

Layouts

Dr. Abraham Campbell
University College Dublin
Abey.campbell@ucd.ie

Layouts?

- In Android, Layouts define the visual structure
- for user interfaces
- Layouts can be declared in two ways:
- Declared UI elements in XML
- Instantiated at runtime

XML vs Code

In other words:

- with XML you create your UI that directly corresponds to the View classes and subclasses automatically
- or you create View and ViewGroup objects manually in your code
- => the XML way is the currently more popular but sometimes coding the layout makes more sense

Why XML?

- XML usage for layouts is not required, but
- highly recommended!
- Advantages:
- GUI-assisted creation
- Separation of generated vs hand-written code
- Allows a UX specialist to work on layout without having to worry about underlying code

Implications?

- XML is compiled into very efficient form for runtime, so performance is not a problem
- Your interfaces should be known beforehand. Manipulating XML interface at runtime is tricky.

Sample Layout (Button)

- `<?xml ... ?>`
- `<Button xmlns:android="..."`
- `android:id="@+id/button"`
- `android:text=""`
- `android:layout_width="fill_parent"`
- `android:layout_height="fill_parent"`
- `/>`

How to Attach the XML layout ?

In your activity:

```
setContentView(R.layout.main);
```

i.e.

```
@Override
```

```
public void onCreate(Bundle icle) {
```

```
    super.onCreate(icle);
```

```
    setContentView(R.layout.main);
```

Text View

```
<?xml version="1.0" encoding="utf8"?>  
<TextView xmlns:android="..."  
    android:layout_width="fill_parent"  
    android:layout_height="wrap_content"  
    android:text="You were expecting  
    something profound?"  
/>
```


ImageView

```
<?xml version="1.0" encoding="utf8"?>  
<ImageView xmlns:android="..."  
    android:id="@+id/icon"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    android:adjustViewBounds="true"  
    android:src="@drawable/molecule"  
/>
```

EditView

```
<?xml version="1.0" encoding="utf8"?>  
<EditText xmlns:android="..."  
    android:id="@+id/field"  
    android:layout_width="fill_parent" *  
    android:layout_height="fill_parent"  
    android:singleLine="false"  
> * note on fill_parent
```

CheckBox

```
<?xml version="1.0" encoding="utf8"?>  
<CheckBox xmlns:android="..."  
  android:id="@+id/check"  
  android:layout_width="wrap_content"  
  android:layout_height="wrap_content"  
  android:text="This checkbox is:  
unchecked" />
```

RadioButton and RadioGroup

```
<RadioButton  
    android:id="@+id/radio1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:text="Rock" />
```

Part of RadioGroup

RadioGroup

```
<?xml version="1.0" encoding="utf8"?>  
<RadioGroup  
  xmlns:android="..."  
  android:orientation="vertical"  
  android:layout_width="fill_parent"  
  android:layout_height="fill_parent"  
> ... add radio buttons here  
</RadioGroup>
```

Views

- All of the above widgets extend Views.
- Useful properties:
 - android:nextFocusDown
 - android:nextFocusLeft
 - android:nextFocusRight
 - android:nextFocusUp
 - android:visibility

Views Cont'd

- Useful Methods:
 - getParent(): Finds the parent widget or container.
 - findViewById(): Finds a child widget with a certain ID.
 - getRootView(): Gets the root of the tree (e.g., what you provided to the activity via setContentView()).

Containers

- Container is a collection of widgets and child containers
- Similar to BorderLayout in Swing

LinearLayout

- Resembles Swing's BorderLayout
- Properties
 - Orientation: horizontal, vertical
 - Match model: width, height, wrap, fill
 - Weight: which widget has more of it?
 - Gravity: alignment
 - Padding: top, bottom, left, right...

LinearLayout Example

```
<?xml version="1.0" encoding="utf8"?>  
<LinearLayout xmlns:android="..."  
    android:orientation="vertical"  
    android:layout_width="fill_parent"  
    android:layout_height="fill_parent"  
    > ... your widgets here ...  
</LinearLayout>
```

RelativeLayout

- Layouts widgets based on their relationship to other widgets in the container and the parent container
- To make your RelativeLayout work, you need ways to reference other widgets within an XML layout file, plus ways to indicate the relative positions of those widgets.

