# Course IPJava: Image Processing with Java
# Chapter C2: The Histo Project





## Projekt histo1 mit leerem Fenster

Im Directory `C:\temp` ein Sub-Directory `histo1` anlegen.
TextPad starten `Datei Neu`
Leere Datei speichern: `Datei Speichern unter C:\temp\histo1\Histo1.java`
Schreiben Sie folgende Zeilen:

```
import java.awt.Container;
import java.awt.BorderLayout;
import java.awt.Color;
import javax.swing.JFrame;

public class Histo1
{
  public static void main(String[] args)
  {
    JFrame frame=new JFrame("Histo1");
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setSize(800,600);
    Container cp=frame.getContentPane();
    cp.setBackground(Color.WHITE);
    cp.setLayout(new BorderLayout());
    frame.setVisible(true);
  }
}
```

Übersetzen mit der Tastenkombination `Strg+1`. Ausführen mit der Tastenkombination `Strg+2`. Es öffnet sich ein Fenster, das sich nur minimieren, maximieren und schließen lässt. Die Klasse `Histo1` dient also lediglich dem Aufbau unserer GUI und (später) allen notwendigen Objekt-Instantiierungen.

## Zeichenfläche

Beenden Sie das Programm.
Um mit dem später eingelesenen Bild arbeiten zu können (es zum Beispiel auf die Grö;ße des Fensters zu strecken), benötigen wir eine "Zeichenfläche", die wir als zusätzliche Komponente in unser `JFrame` einbauen und dessen Größe wir jedes mal neu erfassen, wenn sich die Größe des Fensters verändert.
Legen Sie dazu eine neue Datei in TextPad an: `Datei Neu`
Leere Datei speichern: `Datei Speichern unter C:\temp\histo1\DrawArea.java`
Schreiben Sie folgende Klasse:

```
import java.awt.*;
import java.awt.event.*;
import java.awt.image.BufferedImage;
import java.awt.image.IndexColorModel;
import javax.swing.JComponent;

public class DrawArea extends JComponent
{
  final int HISTO_FIELD_W=257;
  final int HISTO_FIELD_H=102;
  BufferedImage image;
  BufferedImage oneBitImage;
```

```
  int imgWidth, imgHeight;
  int myWidth, myHeight, fontHeight=0;
  String message="Open a .bmp file";
  int[] histogram=new int[256];
  int[] grayArray;//gray values of all pixels
  int[] blackAndWhite;
  boolean clickInHisto;
  public DrawArea()
  {
    Dimension size=getSize();
    myWidth=size.width;
    myHeight=size.height;
    addComponentListener(new ComponentAdapter()
    {
      public void componentResized(ComponentEvent ev)
      {
        Dimension size=getSize();
        myWidth=size.width;
        myHeight=size.height;
      }
    });
    addMouseListener(new MouseAdapter()
    {
      public void mousePressed(MouseEvent ev) {}
      public void mouseReleased(MouseEvent ev) {}
    });
    addMouseMotionListener(new MouseMotionAdapter()
    {
      public void mouseDragged(MouseEvent ev) {}
    });
  }
  public void paintComponent(Graphics g)
  {
    super.paintComponent(g);
    if(fontHeight==0)
      fontHeight=g.getFont().getSize();
    if(image!=null)
    {
      g.drawImage(image, 0, 0, myWidth, myHeight, this);
      drawHisto(g);
    }
    else
    {
      g.setColor(Color.RED);
      g.drawString(message, 0, fontHeight);
      g.setColor(Color.BLACK);
    }
  }
  public void setImage(int[] img, int[] palette, int colorDepth,
      int width, int height)
  {
  }
  public void setMessage(String message)
  {
  }
  private void drawHisto(Graphics g)
  {
  }
  private void setOneBitImagePixels(int threshold)
  {
  }
  private void drawOneBitImage(Graphics g, int threshold)
  {
  }
}
```

Übersetzen mit `Strg+1`.
Es wäre ebenso möglich die neue Klasse in die Datei `Histo1.java` zu schreiben und nicht in ein extra File zu speichern. Allerdings ist eine zusätzliche Datei wesentlich übersichtlicher.
Nun müssen wir unserem `JFrame` noch eine Instanz der neuen Klasse als GUI-Komponente zuweisen.
Wechseln Sie dazu in TextPad zur Datei `Histo1`: `Fenster 1 C:\...\Histo1.java`
Ergänzen Sie die Methode `public static void main(String[] args)` von `Histo1`, so dass sie so aussieht:

```
    public static void main(String[] args)
    {
      JFrame frame=new JFrame("Histo (Histogramm und Binarisierung)");
      DrawArea drawArea=new DrawArea();
      frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frame.setSize(800,600);
      Container cp=frame.getContentPane();
      cp.setBackground(Color.WHITE);
      cp.setLayout(new BorderLayout());
      cp.add(drawArea);
      frame.setVisible(true);
    }
```

