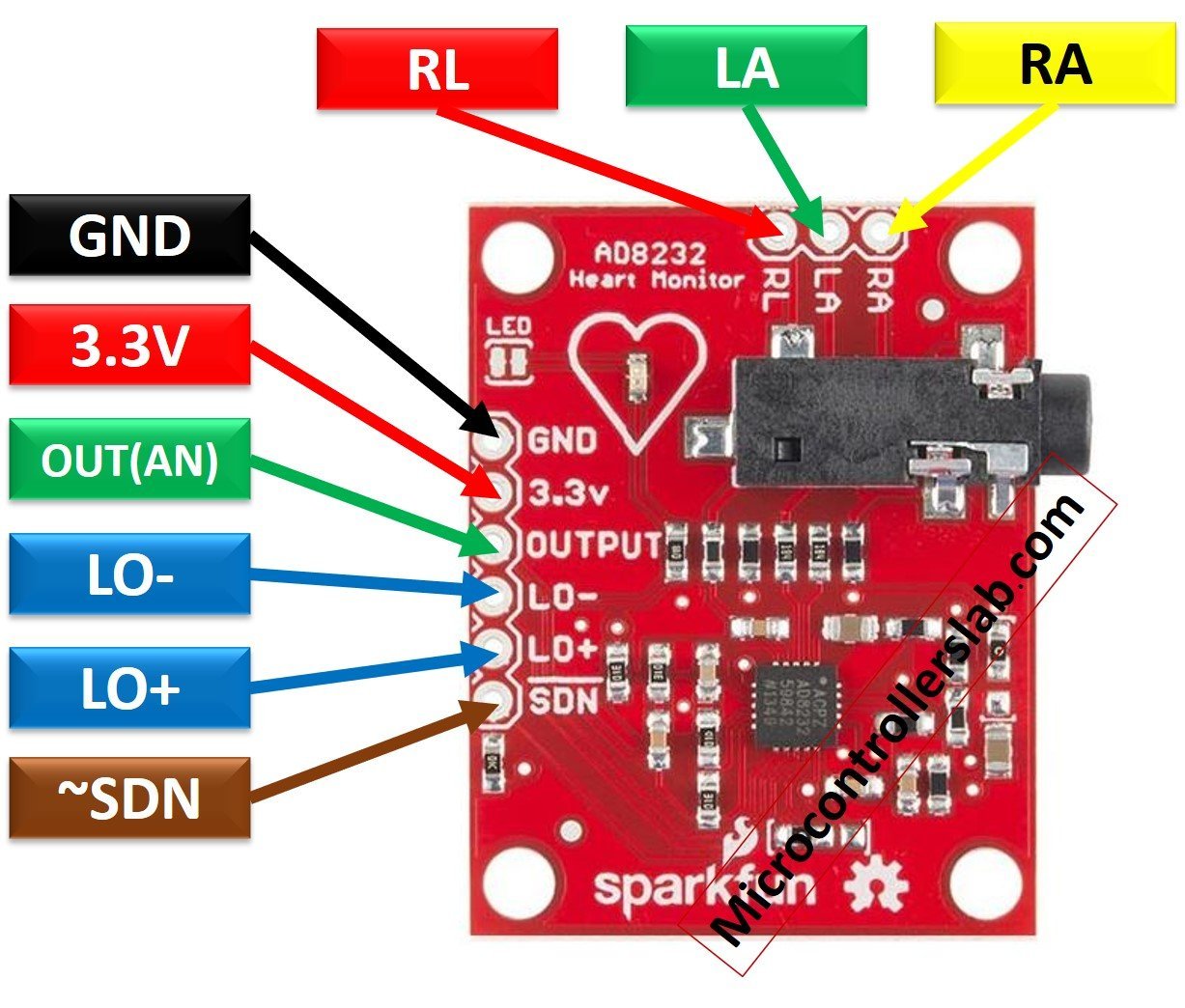
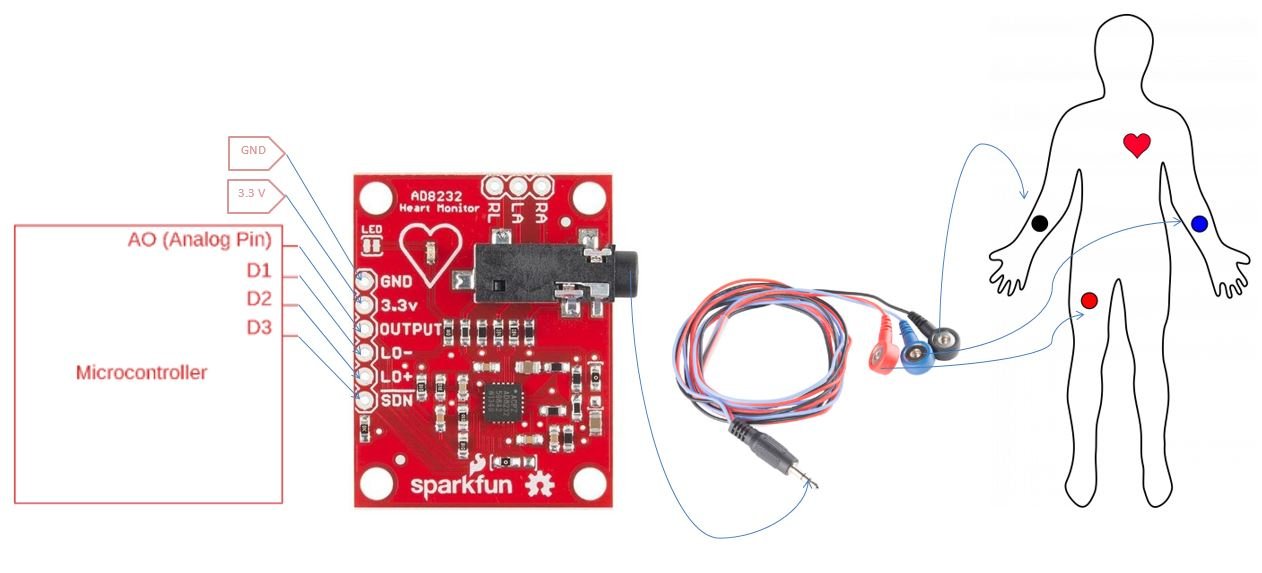
**HEALTH ADVISOR DESKTOP APPLICATION USING JAVA**

