Assignment 1

Start:	3rd October 2016
End:	13th October 2016

Objectives

The goal of this assignment is to familiarize yourself with the Android development process, to think about user interface design, and to learn how to access sensors on a smartphone. As the emulator cannot emulate all sensors, you will have to test your code on a physical device. Besides implementing the application, you have to answer a mini-test given below. More details are available in the *Deliverables* section. With this assignment you can gain 10 points out of the total 45.

Task	Points
1	2
2	3
3	3
4	2
Total	10

1 Sensing with Android (2 Points)

Every Android application must provide an Activity as an interface for user interaction. In this exercise, you will let the user access sensors and the corresponding values.

1. Create a new Android project called `VS_nethz_Sensors`. Set the application name to `VS_nethz_Sensors` and the package name to `ch.ethz.inf.vs.a1.nethz.sensors` (*nethz* here and also later means the group leader's nethz ID). Create the first Activity and name it `MainActivity`.

2. In the `MainActivity`, design a user interface to list all available sensors of the smartphone. The sensors should be contained in a ListView which automatically resizes with different input sizes.

3. Create a second Activity called `SensorActivity`. When the user clicks on an entry in the ListView, the `SensorActivity` should be started through an intent. The intent should carry information about the sensor. Display at least the name of the sensor in the `SensorActivity`.

4. In the `SensorActivity`, the sensor's values should be displayed. First, create a simple `TextView`, to show the current values. Note, that you should handle and display the values you obtain from the sensor correctly according to the type of the sensor, in terms of number of values (e.g. one for light intensity, three for acceleration) and the corresponding units. For that reason, create a class which implements the interface `SensorTypes` we provide. This class should offer the number of values and the unit for a specific sensor type. Moreover, be careful to copy the values before using them (see `https://developer.android.com/reference/android/hardware/SensorEventListener.html`).

5. Next, you should use a graph to display the history of the sensor values. Place the graph below the text view. Use the library `Graph View` (`http://www.android-graphview.org/`).

The library can be imported very easily by modifying the Gradle scripts as shown in the documentation and in the listing below.

Listing 1: GraphView setup in the app's gradle script.

```
android {
        ...
}

dependencies {
    ...
    compile 'com.jjoe64:graphview:4.2.0'
}
```

The graph should show the latest 100 values from the sensor (i.e. a moving window over the sensor values). If a sensor generates multiple values (e.g. the acceleration sensor), all of them should be shown in different series (and do use different colours for each series). The y-axis should be labelled with the sensor's corresponding unit, the x-axis with the time of the sensor readings since starting the SensorActivity (i.e. these time points are changing as well). Please create a class which implements the interface `GraphContainer` we provide, which holds the `GraphView` and applies all modifications to it.

6. We will run automated `JUnit` and Android instrumentation tests on your code and also provide some of them to you. To do do it is necessary that you implemented the two aforementioned interfaces. To run them make sure to add the following lines to your app gradle script in the according sections.

Listing 2: Setup for testing.

```
android {
    defaultConfig {
        testInstrumentationRunner "android.support.test.runner.AndroidJUnitRunner"
    }
}

dependencies {
        testCompile 'junit:junit:4.12'
    androidTestCompile 'com.android.support:support-annotations:23.4.0'
    androidTestCompile 'com.android.support.test:runner:0.4.1'
    androidTestCompile 'com.android.support.test:rules:0.4.1'
}
```

Since the sensors are still active during the test do not move your smartphone during the test since that might interfere with the artificial values injected in the test (or use the emulator).

2 The Anti-Theft Alarm (3 Points)

In this exercise, you will create an application to secure an Android device against unauthorized usage. When the device is locked, movements should be registered. If the user (thief) keeps moving the phone for a certain amount of time, the phone should raise an alarm (e.g. by playing a sound file or sending a silent notification). You should create an `Activity` to control the sensitivity of the alarm and the timeout after which an alarm is raised, and a `Service` to deal with the readings from the accelerometer. **You must use the code skeleton provided on the course website which is described below**.

1. Create a new project called `VS_`*`nethz`*`_AntiTheft` with an Activity called `MainActivity`. The package name should be `ch.ethz.inf.vs.a1.`*`nethz`*`.antitheft`.

2. The Activity will also need some means to start and stop the background process running the alarm logic. We suggest you to use a ToggleButton to change the state of the alarm.

