Android Summary

Lucas Waelti

November 11, 2018

Contents

1 Printing Statements to Logcat

```
private final String TAG = this.getClass().getName();
// A function printing to logcat
private void demo_logcat() {
    Log.v(TAG, "Verbose");
    Log.d(TAG, "Debug");
    Log.i(TAG, "Information");
    Log.w(TAG, "Warning");
    Log.e(TAG, "Error");
}
```

2 Android User Interace

The UI is composed of

- View objects (widgets as TextView, ImageView, Button, ...)
- ViewGroup objects (invisible view containers)

2.1 LinearLayout

In an XML layout:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" />
```

Using weihted spacing (Space example):

```
<Space
    android:layout_width="match_parent"
    android:layout_height="0dp"
    android:layout_weight="1"/>
```

2.2 ConstraintLayout

Example in the case of the watch:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent"
    android:background="@android:color/white"
    tools:deviceIds="wear">
</android.support.constraint.ConstraintLayout>
```

Use following constraints to place Views:

```
app:layout_constraintBottom_toTopOf="@id/aViewId"
app:layout_constraintLeft_toLeftOf="parent"
app:layout_constraintRight_toRightOf="parent"
app:layout_constraintTop_toTopOf="parent"
```

2.3 Other ViewGroups

RelativeLayout, GridLayout, FrameLayout, TableLayout, TableRow.

3 Callbacks

3.1 XML callbacks

From the XML layout file:

```
<Button
    ...
    android:id="@+id/button"
    android:onClick="clickedButtonXMLCallback" />
```

Then add the callback to the corresponding activity Java code:

```
public void clickedLoginButtonXmlCallback(View view) {
    TextView textView = findViewById(R.id.atextviewid);
    textView.setText("We used an XML callback!");
}
```

3.2 Java callbacks

More dynamic than XML callbacks. A Java callback is declared as follows in the Java source code:

```
@Override
protected void onCreate(Bundle savedInstanceState) {

    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    Button button = findViewById(R.id.RegisterButton);
    button.setOnClickListener(new View.OnClickListener() {

        @Override // Override when instantiating a new OnClickListener
        public void onClick(View view) {
```

```
            TextView textView = findViewById(R.id.LoginMessage);
            textView.setText("We used the Java callback!");
        }
    });
}
```

4 Activities and Intents

An activity can register for specifics events by declaring the **intent-filter** in the manifest as follows, with

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />
    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

4.1 Starting an activity for a result (explicit)

In the Activity class:

```
private static final int INTENT_ID = 1;

Intent intent =
new Intent(EmittingActivity.this,ReceivingActivity.class);
startActivityForResult(intent,INTENT_ID);
```

4.2 Starting an activity for a result (implicit)

In a given function:

```
Intent intent = new Intent();
intent.setType("image/*"); // Content is of type image/*
intent.setAction(Intent.ACTION_GET_CONTENT); // We want to get some content
// createChooser(...) defines the action to perform
startActivityForResult(Intent.createChooser(intent, "Select Picture"), INTENT_ID);
```

The Chooser allows to select the app that should be used to perform the action.

4.3 Retrieve Activity Results

Override the onActivityResult(...) method from the class AppCompatActivity.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);

    if (requestCode == INTENT_ID && resultCode == RESULT_OK) {
        Uri imageUri = data.getData(); // Get data from activity result
```

```
        ...
        // do some stuff...
    }
```

4.4 Sending back results (explicit)

Results can be sent back by doing the following:

```
Intent intent = new Intent(EmittingActivity.this,ReceivingActivity.class);
intent.putExtra("someInfos", instanceWithInfos); // Add supplementary data by putting
    Extras
setResult(AppCompatActivity.RESULT_OK, intent);
finish();
```

4.5 Get the data of an Intent

Retreive a String for instance (from a Fragment):

```
Intent intent = getActivity().getIntent();
String userID = intent.getExtras().getString(USER_ID);
```

5 Convert Uri to Bitmap and store it (image)

When getting a result from an intent, the data is indicated as a Uri. This form is not permanent and has to be converted to be then stored if necessary. For instance, for an image:

