Criterion C: Development

Classes

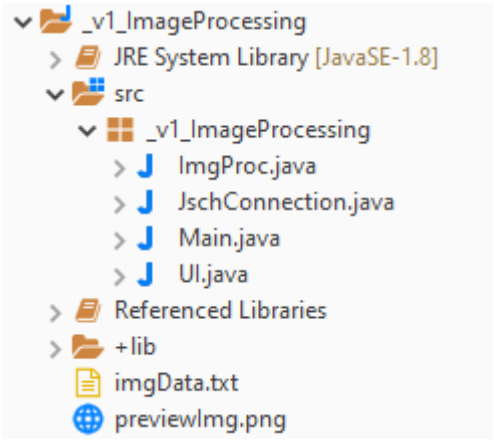
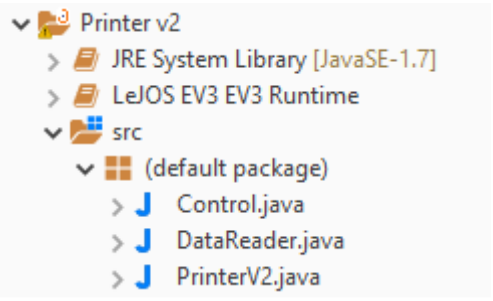


Image Processing



Printing System

Techniques used

- **Image processing**
  - 2D array – store pixel data
  - BufferedImage – read image file and create new image
  - BufferedWriter – save new image
- **Dithering algorithms and error diffusion**
  - Complex loops
- **SSH file transfer using JSch**
  - File handling
- **Java Swing UI**

Image Processing

Class: ImgProc

The chosen image is reformatted in this class. To be able to work with the image it is converted into a 2D array containing the values of each pixel.

```
15  public class ImgProc {
16
17
18      BufferedImage rescaledImg;
19
20      //Maximum resolution
21      final int MAXHEIGHT = 500;
22      final int MAXWIDTH  = 200;
23
24      int imgHeight;  //Number of Rows
25      int imgWidth;   //Number of Columns
26
27      //Greyscale value => Range (0-255)
28      //0 = Black, 255 = White
29      int[][] imgValues = null;
30
31      //1 = Black, 0 = White
32      int[][] binaryImg = null;
33
```

Class variables

imgValues stores the greyscale values of the chosen image.

binaryImg stores the values of the image after the dithering process, each pixel consists of one of two colours, black or white. This is the image that is printed.

The method **generateRescaledImage** takes the chosen image and resizes it using a library called Thumbnailator[1] and then saves the image to the BufferedImage rescaledImg.

```
185     public void generateRescaledImage(File file) {
186
187         try {
188
189             BufferedImage tmpBfImg = ImageIO.read(file);
190
191             int tmpHeight = tmpBfImg.getHeight();
192             int tmpWidth  = tmpBfImg.getWidth ();
193
194             //Resize image to fit into MAXHEIGHT and MAXWIDTH
195             if ((double) MAXHEIGHT / tmpBfImg.getHeight() < (double) MAXWIDTH / tmpBfImg.getWidth()) {
196                 rescaledImg = Thumbnails.of(tmpBfImg)
197                             .scale((double) MAXHEIGHT/tmpHeight)
198                             .asBufferedImage();
199             } else {
200                 rescaledImg = Thumbnails.of(tmpBfImg)
201                             .scale((double) MAXWIDTH/tmpWidth)
202                             .asBufferedImage();
203             }
204
205         } catch (IOException e) {
206             e.printStackTrace();
207         }
208     }
```

[1] Coobird. Thumbnailator. https://github.com/coobird/thumbnailator. (accessed 12 June 2020)

