1) Pixel Graber and Pixel Packer

i. Pixel Graber

```
int[] pixel_grab(Image img,int width,int height)
  {
        int pixSrc[] = new int[w*h];
        try
        {
              PixelGrabber pg = new PixelGrabber(img,0,0,w,h,pixSrc,0,w);
              pg.grabPixels();
        }
        catch (Exception e){System.out.println("Exception at Pixel Grabbing");}

        for(int i=0; i<w*h; i++)  //  Convertion of pixle form to normal single Value...
        {
              int a = pixSrc[i];
              int r = (0xff & (a>>16));
              int g = (0xff & (a>>8));
              int b = (0xff & a);
              int avg = (int)((.33*r+.56*g+.11*b)); // Add a constant to each pixel
              pixSrc[i]=avg;
        }
        return(pixSrc);
  }
```

public PixelGrabber(Image img, int x, int y, int w, int h, int[] pix,int off, int scansize)
Create a PixelGrabber object to grab the (x, y, w, h) rectangular section of pixels from the specified image into the given array. The pixels are stored into the array in the default RGB ColorModel. The RGB data for pixel (i, j) where (i, j) is inside the rectangle (x, y, w, h) is stored in the array at
pix[(j - y) * scansize + (i - x) + off].

Parameters:

img - the image to retrieve pixels from

x - the x coordinate of the upper left corner of the rectangle of pixels to retrieve from the image, relative to the default (unscaled) size of the image

y - the y coordinate of the upper left corner of the rectangle of pixels to retrieve from the image

w - the width of the rectangle of pixels to retrieve

h - the height of the rectangle of pixels to retrieve

pix - the array of integers which are to be used to hold the RGB pixels retrieved from the image

off - the offset into the array of where to store the first pixel

scansize - the distance from one row of pixels to the next in the array

Convertion of pixle form to normal single

pixel is an int an int has 4 bytes in it, each byte has 8 bits in it.
as an example 0xA3 0x41 0x28 0x76 is an example pixel (1 int, with 4 bytes in it and 32 bits)
Now packed in this pixel is information about the transparency (A) the red, green and blue components.
The transparency is the first byte (0xA3) then red (0x41) then green (0x28) then blue (0x76)
so to get just the **red part out, you shift to the right by 16 bits**, which gives
0x00 0x00 0xA3 0x41
now the red is the right most spot, but you got that transparency byte there, which isn't good, so you have to remove it
doing & 0xFF, is really
0xA341 & 0x00FF
which means you **AND** each bit, so the bits of A3 are anded with 00 which gives 0, and the bits of 41 are anded with FF which since FF is all ones, gives you the original value, which is 41

so red is

0x00 0x00 0x00 0x41

So this code, is pulling out each component by itself from an integer.

ii. Pixel Packer

```
int [] pix_pack(int pixSrc[],int wid,int hgt)
        {
                for(int i=0;i<wid*hgt;i++)
        {
                pixSrc[i] = (0xff000000|pixSrc[i]<<16|pixSrc[i]<<8|pixSrc[i]);
        }

        return(pixSrc);
        }
```

To positioned Alpha, red, green and blue to its position in integer(4bytes) the above method is used.
First, right shifted values are leftshifted. Then perform **OR** operation.

2) Mask Operation with example

I. Low Pass Fiter

```
if(e.getSource() == mlowPass)
        {
                int mask[][] = {{1,1,1},{1,1,1},{1,1,1}};

                pixel_result = pixel_grab(image,w,h);

                int pix_tempLow[][] = new int[h][w];


                pix_tempLow =  OneD_ArrayToTwoD_Array(w,h,pixel_result);

                pix_tempLow= MaskOperation(w,h,pix_tempLow,mask,9);
                int pix_temp1D[] = new int[(w)*(h)];



                pix_temp1D = TwoD_ArrayToOneD_Array(w,h,pix_tempLow);


                pix_temp1D = pix_pack(pix_temp1D,w-2,h-2);
                imageResultDisplay(w-2,h-2,pix_temp1D);

        }
```

- Low pass 3*3 mask is manually created for masking operation.
- Using the pixel grab method, the values of R, G, B of input image is calculated.
- Basically, the value of pixel grab method is in 1D array which is then converted to 2D and assign to new 2D Array.
- Mask operation is performed using Maskoperation class, which takes width, height, 2d array, mask and 9 as the the inputs. Here 9 is sum of values in the mask array.
- The resultant 2D array is converted to 1D array and use the pixel pack method to set the image.

