Project 3 (Java) : Implementation of the four basic Morphology Operations as taught in class. (updated version.)

What you have to do for this project:

1) You will be given:
   - two data files: data1 and data2;
   - two structuring elements: Elem1 and Elem2;
   - An answer file: testAnswer, for you to verify the correctness of the 4 basic operations on data1.
2) Implement your program according to the specs below, until the program passes the compilation.
3) Run and debug your program using data1 with Elem1 until the results of your four basic morphological operation (in outFile1) are the same as shown in testAnswer.
4) Run1: Run your program using data1 with Elem1.
5) Run2: Run your program using data2 with Elem2.
6) Your hard copies include:
   - cover sheet // Don't forget to include algorithm steps of main()
   - source code
   - outFile1 and outFile2 from Run1 (step 4 in the above.)
   - outFile1 and outFile2 from Run2 (step 5 in the above.)

**********

Language: Java
Project points: 10 pts
Due Date: Soft copy (*.zip) and hard copies (*.pdf):

10/10 (on time): 2/27/2023 Monday before midnight.
-10/10 (non-submission): 2/27/2023 Monday after midnight.
-5/10: Program does not pass compilation.
0/10: Program has runtime error or does not produce any output.
0/10: Did not submit pdf file.

*** Name your soft copy and hard copy files using the naming convention as given in the project submission requirement.
*** All on-line submission MUST include Soft copy (*.zip) and hard copy (*.pdf) in **the same email attachments** with correct email subject as stated in the email requirement; otherwise, your submission will be rejected.
*** Follow "Project Submission Requirement" to submit your project.

**********

I. Inputs: (-2 pts for hardcode your file name.)
**********

a) imgFile (args[0]): a binary image with header. See data1 and data2.
b) elmFile (args[1]): the structuring element with header and the origin information. The format of the structuring element is as follows: 1st text line is the header; the 2nd text line is the origin (row and col position) of the structuring element then follows by the rows and column of the structuring element. See Elm1 and Elem2.

** Note: when a structure element contains zeros, only those 1's in the structuring element take effect in operation!

**********

II. Outputs: There are two output files. (-2 pts for hardcode your file name.)
**********

a) outFile1: (args[2]): for basic operation output.
b) outFile2: (args[3]): for complex operation output.

**********

III. Data structure:
**********

- a Morphology class
  - (int) numImgRows
  - (int) numImgCols
  - (int) imgMin
  - (int) imgMax
  - (int) numStructRows

- (int) numStructCols
- (int) structMin
- (int) structMax
- (int) rowOrigin
- (int) colOrigin
- (int) rowFrameSize // set to (numStructRows / 2), integer division, i.e., 3/2 is 1; 4/2 is 2; 5/2 is 2.
- (int) colFrameSize // set to (numStructCols / 2).
- (int) extraRows // set to (rowFrameSize * 2)
- (int) extraCols // set to (colFrameSize * 2)
- (int) rowSize // set to (numImgRows + extraRows)
- (int) colSize // set to (numImgCols + extraCols
- (int) zeroFramedAry[][] // a dynamically allocate 2D array, size of rowSize by colSize, initialize to zero.
- (int) morphAry[][] // Same size as zeroFramedAry, initialize to zero.
- (int) tempAry[][] // Same size as zeroFramedAry, initialize to zero.
// tempAry is used to store the intermediate result in opening and closing operations or in closing and opening.
- (int) structAry [][]//a dynamically allocate 2D array of size numStructRows by numStructCols, for structuring element.

