Institute of Technology, Carlow

B.Sc. Hons. in Software Engineering

CW228

Code Listing

Project Title: **Number Plate Recognition**

Name: Dongfan Kuang

Login ID: C00131031

Supervisor: Nigel Whyte

Date: 16 April 2010

Table of Contents

1. GrayScale.java

```
package numberplaterecognition;
import java.awt.image.BufferedImage;

/**
 * This class contains the graysclae algorithm, when create a
 * new GrayScale, first specifiy a bufferedImage type source
 * image by calling setSourceImage(), then call grayscale()
 * function to perform grayscale algprithm to the source image,
 * finally, use getGrayscaleImage() function to get the graysclaed
 * image.
 */
public class GrayScale {

   int sourceImageHeight;
   int sourceImageWidth;
   int imageSize;
   BufferedImage sourceImage;
   BufferedImage grayscaleImage;

   /**
    * set the source image to be used to grayscale
    */
   public void setSourceImage(BufferedImage image)
   {
      sourceImage = image;
   }

   /**
    * get the bufferedImage with grayscaled image after
    * called the grayscale() function
    */
   public BufferedImage getGrayscaleImage()
   {
      return grayscaleImage;
   }
```

```
  /**
   * perform grayscale algorithm to the souece image
   */
  public void grayscale()
  {
    sourceImageWidth = sourceImage.getWidth();
    sourceImageHeight = sourceImage.getHeight();
    imageSize = sourceImageWidth * sourceImageHeight;

    byte[] grayscaleData = new byte[imageSize];
    byte[] pixels = (byte[]) sourceImage.getData().getDataElements(0, 0,
        sourceImageWidth, sourceImageHeight, null);

    int offset = 0;
    for (int i = 0; i < imageSize; i++)
    {
      //read the pixel information from source image
      int b = pixels[offset++] & 0xff;
      int g = pixels[offset++] & 0xff;
      int r = pixels[offset++] & 0xff;

      //change the pixel into grayscale
      int gb = ((b*7472)>>16)&0xff;
      int gg = ((g*38469)>>16)&0xff;
      int gr = ((r*19595)>>16)&0xff;

      //store the grayscale pixel back into the array.
      grayscaleData[i] = (byte)((gb + gg + gr)&0xFF);
    }

    grayscaleImage = new BufferedImage(sourceImageWidth, sourceImageHeight,
        BufferedImage.TYPE_BYTE_GRAY);
    grayscaleImage.getRaster().setDataElements(0, 0, sourceImageWidth,
        sourceImageHeight, grayscaleData);
  }
}
```

