Project 5 (Java): You are to implement both 4-connected and 8-connected component algorithms as taught in class. Your program let the user to choose which connectness (4-CC or 8-CC) to run the program, via args[1].

*** You will be given two data files, data1 and data2, and the answer for 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 answer.
c) Test and debug your program using data1 for 4-connected until it produces the same result as given in answer.
d) Then, run your program twice on data2; first using 8 and then using 4. (Eyeball the result for correctness.)

** On each run, your program will produce three files: RFprettyPrintFile, LabelFile, and propertyFile.
Your hard copies include:

- Cover page
- Source code
- RFprettyPrintFile for 8-connectness run on data1
- labelFile for 8-connectness run on data1
- propertyFile for 8-connectness run on data1

- RFprettyPrintFile for 4-connectness run on data1
- labelFile for 4-connectness run on data1
- propertyFile for 4-connectness run on data1

- RFprettyPrintFile for 8-connectness run on data2
- labelFile for 8-connectness run on data2
- propertyFile for 8-connectness run on r data2

- RFprettyPrintFile for 4-connectness run on data2
- labelFile for 4-connectness run on data2
- propertyFile for 4-connectness run on data2

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

+1 (13/12 pts): early submission, 10/23/2022, Sunday before midnight
-0 (12/12 pts): on time, 10/27/2022 Thursday before midnight
-1 (11/12 pts): 1 day late, 10/28/2022 Friday before midnight
-2 (10/12 pts): 2 days late, 10/29/2022 Saturday before midnight
(-12/12 pts): non submission, 10/29/2022 Saturday 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 **<u>the same email attachments</u>** with correct email subject as stated in the email requirement; otherwise, your submission will be rejected.
****

I. Inputs: a) inFile (args[0]): A binary image.
b) whichConnectness (args[1]): first 8, then 4.

II. Outputs: a) RFprettyPrintFile (args[2]): (include in your hard copy) for the followings:
** a proper caption means the caption should say what the printing is.

- reformatPrettyPrint of the result of the Pass-1 with proper captions
- print newLabel and the EQAry after Pass-1, with proper captions
- reformatPrettyPrint of the result of the Pass-2 with proper captions
- print newLabel and the EQAry after Pass-2, with proper captions
- Print the EQAry after manage the EQAry, with proper caption
- reformatPrettyPrint of the result of the Pass-3 with proper captions
- reformatPrettyPrint of the result bounding boxes drawing.

b) labelFile (args[3]): to store the result of Pass-3 -- the labelled image file
with image header, numRows numCols newMin NewMax. ** This file to be used in future processing.

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
- upperLftR upperLftC //the r c coordinated of the upper left corner
- lowerRgtR lowerRgtC //the r c coordinated of lower right corner

:

For an example:

```
45  40  0   9  // image header
9               // 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
        :
```

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

- 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
    // It will be determined in manageEQAry method.
  - (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) EQAry [] // an 1-D array, of size (numRows * numCols) / 4
    // dynamically allocate at run time, and initialize to its index, i.e., EQAry[i] = i.
  - Property (1D struct or class)
    - (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 the cc can fit in the box; object pixels can be on the border of the box.

- (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 (...) // ** Initialized a 2-D array to zero. You must implement this method, don't count on Java.
    - minus1D (...) // ** Initialized a 1-D array to -1.
    - loadImage (...) // read from input file and write to zeroFramedAry begin at(1,1)
    - imgReformat (zeroFramedAry, RFprettyPrintFile)
        // Print zeroFramedAry to RFprettyPrintFile. Reuse code from your previous project.
    - connect8Pass1 (...) // On your own, as taught in class.
    - connect8Pass2 (...) // On your own, as taught in class.
    - connect4Pass1 (...) // On your own, as taught in class.
    - connect4Pass2 (...) // On your own, as taught in class.
    - connectPass3 (...) // See algorithm below.
    - drawBoxes (...) // Draw the bounding boxes in zeroFramedAry. See algorithm below
    - updateEQ (...) // Update EQAry for all non-zero neighbors to minLabel.
        // It will be easier to use NonZeroNeighborAry to store all non-zero neighbors to find min label.
    - (int) manageEQAry (...) // The algorithm was taught in class.
        // The method returns the true number of CCs in the labelled image.
    - printCCproperty (...) // Prints the component properties to propertyFile using the format given in the above.
    - printEQAry (...) // Print EQAry with index up to newLabel, not beyond. On your own
    - printImg (...) // <u>Output image header</u> and zeroFramedAry (inside of framing) to labelFile. On your own.