Übersetzen mit `Strg+1`, ausführen mit `Strg+2`. Sie werden feststellen, dass sich (bis auf den zur Zeit noch bedeutungslosen Text `Open a .bmp file`) das Aussehen und Verhalten der Applikation überhaupt nicht verändert hat.
**Wichtig:**
**Wenn die Java Runtime Environment eine Java-Klasse ausführt, dann sucht sie automatisch nach der `main()`-Methode in dieser Klasse. Führen Sie immer nur die Klasse mit der `main()`-Methode aus - in unserem Fall also immer nur die Klasse `Histo1` - ansonsten erhalten Sie eine Fehlermeldung.**

# Menüleiste

Beenden Sie das Programm.
Um nun `*.bmp` Dateien auf unsere Zeichenfläche laden zu können, benötigen wir eine Menüleiste mit dem Menü `File` und dem Menüpunkt `Read`.
Legen Sie dazu eine neue Datei in TextPad an: `Datei Neu`
Leere Datei speichern: `Datei Speichern unter C:\temp\histo1\MenuBar.java`
Schreiben Sie folgende zwei Klassen:

```
import java.awt.Graphics;
import java.awt.Color;
import java.awt.event.ActionListener;
import java.awt.event.ActionEvent;
import java.awt.event.KeyEvent;
import javax.swing.*;
import javax.swing.filechooser.FileFilter;
import java.io.*;

public class MenuBar extends JMenuBar
{
  JMenu fileMenu;
  public MenuBar(DrawArea drawArea)
  {
    add(fileMenu=new FileMenu(drawArea));
  }
}
//============================================================================
```

```
class FileMenu extends JMenu
{
  static public final String READ_ITEM_CMD="readFile";
  static public final String EXIT_ITEM_CMD="exitProgram";
  JMenuItem readItem, exitItem;
  DrawArea drawArea;
  public FileMenu(DrawArea draw)
  {
    super("File");
    drawArea=draw;
    setMnemonic(KeyEvent.VK_F);
    ActionListener listener=new MenuActionListener();
    readItem=new JMenuItem("Read", KeyEvent.VK_R);
    exitItem=new JMenuItem("Exit");
    readItem.setAccelerator(KeyStroke.getKeyStroke("control R"));
    readItem.setActionCommand(READ_ITEM_CMD);
    exitItem.setActionCommand(EXIT_ITEM_CMD);
    readItem.addActionListener(listener);
    exitItem.addActionListener(listener);
    add(readItem);
    add(exitItem);
  }
  //-------------------------------------------------------------------------
  class MenuActionListener implements ActionListener
  {
    public void actionPerformed(ActionEvent ev)
    {
      String cmd=ev.getActionCommand();
      if(cmd.equals(FileMenu.READ_ITEM_CMD))
      {
      }
      else if(cmd.equals(FileMenu.EXIT_ITEM_CMD))
        System.exit(0);
    }
  }
}
```

Übersetzen mit `Strg+1`.

Nun müssen wir (wie vorher die `DrawArea`) unserem `JFrame` noch die Menüleiste zuweisen.

Wechseln Sie dazu in TextPad zur Datei `Histo1`: `Fenster 1 C:\...\Histo1.java`

Fügen Sie die folgende Code-Zeile als 3. Anweisung (unter die Instantiierung der `drawArea`) in die Methode `public static void main(String[] args)` von `Histo1` ein:

```
    frame.setJMenuBar(new MenuBar(drawArea));
```

Übersetzen mit `Strg+1`, ausführen mit `Strg+2`. Erproben Sie das neue Menü. Allerdings funktioniert bisher nur der Menüpunkt `Exit`.