The BufferedImage rescaledImg is then used in **generateImgValues** to generate the greyscale image and store it in imgValues. The brightness (chosen in UI) is also considered in the calculation.

```
212    private void generateImgValues(int brightness) {
213
214        //Range (0 - 255)
215        //0   = Black
216        //255 = White
217
218        imgHeight = rescaledImg.getHeight();
219        imgWidth  = rescaledImg.getWidth();
220
221        imgValues = new int [imgHeight][imgWidth];
222        binaryImg = new int [imgHeight][imgWidth];
223
224        for (int row = 0; row < imgHeight; row++) {
225            for (int col = 0; col < imgWidth; col++) {
226
227                Color curColor = new Color(rescaledImg.getRGB(col, row));
228
229                //Calculates greyscale pixel value (0-255) with brightness taken into account
230                imgValues[row][col] = (int) (  (curColor.getRed() + curColor.getGreen() + curColor.getBlue()) / 3.0  + (765.0/50.0)*(brightness-50) + 0.5);
231
232                //Makes sure to cap imgValues if the brightness makes them exceed the range (0-255)
233                if      (imgValues[row][col] > 255) imgValues[row][col] = 255;
234                else if (imgValues[row][col] < 0)   imgValues[row][col] = 0;
235
236            }
237        }
238    }
```

These values are then used by the dithering algorithms to generate the binary image that is stored in the 2D array binaryImg.

Dithering Algorithms

Dithering is an image processing technique used to simulate shading. Due the limited colour palette of the printer I needed to be able to convert full colour images into binary images. The coloured image is first converted to a greyscale image (Image 1) and further techniques are used to generate the binary image. The easiest technique is “Thresholding” (Image 2), here each pixel is compared against a fixed threshold, though this results in considerable loss of detail. Other techniques are “Error Diffusion dithering” (Image 3) and “Randomized dithering” (Image 4), both being able to simulate shading. “Randomized dithering” is similar to “Thresholding” except that the threshold that each pixel is compared to is randomized.

Conversion techniques on a gradient

Image 1: Greyscale

Image 2: Thresholding

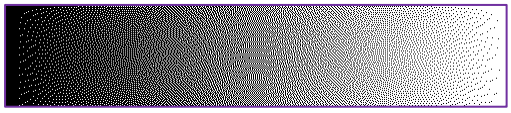


Image 3: Error Diffusion dithering (Floyd-Steinberg kernel)

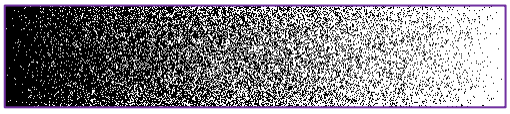


Image 3: Randomized dithering

Conversion techniques on an example image

Image 1: Greyscale

Image 2: Thresholding

Image 3: Error Diffusion dithering (Floyd-Steinberg kernel)

Image 3: Randomized dithering

Error Diffusion

Error-diffusion works by distributing the **error** ("difference between the exact pixel value from the original image and the approximated value being displayed in the result"[2]) to neighbouring pixels. This is beneficial as the decision regarding the next pixel "now includes both the original image intensity and the errors that have been added from previously processed pixels"[3]. This introduces a "kind of "smoothing" into the dithered image"[4]. By distributing this **error** to neighbouring pixels, shading can be simulated using only two colours, black and white.

---

[2] Ping Wah Wong. 2005. Image Quantization, Halftoning, and Printing. https://www.sciencedirect.com/topics/computer-science/error-diffusion. (accessed 22 June 2020)
[3] Ibid.
[4] Ibid.

Example of error-diffusion:

In the following example grey boxes represent pixels that have already been processed, the blue box represents the current pixel that is being processed. The numbers represent greyscale values ranging from 0 (black) to 255 (white). The closest binary value of the current pixel would be 0, thus the **error** would be 32 (32-0).

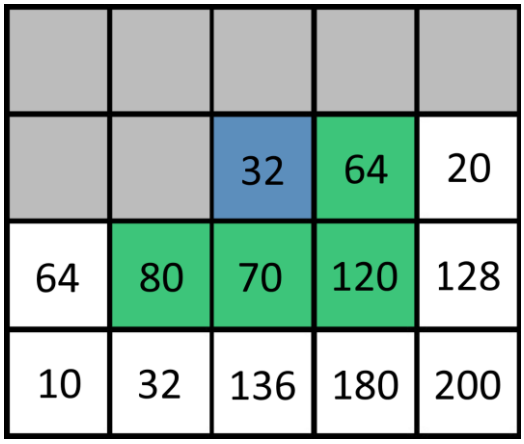


Diagram 1

The **error** is then distributed to neighbouring pixels, how they are distributed is dependent on the kernel you choose (here Floyd-Steinberg kernel).

