Latihan 5_1

1 First steps in ImageJ

1.1 Display of images, histograms

- Start `ImageJ`.
- Open some of the images `*mr*` and `*ct*` located in the folder mentioned above. `(File – Open…)`
- Create a histogram of each image. `(Analyze – Histogram)`
  Try also logarithmic scaling.

1.2 Contrast modification

- Open one of the images `CT-head-*`.
- Create an image, which shows the bone only, by modification of contrast and brightness.
  `(Image – Adjust – Contrast/Brightness)`
- Open the image `CT-abdomen`.
  Try different settings for level and window in order to show a certain matter with good contrast.
  `(Image – Adjust – Window/Level)`
- Open the image `exc02`.
  How many stripes does each of the circles have?
  Is it possible to show the stripes of all circles in one image by contrast modification?

1.3 Lookup-Tables (LUT)

- Open the image `CT-head-2`.
  Create an image, which shows bone, tissue and air in different colors.
  `(Image – Color – Edit LUT)`
  Create two versions:
  1. One color per matter.
  2. One color range per matter leaving an impression of the intensities.
- Open the image `exc02`.
  Create an image, which shows the stripes of all circles at the same time.
- Open the image `exc03.tif`.
  How many squares are in this image?
  Create an image, which clearly shows all squares at the same time.

1.4 Threshold and Regions

- Use the image `CT-abdomen.dcm`
- Create a binary image, which shows the bone only `(Image – Adjust – Threshold…)`
- Create other binary images, showing the kidneys only and showing the muscles only.

Latihan 5_2

2 Introduction to Plugin-Programming in ImageJ

- Create a folder *MIP* in your personal directory.
- Start the program *ImageJ* from the Windows Start-menu or by double clicking ImageJ.exe.

2.1 Programming a PluginInFilter

In this section, you will create your first image processing program in *ImageJ*.

- Download the file Example0_.java from *Canvas* (folder *plugins*) and store it into the folder C:\ImageJ\plugins (or other folder where your *ImageJ* is installed)
- Use the Windows-Explorer to move into this folder.
- Open the file Example0_.java with an editor-program.

You will see a very short source-code written in Java. Most of the lines are currently uninteresting. One interesting line is

```
public void run(ImageProcessor ip) {
```

This line is important, because it defines the beginning of the method (function), which is executed, if we later start the plugin from the *ImageJ*-menu. The line contains the important object (variable) ip, which contains all information of the image which shall be processed by the plugin.

The following line(s) will always include the functionality of our plugin. This first example-plugin does only contain one line:

```
ip.putPixel( 10,10, 255 );
```

This line can be understood as follows: For the image stored in ip, the method putPixel is called. This method has three parameters. The first two are the x- and y-coordinates of a pixel, and the third one is the intensity-value, which shall be written to this position.

The plugin shall now be started:

- In *ImageJ*, select Plugins - Compile and Run...
- In the next dialog, move into the folder plugins.
- Double-click on your file Example0_.java .

The plugin is now compiled and executed. The execution happens on the currently selected image in *ImageJ*.

In your image, you should now see a white pixel near the top left corner.

2.2 Exercise: Short white line

Preparations: Create a copy the file Example0_.java, and call the new file Exercise0202_.java . Open the new file with an editor.

In *ImageJ*-plugins it is important that the last character of the filename is an underscore "_"!

It is also important that the internal name of the plugin is the same as the filename except of the extension ".java". This internal name is specified in the source code line, which starts with public class.

In this exercise this line must therefore be

```
public class Exercise0202_ implements PlugInFilter {
```

Exercise: Modify and extend the source code such that the plugin creates a short white line from pixel (30,10) to pixel (39,10). Apply this plugin to the image CT-abdomen.bmp.

2.3 Storage of plugins on the PC

Your plugins must be located in the *plugin*-folder of the *ImageJ*-root-directory and you should edit them directly in this folder.

2.4 Loop in one direction

Now we want to draw a line of 250 pixels into an image. This procedure is best implemented by a loop.

- Open the file Example0_.java with an editor.
- Enter the following lines into the body of the *run*-method

```
for( int x=0; x<250; x++ ){
        ip.putPixel( x,50, 255 );
}
```

- Run the plugin on the image.

