Convert Gray-Scale Image to Binary-Image (Static Threshold)

Facultatea de Automatica si Calculatoare, UPB

Cercelaru Alexandra

331 AB

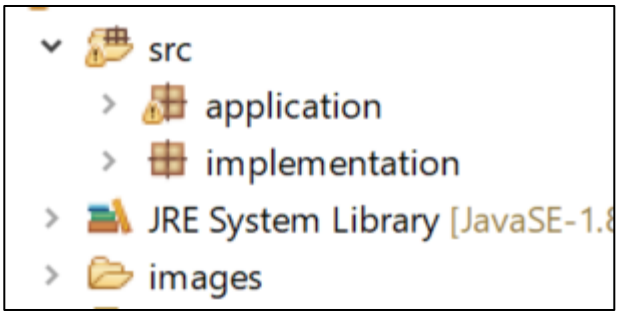
Cercelaru Alexandra, 331 AB

1. Scop

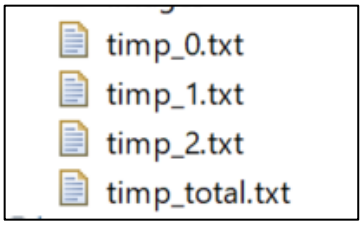
Proietul are ca scop intarirea cunostiintelor limbajului Java si a conceptelor OOP prin citirea unei imagini gray-scale dintr-un fisier sursa, convertirea acesteia intr-o imagine binara, si scrierea inapoi in fisier a imaginii rezultate.

2. Continutul aplicatiei

Aplicatia este construita pe baza a 2 foldere importante. Folderul **src** (continut de fiecare aplicatie java) si folderul **images**, unde se gasesc pozele ce vor fi prelucrate in cadrul aplicatiei si pozele finale rezultate in urma prelucrarii.



Folderul src se imparte apoi in 2 pachete: pachetul ‘application’ si pachetul ‘implementation’. Primul continue aplicatia in sine, adica clasa Main, in cadrul careia programul meu devine functional. Al doilea pachet contine 12 clase ce fac posibila implementarea functionalitatilor ce alcatuiesc acest program.



Programul mai tine si cele 4 fisiere .txt ce contin timpii rezultati din aplicatie.

3. Descrierea claselor

- Buffer – retine dimensiunea pozei si face comunicatia intre cele 2 thread-uri cu ajutorul functiilor putBytes si getBytes ce citesc bitii din imagine si ii transmit mai departe spre a fi prelucrati.
- Producer – Extinde clasa Thread si contine thread-ul care citeste imaginea cate ¼ la un moment de timp dat.
- ImageReader – Extinde clasa ReadingTimeCounter, porneste si opreste contorii pentru timpul de citire, si citeste imaginea cu ajutorul functiei putBytes din clasa Buffer.

- Consumer – Extinde clasa Thread si continue thread-ul care consuma ¼ transmis de Producer din imagine. Dupa ce imaginea este transmisa in intregime, este memorata intr-o variabila de tip BufferedImage si este trimisa mai departe spre prelucrare si apoi spre scrierea in fisier.
- ImageWriter – Extinde WritingTimeCounter, porneste si opreste contorii pentru timpul de scriere, si scrie imaginea in fisierul de iesire.
- GrayScaleToBinaryConverter – Extinde ProcessingTimeCounter, porneste si opreste contorul pentru timpul de prelucrare si implementeaza algoritmul de convertire a imaginii din gray-scale image in binary image.
- ExecutionTimeCounter – Interfata proiectului care implementeaza 3 metode de reset, start si stop pentru timpii folositi in program.
- AbstractExecutionTimeCounter – Clasa abstracta ce implementeaza interfata si inca 2 metode abstracte care scriu timpii in consola, respectiv in fisierele .txt
- ReadingTimeCounter – Extinde clasa abstracta si implementeaza 2 metode de scriere a timpului de citire in consola si in fisier.
- ProcessingTimeCounter – Extinde clasa abstracta si implementeaza 2 metode de scriere a timpului de procesare in consola si in fisier.
- WritingTimeCounter – Extinde clasa abstracta si implemeteaza 2 metode de scriere a timpului de scriere in consola si in fisier.
- TimeManager – Foloseste varargs si scrie timpii din program in fisierele txt
- Main – Deschide fisierul, listeaza imaginile disponibile pentru prelucrare in consola, pornste cele 2 thread-uri si timpii de management.

