

Layouts (and dynamic view switching)

CE881: Mobile and Social Application Programming

Simon Lucas & Spyros Samothrakis

January 20, 2015

- 1 Interesting Cultural Artefacts
- 2 Layouts
- 3 LinearLayout
- 4 Threads and Content Switching

Movies, Books and Websites

- Design of everyday things
 - Great book on usability
- <http://androidniceties.tumblr.com/>
 - Collection of screenshots of good looking apps
- Minority Report
 - User Interface

Apps

- Apps that (I think) look great
 - Cookbook - Beautiful Recipes
 - Uber
 - DuoLingo
 - Inbox by Gmail
 - Reddit News Pro

Layouts

- Layouts are concerned with organising component views
 - i.e. allocating each child component a rectangular area of the parent view
 - The parent view could be fixed or scrollable
 - Each rectangular area is normally non-overlapping
- We've already used a LinearLayout
 - Let's look at this in a bit more detail
 - And also some other Layout types

Importance of Layouts (1)

- Very important
- Difficult to get right
- Challenge: must cope with
 - Different screen resolutions and aspect ratios
 - Different device orientations

Importance of Layouts (2)

- MUST be functional in all cases
 - No missing components
- SHOULD look good too
 - Evenly spaced child views
- Be appropriately sized
 - Text not too big or too tiny

LinearLayout Concepts

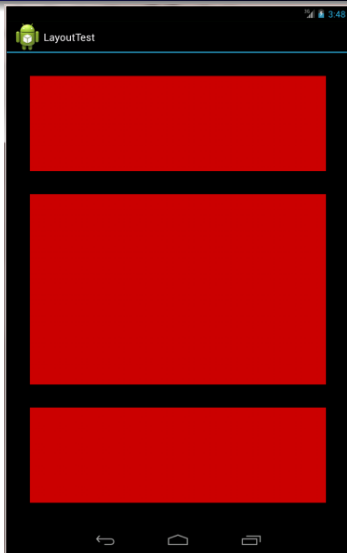
- LinearLayout Orientation: vertical or horizontal
 - This is distinct from device orientation
 - You may or may not want to make it dependent on device orientation
- Child View properties:
 - Width and Height
 - wrap_content or match_parent
 - set number in units of **px** or **dp**
 - Gravity
 - Each child view within a view can specify it's gravity, which is where it is attracted to (e.g. left, centre or right for a horizontal orientation)
 - Margins: set child margins to provide clear separation and better appearance

Example: Evenly Spaced Children

- We'll work through a common case: we want the children in the layout to fill the available space
- Each one should have a defined proportion of the space; proportions do not have to be equal
- In this case we'll have three components make the middle component twice the size of the others
- We'll use the default View class
- And set it's background color in the XML layout file

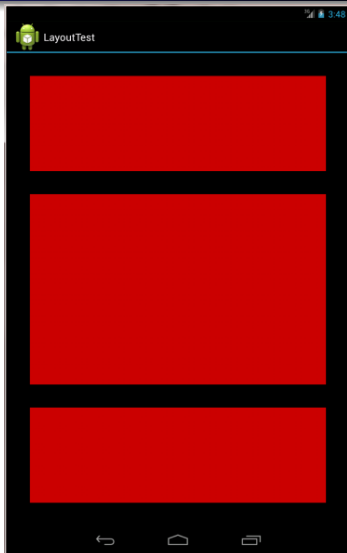
LinearLayout (1): Vertical

- On the next slide we'll see the XML for this
- Set height of each child to zero (0dp)
- Set weight of each one in proportion to vertical space it takes



LinearLayout (2): Vertical

- In this case 1 : 2 : 1
- Set width to the **match_parent** (aka **fill_parent**)
- Use margins to make it look good
- But if used naively in a landscape mode it may lead to poor proportions as we'll see ...



LinearLayout (3): XML

```
<LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_margin="20dp"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <View android:layout_width="fill_parent"
        android:layout_height="0dp"
        android:layout_margin="20dp"
        android:layout_weight="1"
        android:background="@android:color/holo_red_dark"
        android:layout_gravity="right|center_vertical"/>
    <View android:layout_width="fill_parent"
        android:layout_margin="20dp"
        android:layout_height="0dp"
        android:layout_weight="2"
        android:background="@android:color/holo_red_dark"
        />
    <View android:layout_width="fill_parent"...>
</LinearLayout>
```

The Vertical Layout in Device Landscape Mode

- This may or may not be what we need

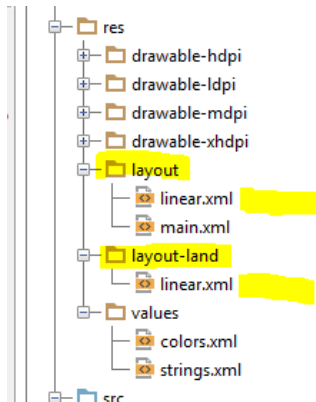


Adding a Landscape XML file

- Place a layout file of the same name (e.g. linear.xml) in the res/land-Layout directory
- This might be a converse version of the original portrait one
 - E.g. swap width for height

Adding a Landscape XML file

- Place a layout file of the same name (e.g. linear.xml) in the res/layout-Landscape directory
- This might be a converse version of the original portrait one
 - E.g. swap width for height