```
private File imageFile;

public void extractFromUri(Uri imageUri){
    imageFile = new File(getExternalFilesDir(null), "profileImage");

    try {
        copyImage(imageUri, imageFile);
    } catch (IOException e) {
        e.printStackTrace();
    }
    final InputStream imageStream;
    try {
        imageStream = getContentResolver().openInputStream(Uri.fromFile(imageFile));
        final Bitmap selectedImage = BitmapFactory.decodeStream(imageStream);
        ImageView imageView = findViewById(R.id.userImage);
        imageView.setImageBitmap(selectedImage);
    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }
}
```

With the copyImage(...) function that converts to a bitmap:

```
private void copyImage(Uri uriInput, File fileOutput) throws IOException {
        InputStream in = null;
        OutputStream out = null;

        try {
            in = getContentResolver().openInputStream(uriInput);
            out = new FileOutputStream(fileOutput);
            // Transfer bytes from in to out
            byte[] buf = new byte[1024];
            int len;
            while ((len = in.read(buf)) > 0) {
                out.write(buf, 0, len);
            }
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            in.close();
            out.close();
        }
    }
```

6 Convert image from a View into raw bytes for upload

The transformation can be done as follows, **data** is our output:

```
BitmapDrawable bitmapDrawable = (BitmapDrawable) ((ImageView)
    findViewById(R.id.userImage)).getDrawable();

Bitmap bitmap = bitmapDrawable.getBitmap();
ByteArrayOutputStream baos = new ByteArrayOutputStream();
bitmap.compress(Bitmap.CompressFormat.JPEG, 90, baos);
byte[] data = baos.toByteArray();
```

7 Android Wear

7.1 Idle display

To use the watch, add following lines to the manifest above <application>:

```
<uses-feature android:name="android.hardware.type.watch" />
```

Important to reduce energy consumption. In the activity java code that implements the watch, create following methods:

```
public class MainActivity extends WearableActivity {
    private TextView mTextView;
    private ConstraintLayout mLayout;
```

```
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        mTextView = (TextView) findViewById(R.id.textView);
        mTextView.setText("Hello Round World!");
        mLayout = findViewById(R.id.container);
        // Enables Always-on
        setAmbientEnabled();
    }
    @Override
    public void onEnterAmbient(Bundle ambientDetails) {
        super.onEnterAmbient(ambientDetails);
        updateDisplay();
    }
    @Override
    public void onExitAmbient() {
        super.onExitAmbient();
        updateDisplay();
    }
    private void updateDisplay() {
        if (isAmbient()) {
            mLayout.setBackgroundColor(getResources().getColor(android.R.color.black,
            getTheme()));
        } else {
            mLayout.setBackgroundColor(getResources().getColor(android.R.color.white,
            getTheme()));
        }
    }
}
```

Also make sure the manifest has the following permission:

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

7.2 Interfacing with Android Wear

This WearService is relying on constants generated at build time to prevent typing mistakes. The project's build.gradle files must be modified:

```
allprojects {
    repositories {
        ...
    }
    // Constants defined for all modules, to avoid typing mistakes
    // We use it for communication using the Wear API
    // It is a key-value mapping, auto-prefixed with "W_" for convenience
    project.ext {
        constants = [
            path_start_activity : "/START_ACTIVITY",
```

```
            path_acknowledge : "/ACKNOWLEDGE",
            example_path_asset : "/ASSET",
            example_path_text : "/TEXT",
            example_path_datamap : "/DATAMAP",
            mainactivity : "MainActivity",
            // Add all other required key/value paires required for the application below
            a_key : "a_value",
            some_other_key : "some_other_value",
        ]
    }
}
```

To make both mobile and wear modules aware of this, both their gradle files must be edited too:

```
android {
    ...
    buildTypes {
            ...
        buildTypes.each {
            project.ext.constants.each {
        // - String constants used in Java as `BuildConfig.W_a_key`
        // - Resources are used as usual:
        // - in Java as:
        // `[getApplicationContext().]getString(R.string.W_a_key)`
        // - in XML as:
        // `@string/W_a_key`
                k, v ->
                    it.buildConfigField 'String', "W_${k}", "\"${v}\""
                    it.resValue 'string', k, v
            }
        }
    }
}
```

The manifest needs as well some editing to register the service for both mobile and wear modules:

```
<service android:name=".WearService">
    <intent-filter>
        <action android:name="com.google.android.gms.wearable.DATA_CHANGED" />
        <data
            android:host="*"
            android:pathPrefix=""
            android:scheme="wear" />
    </intent-filter>

