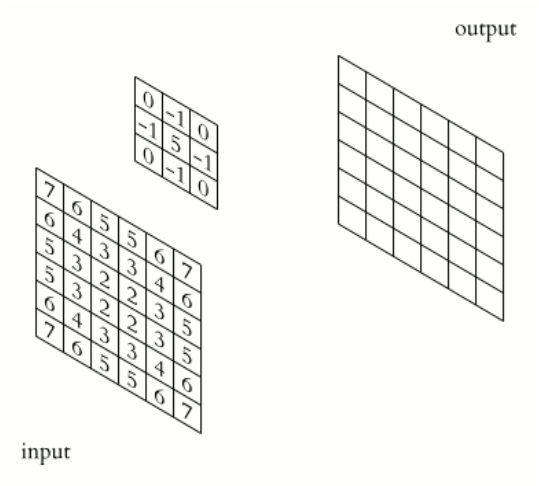
Laborator 1

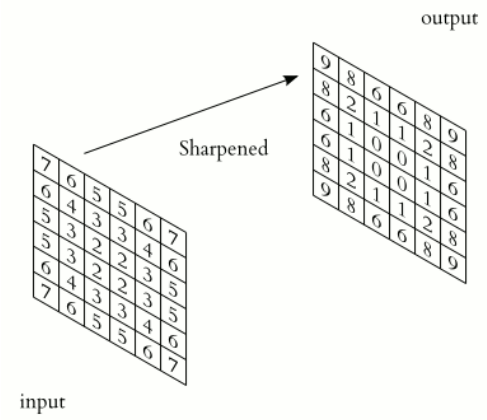
1. Cerinta

Se considera o imagine reprezentata printr-o matrice de pixeli, F , de dimensiune (NxM). Se cere transformarea ei aplicand o filtrare cu o fereastra definita de multimea de indici W cu coeficientii *wkl (*reprezentati prin matricea W[k,l], unde 0<=k<n, 0<=l<m; si n<N, m<M).

Postconditii :

- Matricea rezultat V contine imaginea filtrata a imaginii initiale F (V<>F)
- program secvential + paralel cu nr customizabil de threaduri (in Java & C++)
- folosirea directa a threadurilor





2. Arhitectura aplicatiei

A) JAVA
In varianta secventiala, clasa in care se “petrec” procesarile este Matrix, cu urmatoarele campuri : width (int), height (int), matrix (int[][]) ce descriu dimensiunile respectiv continutul matricei input. Clasa are 2 constructori,

- `Matrix(int width, int height)`
- `Matrix(int width, int height, int[] numbers)`

o metoda principala ce face procesarea matricei :

- `public int[][] convolution(Matrix kernel)`

si 10 metode de care se foloseste cea principala :

- `private void borderRightAndDown(Matrix someMatrix)`
- `private void borderDown(Matrix someMatrix, int someMatrixColumn)`
- `private void borderLeftAndDown(Matrix someMatrix)`
- `public void borderLeftAndUp(Matrix someMatrix)`

- `public void borderUp(Matrix someMatrix, int someMatrixColumn)`
- `public void borderRightAndUp(Matrix someMatrix)`
- `public void borderLeft(Matrix someMatrix, int someMatrixLine)`
- `public void borderRight(Matrix someMatrix, int someMatrixLine)`
- `private int process(Matrix kernel)`
- `private void copy(Matrix someMatrix, int i, int j)`

Metoda convolution primeste ca parametru matricea kernel cu care se doreste sa se faca procesarea inputului. Se considera o submatrice a inputului de aceeasi dimensiune cu kernelul si se calculeaza numarul de “shiftari” orizontale si verticale pe care le face submatricea peste matricea input (practic numarul de shiftari orizontale si numarul de shiftari verticale vor fi latimea respectiv inaltimea matricei output).

In 2 foruri imbricate de forma :

```
for (int i = 0; i < nrVerticalShifts; i++)
    for (int j = 0; j < nrHorizontalShifts; j++)
```

sunt scrise niste conditii legate de “pozitia” submatricei in input pentru a determina ce metoda de bordare se aplica (de exemplu, cand i=0 si j=0, submatricea se alfa chiar in prima pozitie respectiv coltul dis stanga sus al inputului => se va aplica o bordare de tipul borderLeftAndUp).In cazul in care submatricea se afla complet in input aceasta nu se va borda, ci se va folosi metoda copy, care copiaza valorile respective din input in submatrice).

