Legende:

Globale Deklarationen
setup()
draw()
serialEvent()
keyPressed()
Digitale Filter (IIR)

```
// EKG-Monitor (Jens Bongartz, RheinAhrCampus Remagen)
// Stand: 14.05.2018
// roter EKG-Clip  >> rechter Arm
// weißer EKG-Clip >> linker Arm
// schwarzer EKG-Clip >> Bein

// Bibliotheken importieren
// ========================
import processing.serial.*;
import java.awt.event.KeyEvent;
// Globalen Speicher fuer verschiedene Objekte reservieren
Serial myPort;
int serialCount;
PFont f;
Biquad notch50Hz;
Biquad TP40Hz;
Biquad HP15Hz;
int fa = 200;
int width  = 800;
// height = 550 >> Normaler Monitor
// height = 400 >> 7" Touch
// height = 480 >> 7" Touch Fullscreen
int height = 400;
```

```
// data1[] ist das Array, aus dem die Daten auf den Bildschirm gezeichnet werden
// Organisiert als Ringspeicher: data_index zeigt auf das aktuelle einzufügende Element
int[] data1 = new int[width];          // Indices von 0 ... width-1
int data_index = 0;

// Im inBuffer stehen die Daten, die ueber die serielle Schnittstelle gesendet wurden
//
byte[] inBuffer = new byte[17];
int newDataPoint1 = 0;

PrintWriter output;

boolean isNotch50Hz = false;
boolean isTP40Hz = false;
boolean isHP15Hz = false;
boolean isRecording = false;
boolean isTimingInfo = false;

int serialEvent_t0 = 0;
int serialEvent_time;
int x_scale = 1;

void setup() {
  //
  // Display initialisieren
  // ======================
  //size(800, 550,P2D);
  // 7" Display hat 800 x 480 Pixel
  size(800,400,P2D);
  // fullScreen(P2D);
  // Render-Mode P2D macht Grafikausgabe deutlich schneller
  frameRate(60);

  // Auflistung aller verfuegbaren seriellen Schnittstellen
  // ======================================================
  println(Serial.list());
  // Serielle Schnittstelle oeffnen
  // >> Hier bitte den String aus der Fusszeile der Arduino-IDE einfuegen
  myPort = new Serial(this, "/dev/cu.usbmodem14101", 115200);
  // serialEvent ausloesen, wenn CR/LF auftritt
  myPort.bufferUntil('\n');
  // Abtastrate festlegen
  myPort.write("fa="+fa+"\n");
  // Ausgabedatei fuer Recording festlegen
  // =====================================
  output = createWriter("data.txt");
  //
  // Textfont fuer Bildschirmausgabe formatieren
  // ===========================================
  f = createFont("Courier",20,true);
  textFont(f);
  //
  // Datenarray data1[] auf Null setzen
  // aufgrund der Mittelwertberechnung in SerialEvent
  // ================================================
  for(int i = 0; i < width-1; i++)
  {
    data1[i]=0;
  }
  //
  // Biquad-Filter initialisieren
  // ============================
  // Notch 50 Hz Filter bei fa = 250 Hz
  float[] NotchCoeff = calcNotchCoeff(fa,50.0);
  //println(NotchCoeff);
  notch50Hz = new Biquad(NotchCoeff[0],NotchCoeff[1],NotchCoeff[2],NotchCoeff[3],NotchCoeff[4]);
  // Tiefpass 40 Hz bei fa = 250 Hz
  float[] TPCoeff = calcTPCoeff(fa,40.0);
  //println(TPCoeff);
  TP40Hz = new Biquad(TPCoeff[0],TPCoeff[1],TPCoeff[2],TPCoeff[3],TPCoeff[4]);
  // Hochpass 15 Hz Filter
  float[] HPCoeff = calcHPCoeff(fa,15.0);
  println(HPCoeff);
  HP15Hz = new Biquad(HPCoeff[0],HPCoeff[1],HPCoeff[2],HPCoeff[3],HPCoeff[4]);
}

void draw()
{
  background(255,255,255);
  // Mittellinie in schwarz
  strokeWeight(1);
  stroke(0,0,0);
  line(0,height/2,width,height/2);
  stroke(255,0,0);                          // Zeichenfarbe ist rot
  if (isRecording)
  {
    fill(255,0,0);
    text("REC",10,height-20);
  }
  if (isNotch50Hz)
  {
    fill(0,255,0);
    text("Notch50Hz",100,height-20);
  }
  if (isTP40Hz)
  {
    fill(0,255,0);
    text("TP40Hz",250,height-20);
  }
  if (isHP15Hz)
  {
    fill(0,255,0);
    text("HP15Hz",400,height-20);
  }
  if (isTimingInfo)
  {
    fill(0,255,0);
    text(serialEvent_time+"ms "+serialCount,700,height-20);
  }
  // Mittelwert berechnen
  // ====================
  long datensumme = 0;
  int data_max = 0;
  int data_min = 1023;

  for(int i = 0; i < width-1; i++)
  {
    if (data1[i] > data_max) data_max = data1[i];
    if (data1[i] < data_min) data_min = data1[i];
    datensumme += data1[i];
  }
  int mittelwert = int(datensumme / width);
  float y_scale = 1;
  if (data_max > height)
  {
    y_scale = (float(height) / float(data_max))*0.85