2. CannyEdgeDetection.java

```
package numberplaterecognition;
import java.awt.image.BufferedImage;
import java.util.Arrays;

/**
 * This class contains the Canny edge detection algorithm,
 * when create a new CannyEdgeDetection, first specifiy a
 * bufferedImage type source image by calling setSourceImage(),
 * then call process() function to perform edge detection
 * algprithm to the source image, finally, use getEdgesImage()
 * function to get the image only have edges information.
 */
public class CannyEdgeDetection
{

   int sourceImageHeight;
   int sourceImageWidth;
   int ImageSize;  //ImageSize = sourceImageHeight * sourceImageWidth
   int[] imageData;
   int[] edgeData;
   int[] magnitude;

   BufferedImage sourceImage;
   BufferedImage edgesImage;

   int gaussianKernelWidth;
   float gaussianKernelRadius;
   float lowThreshold;
   float highThreshold;
```

```
float[] xConv;
float[] yConv;
float[] xGradient;
float[] yGradient;

/**
 * set default values
 */
public CannyEdgeDetection()
{
   lowThreshold = 7f;
   highThreshold = 8f;
   gaussianKernelRadius = 2f;
   gaussianKernelWidth = 16;
}

/**
 * set the source image to be used to detect edges
 */
public void setSourceImage(BufferedImage image)
{
   sourceImage = image;
}

/**
 * perform the edge detect algorithm to the source image
 */
public void runProcess()
{
   sourceImageWidth = sourceImage.getWidth();
   sourceImageHeight = sourceImage.getHeight();
   ImageSize = sourceImageWidth * sourceImageHeight;
   imageData = new int[ImageSize];
```

```
    edgeData = new int[ImageSize];
    magnitude = new int[ImageSize];
    xConv = new float[ImageSize];
    yConv = new float[ImageSize];
    xGradient = new float[ImageSize];
    yGradient = new float[ImageSize];

    readSourceImageData();
    computeGradients(gaussianKernelRadius, gaussianKernelWidth);
    int low = Math.round(lowThreshold * 100F);
    int high = Math.round( highThreshold * 100F);
    edgeDetect(low, high);
    thresholdEdges();
    writeEdges(edgeData);
}

/*
 * get the bufferedImage with edge information
 * after called the process() function
 */
public BufferedImage getEdgesImage()
{
    return edgesImage;
}

/**
 * return the array which stores grayscale data
 * after called the process() function
 */
public int[] getGraysclaeData()
{
    return imageData;
}
```

```
/*
 * Read the pixels information from the source image,
 * Only suport image with TYPE_3BYTE_BGR format
 */
private void readSourceImageData()
{
   byte[] pixels = (byte[]) sourceImage.getData().getDataElements(0, 0,
         sourceImageWidth, sourceImageHeight, null);
   for (int i = 0; i < ImageSize; i++)
   {
      imageData[i] = (int)(pixels[i] & 0xFF);
   }
}



/*
 * perform convolution and non-maximal supression
 */
private void computeGradients(float kernelRadius, int kernelWidth)
{

   //Generate the gaussian convolution masks Default
   //set kernelRadius = 2, kernelWidth = 16.
   float kernel[] = new float[kernelWidth];
   float diffKernel[] = new float[kernelWidth];
   int kwidth;
   for (kwidth = 0; kwidth < kernelWidth; kwidth++)
   {
      float g1 = gaussian(kwidth, kernelRadius);
      if (g1 <= 0.005f && kwidth >= 2)
         break;
      float g2 = gaussian(kwidth - 0.5f, kernelRadius);
```

```
   float g3 = gaussian(kwidth + 0.5f, kernelRadius);
   kernel[kwidth] = (g1 + g2 + g3) / 3f /
         (2f * (float) Math.PI * kernelRadius * kernelRadius);
   diffKernel[kwidth] = g3 - g2;
}

int initX = kwidth - 1;
int maxX = sourceImageWidth - (kwidth - 1);
int initY = sourceImageWidth * (kwidth - 1);
int maxY = sourceImageWidth * (sourceImageHeight - (kwidth - 1));

//perform convolution in x and y directions
for (int x = initX; x < maxX; x++)
{
   for (int y = initY; y < maxY; y += sourceImageWidth)
   {
      int index = x + y;
      float sumX = imageData[index] * kernel[0];
      float sumY = sumX;
      int xOffset = 1;
      int yOffset = sourceImageWidth;
      for(; xOffset < kwidth ;)
      {
         sumY += kernel[xOffset] * (imageData[index - yOffset]
               + imageData[index + yOffset]);
         sumX += kernel[xOffset] * (imageData[index - xOffset]
               + imageData[index + xOffset]);
         yOffset += sourceImageWidth;
         xOffset++;
      }
      yConv[index] = sumY;
      xConv[index] = sumX;
   }
```

```
}

//calculate gradients in x direction
for (int x = initX; x < maxX; x++)
{
   for (int y = initY; y < maxY; y += sourceImageWidth)
   {
      float sum = 0f;
      int index = x + y;
      for (int i = 1; i < kwidth; i++)
         sum += diffKernel[i] *
               (yConv[index - i] - yConv[index + i]);
      xGradient[index] = sum;
   }
}

//calculate gradients in y direction
for (int x = kwidth; x < sourceImageWidth - kwidth; x++)
{
   for (int y = initY; y < maxY; y += sourceImageWidth)
   {
      float sum = 0.0f;
      int index = x + y;
      int yOffset = sourceImageWidth;
      for (int i = 1; i < kwidth; i++)
      {
         sum += diffKernel[i] * (xConv[index - yOffset]
               - xConv[index + yOffset]);
         yOffset += sourceImageWidth;
      }
      yGradient[index] = sum;
   }
}
```

```
initX = kwidth;
maxX = sourceImageWidth - kwidth;
initY = sourceImageWidth * kwidth;
maxY = sourceImageWidth * (sourceImageHeight - kwidth);
for (int x = initX; x < maxX; x++)
{
   for (int y = initY; y < maxY; y += sourceImageWidth)
   {
      int index = x + y;
      int indexN = index - sourceImageWidth;
      int indexS = index + sourceImageWidth;
      int indexW = index - 1;
      int indexE = index + 1;
      int indexNW = indexN - 1;
      int indexNE = indexN + 1;
      int indexSW = indexS - 1;
      int indexSE = indexS + 1;

      float xGrad = xGradient[index];
      float yGrad = yGradient[index];
      float gradMag = (float) Math.hypot(xGrad, yGrad);

      //perform non-maximal supression
      float nMag = (float) Math.hypot
            (xGradient[indexN], yGradient[indexN]);
      float sMag = (float) Math.hypot
            (xGradient[indexS], yGradient[indexS]);
      float wMag = (float) Math.hypot
            (xGradient[indexW], yGradient[indexW]);
      float eMag = (float) Math.hypot
            (xGradient[indexE], yGradient[indexE]);
      float neMag = (float) Math.hypot
```

```
        (xGradient[indexNE], yGradient[indexNE]);
float seMag = (float) Math.hypot
        (xGradient[indexSE], yGradient[indexSE]);
float swMag = (float) Math.hypot
        (xGradient[indexSW], yGradient[indexSW]);
float nwMag = (float) Math.hypot
        (xGradient[indexNW], yGradient[indexNW]);
float tmp;

if (xGrad * yGrad <= (float) 0
        ? Math.abs(xGrad) >= Math.abs(yGrad)
            ? (tmp = Math.abs(xGrad * gradMag)) >= Math.abs
            (yGrad * neMag - (xGrad + yGrad) * eMag)
                && tmp > Math.abs(yGrad * swMag -
                (xGrad + yGrad) * wMag)
            : (tmp = Math.abs(yGrad * gradMag)) >=
            Math.abs(xGrad * neMag - (yGrad + xGrad) * nMag)
                && tmp > Math.abs(xGrad * swMag -
                (yGrad + xGrad) * sMag)
        : Math.abs(xGrad) >= Math.abs(yGrad)
            ? (tmp = Math.abs(xGrad * gradMag)) >=
            Math.abs(yGrad * seMag + (xGrad - yGrad) * eMag)
                && tmp > Math.abs(yGrad * nwMag +
                (xGrad - yGrad) * wMag)
            : (tmp = Math.abs(yGrad * gradMag)) >=
            Math.abs(xGrad * seMag +
            (yGrad - xGrad) * sMag)
                && tmp > Math.abs(xGrad * nwMag +
                (yGrad - xGrad) * nMag)
        )
    magnitude[index] = gradMag >= 1000F ? 100000 :
        (int) (100F * gradMag);
else
```

```
            magnitude[index] = 0;
        }
    }
}

/*
 * calculate the gaussian operator
 */
private float gaussian(float x, float sigma)
{
    return (float) Math.exp((-x * x) / (2f * sigma * sigma));
}

/*
 * search the edge in the image by using the gradients
 * information stored in magnitude array.
 */
private void edgeDetect(int low, int high)
{
    //initialize all the elements in edgeData array to 0.
    Arrays.fill(edgeData, 0);
    int offset = 0;
    for (int x = 0; x < sourceImageWidth; x++)
    {
        for (int y = 0; y < sourceImageHeight; y++)
        {
            //when find a pixel with gradient bigger than
            //high threshold, set it as a start point of an edge,
            //and begin to search the other part of the edge until
            //reach the end point.
            if (edgeData[offset] == 0 && magnitude[offset] >= high)
            {
                edgeSearch(x, y, offset, low);
```

```
        }
        offset++;
      }
    }
}

/*
 * recursively call the edgeSearch function to search
 * the edge until fount a point with gradient smaller
 * than the low threshold.
 */
private void edgeSearch(int x1, int y1, int i1, int threshold)
{
    int x0 = x1 == 0 ? x1 : x1 - 1;
    int x2 = x1 == sourceImageWidth - 1 ? x1 : x1 + 1;
    int y0 = y1 == 0 ? y1 : y1 - 1;
    int y2 = y1 == sourceImageHeight -1 ? y1 : y1 + 1;

    edgeData[i1] = magnitude[i1];
    for (int x = x0; x <= x2; x++)
    {
      for (int y = y0; y <= y2; y++)
      {
        int i2 = x + y * sourceImageWidth;
        if ((y != y1 || x != x1)&& edgeData[i2] == 0
              && magnitude[i2] >= threshold)
        {
          edgeSearch(x, y, i2, threshold);
          return;
        }
      }
    }
}
```

```
    /*
     * set the edges pixel to white, background to black
     */
    private void thresholdEdges()
    {
        for (int i = 0; i < ImageSize; i++)
        {
            edgeData[i] = edgeData[i] > 0 ? 0xffffffff : 0xff000000;
        }
    }

    /*
     * store the edges information into a BufferedImage edgesImage.
     */
    private void writeEdges(int pixels[])
    {
        if (edgesImage == null)
        {
            edgesImage = new BufferedImage(sourceImageWidth, sourceImageHeight,
                    BufferedImage.TYPE_INT_ARGB);
        }
        edgesImage.getWritableTile(0, 0).setDataElements(0, 0, sourceImageWidth,
                sourceImageHeight, pixels);
    }
}
```

