Project 6 (Java): You are to implement both 4-connected and 8-connected component algorithms as taught in class. Your program will let the user to choose which connectness (4 or 8) to run the program, via args [].

*** You will be given two data files, data1 and data2, and the answer file of data1.
What you need to do as follows:
a) Implement your program based on the specs given below.
b) Test and debug your program using data1 for 8-connected until it produces the same result as given in the answer file.
c) Test and debug your program using data1 for 4-connected until it produces the same result as given in the answer file.
d) Run your program using data2 for 8-connected (Eyeball the result for correctness.)
e) Run your program using data2 for 4-connected (Eyeball the result for correctness and See if you know the meaning of the result in e).
** On each run, your program will produce four files: prettyPrintFile, LabelFile, propertyFile and logFile.
** labelFile and propertyFile will be used as input in your future project(s).

Your hard copies include:
- Cover page
- Source code
- prettyPrintFile for 8-connectness run on data1
- labelFile for 8-connectness run on data1
- propertyFile for 8-connectness run on data1
- logFile // limited to 2 pages if more than 2.

- prettyPrintFile for 4-connectness run on data1
- labelFile for 4-connectness run on data1
- propertyFile for 4-connectness run on data1
- logFile // limited to 2 pages if more than 2.

- prettyPrintFile for 8-connectness run on data2
- labelFile for 8-connectness run on data2
- propertyFile for 8-connectness run on r data2
- logFile // limited to 2 pages if more than 2.

- prettyPrintFile for 4-connectness run on data2
- labelFile for 4-connectness run on data2
- propertyFile for 4-connectness run on data2
- logFile // limited to 2 pages if more than 2.

*******************************
Language: Java
*************************************
Project name: Connected Components algorithms
Project points:12 pts
Due Date: <u>Soft copy (*.zip) and hard copies (*.pdf)</u>:
(13/12 pts): early submission. 11/3/2024 Sunday before midnight
(12/12 pts): on time, 11/6/2024 Wednesday before midnight
(-12/12 pts): non-submission, 11/6/2024 Wednesday after midnight
**Name your soft copy and hard copy files the naming convention given in Project Submission Requirements. (-2 if not.)
*** All on-line submission MUST include Soft copy (*.zip) and hard copy (*.pdf) in the same email attachments with correct email subject as below; otherwise, your submission will be rejected.

Email subject: (CV) first name last name <Project 6: Connected Components algorithms (Java)>

*** Inside the email body include your answer to the 4 questions. Optional screen recording if you wish.

***********************************
I. Inputs:
a) inFile (args [0]): A binary image.
b) Connectness (args [1]): 4 for 4-connectness, 8 for 8-connectness.
II. Outputs:
a) prettyPrintFile (args [2]): as specs dictates.

b) labelFile (args [3]): to store the connected component labels of Pass-3 -- the labelled image file with image header, numRows numCols newMin NewMax.

** This file to be used in future processing, therefore, it should include background 0.

c) propertyFile (args [4]): To store the connected component properties.

*** This file to be used in future processing.

The format is to be as below:

- 1st text-line, the header of the input image,
- 2nd text-line is the total number of connected components.
- label
- number of pixels
- upperLftR upperLftC //the r c coordinated of the upper left corner
- lowerRgtR lowerRgtC //the r c coordinated of lower right corner
- label
- number of pixels
- minR, minC //the r c coordinated of the upper left corner
- maxR, maxC //the r c coordinated of lower right corner

For an example:

```
45  40  0   9  // image header
9              // indicates there are a total of 9 CCs in the image
1              // CC label 1
187         // 187 pixels in CC label 1
4    9   // upper left corner of the bounding box at row 4 column 9
35  39  // lower right corner of the bounding box at row 35 column 39
2                  // CC label 2
36          // 36 pixels in CC label 2
14    19   // upper left corner of the bounding box at row 14 column 19
25  49  // lower right corner of the bounding box at row 25 column 49                    :
:
```

d) logFile (argv [5]) : As specs dictates.

*******************************

III. Data structure:

*******************************

- A Property (1D struct)
  - (int) label        // The component label
  - (int) numPixels // total number of pixels in the cc.
  - (int) minR // with respect to the input image.
  - (int) minC // with respect to the input image.
  - (int) maxR // with respect to the input image.
  - (int) maxC // with respect to the input image.