    ;
  }
  //println(data_min, mittelwert, data_max, y_scale);
  beginShape(LINES);
     int draw_index = data_index;
     for(int i = 0; i < width-1; i++)
     {
       vertex(i*x_scale,height/2 - (data1[draw_index]-mittelwert)*y_scale);
       // Ringspeicher-Ende beachten
       if (draw_index == width-1) { draw_index = -1; }
       vertex((i+1)*x_scale,height/2 - (data1[draw_index+1]-mittelwert)*y_scale);
       draw_index++;
     }
  endShape();
}
// Daten vom Arduino ueber die serielle Schnittstelle empfangen
// ==============================================================
// Daten werden als ASCII-Zeichenfolge mit CR/LF gesendert
//
void serialEvent(Serial myPort) {
  serialEvent_time = millis() - serialEvent_t0;
  serialEvent_t0 = millis();
  serialCount = myPort.available();
  // count ist in der Regel 5: 3 Ziffern Abtastwert + CR und LF
  if (serialCount > 5) println("Count-Warning: ",serialCount);
  String inBuffer = myPort.readString();
  newDataPoint1 = int(trim(inBuffer));     // Typumwandlung String >> int

  if (isNotch50Hz) newDataPoint1 = notch50Hz.calc(newDataPoint1);
  if (isTP40Hz)    newDataPoint1 = TP40Hz.calc(newDataPoint1);
  if (isHP15Hz)    newDataPoint1 = HP15Hz.calc(newDataPoint1);

  data1[data_index] = newDataPoint1;        // neuen Messwert im Ringarray ablegen

  // data_index inkrementieren und Ringstruktur beachten

  if (data_index != width-1)
  {
    data_index ++;
  }
  else
  {
    data_index = 0;
  }
  if (isRecording)
  {
    output.println(newDataPoint1);
  }
}

void keyPressed() {
  // r - Recording
  // n - Notch-Filter
  // t - 40Hz Tiefpass
  // h - 15Hz Hochpass
  // q - Quit
  // i - Timing-Info

  if (key == 'q')                     // quit Program
  {
    output.flush();                   // Writes the remaining data to the file
    output.close();                   // Finishes the file
    exit();                           // Stops the program
  }
  if (key == 'r')
  {
    isRecording = !isRecording;
  }
  if (key == 'n')
  {
    isNotch50Hz = !isNotch50Hz;
  }
  if (key == 't')
  {
    isTP40Hz = !isTP40Hz;
  }
  if (key == 'h')
  {
    isHP15Hz = !isHP15Hz;
  }
  if (key == 'i')
  {
    isTimingInfo = !isTimingInfo;
  }
}

//
// Koeffizientenberechnung von shepazu.github.io/Audio-EQ-Cookbook/audio-eq-cookbook.html
//
float[] calcTPCoeff(float fs,float f0)
{
   float w0 = 2*PI*(f0/fs);
   float Q = 1/sqrt(2);
   float alpha = sin(w0)/(2*Q);
   float b0 = (1-cos(w0))/2;
   float b1 = 1-cos(w0);
   float b2 = b0;
   float a0 = 1 + alpha;
   float a1 = (-2)*cos(w0);
   float a2 = 1 - alpha;
   float[] coeff = {(b0/a0),(b1/a0),(b2/a0),(a1/a0),(a2/a0)};
   return coeff;
}

float[] calcHPCoeff(float fs,float f0)
{
   float w0 = 2*PI*(f0/fs);
   float Q = 1/sqrt(2);
   float alpha = sin(w0)/(2*Q);
   float b0 = (1+cos(w0))/2;
   float b1 = -(1+cos(w0));
   float b2 = b0;
   float a0 = 1 + alpha;
   float a1 = (-2)*cos(w0);
   float a2 = 1 - alpha;
   float[] coeff = {(b0/a0),(b1/a0),(b2/a0),(a1/a0),(a2/a0)};
   return coeff;
}

float[] calcNotchCoeff(float fs,float f0)
{
   float w0 = 2*PI*(f0/fs);
   float Q = 1/sqrt(2);
   float alpha = sin(w0)/(2*Q);
   float b0 = 1;
   float b1 = (-2)*cos(w0);
   float b2 = 1;
   float a0 = 1 + alpha;
   float a1 = (-2)*cos(w0);
   float a2 = 1 - alpha;
   float[] coeff = {(b0/a0),(b1/a0),(b2/a0),(a1/a0),(a2/a0)};
   return coeff;
}

class Biquad
{
  // lokale Variablen des Objektes >> bleiben erhalten
  double a0, a1, a2, b1, b2;
  double x0, x1, x2, y1, y2;
  // Konstruktor
  // ===========
  Biquad(double t_a0, double t_a1, double t_a2, double t_b1, double t_b2)
  {
    x0 = 0; x1 = 0; x2 = 0; y1 = 0; y2 = 0;
    a0 = t_a0; a1 = t_a1; a2 = t_a2; b1 = t_b1; b2 = t_b2;
  }
  int calc(int in)
  {
    double out;
    // Eingangstufen des digitalen Filters verschieben
    // ===============================================
    x2 = x1;
    x1 = x0;
    x0 = (double) in;
    // Stufen des rekursiven Teils verschieben
    // =======================================
    y2 = y1;
    // Ausgangsergebnis des Filters berechnen
    // ======================================
    out = a0*x0 + a1*x1 + a2*x2 - b1*y1 - b2*y2;
    y1 = out;
    return round((float) out);
  }
}
```