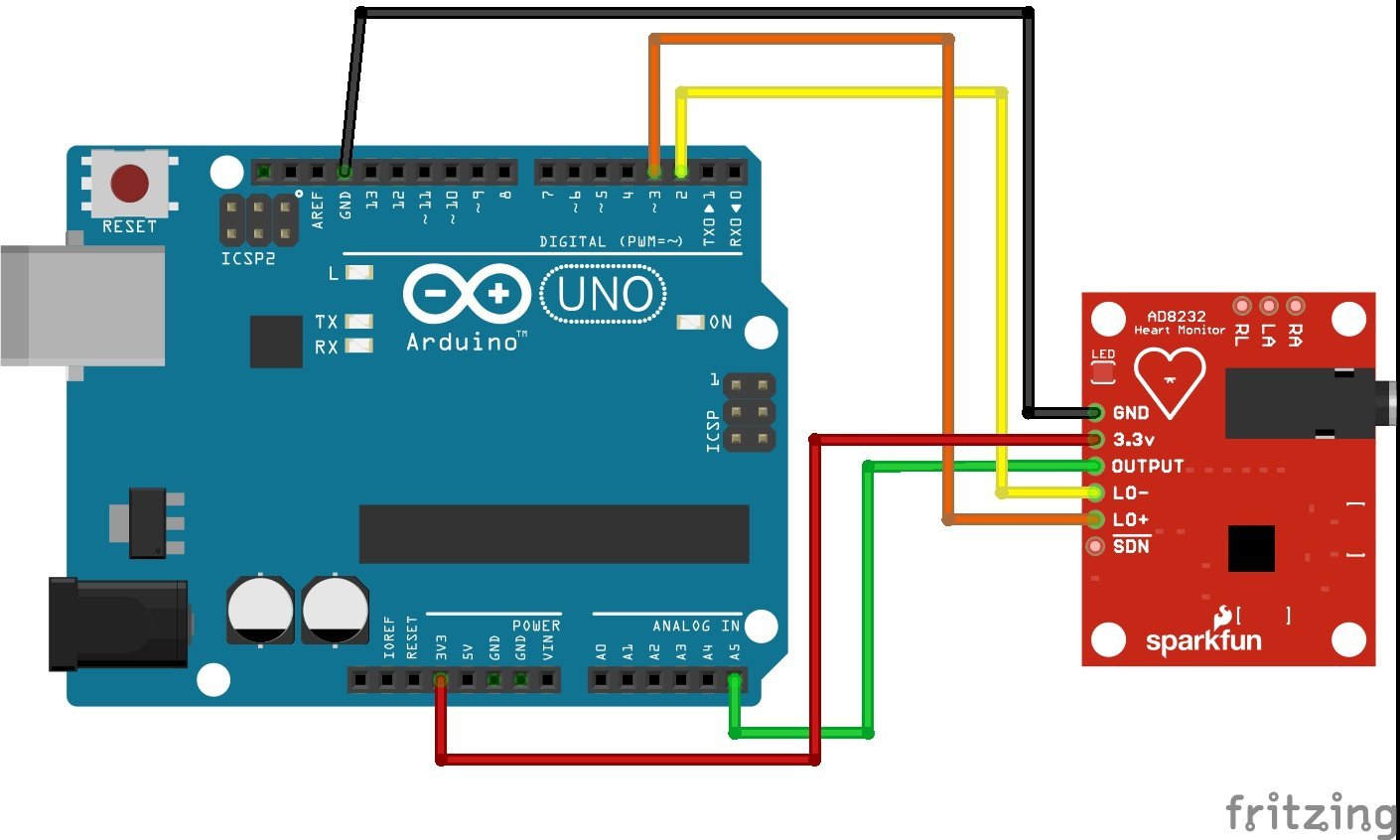
Technology- SWING UI java,

