CSC1130 Introduction to Computing Using Java 2019-2020 First Term

Department of Computer Science and Engineering, The Chinese University of Hong Kong

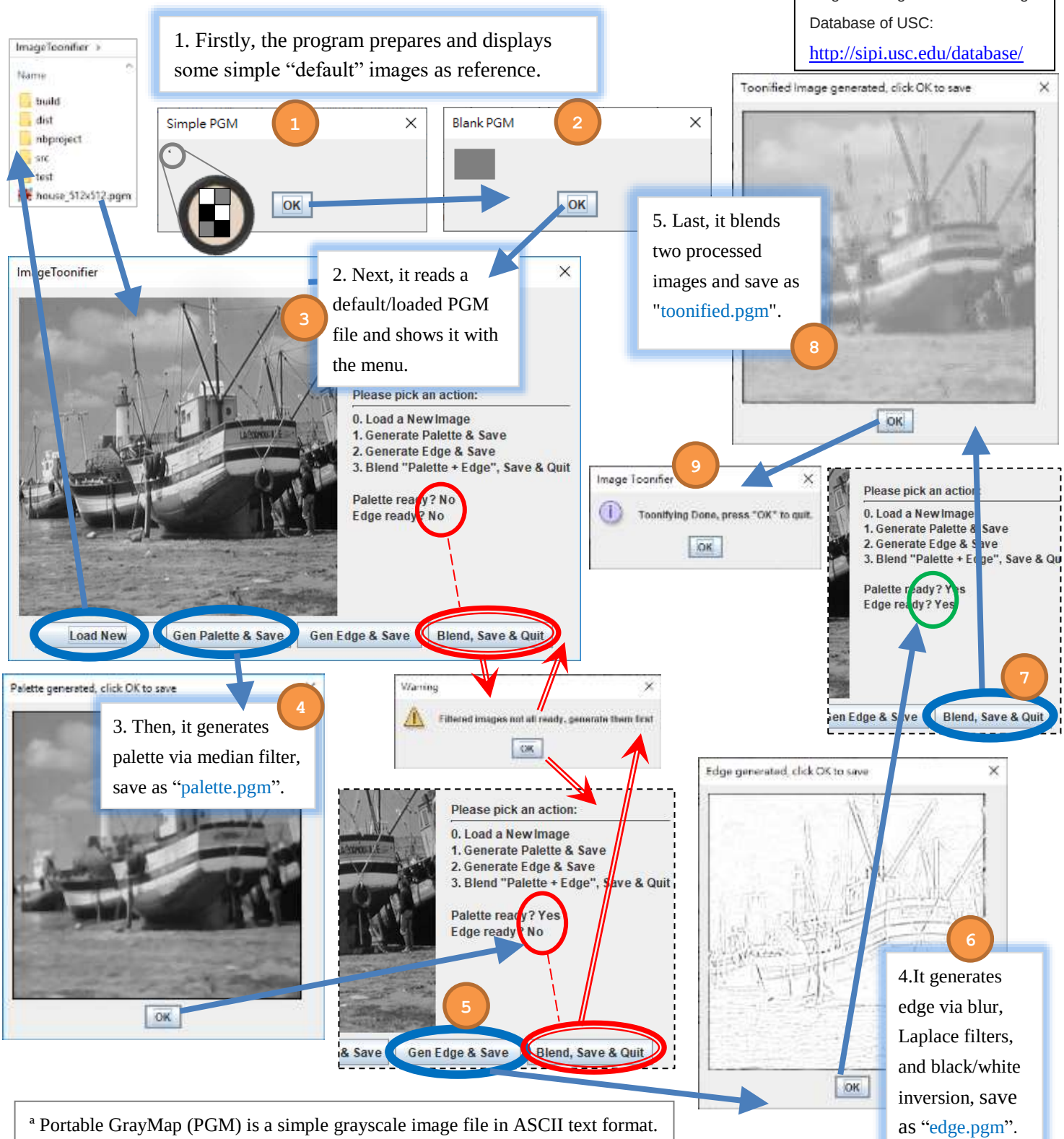
Due date: 2720 November 2019 (Wed) **Assignment 5** Full mark: 100

Expected normal time spent: 8 hours

Image Toon-ifier

Aim: - build an image processing app for to convert PGM [a] images to cartoon-like style ones using Java.
- practise using existing classes/objects and creating our own methods.
- practise file I/O, String and 1D/2D array processing.