3. PlateIsolation.java

```
package numberplaterecognition;
import java.awt.image.BufferedImage;

/**
 * This class contains the plate isolation algorithm when create a new
 * PlateIsolation, first specifiy a bufferedImage type source image by
 * calling setSourceImage(), then call isolation() function to perform
 * isolation algprithm to the source image, finally, use getPlateImage()
 * function to get the result image.
 */
public class PlateIsolation {
   int width;
   int height;
   int slideWindowWidth = 170;
   int slideWindowHeight = 50;
   int [] slideWindow = new int[slideWindowWidth*slideWindowHeight];
   int [] data;
   int max, maxW, maxH;

   BufferedImage sourceImage;
   BufferedImage plateImage;

   /**
    * set the source image
    */
   public void setSourceImage(BufferedImage image)
   {
      sourceImage = image;
   }
```

```
/**
 * get the isolated plate image after called the isolation() function
 */
public BufferedImage getPlateImage()
{
   return plateImage;
}

/**
 * move the silidewindow around in the edge detected image, count the number
 * of white pixels in each pozition to find the area has the largest number
 * of white pixels which means this area contains number plate.
 * set the source image to be used to detect edges
 */
public void isolation(boolean tiltCorrection)
{
   max = 0;
   maxW = 0;
   maxH = 0;
   width = sourceImage.getWidth();
   height = sourceImage.getHeight();
   data = (int[]) sourceImage.getData().getDataElements(0, 0, width, height, null);

   int m, n, q, p, currentIndex, whitePixels = 0,  i=0;

   for (n=0; n+slideWindowHeight <= height; n = n+2)
   {
      for(m=0; m+slideWindowWidth <= width; m = m+5)
      {
         //count white pixels in current pozition
         for(p=0; p < slideWindowHeight; p++)
         {
            for(q=0; q < slideWindowWidth; q++ )
```

```
        {
            currentIndex = width*(n+p)+(m+q);
            int red   = (data[currentIndex] >> 16) & 0xff;
            int green = (data[currentIndex] >>  8) & 0xff;
            int blue  = (data[currentIndex]      ) & 0xff;

            if(red == 255 && green == 255 && blue == 255)
            {
               whitePixels++;
            }
        }
    }
    if(max <= whitePixels)
    {
        max = whitePixels;
        maxH = n;
        maxW = m;
    }
    whitePixels = 0;
  }
}

//copy the number plate area into a new array called slideWindow
for(n = maxH; n < maxH+slideWindowHeight; n++)
{
  for(m = maxW; m < maxW+slideWindowWidth; m++)
  {
    slideWindow[i] = data[width*n+m];
    i++;
  }
}

//create a bufferedImage use the pixels information stores in array
```

```
//slideWindow, display it in localizationImagePanel
plateImage = new BufferedImage(slideWindowWidth, slideWindowHeight,
      BufferedImage.TYPE_INT_ARGB);
plateImage.getWritableTile(0, 0).setDataElements(0, 0,
      slideWindowWidth, slideWindowHeight, slideWindow);

//copy the number plate area from the grayscaled image into an array
i = 0;
byte [] numberPlateGray = new byte
      [slideWindowHeight * slideWindowWidth];

for(n = maxH; n < maxH+slideWindowHeight; n++)
{
   for(m = maxW; m < maxW+slideWindowWidth; m++)
   {
      numberPlateGray[i] =
            (byte)(im.grayScaledData[width*n+m] & 0xFF);
      i++;
   }
}

//store the grayscaled number plate area into a BufferedImage
BufferedImage grayscaleImage = new BufferedImage(slideWindowWidth,
      slideWindowHeight, BufferedImage.TYPE_BYTE_GRAY);
grayscaleImage.getRaster().setDataElements(0, 0, slideWindowWidth,
      slideWindowHeight, numberPlateGray);

//convert the grayscaled number plate image into binarization format.
OTSUBinarization binarization = new OTSUBinarization();
binarization.setSourceImage(grayscaleImage);
binarization.OTSU();
im.binaplateim = binarization.getBinaryImage();
```

```
//if tiltCorrection is true, currect the tilt
if(tiltCorrection == true)
{
   TiltCorrection correctionim = new TiltCorrection();
   correctionim.setSourceImage(plateImage);
   correctionim.correction(0);
   plateImage = correctionim.getCorrectedImage();
   double angle = correctionim.getRotatedAngle();
   if(angle != 0)
   {
      correctionim.setSourceImage(im.binaplateim);
      correctionim.correction(angle);
      im.binaplateim = correctionim.getCorrectedImage();
      im.plateim = im.binaplateim;
   }
}

int [] pixel = (int[]) im.binaplateim.getData().getDataElements(0, 0,
      slideWindowWidth, slideWindowHeight, null);

//resize the number plate area
int [] horCount = new int [slideWindowHeight];
int change = 0;

for(n = 0; n < slideWindowHeight; n++)
{
   for(m = 0; m < slideWindowWidth-1; m++)
   {
      if(slideWindow[n*slideWindowWidth+m]!=
            slideWindow[n*slideWindowWidth+m+1])
      {
         change++;
      }
```

```
    }
    if (change > 20)
        horCount[n] = 1;
    else
        horCount[n] = 0;
    change = 0;
}

//find the top line of the number plate
int top = 0;

for(n = 1; n < slideWindowHeight; n++)
{
    if(horCount[n-1] != horCount[n])
    {
        top = n-1;
        break;
    }
}

//find the botton line of the number plate
int bottom = 0;

for(n = slideWindowHeight-1; n > 0; n--)
{
    if(horCount[n-1] != horCount[n])
    {
        bottom = n;
        break;
    }
}

//copy the resized number plate area into a new array.
```

```
    int [] resizedData = new int [slideWindowWidth*(bottom - top)];

    for(m = 0; m  < bottom - top; m++)
    {
       for(n = 0; n < slideWindowWidth; n++)
       {
          resizedData[m*slideWindowWidth+n] = pixel[(top+m)
                *slideWindowWidth+n];
       }
    }

    int newHeight = bottom-top;

    if(newHeight <= 5)
    {
       newHeight = slideWindowHeight;
       resizedData = slideWindow;
    }

    BufferedImage resizedImage = new BufferedImage(slideWindowWidth,
          newHeight, BufferedImage.TYPE_INT_ARGB);
    resizedImage.getRaster().setDataElements(0, 0, slideWindowWidth,
          newHeight, resizedData);

    im.binaplateim = resizedImage;

    //store the location of the number plate into the plateLocation array.
    im.plateLocation[0] = maxH+top;
    im.plateLocation[1] = maxH+bottom;
    im.plateLocation[2] = maxW;
    im.plateLocation[3] = maxW+170;
  }
}
```

4. CharacterSegmentation.java

```
package numberplaterecognition;
import java.awt.image.BufferedImage;

/**
 * This class contains the character segmentation algorithm,
 * when create a new CharacterSegmentation, first specifiy a
 * bufferedImage type source image by calling setSourceImage(),
 * then call segmentation() function to perform segmentation
 * algprithm to the source image, finally, use getSegmentedChar()
 * function to get the bufferedImage array which stores the segmented
 * character image..
 */
public class CharacterSegmentation {

   int sourceImageHeight;
   int sourceImageWidth;
   int charPos [] = new int [30];
   int[] pixels;
   int [] verCount;

   BufferedImage [] characters = new BufferedImage [15];
   BufferedImage sourceImage;
   BufferedImage binarizationImage;
   BufferedImage resizedPlateImage;


   /**
    * set the source image to be used for segmentation
    */
   public void setSourceImage(BufferedImage image)
```

```
{
   sourceImage = image;
}

/**
 * get the segmentation BufferedImage array after
 * called the segmentation() function
 */
public BufferedImage [] getSegmentedChar()
{
   return characters;
}

/**
 * perform segmentation algorithm to the source iamge
 * to segmente the character.
 */
public void segmentation()
{
   int n;
   int m;
   int i;
   int len = 0;
   int count = 0;

   sourceImageWidth = sourceImage.getWidth();
   sourceImageHeight = sourceImage.getHeight();
   verCount = new int [sourceImageWidth];
   pixels = new int [sourceImageWidth*sourceImageHeight];
   pixels = (int[]) sourceImage.getData().getDataElements(0, 0,
         sourceImageWidth, sourceImageHeight, null);

   //do virtical projection to segment the character
```

```
for(n = 0; n < sourceImageWidth; n++)
{
   //count the nunber of white pixels in each
   //column from top to bottom
   for(m = 0; m < sourceImageHeight; m++ )
   {
      if(pixels[m*sourceImageWidth + n] == -1)
         count++;
   }
   verCount[n] = count;
   count = 0;
}

//analysis the vertical projection information, find the start and end
//position of each character, store the position information into
//array charPos
int b = 0;  //stores the beginning position of a character
int e = 0;  //stores the ending position of a character
n = 0;
while(b < verCount.length)
{
   if(verCount[b] < 2)
      b++;
   else
   {
      e = b;
      while(e != verCount.length-1 && verCount[e+1] >= 2)
         e++;
      len = e - b +1;
      if(len >= 2)
      {
         charPos[n] = b;
         charPos[n+1] = e;
```

```
            n = n+2;
        }
        b = e+1;
    }
}

//if the first two characters is fake, delete its position
for(n = 2; n > 0; n--)
{
    if(charPos[n*2] - charPos[n*2-1] > 6)
    {
        for(m = 0; m < charPos.length-n*2; m++)
            charPos[m] = charPos[n*2+m];
        for(; m < charPos.length; m++)
            charPos[m] = 0;
        break;
    }
}

//if the last character is fake, delete its position
i= charPos.length-1;
while(charPos[i] == 0)
    i--;

if(charPos[i-1] - charPos[i-2] > 5)
{
    charPos[i] = 0;
    charPos[i-1] = 0;
}
if(charPos[i-3] - charPos[i-4] > 5)
{
    charPos[i-2] = 0;
    charPos[i-3] = 0;
```

```
}

for(n = 0; n < 15; n++)
{
   len = charPos[n*2+1]-charPos[n*2];
   int [][]sigChar = new int [sourceImageHeight][len];
   int [][]resizedSigChar;
   int []sig;
   int r, l;
   int top = -1, bottom = sourceImageHeight;



   if(len > 1)
   {
      //copy the single character from resized
      //number plate into array.
      for(r = 0; r < sourceImageHeight; r++)
      {
         count = 0;
         for(l=0; l<len; l++)
         {
            //Array sigChar is used to character racognition.
            sigChar[r][l] =
                  pixels[r*sourceImageWidth+charPos[n*2]+l];


            if(r <= 2 || r >= sourceImageHeight - 2)
            {
               if(sigChar[r][l] == -1)
                  count++;
            }
         }

         //redefine the top and bottom boundary for current character
```

```
    if(r <= 2)
    {
        //decide the top boundary for the character.
        if(count <= 3 && top < r)
            top = r;
    }
    if(r >= sourceImageHeight - 2)
    {
        //decide the bottom boundary for the character.
        if(count <= 1 && bottom > r)
            bottom = r;
    }
}
top = top +1;
bottom = bottom -1;
int charH = bottom - top +1;
sig = new int [(charH)*len];

//copy the resized single charcter into a new array.
resizedSigChar = new int [charH][len];
for (r = 0 ; r< charH; r++)
{
    for(l = 0; l < len; l++)
    {
        resizedSigChar[r][l] = sigChar[top + r][l];

        //Array sig is used to display the image onto screen.
        sig[r*len+l] = resizedSigChar[r][l];
    }
}

//display the sigmented character onto recognitionPanel
BufferedImage sigCharacter = new BufferedImage
```

```
                (len, charH, BufferedImage.TYPE_INT_ARGB);
            sigCharacter.getWritableTile(0, 0).setDataElements
                (0, 0, len, charH, sig);

            //store the single character image into array
            characters[n] = sigCharacter;
          }
        }
    }
}
```