4. Descrierea functionalitatii programului

Programul incepe din clasa Main prin crearea celor 4 obiecte de care are nevoie: TimeManager, ImageReader, ImageWriter si GrayScaleToBinaryConverter. Ia sursa fisierului ce contine imaginile ce urmeaza a fi prelucrate (

```
File currentDir = new File("./images");
```

```
Bună ziua!
Tastați 'list' pentru a vedea imaginile disponibile.
Tastați ENTER pentru a continua.
list
floare.bmp
fluture.bmp
veverita.bmp
Tastați numele imaginii cu extensia .bmp!!
Tastați 'exit' pentru a închide aplicația.
```

si apoi incepe comunicarea cu utilizatorul prin intermediul consolei. Se afiseaza o lista cu imaginile si introducem numele uneia dintre ele. Sa nu uitam sa punem si extensia **.bmp**. Putem prelucra o imagine cat timp nu tastam cuvantul ‘exit’.

Dupa ce numele pozei este citit se creaza obiectul de tip Buffer cu parametru lungimea pozei, obiectul de tip Producer si Consumer cu parametrii imaginea curenta si obiectele create la inceputul programului, dupa care sunt pornite cele 2 thread-uri.
Primul thread este cel din clasa Producer care citeste pe rand cate ¼ din informatie prin intermediul functiei readQuarter() din clasa ImageReader, apoi cu functia putBytes din clasa Buffer pune in vectorul ‘data’ informatia respectiva, dupa care notifica celalalt thread din clasa Consumer, si apoi asteapta 1 secunda.
Al doilea thread, cel din clasa Consumer, este pornit atunci cand primeste notificare ca cel dinaintea lui asteapta. Atunci cand producer asteapta 1 secunda, consumer consuma din imagine acel sfert. Adica prin intermediul functiei getBytes din Buffer primeste respectivul ¼ din informatie. Notifica apoi ca a terminat si asteapta si el o secunda.
Acest proces se repeta pana cand cele 2 thread-uri au terminat de citit imaginea si scris imaginea.

```
Producer started execution
Consumer started execution
Se produce sfertul 1 din imagine
Se consuma sfertul 1 din imagine
Se produce sfertul 2 din imagine
Se consuma sfertul 2 din imagine
Se produce sfertul 3 din imagine
Se consuma sfertul 3 din imagine
Se produce sfertul 4 din imagine
Se consuma sfertul 4 din imagine
Producer ended execution
Consumer ended execution
```

Dupa care imaginea se pune intr-un obiect de timp BufferedImage si se apelaza metoda process() din clasa GrayScaleToBinaryConverter pentru algoritmul propriu zis. Se porneste contorul pentru timpul de procesare a imaginii si se seteaza threshold-ul la valoarea 125, fiind jumatatea unde impartim nuantele de gri in alb si negru. Se parcurge matricea imaginii, se selecteaza componenta rosie din imagine si se compara cu threshold-ul stabilit de noi. Daca este mai mare, setam noul

pixel pe 255, adica alb, iar in caz contrar pe 0, adica negru. Dupa care setam toate cele 3 valori ale pixelului cu cea a threshold-ului cu ajutorul fucntiei colorToRGB si apoi punem noul pixel in imaginea noastra pe aceeasi pozitie i si j. Dupa care la final returnam imaginea procesata si o scriem in fisier cu extenia ‘final_’ in fata pentru a stii care este cea noua.

```
try {
    BufferedImage inputImage = ImageIO.read(new ByteArrayInputStream(data));
    BufferedImage outputImage = grayScaleToBinaryConverter.process(inputImage);
    imageWriter.writeImage(outputImage, "final_" + file.getName());
} catch (IOException e) {
    e.printStackTrace();
}
```

