

Testing, Layouts (and dynamic view switching)

CE881: Mobile and Social Application Programming

Spyros Samothrakis

January 18, 2015

1 / 39

Interesting Cultural Artefacts

Testing

Layouts

LinearLayout

Threads and Content Switching

2 / 39

MOVIES, BOOKS AND WEBSITES

► Design of everyday things

- Great book on usability

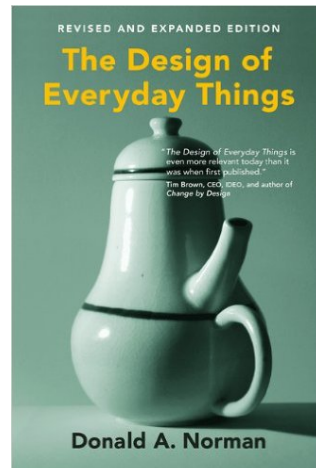
►

<http://androidniceties.tumblr.com/>

- Collection of screenshots of good looking apps

► Minority Report

- User Interface
- Augmented reality app?



3 / 39

KEYBOARD PROPAGANDA (1)

IntelliJ IDEA Default Keypmap

Editing	
Ctrl + Space	Basic code completion (the name of any class, method or variable)
Ctrl + Shift + Space	Smart code completion (filters the list of methods and variables by expected type)
Ctrl + Shift + Enter	Complete statement
Ctrl + P	Parameter info (shows method call arguments)
Ctrl + Q	Quick documentation lookup
Shift + F1	External doc
Ctrl + mouse over code	Hover info
Ctrl + F1	Show line actions of error or warning at caret
Alt + Insert	Generate code... (Getters, Setters, Constructors, hashCode(), equals(), toString())
Ctrl + O	Override methods
Ctrl + I	Implement methods
Ctrl + Alt + T	Surround with... if else, try catch, for, synchronized, etc.
Ctrl + /	Comment/uncomment with line comment
Ctrl + Shift + /	Comment/uncomment with block comment
Ctrl + W	Extract method (moving code blocks)
Ctrl + Shift + W	Increase current selection to previous state
Alt + Q	Comment info
Alt + Enter	Show intention actions and quick fixes
Ctrl + Alt + L	Reformat code
Ctrl + Alt + O	Optimize imports
Alt + I	Auto-indent lines
Tab	Indent current selected lines
Ctrl + X, Shift + Delete	Cut current line or selected block to clipboard
Ctrl + C, Ctrl + Insert	Copy current line or selected block to clipboard
Ctrl + V, Shift + Insert	Paste from clipboard
Ctrl + Shift + V	Paste from recent buffers
Ctrl + D	Duplicate current line or selected block
Ctrl + Y	Delete line at caret
Ctrl + Shift + Y	Smart line split
Ctrl + Enter	Smart line split
Shift + Enter	Smart line split
Ctrl + Shift + U	Toggle case for word at caret or selected block
Ctrl + Shift + J	Select all code block and start
Ctrl + Delete	Delete to word end
Ctrl + Backspace	Delete to word start
Ctrl + NumPad++	Expand collapse code block
Ctrl + Shift + NumPad++	Expand all
Ctrl + Shift + NumPad--	Collapse all
Ctrl + F4	Close active editor tab