    <intent-filter>
        <action android:name="com.google.android.gms.wearable.MESSAGE_RECEIVED" />
        <data
            android:host="*"
            android:pathPrefix=""
            android:scheme="wear" />
```

```
    </intent-filter>
</service>
```

7.3 Using the Wear Service

The service uses two facets of the Wear API:

- Message API, a one-way communication mechanism that's good for remote procedure calls and message passing.
- Data API, which synchronizes between all connected devices (nodes) the data. The 2 kinds of data are:
  - **DataMap** (corresponds to the **Bundle** object sent between Intents) is an object which stores key-value associations. It rejects any type that cannot be transfered through the Wear API.
  - **Asset** (designed to contain binary data). In the service, we use it to serialize bitmap (image) data by compressing it as a PNG file, and creating the Asset from the raw bytes. Reading back the data is the same process in the other way: read and decode the bytes from the Asset as a PNG file to get the Bitmap object.

7.3.1 Four functions to interact with the WearService

```
public void sendStart(View view) {
    Intent intent = new Intent(this, WearService.class);
    intent.setAction(WearService.ACTION_SEND.STARTACTIVITY.name());
    intent.putExtra(WearService.ACTIVITY_TO_START, BuildConfig.W_mainactivity);
    startService(intent);
}

public void sendMessage(View view) {
    Intent intent = new Intent(this, WearService.class);
    intent.setAction(WearService.ACTION_SEND.MESSAGE.name());
    intent.putExtra(WearService.MESSAGE, "Messaging other device!");
    intent.putExtra(WearService.PATH, BuildConfig.W_example_path_text);
    startService(intent);
}

public void sendDatamap(View view) {
    int some_value = 420;
    ArrayList<Integer> arrayList = new ArrayList<>();
    Collections.addAll(arrayList, 105, 107, 109, 1010);
    Intent intent = new Intent(this, WearService.class);
    intent.setAction(WearService.ACTION_SEND.EXAMPLE_DATAMAP.name());
    intent.putExtra(WearService.DATAMAP_INT, some_value);
    intent.putExtra(WearService.DATAMAP_INT_ARRAYLIST, arrayList);
    startService(intent);
}
```

```
public void sendBitmap(View view) {
    // Get bitmap data (can come from elsewhere) and
    // convert it to a rescaled asset
    Bitmap bmp = BitmapFactory.decodeResource(
    getResources(), R.drawable.wikipedia_logo);
    Asset asset = WearService.createAssetFromBitmap(bmp);
    Intent intent = new Intent(this, WearService.class);
    intent.setAction(WearService.ACTION_SEND.EXAMPLE_ASSET.name());
    intent.putExtra(WearService.IMAGE, asset);
    startService(intent);
}
```

8 Fragments and Menus

Fragments are behaviours or portions of user interface in an Activity. A Fragment has its own layout and it lives in a ViewGroup inside the Activity's view hierarchy. There are 2 ways of adding a fragment:

- Declaring it inside the activity's layout file, as a fragment element, specifying the properties as if it were a view. The android:name specifies the Fragment class to instantiate.
- Programmatically, adding it through the **FragmentManager**, which manages fragments, such as adding or removing them from the activity.

8.1 Adding Fragments

1. Add a **Fragment** class to the package (New → Fragment(Blank)) and give a name to the **fragment's layout**.
2. Edit the **onCreateView**(...) method of the Fragment that will inflate it:

```
private View fragmentView;

@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle
    savedInstanceState) {
    // Inflate the layout for this fragment
    fragmentView = inflater.inflate(R.layout.my_fragment, container, false);

    // Do some stuff...

    return fragmentView;
}
```

3. The activity that constains the Fragment has to implement the interface **OnFragmentInteractionListener** by writing:

```
public class ActivityWithFragment implements
    MyFragmentClass.OnFragmentInteractionListener{
```

```
    ...