All given images from SIPI Image Database of USC: http://sipi.usc.edu/database/



[a] Portable GrayMap (PGM) is a simple grayscale image file in ASCII text format.
Reference: https://en.wikipedia.org/wiki/Netpbm_format#PGM_example

Procedure (Step-by-Step):

1. Create a new NetBeans project **`ImageToonifier`** with a default main class **`imagetoonifier.ImageToonifier`**. Create another class **`PGM`** under the package **`imagetoonifier`**. You just have to complete these two classes.

2. Client class **`ImageToonifier`** is given with various helper methods. You are expected to complete the implementation of **`main()`** method in **`ImageToonifier`** and also another class **`PGM`**.

3. Download given **`ImageToonifierResources.zip`** from Blackboard.

4. Your class **`ImageToonifier`** should include at least the following fields and methods in addition to some other given methods:

```
public class ImageToonifier {
  private static boolean isPaletteReady = false, isEdgeReady = false;

  // Show image on screen
  public void showImageOnly(String title, PGM imgPGM) {
    // given some code to display image with given title
  }

  // Show a menu of choices and get user's input
  public int showImageWithMenu(PGM imgPGM) {

    String menuHTML = "<html>";
    menuHTML += "Please pick an action:<hr>";
    menuHTML += "0. Load a New Image<br>";
    menuHTML += "1. Generate Palette & Save<br>";
    menuHTML += "2. Generate Edge & Save<br>";
    menuHTML += "3. Blend \"Palette + Edge\", Save & Quit<br>";
    menuHTML += "<br>";
    menuHTML += "Palette ready? " + (isPaletteReady ? "Yes" : "No") + "<br>";
    menuHTML += "Edge ready?    " + (isEdgeReady ? "Yes" : "No") + "<br>";
    menuHTML += "</html>";
    String[] options = {"Load New", "Gen Palette & Save",
         "Gen Edge & Save", "Blend, Save & Quit"};
    // given some code to display image, handle and return user's selected option
  }

  // Show a file dialog and get user's input of filename
  public static String getNameFromFileDialog() {
    // given some code to return user's selected file from a file dialog
  }

  public static void main(String[] args) {
    ImageToonifier toonifier = new ImageToonifier();
```

```
    PGM imgDefault;
    imgDefault = new PGM();
    toonifier.showImageOnly("Simple PGM", imgDefault);

    PGM imgBlank;
    imgBlank = new PGM("Mid Gray", 40, 30);
    toonifier.showImageOnly("Blank PGM", imgBlank);

    String filename = "boat_316x316.pgm"; // default file at start
    PGM imgFileCurrent;
    imgFileCurrent = new PGM(filename);

    /*** student's work here to repeatedly obtain and process user's input ***/
  }
}
```

5. Your **`main()`** method in **`ImageToonifier`** should do the followings:
   a) Create a new **`ImageToonifier`** object.
   b) Display two initial PGM files using a method **`showImageOnly()`** and the given image file "**`boat_316x316.pgm`**" using another method **`showImageWithMenu()`**.
   c) Request filename from file dialog calling method **`getNameFromFileDialog()`** as needed and reset flags.
   d) Process images and repeatedly call **`showImageWithMenu()`** and **`showImageOnly()`**.

   Note that file operations and image displaying may fail. Thus, methods dealing with such operations should try-catch and silent all related Exceptions (i.e., to avoid run-time failure and do not propagate.)