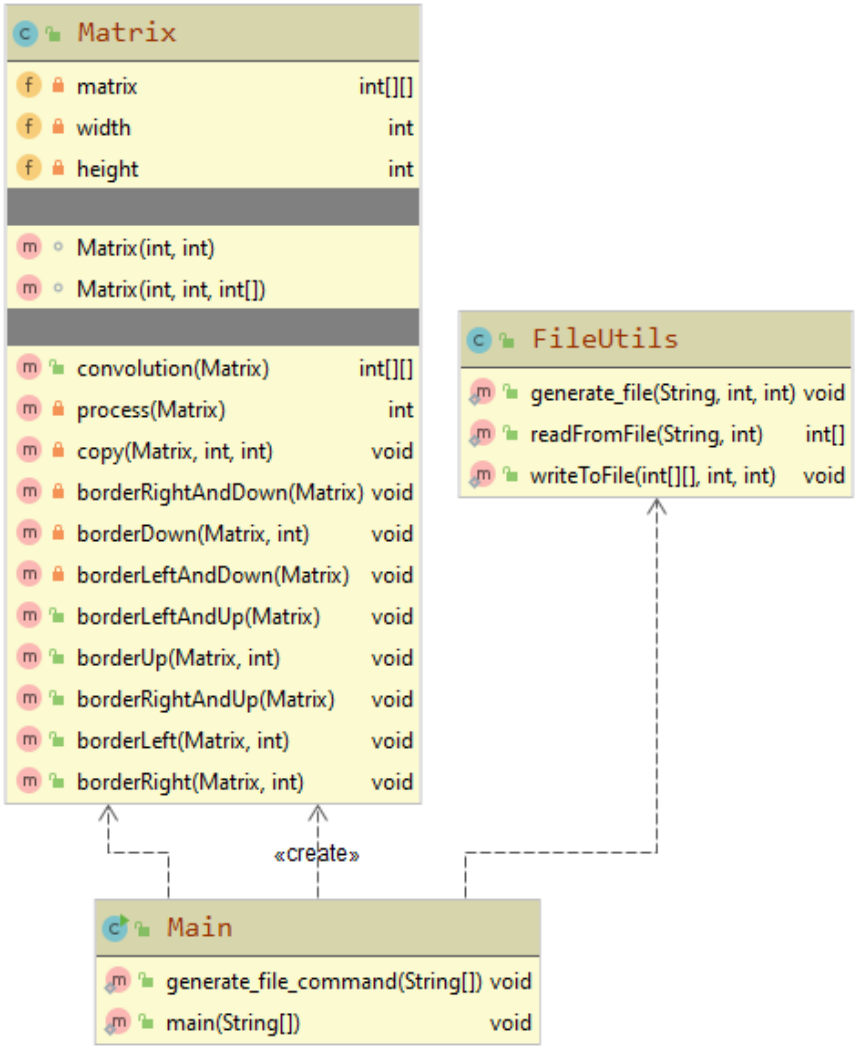
Metoda process va fi apelata din submatrice; primeste ca parametru matricea kernel si face suma produselor dintre submatrice.matrice[i][j] si kernel[i][j], 0 < i < kernel.height, 0 < j < kernel.height

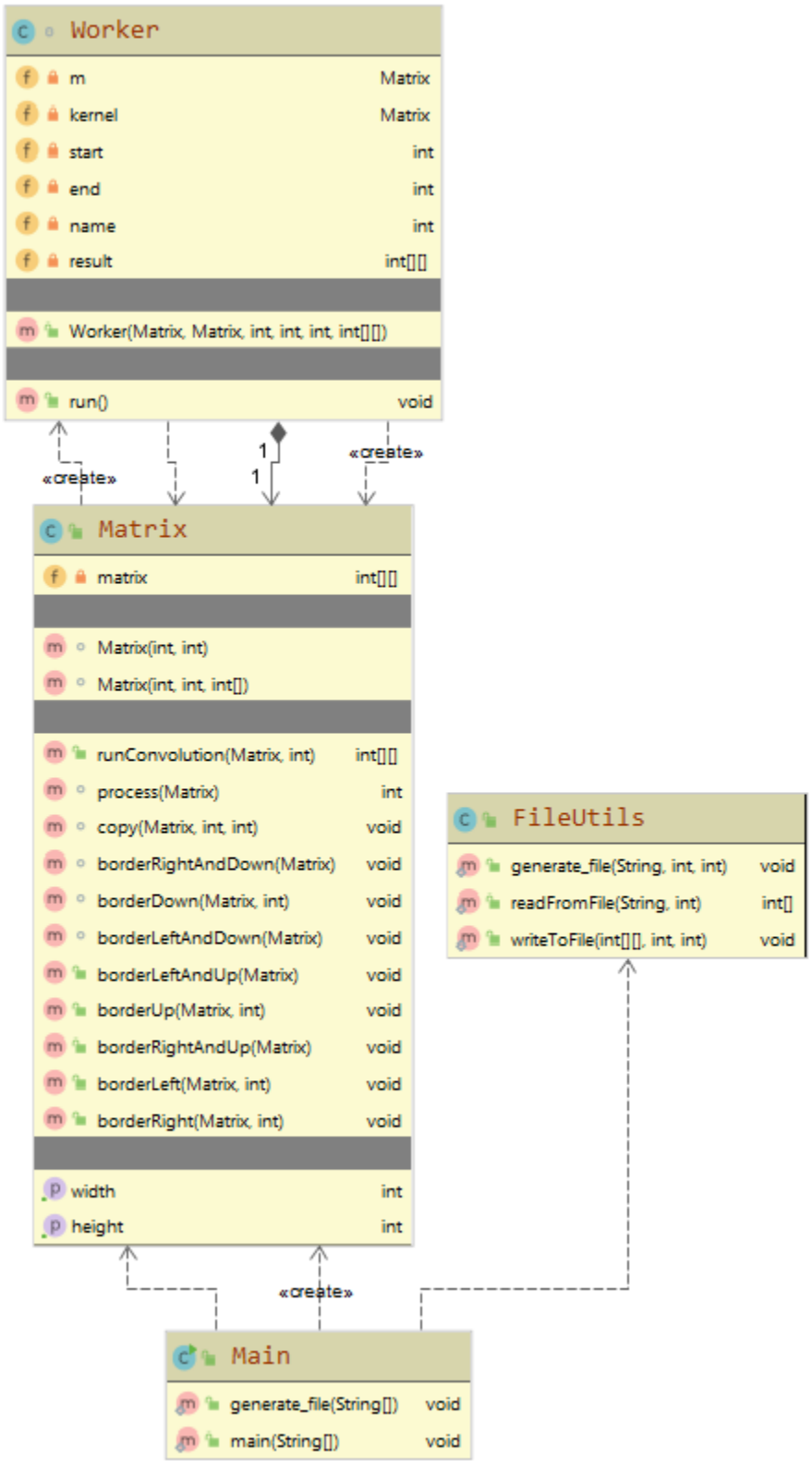
Clasa principala Main se foloseste de metodele statice din clasa FileUtils pentru a genera in prealabil un fisier cu dimensiunile specificate (daca args[1] este termenul “generate”, se genereaza fisierul de input, iar procesarea efectiva necesita o alta comanda – asta pentru a avea acelasi fisier pe tot parcursul unei testari cu mai multe rulari).