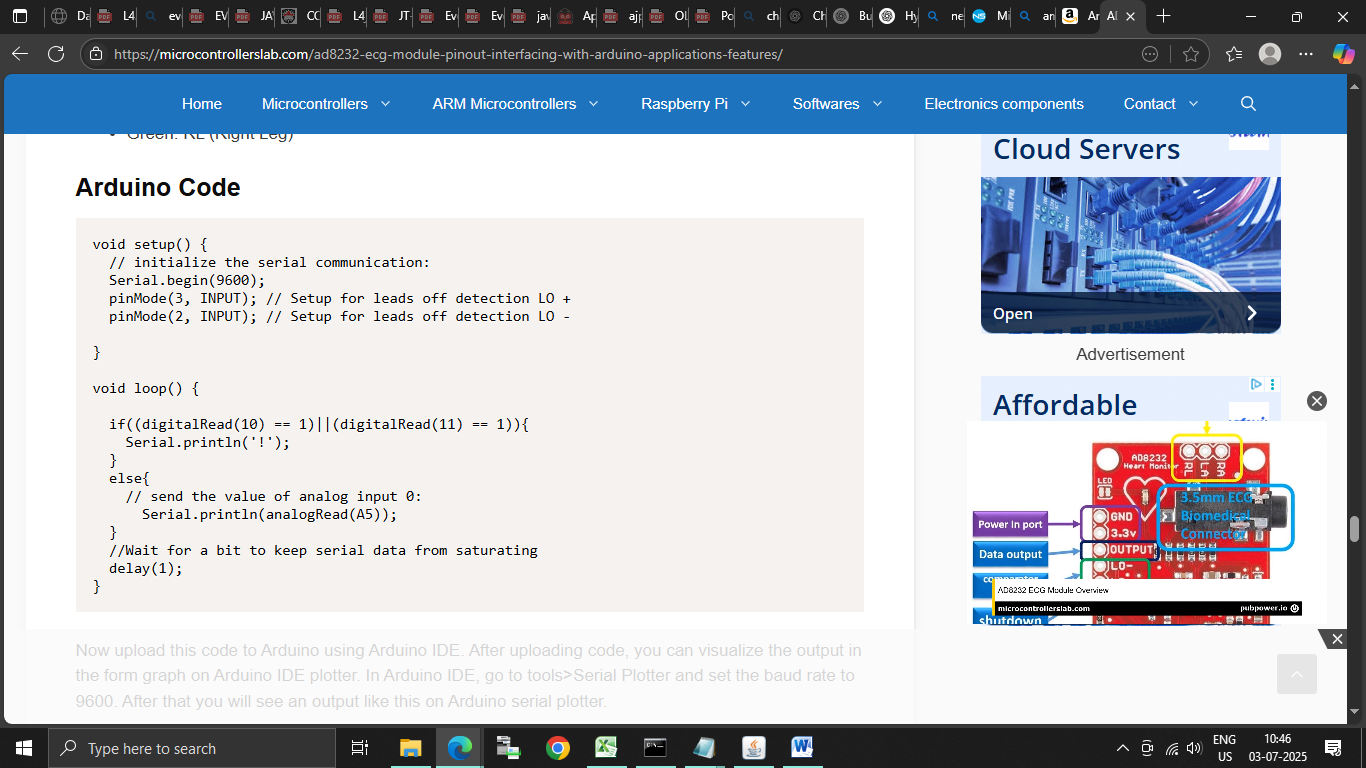
Hardware -AD8232 heart rate monitor ecg sensor, ARDUINO

