ECE155: Engineering Design with Embedded Systems Winter 2016

Lecture 13 — Android III

Patrick Lam

Android Persistent Storage

So far, we haven't seen any ways to store data so that you can re-load it across different instances of your application (for instance, when the phone is powered off and then back on). Here are four and a half ways to persist data, from simplest to most complicated:

- shared preferences;
- files (internal and external storage);
- SQLite; and
- the Internet.

We'll discuss the first two of these in this lecture. The Internet needs no explanation: you just save an load data remotely. A quick description of SQLite is: "SQLite is a software library that implements a self-contained, serverless, zero-configuration, transactional SQL [structured query language] database engine" [SQL14]. If you need a database rather than some simple storage, SQLite is the way to do it. For the programming you will need to do in the labs, this is overkill; shared preferences and file storage will be enough.

Shared Preferences

The simplest way to store persistent state is using *shared preferences* in Android. This is just a set of key-value pairs. Unlike the key-value pairs that you can store in the `Bundle` when you are implementing `onSaveInstanceState()`, these pairs persist across different invocations of your app.

Shared preferences can be private to your application or shared across applications; I'll only talk about private shared preferences.

You'll find source code for this demo in the SVN repository at `https://ecesvn.uwaterloo.ca/courses/ece155/s14/materials/lecture/SharedPreferenceDemo`. (By the way, I had to go to Project Properties | Android | Project Build Targets, deselect "EDK", and select "Android 2.3.3" to get the project to work after I imported the code from an existing directory.)

There are two important points: putting data into the shared preferences and getting it from the shared preferences. But first, how do we get our hands on the shared preferences? Here's how.

```
SharedPreferences settings = getPreferences(0);
```

Or, call `getSharedPreferences()` and pass it a preferences file name as the first parameter.

Reading data from shared preferences. Now that you have a `SharedPreferences` object, you can get data from it, mostly like you get data from a `Bundle`:

```
v = settings.getString("textFieldValue", "");
```

This retrieves the value of the `String` preference `textFieldValue`; if no value is present, the `getString` call returns the default value `""`.

Writing data to shared preferences. It's a bit more complicated to write to the `SharedPreferences` object. You have to first get a `SharedPreferences.Editor` and then commit it once you're done with the changes.

```
SharedPreferences settings = getPreferences(0);
SharedPreferences.Editor editor = settings.edit();
editor.putString("textFieldValue", newFieldValue);
editor.commit();
```

Files: Internal and External Storage

Android allows your app to read and write to internal storage; and also to external storage, if it has the appropriate permissions. You basically use the same calls to access internal and external storage, with a minor change. File I/O we have already examined in an earlier lecture.

All Android devices have internal storage, which is generally invisible to the user, other apps, and when the phone is mounted as USB Mass Storage. When you remove an app, Android automatically erases its internal storage. (It is in the `/data` directory of the phone under DDMS).

External storage may be on an SD card (as on many phones) or it may be enclosed in the device (as on the Google Nexus 7 tablet); it is not required to correspond to the SD card slot. Applications may access their own external storage space, shared storage space, or even other apps' storage space. However, external storage may go away anytime, since the user could just pull out the SD card from the slot.

Checking External Storage availability. The following code determines whether or not external storage is accessible and writable at the moment. Of course, that can change anytime.

```
[basicstyle=\scriptsize]
    String state = Environment.getExternalStorageState();

    if (Environment.MEDIA_MOUNTED.equals(state)) {
        // We can read and write the media
    } else if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(state)) {
        // We can only read the media
    } else {
        // Something else is wrong.
        // It may be one of many other states, but all we need
        //  to know is we can neither read nor write
    }
```

External Storage Example: MapLoader. Our provided Lab 4 code reads from external storage. We'll see what it does.

```
[basicstyle=\scriptsize]
    NavigationalMap map =
       MapLoader.loadMap(getExternalFilesDir(null),
```

```
                        "Lab-room-peninsula.svg");
    mapView.setMap(map);
```

Our `MapLoader` contains this line:

```
[basicstyle=\scriptsize]
  File map = new File(dir, filename);
```

and then it builds a parser using the library call:

```
[basicstyle=\scriptsize]
  doc = docBuilder.parse(map);
```

which we call to get specific information about the map. It is actually the `docBuilder` that does all the I/O.