Note how we switched height and width declarations

```
<LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_margin="20dp"
    xmlns:android="http://schemas.android.com/apk/res/android">
    <View android:layout_height="fill_parent"
        android:layout_width="0dp"
        android:layout_margin="20dp"
        android:layout_weight="1"
        android:background="@android:color/holo_red_dark"
        android:layout_gravity="right|center_vertical"/>
    <View android:layout_height="fill_parent" ...>
    <View android:layout_height="fill_parent" ...>
</LinearLayout>
```

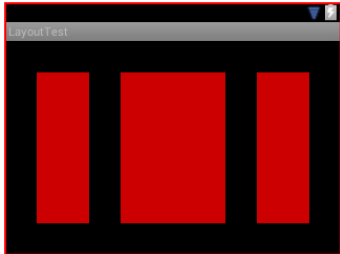

onCreate behaviour

- Note: the onCreate method will use the correct orientation automatically
- The call to setContentView will automatically choose the correct version of **R.layout.linear**

```
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.linear);  
}
```

New Look

- Now with the layout-land/linear.xml
 - The proportions look more natural, but not necessarily what we need
 - This will be application specific
 - Main point to note here is that we can define the appropriate layout in XML
- This will be automatically selected in the onCreate method when called by name



More pain

- Still Need to Be Careful!
- Suppose we now have Views that do something, such as buttons with text
- Be careful to avoid this, but how?
- DISCUSS!!!



Switching Layouts at Runtime

- Sometimes it may be necessary to switch a layout in response to some user activity
- Time-based application events can trigger layout changes
- After some elapsed time (e.g. show a Splash screen before the main app screen)
- After a file has loaded
- However, there is a problem to be overcome. . .

Threads...

- Threads are hard - really hard
- Hard to debug
- A necessary evil
- GUI events spawn new threads
- Users things apps have frozen if they wait too long

UI Thread

- The UI (User Interface) thread is what calls the onCreate() method of your main activity
- And also any event handling methods
- From this thread it is okay to “touch” a view (i.e. update or modify it in some way)
- This includes setting new content
- Consider the next example:
 - We now have two Layouts, **linear.xml**, and **button.xml**
 - And use event handling to switch between them

Content Switching (1)

- Methods in myActivity

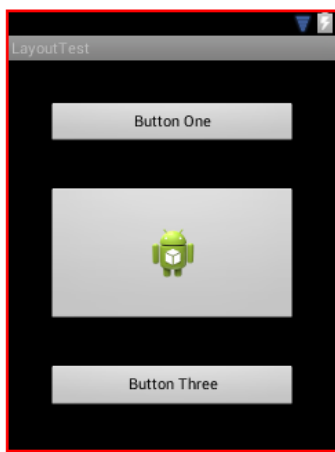
```
public void handleButtonOne(View view) {  
    setContentView(R.layout.linear);  
}
```

```
public void handleButtonTwo(View view) {  
    setContentView(R.layout.buttons);  
}
```

- With button clicks defined in the XML, e.g.

```
<Button android:layout_width="fill_parent"  
        android:layout_height="0dp"  
        android:onClick="handleButtonOne"
```

Content Switching (2)



Hacking onCreate()

```
try {
    Thread thread = new Thread() {
        @Override
        public void run() {
            try {
                Thread.sleep(2000);
                handleButtonOne(null);
            } catch (Exception e) {
                System.out.println("MyThreadTest Inner Exception: " + e);
            }
        }
    };
    thread.start();
} catch (Exception e) {
    System.out.println("MyThreadTest Outer Exception: " + e);
    e.printStackTrace();
}
```

Kabooooom!

- This results in an exception
- *MyThreadTest Inner Exception: android.view.ViewRootImpl\$CalledFromWrongThreadException: Only the original thread that created a view hierarchy can touch its views.*

Solution: Run the task on the UI Thread

- Several ways of doing this
- Simplest (and probably best) is to use an `AsyncTask`
- Write a class that extends `AsyncTask`
 - Typically override at least:
 - **`doInBackground()`**
 - **`onPostExecute()`**
 - Can also override: **`publishProgress()`**
 - Useful to update progress bars when loading files

Example AsyncTask (inner class of MyActivity)

```
private class MyTask extends AsyncTask<Void, Void, Void> {  
    @Override  
    protected Void doInBackground(Void... voids) {  
        try {  
            Thread.sleep(2000);  
        } catch (Exception e) {}  
        return null;  
    }  
  
    protected void onPostExecute(Void result) {  
        // new Toast(this)  
        handleButtonOne(null);  
    }  
}
```

The new onCreate method...

- Note how the argument types passed to the constructor must match the declared types (see previous slide)
- In this case they are never used, and null can be passed for each one

```
@Override  
public void onCreate(Bundle savedInstanceState) {  
    super.onCreate(savedInstanceState);  
    setContentView(R.layout.buttons);  
    new MyTask().execute(null, null, null);  
}
```

Recap

- So far we can do some powerful things
 - We've defined custom views (lab 1 and (more) on lab 2)
- Learned how to lay them out effectively in linear layouts
- Handled onClick and onTouch events
- Switched views at runtime
- Used AsyncTask derived objects to perform tasks on the UI thread
- Next we need to gain a better understanding of good app design
- Knowledge of the Activity lifecycle will be needed for this