Ref - [AD8232 ECG Module Pinout, Interfacing with Arduino, Applications](https://microcontrollerslab.com/ad8232-ecg-module-pinout-interfacing-with-arduino-applications-features/)





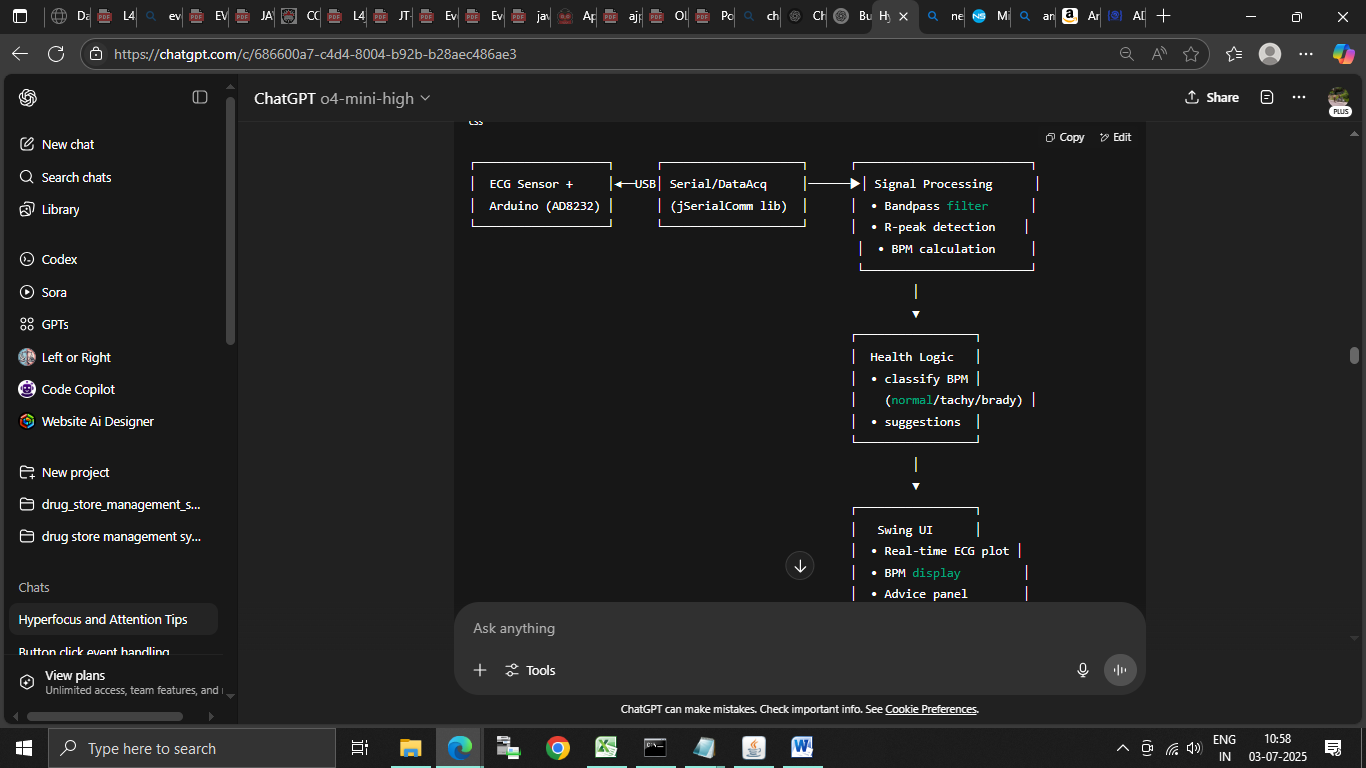




**Application scenario:**

Below is a high‐level plan plus starter code snippets to build a Java Swing desktop application that:

1. Reads raw ECG voltage data from a sensor (e.g. **AD8232 via Arduino** over serial)
2. Processes the signal in real time (filters, R-peak detection → heart rate)
3. Displays both the live ECG trace and simple health suggestions on a **Swing UI**
4. **System Architecture**



**2. Key Dependencies**

* **Serial Communication:** jSerialComm
* **Signal Processing:** pure Java (we’ll implement simple IIR/FIR filters and thresholding)
* **Charting (optional):** XChart or draw directly via JPanel
* **UI:** Java Swing

Add to your pom.xml (if using Maven):

**<dependencies>**

**<dependency>**

**<groupId>com.fazecast</groupId>**

**<artifactId>jSerialComm</artifactId>**

**<version>2.9.2</version>**

**</dependency>**

**<dependency>**

**<groupId>org.knowm.xchart</groupId>**

**<artifactId>xchart</artifactId>**

**<version>3.8.1</version>**

**</dependency>**

**</dependencies>**

**Links for jar files:**

[**https://fazecast.github.io/jSerialComm/**](https://fazecast.github.io/jSerialComm/)

[**https://knowm.org/open-source/xchart/**](https://knowm.org/open-source/xchart/)

**3. Module Breakdown & Starter Code**

**3.1. Serial Data Acquisition**

**import com.fazecast.jSerialComm.SerialPort;**

**public class SerialReader {**

**private SerialPort port;**

**private DataListener listener;**

**public SerialReader(String portDescriptor, DataListener listener) {**

**this.listener = listener;**

**port = SerialPort.getCommPort(portDescriptor);**

**port.setBaudRate(115200);**

**port.openPort();**

**port.addDataListener(new SerialPortDataListener() {**

**@Override public int getListeningEvents() { return SerialPort.LISTENING\_EVENT\_DATA\_AVAILABLE; }**

**@Override public void serialEvent(SerialPortEvent event) {**

**byte[] buf = new byte[port.bytesAvailable()];**

**port.readBytes(buf, buf.length);**

**// parse ASCII or binary into doubles:**

**String s = new String(buf).trim();**

**for (String part : s.split("\\s+")) {**

**try {**

**double v = Double.parseDouble(part);**

**listener.onData(v);**

**} catch (NumberFormatException ignored) {}**

**}**

**}**

**});**

**}**

**public interface DataListener {**

**void onData(double voltage);**

**}**

**}**

**3.2. Signal Processing & BPM Calculation**

import java.util.LinkedList;

public class SignalProcessor {

private LinkedList<Double> buffer = new LinkedList<>();

private double samplingRate = 250.0; // Hz

public void addSample(double v) {

buffer.add(v);

if (buffer.size() > samplingRate \* 5) // keep last 5 seconds

buffer.removeFirst();

}

// Simple bandpass: e.g. 0.5–40 Hz via two cascaded IIR filters

public double bandpass(double raw) {

// implement a filter or use a library...

return raw; // placeholder

}

// Naïve R-peak detection & BPM over sliding window

public int computeBPM() {

// find peaks above threshold in buffer, count per minute

int count = 0;

double thresh = 0.8 \* buffer.stream().mapToDouble(d->d).max().orElse(1);

for (int i = 1; i < buffer.size()-1; i++) {

if (buffer.get(i) > thresh &&

buffer.get(i) > buffer.get(i-1) &&

buffer.get(i) > buffer.get(i+1)) {

count++;

}

}

// windowDuration in seconds

double winSec = buffer.size() / samplingRate;

return (int) ((count / winSec) \* 60);