6. Your class **`PGM`** should include at least the following fields and methods:

```
public class PGM {
  // instance fields
  private String imageName;
  private int width, height;
  private int maxValue;
  private int[][] image;

  // Default constructor for creating a simple checker PGM image of 2 x 3
  public PGM() {
    imageName = "Simple";
    width = 2;
    height = 3;
    maxValue = 255;                    // value for white in default/given file
    image = new int[height][width];    // note: height as row, width as column
    image[0][0] = image[1][1] = 255;  // white dots
    image[0][1] = image[2][0] = 127;  // gray dots
    image[1][0] = image[2][1] = 0;    // white dots
  }
```

```
  // Constructor for creating a "mid-gray" PGM image of w x h
  // All pixels shall carry value of 127
  public PGM(String name, int w, int h) {
    /*** student's work here to construct a gray box ***/
  }

  // Constructor for reading a PGM image file
  public PGM(String filename) {
    this.imageName = filename;
    read(filename);
  }

  public int getWidth() { return width; }
  public int getHeight() { return height; }
  public int[][] getImage() { return image; }

  public void read(String filename) {
    try {
      File f = new File(filename);
      Scanner reader = new Scanner(f);
      String header = reader.nextLine();

      if (header == null || header.length() < 2 ||
          header.charAt(0) != 'P' || header.charAt(1) != '2') {
        throw new Exception("Wrong PGM header!");
      }

      do { // skip lines start with '#' (if any)
        header = reader.nextLine();
      } while (header.charAt(0) == '#');

      // get from last line instead of file
      Scanner readStr = new Scanner(header);
      width = readStr.nextInt();
      height = readStr.nextInt();

      // get the rest from file
      maxValue = reader.nextInt();
      /*** student's work here to read the image ***/
    } catch (...){
      /*** student's work here to handle exception(s) ***/
    }
  }

  public void write(String filename) {
    /*** student's work here to write the PGM image ***/
  }
}
```

7. Here is an example of a standard PGM file format, *free from additional features* for comment:

```
P2
24 30
255
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  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 15 15 15 15  0…
…
```

- The first two characters on the first line must be "**P2**".
- Then 3 numbers follow: *width* and *height* define the size of the image; *max_value* is usually 255 which indicates maximum possible value of a pixel component value.
- Subsequent values defining an image in grayscale. All values shall fall within 0 and *max_value*, i.e., usually within 0 – 255.
- There are *height* pixel lines run from top to bottom row-by-row. Each pixel line contains *width* number of pixels.
- All numbers are delimited by white-spaces (including space, tab and newline.)
- Java `Scanner` class is well suited for reading the PGM file in plain text ASCII format.

8. The *origin* of the image and window coordinates system is always at the *top-left corner*.

9. Apply the filters on the image, i.e., performing *median filtering* or *convolution*-based operations including *box-blur* and *Laplace*. We use 3×3 filters in all our processing. See **Appendix** for the definition and process of 2D convolution of matrices and default values of different filter operations. Note that the current image shall **NOT** be updated after filtering, just a new image is created.

   a) To generate **palette** of the image, we apply a median filter on the current image by implementing:

```
public PGM palette() {
    /*** student's work here to generate palette by median filter ***/
}

// define some help methods e.g. PGM median() { //... }
```

   Results would be better by applying multiple passes but only once is required here; newly filtered image should be saved into a file named "`palette.pgm`", and then go back to the menu to show the original image.

   b) To generate **edge** of the image, we first apply a box blur filter on the current image and further apply another Laplace filter on the intermediate image by implementing:

```
public PGM edge() {
  /*** student's work here to generate edge by convolution-based filters ***/
  // define some kernel here e.g. double[][] kernel_blur, kernel_laplace;
  // invert the intensity value i.e new_value = 255 - old_value;
}

// define some help methods e.g. PGM convolute(...) { //... }
```

   Generated edges are white on top of black background, so inverting black and white is needed. Save the newly generated image into a file named "`edge.pgm`" and go back to the menu again.

10. Stepwise process for filtering an image is given as follows:
    a) Starting from top-left corner of the image, **slide** the filter, from top-to-bottom, left-to-right. to **overlap on every sub-area** of the image. Sub-area dimensions are identical to the dimensions of the filter.
    b) For convolution-based operations, let the filter to **blend on every sub-area** of the image. Each blend operation shall generate a new pixel value by summing up the products of each pair of corresponding filter element and pixel.
    c) Clip the new pixel value: if the value < 0, set it to 0; if the value > 255, set it to 255.
    d) Store the clipped pixel value at the top-left corner of a sub-area in the resultant image.
    e) When the filter is applied to border pixels, part of the filter is covering out-of-bound area of image. In this case, we assume a pixel value of **ZERO** for any outside area. Examples are given in Appendix.

11. Blend the two Palette and Edge images by outputting average of pixel value from both inputs. Warning should be given if either of the filtered images is not ready; otherwise save the blended image into a file named "`toonified.pgm`".

12. Save means saving the current image to an ASCII PGM file which should usually reside in your NetBeans application project folder by default, in the PGM format, again *free from additional features* such as comment.