    @Override
    public void onFragmentInteraction(Uri uri) {

    }
}
```

Add as many implementation as there are Fragment classes that the activity should have. Generate the method **onFragmentInteraction**(...) as required by the interface.

4. Create a new Java class that extends a **FragmentStatePagerAdapter** (this is an implementation of a **PagerAdapter**). This will allow to manage an *arbitrary* number of Fragments. Implement following methods:

```
class SectionsStatePagerAdapter extends FragmentStatePagerAdapter {

    private final String TAG = this.getClass().getSimpleName();

    // List of fragments
    private final List<Fragment> mFragmentList = new ArrayList<>();
    // List of fragment titles
    private final List<String> mFragmentTitleList = new ArrayList<>();

    public SectionsStatePagerAdapter(FragmentManager fm) {
        super(fm);
    }
    @Override
    public Fragment getItem(int i) {
        return mFragmentList.get(i);
    }
    @Override
    public int getCount() {
        return mFragmentList.size();
    }
    public void addFragment(Fragment fragment, String title) {
        mFragmentList.add(fragment);
        mFragmentTitleList.add(title);
    }
    public int getPositionByTitle(String title) {
        return mFragmentTitleList.indexOf(title);
    }
    @Nullable
    @Override
    public CharSequence getPageTitle(int position) {
        return mFragmentTitleList.get(position);
    }
}
```

5. Setup the layout of the Activity containing the Fragments:

```
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/mainViewPager"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <android.support.v4.view.PagerTabStrip
        android:id="@+id/pagerTabStrip"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_gravity="top"
        android:background="#20B2AA"
        android:textColor = "#fff"
        android:paddingTop="15dp"
        android:paddingBottom="15dp" />

</android.support.v4.view.ViewPager>
```

PagerTabStrip adds the title tabs under the action bar and enables to swipe through the tabs.

6. Add the Fragments to the **SectionsStatePagerAdapter** and set the **ViewPager**. This is done in the **onCreate**(...) method of the activity containing the Fragments:

```
public class MyActivityWithFragments extends AppCompatActivity implements
    MFragment.OnFragmentInteractionListener{
    private final String TAG = this.getClass().getSimpleName();

    private MyFragment myFragment;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.my_activity_with_fragments);

        mSectionStatePagerAdapter = new
            SectionsStatePagerAdapter(getSupportFragmentManager());

        myFragment = new MyFragment();

        ViewPager mViewPager = findViewById(R.id.mainViewPager);
        setUpViewPager(mViewPager);

        // Set MyFragment as default tab once started the activity
        mViewPager.setCurrentItem(mSectionStatePagerAdapter.getPositionByTitle(
        getString(R.string.my_fragment_name)));
    }
    private void setUpViewPager(ViewPager mViewPager) {
        mSectionStatePagerAdapter.addFragment(myFragment,
            getString(R.string.my_fragment_name));
```

```
    }
}
```

8.2 Adding Action Bar Menus

A menu lets display buttons with important functions on top of the application display. To create a menu, do:

1. Add a res/menu folder (New → Android Resource Directory)
2. Add a new XML menu file (New → Menu Resource File)
3. Edit the XML file:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto">
    <item
        android:id="@+id/action_edit"
        android:icon="@android:drawable/ic_menu_edit"
        android:title="@string/edit_data"
        app:showAsAction="ifRoom" />
</menu>
```

   The option app:showAsAction=”ifRoom” allows to always show the menu item as a button in the app action bar.

4. In the **onCreate**(...) method of the **Fragment** that needs the menu, add the line:

```
setHasOptionsMenu(true);
```

5. In the same file (**Fragment** class), add the method:

```
@Override
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater) {
    super.onCreateOptionsMenu(menu, inflater);
    inflater.inflate(R.menu.my_menu, menu);
}
```

8.3 Reacting to menu interactions

1. In the Fragment that has the menu, override the method **onOptionsItemSelected**(...) to react when a button of the menu is pushed:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.action_edit:
            // do stuff...
```

```
            break;
        }
    return super.onOptionsItemSelected(item);
}
```

9 Toasts

A toast can be displayed by calling the line:

```
Toast.makeText(CurrentActivity.this, "String to display", Toast.LENGTH_SHORT).show();
// or LENGTH_LONG
```

10 Firebase

To setup anything using Firebase, go to Tools → Firebase and select whatever you need to activate. The gradle files need to be updated in order to embed the desired functionalities.

The dependencies of the gradle files will be updated automatically. Errors may occur though. The added packages are located in the **dependencies** section of each gradle files. Make sure first the following line in the gradle main file indicates the last version of the package:

```
classpath 'com.google.gms:google-services:4.1.0'
```

10.1 Add Internet permission

In the Manifest.xml file:

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

10.2 Write to Firebase Realtime Database

10.2.1 Create fields in the database

In the class that interacts with the database, add following declarations:

```
// Get instance of the database
private static final FirebaseDatabase database = FirebaseDatabase.getInstance();
// Create a new field in database
private static final DatabaseReference profileGetRef =
    database.getReference("field_name");
// Create an unique key under "field_name" that will then contain infos
private static final DatabaseReference profileRef = profileGetRef.push();
```

10.2.2 Uploading data to database

We take the example of the upload of a profile. In the class that interacts with the database:

```
private void addDataToFirebaseDB() {
    profileRef.runTransaction(new Transaction.Handler() {
        @NonNull
        @Override
        public Transaction.Result doTransaction(@NonNull MutableData mutableData){
            mutableData.child("username").setValue(userProfile.username);
            mutableData.child("password").setValue(userProfile.password);
            mutableData.child("height").setValue(userProfile.height_cm);
            mutableData.child("weight").setValue(userProfile.weight_kg);
            return Transaction.success(mutableData);
        }
        @Override
        public void onComplete(@Nullable DatabaseError databaseError, boolean b,
            @Nullable DataSnapshot dataSnapshot) {}
    });
}
```

Note: As the code becomes hard to read, we can refactor the **Transaction.Handler(){}** into its own function declaration, renaming it for instance **ProfileDataUploadHandler**, so that the code above becomes:

```
private void addProfileToFirebaseDB() {
    profileRef.runTransaction(new ProfileDataUploadHandler());
}
```

10.3 Upload data/image to Firebase Storage

Get reference to the right field in the storage:

```
StorageReference storageRef = FirebaseStorage.getInstance().getReference();
StorageReference photoRef = storageRef.child("photos").child(profileRef.getKey() +
    ".jpg");
```

Upload the data to the storage(Check the section 6 for image conversion into raw bytes):

```
UploadTask uploadTask = photoRef.putBytes(data);

uploadTask.addOnFailureListener(new OnFailureListener() {
    @Override
    public void onFailure(@NonNull Exception exception) {
        // Handle unsuccessful uploads
    }
}).addOnSuccessListener(new OnSuccessListener<UploadTask.TaskSnapshot>() {
    @Override
    public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
        // Handle successful uploads
    }
});
```

Note: Again, the **OnSuccessListener<>** can be refactored into its own function as a lot of code might be added here! Rename it for instance **PhotoUploadSuccessListener** for more clarity.

10.4 Retrieve URL from data successfully uploaded to a storage for later usage

Let's implement the **OnSuccessListener<>** from previous section to get its URL, so that the image can be found again!

```
private class PhotoUploadSuccessListener implements
    OnSuccessListener<UploadTask.TaskSnapshot> {
    @Override
    public void onSuccess(UploadTask.TaskSnapshot taskSnapshot) {
        Task<Uri> downloadUrl =
            taskSnapshot.getMetadata().getReference().getDownloadUrl().addOnSuccessListener(new
            OnSuccessListener<Uri>() {
            @Override
            public void onSuccess(final Uri uri) {
                // Save the URL wherever it is useful
                userProfile.photoPath = uri.toString();
            }
        });
    }
}
```

10.5 Read data from Firebase database

Go through a section of the database. For example, finding a given user:

```
final FirebaseDatabase database = FirebaseDatabase.getInstance();
final DatabaseReference profileRef = database.getReference("profiles");

final String usernameInput = "The_user_name";
final String passwordInput = "His_pathword";

String userID = "";

profileRef.addValueEventListener(new ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot dataSnapshot) {
        boolean notMember = true;
        for (final DataSnapshot user : dataSnapshot.getChildren()) {
            String usernameDatabase = user.child("username").getValue(String.class);
            String passwordDatabase = user.child("password").getValue(String.class);
            if (usernameInput.equals(usernameDatabase) &&
                passwordInput.equals(passwordDatabase)) {
                userID = user.getKey();
                notMember = false;
                break;
            }
        }
```

```
        if (notMember) {
            // Display error of message, user not known
        } else {
            Intent intent = new Intent(LoginActivity.this, MainActivity.class);
            intent.putExtra(MyProfileFragment.USER_ID, userID);
            startActivity(intent);
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError databaseError) {
    }
});
```

10.6 Read from Firebase Storage

Download an image from the database:

```
private void setUserImageAndProfileInfo() {
    // Reference to an image file in Firebase Storage
    StorageReference storageRef =
        FirebaseStorage.getInstance().getReferenceFromUrl(userProfile.photoPath);
    storageRef.getBytes(Long.MAX_VALUE).addOnSuccessListener(new
        OnSuccessListener<byte[]>() {
        @Override
        public void onSuccess(byte[] bytes) {
            if (isAdded()) {
                final InputStream imageStream;
                final Bitmap selectedImage = BitmapFactory.decodeByteArray(bytes, 0,
                    bytes.length);

                // Add the image to the Activity display for instance:
                ImageView imageView = fragmentView.findViewById(R.id.userImage);
                imageView.setImageBitmap(selectedImage);
            }
        }
    });
}
```

10.7 ListViews

When we implemented the listview in the lab, we wanted to display a list of our objects class (called "Recording"). There is a provided class in Android Studio called "ArrayAdapter", which is supposed to manage the displaying of an array of objects, but it can't manage objects of unknown class, so we had to do code own adapter (a private class inside our fragment or activity):

```
private class RecordingAdapter extends ArrayAdapter<Recording> {
    private int row_layout;
    RecordingAdapter(FragmentActivity activity, int row_layout) {
```

```
        super(activity, row_layout);
        this.row_layout = row_layout;
    }
    @NonNull
    @Override
    public View getView(int position, @Nullable View convertView,
    @NonNull ViewGroup parent) {
        //Reference to the row View
        View row = convertView;
        if (row == null) {
            //Inflate it from layout
            row = LayoutInflater.from(getContext()).inflate(row_layout,
            parent, false);
        }
        SimpleDateFormat formatter = new SimpleDateFormat("dd/MM/yyyy " +
            "hh:mm:ss", Locale.getDefault());
        ((TextView) row.findViewById(R.id.exerciseType)).setText(getItem
            (position).exerciseType);
        ((TextView) row.findViewById(R.id.exerciseDateTime)).setText
            (formatter.format(new Date(getItem(position)
                .exerciseDateTime)));
        ((TextView) row.findViewById(R.id.exerciseDevice)).setText
            (getString(R.string.smartwatch_switch_value, getItem
                (position).exerciseSmartWatch ? "yes" : "no"));
        ((TextView) row.findViewById(R.id.exerciseDevice2)).setText
            (getString(R.string.hr_belt_switch_value, getItem
                (position).exerciseHRbelt ? "yes" : "no"));
        return row;
    }
}
```

In the overriden getView() method we are setting the view of the row layout with the data we want. In our case, the data is coming from the recording objects, in particular from its fields.

After implementing this class, we need to set this adapter as the official adapter for our ListView. We do this in the onCreateView(...) of MyHistoryFragment by adding the following code:

```
listView = fragmentView.findViewById(R.id.myHistoryList);
adapter = new RecordingAdapter(getActivity(), R.layout
    .row_myhistory_layout);
listView.setAdapter(adapter);
```

10.8 Identified listeners (vs anonymous listeners)

Until now, we were getting data from Firebase by using an anonymous implementation of ValueEventListener, which does not have a name and lets you to declare and instantiate a class at the same time and use it only once. This is how it looked:

```
someRef.addValueEventListener(new ValueEventListener() {
    // Some methods to override
    // Some more code as a normal class would be implemented
```

```
});
```

When we want to have add a listener related to a certain class (recordings in the lab), and active in only one fragment (recordings history), we may want to do add this listener as an inner class as we did for the ArrayAdapter:

```
private class MyFirebaseRecordingListener implements ValueEventListener {
    @Override
    public void onDataChange(DataSnapshot dataSnapshot) {
        adapter.clear();
        for (final DataSnapshot rec : dataSnapshot.getChildren()) {
            final Recording recording = new Recording();
            recording.exerciseType = rec.child("exercise_type")
                .getValue().toString();
            recording.exerciseDateTime = Long.parseLong(rec.child
                ("datetime").getValue().toString());
            recording.exerciseSmartWatch = Boolean.parseBoolean(rec
                .child("switch_watch").getValue().toString());
            recording.exerciseHRbelt = Boolean.parseBoolean(rec
                .child("switch_hr_belt").getValue().toString());
            adapter.add(recording);
        }
    }
    @Override
    public void onCancelled(DatabaseError databaseError) {
        Log.v(TAG, databaseError.toString());
    }
}
```

Then we can register and unregister the listener on onPause() and on onResume() of the fragment. We do this because reading from an external source such as Firebase, might create problems if active fragments are trying to change UI elements while being detached because of a configuration change (tablet rotation for example, see Section 5):