}

}

**3.3. Health Suggestion Engine**

public class HealthAdvisor {

public String getAdvice(int bpm) {

if (bpm < 60)

return "Bradycardia detected: consider light exercise and consult a physician if sustained.";

else if (bpm <= 100)

return "Heart rate normal. Keep up good sleep, hydration, and stress management.";

else

return "Tachycardia detected: try relaxation techniques; see a doctor if persists.";

}

}

**3.4. Swing UI Skeleton**

**import javax.swing.\*;**

**import java.awt.\*;**

**import java.util.Timer;**

**import java.util.TimerTask;**

**public class ECGApp extends JFrame**

**implements SerialReader.DataListener {**

**private SerialReader reader;**

**private SignalProcessor proc = new SignalProcessor();**

**private HealthAdvisor advisor = new HealthAdvisor();**

**private JLabel bpmLabel = new JLabel("BPM: --");**

**private JTextArea adviceArea = new JTextArea(3, 20);**

**private ECGPanel plotPanel = new ECGPanel();**

**public ECGApp() {**

**super("ECG Monitor");**

**setDefaultCloseOperation(EXIT\_ON\_CLOSE);**

**setSize(800, 600);**

**JPanel top = new JPanel();**

**top.add(bpmLabel);**

**add(top, BorderLayout.NORTH);**

**add(plotPanel, BorderLayout.CENTER);**

**adviceArea.setLineWrap(true);**

**adviceArea.setWrapStyleWord(true);**

**add(new JScrollPane(adviceArea), BorderLayout.SOUTH);**

**// replace "COM3" with your port**

**reader = new SerialReader("COM3", this);**

**// Repaint & compute BPM/suggestions every second**

**new Timer().scheduleAtFixedRate(new TimerTask() {**

**@Override public void run() {**

**int bpm = proc.computeBPM();**

**bpmLabel.setText("BPM: " + bpm);**

**adviceArea.setText(advisor.getAdvice(bpm));**

**plotPanel.setData(proc.buffer);**

**plotPanel.repaint();**

**}**

**}, 1000, 1000);**

**setVisible(true);**

**}**

**@Override**

**public void onData(double voltage) {**

**double filtered = proc.bandpass(voltage);**

**proc.addSample(filtered);**

**}**

**public static void main(String[] args) {**

**SwingUtilities.invokeLater(ECGApp::new);**

**}**

**}**

And a minimal custom panel to draw the ECG line:

**class ECGPanel extends JPanel {**

**java.util.List<Double> data = new java.util.ArrayList<>();**

**public void setData(java.util.List<Double> buf) {**

**data = new java.util.ArrayList<>(buf);**

**}**

**@Override**

**protected void paintComponent(Graphics g) {**

**super.paintComponent(g);**

**if (data.isEmpty()) return;**

**int w = getWidth(), h = getHeight();**

**double min = data.stream().mapToDouble(d->d).min().orElse(0);**

**double max = data.stream().mapToDouble(d->d).max().orElse(1);**

**for (int i = 1; i < data.size(); i++) {**

**int x1 = (i-1) \* w / data.size();**

**int y1 = h - (int)((data.get(i-1)-min)/(max-min) \* h);**

**int x2 = i \* w / data.size();**

**int y2 = h - (int)((data.get(i)-min) /(max-min) \* h);**

**g.drawLine(x1,y1, x2,y2);**

**}**

**}**

**}**

**4. Next Steps & Enhancements**

1. **Filter Tuning:** Implement a proper band-pass filter (e.g. Butterworth) for cleaner R-peak detection.
2. **Advanced Peak Detection:** Use the Pan-Tompkins algorithm for robustness.
3. **User Calibration:** Let users set thresholds or perform a quick calibration routine.
4. **Data Logging:** Save raw/processed data to CSV or database for later review.
5. **Alerts & Notifications:** Pop up dialogs or send emails/SMS when dangerous patterns are detected.
6. **Cross-Platform Packaging:** Use jpackage to create native installers for Windows/macOS/Linux.

