Exercice 1

```
/**
 * Exercice 1
 * @param t
 * @return
 */
public Image seuillage(int t){
    for(int i=0; i<this.pixels.length; i++){
        if(this.pixels[i] < t){
            this.pixels[i] = 0;
        }else{
            this.pixels[i] = 255;
        }
    }
    return this;
}
```

Exercice 2

```
/**
 * Exercice 2
 * @return
 */
public Image otsu(){
    int[] histo = this.histogram();
    double somme = 0;
    for (int t=0 ; t<256 ; t++){
        somme += t * histo[t];
    }
    double sommeA = 0;
    int wA = 0;
    int wB = 0;
    double varMax = 0;
    int threshold = 0;
    for (int i=0 ; i<256 ; i++) {
        wA += histo[i];
        if (wA == 0) continue;
        wB = this.pixels.length - wA;
        if (wB == 0) break;
        sommeA += (double) (i * histo[i]);
        double moyA = sommeA / wA;
        double moyB = (somme - sommeA) / wB;
        double var = (double)wA * (double)wB * (moyA - moyB) * (moyA - moyB);
        if (var > varMax) {
            varMax = var;
            threshold = i;
        }
    }
    this.seuillage(threshold);
    return this;
}

/**
 * Exercice 2
 * @return
 */
public Image iterativeSelectionThresholding(){
    double [] histo = this.normalizedHistogram();
    int t = 128;
    int[] ancienT = new int[256];
    while (ancienT[t] != 1){
        ancienT[t] = 1;
        int indexA=0;
        double moyA = 0.0;
        for(int i=0; i<t; i++){
```

```
            moyA += i * histo[i];
        }
        moyA = moyA/255;
        for(int j=0; j<t && moyA>=0;j++){
            moyA -= histo[j];
            indexA=j;
        }
        int indexB=0;
        double moyB = 0.0;
        for(int i=t; i<256; i++){
            moyB += i * histo[i];
        }
        moyB = moyB/255;
        for(int j=t; j<256 && moyB>=0;j++){
            moyB -= histo[j];
            indexB=j;
        }
        t = (indexA+indexB)/2;
    }
    this.seuillage(t);
    return this;
}

/** Exercice 2
 * @return
 */
public Image otsuNormalized(){
    double[] histo = this.normalizedHistogram();
    double variance = 0.0;
    double varianceMax = 0.0;
    int seuil = 0;
    for(int k = 0 ; k < histo.length ; k ++) {
        double moyA = 0.0;
        double moyB = 0.0;
        double probaA = 0.0;
        double probaB = 0.0;
        for(int i = 0 ; i < k ; i++) {
            probaA = probaA + histo[i];
            moyA = moyA + i * histo[i];
        }
        for(int j = k ; j < histo.length ; j ++) {
            probaB = probaB + histo[j];
            moyB = moyB + j * histo[j];
        }
        moyA = moyA / probaA;
        moyB = moyB / probaB;
        variance = probaA * probaB * ((moyA - moyB) * (moyA - moyB));
        if(variance > varianceMax){
            varianceMax = variance;
            seuil = k;
        }
    }
    this.seuillage(seuil);
    return this;
}
```

