Sensing with AWARE

AWARE is a third-party library that simplifies sensor sampling on Android and iOS mobile phones. The simplification comes from a unified interface. For example, instead of using Android native approach to sensing android.hardware sensors, and a different one for sensing location or activity, or a completely different approach to sensing WiFi and Bluetooth, we can use AWARE's methods to handle all these sensors in the same way. Furthermore, AWARE brings additional "sensors", so called "soft-sensors", for sensing different application usage or communication patterns (e.g. calls and messages exchanged).

In this lab we will create a simple application that senses different sensors using AWARE framework.

Sampling acceleration with the app in the foreground

Create a new Android Studio project with an empty activity. The language of choice should be set to Kotlin and the minimum API set to 29. We will now add the AWARE library among our dependencies.

1) *If you are using gradle version 7.1.* or newer:*

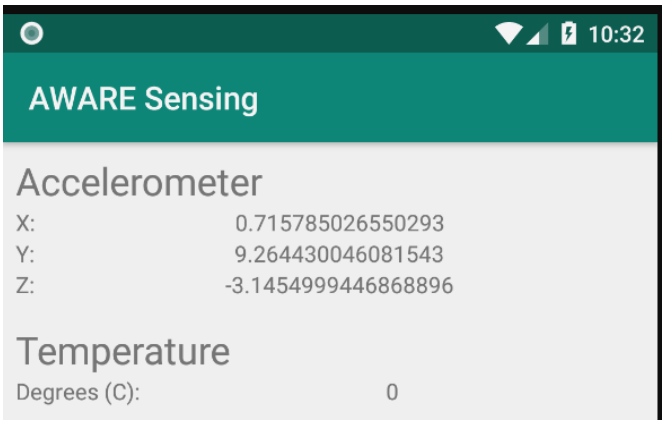
   In the project's *settings.gradle* file add `maven { url "https://jitpack.io" }` and `jcenter()` as repositories from which external libraries can be pulled (under `dependencyResolutionManagement { repositories {`)

   If you are using gradle version 7.0. or older:*

   In the project's gradle file add `maven { url "https://jitpack.io" }` as a repository from which external libraries can be pulled;

2) In the module's gradle file add `implementation 'com.github.denzilferreira:aware-client:master-SNAPSHOT'` as a dependency.

Rebuild your project to make sure that the dependency can be successfully pulled to your project.



We have one activity (`MainActivity`) whose layout (i.e. "face") is described in `res/layout/activity_main.xml`. Not much is there at the moment. We will change this so that something similar to the image on the right is shown to the user.

To save time, you can **download the XML layout file from Ucilnica**. Please read it carefully. We will update the TextViews with values read from the sensors (accelerometer and temperature). In addition to the `activity_main.xml` file, download also the `strings.xml` and `bools.xml` files and save them in the `res/values/` folder of the app project.

Let's initiate AWARE sensing in MainActivity! Changes should be made at three points during the Activity lifecycle. First, when the Activity is created, we should initialize core AWARE services and adjust the

sampling settings; when the user is engaged with our Activity (Activity is visible and in focus) we should start sampling; finally, when the user leaves our Activity, we should stop sensing.

Identify callbacks (`onCreate`, `onPause`, `onResume`, etc.) that you should override and place the following code where appropriate:

1)
```
Aware.startAWARE(this); //initialise core AWARE service

//sampling frequency in microseconds
Aware.setSetting(this, Aware_Preferences.FREQUENCY_ACCELEROMETER, 200000);


// intensity threshold to report the reading
Aware.setSetting(this, Aware_Preferences.THRESHOLD_ACCELEROMETER, 0.02f);
```

2)
```
Aware.startAccelerometer(this);
```

3)
```
Aware.stopAccelerometer(this);
```

AWARE will now ensure that the accelerometer is read and that the data is available via a `ContentProvider` to whoever is interested. We will instantiate a sensor data observer in order to capture the data in our app. Use `Accelerometer.setSensorObserver` function and set an observer to access the newly sensed data. Read about the content of this data object and extract acceleration x, y, and z axis values from the object. Next, we should update the UI with these values.

`SensorObserver` runs on a separate thread, thus, you cannot directly modify the UI. Instead, call the UI changing code from a `runOnUiThread()` function within the observer function.

Rebuild your app and run it in an emulator or on a physical phone. To facilitate debugging, use the `Log.d` function throughout the code and read the output in Android Studio's LogCat window.

Sampling temperature with the app in the foreground

The beauty of the AWARE framework is that all sensors are used via the same API. Thus, it is easy to sense yet another modality. In this case, we will add temperature sensing.

Keeping everything else as it was before, simply add `Aware.startTemperature(this);` to `onResume` and `Aware.stopTemperature(this);` to `onPause` callbacks. Of course, you should also set the observer using `Temperature.setSensorObserver` in `onCreate`. In the observer, make sure you read the temperature from the data object
(e.g.
`data.getAsDouble(Temperature_Provider.Temperature_Data.TEMPERATURE_CELSIUS)`
)
And set the value of the temperature TextView.

Recompile your app and test it in the emulator or on the phone.

Sampling with the app in the background

In mobile sensing applications we often want to sense even when a user does not actively interact with our app. On the other hand, the above approach ensures that sensing happens *only* when the app is used by the user.

WARNING: Android OS is becoming increasingly restrictive in terms of running processes in the background. Most of the tutorials on background processing that you can find online at the moment (February 2022) are not applicable to Android 11.

Currently, the best way to run processing in the background is to use the `WorkManager` class. This class schedules one off or periodic tasks (defined in a `Worker` class) to be run in the background. Note, while you can define the exact time when you would like the task to be fired (or the exact period at which the task will be repeated), it is up to the OS to schedule your task execution, thus, the exact time might be a bit off.

`WorkManager` is a part of AndroidX, a set of libraries that you need to add via Gradle. Open the module's gradle file and add the following line: `implementation 'androidx.work:work-runtime:2.6.0'`.

Then, create a new class `SensingWorker` that extends `Worker` (from `androidx.worker` package). Create the class primary constructor. You will automatically get `doWork` method where you can put work that you would like to get done in the background. Put the sensor observer for acceleration and `Aware.startAccelerometer` in `doWork`. You will notice that you cannot use "`this`" for context any more. Instead, you can directly access application context by calling `getApplicationContext`(). Since we cannot update the UI from the background thread if we are not even sure whether the UI exists, we will simply print the result of the sensing in LogCat. Make sure you stop the accelerometer after printing the result.

Finally, to schedule periodic execution of your SensingWorker, in MainActivity's onCreate, right after you start AWARE service put:

```
val myWorkBuilder = PeriodicWorkRequest.Builder(SensingWorker::class.java, 15, TimeUnit.MINUTES).build()

WorkManager

    .getInstance(this)

    .enqueue(myWorkBuilder)
```

Test your code in the emulator. Hopefully, you should see data printed in LogCat (roughly) every 15 minutes.

If you were not present in the lab for solving this assignment you should commit your solution to a private Bitbucket repository named **FRIMS2021-LAB-1** and a user **pbdfrita (pbdfrita@gmail.com)** should be added as a read-only member. The solutions will be pulled from your repository on **Sunday, February 27, 23:59**.

Happy coding!