Submission:

1. Locate your NetBeans project folder, e.g., **`C:\Users\your_name\Documents\NetBeansProjects\ImageToonifier\`**.

2. ZIP the whole NetBeans project folder **`ImageToonifier`** and Submit the file **`ImageToonifier.zip`** via our online Assignment Collection Box on Blackboard.

Marking Scheme and Notes:

1. The submitted program should be free of any typing mistakes, compilation errors and warnings.
2. Comment/remark, indentation, style are under assessment in every programming assignments unless specified otherwise. Variable naming, proper indentation for code blocks and adequate comments are important. Include also your personal particulars and your academic honesty declaration statement in a header comment block of each source file.

3. Remember to upload your submission and **click Submit** before 18:00 p.m. of the due date. No late submission would be accepted.

4. If you submit multiple times, ONLY the content and timestamp of the latest one would be counted. You may delete (i.e. take back) your attached file and re-submit. We ONLY take into account the last submission.

University Guideline for Plagiarism:

Attention is drawn to University policy and regulations on honesty in academic work, and to the disciplinary guidelines and procedures applicable to breaches of such policy and regulations. Details may be found at http://www.cuhk.edu.hk/policy/academichonesty/.

With each assignment, students are required to submit a statement that they are aware of these policies, regulations, guidelines and procedures, in a header comment block.

Faculty of Engineering Guidelines to Academic Honesty:

MUST read: https://www.erg.cuhk.edu.hk/erg/AcademicHonesty
(you may need to access via CUHK campus network/ CUHK1x/ CUHK VPN)

Further Possibilities:

1. Can you upgrade the toon-ifying capability by using larger filter kernel, e.g. 5x5, or applying other combinations of filter: e.g. palette via multi-pass median, or blend multiple images via min filter?

2. The current application works on gray scale images only. Can you extend it to color image, for example to support PPM. Note that applying different image filters on a color image could be tricky; for example, would median or mean being applied to individual R, G & B channels still make sense?

Appendix: Description and Examples on Filters and Convolution

Entity & Description	**Notations & Examples**
Blend two Matrices $\oplus$ Sum of the entry-wise product of each pair of corresponding elements from two matrices of identical dimensions	$\begin{bmatrix} m_a & m_b \\ m_c & m_d \end{bmatrix} \oplus \begin{bmatrix} i_a & i_b \\ i_c & i_d \end{bmatrix} = m_a \cdot i_a + m_b \cdot i_b + m_c \cdot i_c + m_d \cdot i_d$
Matrix Convolution $\circledast$ Convolution generates a new 2D image by sliding and blending a filter over all possible sub-areas of an original image. Assuming zero value in out-of-image region, we can pad zero around border before blending.	$\begin{bmatrix} m_a & m_b & m_c \\ m_d & m_e & m_f \\ m_g & m_h & m_i \end{bmatrix} \circledast \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix}$, Assuming Zero pixel value out-of-image, we have: $= \begin{bmatrix} m_a & m_b & m_c \\ m_d & m_e & m_f \\ m_g & m_h & m_i \end{bmatrix} \circledast \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 3 & 0 \\ 0 & 4 & 5 & 6 & 0 \\ 0 & 7 & 8 & 9 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$ while the entry blending is only applied within image $= \begin{bmatrix} \begin{bmatrix} m_a & m_b & m_c \\ m_d & m_e & m_f \\ m_g & m_h & m_i \end{bmatrix} \oplus \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 4 & 5 \end{bmatrix} & \begin{bmatrix} m_a & m_b & m_c \\ m_d & m_e & m_f \\ m_g & m_h & m_i \end{bmatrix} \oplus \begin{bmatrix} 0 & 0 & 0 \\ 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} & \begin{bmatrix} m_a & m_b & m_c \\ m_d & m_e & m_f \\ m_g & m_h & m_i \end{bmatrix} \oplus \begin{bmatrix} 0 & 0 & 0 \\ 2 & 3 & 0 \\ 5 & 6 & 0 \end{bmatrix} \\ \begin{bmatrix} m_a & m_b & m_c \\ m_d & m_e & m_f \\ m_g & m_h & m_i \end{bmatrix} \oplus \begin{bmatrix} 0 & 1 & 2 \\ 0 & 4 & 5 \\ 0 & 7 & 8 \end{bmatrix} & \begin{bmatrix} m_a & m_b & m_c \\ m_d & m_e & m_f \\ m_g & m_h & m_i \end{bmatrix} \oplus \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} & \begin{bmatrix} m_a & m_b & m_c \\ m_d & m_e & m_f \\ m_g & m_h & m_i \end{bmatrix} \oplus \begin{bmatrix} 2 & 3 & 0 \\ 5 & 6 & 0 \\ 8 & 9 & 0 \end{bmatrix} \\ \begin{bmatrix} m_a & m_b & m_c \\ m_d & m_e & m_f \\ m_g & m_h & m_i \end{bmatrix} \oplus \begin{bmatrix} 0 & 4 & 5 \\ 0 & 7 & 8 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} m_a & m_b & m_c \\ m_d & m_e & m_f \\ m_g & m_h & m_i \end{bmatrix} \oplus \begin{bmatrix} 4 & 5 & 6 \\ 7 & 8 & 9 \\ 0 & 0 & 0 \end{bmatrix} & \begin{bmatrix} m_a & m_b & m_c \\ m_d & m_e & m_f \\ m_g & m_h & m_i \end{bmatrix} \oplus \begin{bmatrix} 5 & 6 & 0 \\ 8 & 9 & 0 \\ 0 & 0 & 0 \end{bmatrix} \end{bmatrix}$