# Bild lesen und anzeigen

Beenden Sie das Programm.
Wechseln Sie in TextPad zur Datei `DrawArea`: `Fenster 2 C:\...\DrawArea.java`
Erweitern Sie die Methode `public void setImage(...)` der Klasse `DrawArea`, bis sie so aussieht:

```
public void setImage(int[] img, int[] palette, int colorDepth,
    int width, int height)
{
  for(int i=0; i<256; i++)
    histogram[i]=0;//empty histogram
  int gray;
  imgWidth=width;
  imgHeight=height;
  if(colorDepth==8)
  {
    int colorCount=palette.length/3;
    byte[] red=new byte[colorCount];
    byte[] green=new byte[colorCount];
    byte[] blue=new byte[colorCount];
    for(int i=0; i<colorCount; i++)
    {
      red[i]=(byte)palette[i*3];
      green[i]=(byte)palette[i*3+1];
      blue[i]=(byte)palette[i*3+2];
    }
    IndexColorModel icm=new IndexColorModel(8, colorCount, red, green, blue);
    image=new BufferedImage(
        width, height, BufferedImage.TYPE_BYTE_INDEXED, icm);
    image.getRaster().setPixels(0, 0, width, height, img);
    //histogram for 8 bit:
    //-----------------------------
  }
  else
  {
    image=new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
    image.getRaster().setPixels(0, 0, width, height, img);
    //histogramm for 24 bit:
    //-------------------------------
  }
  // normalize histogram:
  //-------------------------
  repaint();
}
```

Erweitern Sie weiterhin die Methode `public void setMessage(String message)` der Klasse `DrawArea`, bis sie so aussieht:

```
public void setMessage(String message)
{
  this.message=message;
  image=null;
  repaint();
}
```

Übersetzen mit `Strg+1`.
Zum einlesen einer `*.bmp` Datei benötigen wir einen `JFileChooser`, der sich dann öffnet, wenn in der Menüleiste auf den Menüpunkt `Read` geklickt wird. Nachdem die Datei ausgewählt wurde und der OK-Button gedrückt wurde, wird die Datei mit Hilfe eines `BufferedInputStream` Byte für Byte eingelesen.
Wechseln Sie dazu in TextPad zur Datei `MenuBar`: `Fenster 3 C:\...\MenuBar.java`