```
@Override
public void onResume() {
    super.onResume();
    databaseRef = FirebaseDatabase.getInstance().getReference();
    mFirebaseRecordingListener = new MyFirebaseRecordingListener();
    databaseRef.child("profiles").child(idUser).child("recordings")
        .addValueEventListener(mFirebaseRecordingListener);
}
@Override
public void onPause() {
    super.onPause();
    databaseRef.child("profiles").child(idUser).child("recordings")
        .removeEventListener(mFirebaseRecordingListener);
}
```

10.9 Handling configuration changes

When we change the orientation of the tablet, the activity is destroyed and recreated. This can lead to data losses (typically for images). To avoid this we have to save the data that can be lost in a "Bundle" object. We can save an image like this:

```
@Override
protected void onSaveInstanceState(Bundle outState) {
    super.onSaveInstanceState(outState);
    outState.putParcelable("ImageUri", savedImageUri);
}
```

And we can retrieve the data like this:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...
    if (savedInstanceState != null) {
        savedImageUri = savedInstanceState.getParcelable("ImageUri");
        if (savedImageUri != null) {
            // Put the image in the ImageView,
            // as you did in onActivityResult(...)
        }
    }
}
```

11 Sensors

11.1 Permissions

First we need to add some permissions in AndroidManifest.xml.

To keep the watch awake while performing some processing:

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

To allow access to data coming from sensors that measure properties of the user's body:

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

And add this in the recording activity of the watch:

```
if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M &&
checkSelfPermission("android.permission.BODY_SENSORS") ==
    PackageManager.PERMISSION_DENIED) {
    requestPermissions(new String[]{"android.permission" +
        ".BODY_SENSORS"}, 0);
    }
```

11.2 Recording the HR

First, we need to launch the recording in the recording button callback:

```
if (switchWatch.isChecked()) {
    startRecordingOnWear();
}
```

Then, somewhere else outside onCreate():

```
private void startRecordingOnWear() {
    Log.d(TAG, "Entered smartwatch hr reading");
    Intent intentStartRec = new Intent(getActivity(),
        WearService.class);
    intentStartRec.setAction(WearService.ACTION_SEND
        .STARTACTIVITY.name());
    intentStartRec.putExtra(WearService
        .ACTIVITY_TO_START, BuildConfig
            .W_recordingactivity);
    getActivity().startService(intentStartRec);
}
```

Do not forget to add the "recordingactivity" key in the gradle of the project. For this key again, in the WearService of the wear module we add a case in the switch(data) in the onMessageReceived(...) method:

```
case BuildConfig.W_recordingactivity:
    Log.d(TAG, "Start recording message received");
    startIntent = new Intent(this, RecordingActivity.class);
    break;
```

Then, to save the recordings to Firebase, we add this at the end of the onClick callback of the newRecording button:

```
Intent intentStartLive = new Intent(getActivity(), ExerciseLiveActivity.class);
intentStartLive.putExtra(USER_ID, userID);
intentStartLive.putExtra(RECORDIND_ID, recordingKeySaved);
startActivity(intentStartLive);
```

11.3 Monitor data from sensor (on the watch)

First you have to extend WearableActivity and implement SensorEventListener. The second one is registered in the activity in onCreate():

```
SensorManager sensorManager = (SensorManager) getSystemService
    (MainActivity.SENSOR_SERVICE);
Sensor hr_sensor = sensorManager.getDefaultSensor(Sensor.TYPE_HEART_RATE);
sensorManager.registerListener(this, hr_sensor, SensorManager.SENSOR_DELAY_UI);
```

Then we can write the heart rate in a TextView with the next overriden method:

```
@Override
public void onSensorChanged(SensorEvent event) {
    TextView textViewHR = findViewById(R.id.hrSensor);
    if (textViewHR != null)
        textViewHR.setText(String.valueOf(event.values[0]));
}
```

The event object contains information about the new sensor data, including the accuracy, the sensor that generated it, the timestamp of generation, and the new data.