Shared External Storage Directories. The `getExternalFilesDir()` call takes a parameter. `null` means that we're asking for the app's external storage directory. Android also provides a number of shared directories:

- DIRECTORY_MUSIC
- DIRECTORY_PICTURES
- DIRECTORY_RINGTONES

which all apps on the system can access.

Reading from Files The following code will read a line of text into the `String` variable `i`. It currently reads from internal storage, but if you comment out the `openFileInput` line and uncomment the `new FileInputStream` line, it'll read from external storage instead.

```
[basicstyle=\scriptsize]
try {
    // internal storage:
    FileInputStream os = openFileInput("internal.txt");
    //InputStream os = new FileInputStream(new File(getExternalFilesDir(null), "external.txt"));
    BufferedReader br = new BufferedReader(new InputStreamReader(os));
    String i = br.readLine();
    // --> i contains the line you just read.
    os.close();
} catch (IOException e) {}
```

Writing to Files. You can use the following code to write to the filesystem. Again, the external storage code will work if you comment out the `openFileOutput` call for internal storage and uncomment the `new FileOutputStream` call for external storage.

```
[basicstyle=\scriptsize]
try {
    // internal storage:
```

```
    FileOutputStream os = openFileOutput("internal.txt", Context.MODE_PRIVATE);
    //FileOutputStream os = new FileOutputStream(new File(getExternalFilesDir(null), "external.txt"));
    PrintWriter osw = new PrintWriter(new OutputStreamWriter(os));
    // --> write out the contents of string i.
    osw.println(i);
    osw.close();
} catch (IOException e) {}
```

Digression: About the Lab. First, a quick preview of the next lab (which you are free to start working on). Since this course is "Engineering Design *with Embedded Systems*," we'd better actually learn something practical about embedded systems.

This lab is about interpreting sensor data. In particular, you are going to read the accelerometer data and count the number of steps taken by the holder of the phone. There are two main problems: 1) sensor data is noisy; and 2) you need to recognize when a step occurs.

To deal with sensor noise, you will probably benefit from smoothing the data. Below is a very simple low-pass filter:

```
smoothedAccel += (newValue - smoothedAccel) / C;
```

To recognize a step, you'll need to identify a change in the value of the y-axis acceleration. Identifying a change means that you'll need the previous value along with the current value. You could do that using a finite state machine (as described in the lab manual), or you could do that by doing tests both on the previous value and the current value. Can you think of any pattern that might describe a step?

Toast, Broadcast Receivers, and Lists

Let us now continue with some more Android material.

Toast. Sometimes you want to display a short message to the user. Use `Toast` to do that. Just include the following code:

```
Toast.makeText(getApplicationContext(), "A Toast!",
               Toast.LENGTH_LONG).show();
```

It's also OK to use in your onClick event listener:

```
Toast.makeText(MainActivity.this, "A Toast!",
               Toast.LENGTH_LONG).show();
```

You can store all of these parameters, as well as the `Toast` object itself, in local variables.

Broadcast Receivers

We talked in an earlier lecture about the subject of Android Intents.

Android also uses Intents to broadcast information about what's happening on the system between applications. Applications want to know about events such as the phone being plugged into a power

source; or, screen rotation. One tutorial [Fra12] proposes an analogy of a "party line" for Android Broadcast Receivers: many different applications can register an event listener, and they all get notified whenever something happens.