3. Create a Service called `AntiTheftService`. The service must implement the `AlarmCallback` interface provided to you. This interface class contains a method `onDelayStarted()`, which is called from the movement detector as described below. The service should run in the background and post an *ongoing* notification which cannot be cleared by the user. This notification should only disappear when the Service is shut down. The notification is used for resuming of the `MainActivity`. Unlocking the device should leads to the termination of the service and the cancellation of the notification.
   Hint: Consider the guidelines of the Android website provided at `https://developer.android.com/guide/components/services.html` for services and `https://developer.android.com/guide/topics/ui/notifiers/notifications.html` for notifications.

4. The user should be able to access settings for the sensitivity and the length of the delay (i.e. the duration before the alarm sound is activated). To store settings, Android offers Preferences, a very convenient way store and retrieve values, which should persist in the application independently from the application cycle. Refer to the Settings guide (`https://developer.android.com/guide/topics/ui/settings.html`) for further information. An example of the use of the methods to store and retrieve a value in code are shown below.

Listing 3: Examples for preferences.

```
// Store value
boolean value = true;
SharedPreferences.Editor editor = getPreferences(Context.MODE_PRIVATE).edit();
editor.putBoolean("key_stored_boolean", value);
editor.apply();


// Retrieve stored value
boolean val = getPreferences(Context.MODE_PRIVATE).getBoolean("key_stored_boolean",
    false);
```

   Moreover, the user should be able to change the sensitivity of the alarm. To do so, you should implement a `PreferenceFragment` which is managed by an `Activity`. This `Activity` should be started by an intent when the user selects the settings in the options menu. All these topics are also mentioned in the Android settings guide. Changes should only take effect when the device is in the unlocked state. Otherwise the user should be notified by a `Toast` that nothing has changed.

5. To implement a movement detector, create a class which extends the provided `AbstractMovementDetector`, which implements the `SensorEventListener` interface. The movement detector should contain the logic necessary to trigger the alarm. Use the linear acceleration sensor, where gravity is already subtracted. Your logic should recognize a *deliberate* movement (which we will arbitrarily define as a significant change in sensor readings). The way you detect a deliberate movement should be a spike detector, i.e. if the element-wise sum of the absolute linear acceleration values exceeds the sensitivity threshold, a movement is detected. Additionally, add a second movement detector of your choice with a different mechanism. Add an option in the settings through which the user can select which detector is used (e.g. by using a list preference).

6. The user should have a certain period of time, the delay (Δ_t) defined in the settings, during which he/she can still disarm the device.

7. When the delay has elapsed after a deliberate movement, the phone should raise an alarm (e.g. play a sound file). Disarming the device (i.e. turning off the service) should also stop the alarm.

8. Pay attention to typical Android crashes like on rotating the screen, pushing the back button, etc. At the end, do not forget to unregister the sensor event listener, i.e. the movement detector. Failing to do so can drain the battery in just a few hours because some sensors have substantial power requirements and can use up battery power quickly. In contrast to earlier Android versions, the system will not disable sensors automatically when the screen turns off.

3 Sensing using Bluetooth Low Energy (3 Points)

In this task, you will use Android's built-in platform support for Bluetooth Low Energy (BLE) to connect to the Sensirion HumiGadget to sense humidity and temperature. You should display the current measurements to the user.

Helpful resources for this task are

- the Android guide on BLE: `https://developer.android.com/guide/topics/connectivity/bluetooth-le.html`. Note, that it partly uses deprecated methods and you should use the current methods which are available from version 5.0 onwards (see `https://developer.android.com/about/versions/lollipop.html` under Advanced Connectivity)

1. Create a new project called `VS_`*`nethz`*`_BleSensirion` with an Activity called `MainActivity`. The package name should be `ch.ethz.inf.vs.a1.`*`nethz`*`.ble`.

2. The Activity should first check if BLE is supported on the device. It should then check if bluetooth is enabled, and display a dialog to the user requesting permission to enable it if it is disabled.

3. Implement a scan for available devices. You should limit the time of the scan and you should only show Sensirion HumiGadget devices (e.g. by using a `ScanFilter`). Scanning should stop once the user selects one of the listed devices, or the scan-time expires.

4. Note, that you also need the permission `ACCESS_FINE_LOCATION`. Since Android 6.0 you have to request this permission from the user the first time the app is started. Find more information on `https://developer.android.com/training/permissions/requesting.html`. Additionally, also since Android 6 the location service has to be turned on to be able to perform a BLE-scan. You have to request the user to activate the location services.

5. Basically, scanning for BLE devices works as follows:

   (a) Request a `BluetoothManager` from the system and retrieve the `BluetoothAdapter` from it.