$$\frac{1}{16}\begin{bmatrix} - & \# & 7 \\ 3 & 5 & 1 \end{bmatrix}$$

Floyd-Steinberg kernel

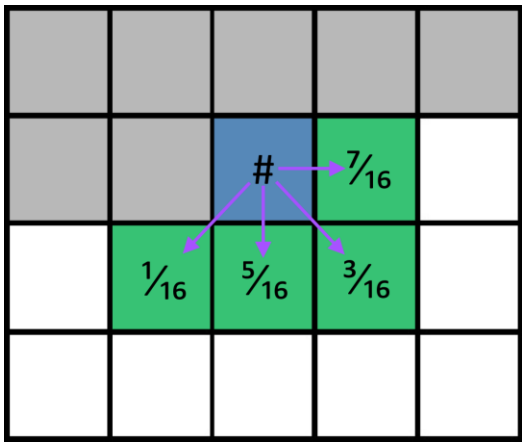


Diagram 2

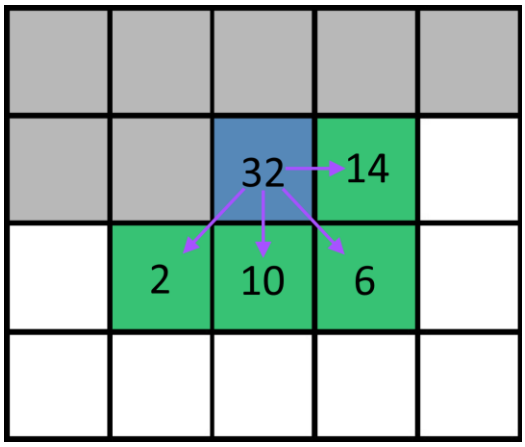


Diagram 3

Green highlighted values from *Diagram 3* are added to the highlighted values in *Diagram 1*.

		0	78	20
64	82	80	126	128
10	32	136	180	200

Diagram 4

The program moves to the next pixel and repeats the process *Diagram 5* until all pixels have been processed *Diagram 6*.

		0	78	20
64	82	80	126	128
10	32	136	180	200

Diagram 5

255	255	0	255	0
0	255	0	0	0
0	0	255	0	255
0	0	255	255	255

Diagram 6

Implemented dithering algorithms

The method **dither** is used to generate the binary images. How the image is converted to binary is determined by nKernel using a switch statement. The value of nKernel is dependent on what conversion technique was chosen in the UI, with the values 0 to 6 each representing a different technique (thresholding, randomized dithering, or the different error diffusion kernels).

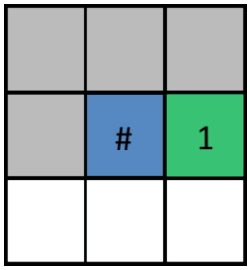
The methods that are executed in the switch statement distributes the error to neighboring pixels in the 2D array imgValues. The parameters row and col define what the current pixel is and the errorMargin defines how much error is distributed. Each method contains a different kernel that distributes the error to different pixels at different ratios.

```
37      //Implemented dithering algorithms
38      private void dither(int nKernel) {
39
40          int errorMargin;
41          int curValue;
42
43          //Special case as user has chosen randomized dithering
44          //This is done in the separate method randomizedThreshold()
45          if (nKernel == 1) {
46              randomizedThreshold();
47              return;
48          }
49
50          for (int row = 0; row < imgHeight; row++) {
51
52              for (int col = 0; col < imgWidth; col++) {
53
54                  curValue = imgValues[row][col];
55
56                  if (curValue <= 127) {          //Black pixel
57                      binaryImg[row][col] = 1;
58                      errorMargin = curValue;
59                  } else {                        //White pixel
60                      binaryImg[row][col] = 0;
61                      errorMargin = curValue - 255;
62                  }
63
64                  //Chooses which diffusion kernel to use
65                  switch ( nKernel ) {
66                  case 0:
67                      //User has chosen thresholding
68                      //No diffusion is required
69                      break;
70                  case 2:
71                      diffusion1D(row, col, errorMargin);
72                      break;
73                  case 3:
74                      diffusion2D(row, col, errorMargin);
75                      break;
76                  case 4:
77                      floydSteinbergDith(row, col, errorMargin);
78                      break;
79                  case 5:
80                      burkesDith(row, col, errorMargin);
81                      break;
82                  case 6:
83                      jarvisJudiceNinkeDith(row, col, errorMargin);
84                      break;
85
86                  }
87              }
88          }
89      }
```