Thought for a couple of seconds

To bridge your **ECG (or any analog) sensor → Arduino → Java Swing app, you’ll do three things:**

1. **Wire & program the Arduino to read the sensor and spit data over Serial**
2. **Install a Java serial-IO library** (e.g. jSerialComm) in your Swing project
3. **Open the serial port in Java, read the incoming bytes, and feed them into your Swing UI**

**1. Arduino Side**

**A. Wiring (using AD8232 ECG as example)**

| **Module Pin** | **Arduino Pin** |
| --- | --- |
| 5 V (or 3.3 V) | 5 V (or 3.3 V) |
| GND | GND |
| OUTPUT | A0 |

**Tip:** Consult your sensor’s datasheet—some modules have separate LO+/LO– pins for lead-off detection that you can ignore at first.

**B. Arduino Sketch**

**// Simple analog ECG sender**

**const int ECG\_PIN = A0;**

**void setup() {**

**Serial.begin(115200); // match this in Java**

**pinMode(ECG\_PIN, INPUT);**

**}**

**void loop() {**

**int raw = analogRead(ECG\_PIN); // 0–1023**

**float voltage = raw \* (5.0 / 1023.0);**

**Serial.println(voltage, 3); // e.g. "1.234"**

**delay(4); // ~250 Hz sampling**

**}**

1. Upload this to your Arduino.
2. Open the Serial Monitor at **115200 baud** to confirm you see a stream of numbers.

**2. Java Project Setup**

**A. Add jSerialComm to your build**

**Maven** (pom.xml):

**<dependency>**

**<groupId>com.fazecast</groupId>**

**<artifactId>jSerialComm</artifactId>**

**<version>2.9.2</version>**

**</dependency>**

Or grab the JAR from https://fazecast.github.io/jSerialComm/ and add it to your classpath.

**3. Java Swing Code to Read Serial Data**

**import com.fazecast.jSerialComm.\*;**

**import javax.swing.\*;**

**import java.awt.\*;**

**import java.util.Scanner;**

**public class SensorSwingApp extends JFrame {**

**private JLabel valueLabel = new JLabel("--");**

**private SerialPort comPort;**

**public SensorSwingApp() {**

**setTitle("Sensor Reader");**

**setDefaultCloseOperation(EXIT\_ON\_CLOSE);**

**setSize(300, 150);**

**// UI**

**valueLabel.setFont(new Font("Monospaced", Font.BOLD, 24));**

**valueLabel.setHorizontalAlignment(SwingConstants.CENTER);**

**add(valueLabel, BorderLayout.CENTER);**

**setVisible(true);**

**// Find & open the first available port**

**comPort = SerialPort.getCommPorts()[0]; // or match by name: "COM3" / "/dev/ttyACM0"**

**comPort.setBaudRate(115200);**

**if (!comPort.openPort()) {**

**JOptionPane.showMessageDialog(this, "Failed to open port!");**

**System.exit(1);**

**}**

**// Launch a thread to read incoming data**

**new Thread(this::readLoop).start();**

**}**

**private void readLoop() {**

**Scanner scanner = new Scanner(comPort.getInputStream());**

**while (scanner.hasNextLine()) {**

**String line = scanner.nextLine().trim();**

**SwingUtilities.invokeLater(() -> valueLabel.setText(line + " V"));**

**}**

**scanner.close();**

**}**

**public static void main(String[] args) {**

**SwingUtilities.invokeLater(SensorSwingApp::new);**

**}**

**}**

**Key Points**

* **Port Selection:** SerialPort.getCommPorts() lists all ports; pick the one matching your Arduino.
* **Baud Rate:** Must match Serial.begin(115200) on Arduino.
* **Threading:** Reading serial blocks, so do it off the EDT (Swing’s UI thread) and update the UI via SwingUtilities.invokeLater.

**4. Next Steps & Troubleshooting**

* **Port Permissions (Linux/macOS):** You may need to add your user to the dialout group or grant /dev/ttyACM0 permissions.
* **Signal Scaling & Filtering:** Raw analog might be noisy—consider smoothing (moving average) or proper band-pass filtering before plotting/advising.
* **Error Handling:** Watch for disconnects (comPort.isOpen()), and handle IOException in your read thread.
* **Enhancements:**
  + Plot the data in real-time (see the earlier ECGPanel example).
  + Add BPM calculation and health advice beside the live voltage.
  + Log raw data to a file or database for later analysis.

With these pieces in place, your Swing app will seamlessly display live sensor readings from the Arduino—ready for you to layer on signal processing, health logic, and polished UI components.

This is version 1 thank you