Project 2 (Java): You are to implement the three image enhancement methods taught in class: (1) 3x3 averaging, (2) 3x3 median filter, and (3) 3x3 2D-Gaussian filter.

****

Project points: 12 pts
Language: Java
Due Date: Soft copy (*.zip) and hard copies (*.pdf):

+1 (13/12 pts): early submission, 9/11/2022, Sunday before midnight
-0 (12/12 pts): on time, 9/14/2022 Wednesday before midnight
-1 (11/12 pts): 1 day late, 9/15/2022 Thursday before midnight
-2 (10/12 pts): 2 days late, 9/16/2022 Friday before midnight
(-12/12 pts): non submission, 9/16/2022 Friday after midnight

*** 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.

****

Include in your hard copy *.pdf file as follows:

- Cover page
- Source code
- inputImg
- AvgOut
- AvgThreshold
- MedianOut
- MedianThreshold
- GaussOut
- GaussThreshold

****

I. Input files:

a) inFile (args[0]): A txt file representing a grey-scale image with image header.
b) maskFile (args[1]): a mask for convolution, with the following format:
MaskRows MaskCols MaskMin MaskMax,
follow by MaskRows by MaskCols of pixel values
For example, a 3 by 3 mask may be
3 3 1 4
1 2 1
2 4 2
1 2 1

c) a threshold value (args[2]) // USE 38

****

II. Output files:

1) inputImg (args [3]): the input image after reformatting.
2) AvgOut (args[4]): the result of 3x3 averaging, after reformatting.
3) AvgThreshold (args[5])::The threshold result of 3x3 averaging, after reformatting.
4) MedianOut (args[6]): the result of 3x3 median filter, after reformatting.
5) MedianThreshold (args[7]): The threshold result of 3x3 median filter, after reformatting.
6) GaussOut (args[8]): the result of 3x3 Gaussian filter, after reformatting.
7) GaussThreshold (args[9]): The threshold result of 3x3 Gaussian filter, after reformatting.

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

- imageProcessing class
  - (int) numRows
  - (int) numCols
  - (int) minVal
  - (int) maxVal
  - (int) maskRows
  - (int) maskCols
  - (int) maskMin
  - (int) maskMax
  - (int) newMin
  - (int) newMax
  - (int) thrVal // from args[2]
  - (int) mirrorFramedAry [][] // a 2D array of size numRows + 2 by numCols + 2. dynamically allocate.
  - (int) avgAry [][] // a 2D array of size numRows + 2 by numCols + 2. dynamically allocate.
  - (int) medianAry [][] // a 2D array of size numRows + 2 by numCols + 2. dynamically allocate.
  - (int) GaussAry [][] // a 2D array of size numRows + 2 by numCols + 2. dynamically allocate.
  - (int) thrAry // a 2D array of size numRows + 2 by numCols + 2. dynamically allocate.
    // to hold the threshold result.
  - (int) mask2DAry [][] // a 2D Gaussian mask of size maskRows by maskCols used in the convolution,
  - (int) neighbor1DAry [9] //1-D array to hold a pixel[i,j]'s 3x3 neighbors for easy computation.
  - (int) mask1DAry [9] // to hold the 9 pixels of mask for easy computation.

methods:

- threshold (…)// see algorithm below.
- imgReformat (...) // see algorithm below.
  - mirrorFraming (...) // On your own. The algorithm of Mirror framing was taught in class
  - loadImage (...) // On your own. Read from input file and load onto mirrorFramedAry begin at [1][1].
  - loadMask (...)// // load maskFile onto mask2DAry. On your own.
  - loadMask1DAry (...) // On your own. Load 9 pixels of mask into mask1DAry;
    // using 2 loops; do NOT write 9 assignment.
  - loadNeighbor1DAry (...) // On your own. Load the 3 x 3 neighbors of mirrorFramedAry (i,j) into
    //neighbor1DAry, using 2 loops; do NOT write 9 assignment.
  - sort (neighborAry) // Use any sorting algorithm. Call build-in or write your own.
  - computeAvg (...) // process the mirrorFramedAry begin at [1][1]; keep track of newMin and newMax.
    // On your own. Similar to the computeMedian algorithm steps
  - computeMedian (...) // process the mirrorFramedAry begin at [1][1]; keep track of newMin and newMax.
    // See algorithm below.
  - computeGauss (...) // process the mirrorFramedAry begin at [1][1]; keep track of newMin and newMax.
    // See algorithm below.
  - (int) convolution (neighbor1DAry, mask1DAry) // See algorithm below.
  - imgReformat (…) // See algorithm below.

