

Android 101



HackWeek Code Class - Q1 2013

Jan Chong
Twitter for Android, Manager
@lessachu

Mohammad Almalkawi
Twitter for Android, Engineer
@moh

Pre-Requisites

- Does everyone have intelliJ or Eclipse installed?
 - intelliJ available here: <http://download.jetbrains.com/idea/ideaIC-12.0.1.dmg>
- Does everyone have the Android SDK installed?
 - http://dl.google.com/android/android-sdk_r21.0.1-macosx.zip
 - Install:
 - Android SDK tools
 - Android SDK Platform-tools
 - Android 4.2 (API 17) Documentation, SDK Platform, Google APIs, Sources
 - ARM EABI v7a System Image
- Campfire Room: **androidclass**

Syllabus

Day 1	Day 2
INTRO TO ANDROID <ul style="list-style-type: none">• The State of Android• Intro to Dev Tools and Debugging• Hello World	DYNAMIC UI <ul style="list-style-type: none">• Using Fragments• Class Exercise
LUNCH	LUNCH
APPLICATION BASICS <ul style="list-style-type: none">• Activities• Intents• Activity Lifecycle• AdapterViews and Adapters	DATA PERSISTENCE <ul style="list-style-type: none">• SQLite Tools• Database APIs and Cursors• Content Provider
BREAK	BREAK
LAB <ul style="list-style-type: none">• Create TODO App	LAB <ul style="list-style-type: none">• <i>Class Choice: Advanced Topic</i>

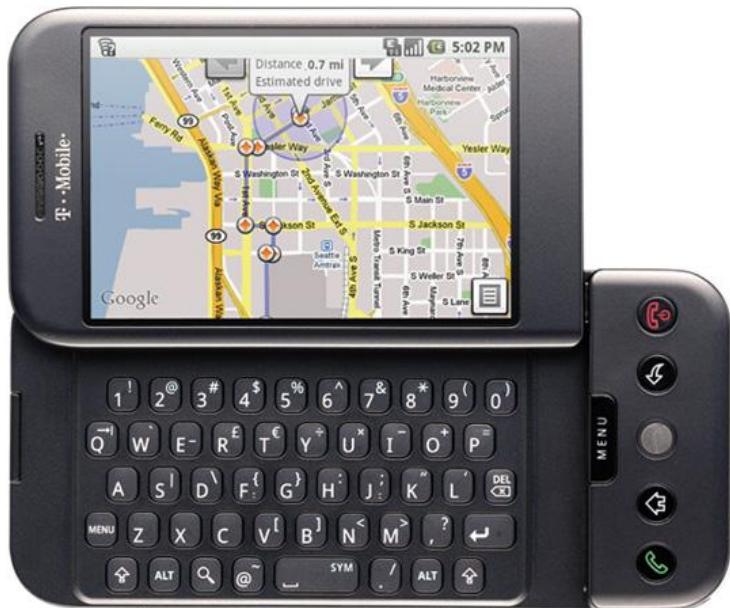
Android History

What is Android?

- Linux-based open source mobile operating system
 - Released under the Apache License
 - Source available at source.android.com

Early History

- Originally developed by a startup called Android, Inc in 2003
- Acquired by Google in 2005
- First commercially available phone released on Oct 22, 2008 (T-Mobile G1)



