ECE155: Engineering Design with Embedded Systems Winter 2013

Lecture 13 — February 1, 2013

Patrick Lam *version 2*

Note about Assignment 4. Many students have reported that they can't get a value back from the second Activity that they launch. You need to use `startActivityForResult()` to launch the second Activity if you're hoping to get a result back.

Digression: About the Lab. First, a quick preview of next week's 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. There's a line of code on the bottom of line 3 which implements a 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?

Android Friday: Toast, Broadcast Receivers, and Lists

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

Last Friday, we talked about Intents and how they can launch an Activity and return a value from the launched Activity; you're doing this for Assignment 4 (do it!)

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[1] 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.

[1] `http://www.techrepublic.com/blog/app-builder/an-android-coders-introduction-to-broadcast-receivers/1173`, accessed January 31, 2013; code doesn't actually work.

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 to 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.

Although this example didn't work in class, there's nothing wrong with the code. I ran it again afterwards and it worked fine.

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.