Ergänzen Sie die `MenuActionListener` Klasse (innere Klasse von `FileMenu`), bis sie so aussieht:

```
    class MenuActionListener implements ActionListener
    {
      JFileChooser fileChooser;
      int[] bitmapFileHeader=new int[14];
      int[] bitmapInfoHeader=new int[40];
      int[] bitmapColorTable;
      int[] bitmapData;
      static final String NOT_BMP_FILE_ERROR_MSG=
        "File is not supported or corrupt. Choose a Windows Histo-File (.bmp).";
      static final String WRONG_BIT_COUNT_ERROR_MSG=
        "Choose a 8 or 24 bit Windows Histo-File (.bmp).";
      static final String FILE_NOT_FOUND_ERROR_MSG=
        "The chosen file does not exist.";
      static final String IO_ERROR_MSG=
        "An IOException occured when reading the file.";
      public void actionPerformed(ActionEvent ev)
      {
        String cmd=ev.getActionCommand();
        if(cmd.equals(FileMenu.READ_ITEM_CMD))
        {
          if(fileChooser==null)
          {
            fileChooser=new JFileChooser(".\\");
            fileChooser.setFileFilter(new FileFilter()
            {
              public boolean accept(File f)
              {
                String path=f.getAbsolutePath().toLowerCase();
                if ((f.isDirectory()) || (path.endsWith(".bmp")))
                  return true;
                return false;
              }
              public String getDescription()
              {
                return "Windows Histo (*.bmp)";
              }
            });
            fileChooser.setAcceptAllFileFilterUsed(false);
          }
          int option=fileChooser.showOpenDialog(drawArea.getTopLevelAncestor());
          if(option==JFileChooser.APPROVE_OPTION)
          {
            File f=fileChooser.getSelectedFile();
            BufferedInputStream input=null;
            try
            {
              input=new BufferedInputStream(
                  new FileInputStream(f));
              for(int i=0; i<bitmapFileHeader.length; i++)
                bitmapFileHeader[i]=input.read();
              char firstByte=(char)bitmapFileHeader[0];
              char secondByte=(char)bitmapFileHeader[1];
              if(firstByte!='B' || secondByte!='M')
              {
                drawArea.setMessage(NOT_BMP_FILE_ERROR_MSG);
                return;
              }
              int bfSize=getValueOfBytes(bitmapFileHeader, 2, 5);
              if(bfSize<=54)
              {
                drawArea.setMessage(NOT_BMP_FILE_ERROR_MSG);
                return;
              }
```

```
int bfOffBits=getValueOfBytes(bitmapFileHeader, 10, 13);
if(bfOffBits<54)
{
  drawArea.setMessage(NOT_BMP_FILE_ERROR_MSG);
  return;
}
for(int i=0; i<bitmapInfoHeader.length; i++)
  bitmapInfoHeader[i]=input.read();
int biSize=getValueOfBytes(bitmapInfoHeader, 0, 3);
int biWidth=getValueOfBytes(bitmapInfoHeader, 4, 7);
int biHeight=getValueOfBytes(bitmapInfoHeader, 8, 11);
int biPlanes=getValueOfBytes(bitmapInfoHeader, 12, 13);
int biBitCount=getValueOfBytes(bitmapInfoHeader, 14, 15);
if(biBitCount!=8 && biBitCount!=24)
{
  drawArea.setMessage(WRONG_BIT_COUNT_ERROR_MSG);
  return;
}
int biCompression=getValueOfBytes(bitmapInfoHeader, 16, 19);
int biSizeImage=getValueOfBytes(bitmapInfoHeader, 20, 23);
int biXPelsPerMeter=getValueOfBytes(bitmapInfoHeader, 24, 27);
int biYPelsPerMeter=getValueOfBytes(bitmapInfoHeader, 28, 31);
int biClrUsed=getValueOfBytes(bitmapInfoHeader, 32, 35);
int biClrImportant=getValueOfBytes(bitmapInfoHeader, 36, 39);
if(biBitCount==8)
{
  //read palette:
  int paletteSize=bfOffBits-54;
  int sizeWithoutReservedBytes=paletteSize-paletteSize/4;
  bitmapColorTable=new int[sizeWithoutReservedBytes];
  for(int i=0; i<bitmapColorTable.length; i++)
  {
    //convert BGR to RGB and skip Reserved Byte
    if(i%3==0)//blue
      bitmapColorTable[i+2]=input.read();
    if(i%3==1)//green
      bitmapColorTable[i]=input.read();
    if(i%3==2)//red
    {
      bitmapColorTable[i-2]=input.read();
      input.skip(1);
    }
  }
  //read bitmapBytes:
  int emptyBytes=4-biWidth%4;
  if(emptyBytes==4)
    emptyBytes=0;
  bitmapData=new int[biWidth*biHeight];
  for(int i=biHeight-1; i>=0; i--)//turns over the rows
  {
    for(int j=0; j<biWidth; j++)
      bitmapData[i*biWidth+j]=input.read();
    input.skip(emptyBytes);
  }
}
else
```

```
            {
              int emptyBytes=4-(biWidth*3)%4;
              if(emptyBytes==4)
                emptyBytes=0;
              bitmapData=new int[(biWidth*3)*biHeight];
              for(int i=biHeight-1; i>=0; i--)//turns over the rows
              {
                for(int j=0; j<(biWidth*3); j++)//converts BGR to RGB
                {
                  if(j%3==0)//blue
                    bitmapData[i*(biWidth*3)+j+2]=input.read();
                  if(j%3==1)//green
                    bitmapData[i*(biWidth*3)+j]=input.read();
                  if(j%3==2)//red
                    bitmapData[i*(biWidth*3)+j-2]=input.read();
                }
                input.skip(emptyBytes);
              }
            }
            drawArea.setImage(bitmapData, bitmapColorTable, biBitCount,
                biWidth, biHeight);
          }
          catch(FileNotFoundException ex)
          {
            drawArea.setMessage(FILE_NOT_FOUND_ERROR_MSG);
            return;
          }
          catch(IOException ex)
          {
            drawArea.setMessage(IO_ERROR_MSG);
            return;
          }
          try
          {
            input.close();
          }
          catch(IOException ex)
          {
            drawArea.setMessage(IO_ERROR_MSG);
            return;
          }
        }
      }
      else if(cmd.equals(FileMenu.EXIT_ITEM_CMD))
        System.exit(0);
    }
    private int getValueOfBytes(final int[] bytes, int startByte, int stopByte)
    {
      int value=0;
      int exp=0;
      for(int i=startByte; i<stopByte+1; i++)
        value+=bytes[i]*(int)Math.pow(2,exp++*8);
      return value;
    }
  }
```

Übersetzen mit `Strg+1`. Wechseln Sie in TextPad zur Datei `Histo1`: `Fenster 1 C:\...\Histo1.java`. Ausführen mit `Strg+2`.

Erproben Sie die hinzugefügte Funktionalität des Menüpunktes `Read`. Testen Sie außerdem den Accelerator (Tastenkombination als Alternative für einen Menüpunkt) `Strg+R`. Lesen Sie verschiedene `*.bmp` Dateien und erproben Sie das Programm.