II. Mask Operation

```
int[][] MaskOperation(int width,int height,int pix_tempImg[][], int
temp_mask[][],int d)
 {

    int pix_tempImg1[][] = new int[height][width];

    for(int i =1;i<height-1;i++)
    {
            for(int j=1;j<width-1;j++)
            {
                    pix_tempImg1[i][j] = ((((temp_mask[0][0]*pix_tempImg[i-1][j-1])
+ (temp_mask[1][0]*pix_tempImg[i][j-1]) + (temp_mask[2][0]*pix_tempImg[i+1][j-
1]))+

    ((temp_mask[0][1]*pix_tempImg[i-1][j]) + (temp_mask[1][1]*pix_tempImg[i][j])
+ (temp_mask[2][1]*pix_tempImg[i+1][j]))+

    ((temp_mask[0][2]*pix_tempImg[i-1][j+1]) +
(temp_mask[1][2]*pix_tempImg[i][j+1]) +
(temp_mask[2][2]*pix_tempImg[i+1][j+1])))/d);
            }
            System.out.println("mask is00"+temp_mask[0][0]);
                    System.out.println("mask is01"+temp_mask[0][1]);
                    System.out.println("mask is02"+temp_mask[0][2]);
                    System.out.println("mask is10"+temp_mask[1][0]);
                    System.out.println("mask is11"+temp_mask[1][1]);
                    System.out.println("mask is12"+temp_mask[1][2]);
                    System.out.println("mask is20"+temp_mask[2][0]);
                    System.out.println("mask is21"+temp_mask[2][1]);
                    System.out.println("mask is22"+temp_mask[2][2]);

    }

    return(pix_tempImg1);



 }
```

- 2D array is created for storing the resultant matrix.
- This 3*3 mask matrix is multiplied with each pixel of input image (pixelgrabed 2D array)
- To normalize it the resultant 2D array is divide by sum of values in the mask array ie. 9

```
int[] TwoD_ArrayToOneD_Array(int width,int height,int pix_tempLow[][])
 {
    int pix1D[] = new int[(height-2)*(width-2)];
    int c=0;

    for(int i = 1;i<height-1;i++)
    {
            for(int j =1;j<width-1;j++)
            {
                    pix1D[c] = pix_tempLow[i][j];
                    c++;
            }
    }

    return(pix1D);
 }
```

```
int[][] OneD_ArrayToTwoD_Array(int width,int height,int pix_temp[])
 {

    int pix2D[][] = new int[height][width];
    int c = 0;

    for(int i=0;i<height;i++)
    {
            for(int j =0;j<width;j++)
            {

                    pix2D[i][j] = pix_temp[c++];

            }
    }

    return(pix2D);

 }
```

3) Histogram Equalization

```
if(arg.equals("Histogram Equalization"))
        {
            int[] pix_grabed = new int[h*w];
            int pres[] = new int[h*w];
            int cnt_pix[] = new int[256];
            int pixx[] = new int[h*w];
            int pix_pcked[] = new int[h*w];
            int pix_res[] = new int[h*w];
            int len = h*w;
            int res = 0;

            pix_grabed = pixel_grab(image,w,h);


            for(int i = 0;i<h*w;i++)
          {
            pixx[i] = pix_grabed[i];

          }

            for(int i = 0; i < 256; i++)
            {

                cnt_pix[i] = 0;
        }

            for(int i = 0; i < len; i++)
            {

                    int ind = pixx[i];
                    cnt_pix[ind]++;
            }

          for(int i=0;i<w*h;i++)
          {
            float a = 0;

            for(int j =0;j<(pixx[i]+1);j++)
            {
                float b  = (float)cnt_pix[j];

                    float c = (float)(h*w);
```

```
                a = a + (b/c);
            }

            res =(int)(a*255);
            if(res > 255)
             res = 255;

            pix_res[i] = ( 0xff000000 | (res << 16) | (res << 8) | res);

        }

        pix_pcked = pix_pack(pix_res,w,h);
            imageResultDisplay(w,h,pix_pcked);



    }
```

- 4 temporary 1d array of size w*h is created.
- One 1d array of size 256 is created to get the count of the pixel values.
- The pixel grab method is called and assign to **pixel_grabed** array which each value of it assign to **pixx** array.
- Initially the count array ie **cnt_pix** made 0.
- How many times pixel value is repeated is counted and stored in **cnt_pix. This is called as Histogram.**
- Perform Histogram **Equalization $P_r(r_k) = n_k/MN$** ie **a = a + (b/c)**
  Here, a is initially 0.
  b is count.
  c is length.
- The resultant value is multiplied to 255(spreading the histogram on entire scale) and check resultant values, if resultant value is greater than 255 then assign it as 255.
- Perform the packing operation.