5. CharacterRecognition.java

```
package numberplaterecognition;
import java.awt.Point;
import java.awt.image.BufferedImage;

/**
 * This class contains the character recognition algorithm,
 * when create a new CharacterRecognition, the first thing need
 * to do is specifiy two bufferedImage type source images:
 * binary image and thinning image by calling setSourceImage().
 * Both of these two image should contains the same single character.
 * then call characterReg() function to perform character recognition
 * algprithm to the source images and return the result.
 */
public class CharacterRecognition {

    int binaSigchar[][];
    int thinSigchar[][];
    int binaPixels[];
    int thinPixels[];
    Point endPointsPos[];
    Point pos;

    BufferedImage sourceBinaImage;
    BufferedImage sourceThinImage;

    int charHeight;
    int charWidth;
    int thinCharHeight;
    int thinCharWidth;
```

```
int letterLen = 1;
char preLetter = '$';


/**
 * two source image need to set: binary image and thinning image,
 * which should contains the same single character
 */
public void setSourceImage(BufferedImage bimage, BufferedImage timage)
{
   sourceBinaImage = bimage;
   sourceThinImage = timage;
}

/**
 * set the number of letters in the number plate,
 * used only when recognize character. Initial value is 1,
 * because there is at least one letter in a number plate
 */
public void setLetterLen(int len)
{
   letterLen = len;
}

/**
 * if there are two letters in number plate, set the
 * first letter befor recognize the second letter. Used
 * only when there is two letters in number plate
 */
public void setPreLetter(char pre)
{
   preLetter = pre;
}
```

```
/**
 * perform the character recognition algorithm to the source image
 * the return value is the result.
 */
public char characterReg(char type)
{
   charHeight = sourceBinaImage.getHeight();
   charWidth = sourceBinaImage.getWidth();
   thinCharHeight = sourceThinImage.getHeight();
   thinCharWidth = sourceThinImage.getWidth();

   binaPixels = (int[]) sourceBinaImage.getData().getDataElements
         (0, 0, charWidth, charHeight, null);
   thinPixels = (int[]) sourceThinImage.getData().getDataElements
         (0, 0, thinCharWidth, thinCharHeight, null);
   binaSigchar = new int [charHeight][charWidth];
   thinSigchar = new int [thinCharHeight][thinCharWidth];
   endPointsPos = new Point [15];

   //convert array binaPixels into 2-dementional array
   int w, h;
   for(h = 0; h < charHeight; h++)
   {
      for(w = 0; w < charWidth; w++)
      {
         binaSigchar[h][w] = binaPixels[h*charWidth +w];
      }
   }

   //convert array thinPixels into 2-dementional array
   for(h = 0; h < thinCharHeight; h++)
   {
```

```
        for(w = 0; w < thinCharWidth; w++)
        {
            thinSigchar[h][w] = thinPixels[h*thinCharWidth +w];
        }
    }

    //find the end point information by analysis the thinning image
    //this infoamtion will be used to identify character.
    findEndPoint();

    //call recognition functions
    if(type == 'n')     //recognize it as a number
        return numberReg();
    else if(type == 'l')    //recognize it as a letter
    {
        //if it is the second letter, set the previews letter
        if(letterLen == 2)
            return letterReg(preLetter);
        else
            return letterReg();
    }
    else
        return '$';
}

/**
 * used when only have one letter on number plate, in this situation,
 * the result could only be C, D, G, L and W
 */
public char letterReg()
{
    if(charWidth >8)
    {
```

```
      if(charWidth > 8)
      {
         //if no end point, the result could only be D
         if(endPointsPos[5].x == 0 ||endPointsPos[5].x == 1)
            return 'D';
         //if the number of end point is greater
         //or equal then 4, the result could only be W
         if(endPointsPos[5].x >= 4)
            return 'W';
         if(checkEmptyPoint(binaSigchar, new Point(1,charWidth/2+1))
               ==true)
            return 'L';
         else
         {
            //bottom middle point is filled
            if(endPointsPos[1].x > thinCharHeight/2-3 &&
                  endPointsPos[1].x < thinCharHeight/2+3)
               return 'G';
            else
               return 'C';
         }
      }
   }
   return '$';
}

/**
 * if there are two characters in number plate, the first one
 * would only be D, C, R, L, S, T, K, W, M, and the second letter
 * could be identyfied by the first letter.
 */
public char letterReg(char pre)
{
```

```
//Previews letter is $, means this letter is the first
//letter in the two letters, so the result could only
//be C, D, K, L, M, O, R, S, T and W
if(pre == '$')
{
   if(endPointsPos[5].x == 0)
      return 'D';
   else if(endPointsPos[5].x == 2)
   {
      if(endPointsPos[0].y > thinCharWidth/2
            && endPointsPos[1].y > thinCharWidth/2)
         return 'C';
      else if(endPointsPos[0].x > thinCharHeight/2
            && endPointsPos[1].x > thinCharHeight/2)
         return 'R';
      else if(endPointsPos[0].y < endPointsPos[1].y)
         return 'L';
      else if(endPointsPos[0].y > endPointsPos[1].y)
         return 'S';
      else
         return 'x';
   }
   else if(endPointsPos[5].x == 3)
      return 'T';
   else if(endPointsPos[5].x == 4)
   {
      if(checkEmptyPoint(binaSigchar,
            new Point(charHeight/2,charWidth-2)) == true)
         return 'K';
      else if(checkEmptyPoint(binaSigchar,
            new Point(1,charWidth/2)) == true)
         return 'W';
      else
```

```
            return 'M';
        }
        else if(endPointsPos[5].x == 5)
        {
            if(endPointsPos[2].x < thinCharHeight/2)
                return 'W';
            else
                return 'M';
        }
    }

    if(pre == 'C')
    {
        if(checkEmptyPoint(binaSigchar,
                new Point(charHeight*25/100,charWidth-2)) == true)
            return 'E';
        else if(endPointsPos[5].x == 5)
            return 'W';
        else if(checkEmptyPoint(binaSigchar,
                new Point(charHeight-2,charWidth*40/100)) == true)
            return 'N';
        else
            return 'W';
    }

    if(pre == 'D')
    {
        if(endPointsPos[5].x == 2)
            return 'L';
        else if(checkPoint(binaSigchar,
                new Point(charHeight/2,charWidth/2)) == true)
            return 'Y';
        else
```

```
        return 'L';
}

if(pre == 'K')
{
   if(endPointsPos[5].x >= 4 &&
         checkEmptyPoint(binaSigchar,
         new Point(1,charWidth/2-1)) == true)
      return 'K';
   else if(checkPoint(binaSigchar,
         new Point(charHeight-2,1)) == true
         || checkPoint(binaSigchar,
         new Point(charHeight-2,charWidth-2)) == true)
      return 'E';
   else
      return 'Y';

}

if(pre == 'L')
{
   if(endPointsPos[5].x == 0 ||endPointsPos[5].x == 1)
      return 'D';
   if(endPointsPos[5].x == 2 || endPointsPos[5].x == 3)
      return 'S';
   if(checkEmptyPoint(binaSigchar,
         new Point(1,charWidth/2)) == true &&
         checkEmptyPoint(binaSigchar,
         new Point(charHeight-2,charWidth/2)) == true)
   {
      if(checkEmptyPoint(binaSigchar,
            new Point(charHeight/2,charWidth-2)) == true)
         return 'K';
```

```
      else
         return 'H';
   }
   else
      return 'M';
}

if(pre == 'M')
{
   if(endPointsPos[5].x == 0)
      return 'O';
   else if(endPointsPos[5].x == 4)
   {
      if(Math.abs(endPointsPos[2].x - endPointsPos[3].x) < 2)
         return 'H';
      else
         return 'N';
   }
}

if(pre == 'O')
{
   return 'Y';
}

if(pre == 'R')
{
   return 'N';
}

if(pre == 'S')
{
   if(endPointsPos[5].x == 0)
```

```
        return 'O';
    else if(checkEmptyPoint(binaSigchar,
            new Point(1,charWidth/2+1)) == true
            || checkEmptyPoint(binaSigchar,
            new Point(charHeight-2,charWidth/2-1)) == true)
        return 'N';
    else
        return 'O';
}

if(pre == 'T')
{
    if(checkPoint(binaSigchar, new Point(1,charWidth/2)) == true
            || checkPoint(binaSigchar,
            new Point(charHeight-2,charWidth/2)) == true)
        return 'S';
    else
        return 'N';
}

if(pre == 'W')
{
    if(endPointsPos[5].x == 0)
        return 'D';

    if(endPointsPos[5].x == 5 && endPointsPos[2].x < thinCharHeight/2)
        return 'W';
    if(checkEmptyPoint(binaSigchar,
            new Point(1,charWidth/2)) == true &&
            checkEmptyPoint(binaSigchar,
            new Point(charHeight-2,charWidth/2)) == true)
    {
        if(checkEmptyPoint(binaSigchar,
```

```
                new Point(charHeight/2,charWidth-2)) == true
                || checkEmptyPoint(binaSigchar,
                new Point(charHeight/2,1)) == true)
            return 'X';
          else
            return 'H';
      }
      else
        return 'W';
   }
   return 'x';
}

/**
 * function used to identify a number