RelativeLayout Example

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
xmlns:android="..."
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:padding="5px">
<TextView android:id="@+id/label"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="URL:"
android:paddingTop="15px"/> ... </RelativeLayout>
```

TableLayout

- Table Layout works in conjunction with TableRow. TableLayout controls the overall behavior of the container, with the widgets themselves poured into one or more TableRow containers, one per row in the grid.

TableRow Example

```
<TableRow>
```

```
<TextView android:text="URL:" />
```

```
<EditText
```

```
android:id="@+id/entry"
```

```
android:layout_span="3"/>
```

```
</TableRow>
```

TableLayout Example

```
<?xml version="1.0" encoding="utf8"?>  
<TableLayout  
  xmlns:android="..."  
  android:layout_width="fill_parent"  
  android:layout_height="fill_parent"  
  android:stretchColumns="1">  
  <TableRow>...
```

ScrollView

- ScrollView is a container that provides scrolling for its contents. You can take a layout that might be too big for some screens, wrap it in a ScrollView
- Remember to add ScrollView first to avoid issues with your page.
- There is HorizontalScrollView as well...

ScrollView Example

```
<?xml version="1.0" encoding="utf8"?>  
<ScrollView  
  xmlns:android="..."  
  android:layout_width="fill_parent"  
  android:layout_height="wrap_content">  
  <TableLayout  
    android:layout ...
```

Fragments

- Introduced in Android 3.0
- Gives the developer more control
- Allows for greater flexibility in dealing with different screen sizes
- Designed to create UI reusability
- Will be covered in future lectures as the need for Fragments will not be apparent until you have done more development.

Filling in Layouts Programmatically

- Remember that for everything you do in XML
- Can be done in code.
- Of using XML

```
import android.widget.LinearLayout;
```

```
.....
```

```
LinearLayout mylayout = new LinearLayout(this);  
mylayout.setOrientation(LinearLayout.VERTICAL);  
setContentView(MyLinearLayout);
```

Fill in lists programmatically

- LISTVIEW

```
import android.widget.ListView;
```

```
.....
```

```
lv = new ListView(this);
```

```
lv= (ListView)findViewById(R.id.list);
```

You must give it an Adaptor which will manage the list for you.

```
lv.setAdapter(myAdaptor)
```

Simple Adaptor

```
android.widget.SimpleAdapter.SimpleAdapter(Context context,  
                                           List<? extends Map<String, ?>> data,  
                                           int resource,  
                                           String[] from,  
                                           int[] to)
```

Parameters:

context The context where the View associated with this SimpleAdapter is running
data A List of Maps. Each entry in the List corresponds to one row in the list. The Maps contain the data for each row, and should include all the entries specified in "from"

resource Resource identifier of a view layout that defines the views for this list item. The layout file should include at least those named views defined in "to"

from A list of column names that will be added to the Map associated with each item.

to The views that should display column in the "from" parameter. These should all be TextViews. The first N views in this list are given the values of the first N columns in the from parameter.

Example

```
SimpleAdapter adapter = new SimpleAdapter(  
    this,  
    Tdata,  
    R.layout.fruitlistview,  
    new String[]{"image","name"},  
    new int[]{R.id.imageview1,  
        R.id.name});  
  
lv.setAdapter(adapter);  
lv.setOnItemClickListener(new ListItemClickListener());
```

Example of OnItemClickListener

```
class ListItemClickListener implements OnItemClickListener{
```

```
@Override
```

```
public void onItemClick(AdapterView<?> adapterView, View view,  
    int position, long id) {
```

```
    Intent intent = new Intent(FruitList.this, detail.class);
```

```
    intent.putExtra("name",(String)Tdata.get(position).get("name")) );
```

```
    intent.putExtra("Bigimage",  
        (Integer)Tdata.get(position).get("Bigimage"));
```

```
    startActivity(intent);
```

```
}
```

Generating your own Map<String,object>

```
Map<String, Object> data = new HashMap<String,  
Object>();
```

Putting objects use the

```
put(“key”,String ToBeInserted)
```

Similar to both Adding to Bundles and Shared Preferences