  // In the Cartesian coordinate system, any rectangular box can be represented by two points: upper-left corner and the lower-right of the box. Here, the two points:(minR minC) and (maxR maxC) represents the smallest rectangular box that a cc can fit in the box; object pixels can be on the border of the box.
- A ccLabel class
  - (int) numRows
  - (int) numCols
  - (int) minVal
  - (int) maxVal
  - (int) newLabel // initialize to 0
  - (int) trueNumCC // the true number of connected components in the image
  - (int) newMin // set to 0
  - (int) newMax // set to trueNumCC
  - (int [][]) zeroFramedAry// a 2D array of size numRows + 2 by numCols + 2, dynamically allocate at run time
  - (int) NonZeroNeighborAry [5] // 5 is the max number of neighbors you have to check. For easy programming, //you may consider using this 1-D array to store pixel [i, j]'s non-zero neighbors during pass 1 and pass2.
  - (int []) EQTable // a 1-D array, of size (numRows * numCols) / 4 // dynamically allocate at run time, and initialize to its index, i.e., EQTable [i] = i.

- (Property []) CCproperty // A struct 1D array (the size is the trueNumCC+1) to store components' properties.
    // dynamically allocate at runtime.

- methods:
    - constructor(...) // need to dynamically allocate all arrays; and assign values to numRows, etc.
    - zero2D (...) // Initialize 2D array to zero.
    - negative1D (...) // Initialize a 1-D array to -1.
    - loadImage (...) // read from input file and write to zeroFramedAry begin at (1,1)
    - printImg (zeroFramedAry, labelFile) // Output the result of pass3 inside of zeroFramedAry,
      //including 0's and image header. On your own.
    - prettyDotPrint (zeroFramedAry, prettyPrintFile)
      // Print zeroFramedAry to prettyPrintFile. Modify prettyPrint to replace one of blanks with a dot '.'.
    - connect8Pass1 (...) // On your own, algorithm was presented in class.
    - connect8Pass2 (...) // On your own, algorithm was presented in class.
    - connect4Pass1 (...) // On your own, algorithm was presented in class.
    - connect4Pass2 (...) // On your own, algorithm was presented in class.
    - connectPass3 (...) // See algorithm below.
    - updateEQTable (...) // Update EQTable for all non-zero neighbors to minLabel.
      // In case 3 of the pass1 and pass2 of both connectness, the method needs to update EQTable for those non-minimum label of neighbors of p(i, j) to minLabel; it is easier to use NonZeroNeighborAry, at first to store all non-zero neighbors of p(i, j) in ascending order to find minLabel, then update EQTable .
    - (int) manageEQTable (...) // The algorithm was given in class. The method returns the true number of CCs.
    - printCCproperty (...) // Prints the component properties to propertyFile using the format given in the above.
    - printEQTable (...) // Print EQTable with index up to newLabel, not beyond, in the follow format:

| Index | EQTable [index] |
|---|---|
| 0 | EQTable [0] |
| 1 | EQTable [1] |
| 2 | EQTable [2] |
| : | : |
| newLabel | EQTable [newLabel] |

    - drawBoxes (...) // Draw the bounding boxes of CC in zeroFramedAry. See algorithm below.

*******************************
IV. main(...)
*******************************

step 0: inFile, prettyPrintFile, labelFile, propertyFile, logFile ← open via args []
    Connectness ← args [1]
    numRows, numCols, minVal, maxVal ← read from inFile
    zeroFramedAry ← dynamically allocate.
    newLabel ← 0
step 1: zero2D (zeroFramedAry)
step 2: loadImage (inFile, zeroFramedAry)
step 3: if connectness == 4
    connected4 (zeroFramedAry, newLabel, EQTable, prettyPrintFile, logFile)
step 4: if connectness == 8
    connected8 (zeroFramedAry, newLabel, EQTable, prettyPrintFile, logFile)
step 5: labelFile ← output numRows, numCols, newMin, newMax to labelFile
step 6: printImg (zeroFramedAry, labelFile) // Output the result of pass3 inside of zeroFramedAry.
step 7: printCCproperty (propertyFile) // print cc properties to propertyFile
step 8: drawBoxes (zeroFramedAry, CCproperty, trueNumCC, logFile) // draw on zeroFramed image.
step 9: prettyDotPrint (zeroFramedAry, prettyPrintFile)
step 10: prettyPrintFile ← print trueNumCC to prettyPrintFile with proper caption
step 12: close all files

******************************

V. connected4 (zeroFramedAry, newLabel, EQTable, prettyPrintFile, logFile)

******************************

Step 0: logFile ← "entering connected4 method"

Step 1: connect4Pass1 (zeroFramedAry, newLabel, EQTable, logFile)
    logFile ← "After connected4 pass1, newLabel =" // print newLabel
    prettyDotPrint (zeroFramedAry, prettyPrintFile)
    printEQTable (newLabel, prettyPrintFile) // print the EQTable up to newLabel with proper caption

Step 2: Connect4Pass2 (zeroFramedAry, EQTable, logFile)
    logFile ← "After connected4 pass2, newLabel =" // print newLabel
    prettyDotPrint (zeroFramedAry, prettyPrintFile)
    printEQTable (newLabel, prettyPrintFile) // print the EQTable up to newLabel with proper caption

Step 3: trueNumCC ← manageEQTable (EQTable, newLabel)
    printEQTable (newLabel, prettyPrintFile) // print the EQTable up to newLabel with proper caption
    newMin ← 0
    newMax ← trueNumCC
    CCproperty ← dynamically allocate size of trueNumCC+1
    logFile ← "In connected4, after manage EQAry, trueNumCC =" // print trueNumCC

Step 4: connectPass3 (zeroFramedAry, EQAry, CCproperty, trueNumCC, logFile) // see algorithm below.