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

```
step 0: inFile ← open the input file
        RFprettyPrintFile , labelFile, propertyFile ← open from args[]
        numRows, numCols, minVal, maxVal ← read from inFile
        dynamically allocate zeroFramedAry.
        newLabel ← 0
step 1: zero2D (zeroFramedAry)
step 2: loadImage (inFile, zeroFramedAry)
step 3: Connectness ← args[1]
step 4: if connectness == 4
            connect4Pass1 (zeroFramedAry, newLabel, EQAry)
            imgReformat (zeroFramedAry, RFprettyPrintFile)
            printEQAry (newLabel, RFprettyPrintFile)  // print the EQAry up to newLable with proper caption
            Connect4Pass2 (zeroFramedAry, EQAry)
            imgReformat (zeroFramedAry, RFprettyPrintFile)
            printEQAry (newLabel, RFprettyPrintFile) // print the EQAry up to newLabel with proper caption
step 5: if connectness == 8
            connect8Pass1 (zeroFramedAry, newLabel, EQAry)
            imgReformat (zeroFramedAry, RFprettyPrintFile)
            printEQAry (newLabel, RFprettyPrintFile) // print the EQAry up to newLabel with proper caption
            Connect8Pass2 (zeroFramedAry, EQAry)
            imgReformat (zeroFramedAry, RFprettyPrintFile)
            printEQAry (newLabel, RFprettyPrintFile) v// print the EQAry up to newLabel with proper caption

step 6: trueNumCC ← manageEQAry (EQAry, newLabel)
            printEQAry (newLabel, RFprettyPrintFile) // print the EQAry up to newLabel with proper caption
            newMin ← 0
            newMax ← trueNumCC
            dynamically allocate CCproperty []  size of trueNumCC+1
```

step 7: connectPass3 (zeroFramedAry, EQAry, CCproperty) // see algorithm below.
step 8: imgReformat (zeroFramedAry, RFprettyPrintFile)
step 9: printEQAry (newLabel, RFprettyPrintFile) // print the EQAry up to newLabel with proper caption
step 10: labelFile ← output numRows, numCols, newMin, newMax to labelFile
step 11: printImg (labelFile) // Output the result of pass3 inside of zeroFramedAry
step 12: printCCproperty (propertyFile) // print cc properties to propertyFile
step 13: drawBoxes(zeroFramedAry, CCproperty) // draw on zeroFramed image.
step 14: imgReformat (zeroFramedAry, RFprettyPrintFile)
step 15: print trueNumCC to RFprettyPrintFile with proper caption
step 16: close all files

****
V. connectPass3 (zeroFramedAry, EQAry, CCproperty)
****

```
Step 0: for i = 1 to trueNumCC
            CCproperty[i].label ← i
            CCproperty[i].numPixels ← 0
            CCproperty[i].MinR ← numRow
            CCproperty[i].MaxR ← 0
            CCproperty[i].MinC ← numCol
            CCproperty[i].MaxC ← 0

Step 1: scan inside of the zeroFramedAry left-right & top-bottom
        p(r, c) ← next pixel

Step 2: if p(r, c) > 0
            zeroFramedAry [r, c] ← EQAry[p(r, c)]
            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 3: repeat Step 1 to Step 2 until all pixels inside of zeroFramedAry are processed
```

****
VI. drawBoxes (zeroFramedAry, CCproperty)
****

```
// This method may contain bugs, report bugs to Dr. Phillips
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 all pixels on minRow from minCol to maxCol ← label
        Assign all pixels on maxRow from minCol to maxCol ← label
        Assign all pixels on minCol from minRow to maxRow ← label
        Assign all pixels on maxCol from minRow to maxRow ← label

step 4: index++
step 5: repeat step 2 to step 4 while index <= trueNumCC
```