   (b) With the `BluetoothAdapter` you can check whether Bluetooth is turned on on the device or not. If not, display the dialog.

   (c) From the `BluetoothAdapter`, you can get the `BluetoothLeScanner`. With this you can scan for Bluetooth devices. You have to provide a `ScanCallback` to receive the results.

6. When the user selects a listed device, you should start a new activity which connects to it. This is done by the method `connectGatt(...)`, to which you have to pass a `BluetoothGattCallback`, which you have to implement. This callback monitors the connection state. The method returns a `BluetoothGatt` object with which you can discover and access the device's services by `discoverServices()` and `getService(...)`. When the connection state changes to `BluetoothProfile.STATE_CONNECTED`, you should attempt to start the service discovery on the device.

7. To access a service you can either iterate through the discovered services or use its unique identifier ID (UUID) in `getService(...)`. You should access two services, the humidity and the temperature service. All UUIDs needed are listed below and are available in the class `SensirionSHT31UUIDS` which we provide (To be able to omit the class name you can use a static import for the class).

8. A service is composed of at least one characteristic (containing a single value), and a characteristic holds descriptors (describing the characteristic's value, optional).

9. Once you hold a service, you can access its characteristics using `service.getCharacteristic(...)` with the corresponding UUID. Attempt to read a value from the data characteristic.

10. You will realise, that there is a problem with reading data values. That is because the service has to be activated on the device to use it. This is usually done by writing a one-byte in the service's configuration characteristic. The problem is that the characteristic has no permissions set for writing. To change that we add our own characteristic with the same UUID and the correct permissions using `addCharacteristic(...)`. Basically, we are overwriting the previous characteristic for that UUID.

11. Now you can read values from the data characteristic. Note, that these have to be converted to decimal format with the following code.

Listing 4: Conversion for humidity values.

```
1 private float convertRawValue(byte[] raw) {
2         ByteBuffer wrapper = ByteBuffer.wrap(raw).order(ByteOrder.LITTLE_ENDIAN);
3         return wrapper.getFloat();
4 }
```

12. Rather than reading or continuously polling the device, we want to be informed of new values. This can be done via notifications. To enable notification you have to set them for the app and the bluetooth device. First, enable them on the app by calling `bluetoothGatt.setCharacteristicNotification(...)` with the corresponding characteristic as argument. Second, for the bluetooth device, add a descriptor to the characteristic with the following UUID: `UUID_NOTIFICATION_DESCRIPTOR`. Set the value of the descriptor to `BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE` and call `writeDescriptor` to the descriptor of the data characteristic. Hence forth, whenever the values changes the method `onCharacteristicChanged(...)` in the `BluetoothGattCallback` will be called, from where you can retrieve the value.

13. The process above is the same for both the humidity and the temperature service.
Hint: Note, that to transfer anything to the BLE device you have to set the value and then call the corresponding write-method. However, you always have to make sure that a read or write on the bluetooth device has finished before you can call the next one. Take this into account for configuring the second service.

14. Finally, use a graph as in task 1 to display the values.

15. Be aware of handling the resources correctly, i.e. closing the connection when closing the app etc. **Hint:** One source of errors we found is pairing the BLE device with your smartphone in the Android settings. Do not do this.

- `UUID_HUMIDITY_SERVICE`: 00001234-b38d-4985-720e-0f993a68ee41
- `UUID_HUMIDITY_CHARACTERISTIC`: 00001235-b38d-4985-720e-0f993a68ee41
- `UUID_TEMPERATURE_SERVICE`: 00002234-b38d-4985-720e-0f993a68ee41
- `UUID_TEMPERATURE_CHARACTERISTIC`: 00002235-b38d-4985-720e-0f993a68ee41
- `UUID_NOTIFICATION_DESCRIPTOR`: 00002902-0000-1000-8000-00805f9b34fb

4 Mini-Test (2 Points)

As part of the assignment, you should answer the mini-test given below and submit the answers as a PDF file. The test covers general Android knowledge questions, as well as questions related to how you tackled the assignment.

1. (Sensor Framework)
   A) Write a small code snippet to show how to perform the following tasks:

   a) List available sensors on a device

   b) Retrieve the value range of a specific sensor

   c) Register for monitoring `accelerometer` sensor changes at the maximum available rate

   B) The following code snippet is used to monitor the values of the `ACCELEROMETER` and `PROXIMITY` sensors. The goal is to log the sensor values to create a history of them. The class `SensorValuesDetector` is used as the `SensorEventListener` in an activity (the code of the activity is omitted for brevity). The code for the `log` method is not given, it is just a placeholder for an implementation. Be aware that this method may take some time. **Lines 13-24** in the `onSensorChanged` method contain a mistake. Point out the mistake, and explain how it may cause a problem.

