Project 6 (in Java): You are to implement quadtree representation of a given image.

You will be given two images (img1 and img2) to test your program.
Run your program twice, one on each test image.
For each image, print the image on the top of a blank page, then draw the quadtree representation of the image on the bottom of the blank.

Hard copy includes:

- Cover sheet (one page)
- The page has img1 on the top and its quadtree representation on the bottom
- The page has img2 on the top and its quadtree representation on the bottom
- Source code
- outFile1 for img1
- outFile2 for img1
- outFile1 for img2
- outFile2 for img2

****

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

+1 (11/10 pts): early submission, 4/6/2022 Wednesday before midnight.
-0 (10/10 pts): on time, 4/8/2022 Friday before midnight.
-1 (9/10 pts): 1 day late, 4/9/2022 Saturday before midnight.
-2 (8/10 pts): 2 days late, 4/10/2022 Sunday before midnight.
(-10/10 pts): none submission, 4/10/2022 Sunday after midnight
(-5/10): does not pass compilation.
(0/10): program produces no output.
(0/10): does not submit hard copy.

*** 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.

****

I. Input: inFile(args[0]) a text file contains a binary image with header information. The first text-line is the header information: numRows numCols minVal maxVal follows by rows and columns of pixels, 0's and 1's. An example below is a binary image has four (4) rows and five (5) columns, minVal is 0 and maxVal is 1:

```
4 5 0 1 ← header
0 0 0 1 0
0 1 1 0 1
0 0 1 0 1
1 0 1 1 1
```

****

II. Outputs:

outFile1 (arvs[1]): the pre-order
and post-order traversals of the quadtree
(needs caption for each traversal!)

outFile2 (args[2]): all the debugging outputs to get some partial credits.

****

III. Data structure: // YOU MUST implement all methods!!!

****

- QtNode class
  - (int) color // 0, 1 or 5
  - (int) upperR // the row coordinate of the upper corner
    //of the image area in which this node is representing

- (int) upperC // the column coordinate of the upper corner
//of the image area in which this node is representing
- (int) Size
- (QtNode) NWkid // initialize to null
- (QtNode) NEkid // initialize to null
- (QtNode) SWkid // initialize to null
- (QtNode) SEkid // initialize to null

Method:
- constructor (upperR, upperC, Size, NWkid, NEkid, SWkid, SEkid) // to create a QtTreeNode
- printQtNode (node, outFile) // output to outFile: node's: color, upperR, upperC, NWkid's color,
//NEkid's color, SWkid's color, SEkid's color), one node per text line

- A QuadTree class
- (int) numRows
- (int) numCols
- (int) minVal
- (int) maxVal
- (int) power2
- (int [][]) imgAry
// a 2D array, need to dynamically allocate at run time of size power2 by power2.
- (QtNode) QtRoot

methods:
- constructor (…) // performs necessary allocations and initializations
- (int) computePower2 (...) // Algorithm is given below
// The method determines the smallest power of 2
// that fits the input image. It returns the
// the size (one side) of the power2Size
- loadImage (...) //load the input data onto imgAry, begins at (0, 0)
// Algorithm is given below
- zero2DAry (...) // set the given 2D array to zero; may not needed this since Java initializes array to zeros

methods:
- (QtTreeNode) buildQuadTree (...) // A recursive method. Algorithm is given below.
- sumKidsColor (node) // add up all node's 4 kids colors; on your own.
- setLeaf (node) // set node's four kid to null.
- (bool) isLeaf (node) // returns true if node is a quadtree leaf node (its color is 0 or 1), otherwise returns false.
- preOrder (...) // see algorithm steps below
- postOrder (...) // similar to preOrder except where to print the node; on your own

****

IV. Main (...)

****

step 0: inFile, outFile1, outFile2 ← open
numRows, numCols, minVal, maxVal ← read from inFile
power2 ← computePower2(numRows, numCols)
outFile2 ← output power2 to outFile2 with caption
imgAry ← dynamically allocate the array size of power2 by power2Size
zero2DAry (imgAry) // May not need it.
step 1: loadImage (inFile, imgAry)
step 2: outFile2 ← output imgAry to outFile2 with caption
step 3: QtRoot ← BuildQuadTree (imgAry, 0, 0, power2, outFile2)
step 4: preOrder (QtRoot, outFile1)
step 5: postOrder (QtRoot, outFile1)
step 6: close all files

```
*******************************
V. loadImage(inFile, imgAry)
*******************************
Step 0: r ← 0
Step 1: c ← 0
Step 2: data ← read one data from inFile // either 0 or 1.
Step 3: imgAry[r][c] ← data
Step 4: c++
Step 5: repeat step 2 to Step 4 while c < numCols
Step 6: repeat Step 1 to Step 5 while r < numRows
*******************************
VI. (QtTreeNode*) buildQuadTree (imgAry, upR, upC, size, outFile2)
*******************************
Step 1: newQtNode ← get a new QtTreeNode (-1, upR, upC, size, null, null, null, null)
        printQtNode (newQtNode, outFile2)
Step 2: if size == 1 // one pixel
                newQtNode's color ← imgAry[upR, upC] // either 1 or 0
        else    {
                halfSize ← size / 2
                newQtNode's NWkid ← buildQuadTree (imgAry, upR, upC, halfSize)
                newQtNode's NEkid ← buildQuadTree (imgAry, upR, upC + halfSize, halfSize)
                newQtNode's SWkid ← buildQuadTree (imgAry, upR + halfSize, upC, halfSize)
                newQtNode's SEkid ← buildQuadTree (imgAry, upR + halfSize, upC + halfSize, halfSize)
                sumColor ← sumKidsColor (newQtNode) // add up all newQtNode's 4 kids colors
                if sumColor == 0  // all 4 kids are 0
                                newQtNode's color ← 0
                                setLeaf (newQtNode)
                else { if sumColor == 4  // all 4 kids are 1
                                newQtNode's color ← 1
                                setLeaf (newQtNode)
                        else
                                newQtNode's color ← 5
                        }
                }
step 3: return newQtNode
*******************************
VI. (int) computePower2 (numRows, numCols)
*******************************
step 0: size ← max (numRows, numCols)
step 1: power2 ← 2
step 2: if size > power2
                power2 *= 2
step 3: repeat step 2 until condition fail
step 4: return power2
*******************************
 void preOrder (Qt, outFile1)
*******************************
If  isleaf  (Qt)
        printQtNode (Qt, outFile1)
else
        printQtNode (Qt, outFile1)
        preOrder (Qt's NWkid)
        preOrder (Qt's NEkid)
        preOrder (Qt's SWkid)
        preOrder (Qt's SEkid)
```