2.5 Exercise: Vertical line of 250 pixels

Create a new plugin, which draws a vertical white line, which starts at the pixel (50,0) and has a length of 250 pixels.

Latihan Minggu 6_1

2.6 Using the image dimensions
If we want to draw a line across the whole image, we have to know the width and/or the height of the image. These dimensions are provided by the methods getWidth() and getHeight(). Usually, the image dimensions are first copied to local variables before they are used:

```
int width = ip.getWidth();
int height = ip.getHeight();
```

2.7 Exercise: Vertical middle line
Create a new plugin, which draws a vertical white line. The line shall extend from top to bottom of the image and its horizontal position shall be in the middle of the image.

2.8 Two nested loops
In image processing methods it is very common to work on the whole image or at least on a rectangle of pixels, which leads to a construction with two nested loops:

```
for( int y=y_start; y<=y_end; y++ ){
    for( int x=x_start; x<=x_end; x++ ){
        // more commands
    }
}
```

In this case (x_start, y_start) is the top left corner of the rectangle and (x_end, y_end) is the bottom right corner.

In case of working on the whole image, we have x_start=y_start=0, x_end=width-1, y_end=height-1. In this case we usually use "<" instead of "<=" and omit "-1":

```
for( int y=0; y<height; y++ ){
    for( int x=0; x<width; x++ ){
        // more commands
    }
}
```

2.9 Exercise: Black top
Write a plugin, which fills the upper half of the image with zero intensity (black).

2.10 Exercise: Black square
Write a plugin, which draws a black box into the middle of the current image. The width and the height of the box shall be 50% of the width and the height of the image.

2.11 Reading intensities from the image
So far, we have only written intensities to pixels. However, it is of course also very relevant to read existing intensities from (input) pixels. This is done by the method *getPixel*:

```
v=ip.getPixel( x,y );
```

This command stores the intensity, which is present at the pixel *(x,y)* in the variable *v*.

2.12 Exercise: Inverted Image
Write a plugin, which inverts the current image: $I_{out} = 255 - I_{in}$
(White input-pixels will be turned into black, black input pixels will be turned into white, light pixels will become dark and dark pixels will become light.)

2.13 Exercise: Top left quarter copied
Write a plugin, which copies the top left quarter of the image to the three other quarters.

2.14 Exercise: Duplicated in a mirror
Write a plugin, which mirrors the upper half of the current image to the lower half.

2.15 Conditions
In order to program conditional behavior, we use the Java if-else syntax, e.g.

```
if( ip.getPixel(x,y) > 128 ){
    ip.putPixel(x,y, 255 );
}
else {
    ip.putPixel(x,y, 255 );
}
```

In order to check equality, we have to use the == operator, and for inequality we have to use the != operator,e.g.

```
if( ip.getPixel(x,y) == 128 ) {          // if intensity is equal to 128
...

if( ip.getPixel(x,y) != 128 ) {          // if intensity is NOT equal to 128
...
```

Two conditions can be linked by the operators AND && or OR ||, e.g.

```
if( ip.getPixel(x,y) == 128 && ip.getPixel(x+1,y) == 128 ){
...

if( ip.getPixel(x,y) >=100 || ip.getPixel(x,y) <= 200 ){
...
```

2.16 Exercise: Black pixels to white
Write a plugin, which turns the black pixels (intensity=0) into white pixels (intensity=255) and leaves all other pixels unchanged.

2.17 Exercise: Threshold
Write a plugin, which turns all pixels with intensities greater or equal than 200 into white and all other pixels into black. (The value 200 is called the *threshold* for this operation.)

2.18 Text-Output
In order to output text information for the user, you can use the method *IJ.showMessage*, e.g.

```
IJ.showMessage( "n=" + n + " pixels." );
```

You can output many items linked by +-signs. Characters inside of double quotes will be output without any change. All other characters are interpreted as variable names, and the value of the variable is output. In the above example, the output may be

```
n=218 pixels.
```

if the variable *n* had the value 218.

2.19 Counting
Counting is a general and useful procedure in many programs. The general structure is always like this:

```
int n=0;    // Initializing the count
for (...){
        ...
        n=n+1;    // Counting, can be abbreviated by n++;
}
```

2.20 Exercise: Counting pixels
Write a plugin, which outputs the number of pixels with zero intensity and the number of pixels with intensity larger or equal than 200.

