**ECE155: Engineering Design with Embedded Systems** Winter 2013

# Lecture 10 — January 25, 2013

*Patrick Lam* *version 1*

Today's Android Friday is going to be about `Intent`s and also about saving and restoring state.

## Programming Tips

First, a programming tip for object-oriented systems. Sometimes, you have code in a class, say an `EventListener`, which needs access to the object that created it, say the `MainActivity`. Add a field to the `EventListener` with a reference to the `MainActivity`, as follows:

```
public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    // ...
    LightSensorEventListener el = new LightSensorEventListener(this);
  }
}

class LightSensorEventListener implements SensorEventListener {
  MainActivity m;

  public LightSensorEventListener(MainActivity m) {
    this.m = m;
  }

  // ...
}
```

A second tip: don't put your app in the `com.example` namespace. Put it in `ca.uwaterloo`.

## Intents

So far, we've seen Activities in isolation, and that'll be all you need for the labs. However, real apps often have more than one Activity. Today's topic is going to be about how to link Activities. Android uses Intents.

> An *Intent* specifies a request or describes an event.

**Examples: Starting another Activity.** The simplest possible Intent explicitly names the Activity it would like to start.

```
Intent intent = new Intent(this, OtherActivity.class);
startActivity(intent);
```

**Examples: Requests.** The Map activity puts up hyperlinks to webpages and phone numbers. Upon click, the Map broadcasts either the web browser Intent or the call Intent and trusts that some other application will handle the Intent.

**Examples: Events.** The system broadcasts an Intent when the phone goes into airplane mode.

**How Intents Work.** One component broadcasts an Intent. Then, 0 or more components receive the Intent. Android may pick a component to act upon the Intent.

**Parts of an Intent.** Each intent contains an *action* which represents the requested action, along with optional data. For instance:

| | |
|---|---|
| ACTION_MAIN | Launch an activity |
| ACTION_DIAL | Dial a phone number |
| ACTION_SEARCH | Perform a search |

The two following code fragments yield identical Intents with an action (but no data):

```
new Intent(Intent.ACTION_EDIT);    |  Intent intent = new Intent();
                                   |  intent.setAction(Intent.ACTION_EDIT);
```

The *data* is the payload of the event. Android intent data is in URI (Universal Resource Identifier) format[1]. The obvious thing to put into data is a web address, but other data formats are possible as well:

```
new Intent(Intent.ACTION_DIAL,           | Intent intent =
                                         |   new Intent(Intent.ACTION_DIAL);
          Uri.parse("tel:6175551212")); | intent.setData(
                                         |   Uri.parse("tel:6175551212"));
```

You can also tell Android what type of data to expect it to find at the URI, using the `setType` method. This is optional. Example: `intent.setType("audio/mp3");`

Beyond the data, Intents may also contain *extras*. The extras consist of key-value pairs: they contain more information than what you can easily put into a URI. Here's an example.

```
Intent intent = new Intent(Intent.ACTION_SEND);
intent.putExtra(android.content.Intent.EXTRA_EMAIL,
  new String[] {
    "p.lam@ece.uwaterloo.ca", "root@uwaterloo.ca"
  });
```

(Key-value pairs are a key concept you'll encounter over and over, by the way.)

Finally, Intents may also contain *flags*, which modify how the Intent gets launched and how it will be processed by the recipient. They don't affect which activity gets launched. Examples: `FLAG_ACTIVITY_NO_HISTORY` and `FLAG_ACTIVITY_RESET_TASK_IF_NEEDED`.

[1] What's a URI? It's like a URL. The exact distinction is unimportant for ECE155.

## Intent Resolution

We've seen how to explicitly name which Activity we want to launch, and I've alluded to how we can implicitly launch an activity by describing what we're looking for. In either case, we use the `startActivity` method to launch the intent.

In the case of implicit intent resolution, the system searches the available Activities on a system, using the Intent's action, data and category.

**Requesting Intent Resolution.** Let's first look at an example of requesting an image from any suitable Activity on the system[2]:

```
private static final int REQUEST_CODE = 1;

public void pickImage(View View) {
  Intent intent = new Intent();
  intent.setType("image/*");
  intent.setAction(Intent.ACTION_GET_CONTENT);
  intent.addCategory(Intent.CATEGORY_OPENABLE);
  startActivityForResult(intent, REQUEST_CODE);
}
```

This specifies a requested type, action, and category, and asks that some other activity returns an image. We'll see how to process activity results below.

**Responding to Intent Resolution Requests.** In your application's manifest, there's an XML tag for each activity. That tag can take an *intent filter* describing the Intents that the activity is prepared to handle. For example:

```
<activity
    android:name=".BrowserActivity"
    android:label="@string/app_name" >
    <intent-filter>
        <action android:name="android.intent.action.VIEW" />

        <category android:name="android.intent.category.DEFAULT" />

        <data android:scheme="http" />
    </intent-filter>
</activity>
```

This activity is able to view any URI that begins with `http`.

[2] http://www.vogella.com/articles/AndroidIntent/article.html, accessed January 25, 2013

**Receiving Activity results.** When you launch a sub-Activity, sometimes you're hoping for a result back from that activity. For instance, you may be asking for the user to pick a contact from the contact list. Or you may be asking the user for a favourite colour. The sub-activity should create a new Intent and call its `setResult` method:

```
Intent retval = new Intent();
retval.putExtra("color", "blue");
setResult(RESULT_OK, retval);
```

Then in the caller, we provide a new callback, `onActivityResult()`.

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
  if (resultCode == RESULT_OK && requestCode == YOUR_REQUEST_CODE) {
    if (data.hasExtra("color")) {
      String favouriteColor = data.getExtras().getString("color");
    }
 }
}
```

The `resultCode` is an `int` you provided while starting the sub-activity. It allows you to distinguish different sub-activities you may have started.

In the case of the image example above, the appropriate set of statements is:

```
if (bitmap != null) {
  bitmap.recycle();
}
stream = getContentResolver().openInputStream(data.getData());
bitmap = BitmapFactory.decodeStream(stream);
```

## Saving and Restoring State

Sometimes, Android kills your activity but brings it back later. You want the activity to have the same data when it comes back from the dead. This will happen by default for anything in a UI element, but not for any fields stored in the activity (like, say, step counts). In your activity, implement the callback `onSaveInstanceState()` to save the data:

```
class MainActivity ... {
  String color;

  @Override
  protected void onSaveInstanceState(Bundle b) {
    super.onSaveInstanceState(b);
    b.putString("color", color);
  }
}
```

The Bundle `b` is another key/value map. You can then restore whatever data you saved in the `onCreate` method of the same Activity:

```
class MainActivity ... {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    if (savedInstanceState != null)
      color = b.getString("color");
  }
}
```