Beachten Sie, das unser `JFileChooser` nur in der Lage ist, Files mit der Endung `.bmp` anzuzeigen. Versuchen Sie das Verhalten zu ändern, indem Sie die Zeile `fileChooser.setAcceptAllFileFilterUsed(false);` in der Methode `public void actionPerformed(ActionEvent ev)` der Klasse `MenuActionListener` auskommentieren.

# Code für Histogramm

Beenden Sie das Programm.
Wechseln Sie in TextPad zur Datei `DrawArea`: `Fenster 2 C:\...\DrawArea.java`
Version 2: Schreiben Sie in die Methode `public void setImage(...)` der Klasse `DrawArea` folgenden Code unterhalb der Kommentarzeile `//histogramm for 8 bit:`

```
int[] paletteGray=new int[colorCount];
for(int i=0; i<colorCount; i++)
{
  gray=(palette[i*3]+palette[i*3+1]+palette[i*3+2])/3;
  paletteGray[i]=gray;
}
grayArray=new int[img.length];
for(int i=0; i<img.length; i++)
{
  histogram[paletteGray[img[i]]]++;
  grayArray[i]=paletteGray[img[i]];
}
```

Schreiben Sie in die Methode `public void setImage(...)` der Klasse `DrawArea` folgenden Code unterhalb der Kommentarzeile `//histogram for 24 bit:`

```
grayArray=new int[img.length/3];
for(int i=0; i<img.length; i+=3)
{
  gray=(img[i]+img[i+1]+img[i+2])/3;
  histogram[gray]++;
  grayArray[i/3]=gray;
}
```

Schreiben Sie in die Methode `public void setImage(...)` der Klasse `DrawArea` folgenden Code unterhalb der Kommentarzeile `// normalize histogram:`

```
int hmax=0;
for(int i=0; i<256; i++)
  if(histogram[i]>hmax)
    hmax=histogram[i];
for(int i=0; i<256; i++)
  histogram[i]=(100*histogram[i])/hmax;
```

Schreiben Sie in die Methode `public void setImage(...)` der Klasse `DrawArea` folgenden Code als vorletzte Anweisungen noch vor die Zeile `repaint();`:

```
oneBitImage=new BufferedImage(width, height,
    BufferedImage.TYPE_BYTE_BINARY);
blackAndWhite=new int[grayArray.length];
```

Erweitern Sie die Methode `private void drawHisto(Graphics g)` der Klasse `DrawArea`, bis sie so aussieht:

```
private void drawHisto(Graphics g)
{
  g.setColor(Color.WHITE);
  g.fillRect(myWidth-(HISTO_FIELD_W+20), myHeight-(HISTO_FIELD_H+20),
      HISTO_FIELD_W, HISTO_FIELD_H);
  g.setColor(Color.RED);
  g.drawRect(myWidth-(HISTO_FIELD_W+20), myHeight-(HISTO_FIELD_H+20),
      HISTO_FIELD_W, HISTO_FIELD_H);
  g.setColor(Color.BLACK);
  for(int i=0; i<256; i++)
    g.drawLine((myWidth-(HISTO_FIELD_W+19))+i, myHeight-21,
        (myWidth-(HISTO_FIELD_W+19))+i, (myHeight-21)-histogram[i]);
}
```

Übersetzen mit `Strg+1`. Wechseln Sie in TextPad zur Datei `Histo1`: `Fenster 1 C:\...\Histo1.java`.
Ausführen mit `Strg+2`. Erproben Sie das Histogramm.

# Mausereignisse und Schwellwert

Beenden Sie das Programm.
Wechseln Sie in TextPad zur Datei `DrawArea`: `Fenster 2 C:\...\DrawArea.java`
Version 3: Ergänzen Sie die Methode `private void setOneBitImagePixels(int threshold)` der Klasse `DrawArea`, bis sie so aussieht:

```
    private void setOneBitImagePixels(int threshold)
    {
      for(int i=0; i<grayArray.length; i++)
        blackAndWhite[i]=grayArray[i]<threshold?0:1;
      oneBitImage.getRaster().setPixels(0, 0, imgWidth, imgHeight,
          blackAndWhite);
    }
```

