forma imaginea nouă în format .bmp, cu nivelurile de gri +/- 5 față de valoarea de gri care apare în mod predominant.
3. Într-un fișier .txt este scris numărul de apariții al fiecărui nivel de gri și se reprezintă histograma folosind semnul (-), reprezentarea fiind orientativă, deoarece afișarea numărului total de (-) pentru fiecare apariție a fiecărui nivel de gri ar fi fost ineficientă (o valoare poate apărea de multe ori), așadar am afișat un număr redus al aparițiilor, împărțind numărul acestora la un număr mare (4000).

**Spre exemplu:**

Se obține matricea cu

nivelurile de gri:

****

Imaginea colorată

 Nivelul de gri al fiecărui pixel

 s-a calculat cu formula:

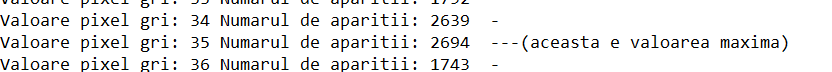


În acest vector salvez numărul de apariții al fiecărui nivel de gri:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **Nivel gri** | 0 | 1 | ... | **35** | ... | 254 | 255 |
| **Nr. apariții** | 120 | 140 | ... | **2694** | ... | 2 | 17 |

Se caută valoarea maximă din vector, iar nivelul de gri care apare în mod predominant în histogramă are valoarea 35, deci imaginea rezultată va conține doar nivelurile cuprinse între 30 (35-5) și 40 (35+5), adică 11 niveluri de gri !

Nivelul predominant de gri în histogramă:





Deci, imaginea rezultată va arăta în felul următor:

1. **Descrierea modului de implementare**

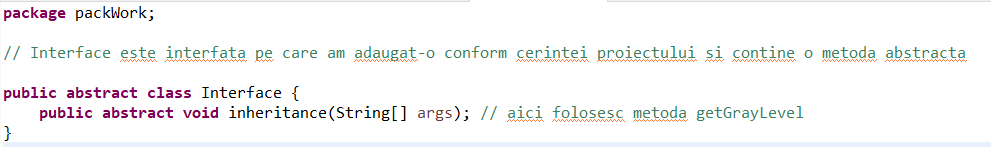
Precum este specificat în cerințele de implementare, am folosit 2 pachete: primul este numit **packTest**, în care este localizată clasa de test, Main.java, iar în al doilea pachet, **packWork**, am implementat restul claselor necesare pentru realizarea cerințelor. Această împărțire a condus la ușurința realizării modificărilor atunci când aveam nevoie, dar și la structurarea mai eficientă a etapelor proiectului.

Clasele folosite în cadrul proiectului sunt:

1. **Main** (aplicația de test) – aici am pus operațiile de deschidere și citire ale imaginii .bmp, am procesat imaginea folosind multi-threading, are loc astfel secționarea imaginii în 4 secvențe. De asemenea, am aplicat algoritmul pe imaginea procesată și am creat noua imagine. Pentru fiecare etapă am monitorizat timpul de execuție și l-am afișat.
2. **Histogram –** aici am realizat algoritmul pentru rezolvarea cerinței proiectului.
3. **ConvertGrayscale -** clasă necesară pentru implementarea algoritmului, convertește pixelii colorați în grayscale.
4. **GrayLevel** – clasă necesară pentru implementarea algoritmului, calculează nivelul de gri pentru fiecare pixel.
5. **Imagine** – clasă folosită pentru încărcarea și crearea noii imagini rezultate în urma aplicării algoritmului din clasa Histogram. **Moștenirea** folosită este evidențiată în această clasă, aceasta fiind pe 4 niveluri:

**Interface GrayLevel ConvertGrayscale Histogram**

1. **Interface** – clasă folosită pentru a crea o interfață și pentru a folosi o metodă abstractă.



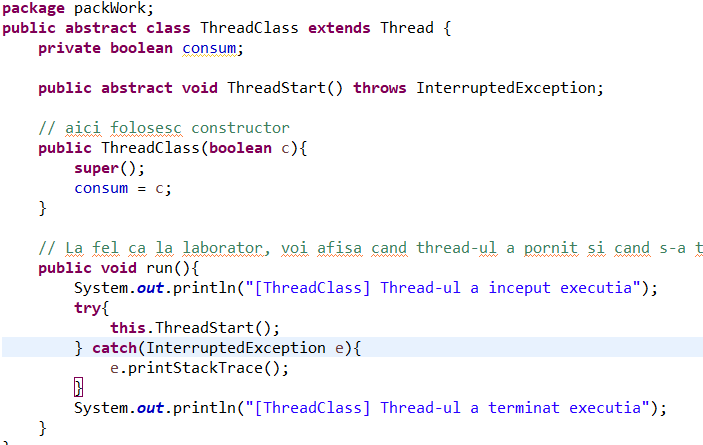
**Multi-threading-ul** a fost folosit pentru secționarea imaginii în 4 părți egale:

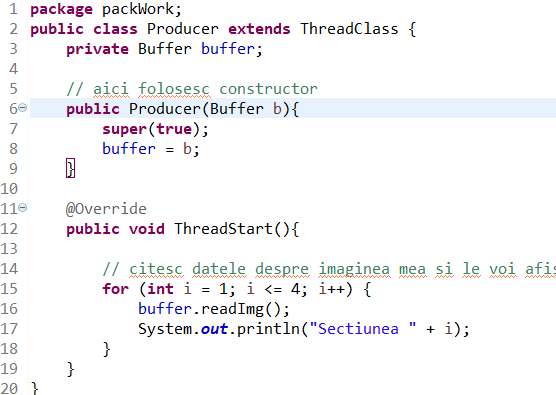
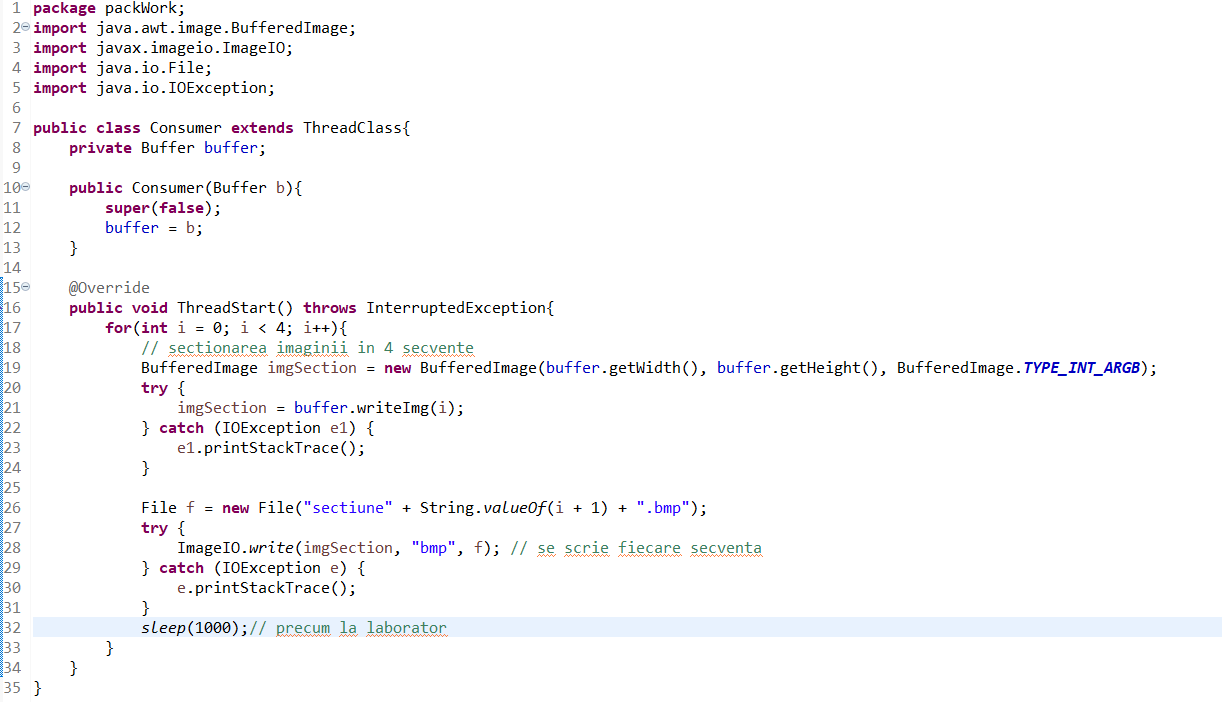