Step 5: prettyDotPrint (zeroFramedAry, prettyPrintFile)

Step 6: printEQTable (newLabel, prettyPrintFile) // print the EQTable up to newLabel with proper caption

Step 7: logFile ← "Leaving connected4 method"

******************************

VI. connected8 (zeroFramedAry, newLabel, EQTable, RFprettyPrintFile, logFile)

******************************

Step 0: logFile ← "entering connected8 method"

Step 1: connect8Pass1 (zeroFramedAry, newLabel, EQTable, logFile)
    logFile ← "After connected8 pass1, newLabel =" // print newLable
    prettyDotPrint (zeroFramedAry, prettyPrintFile)
    printEQTable (newLabel, prettyPrintFile) // print the EQTable up to newLabel with proper caption

Step 2: Connect8Pass2 (zeroFramedAry, EQTable, logFile)
    logFile ← "After connected8 pass2, newLabel =" // print newLable
    prettyDotPrint (zeroFramedAry, prettyPrintFile)
    printEQTable (newLabel, prettyPrintFile) // print the EQTable up to newLabel with proper caption

Step 3: trueNumCC ← manageEQTable (EQTable, newLabel, logFile)
    printEQTable (newLabel, prettyPrintFile) // print the EQTable up to newLabel with proper caption
    newMin ← 0
    newMax ← trueNumCC
    CCproperty ← dynamically allocate size of trueNumCC+1
    logFile ← "In connected8, after manage EQTable, trueNumCC =" // print trueNumCC

Step 4: connectPass3 (zeroFramedAry, EQTable, CCproperty, trueNumCC, logFile) // see algorithm below.

Step 5: prettyDotPrint (zeroFramedAry, prettyPrintFile)

Step 6: printEQTable (newLabel, prettyPrintFile) // print the EQTable up to newLabel with proper caption

Step 7: logFile ← "Leaving connected8 method"

******************************
VII. connectPass3 (zeroFramedAry, EQTable, CCproperty, trueNumCC, logFile)
*******************************

Step 0: logFile ← "entering connectPas3 method"
Step 1: for i = 1 to trueNumCC
    CCproperty[i].label ← i
    CCproperty[i].numPixels ← 0
    CCproperty[i].minR ← numRows
    CCproperty[i].maxR ← 0
    CCproperty[i].minC ← numCols
    CCproperty[i].maxC ← 0

Step 2: scan inside of the zeroFramedAry left-right & top-bottom
    p(r, c) ← next pixel

Step 3: if p(r, c) > 0
    zeroFramedAry [r, c] ← EQTable [p(r, c)] // relabeling.
    k ← zeroFramedAry [r, c]
    CCproperty[k].numPixels++
    if r < CCproperty[k].minR
        CCproperty[k].minR ← r
    if r > CCproperty[k].maxR
        CCproperty[k].maxR ← r
    if c < CCproperty[k].minC
        CCproperty[k].minC ← c
    if c > CCproperty[k].maxC
        CCproperty[k].maxC ← c

Step 4: repeat Step 2 to Step 3 until all pixels inside of zeroFramedAry are processed
Step 5: logFile ← "leaving connectPas3 method"

*******************************
VIII. drawBoxes (zeroFramedAry, CCproperty, trueNumCC, logFile)
*******************************

Step 0: logFile ← "entering drawBoxes method"
step 1: index ← 1
step 2: minRow ← CCproperty[index]'s minR + 1
    minCol ← CCproperty[index]'s minC + 1
    maxRow ← CCproperty[index]'s maxR + 1
    maxCol ← CCproperty[index]'s maxC + 1
    label ← CCproperty[index]'s label

step 3: Assign label to all pixels on minRow of zeroFramedAry, from minCol to maxCol ← label //use a loop.
    Assign label to all pixels on maxRow of zeroFramedAry, from minCol to maxCol ← label //use a loop.
    Assign label to all pixels on minCol of zeroFramedAry, from minRow to maxRow ← label //use a loop.
    Assign label to all pixels on maxCol of zeroFramedAry, from minRow to maxRow ← label //use a loop.

step 4: index++
step 5: repeat step 2 to step 4 while index <= trueNumCC
Step 6: logFile ← "leaving drawBoxes method"