Exercice 3

```
/**
 * Exercice3
 * @param s
 * @return
 */
public Image otsu2(int s) {
    for(int x = 0 ; x < this.width ; x ++) {
```

```
        for(int y = 0 ; y < this.height ; y ++) {
            if(this.getValue(x, y) < this.intOtsu(s, x, y)) {
                this.setValue(x, y, 0);
            } else {
                this.setValue(x, y, 255);
            }
        }
    }
    return this;
}

/**
 * Exercice 3
 * @param s
 * @param x
 * @param y
 * @return
 */
public int intOtsu(int s, int x, int y) {
    double[] histo = this.normalizedHistogramSeuillage(s, x, y);
    double[] variance = new double[histo.length];
    for(int k = 0 ; k < histo.length ; k ++) {
        double moyA = 0.0;
        double moyB = 0.0;
        double probaA = 0.0;
        double probaB = 0.0;
        for(int i = 0 ; i < k ; i++) {
            probaA = probaA + histo[i];
            moyA = moyA + i * histo[i];
        }
        for(int j = k ; j < histo.length ; j ++) {
            probaB = probaB + histo[j];
            moyB = moyB + j * histo[j];
        }
        moyA = moyA / probaA;
        moyB = moyB / probaB;
        variance[k] = probaA * probaB * ((moyA - moyB) * (moyA - moyB));
    }
    double varianceMax = 0.0;
    int seuilMax = 0;
    for(int l = 0 ; l < variance.length ; l ++) {
        if(variance[l] > varianceMax) {
            varianceMax = variance[l];
            seuilMax = l;
        }
    }
    return (int)seuilMax;
}

/**
 * Exercice 3
 * @param s
 * @param x
 * @param y
 * @return
 */
public double[] normalizedHistogramSeuillage(int s, int x, int y) {
    int[] histo = new int[256];
    double[] ret = new double[256];
    int pixels = 0;
    for(int xs = x - s ; xs < x + s ; xs ++) {
        for(int ys = y - s ; ys < y + s ; ys ++) {
            if(xs >= 0 && xs < this.width && ys >= 0 && ys < this.height) {
                histo[this.getValue(xs, ys)] ++;
```

```
                pixels ++;
            }
        }
    }
    for(int i = 0 ; i < 256 ; i ++) {
        ret[i] = histo[i] / (double)(pixels);
    }
    return ret;
}
```

Exercice 4

```
/**
 * Exercice 4
 * @return
 */
public Image erosion3x3(){
    Image ret = new Image(this.getWidth(),this.getHeight());
    int x,y;
    int max;
    for(int i=0; i<this.pixels.length; i++){
        max = 0;
        x = i/ret.getWidth();
        y = i-(x*ret.getWidth());
        for(int k=y-1; k<=y+1; k++){
            if(k>=0 && k<ret.getWidth()){
                for(int j=x-1; j<=x+1; j++){
                    if(j>=0 && j<ret.getHeight()){
                        if(this.getValue(k, j) > max){
                            max = this.getValue(k, j);
                        }
                    }
                }
            }
        }
        ret.pixels[i] =  max;
    }
    return ret;
}

/**
 * Exercice 4
 * @return
 */
public Image dilatation3x3(){
    Image ret = new Image(this.getWidth(),this.getHeight());
    int x,y;
    int min;
    for(int i=0; i<this.pixels.length; i++){
        min = 255;
        x = i/ret.getWidth();
        y = i-(x*ret.getWidth());
        for(int k=y-1; k<=y+1; k++){
            if(k>=0 && k<ret.getWidth()){
                for(int j=x-1; j<=x+1; j++){
                    if(j>=0 && j<ret.getHeight()){
                        if(this.getValue(k, j) < min){
                            min = this.getValue(k, j);
                        }
                    }
                }
            }
        }
        ret.pixels[i] =  min;
    }
```

```
    return ret;
}
```

5 Exercice 5

```
/**
 * Exercice 5
 * @param n
 * @return
 */
public Image erosion(int n){
    Image ret = new Image(this.getWidth(),this.getHeight());
    int x,y;
    int max;
    for(int i=0; i<this.pixels.length; i++){
        max = 0;
        x = i/ret.getWidth();
        y = i-(x*ret.getWidth());
        for(int k=y-n; k<=y+n; k++){
            if(k>=0 && k<ret.getWidth()){
                for(int j=x-n; j<=x+n; j++){
                    if(j>=0 && j<ret.getHeight()){
                        if(this.getValue(k, j) > max){
                            max = this.getValue(k, j);
                        }
                    }
                }
            }
        }
        ret.pixels[i] =  max;
    }
    return ret;
}
```

```
/**
 * Exercice 5
 * @return
 */
public Image dilatation(int n){
    Image ret = new Image(this.getWidth(),this.getHeight());
    int x,y;
    int min;
    for(int i=0; i<this.pixels.length; i++){
        min = 255;
        x = i/ret.getWidth();
        y = i-(x*ret.getWidth());
        for(int k=y-n; k<=y+n; k++){
            if(k>=0 && k<ret.getWidth()){
                for(int j=x-n; j<=x+n; j++){
                    if(j>=0 && j<ret.getHeight()){
                        if(this.getValue(k, j) < min){
                            min = this.getValue(k, j);
                        }
                    }
                }
            }
        }
        ret.pixels[i] =  min;
    }
    return ret;
}
```