Android Ecosystem

Google



htc
quietly brilliant

SAMSUNG

SONY

amazon

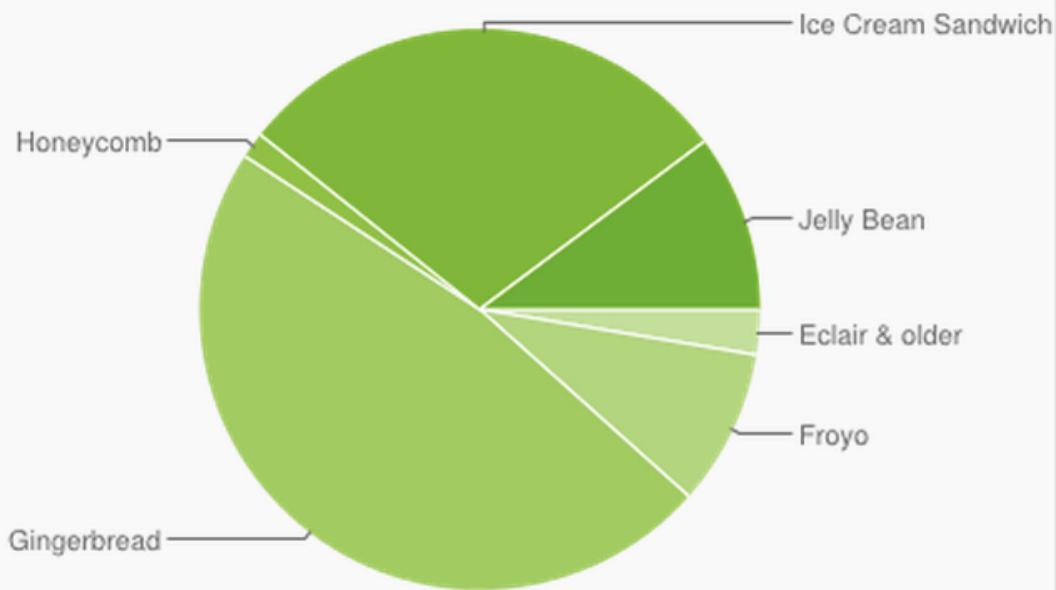


Android API History

Android 1	API 1	Oct 2008	
Android 1.1	API 2	Feb 2009	Released for the G1 only, bug fixes
Android 1.5 (Cupcake)	API 3	Apr 2009	UI tweaks, on screen keyboard, widgets
Android 1.6 (Donut)	API 4	Sept 2009	multiple screen resolutions, market place
Android 2.0-2.1 (Eclair)	API 7	Oct 2009	multiple account support, live wallpapers
Android 2.2 (FroYo)	API 8	May 2010	enterprise support, redesigned home screen
Android 2.3/2.3.3 (Gingerbread)	API 9/10	Dec 2010	reskin, lower level access to hardware for gaming, front facing camera
Android 3.0-3.2 (Honeycomb)	API 11/12/13	Feb 2011	Tablet-only
Android 4.0 (Ice Cream Sandwich)	API 14	Oct 2011	UI redesign, menu button removed
Android 4.2 (Jelly Bean)	API 17	June 2012	touch and visual performance improvements, google now