Listing 5: Code for question 1.b

```
1  import android.hardware.Sensor;
2  import android.hardware.SensorEvent;
3  import android.hardware.SensorEventListener;
4
5  public class SensorValuesDetector implements SensorEventListener {
6          float[] proximityValues;
7          float[] accelerometerValues;
8
9          @Override
10         public void onSensorChanged(SensorEvent event) {
11           int sensorType = event.sensor.getType();
12
13           switch (sensorType) {
14             case Sensor.TYPE_ACCELEROMETER:
15                 accelerometerValues = event.values;
16                 // log the values
17                 log(Sensor.TYPE_ACCELEROMETER, accelerometerValues);
18                 break;
```

```
19              case Sensor.TYPE_PROXIMITY:
20                  proximityValues = event.values;
21                  // log the values
22                  log(Sensor.TYPE_PROXIMITY, proximityValues);
23                  break;
24            }
25          }
26
27          @Override
28          public void onAccuracyChanged(Sensor sensor, int accuracy) {
29           // Do Nothing
30          }
31
32          public void log(int type, float[] values) {
33           // Logging the values. This operation may take some time.
34           // ...
35          }
36  }
```

2. (Activity lifecycle) The lifecycle of an activity comprises a set of states, and a set of callback functions which are invoked when transitioning from one state to another such as onStart(). However, only three states can be static, that is, the activity can exist in only one of these three states for an extended period of time. Name these three states and the corresponding callback functions which are invoked when transitioning to these states.

3. (Resources) It is discouraged to use literal strings in the app code and instead define them as resources. Where should strings be defined instead? What is the advantage of this approach?

4. What are `Intents`? What are they used for? What is the difference between `Explicit Intents` and `Implicit Intents`?

5. (Service lifecycle) State whether each of the following sentences is true or false:

   a) A service started by calling `startService()` can never be stopped before it finishes its job.

   b) Both bound and unbound services can interact with more than one client processes.

   c) A bound service can only be running as long as there is a component bound to it.

   d) Services which perform lengthy computations are automatically started in a separate thread from the main thread.

6. (AndroidManifest file) Suppose that you are implementing an Android application that consists of one activity `MainActivity` and one service `LocationService`. The service retrieves the user's current fine-grained location at regular intervals using Android's `Location Services` framework. The application sends a text message to a phone number with the current location every hour.

   Below is the AndroidManifest file for such an application. The file is missing **three** necessary tags for the application to work. List the needed tags.

Listing 6: AndroidManifest file for question 6

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      package="ch.ethz.inf.vs.android.nethz.antitheft"
4      android:versionCode="1"
5      android:versionName="1.0" >
6
```

```
7       <uses-sdk
8           android:minSdkVersion="8"
9           android:targetSdkVersion="18" />
10
11      <application
12          android:allowBackup="true"
13          android:icon="@drawable/ic_launcher"
14          android:label="@string/app_name"
15          android:theme="@style/AppTheme" >
16
17          <activity
18              android:name="ch.ethz.inf.vs.android.nethz.antitheft.MainActivity"
19              android:label="@string/app_name" >
20              <intent-filter>
21                  <action android:name="android.intent.action.MAIN" />
22
23                  <category android:name="android.intent.category.LAUNCHER" />
24              </intent-filter>
25          </activity>
26
27      </application>
28  </manifest>
```

Deliverables

The following deliverables have to be submitted by **11:59 p.m., 13th October 2016**:

1. **code.zip** You should create a zip file containing the Eclipse projects created in this assignment. The projects should have been tested both on the mobile phone and on the emulator. The code must compile on our machines as well, so always use relative paths if you add external libraries to your project. Do not forget to include those libraries in the zip file. Please use UTF-8 encoding for your documents and avoid special characters like umlauts.

2. **answers.pdf** Your answers to the mini-test in **PDF** format.

Submission

The deliverables must be uploaded through:
`https://www.vs.inf.ethz.ch/edu/vs/submissions/`
The group leader can upload the files, the other group members have to sign them in the online system to express their consent with the submission. Use your nethz accounts to log in. The submission script will not allow you to submit any part of this exercise after the deadline. However, you can re-submit as many times as you like until the deadline.