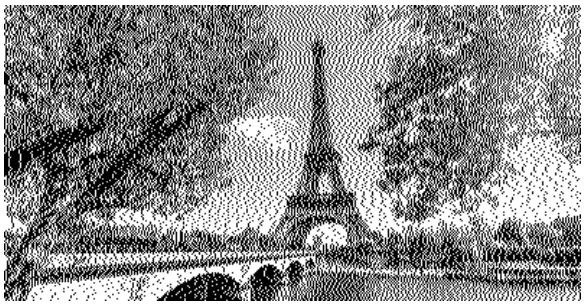
Method: dither

Implemented Error Diffusion kernels

1D diffusion



Kernel:
1D diffusion

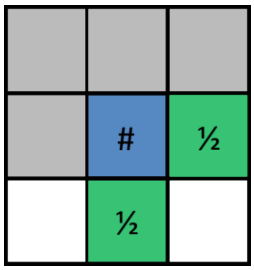


Example: 1D diffusion

```
93      private void diffusion1D(int row, int col, int errorMargin) {
94
95          // 1    | # 1 |
96
97          if (col+1 < imgWidth) imgValues[row][col+1] = imgValues[row][col+1] + errorMargin;
98
99      }
```

Method: diffusion1D

2D diffusion



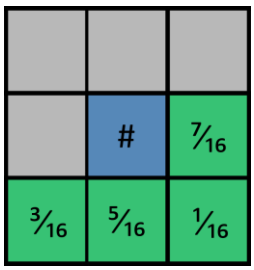
Kernel:
2D diffusion

Example: 2D diffusion

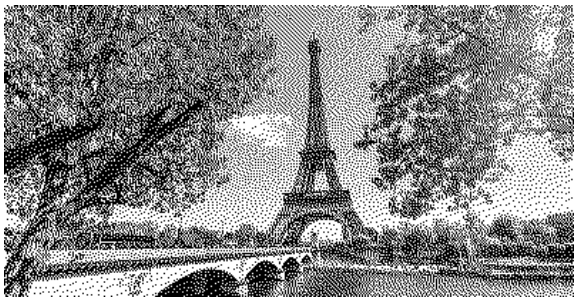
```
101     private void diffusion2D(int row, int col, int errorMargin) {
102
103         // 1/2  | # 1 |
104         //      | 1 0 |
105
106         if (col+1 < imgWidth)  imgValues[row][col+1] = (int) (imgValues[row][col+1] + errorMargin*0.5 + 0.5);
107         if (row+1 < imgHeight) imgValues[row+1][col] = (int) (imgValues[row+1][col] + errorMargin*0.5 + 0.5);
108
109     }
```

Method: diffusion2D

Floyd-Steinberg dithering



Kernel: Floyd-Steinberg dithering

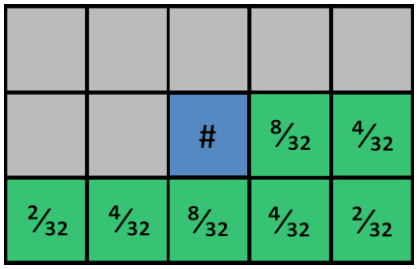


Example: Floyd-Steinberg dithering

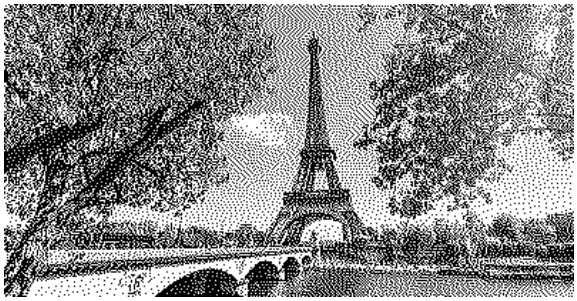
```
111    private void floydSteinbergDith(int row, int col, int errorMargin) {
112
113        // 1/16 | - # 7 |
114        //      | 3 5 1 |
115
116        if (col+1 < imgWidth)                    imgValues[row][col+1]   = (int) (imgValues[row][col+1]   + errorMargin*(7.0/16.0) + 0.5);
117        if (col-1 >= 0 && row+1 < imgHeight)     imgValues[row+1][col-1] = (int) (imgValues[row+1][col-1] + errorMargin*(3.0/16.0) + 0.5);
118        if (row+1 < imgHeight)                   imgValues[row+1][col]   = (int) (imgValues[row+1][col]   + errorMargin*(5.0/16.0) + 0.5);
119        if (col+1 < imgWidth && row+1 < imgHeight) imgValues[row+1][col+1] = (int) (imgValues[row+1][col+1] + errorMargin*(1.0/16.0) + 0.5);
120
121    }
```

