ECE155: Engineering Design with Embedded Systems Winter 2013

Lecture 25 — March 15, 2013

Patrick Lam *version 1*

Android Friday: 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 today. I started talking about SQLite in class, but I'll say more about that in its own lecture.

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/w13/materials/lecture/L24/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 from 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 is quite similar to that in C#.

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.

```
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. Recall that you call:

```
NavigationalMap map =
   MapLoader.loadMap(getExternalFilesDir(null),
                     "Lab-room-peninsula.svg");
mapView.setMap(map);
```

Our `MapLoader` contains this line:

```
File map = new File(dir, filename);
```

and then it builds a parser using the library call:

```
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.

File Demo. You'll find source code for this demo in the SVN repository at `https://ecesvn.uwaterloo.ca/courses/ece155/w13/materials/lecture/L24/FileDemo`.

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.

```
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.

```
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) {}
```

(In lecture I had some bogosity about writing to a `byte` array. I figured out how to get rid of it.)