Lab 5 - Signal Sampling from Wearable Device (Fitpolo Wristband)

Fitpolo develops wristbands equipped with sensors. The wristbands can be used as standalone devices or in combination with Fitpolo smartphone application. Additionally, Fitpolo provides SDK for Andorid (and IOS) devices (https://www.fitpolo.net/fitpolo-h701-sdk-android/) which supports development of smartphone applications.

In this exercise we will develop a smartphone application which will:

- ask for Bluetooth permissions
- establish Bluetooth connectivity with the wristband
- read data from the wristband
- adjust the sampling frequency based on the user's activity
- update UI based on the user's activity

To shortcut the development process, you can download starting code from https://bitbucket.org/msfrita/lab_5 This code is based on the Fitpolo Android SDK App source code (available here). Since this app is written in Java, we will also work in Java for this lab assignment.

Download and import your project. During the import or the first build, you might be prompted to upgrade your Android Gradle Plugin. Please **do not upgrade it**! It will cause compatibility issues and the app might not build anymore.

The code contains the Fitpolo SDK which has several components. The most important are:

- Several Android **Activities** (example usage of the SDK).
- Two **Adapter** classes (e.g., contain specialized Fitpolo ListAdapter interfaces)
- Four **Service** classes. Service is an application component that can perform long-running operations in the background (e.g., communication with the wristband), and it doesn't provide a user interface.

Bluetooth permissions

First, check if the manifest contains the necessary Bluetooth permissions. If not add them:

```
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
```

In the **OnCreate()** of the **myMainActivity**, the function calls **checkBluetoothConnection()**. This checks whether the smartphone Bluetooth is enabled. If it is, it should call **startNextActivity().** If it is not, it should prompt the user to enable the Bluetooth. The following code does that. Copy it in the function **checkBluetoothConnection().**

```
if (!MokoSupport.getInstance().isBluetoothOpen()) {
    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, AppConstants.REQUEST_CODE_ENABLE_BT);
    return;
}
startNextActivity();
```

The results of the user's action (e.g., whether she enables the Bluetooth or not) will be visible in the function: **onActivityResult()**. Update this function so that it calls **startNextActivity()** if the user has enabled the Bluetooth.

```
if (requestCode == AppConstants.REQUEST_CODE_ENABLE_BT) {
    startNextActivity();
}
```

Finally, you need to edit the function **startNextActivity().** It should:

- start the service **MokoService**. This will start the wristband's main service.
- start the Activity **BtScanActivity**. This will start the Activity for scanning Bluetooth devices.

Establish Bluetooth connectivity with the wristband

The activity **BtScanActivity** contains only one button "Scan", which activates the Bluetooth scanning. The discovered devices are shown in the list below the button. Once the user selects a device from the list, the function **onItemClick()** is called. In the function, we need to inform the user that the app is trying to connect and we need to initiate the connection. Add the following code to do that:

```
mDialog.setMessage("Connect...");
mDialog.show();
BleDevice device = (BleDevice) parent.getItemAtPosition(position);
saveToSharedPreferences(device);
initiateConnection(device);
```

The result of the **initiateConnection()** call will be visible in the **onReceive()** function of the **BroadcastReciever** mReciever.
If the app has successfully connected to the Bluetooth device we need to call the next Activity, **SmartSensing**. To do that, add the following code after the line:

```
Toast.makeText(BtScanActivity.this, "Connect success", Toast.LENGTH_SHORT).show();

Intent orderIntent = new Intent(BtScanActivity.this, SmartSensing.class);
orderIntent.putExtra("device", mDevice);
startActivity(orderIntent);
```

Smart sampling

In the Activity **SmartSensing**, there is a service **mServiceConnection** through which we will communicate with the wristband. Additionally, the **SmartSensing** activity will receive the data from the service through the BroadcastReceiver **mReciever.**

To make sure that you are connected to the right wristband, program the button "**shakeMyBand**" from the layout **mReciever** to call the function:

```
MokoSupport.getInstance().sendDirectOrder(new ZWriteShakeTask(mService));
```

This function will cause the wristband to vibrate for a few seconds.

Infer user's activity and adjust sampling frequency

The wristband can send updates to a paired android application on a step-count change. To register our application for the step-count change updates, we need to register a specific listener. The listener should be registered once the **mService** is created and connected. Thus, in the function **onServiceConnected()** add the code that registers the step-count change listener.

```
MokoSupport.getInstance().sendOrder(new ZOpenStepListenerTask(mService));
```

The step-count updates will be received through the **onReceive** function **BroadcastReceiver mReciever**. The specific part of the code that provides the updates is:

```
case Z_STEPS_CHANGES_LISTENER:
   Toast.makeText(getApplicationContext(),"Steps changed",Toast.LENGTH_SHORT).show();
```

Thus, if you start the application now, it should send a message **"Steps changed"** upon each step-count change. Try it!

The specific step count can be read from the wristband using:

```
DailyStep dailyStep = MokoSupport.getInstance().getDailyStep();
```

Update the code, so that after the Toast.*makeText()* performs the following:

- get current daily steps and save the info in the variable ***DAILY_STEPS***
- save the current time in the variable ***DAILY_STEPS_TIME***
- call **UpdateSensing()** to refresh the app

The function **UpdateSensing()** performs the following:
- if the user is sleeping, it sets the sampling frequency to ***SLEEP_SENSING***
- if the user is sedentary, it sets the sampling frequency to ***NORMAL_SENSING***
- if the user is active, it sets the sampling frequency to ***SPORTS_SENSING***
- finally, it updates the smartphone UI and the bandUI (if needed)

Next, you need to update the function **CalculateCardioZone()** which calculates the intensity of the user's activity. For the purpose of simplicity, the intensity is calculated as the number of

steps per second. You need to calculate the number of steps per second using the equation below and save it in the variable **STEPS_PER_SECOND.**
STEPS_PER_SECOND= **DAILY_STEPS**-**PREV_DAILY_STEPS**/time_passed_sesconds
Time_passed_sesconds = (**DAILY_STEPS_TIME**-**PREV_DAILY_STEPS_TIME**)/1000.0;

Check the rest of the code in the function **CalculateCardioZone()** and try to understand it. Why do we need the ***handler***? Try to remove it and see what happens with the app.

Update UI based on the user's activity

Finally, once we have the user's activity and the intensity of the activity, we need to update the smartphone UI and the band UI.

In the function **UpdateUI()** update the code so that it changes the color of the screen to GREEN, if the application is in ***SPORTS_SENSING*** mode, otherwise it should be white.

Update the function **UpdateBand(),** so that it sends a notification to the user every 30-35 seconds. Hints:
- currently, the application sends only one notification (the first one) to the band .
- the variable **BAND_UPDATE_LAST** keeps the time (in milliseconds) about when was the last notification sent to the user

The solutions should be committed to a private Bitbucket repository named **FRIMS2021-LAB-5** 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, April 10, 23:59**.

HAPPY CODING!