Exercice 7

```
/**
 * Exercice 7
```

```
     * @param s
     * @return
     */
    public Image ouverture(int s){
        Image ret = new Image(this);
        return ret.erosion(s).dilatation(s);
    }

    /**
     * Exercice 7
     */
    public Image fermeture(int s){
        Image ret = new Image(this);
        return ret.dilatation(s).erosion(s);
    }

    /**
     *
     * @param n
     * @return
     */
    public Image topHatOuv(int n) {
        Image ret = new Image(this);
        Image ouv = ret.ouverture(n);
        for(int i=0; i<ret.pixels.length; i++) ret.pixels[i]= ret.pixels[i] - ouv.pixels[i];
        return ret;
    }

    /**
     *
     * @param n
     * @return
     */
    public Image topHatFer(int n) {
        Image ret = new Image(this);
        Image fer = ret.dilatation(n);
        for(int i=0; i<ret.pixels.length; i++) ret.pixels[i]= ret.pixels[i] - fer.pixels[i];
        return ret;
    }

    /**
     *
     */
    public Image gradMorpho(int n) {
        Image ret = new Image(this);
        for(int x = 0; x < this.getWidth(); x++) {
            for (int y = 0; y < this.getHeight(); y++){
                ret.setValue(x, y, (ret.dilatation(n).getValue(x, y) - ret.erosion(n).getValue(x,
y)));
            }
        }
        return ret;
    }

    /**
     *
     * @param n
     * @return
     */
    public Image lapMorpho(int n) {
        Image ret = new Image(this);
        for(int x = 0; x < this.getWidth(); x++) {
            for (int y = 0; y < this.getHeight(); y++){
                ret.setValue(x, y, (ret.dilatation(n).getValue(x, y) + ret.erosion(n).getValue(x,
```

```
y) - 2 * ret.getValue(x, y)));
            }
        }
        return ret;
    }

    /**
     *
     * @param n
     * @return
     */
    public Image toutEnRien(int n) {
        Image ret = new Image(this);
        for(int x = 0; x < this.getWidth(); x++) {
            for (int y = 0; y < this.getHeight(); y++){
                ret.setValue(x, y, (ret.erosion(n).getValue(x, y) - ret.dilatation(n).getValue(x,
y)));
            }
        }
        return ret;
    }

    private int nbLabel;

    /**
     *
     * @return
     */
    public int[] etiquetageINIT(){
        int[] etiq = new int[this.pixels.length];
        nbLabel = 0;
        for(int x = 0; x < this.width ; x ++) {
            for(int y = 0; y < this.height ; y ++) {
                etiq = this.etiquetage(etiq,nbLabel,x,y);
            }
        }
        return etiq;
    }

    /**
     *
     */
    public int[] etiquetage(int[] etiq, int nbLabel, int xInit,int yInit){
        if(this.getValue(xInit, yInit)==255 && etiq[xInit+yInit*this.width]==0){
            for(int x=xInit-1; x<=xInit+1; x++) {
                for(int y=yInit-1; y<=yInit+1; y++) {
                    if(x>=0 && y>=0 && y<this.height && x<this.width){
                        if(xInit!=x && yInit!=y){
                            if(this.getValue(x, y)==255){
                                if(etiq[x+y*this.width]!=0){
                                    etiq[xInit+yInit*this.width] = etiq[x+y*this.width];
                                    etiq = this.etiquetage(etiq, nbLabel,x,y);
                                }else if(etiq[x+y*this.width]==0){
                                    nbLabel++;
                                    etiq[xInit+yInit*this.width]= nbLabel;
                                    etiq = this.etiquetage(etiq, nbLabel, x, y);
                                }
                            }
                        }
                    }
                }
            }
        }
        return etiq;
```

```
}
```