www.jetbrains.com/idea

IntelliJ IDEA Default Keypmap

Usage Search	
Alt + F7, Ctrl + F7	Find usages / Find usages in file
Ctrl + Shift + F7	Highlight usages in file
Ctrl + Alt + F7	Show usages
Compile and Run	
Ctrl + F9	Make project (compile modified and dependent)
Ctrl + Shift + F9	Compile selected file, package or module
Alt + Shift + F10	Select configuration and run
Alt + Shift + F9	Select configuration and debug
Shift + F10	Run
Shift + F9	Debug
Ctrl + Shift + F10	Run current configuration from editor
Debugging	
F8	Step over
F7	Step into
Shift + F7	Step out
Alt + F8	Run to cursor
Alt + F8	Evaluate expression
Ctrl + F8	Resume program
Ctrl + Shift + F8	Toggle breakpoint
Ctrl + Shift + F8	View breakpoints
Navigation	
Ctrl + N	Go to class
Ctrl + Shift + N	Go to file
Ctrl + Alt + Shift + N	Go to symbol
Alt + Right, Left	Go to next/previous editor tab
F12	Go back to previous tool window
Esc	Go to editor (from tool window)
Ctrl + Shift + F4	Close active run/messages/debugger tab
Ctrl + G	Go to line
Ctrl + E	Recent files popup
Ctrl + Alt + Left, Right	Navigate back/forward
Ctrl + Shift + Backspace	Navigate to last edit location
Alt + F1	Select current file or symbol in any view
Ctrl + B, Ctrl + Click	Go to declaration
Ctrl + Alt + B	Go to implementation(s)
Ctrl + Shift + I	Open quick reference lookup
Ctrl + Shift + B	Go to type declaration
Ctrl + B	Go to super method/super class
Alt + Up, Down	Go to previous/next method
Ctrl + J	Move to code block and start
Ctrl + F12	File structure popup
Ctrl + Shift + H	Type hierarchy
Ctrl + Shift + H	Method hierarchy
Alt + H	Call hierarchy
F2, Shift + F2	Navigation/highlighted error
F4 / Ctrl + Enter	Edit source / View source
Alt + Home	Show navigation bar
F11	Toggle bookmark
Alt + Home	Toggle bookmark with messages
Ctrl + (G, B)	Go to numbered bookmark's
Shift + F11	Show bookmarks

blog.jetbrains.com/idea

IntelliJ IDEA Default Keypmap

Refactoring	
Copy	Copy
Move	Move
Alt + Delete	Safe Delete
Shift + F6	Rename
Ctrl + F6	Change signature
Ctrl + Alt + N	Invert
Ctrl + Alt + M	Extract Method
Ctrl + Alt + V	Extract Variable
Ctrl + Alt + F	Extract Field
Ctrl + Alt + C	Extract Constant
Ctrl + Alt + P	Extract Parameter
VCS/Local History	
Ctrl + G	Commit project to VCS
Ctrl + T	Update project from VCS
Alt + Shift + C	View recent changes
Alt + BackQuote (`)	VCS quick popup
Live Templates	
Ctrl + Alt + J	Surround with Live Template
Ctrl + J	Insert Live Template
Alt + J	Iteration according to Java SE 1.5 style
Alt + J	Check object type with instanceof and instanceof C
Alt + J	Iterate elements of java.util.Collection
Alt + J	Iterate elements of java.util.List
Alt + J	public static final
Alt + J	throw new
General	
Alt + Shift + J	Open corresponding tool window
Alt + Shift + J	Save all
Ctrl + Alt + V	Synchronize
Ctrl + Shift + F12	Toggle maximizing editor
Alt + Shift + F	Add to Favorites
Alt + Shift + F	Inspect current file with current profile
Ctrl + BackQuote (`)	Quick switch current scheme
Ctrl + Alt + S	Open settings dialog
Ctrl + Alt + Shift + S	Open Project Structure dialog
Ctrl + Shift + A	Find Action
Ctrl + Tab	Switch between tabs and tool window

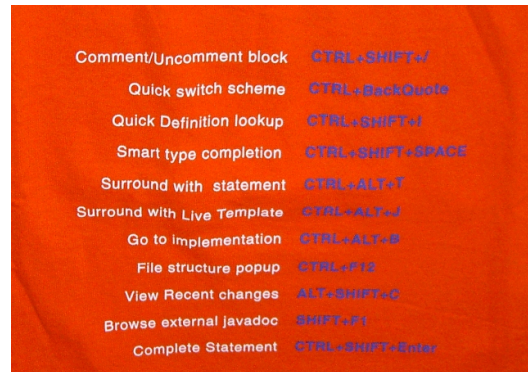
@intellijidea



4 / 39

KEYBOARD PROPAGANDA (2)

- ▶ Learn how to touch type
- ▶ Ctrl+Shift+A (Meta - search for shortcut/action)
- ▶ Ctrl+B (Go to declaration)
- ▶ Ctrl+U (Go to superclass)
- ▶ Ctrl+J (Insert template)



<http://stackoverflow.com/questions/294167/what-are-the-most-useful-intellij-idea-keyboard->

5 / 39

APPS

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

6 / 39

TESTING?

