Android Summary

Lucas Waelti

October 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();
```

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 Android Wear

6.1 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" />
```

6.2 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>
```

6.3 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.

6.3.1 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);
}
```

7 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.

7.1 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));
    }
}
```

7.2 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);
}
```

7.3 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);
}
```