Method: floydSteinbergDith

Burkes dithering



Kernel: Burkes dithering



Example: Burkes dithering

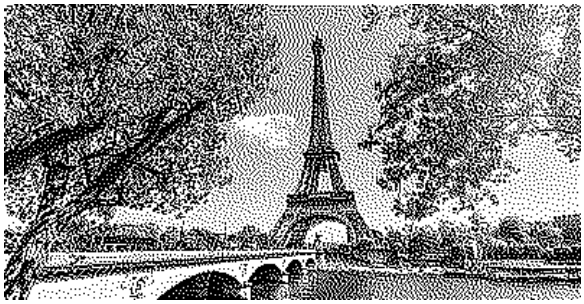
```
123    private void burkesDith(int row, int col, int errorMargin) {
124
125        // 1/32 | - - # 8 4 |
126        //      | 2 4 8 4 2 |
127
128        if (col+1 < imgWidth)                    imgValues[row][col+1]   = (int) (imgValues[row][col+1]   + errorMargin*(8.0/32.0) + 0.5);
129        if (col+2 < imgWidth)                    imgValues[row][col+2]   = (int) (imgValues[row][col+2]   + errorMargin*(4.0/32.0) + 0.5);
130        if (col-2 >= 0 && row+1 < imgHeight)     imgValues[row+1][col-2] = (int) (imgValues[row+1][col-2] + errorMargin*(2.0/32.0) + 0.5);
131        if (col-1 >= 0 && row+1 < imgHeight)     imgValues[row+1][col-1] = (int) (imgValues[row+1][col-1] + errorMargin*(4.0/32.0) + 0.5);
132        if (row+1 < imgHeight)                   imgValues[row+1][col]   = (int) (imgValues[row+1][col]   + errorMargin*(8.0/32.0) + 0.5);
133        if (col+1 < imgWidth && row+1 < imgHeight) imgValues[row+1][col+1] = (int) (imgValues[row+1][col+1] + errorMargin*(4.0/32.0) + 0.5);
134        if (col+2 < imgWidth && row+2 < imgHeight) imgValues[row+1][col+2] = (int) (imgValues[row+1][col+2] + errorMargin*(2.0/32.0) + 0.5);
135
```

