How to integrate transdermal-vitals-scanner library to android app

1. Add tvs.shared-1.0.0.jar library to project and add as local dependency (for example, in gradle something like `implementation(files("libs/com.tvs.shared-1.0.0.jar"))`

2. Add these configuration to access camera to your AndroidManifest file in resource folder:

```
    <uses-permission android:name="android.permission.CAMERA" />
    <uses-feature android:name="android.hardware.camera" />
    <uses-feature android:name="android.hardware.camera.flash" />
    <uses-feature android:name="android.hardware.camera.autofocus" />
    <uses-permission android:name="android.permission.FLASHLIGHT" />
    <uses-permission android:name="android.permission.WAKE_LOCK" />
```

After that you need to check and request camera permissions if needed

```
private boolean checkPermission() {
        return ContextCompat.checkSelfPermission(this, Manifest.
permission.CAMERA) == PackageManager.PERMISSION_GRANTED;
    }

    private void requestPermission() {
        ActivityCompat.requestPermissions(this, new String[]{Manifest.
permission.CAMERA}, PERMISSION_REQUEST_CODE);
    }
```

3. There is one method `processImage` in vitals.*Processor files which you should use in your android app. It accepts the byte array with image data, camera size (weight + height) and some anthropometric data. It returns `ProcessStatus` enum, with 5 statuses

`RED_INTENSITY_NOT_ENOUGH("Not good red intensity to process. Should start again"),`

`MEASUREMENT_FAILED("Measurement Failed. Should Start again"),`

`IN_PROGRESS("Processing in progress"),`

`PROCESS_FINISHED("Processing finished"),`

`NEED_MORE_IMAGES("Need more images to process")`

4. In your codebase with android components you should implement class extends `AppCompatActivity` which opens camera and process images in `onPreviewFrame` method. Example could be found below:

```
class VitalSignsProcess : AppCompatActivity() {
    private var preview: SurfaceView? = null
    private lateinit var vsp: VitalSignsProcessor

    private var previewHolder: SurfaceHolder? = null
    private var camera: Camera? = null
    private var wakeLock: PowerManager.WakeLock? = null
    private var mainToast: Toast? = null
    private var progBar: ProgressBar? = null
    var progP = 0
    var inc = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_vital_signs_process)
        vsp = VitalSignsProcessor(
            User(
                height = 188.0,
                weight = 76.0,
                age = 25,
                gen = 1
            )
        )
        preview = findViewById(R.id.preview)
        previewHolder = preview!!.holder
        previewHolder!!.addCallback(surfaceCallback)
        previewHolder!!.setType(SurfaceHolder.SURFACE_TYPE_PUSH_BUFFERS)
        progBar = findViewById(R.id.VSPB)
        progBar!!.progress = 0
        val pm: PowerManager = getSystemService(Context.POWER_SERVICE)
as PowerManager
        wakeLock = pm.newWakeLock(PowerManager.FULL_WAKE_LOCK, "tvs:
DoNotDimScreen")
    }

    //getting frames data from the camera and start the measuring
process
    private val previewCallback: Camera.PreviewCallback = object :
Camera.PreviewCallback {

        override fun onPreviewFrame(data: ByteArray, cam: Camera) {
            val size = cam.parameters.previewSize

            val width = size.width
            val height = size.height

            when (vsp.processImage(data, width, height)) {
                ProcessStatus.RED_INTENSITY_NOT_ENOUGH -> {
                    inc = 0
                    progP = inc
                    progBar!!.progress = progP
                }
                ProcessStatus.MEASUREMENT_FAILED -> {
                    inc = 0
                    progP = inc
                    progBar!!.progress = progP
                    mainToast =
                        Toast.makeText(applicationContext, "Measurement
Failed", Toast.LENGTH_SHORT)
                    mainToast!!.show()
                }
                ProcessStatus.IN_PROGRESS -> {
                    progP = inc++ / VITALS_PROCESS_DURATION
                    progBar!!.progress = progP
                }
                ProcessStatus.PROCESS_FINISHED -> {
                    //do what you need. Data can be accessed from vsp.*
fields (vsp.o2, vsp.Breath, vsp.Beats, vsp.SP, vsp.DP)
                    finish()
                }
                ProcessStatus.NEED_MORE_IMAGES -> {
                    //do nothing. Need to process more images
                }
            }
        }
    }

    private val surfaceCallback: SurfaceHolder.Callback = object :
SurfaceHolder.Callback {
        override fun surfaceCreated(holder: SurfaceHolder) {
            try {
                camera!!.setPreviewDisplay(previewHolder)
                camera!!.setPreviewCallback(previewCallback)
            } catch (t: Throwable) {
            }
        }

        override fun surfaceChanged(holder: SurfaceHolder, format: Int,
width: Int, height: Int) {
            val parameters = camera!!.parameters
            parameters.flashMode = Camera.Parameters.FLASH_MODE_TORCH
            val size = getSmallestPreviewSize(width, height, parameters)
            if (size != null) {
                parameters.setPreviewSize(size.width, size.height)
            }
            camera!!.parameters = parameters
            camera!!.startPreview()
        }

        override fun surfaceDestroyed(holder: SurfaceHolder) {}
    }

    private fun getSmallestPreviewSize(
        width: Int,
        height: Int,
        parameters: Camera.Parameters
    ): Camera.Size? {
        var result: Camera.Size? = null
        for (size in parameters.supportedPreviewSizes) {
            if (size.width <= width && size.height <= height) {
                if (result == null) {
                    result = size
                } else {
                    val resultArea = result.width * result.height
                    val newArea = size.width * size.height
                    if (newArea < resultArea) result = size
                }
            }
        }
        return result
    }
}
```

5. Results could be accessed from the *Processor object fields (for example, `vsp.o2`, `vsp.Breath`, `vsp.Beats`, `vsp.SP`, `vsp.DP`)