- ▶ An app is not considered complete before testing
- ▶ A method of confirming that your code does what it is expected to do
- ▶ Broadly, three kinds of tests
 - ▶ Functional tests
 - ▶ Unit tests
 - ▶ Integration tests
- ▶ But these are ad-hoc categories

7 / 39

WHY UNIT TESTS?

- ▶ How do you know if what you have done works or not?
 - ▶ Buggy apps going to full deployment
- ▶ Multiple platforms to deploy to - how do you know if your app works in all of them?
 - ▶ Mostly commercial tools to address this
- ▶ What impact does a change in one part of the code have in the rest?
 - ▶ Good software is tested exhaustively
- ▶ Ideally one would have a fully automated cycle of development-testing-deployment

8 / 39

ANDROID/JUNIT

- ▶ The standard method of unit testing in Java is JUnit

```
@RunWith(AndroidJUnit4.class)
@LargeTest
public class MainActivityInstrumentationTest {

    @Rule
    public ActivityTestRule mActivityRule = new ActivityTestRule<>(
        MainActivity.class);

    @Test
    public void sayHello(){
        onView(withText("Say hello!")).perform(click());

        onView(withId(R.id.textView)).check(matches(withText("Hello, World!")));
    }
}
```

9 / 39

JUNIT

- ▶ Notice the heavy use of Aspect Oriented Programming (AOP) features
- ▶ “Interceptors”
<http://developer.android.com/reference/android/support/test/rule/ActivityTestRule.html>
- ▶ Again, apps that have no automated testing break often
- ▶ Embrace change!

10 / 39

MORE THAN GUIs

- ▶ You can simulate most events:
 - ▶ Swipes
 - ▶ Clicks
 - ▶ Text input
- ▶ Should be part of your gradle lifecycle
- ▶ What about external resources?
- ▶ <https://coveralls.io/> - Coverage?

http://developer.android.com/tools/testing/testing_android.html

<http://developer.android.com/tools/testing/testing-tools.html>

11 / 39

CONTINUOUS INTEGRATION

- ▶ Git commit
- ▶ Compilation
- ▶ Test Run
- ▶ Report
- ▶ Travis CI

12 / 39

TEST-DRIVEN DEVELOPMENT

- ▶ Might a good idea to write tests first
- ▶ Why?

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

17 / 39

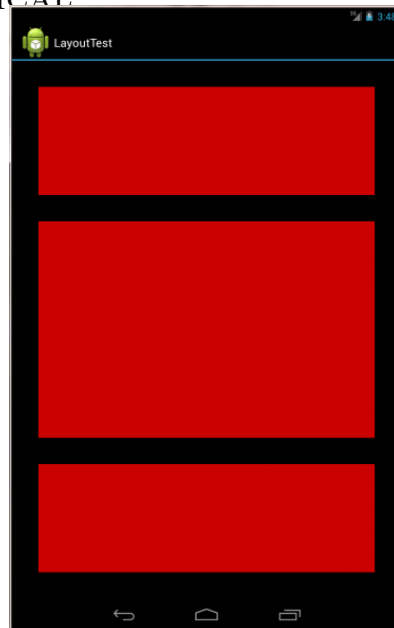
EXAMPLE: EVENLY SPACED CHILDREN

- ▶ 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

18 / 39

LINEARLAYOUT (1): VERTICAL

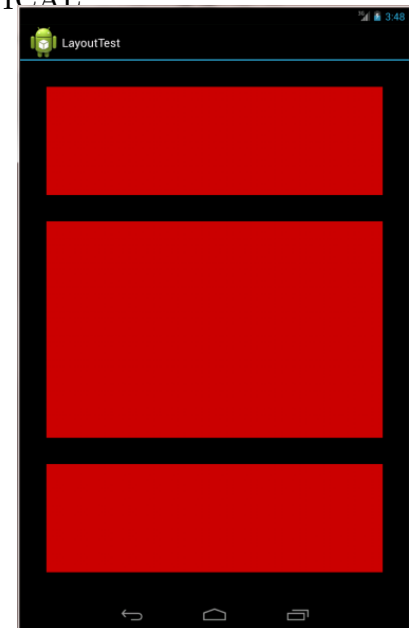
- ▶ On the next side we'll see the XML for this
- ▶ Set height of each child to zero (0dp)
- ▶ Set weight of each one in proportion to vertical space it takes



19 / 39

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 ...