Method: burkesDith

Jarvis-Judice-Ninke dithering

		#	7/48	5/48
3/48	5/48	7/48	5/48	3/48
1/48	3/48	5/48	3/48	1/48

Kernel: Jarvis-Judice-Ninke dithering



Example: Jarvis-Judice-Ninke dithering

```
138    private void jarvisJudiceNinkeDith(int row, int col, int errorMargin) {
139
140        // 1/48 | - - # 7 5 |
141        //      | 3 5 7 5 3 |
142        //      | 1 3 5 3 1 |
143
144        if (col+1 < imgWidth)                    imgValues[row][col+1]   = (int) (imgValues[row][col+1]   + errorMargin*(7.0/48.0) + 0.5);
145        if (col+2 < imgWidth)                    imgValues[row][col+2]   = (int) (imgValues[row][col+2]   + errorMargin*(5.0/48.0) + 0.5);
146        if (col-2 >= 0 && row+1 < imgHeight)     imgValues[row+1][col-2] = (int) (imgValues[row+1][col-2] + errorMargin*(3.0/48.0) + 0.5);
147        if (col-1 >= 0 && row+1 < imgHeight)     imgValues[row+1][col-1] = (int) (imgValues[row+1][col-1] + errorMargin*(5.0/48.0) + 0.5);
148        if (row+1 < imgHeight)                   imgValues[row+1][col]   = (int) (imgValues[row+1][col]   + errorMargin*(7.0/48.0) + 0.5);
149        if (col+1 < imgWidth && row+1 < imgHeight) imgValues[row+1][col+1] = (int) (imgValues[row+1][col+1] + errorMargin*(5.0/48.0) + 0.5);
150        if (col+2 < imgWidth && row+2 < imgHeight) imgValues[row+1][col+2] = (int) (imgValues[row+1][col+2] + errorMargin*(3.0/48.0) + 0.5);
151        if (col-2 >= 0 && row+2 < imgHeight)     imgValues[row+2][col-2] = (int) (imgValues[row+2][col-2] + errorMargin*(1.0/48.0) + 0.5);
152        if (col-1 >= 0 && row+2 < imgHeight)     imgValues[row+2][col-1] = (int) (imgValues[row+2][col-1] + errorMargin*(3.0/48.0) + 0.5);
153        if (row+2 < imgHeight)                   imgValues[row+2][col]   = (int) (imgValues[row+2][col]   + errorMargin*(5.0/48.0) + 0.5);
154        if (col+1 < imgWidth && row+2 < imgHeight) imgValues[row+2][col+1] = (int) (imgValues[row+2][col+1] + errorMargin*(3.0/48.0) + 0.5);
155        if (col+2 < imgWidth && row+2 < imgHeight) imgValues[row+2][col+2] = (int) (imgValues[row+2][col+2] + errorMargin*(1.0/48.0) + 0.5);
156
157    }
```

Method: jarvisJudiceNinkeDith

Further Methods

This class contains four more important methods.

binaryToImg converts *binaryImg* into a PNG file that is needed to show a preview of processed image. This is done using a BufferedImage.

```
242    private void binaryToImg() {
243
244        BufferedImage previewBfImg = new BufferedImage(imgWidth, imgHeight, BufferedImage.TYPE_INT_RGB);
245
246        for (int row = 0; row < imgHeight; row++) {
247
248            for (int col = 0; col < imgWidth; col++) {
249                if (binaryImg[row][col] == 1) previewBfImg.setRGB(col, row, Color.BLACK.getRGB());
250                else                          previewBfImg.setRGB(col, row, Color.WHITE.getRGB());
251            }
252        }
253
254        try {
255            ImageIO.write(previewBfImg, "png", new File("./previewImg.png"));
256        } catch (IOException e) {
257            e.printStackTrace();
258        }
259    }
```

Method: binaryToImg

binaryToTxt converts *binaryImg* into a text file (imgData.txt) that contains the data of the processed image. The printer system uses this text file to determine what to print. This is done with a BufferedWriter.

```
263    private void binaryToTxt() {
264
265        BufferedWriter bw = null;
266
267        try {
268
269            File imgDataFile = new File("./imgData.txt");
270
271            if (!imgDataFile.exists()) {
272                imgDataFile.createNewFile();
273            }
274
275            bw = new BufferedWriter(new FileWriter(imgDataFile));
276
277            for (int row = 0; row < imgHeight; row++) {
278
279                for (int col = 0; col < imgWidth; col++) {
280                    bw.write(String.valueOf(binaryImg[row][col]));
281                }
282                bw.newLine();
283            }
284
285        } catch (IOException e) {
286            e.printStackTrace();
287        } finally {
288
289            if(bw!=null) {
290
291                try {
292                    bw.close();
293                } catch (IOException e) {
294                    e.printStackTrace();
295                }
296            }
297        }
298    }
```

Method: binaryToTxt

All these methods are used in conjunction with another in the methods **generatePreviewImage**, the method that is executed when users press the preview button and **generateImgDataFile**, the method that is run when users press the print button.

```
302    public void generatePreviewImage(int dith, int brightness) {
303
304        generateImgValues(brightness);
305        dither(dith);
306        binaryToImg();
307    }
308
309    public void generateImgDataFile(int dith, int brightness) {
310
311        generateImgValues(brightness);
312        dither(dith);
313        binaryToTxt();
314    }
```

Methods: generatePreviewImage and generateImgDataFile

SSH connection and JSch[5]

For the printer to work, it needs to somehow gain access to the data in imgData.txt. To do this I used ssh network protocol to transfer this file from the PC system to the printer. The library JSch[6] allows the integration of such processes into Java programs.