 */
public char numberReg()
{

   int count = 0;
   for(int l = 0; l < thinCharWidth; l++)
   {
      for(int h =0; h < thinCharHeight; h++)
      {
         if(thinSigchar[h][l] == -1)
            count++;
      }
   }
   if(count < 10)
      return '-';

   if(charWidth >= 8)
   {
```

```
if(endPointsPos[5].x == 0)
{
   if(checkPoint(binaSigchar,
         new Point(charHeight/2, charWidth/2)) == true)
      return '8';
   else
      return '0';
}

if(endPointsPos[5].x == 2 && endPointsPos[0].y < endPointsPos[1].y
      && endPointsPos[1].y < thinCharWidth/2+1
      && checkEmptyPoint(binaSigchar,
      new Point(charHeight*75/100,charWidth-2)) == true)
   return '7';
//check top middle point is filled
//and bottom middle point is filled
if(checkPoint(binaSigchar, new Point(1,charWidth/2))==true &&
      checkEmptyPoint(binaSigchar,
      new Point(charHeight-2,charWidth/2))==false)
{
   //check top left point is filled
   if(checkPoint(binaSigchar,
         new Point(charHeight*38/100,1))==true)
   {
      //check the top right point is empty
      if(checkEmptyPoint(binaSigchar,
            new Point(charHeight*25/100,charWidth-2)) == true)
      {
         //check the lower left point is empty
         if(checkEmptyPoint(binaSigchar,
               new Point(charHeight*69/100,1)) == true)
         {
            if(endPointsPos[5].x == 1 &&
```

```
            endPointsPos[0].x > thinCharHeight/2)
          return '9';
        else
          return '5';
      }
      else
      {
        if(endPointsPos[5].x >= 3)
          return '5';
        else
          return '6';
      }
    }
    else
    {
      //check the lower left point
      if(checkEmptyPoint(binaSigchar,
          new Point(charHeight*73/100,1)) == true)
      {
        if(checkEmptyPoint(binaSigchar,
            new Point(charHeight*73/100,charWidth-2))
            == true)
        {
          if(checkEmptyPoint(binaSigchar,
              new Point(charHeight-2,charWidth-2))
              == true)
            return '7';
          else
          {
            if(endPointsPos[5].x == 1)
              return '9';
            else
              return '2';
```

```
            }
          }
          else
            return '9';
        }
        else
        {
          //check the middle point
          if(checkPoint(binaSigchar,
              new Point(charHeight/2, charWidth/2))
              == true)
          {
            if(checkEmptyPoint(binaSigchar,
                new Point(charHeight*64/100,1)) ==
                true)
              return '9';
            else
            {
              if(checkEmptyPoint(binaSigchar,
                  new Point(charHeight*35/100,
                  charWidth-2)) == true)
                return '6';
              else
                return '8';
            }
          }
          else
            return '0';
        }
      }
    }
    else
    {
```

```
        if(checkEmptyPoint(binaSigchar, new Point
            (charHeight*75/100,charWidth-2)) == true)
        {
          if(checkPoint(binaSigchar, new Point
              (charHeight-2, charWidth-2)) == true)
          {
            if(endPointsPos[endPointsPos[5].x-1].y >
                thinCharWidth/2 ||
                endPointsPos[endPointsPos[5].x-1-1].y
                > thinCharWidth/2)
              return '2';
            else
              return '3';
          }
          else
            return '7';
        }
        else
        {
          if(checkEmptyPoint(binaSigchar, new
              Point(charHeight*85/100,1)) == true)
            return '1';
          else
            return '3';
        }
      }
    }
    else
    {
      if(checkEmptyPoint(binaSigchar,
          new Point(1,charWidth/2-2))==true &&
      checkEmptyPoint(binaSigchar,
          new Point(charHeight-2,charWidth/2-2))==true)
```

```
        {
            if(checkPoint(binaSigchar,
                    new Point(charHeight*58/100,1))==true)
                return '4';
            else
                return '1';
        }
        else
        {
            if(checkPoint(binaSigchar,
                    new Point(charHeight/2, 1)) == true)
            {
                if(checkPoint(binaSigchar,
                        new Point(charHeight/2, charWidth/2)) == true)
                    return '8';
                else
                    return '0';
            }
            else
                return '7';
        }
    }
}
else if (charWidth > 2)
{
    if(endPointsPos[5].x == 2 && endPointsPos[0].y < endPointsPos[1].y
            && endPointsPos[1].y < thinCharWidth/2+1)
        return '7';
    //check top middle point is filled
    //and bottom right point is filled
    if(checkPoint(binaSigchar, new Point(1,charWidth/2))==true &&
            checkPoint(binaSigchar, new Point
            (charHeight-2,charWidth-2))==true)
```

```
        return '1';
    else
    {
        count = 0;
        for(int l = 0; l < thinCharWidth; l++)
        {
            for(int h =0; h < thinCharHeight; h++)
            {
                if(thinSigchar[h][l] == -1)
                    count++;
            }
        }
        if(count > 8)
            return '1';
        else
            return '-';
    }
}
else
{
    count = 0;
    for(int l = 0; l < charWidth; l++)
    {
        for(int h =0; h < charHeight; h++)
        {
            if(binaSigchar[h][l] == -1)
                count++;
        }
    }
    if(count > charHeight*charWidth*70/100)
        return '1';
    else
        return '-';
```

```
    }
}

/**
 * use a 3X3 mask to check if the current point is belongs to
 * a part of the character.
 * count the number of white pixels in the mask if there are
 * equal or more than 5 white pixels, current  point
 * is belongs to a character,
 */
public boolean checkPoint(int sigchar[][], Point pos)
{
    int count = 0;
    if(sigchar[pos.x-1][pos.y-1] == -1)
        count++;
    if(sigchar[pos.x-1][pos.y] == -1)
        count++;
    if(sigchar[pos.x-1][pos.y+1] == -1)
        count++;
    if(sigchar[pos.x][pos.y-1] == -1)
        count++;
    if(sigchar[pos.x][pos.y] == -1)
        count++;
    if(sigchar[pos.x][pos.y+1] == -1)
        count++;
    if(sigchar[pos.x+1][pos.y-1] == -1)
        count++;
    if(sigchar[pos.x+1][pos.y] == -1)
        count++;
    if(sigchar[pos.x+1][pos.y+1] == -1)
        count++;

    if(count >= 5)
```

```
        return true;
    else
        return false;
}

/**
 * opposite of the function checkPoint(), this function is
 * check if current point is am empty point which means it
 * doesn't belong to a character
 */
public boolean checkEmptyPoint(int sigchar[][], Point pos)
{
    int count = 0;
    if(sigchar[pos.x-1][pos.y-1] != -1)
        count++;
    if(sigchar[pos.x-1][pos.y] != -1)
        count++;
    if(sigchar[pos.x-1][pos.y+1] != -1)
        count++;
    if(sigchar[pos.x][pos.y-1] != -1)
        count++;
    if(sigchar[pos.x][pos.y] != -1)
        count++;
    if(sigchar[pos.x][pos.y+1] != -1)
        count++;
    if(sigchar[pos.x+1][pos.y-1] != -1)
        count++;
    if(sigchar[pos.x+1][pos.y] != -1)
        count++;
    if(sigchar[pos.x+1][pos.y+1] != -1)
        count++;

    if(count >= 5)
```

```
        return true;
    else
        return false;
}

/**
 * find the end points and store their positions into
 * an array called endPontsPos. endPointsPos[5] store the
 * number of end points in the thinning image.
 */
public void findEndPoint()
{
    int count = 0;
    int n = 0;
    int h, w;
    endPointsPos[5] = new Point (0,0);
    for(h  = 1; h < thinCharHeight-1; h++)
    {
        for(w = 1; w < thinCharWidth-1; w++)
        {
            //if the current point is white,check its eight neighbors
            //and count the white point
            if(thinSigchar[h][w] == -1)
            {
                count = 0;
                if(thinSigchar[h-1][w-1] == -1)
                    count++;
                if(thinSigchar[h-1][w] == -1)
                    count++;
                if(thinSigchar[h-1][w+1] == -1)
                    count++;
                if(thinSigchar[h][w-1] == -1)
                    count++;
```

```
            if(thinSigchar[h][w+1] == -1)
               count++;
            if(thinSigchar[h+1][w-1] == -1)
               count++;
            if(thinSigchar[h+1][w] == -1)
               count++;
            if(thinSigchar[h+1][w+1] == -1)
               count++;

            //if there is only one white point around current point,
            //the current point is an end point, stores its position.
            //stores the number of end point into the last position
            //of the array
            if(count == 1)
            {
               endPointsPos[n] = new Point(h, w);
               endPointsPos[5] = new Point(n+1, n+1);
               n++;
            }
          }
        }
      }
    }
}
```