Methods:

- zero2DAry (Ary, nRows, nCols) // Set the entire Ary (nRows by nCols) to zero.
- loadImg (…) // load imgFile to zeroFramedAry inside of frame, begins at (rowOrigin, colOrigin). On your own!
- loadstruct (…) // load structFile to structAry. On your own!
- ComputeDilation (…) // Apply dilation to every pixel in zeroFramedAry. // see algorithm below.
- ComputeErosion (inAry, outAry) // Apply erosion to every pixel in zeroFramedAry. // see algorithm below.
- ComputeOpening (…) // apply erosion follow by dilation, see algorithm below.
- ComputeClosing (…) // apply dilation follow by erosion, see algorithm below.
- onePixelDilation (i, j, inAry, outAry, structAry)
// Perform dilation on pixel (i, j) with structAry. See algorithm below.
- onePixelErosion (i, j, inAry, outAry, structAry)
// Perform erosion on pixel (i, j) with structAry. See algorithm below.
- imgReformat (...) // re-use code from your previous project or see project 2 specs.
- prettyPrint (Ary, outFile) // outFile can be outFile1 or outFile2.
// Remark: use "Courier new" font and small font size to fit in a page.
// if Ary [i, j] == 0 output ". " // a period follows by a blank
// else output "1 " // 1 follows by one blank

**********
IV. Main(...)
**********

step 0: imgFile, elmFile, outFile1, outFile2 ← open
step 1: numImgRows, numImgCols, imgMin, imgMax ← read from imgFile
numStructRows, numStructCols, structMin, structMax ← read from elmFile
rowOrigin, colOrigin ← read from elmFile
step 2: zeroFramedAry, structAry, morphAry, tempAry ← dynamically allocate. // see description in the above
step 3: zero2DAry(zeroFramedAry, rowSize, colSize)
step 4: loadImg (imgFile, zeroFramedAry)
step 5: imgReformat (zeroFramedAry, outFile1) // with caption.
prettyPrint (zeroFramedAry, outFile1) // with caption.
step 6: zero2DAry(structAry, numStructRows, numStructCols)
loadstruct (structFile, structAry)
prettyPrint (structAry, outFile1) // with caption.
step 7: basicOperations (zeroFramedAry, morphAry, structAry, tempAry, outFile1)
step 8: complexOperations (zeroFramedAry, morphAry, structAry, tempAry, outFile2)
step 9: close all files.

**********
V. basicOperations (zeroFramedAry, morphAry, structAry, tempAry, outFile1)
**********
Step 0: outFile1 ← "entering basicOperations method"
Step 1: zero2DAry(morphAry, rowSize, colSize)
    ComputeDilation (zeroFramedAry, morphAry, structAry)
    prettyPrint (morphAry, outFile1) // Prior to prettyPrint, write "Printing result of ComputeDilation."
step 2: zero2DAry(morphAry, rowSize, colSize)
    ComputeErosion (zeroFramedAry, morphAry, structAry)
    prettyPrint (morphAry, outFile1) // Prior to prettyPrint, write "Printing result of ComputeErosion".
step 3: zero2DAry(morphAry, rowSize, colSize)
    ComputeOpening (zeroFramedAry, morphAry, structAry, tempAry)
    prettyPrint (morphAry, outFile1) // Prior to prettyPrint, write "Printing result of ComputeOpening"
step 4: zero2DAry(morphAry, rowSize, colSize)
    ComputeClosing (zeroFramedAry, morphAry, structAry, tempAry)
    prettyPrint (morphAry, outFile1) // Prior to prettyPrint, write "Printing result of ComputeClosing.
Step 5: outFile1 ← "exit basicOperations method"