Android API Fragmentation

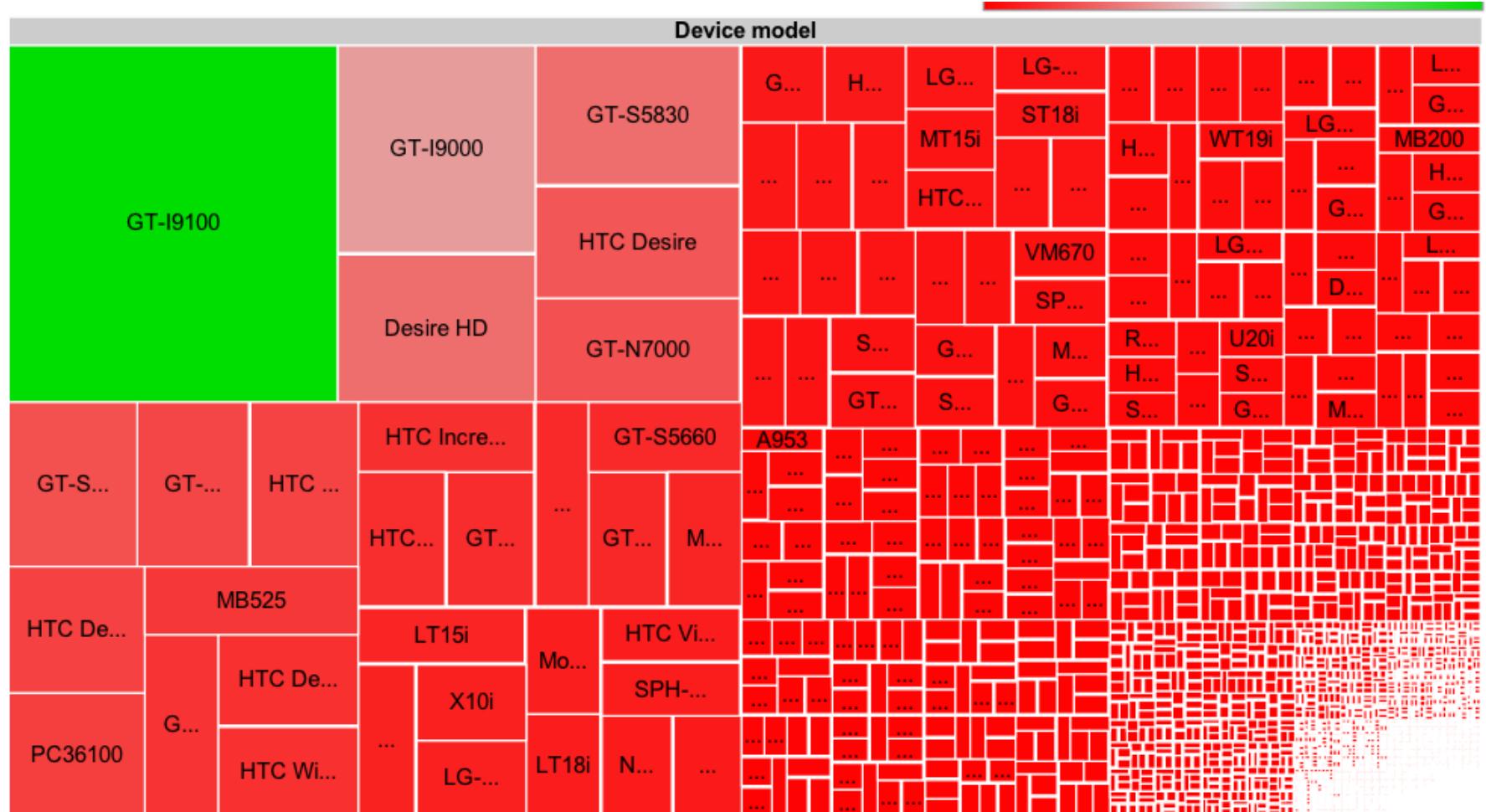
Version	Codename	API	Distribution
1.6	Donut	4	0.2%
2.1	Eclair	7	2.4%
2.2	Froyo	8	9.0%
2.3 - 2.3.2	Gingerbread	9	0.2%
2.3.3 - 2.3.7		10	47.4%
3.1	Honeycomb	12	0.4%
3.2		13	1.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	29.1%
4.1	Jelly Bean	16	9.0%
4.2		17	1.2%