20 / 39

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>
```

21 / 39

THE VERTICAL LAYOUT IN DEVICE LANDSCAPE MODE

- This may or may not be what we need



22 / 39

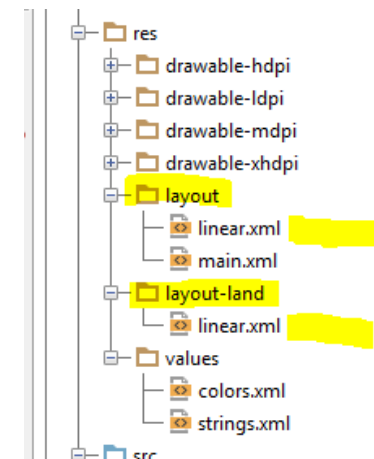
ADDING A LANDSCAPE XML FILE

- 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

23 / 39

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



24 / 39

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>
```

25 / 39

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);
}
```

26 / 39

NEW LOOK

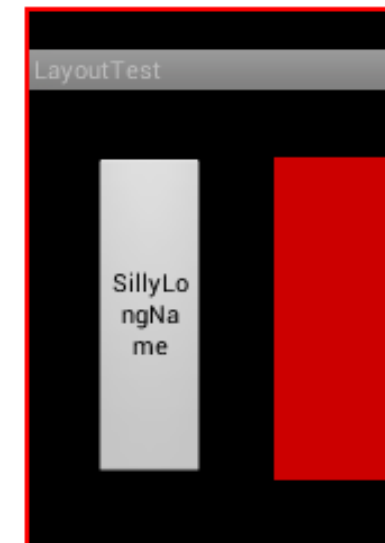
- Now with the layout-land/linear.xml
- The proportions look more natural, but not necessarily what we need
- This will be application specific
- Main point to note here is that we can define the appropriate layout in XML



27 / 39

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!!!



28 / 39

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...

29 / 39

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

30 / 39

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

31 / 39

CONTENT SWITCHING (1)

- ▶ Methods in myActivity

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

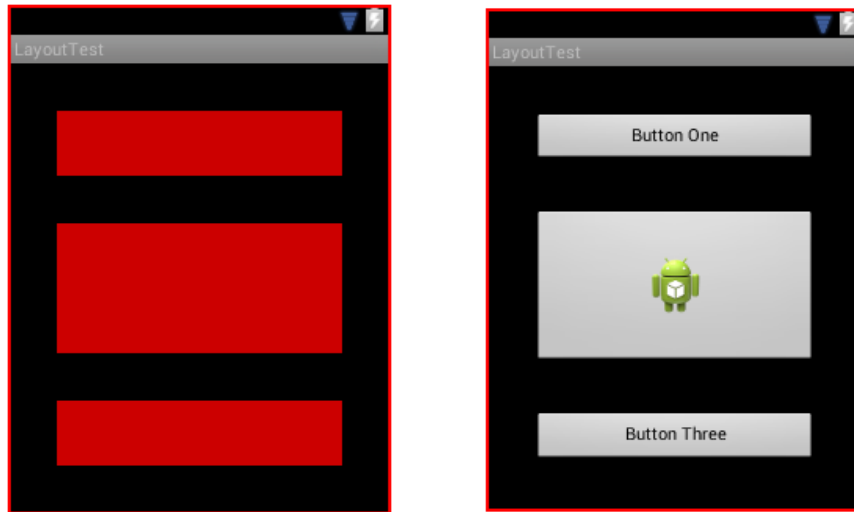
```
public void handleButtonTwo(View view) {
    setContentView(R.layout.buttons);
}
```

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

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

32 / 39

CONTENT SWITCHING (2)



33 / 39

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();
}
```

34 / 39

KABOOOOM!

- ▶ 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.*

35 / 39

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

36 / 39

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);
    }
}
```

37 / 39

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);
}
```

38 / 39

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
- Some of the slides based on Simon Lucas previous course

39 / 39