Latihan Minggu 6_2

2.21 Definition of variables
In the beginning, we will only use integer and floating points variables. Integer variables are defined by `int`, floating point variables by `double`, e.g.

```
int start, end;
double average;
```

2.22 Mathematical functions
Mathematical functions are called from the *Math*-class, e.g.

```
Double u;
u=Math.sqrt( 2.4 );        // Square-Root
u=Math.sin( 3.14 );        // Sine-function in radiants!!
                           // Analogously: Math.cos, Math.tan
u=Math.PI;                 // The circle-number Pi defined as a constant
```

2.23 Exercise: Circle
Write a plugin, which draws a white circle into the current image. The circle shall have its center at the center of the image and its radius shall be a quarter of the image diagonal. It is advantageous to use a parametric representation of the circle, where x_0 and y_0 are the center-coordinates and φ is running from 0 to 2π in a not too big step size:

$$x = x_0 + r \cdot \cos(\varphi)$$
$$y = y_0 + r \cdot \sin(\varphi)$$

2.24 Maximum search
In images we are sometimes seeking maximum intensities or maximum coordinates with certain properties. The following source code searches the maximum intensity of an image.

```
maxIntensity=0;        // Initialize the maximum to a value,
                       // which is for sure LOWER than the maximum
for( int x=0; x<width; x++ ){
        for( int y=0; y<height; y++ ){
                v=ip.getPixel(x,y);

                if( maxIntensity < v ){
                        maxIntensity = v;
                }
        }
}
IJ.showMessage( "Maximum intensity=" + maxIntensity );
```

The probably most difficult step is the initialization of the variable *maxIntensity* in the beginning. It must be initialized to a value, which is definitely smaller than the maximum. (And if we wanted to find a minimum, we must initialize the minimum-variable to a value, which is definitely larger than the minimum.)
After the initialization, the program loops through the whole image and tests at each pixel, if the intensity at this pixel is larger than the maximum intensity found so far. If the intensity is larger, the variable *maxIntensity* gets this intensity as its new value.

At the end of the loop, the variable *maxIntensity* contains the maximum intensity.

2.25 Exercise: Maximum search
a) Implement the source code from above into a plugin *Exercise0126_.java* and apply this plugin to the image CT-abdomen.bmp .
(You will probably not be surprised that the maximum intensity of this image is 255.)

b) Extend the plugin such that it also prints the largest x-coordinate, which has the maximum intensity.
(Hint: For this, you need an additional loop through the whole image. During this loop, you only consider the pixels with maximum intensities. And for all these positions you check if the corresponding x-coordinate is larger than the current maximum-x-coordinate.)

c) Extend the plugin such that it also prints the smallest x-coordinate, which has the maximum intensity.

d) Extend the plugin such that it also prints the smallest and the largest y-coordinate, which have the maximum intensity.

e) Extend the plugin such that finally a white rectangle is drawn into the image, which has the top left corner given by the lowest coordinates determined in steps c) and d) and the bottom right corner given by the largest coordinates determined in steps b) and d).

2.26 Exercise: Diagonal
Write a plugin, which draws a white diagonal line starting from the top left corner. Each new point of the line shall be one row below and one column right of the previous pixel.
The plugin shall work for quadratic images as well as for rectangular images, where one image dimension is smaller than the other. Try e.g. the image MR-head-8bit.tif.

2.27 Exercise: Vertical bars
Write a plugin, which fills the current image with vertical bars, which have a width of 50 pixels. The first bar shall have intensity 0, the second bar shall have intensity 10, the next one intensity 10, and so on.
The last bar will probably not have the full width of 50 pixels, because the image width is not always a multiple of 50.

2.28 Exercise: Vertical bars of growing width
Write a plugin, which fills the first column of the current image with intensity 0, the next two columns with intensity 5, the next three columns with intensity 10, the next four columns with intensity 15, and so on.

2.29 Exercise: Chessboard 3x3
Write a plugin, which fills the current image by a 3x3 chessboard of rectangles. The size of each rectangle shall be one third of the image dimensions. The squares of the rectangles shall be either black or white.

2.30 Exercise: Chessboard inversion
On basis of plugin 2.29 write a plugin, which creates a 3x3 chessboard-pattern, where one group of rectangles shows the inverse image and the other group shows the original image.