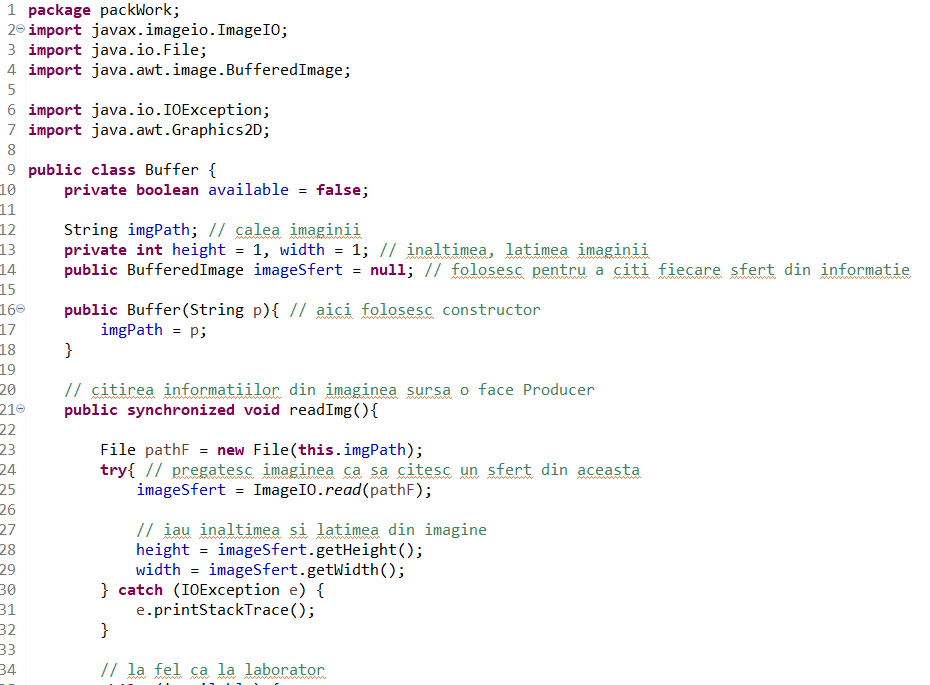


În acest scop, m-am folosit de exemplele lucrate la laboratoarele 5 și 6, dar și de cursul de AWJ, pentru a crea următoarele clase:

* **ThreadClass**

****

* **Producer**
* **Consumer**
* **Buffer**

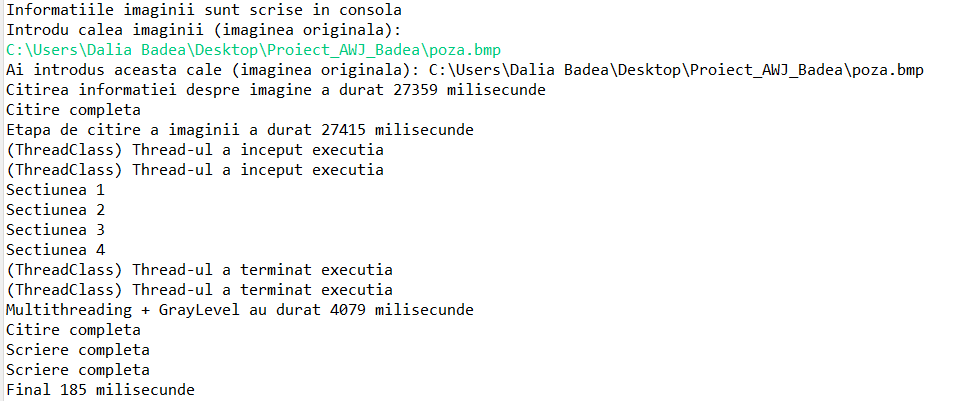
****

1. **Descrierea rezultatelor obținute**

Voi evidenția procesul pentru 2 imagini, pentru a arăta rezultatele finale:

1. **Prima imagine testată**
2. Se încarcă imaginea de prelucrat

1. Afișarea performanțelor



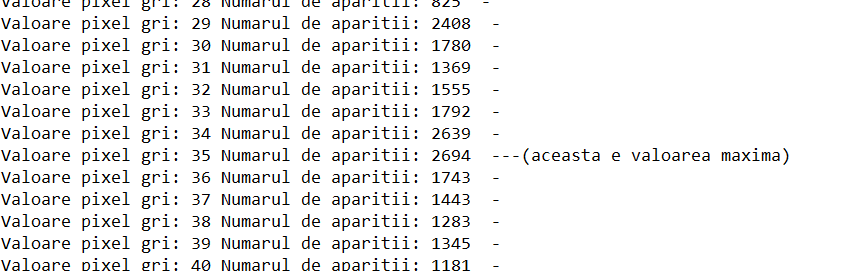
1. Se împarte imaginea în 4 părți egale



1. Imaginea se transformă din RGB în grayscale



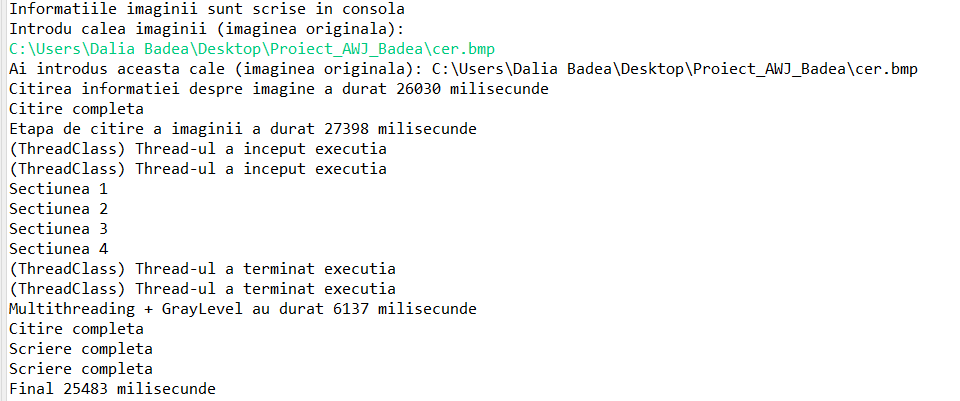
1. Rezultatele histogramei sunt afișate într-un fișier .txt



1. Imaginea obținută în urma aplicării tuturor transformărilor cu ajutorul algoritmului

1. **A doua imagine testată**
2. Se încarcă imaginea de prelucrat

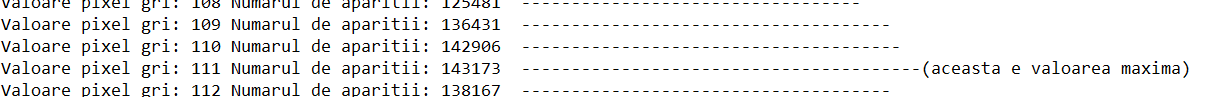


1. Afișarea performanțelor
2. Se împarte imaginea în 4 părți egale

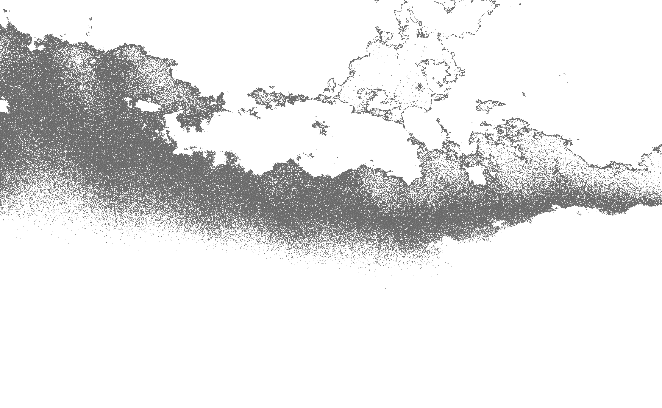




1. Imaginea se transformă din RGB în grayscale
2. Rezultatele histogramei sunt afișate într-un fișier .txt

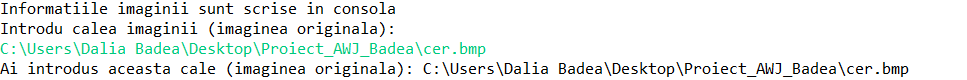


1. Imaginea obținută în urma aplicării tuturor transformărilor cu ajutorul algoritmului



1. **Testarea funcționalității aplicației**

Utilizatorul intoduce în consolă calea către imaginea pe care vrea s-o proceseze:



1. **Concluzii**

În urma realizării proiectului, folosind moșteniri și thread-uri, mi-am perfecționat cunoștințele referitor la aceste noțiuni din Java, pentru că a trebuit să înțeleg cum funcționează fiecare pas de execuție.

Transformarea unei imagini din color în grayscale și lucrul cu histograma acesteia presupun un exercițiu benefic pentru experiența personală.

1. **Bibliografie**
2. Curs AWJ de pe Moodle + arhivele de laborator
3. [Java BufferedImage getting red, green and blue individually - Stack Overflow](https://stackoverflow.com/questions/2615522/java-bufferedimage-getting-red-green-and-blue-individually)
4. [Software: Grey level / Grey value | STEMMER IMAGING (stemmer-imaging.com)](https://www.stemmer-imaging.com/en/knowledge-base/grey-level-grey-value/)
5. <https://studylib.net/doc/7883468/grey_level_enhancement>