**********
VI. complexOperations (zeroFramedAry, morphAry, structAry, tempAry, outFile2)
**********
step 0: outFile2 ← "entering complexOperations method"
step 1: zero2DAry(morphAry, rowSize, colSize)
    ComputeOpening (zeroFramedAry, morphAry, structAry, tempAry)
    prettyPrint (morphAry, outFile2) // Prior to prettyPrint, write " Pretty print the result of Opening.
    copyArys (morphAry, zeroFramedAry) // copy morphAry to zeroFramedAry
step 2: zero2DAry(morphAry, rowSize, colSize)
    ComputeClosing (zeroFramedAry, morphAry, structAry, tempAry)
    prettyPrint (morphAry, outFile2) // Prior to prettyPrint, write " Pretty print the result of Opening follow by Closing.
    copyArys (morphAry, zeroFramedAry) // copy morphAry to zeroFramedAry
step 3: zero2DAry(morphAry, rowSize, colSize)
    ComputeClosing (zeroFramedAry, morphAry, structAry, tempAry)
    prettyPrint (morphAry, outFile2) // Prior to prettyPrint, write " Pretty print the result of Closing.
    copyArys (morphAry, zeroFramedAry) // copy morphAry to zeroFramedAry
step 4: zero2DAry(morphAry, rowSize, colSize)
    ComputeOpening (zeroFramedAry, morphAry, structAry, tempAry)
    prettyPrint (morphAry, outFile2) // Prior to prettyPrint, write " Pretty print the result of Closing follow by Opening.
step 9: outFile2 ← "Exit complexOperations method"

**********
VII. ComputeDilation (inAry, outAry, structAry)
**********
step 1: i ← rowFrameSize
step 2: j ← colFrameSize
step 3: if inAry [i,j] > 0
    onePixelDilation (i, j, inAry, outAry, structAry) // only processing one pixel inAry[i,j]
step 4: j++
step 5: repeat step 3 to step 4 while j < (colSize)
step 6: i++
step 7: repeat step 2 to step 6 while i < (rowSize)

```
*******************************
VIII. ComputeErosion (inAry, outAry, structAry) // process dilation on each pixel in the entire zeroFramedAry
*******************************
step 1: i ← rowFrameSize
step 2: j ← colFrameSize
step 3: if inAry[i,j] > 0
            onePixelErosion (i, j, inAry, outAry, structAry) // only processing one pixel inAry[i,j]
step 4: j++
step 5: repeat step 3 to step 4 while j < (colSize)
step 6: i++
step 7: repeat step 2 to step 6 while i < (rowSize)

*******************************
IX. onePixelDilation (i, j,  inAry,  outAry, structAry)
*******************************
step 0 : iOffset ← i - rowOrigin
            jOffset ← j - colOrigin
              // translation of image's coordinate (i, j) with respected to the origin of the structuring element
step 1: rIndex ← 0
step 2: cIndex ← 0
step 3: if (structAry [rIndex][cIndex] > 0)
            outAry [iOffset + rIndex][jOffset + cIndex] ← 1
step 4: cIndex ++
step 5: repeat step 3 to step 4 while cIndex < numStructCols
step 6: rIndex ++
step 7: repeat step 2 to step 6 while rIndex < numStructRows

*******************************
X. onePixelErosion (i, j, inAry, outAry, structAry)
*******************************
step 0 : iOffset ← i - rowOrigin
            jOffset ← j - colOrigin
              // translation of image's coordinate (i, j) with respected of the origin of the structuring element
            matchFlag ← true
step 1: rIndex ← 0
step 2: cIndex ← 0
step 3: if (structAry[rIndex][cIndex] > 0) and (inAry[iOffset + rIndex][jOffset + cIndex] ) <= 0)
              matchFlag ← false
step 4: cIndex ++
step 5: repeat step 3 to step 4 while (matchFlag == true) and (cIndex <  numStructCols )
step 6: rIndex ++
step 7: repeat step 2 to step 6 while (matchFlag == true) and (rIndex < numStructRows)
step 8: if matchFlag == true
                        outAry[i][j] ← 1
            else
                        outAry[i][j] ← 0

********************************
XI. ComputeClosing (zeroFramedAry, morphAry, structAry, tempAry)
*******************************
step 1: ComputeDilation (zeroFramedAry, tempAry, structAry)
step 2: ComputeErosion (tempAry, morphAry, structAry)
```

**********
XII. ComputeOpening (zeroFramedAry, morphAry, structAry, tempAry)
**********
step 1: Compute Erosion (zeroFramedAry, tempAry, structAry)
step 2: ComputeDilation (tempAry, morphAry, structAry)