Class: JschConnection

Class variables define the IP-address of the printer as well as the password for root access.

```
14  public class JschConnection {
15
16      private static final String HOST     = "169.254.27.254";
17      private static final String USERNAME = "root";
18      private static final String PASSWORD = "robot";
19
```

Class variables

The method **transferData** uses SCP (Secure Copy Protocol) to transfer *dataFile.txt* to the printer.

```
21      //Transfers imageData.txt to printer
22      public void transferData() {
23
24          SshConnection ssh = null;
25
26          try {
27
28              //Connects to Lejos Printer
29              ssh = new SshConnection(HOST, USERNAME, PASSWORD);
30              ssh.connect();
31
32              System.out.println("Connected to printer...");
33
34              //Transfers file to printer
35              ScpFile scpFile = new ScpFile(new File("./imgData.txt") , "/home/lejos/programs/imgData.txt");
36              ssh.executeTask(new ScpUpload(scpFile));
37
38          } catch (SshException e) {
39              System.out.println(e);
40          } finally {
41              if (ssh != null) ssh.disconnect();
42          }
43      }
```

Method: transferData

The method **runProgram** uses is used to start the printing process. It does this by executing the *Printer.jar* file using *jrun*.

[5] JCraft. 1998. JSch - Java Secure Channel. http://www.jcraft.com/jsch/. (accessed 19 July 2020)
[6] Ibid.

```
46      //Start printer
47      public void runProgram() {
48
49          JSch jsch = null;
50          Session session = null;
51
52          try {
53
54              //Connect to printer
55              jsch = new JSch();
56
57              session = jsch.getSession(USERNAME, HOST, 22);
58              session.setPassword(PASSWORD);
59              session.setConfig("StrictHostKeyChecking", "no");
60              session.connect();
61
62              ChannelExec channelExec = (ChannelExec) session.openChannel("exec");
63
64              //Run jar file on printer to begin printing
65              channelExec.setCommand("jrun -jar /home/lejos/programs/Printer.jar");
66              channelExec.connect();
67
68              channelExec.disconnect();
69
70          } catch (JSchException e) {
71              System.out.println(e);
72          } finally {
73              if (session != null) session.disconnect();
74          }
75      }
76
77
78  }
```

Method: transferData

Both these methods are executed when the user presses the print button.

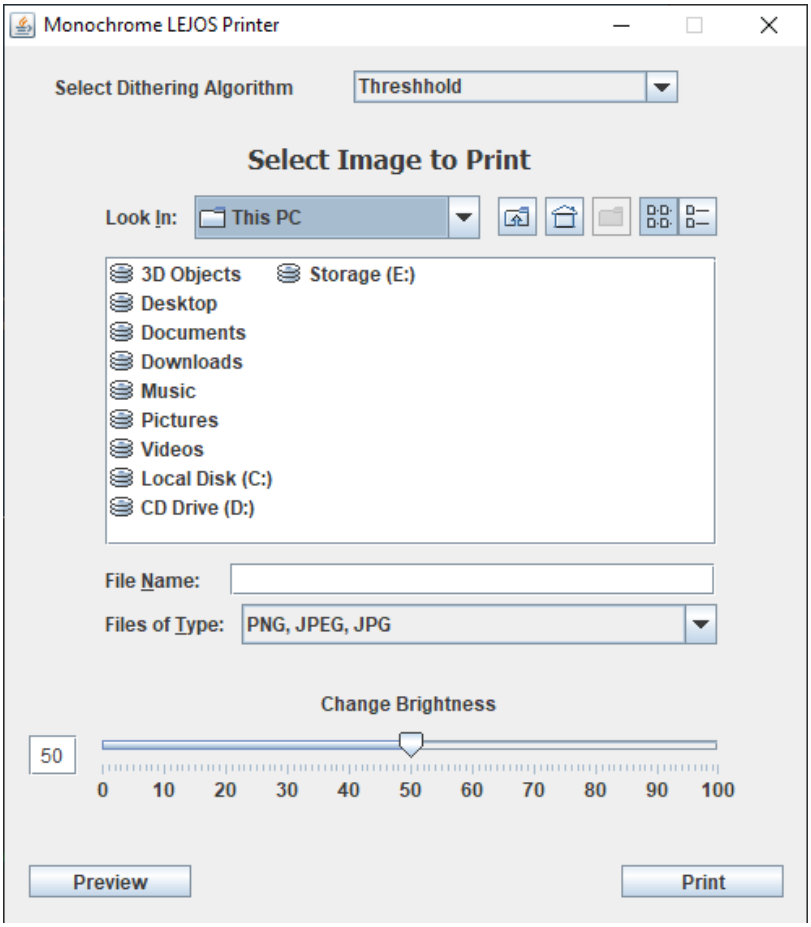
Java Swing User Interface

The UI for the program is implemented using Swing and consists of five main parts: the dithering algorithm selector, image file selector, brightness slider, preview button and print buttons.