In varianta paralela, clasa Matrix contine o metoda runConvolution, ce primeste ca parametri matricea kernel, numarul de threaduri si se ocupa cu crearea de threaduri si alocarea chunk-ului corespunzator fiecarui thread. Se apeleaza pentru fiecare thread impicit metoda run, ce se foloseste de start si end pentru a identifica pozitia curenta a submatricei in matricea input pentru a apela metodele de bordare corespunzatoare.

B) C++

Arhitectura aplicatiei este asemanatoare – clasa Matrix se pastreaza insa tot ce tine de main si generare/scriere/citire de fisiere prezinta un design functional, renuntandu-se la clasele omoloage din Java




Worker
m Matrix
kernel Matrix
start int
end int
name int
result int[][]
Worker(Matrix, Matrix, int, int, int, int[][])
run() void
«create»
1
1
«create»
Matrix
matrix int[][]
Matrix(int, int)
Matrix(int, int, int[])
runConvolution(Matrix, int) int[][]
process(Matrix) int
copy(Matrix, int, int) void
borderRightAndDown(Matrix) void
borderDown(Matrix, int) void
borderLeftAndDown(Matrix) void
borderLeftAndUp(Matrix) void
borderUp(Matrix, int) void
borderRightAndUp(Matrix) void
borderLeft(Matrix, int) void
borderRight(Matrix, int) void
width int
height int
FileUtils
generate_file(String, int, int) void
readFromFile(String, int) int[]
writeToFile(int[][], int, int) void
«create»
Main
generate_file(String[]) void
main(String[]) void

Tip matrice	Nr threads	Timp executie
N=M=10 n=m=3	secvential	0.1603
	4	1.15088
N=M=1000 n=m=5	secvential	193.69906
	2	108.1244
	4	109.03694
	8	190.10598
	16	233.9412
N=10 M=10000 n=m=5	secvential	39.65022
	2	30.952
	4	24.45958
	8	28.1355
	16	46.3285
N=10000 M=10 n=m=5	secvential	32.90262
	2	43.04856
	4	32.67462
	8	40.7904
	16	51.86836

Avand in vedere ca submatricea are ca dimensiuni :

```
nrHorizontalShifts = input.width - (kernel.width - 1) + 1 + 1
nrVerticalShifts = input.height - (kernel.height - 1) + 1 + 1
```

se observa ca cea mai “laborioasa” procesare are loc pentru matricea de dimensiuni 1000 x 1000, iar diferenta intre timpul necesar procesarii paralele si celei secventiale este semnificativa in acest caz. Pentru p = 2 si p = 4 aproape ca se dubleaza viteza de procesara fata de omologul secvential, insa pentru p = 8 diferenta de viteza este neglijabila (in raport cu totalul), iar pentru p = 16 varianta paralela este inferioara ca viteza.

Este notabil, de asemenea, faptul ca pentru matrici de dimensiuni reduse (10 linii si 10 coloane de ex.), varianta secventiala este cu mult superioara celeilalte

Pentru matrici de dimensiuni mari (insa care nu necesita atata de multe procesari precum cea de 1000 x 1000) cel mai bine s-au comportat rularile pe 2 respectiv pe 4 threaduri.