Exercice 8

```
/**
 * Exercice8
 * @return
 */
public Image etiquetage() {
    Image ret = this.otsu();
    int etiquette = 0;
    for(int x = 0 ; x < ret.getWidth(); x ++) {
        for(int y = 0 ; y <ret.getHeight() ; y ++) {
            if(ret.getValue(x, y) == 255) {
                etiquette ++;
                ret = diffusion(x, y, etiquette, ret);
            }
        }
    }
    return ret;
}

/**
 * Exercie 8
 * @param x
 * @param y
 * @param etiquette
 * @param ret
 * @return
 */
public Image diffusion(int x, int y, int etiquette, Image ret) {
    System.out.println(etiquette);
    if(etiquette != 255) {
        if(ret.getValue(x - 1, y - 1) == 255) {
            ret.setValue(x - 1, y - 1, etiquette);
            diffusion(x - 1, y - 1, etiquette, ret);
        }

        if(ret.getValue(x + 1, y - 1) == 255) {
            ret.setValue(x + 1, y - 1, etiquette);
            diffusion(x + 1, y - 1, etiquette, ret);
        }

        if(ret.getValue(x - 1, y + 1) == 255) {
            ret.setValue(x - 1, y + 1, etiquette);
            diffusion(x - 1, y + 1, etiquette, ret);
        }

        if(ret.getValue(x + 1, y + 1) == 255) {
            ret.setValue(x + 1, y + 1, etiquette);
            diffusion(x + 1, y + 1, etiquette, ret);
        }
    }
    return ret;
}
```

Exercice 9

```
//Exercice9
public ArrayList<Integer> tableEtiq() {
    ArrayList<Integer> table = new ArrayList<Integer>();
    Image ret = this;
    int etiquette = 0;
    int etiqMin = 0;
    int boucle1 = 0, boucle2 = 0, boucle3 = 0, boucle4 = 0;
```

```
      for(int x = 0 ; x < ret.getWidth() ; x ++) {
         for(int y = 0 ; y < ret.getHeight() ; y ++) {
            if(ret.getValue(x, y) != 0) {
               etiquette ++;
               table.add(etiquette);
               etiqMin = etiquette;

               if(x - 1 > 0 && y - 1 > 0) {
                  if(ret.getValue(x - 1, y - 1) != 0 && ret.getValue(x - 1, y - 1) < etiqMin)
{
                     etiqMin = ret.getValue(x - 1, y - 1);
                     table.set(etiquette - 1, etiqMin);
                     boucle1 ++;
                  }
               }

               if(y - 1 > 0) {
                  if(ret.getValue(x, y - 1) != 0 && ret.getValue(x, y - 1) < etiqMin) {
                     etiqMin = ret.getValue(x, y - 1);
                     table.set(etiquette - 1, etiqMin);
                     boucle2 ++;
                  }
               }

               if(x + 1 < ret.getWidth() && y - 1 > 0) {
                  if(ret.getValue(x + 1, y - 1) != 0 && ret.getValue(x + 1, y - 1) < etiqMin)
{
                     etiqMin = ret.getValue(x + 1, y - 1);
                     table.set(etiquette - 1, etiqMin);
                     boucle3 ++;
                  }
               }

               if(x - 1 > 0) {
                  if(ret.getValue(x - 1, y) != 0 && ret.getValue(x - 1, y) < etiqMin) {
                     etiqMin = ret.getValue(x - 1, y);
                     table.set(etiquette - 1, etiqMin);
                     boucle4 ++;
                  }
               }
            }
         }
      }
      System.out.println("boucle 1 : " + boucle1 + "\nboucle 2 : " + boucle2 + "\n boucle
3 : " + boucle3 + "\n boucle 4 : " + boucle4);
      return table;
   }
```