11.4 Sending HR from watch to tablet

First add a key and a path in the Gradle:

```
heart_rate_key : "heart_rate_key",
heart_rate_path : "/HEART_RATE_PATH",
```

Then, in the onSensorChanged(), we start an Intent by passing the HR data as an extra:

```
Intent intent = new Intent(RecordingActivity.this, WearService.class);
intent.setAction(WearService.ACTION_SEND.HEART_RATE.name());
intent.putExtra(WearService.DATAMAP_INT_HEART_RATE, heartRate);
startService(intent);
```

Then we have to add a case in the switch of onStartCommand() of the WearService in the wear module:

```
case HEART_RATE:
    putDataMapRequest = PutDataMapRequest.create(BuildConfig
        .W_heart_rate_key);
    putDataMapRequest.getDataMap().putInt(BuildConfig
        .W_heart_rate_key, intent.getIntExtra
    (DATAMAP_INT_HEART_RATE, -1));
    sendPutDataMapRequest(putDataMapRequest);
    break;
```

In the mobile WearService, in the onDataChanged(...) method, we add a case to handle the HR data.

```
case BuildConfig.W_heart_rate_path:
    int heartRate = dataMapItem.getDataMap().getInt
        (BuildConfig.W_heart_rate_key);
    intent = new Intent(ExerciseLiveActivity
        .ACTION_RECEIVE_HEART_RATE);
    intent.putExtra(ExerciseLiveActivity.HEART_RATE,
        heartRate);
    LocalBroadcastManager.getInstance(this).sendBroadcast
        (intent);
    break;
```

11.5 Showing data on the tablet

First, to get the intent extras in the activit'y onCreate(), to be able to access the specific branch:

```
Intent intentFromRec = getIntent();
userID = intentFromRec.getStringExtra(MyProfileFragment.USER_ID);
recID = intentFromRec.getStringExtra(NewRecordingFragment.RECORDIND_ID);
```

We get the info about the exercise from Firebase. In order to get the HR sensor data sent from the watch, we implement the LocalBroadcastManager with an IntentFilter specific to the HR we already defined (ACTION_RECEIVE_HEART_RATE, always remember to add these strings). The BroadcastReceiver for the HR data is registered in the onResume(...) and unregistered in the onPause(...).

```
@Override
protected void onResume() {
    super.onResume();
    //Get the HR data back from the watch
    heartRateBroadcastReceiver = new HeartRateBroadcastReceiver();
    LocalBroadcastManager.getInstance(this).registerReceiver
        (heartRateBroadcastReceiver, new IntentFilter
            (ACTION_RECEIVE_HEART_RATE));
}
@Override
protected void onPause() {
    super.onPause();
    LocalBroadcastManager.getInstance(this).unregisterReceiver
        (heartRateBroadcastReceiver);
    }

private int heartRateWatch = 0;

private class HeartRateBroadcastReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        //Show HR in a TextView
        heartRateWatch = intent.getIntExtra(HEART_RATE, -1);
        TextView hrTextView = findViewById(R.id.exerciseHRwatchLive);
        hrTextView.setText(String.valueOf(heartRateWatch));
    }
}
```

11.6 Live plot

First add the AndroidPlot library through the module's Settings panel (right click on the module (mobile, in our case), then 'Open Module Settings'), then in the 'Dependencies' tab, click on "+" to add another library, and search for 'androidplot'.

To add the plot in the layout:

```
<com.androidplot.xy.XYPlot
```

```
    android:id="@+id/HRplot"
    style="@style/APDefacto.Light"
    android:layout_width="fill_parent"
    android:layout_height="0dp"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="56dp"
    android:orientation="horizontal"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintHorizontal_bias="0.0"
    app:layout_constraintLeft_toLeftOf="parent"
    app:layout_constraintRight_toRightOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/gpsMap"
    app:lineLabels="left" />
```

Then, in onCreate() of the activity, we initiante the XYPlot:

```
heartRatePlot = findViewById(R.id.HRplot);
configurePlot();
```

Then we have to configure the plot (colors, styling, axis formatting) in a private method we call after initiating the XYPlot:

We are using here a String value "heart_ rate " which does not yet exist, so let's add it in the res/values/strings.xml file, giving it the value Heart Rate (bpm). We also are using the integer constants 'MIN_HR', 'MAX_HR' and 'NUMBER_OF_POINTS' which we need to define in the class. We will give them the respective values 40, 200 and 50