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

step 0: open inFile, maskFile
  open all outfiles
  thrVal ← get from args[2]
step 1: numRows, numCols, minVal, maxVal ← read from inFile
  maskRows, maskCols, maskMin, maskMax ← read from maskFile
step 2: dynamically allocate <u>all</u> 1-D and 2-D arrays
step 3: loadMask (...)
  loadMask1DAry (...)

```
step 4: loadImage (...)
step 5: mirrorFraming (...)
Step 6: imgReformat (mirrorFramedAry, minVal, maxVal, inputImg)
step 7: computeAvg (...)
        imgReformat (avgAry, newMin, newMax, avgOut)
        threshold (avgAry, thrAry)
        imgReformat (thrAry, newMin, newMax, MedianThreshold)
Step 8: computeMedian (...)
        imgReformat (medianAry, newMin, newMax, MedianOut)
        threshold (medianAry, thrAry)
        imgReformat (thrAry, newMin, newMax, MedianThreshold)
step 9: computeGauss (...)
        imgReformat (GaussAry, newMin, newMax, GaussOut)
        threshold (GaussAry, thrAry)
        imgReformat (thrAry, newMin, newMax, GaussThreshold)
step 10: close all files

************************************
V. computeMedian (...) //  keep track of newMin and newMax
************************************
step 0: newMin ← 9999; newMax ← 0
step 1: i ← 1
step 2: j ← 1
step 3: loadNeighbor1DAry (i, j, neighbor1DAry)
step 4: sort (neighborAry)
step 5: medianAry [i,j] ← neighborAry[4]
step 6: if newMin > medianAry [i,j]
                newMin ← medianAry [i,j]
        if newMax < medianAry [i,j]
                newMax ← medianAry [i,j]
step 7: j++
step 8: repeat step 3 to step 7 while j <= numCols
step 9: i++
step 10: repeat step 2 to step 9 while i <= numRows

************************************
VI. computeGauss (...) // keep track of newMin and newMax
************************************
step 0: newMin ← 9999; newMax ← 0
step 1: i ← 1
step 2: j ← 1
step 3: loadNeighbor1DAry (i, j, neighbor1DAry)
step 4: GaussAry [i,j] ← convolution (neighbor1DAry, mask1DAry)
step 4: if newMin > GaussAry [i,j]
                newMin ← GaussAry [i,j]
        if newMax < GaussAry [i,j]
                newMax ← GaussAry [i,j]
step 5: j++
step 6: repeat step 3 to step 5 while j <= numCols
step 7: i++
step 8: repeat step 2 to step 6 while I <= numRows
```

```
************************************
 VII. (int) convolution (neighbor1DAry, mask1DAry)
************************************
step 0: result ← 0
step 1: i ← 0
step 2: result += neighbor1DAry[i] * mask1DAry[i]
step 3: i++
step 4: repeat step 2 – step 3 while i < 9
step 5: return result

*******************************
VIII. imgReformat (inAry, newMin, newMax, OutImg)
*******************************
Step 1: OutImg ← output numRows, numCols, newMin, newMax
Step 2: str ← integer.toString(newMax) // Java build-in
            Width ← length of str
Step 3: r ← 1
Step 4: c ← 1
Step 5: OutImg ← inAry[r][c]
Step 6: str ← to_string (inAry[r][c])
            WW ← length of str
Step 7:  OutImg ← one blank space
            WW ++
Step 8: repeat step 7 while WW < Width
Step 9: c++
Step 10: repeat Step 5 to Step 9 while c <= numCols
Step 11: r++
Step 12: repeat Step 4 to Step 10 while r <= numRows

*******************************
VII. threshold (ary1, ary2)
*******************************
step 0: newMin ← 0
          newMax ← 1
step 1: i ← 1
step 2: j ← 1
step 3: if ary1[i][j] >= thrVal
                    ary2[i][j] ← 1
               else
                    ary2[i][j] ← 0
step 4: j++
step 5: repeat step 3 to step 4 while j < numCols+2
step 6: i++
step 7: repeat step 2 to step 6 while i < numRows+2
```