The UI class incudes instances of **ImageProc** and **JschConnection**.

```
23  public class UI {
24
25      //Image Processing & Data Transfer
26      ImgProc imgProc = new ImgProc();
27      JschConnection jschConnection = new JschConnection();
28
```

Class Instances



Implemented User Interface

The two important methods in the UI class are **previewImage** and **printImage**, both executed when their respective buttons are pressed. Both methods use the **isImageInvalid** method, to check if the selected file is a valid image file. If file is invalid it returns true and false if invalid. An error pop-up is also triggered if the chosen file is invalid.

```
164    private boolean isImageInvalid(File f) {
165
166        //Error pop-up if user has not selected a file.
167        if (f == null) {
168            JOptionPane.showMessageDialog(frame,
169                "Please choose a file (not a directory).",
170                "Error",
171                JOptionPane.WARNING_MESSAGE);
172            return true;
173        }
174
175        //Gets the file-type of the selected file
176        String extension = "";
177        int i = f.getAbsolutePath().lastIndexOf('.');
178        if (i > 0) extension = f.getAbsolutePath().substring(i+1);
179
180        //Error pop-up if user has not chosen a valid image file-type.
181        if (!extension.equals("png")  && !extension.equals("PNG")  &&
182            !extension.equals("jpg")  && !extension.equals("JPG")  &&
183            !extension.equals("jpeg") && !extension.equals("JPEG"))
184        {
185            JOptionPane.showMessageDialog(frame,
186                "The file you chose is not an image file.\nPlease choose a png, jpg or jpeg.",
187                "Error",
188                JOptionPane.WARNING_MESSAGE);
189            return true;
190        }
191        return false;
192    }
```

Method: isImageInvalid

Method: previewImage

This method uses **generatePreviewImage** from ImgProc to generate the image preview that is then used in the pop-up.

```
199    private void previewImage() {
200
201        int  index = dithChoser.getSelectedIndex();
202        int  brightness = brigtSlider.getValue();
203
204        File file = fileChoser.getSelectedFile();
205
206        //Stops the preview process if the chosen image is invalid
207        if (isImageInvalid(file)) return;
208
209        imgProc.generateRescaledImage(file);
210        imgProc.generatePreviewImage (index, brightness);
211
212        //Pop-up with print preview
213        JFrame imageFrame = new JFrame("Print Preview - " + dithChoser.getSelectedItem() + " Dithering");
214
215        ImageIcon icon = new ImageIcon("./previewImg.png");
216        icon.getImage().flush();
217        //Reload image after flush
218        icon = new ImageIcon("./previewImg.png");
219        JLabel label = new JLabel(icon);
220
221        imageFrame.add(label);
222        imageFrame.pack();
223        imageFrame.setResizable(false);
224        imageFrame.setDefaultCloseOperation(WindowConstants.DISPOSE_ON_CLOSE);
225        imageFrame.setVisible(true);
226    }
```

Method: previewImage

Method: printImage

This method uses **generateImgDataFile** from ImgProc to generate the *imgData.txt* file. Then both **transferData** and **runProgram** methods from JschConnection are executed to transfer the *imgData.txt* file to the printer and to start it.

```
223    private void printImage() {
224
225        int  index = dithChoser.getSelectedIndex();
226        int  brightness = brigtSlider.getValue();
227        File file = fileChoser.getSelectedFile();
228
229        //Stops the printing process if the chosen image is invalid
230        if (isImageInvalid(file)) return;
231
232        imgProc.generateRescaledImage(file);
233        imgProc.generateImgDataFile  (index, brightness);
234
235        System.out.println("Connecting...");
236
237        //Send Data to Printer
238        jschConnection.transferData();
239
240        System.out.println("Printing...");
241
242        //Run Printer
243        jschConnection.runProgram();
244    }
```

Method: printImage