Apoi se apeleaza cele 2 functii manageTime si manageTotalTime care scriu timpii din program in consola si in fisierele .txt. Functia manageTime primeste un varargs si in functie de argumentul de la contorul i scrie in fisier/consola timpul de citire/procesare/scriere.

```
Timpul de citire este:
3.055 secunde
Execution time successfully persisted to file
Timpul de procesare este:
0.094 secunde
Execution time successfully persisted to file
Timpul de scriere este:
0.048 secunde
Execution time successfully persisted to file
Timpul total este:
3.1970003 secunde
Total execution time successfully persisted to file
```

Dupa care programul se incheie.

```
Tastați numele imaginii cu extensia .bmp!!
Tastați 'exit' pentru a închide aplicația.
exit
O zi bună!
```

5. Cerintele de implementare

Include cele 4 concepte OOP: Abstractizare (prin clasa Abstracta), Incapsularea (numerosi parametrii de tip private si protected), Polimorfismul (in metoda manageTime care asteapta ca parametru un vector de tip AbstractExecutionTimeCounter si primeste copii ai clasei respective si anume imageReader, grayScaleToBinaryConverter si imageWriter) si Mostenirea cu 3 niveluri de ierarhie (GrayScaleToBinary care extinde ProcessingTimeCounter care extinde AbstractExecutionTimeCounter).

Codoul sursa este comentat si respecta operatii de lucru cu fisiere. Este o aplicatie multimodulara si primeste operatii de intrare de la tastatura.

```
// citirea componentei RED din pixelul curent
pixelComponentRed = new Color(inputImage.getRGB(i, j)).getRed();
```

Include constructori si varargs.

```
public void manageTime(AbstractExecutionTimeCounter... counters) {
```

Include tratarea exceptiilor in bloc try-catch.

```
try {
    thread.join();
} catch (InterruptedException e) {
    e.printStackTrace();
    Thread.currentThread().interrupt();
}
```

Cercelaru Alexandra, 331 AB

6. Modul de lucru

Mai jos voi atasa 2 imagini si consola pentru a vedea cum functioneaza intregul program.

Before

After

```
Bună ziua!
Tastați 'list' pentru a vedea imaginile disponibile.
Tastați ENTER pentru a continua.
list
floare.bmp
fluture.bmp
veverita.bmp
Tastați numele imaginii cu extensia .bmp!!
Tastați 'exit' pentru a închide aplicația.
floare.bmp
Producer started execution
Consumer started execution
Se produce sfertul 1 din imagine
Se consuma sfertul 1 din imagine
Se produce sfertul 2 din imagine
Se consuma sfertul 2 din imagine
Se produce sfertul 3 din imagine
Se consuma sfertul 3 din imagine
Se produce sfertul 4 din imagine
Se consuma sfertul 4 din imagine
Producer ended execution
Consumer ended execution
Timpul de citire este:
3.055 secunde
Execution time successfully persisted to file
Timpul de procesare este:
0.094 secunde
Execution time successfully persisted to file
```

```
Timpul de scriere este:
0.048 secunde
Execution time successfully persisted to file
Timpul total este:
3.1970003 secunde
Total execution time successfully persisted to file
Tastați numele imaginii cu extensia .bmp!!
Tastați 'exit' pentru a închide aplicația.
exit
O zi bună!
```

7. Surse de inspiratie

https://onlineimagetools.com/grayscale-image

https://www.researchgate.net/post/How_to_convert_grayscale_image_to_binary_image_in_java

https://stackoverflow.com/questions/11227994/grayscale-to-binary

https://docs.oracle.com/javase/7/docs/api/java/awt/image/BufferedImage.html