6. ImageThinning.java

```
package numberplaterecognition;
import java.awt.image.BufferedImage;

/**
 * This class contains the image thinning algorithm based on an
 * index table, when create a new ImageThinning, first specifiy a
 * bufferedImage type source image by calling setSourceImage(),
 * then call thinning() function to perform thinning algprithm
 * to the source image, finally, use getThinningImage() function
 * to get the result image.
 */
public class ImageThinning {

   BufferedImage sourceImage;
   BufferedImage thinningImage;

   int sourceImageHeight;
   int sourceImageWidth;

   int [] checkTable =
   {
      0,0,1,1,0,0,1,1,
      1,1,0,1,1,1,0,1,
      1,1,0,0,1,1,1,1,
      0,0,0,0,0,0,0,1,

      0,0,1,1,0,0,1,1,
      1,1,0,1,1,1,0,1,
      1,1,0,0,1,1,1,1,
      0,0,0,0,0,0,0,1,
```

```
    1,1,0,0,1,1,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,

    1,1,0,0,1,1,0,0,
    1,1,0,1,1,1,0,1,
    0,0,0,0,0,0,0,0,
    0,0,0,0,0,0,0,0,

    0,0,1,1,0,0,1,1,
    1,1,0,1,1,1,0,1,
    1,1,0,0,1,1,1,1,
    0,0,0,0,0,0,0,1,

    0,0,1,1,0,0,1,1,
    1,1,0,1,1,1,0,1,
    1,1,0,0,1,1,1,1,
    0,0,0,0,0,0,0,0,

    1,1,0,0,1,1,0,0,
    0,0,0,0,0,0,0,0,
    1,1,0,0,1,1,1,1,
    0,0,0,0,0,0,0,0,

    1,1,0,0,1,1,0,0,
    1,1,0,1,1,1,0,0,
    1,1,0,0,1,1,1,0,
    1,1,0,0,1,0,0,0
};

/**
```

```
 * set the source image to be used to thinning
 */
public void setSourceImage(BufferedImage image)
{
   sourceImage = image;
}

/**
 * get the thinning image after called the thiing() function
 */
public BufferedImage getThinningImage()
{
   return thinningImage;
}

/**
 * perform thinning algorithm to the souece image.
 */
public void thinning()
{
   boolean changed = true;
   int x;
   int y;
   int i = 3;
   int n,s,w,e,nw,ne,sw,se;
   int check;
   sourceImageWidth = sourceImage.getWidth();
   sourceImageHeight = sourceImage.getHeight();

   int [] pixel = (int[]) sourceImage.getData().getDataElements
         (0, 0, sourceImageWidth, sourceImageHeight, null);
   int [] imageData = new int [(sourceImageWidth+2)*(sourceImageHeight+2)];
   int [] pixels = new int [(sourceImageWidth+2)*(sourceImageHeight+2)];
```

```
//initialize array
for(x = 0; x < (sourceImageWidth+2)*(sourceImageHeight+2); x++)
{
   pixels[x] = 0xff000000;
}

for(x = 0; x < sourceImageHeight; x++)
{
   for(y = 0; y < sourceImageWidth; y++)
   {
      pixels[sourceImageWidth*(x+1)+i+y] =
            pixel[sourceImageWidth*x+y];
   }
   i = i+2;
}

for(x = 0; x < (sourceImageWidth+2)*(sourceImageHeight+2); x++)
{
   if(pixels[x] == 0xffffffff)
      imageData[x] = 0; //0 is frontground, 1 is background
   else
      imageData[x] = 1;
}

sourceImageWidth = sourceImageWidth+2;
sourceImageHeight = sourceImageHeight+2;

// nw    n    ne
// w  current  e
// sw    s    se
int count = 0;
```

```
while(changed)
{
   changed = false;
   count++;
   //perform thinning in virtical direction
   for(y = 1; y < sourceImageHeight-1; y++)
   {
      for(x = 1; x < sourceImageWidth-1; x++)
      {
         //if current pixel is white
         if(imageData[sourceImageWidth*y+x] == 0)
         {
            w = imageData[sourceImageWidth*y+x-1];
            e = imageData[sourceImageWidth*y+x+1];
            if(w==1 || e==1)
            {
               n  = imageData[sourceImageWidth*(y-1)+x];
               nw = imageData[sourceImageWidth*(y-1)+x-1];
               ne = imageData[sourceImageWidth*(y-1)+x+1];

               s  = imageData[sourceImageWidth*(y+1)+x];
               sw = imageData[sourceImageWidth*(y+1)+x-1];
               se = imageData[sourceImageWidth*(y+1)+x+1];

               check = nw+n*2+ne*4+w*8+e*16+sw*32+s*64+se*128;

               //if current pixel can be deleted,
               //set it into black
               if(checkTable[check] == 1)
               {
                  pixels[sourceImageWidth*y+x] = 0xff000000;
                  imageData[sourceImageWidth*y+x] = 1;
                  x++; //if delete current pixel, skip next pixel;
```

```
                changed = true;
            }
        }
    }
}

//perform thinning in horizontal direction
for(x = 1; x < sourceImageWidth-1; x++)
{
   for(y = 1; y < sourceImageHeight-1; y++)
   {
      //if current pixel is white
      if(imageData[sourceImageWidth*y+x] == 0)
      {
         n  = imageData[sourceImageWidth*(y-1)+x];
         s  = imageData[sourceImageWidth*(y+1)+x];

         if(n==1 || s==1)
         {
            w = imageData[sourceImageWidth*y+x-1];
            nw = imageData[sourceImageWidth*(y-1)+x-1];
            sw = imageData[sourceImageWidth*(y+1)+x-1];

            e = imageData[sourceImageWidth*y+x+1];
            ne = imageData[sourceImageWidth*(y-1)+x+1];
            se = imageData[sourceImageWidth*(y+1)+x+1];

            check = nw+n*2+ne*4+w*8+e*16+sw*32+s*64+se*128;

            //if current pixel can be deleted,
            //set it into black
            if(checkTable[check] == 1)
```

```
                    {
                        pixels[sourceImageWidth*y+x] = 0xff000000;
                        imageData[sourceImageWidth*y+x] = 1;
                        y++; //if delete current pixel, skip next pixel;
                        changed = true;
                    }
                }
            }
        }
    }

    thinningImage = new BufferedImage(sourceImageWidth,
        sourceImageHeight, BufferedImage.TYPE_INT_ARGB);
    thinningImage.getWritableTile(0, 0).setDataElements
        (0, 0, sourceImageWidth, sourceImageHeight, pixels);
  }
}
```

