Android UI Tools

CS 4720 – Mobile Application Development

Resource: developer.android.com
UI vs. UX

• An important distinction to make up front is the difference between UI and UX

• UI – User Interface
  – The components of a piece of software that a user will interact with
  – The design and layout of those components

• UX – User Experience
  – The entire package of software + hardware
UI vs. UX

• We are concerned with both of these, but will first focus on UI

• UX has a bit more to do with the handset + display + processing capabilities + network + ...

• Our apps will depend on these things to have a good UX, but let’s start with a good UI
Views and ViewGroups

• The default components of a base UI in Android are Views and ViewGroups

• A View is an object that draws something on the screen and the user can interact with

• A ViewGroup is an object that holds other Views (or ViewGroups) together in a particular order and defines the layout of those components in the interface
Views and ViewGroups

• In general:
  – ViewGroups are your layout XML files
  – Views are everything that goes in the layout XML files

• ViewGroups and Views can be defined in either the layout XML files or in the code base itself

• A ViewGroup is loaded into a tree hierarchy for display and querying
Views and ViewGroups
Layouts

- Layouts can be defined in either XML or in code

- Why do it in one vs. the other?
  - XML: Good to separate display from business logic for reusability, distribution of labor, etc.
  - Code: Good for dynamic changes

- The wording and terms in the XML and in code look and behave similarly (also to Swing...)

Building a Layout

• To create a layout:
  – You can write the XML yourself (fun...)
  – You can generate the XML using a builder (there are other builders besides the Android Studio builder...)
  – You can do it all in the onCreate() of the Activity (bad for several reasons)
  – You can add to it in later calls in the Activity
  – You can do a mix of all of these
Building a Layout
Accessing Views

• Every View in the UI is assigned a unique integer ID

• Like most global/static/final identifiers, we don’t ever want to write the actual value or know its actual position in memory
Accessing Views

android:id="@+id/my_button"

where @ tells Android to expand this and + means this resource should be added to the R file

android:id="@android:id/entity"

means get the built-in Android thing called entity
Connecting a View to Code

In the Layout XML:

```xml
<Button android:id="@+id/my_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="@string/my_button_text"/>
```

In the Android code:

```java
Button myButton = (Button) findViewById(R.id.my_button);
```

Remember: you can also do this with `android:onClick`!
So Many Layouts...

- Linear Layout – all children are in a row (vertical or horizontal)
- Relative Layout – each item is positioned according to the position of the others
- Table Layout - ... it’s a table with rows and columns
- Absolute Layout – (x,y) coordinates, basically
- Frame Layout – single screen
- And other Views (List, Group) that are similar
Which Layout Do I Use?

• You should make different layouts for different gross categories (i.e. phone vs. tablet) of devices and for vertical vs. horizontal

• Consider:
  – Which device and orientation will the user be in?
  – How will the user be holding the device? One hand or two?
  – Where will the user be (standing, sitting, walking, etc.)?
  – Where should important functions be?
Typical Layouts

• Linear
  – Lists of things is pretty common...

• Relative
  – Really good for changing device sizes as components are dynamically allocated

• Table
  – Good for data presentation

• Absolute
  – Typically not a good option...
Dynamic Layouts

• You’re going to make a list (Linear Layout, List View, etc.) but you don’t know until runtime how many items will be in the list

• How do you dynamically allocate items in the layout?
Adapters

- An Adapter is a class that “hooks together” an AdapterView (like ListView) to a data source.
- Subclasses of Adapter hook up to different types/formats of data:
  - ArrayAdapter looks at arrays, ArrayLists, etc.
  - SimpleCursorAdapter looks at Cursor class (reading 2D data for example).
Adapter

Allocate an Adapter against the layout and data source:

```java
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, myStringArray);
```

Find the view that will show the data and call setAdapter():

```java
ListView listview = (ListView) findViewById(R.id.listview);
listview.setAdapter(adapter);
```

To change how the data is shown in each list item, override toString() in the objects in the array.
Adapter

• To handle clicks on items in the list:

```java
// Create a message handling object as an anonymous class.
private OnItemClickListener mMessageClickedHandler = new OnItemClickListener()
{
    public void onItemClick(AdapterView parent, View v, int position, long id) {
        // Do something in response to the click
    }
};

listView.setOnItemClickListener(mMessageClickedHandler);
```
Building a Basic ListView