Example: rotation listener. We can create a simple `Activity` which uses an inner class to define a `BroadcastReceiver`:

```
public class MainActivity extends Activity {
  BroadcastReceiver broadcastReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context c, Intent i) {
      int orientation = c.getResources().getConfiguration().orientation;
      if (orientation == Configuration.ORIENTATION_PORTRAIT) {
        Toast.makeText(c, "Portrait", Toast.LENGTH_SHORT).show();
      } else if (orientation == Configuration.ORIENTATION_LANDSCAPE) {
        Toast.makeText(c, "Landscape", Toast.LENGTH_SHORT).show();
      } else {
        Toast.makeText(c, "???", Toast.LENGTH_SHORT).show();
      }
    }
  };
  IntentFilter intentFilter = new IntentFilter
                              (Intent.ACTION_CONFIGURATION_CHANGED);

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    registerReceiver(broadcastReceiver, intentFilter);
  }
}
```

In principle, you should also unregister the `BroadcastReceiver`:

```
  @Override
  protected void onDestroy() {
    unregisterReceiver(broadcastReceiver);
    super.onDestroy();
  }
```

Unfortunately, by default Android destroys your app on a screen rotate event, so that doesn't work unless you ask Android to not destroy your app on rotation.

Example: phone ring listener We'll see another example of setting up a broadcast receiver, this time to listen for phone calls. First, you need to modify the `manifest.xml` file to permit the app to listen for phone calls:

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

This time, we'll also choose to register the listener statically in the manifest. Include inside the `<application>` tag:

```
<receiver android:name="ca.patricklam.ece155demo.MyPhoneReceiver" >
    <intent-filter>
        <action android:name="android.intent.action.PHONE_STATE" />
    </intent-filter>
</receiver>
```

This means that we have to create a separate class `MyPhoneReceiver`:

```
package ca.patricklam.ece155demo;

public class MyPhoneReceiver extends BroadcastReceiver {
  @Override
  public void onReceive(Context context, Intent intent) {
    Bundle extras = intent.getExtras();
    if (extras != null) {
      String state = extras.getString(TelephonyManager.EXTRA_STATE);
      Log.w("PHONE", state);
      if (state.equals(TelephonyManager.EXTRA_STATE_RINGING)) {
        String phoneNumber = extras.getString
                              (TelephonyManager.EXTRA_INCOMING_NUMBER);
        Log.w("PHONE", phoneNumber);
      }
    }
  }
}
```

Note the use of the extras on the incoming Intent object.

Lists

Another useful UI element is the `ListView`. We can use it to show a list of items to the user (for instance, so that the user can choose one of the list elements). There are a couple of caveats with using the `ListView`. Let's see how to use it.

Creating and populating a ListView First, create a `ListView` object by dragging it onto the Activity's XML file.

- Note: you have to manually edit the `android:id` attribute so that its value is `@android:id/list`.

Next, change your Activity to be a ListActivity. This will allow us to set the items of the ListView. Also, you need to add a field `listAdapter` of type `ArrayAdapter<String>` to your Activity. (The error in class was that I added an object of type `ListAdapter`.)

We then want to actually populate the list with entries. In the `onCreate` method, add:

```
  List<String> data = new ArrayList<String>();
  data.add("ECE155");
  data.add("ECE106");
  data.add("ECE124");
  listAdapter = new ArrayAdapter<String>(this,
                           android.R.layout.simple_list_item_1,
                           data);
  setListAdapter(listAdapter);
```

Finally, we want something to happen when we click on a list item. Create a new method in the ListActivity:

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
  super.onListItemClick(l, v, position, id);
  String s = (String) getListAdapter().getItem(position);
  Toast.makeText(this, "Aha: "+s, Toast.LENGTH_SHORT).show();
}
```

This will cause the phone to display a toast when the user chooses a list item.

Dynamically updating the ListView. Of course, we can also add items to the `ListView` programmatically. In a click listener (or anywhere else), you can write:

```
String now = String.valueOf(System.currentTimeMillis());
listAdapter.add(now);
```

Some notes. Note that adding elements to the ArrayAdapter also adds them to the ListActivity.

Also, we currently store the ArrayAdapter as a field. That means it'll go away whenever the Activity is destroyed (e.g. rotation). We'll want to do something about that.

References

[Fra12] William J. Francis. An android coder's introduction to broadcast receivers. `http://www.techrepublic.com/blog/software-engineer/an-android-coders-introduction-to-broadcast-receivers/1173/`, 2012. Online; accessed 23-April-2014.

[SQL14] SQLite. About SQLite. `http://sqlite.org/about.html`, 2014. Online; accessed 12-April-2014.