<u>Box Blur Filter (3 × 3)</u> Blur filtering on $P_{i,j}$ based on pixel values within a 3x3 region surrounding target pixel	$B = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$, $Pixels\ (P) = \begin{bmatrix} 1 & 2 & 3 \\ 8 & 9 & 4 \\ 7 & 6 & 5 \end{bmatrix}$ Let $B_{i,j}$ be filtered value of applying Blur filter B to $P_{i,j}$, Eg.1. $B_{1,1} = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} \oplus \begin{bmatrix} 1 & 2 & 3 \\ 8 & 9 & 4 \\ 7 & 6 & 5 \end{bmatrix} = \frac{1+2+3+8+9+4+7+6+5}{9} = 5$ Eg. 2. $B_{1,0} = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix} \oplus \begin{bmatrix} 0 & 1 & 2 \\ 0 & 8 & 9 \\ 0 & 7 & 6 \end{bmatrix} = \frac{0+1+2+0+8+9+0+7+6}{9}$ $= 3.666667$
<u>Laplace Filter (3 × 3)</u> Laplace filtering on $P_{i,j}$ based on pixel values within a 3x3 region surrounding target pixel	$L = \begin{bmatrix} -1 & -1 & -1 \\ -1 & +8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$, $Pixels\ (P) = \begin{bmatrix} 1 & 2 & 3 \\ 8 & 9 & 4 \\ 7 & 6 & 5 \end{bmatrix}$ Let $L_{i,j}$ be filtered value of applying Laplace filter L to $P_{i,j}$, Eg.1. $L_{1,1} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & +8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \oplus \begin{bmatrix} 1 & 2 & 3 \\ 8 & 9 & 4 \\ 7 & 6 & 5 \end{bmatrix} = -1-2-3-8+9\times 8-4-7-6-5 = 36$ Eg. 2. $L_{1,0} = \begin{bmatrix} -1 & -1 & -1 \\ -1 & +8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \oplus \begin{bmatrix} 0 & 1 & 2 \\ 0 & 8 & 9 \\ 0 & 7 & 6 \end{bmatrix} = 0-1-2-0+8\times 8-9-0-7-6 = 39$
<u>Median Filter (3 × 3)</u> Median of pixel values within a 3x3 region inclusive of target pixel $P_{i,j}$	$Pixels\ (P) = \begin{bmatrix} 1 & 2 & 3 \\ 8 & 9 & 4 \\ 7 & 6 & 5 \end{bmatrix}$ Let $Median_{i,j}(P)$ be the median of 9 elements surrounding $P_{i,j}$ Eg.1. $Median_{1,1}(P) = Median(\begin{bmatrix} 1 & 2 & 3 \\ 8 & 9 & 4 \\ 7 & 6 & 5 \end{bmatrix}) = Median(1,2,3,8,9,4,7,6,5)$ $After\ sorting\ the\ values, we\ have$: $= Median(1,2,3,4,5,6,7,8,9) = 5$ Eg.2. $Median_{0,0}(P) = Median(\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 8 & 9 \end{bmatrix}) = Median(0,0,0,0,1,2,0,8,9)$ $After\ sorting\ the\ values, we\ have$: $= Median(0,0,0,0,0,1,2,8,9) = 0$