Ergänzen Sie die Methode `private void drawOneBitImage(Graphics g, int threshold)` der Klasse `DrawArea`, bis sie so aussieht:

```
    private void drawOneBitImage(Graphics g, int threshold)
    {
      setOneBitImagePixels(threshold);
      g.drawImage(oneBitImage, 0, 0, myWidth, myHeight, this);
    }
```

Ergänzen Sie die Methode `public void mousePressed(MouseEvent ev)` der anonymen `MouseListener` Klasse im Konstruktor der Klasse `DrawArea`, bis sie so aussieht:

```
    public void mousePressed(MouseEvent ev)
    {
      int mouseX=ev.getX();
      int mouseY=ev.getY();
      clickInHisto=mouseX>myWidth-(HISTO_FIELD_W+20) && mouseX<myWidth-20 &&
          mouseY>myHeight-(HISTO_FIELD_H+20) && mouseY<myHeight-20;
    }
```

Ergänzen Sie die Methode `public void mouseReleased(MouseEvent ev)` der anonymen `MouseListener` Klasse im Konstruktor der Klasse `DrawArea`, bis sie so aussieht:

```
    public void mouseReleased(MouseEvent ev)
    {
      repaint();
    }
```

Ergänzen Sie die Methode `public void mouseDragged(MouseEvent ev)` der anonymen `MouseMotionListener` Klasse im Konstruktor der Klasse `DrawArea`, bis sie so aussieht:

```
    public void mouseDragged(MouseEvent ev)
    {
      if(!clickInHisto)
        return;
      int newX=ev.getX();
      int newY=ev.getY();
      if(newX<=myWidth-(HISTO_FIELD_W+20) || newX>=myWidth-20 ||
          newY<=myHeight-(HISTO_FIELD_H+20) || newY>=myHeight-20)
        return;
      Graphics g=getGraphics();
      int threshold=newX-(myWidth-(HISTO_FIELD_W+20));
      drawOneBitImage(g, threshold);
      drawHisto(g);
      g.setColor(Color.RED);
      g.drawLine(newX, myHeight-21, newX, myHeight-(HISTO_FIELD_H+19));
      g.setColor(Color.BLACK);
    }
```

Übersetzen mit `Strg+1`. Wechseln Sie in TextPad zur Datei `Histo1`: `Fenster 1 C:\...\Histo1.java`.
Ausführen mit `Strg+2`. Erproben Sie die Binarisierung.

# Beispielbilder

Das Programm sollte alle 8bit bzw. 24bit Bitmaps lesen und anzeigen. Falls Sie keine `*.bmp` Dateien auf Ihrer Harddisk finden, benutzen Sie folgende Beispielbilder:
Download: Butterfly.bmp 217 kB 24Bit-TrueColor-Bild
Download: Madonna.bmp 18 kB 8Bit-Grauwert-Bild
Download: Lena256.bmp 66 kB 8Bit-Grauwert-Bild
Download: Lena512.bmp 258 kB 8Bit-Grauwert-Bild
Download: Angiography.bmp 66 kB 8Bit-Grauwert-Bild

# Weitere Aufgaben

Lesen sie die JavaTM 2 SDK, Standard Edition Documentation zu den Sprachelementen, die Sie geschrieben haben (siehe: http://java.sun.com/j2se/1.4.1/search.htm). Schauen Sie sich vor allem die Dokumentation zu den Konstruktoren `BufferedImage(int width, int height, int imageType)` und `BufferedImage(int width, int height, int imageType, IndexColorModel cm)` der Klasse `java.awt.image.BufferedImage` an. Verstehen Sie in diesem Zusammenhang auch die Code-Zeile `oneBitImage=new BufferedImage(width, height, BufferedImage.TYPE_BYTE_BINARY);` in der Methode `public void setImage(...)`.
Verändern Sie die Position und Hö;he des Histogramms, indem Sie die Konstanten `HISTO_FIELD_W` und `HISTO_FIELD_H` und die Methode `private void drawHisto(Graphics g)` der Klasse `DrawArea` anpassen.
Passen Sie die Hö;hennormierung des Histogramms an.
Programmieren Sie drei zusätzliche Histogrammfenster getrennt für die drei Farben rot, grün und blau.
Programmieren Sie so um, dass man die Originalpixel unterhalb des Schwellwerts sieht und dass nur die Pixel über der Schwelle auf weiss gesetzt werden. Programmieren Sie auch den umgekehrten Fall, wo nur die Pixel unterhalb der Schwelle auf schwarz gesetzt werden.
Erfinden und erproben Sie neue Varianten des Programms in Form von neuen Projekten `histo2`, `histo3` usw. nach obigem Muster.