7. OTSUBinarization.java

```
package numberplaterecognition;
import java.awt.image.BufferedImage;

/**
 * This class contains the image OTSU Binarization algorithm
 * when create a new OTSUBinarization, first specifiy a
 * bufferedImage type source image by calling setSourceImage(),
 * then call OTSU() function to perform thinning algprithm
 * to the source image, finally, use getBinaryImage() function
 * to get the result image.
 */
public class OTSUBinarization {

   int t = 0;
   float wf;
   float uf;
   float wb;
   float ub;
   float u;
   float g;
   float temp;

   BufferedImage sourceImage;
   BufferedImage binaryImage;

   int[] grayData;
   int sourceImageHeight;
   int sourceImageWidth;

   /**
```

```
 * set the source image to be used for binarization
 */
public void setSourceImage(BufferedImage image)
{
   sourceImage = image;
}

/**
 * get the binarization image after called the OTSU() function
 */
public BufferedImage getBinaryImage()
{
   return binaryImage;
}

/**
 * perform OTSU binarization algorithm to souece image.
 */
public void OTSU()
{
   int i;
   int m;

   sourceImageWidth = sourceImage.getWidth();
   sourceImageHeight = sourceImage.getHeight();
   grayData = new int [sourceImageWidth*sourceImageHeight];
   byte [] pixels = (byte[]) sourceImage.getData().getDataElements
         (0, 0, sourceImageWidth, sourceImageHeight, null);

   //initialize array
   for (i = 0; i < sourceImageWidth*sourceImageHeight; i++)
   {
      grayData[i] = (int)(pixels[i] & 0xFF);
```

```
}

//find the best threshold value
g = 0;
for(i = 1; i<= 255; i++)
{
   wb = 0;
   wf = 0;
   ub = 0;
   uf = 0;

   for(m = 0; m < sourceImageWidth*sourceImageHeight; m++)
   {
      if(grayData[m] > i)
      {
         wb++;
         ub = ub + grayData[m];
      }
      else
      {
         wf++;
         uf = uf + grayData[m];
      }
   }

   ub = ub/wb;
   uf = uf/wf;

   wb = wb/(sourceImageWidth*sourceImageHeight);
   wf = wf/(sourceImageWidth*sourceImageHeight);

   u = wf*uf + wb*ub;
```

```
            temp = wb*((ub - u)*(ub - u)) + wf*((uf - u)*(uf - u));
            if(g < temp)
            {
                g = temp;
                t = i;
            }
        }

        //threshold the grayscale humber plate area by
        //using the thresholding value obtained from the
        //OTSU alglgorhtim, store the data into array
        //called resisedData
        for(m = 0; m < sourceImageWidth*sourceImageHeight; m++)
        {
            if(grayData[m] > t)
                grayData[m] = 0xff000000;
            else
                grayData[m] = 0xffffffff;
        }

        binaryImage = new BufferedImage(sourceImageWidth, sourceImageHeight,
                BufferedImage.TYPE_INT_ARGB);
        binaryImage.getWritableTile(0, 0).setDataElements
                (0, 0, sourceImageWidth, sourceImageHeight, grayData);
    }
}
```

8. TiltCorrection.java

```
package numberplaterecognition;
import java.awt.Point;
import java.awt.image.BufferedImage;

/**
 * This class contains the tilt correction algorithm when create a new
 * PlateIsolation, first specifiy a bufferedImage type source image by
 * calling setSourceImage(), then call correction() function to perform
 * tilt correction algprithm to the source image, finally, use
 * getCorrectedImage() function to get the result image.
 */
public class TiltCorrection {

   int pixels[];
   int pixels2D[][];

   int correctPixels[];
   int correctPixels2D[][];

   BufferedImage sourceImage;
   BufferedImage currectImage;

   int sourceImageHeight;
   int sourceImageWidth;
   int imageSize;

   double rotatedAngle;
   double finalAnagle = 0;

   /**
```

```
 * set the source image
 */
public void setSourceImage(BufferedImage image)
{
   sourceImage = image;
}

/**
 * get the tilt corrected image after called the correction() function
 */
public BufferedImage getCorrectedImage()
{
   return currectImage;
}

/**
 * get the tilt corrected angle after called the correction() function
 */
public double getRotatedAngle()
{
   return finalAnagle;
}

/**
 * perform tilt correction algorithm to source image
 */
public void correction(double rotate)
{

   sourceImageWidth = sourceImage.getWidth();
   sourceImageHeight = sourceImage.getHeight();
   imageSize = sourceImageWidth * sourceImageHeight;
```

```
pixels = (int[]) sourceImage.getData().getDataElements
      (0, 0, sourceImageWidth, sourceImageHeight, null);
pixels2D = new int[sourceImageHeight][sourceImageWidth];

correctPixels = (int[]) sourceImage.getData().getDataElements
      (0, 0, sourceImageWidth, sourceImageHeight, null);
correctPixels2D = new int[sourceImageHeight][sourceImageWidth];

//convert array pixels into 2-dementional array
int w, h;
for(h = 0; h < sourceImageHeight; h++)
{
   for(w = 0; w < sourceImageWidth; w++)
   {
      pixels2D[h][w] = pixels[h*sourceImageWidth +w];
   }
}

//initialize image as black
for(w = 0; w < imageSize; w++)
   correctPixels[w] = 0xff000000;

int currentHeight = plateHeight(pixels);

if(rotate == 0)
{
   //rotation 0.5 degree one time, maximun rotate angle is 3 degree,
   //the best rotate angle is which makes the number plate has maxmun
   //height
   for(rotatedAngle = 0; rotatedAngle <= 3*Math.PI/180;
   rotatedAngle = rotatedAngle + 1*Math.PI/180)
   {
```

```
        rotateImageClockwise(rotatedAngle);
        int newHeightClockwise = plateHeight(correctPixels);

        rotateImageCounterclockwise(rotatedAngle);
        int newHeightCounterclockwise = plateHeight(correctPixels);

        if(newHeightClockwise >= newHeightCounterclockwise)
        {
          if(newHeightClockwise >= currentHeight)
          {
            currentHeight = newHeightClockwise;
            finalAnagle = rotatedAngle;
          }
        }

        if(newHeightCounterclockwise > newHeightClockwise)
        {
          if(newHeightCounterclockwise >= currentHeight)
          {
            currentHeight = newHeightCounterclockwise;
            finalAnagle = -rotatedAngle;
          }
        }
      }
      rotateImage(finalAnagle);
    }
    else
      rotateImage(rotate);

    currectImage = new BufferedImage(sourceImageWidth,
        sourceImageHeight, BufferedImage.TYPE_INT_ARGB);
    currectImage.getWritableTile(0, 0).setDataElements
        (0, 0, sourceImageWidth, sourceImageHeight, correctPixels);
```

```
}

/**
 * Rotate the image by an given angle
 */
public void rotateImage(double rotate)
{
   if(rotate >= 0)
      rotateImageClockwise(rotate);
   else
      rotateImageCounterclockwise(-rotate);
}

/**
 * Rotate the image clockwise
 */
public void rotateImageClockwise(double rotate)
{

   //Rotate around the image center point
   Point centerP = new Point(sourceImageWidth/2, sourceImageHeight/2);

   int rotatedX, rotatedY;
   int x, y, w;

   //initialize image as black
   for(w = 0; w < imageSize; w++)
      correctPixels[w] = 0xff000000;

   correctPixels2D = new int[sourceImageHeight][sourceImageWidth];

   for(y = 0; y < sourceImageHeight; y++)
   {
```

```
        for(x = 0; x < sourceImageWidth; x++)
        {
            if(pixels2D[y][x] == -1) //only rotate white pixels
            {

                rotatedX = (int)((x-centerP.x)*Math.cos(rotate)
                        + (centerP.y-y)*Math.sin(rotate));
                rotatedY = (int)(-(x-centerP.x)*Math.sin(rotate)
                        + (centerP.y-y)*Math.cos(rotate));

                if(centerP.x + rotatedX >= 0 && centerP.x
                        + rotatedX < sourceImageWidth &&
                        centerP.y - rotatedY >= 0 && centerP.y
                        - rotatedY < sourceImageHeight)
                {
                    correctPixels2D[centerP.y - rotatedY]
                            [centerP.x + rotatedX] = pixels2D[y][x];
                    correctPixels[(centerP.y - rotatedY)*sourceImageWidth
                            + (centerP.x + rotatedX)] =
                            correctPixels2D[centerP.y - rotatedY]
                            [centerP.x + rotatedX];
                }
            }

        }
    }
    correctPixels[centerP.y*sourceImageWidth + centerP.x] =
            pixels2D[centerP.y][centerP.x];
}

/**
 * Rotate the image clockwise counterclockwise
 */
```

```
public void rotateImageCounterclockwise(double rotate)
{

   //Rotate around the image center point
   Point centerP = new Point(sourceImageWidth/2, sourceImageHeight/2);

   int rotatedX, rotatedY;
   int x, y, w;

   //initialize image as black
   for(w = 0; w < imageSize; w++)
      correctPixels[w] = 0xff000000;

   correctPixels2D = new int[sourceImageHeight][sourceImageWidth];

   for(y = 0; y < sourceImageHeight; y++)
   {
      for(x = 0; x < sourceImageWidth; x++)
      {
         if(pixels2D[y][x] == -1) //only rotate white pixels
         {

            rotatedX = (int)((x-centerP.x)*Math.cos(rotate)
                  - (centerP.y-y)*Math.sin(rotate));
            rotatedY = (int)((x-centerP.x)*Math.sin(rotate)
                  + (centerP.y-y)*Math.cos(rotate));

            if(centerP.x + rotatedX >= 0 && centerP.x +
                  rotatedX < sourceImageWidth &&
                  centerP.y - rotatedY >= 0 && centerP.y - rotatedY
                  < sourceImageHeight)
            {
               correctPixels2D[centerP.y - rotatedY]
```

```
                    [centerP.x + rotatedX] = pixels2D[y][x];
                correctPixels[(centerP.y - rotatedY)*
                    sourceImageWidth + (centerP.x + rotatedX)] =
                    correctPixels2D[centerP.y - rotatedY]
                    [centerP.x + rotatedX];
            }
        }
    }
  }
  correctPixels[centerP.y*sourceImageWidth + centerP.x] =
      pixels2D[centerP.y][centerP.x];
}

/**
 * get the current height of the number plate
 */
public int plateHeight(int [] data)
{
  //resize the number plate area
  int [] horCount = new int [sourceImageHeight];
  int change = 0;
  int n, m;
  for(n = 0; n < sourceImageHeight; n++)
  {
    for(m = 0; m < sourceImageWidth-1; m++)
    {
      if(data[n*sourceImageWidth+m]!= data[n*sourceImageWidth+m+1])
      {
        change++;
      }
    }
    if (change > 20)
      horCount[n] = 1;
```

```
            else
                horCount[n] = 0;
            change = 0;
        }

        //find the top line of the number plate
        int top = 0;
        for(n = 1; n < sourceImageHeight; n++)
        {
            if(horCount[n-1] != horCount[n])
            {
                top = n-1;
                break;
            }
        }

        //find the botton line of the number plate
        int bottom = 0;
        for(n = sourceImageHeight-1; n > 0; n--)
        {
            if(horCount[n-1] != horCount[n])
            {
                bottom = n;
                break;
            }
        }
        return bottom-top;
    }
}
```