from <http://developer.android.com/about/dashboards/index.html>

Data collected during a 14-day period ending on January 3, 2013

Android Device Fragmentation



App Distribution

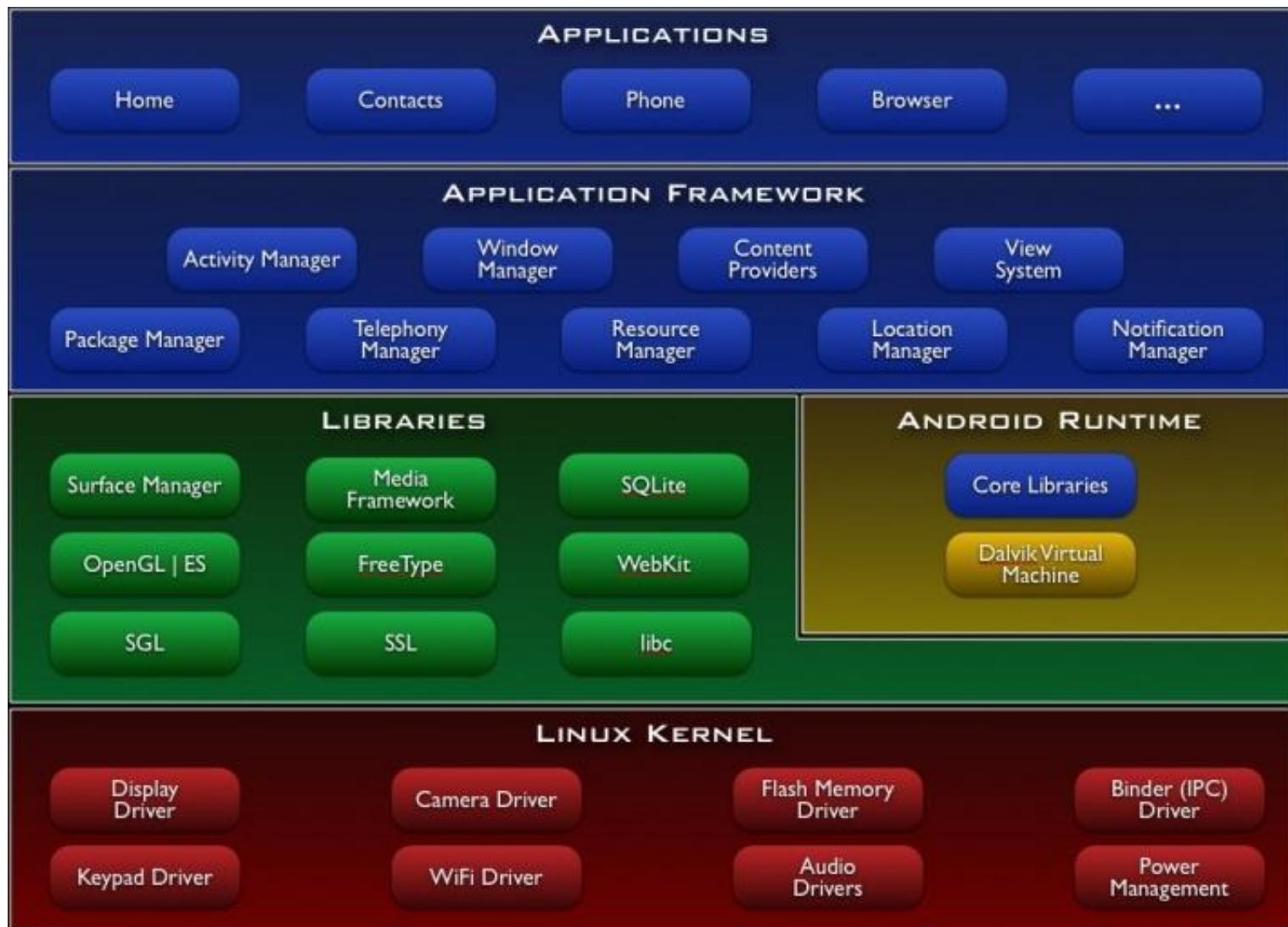


Google play



Google Play	Amazon AppStore
\$25 fee, one time	\$99 yearly fee (currently waived)
Google takes 30% of all sales	Amazon takes 30% of all sales
No certification, apps released immediately (1-4 hr distribution time)	Certification takes about a week
developer.android.com/distribute/index.html	developer.amazon.com/welcome.html

Android Architecture



Hello Android!

Building your first Android Project

Setting up your phone for dev

Enable USB debugging

Android 3.2 and below	Settings > Application > Development
Android 4.0	Settings > Developer options
Android 4.2	It may be hidden. If so, first go to: Settings > About phone and tap "Build number" seven times

You may also want:

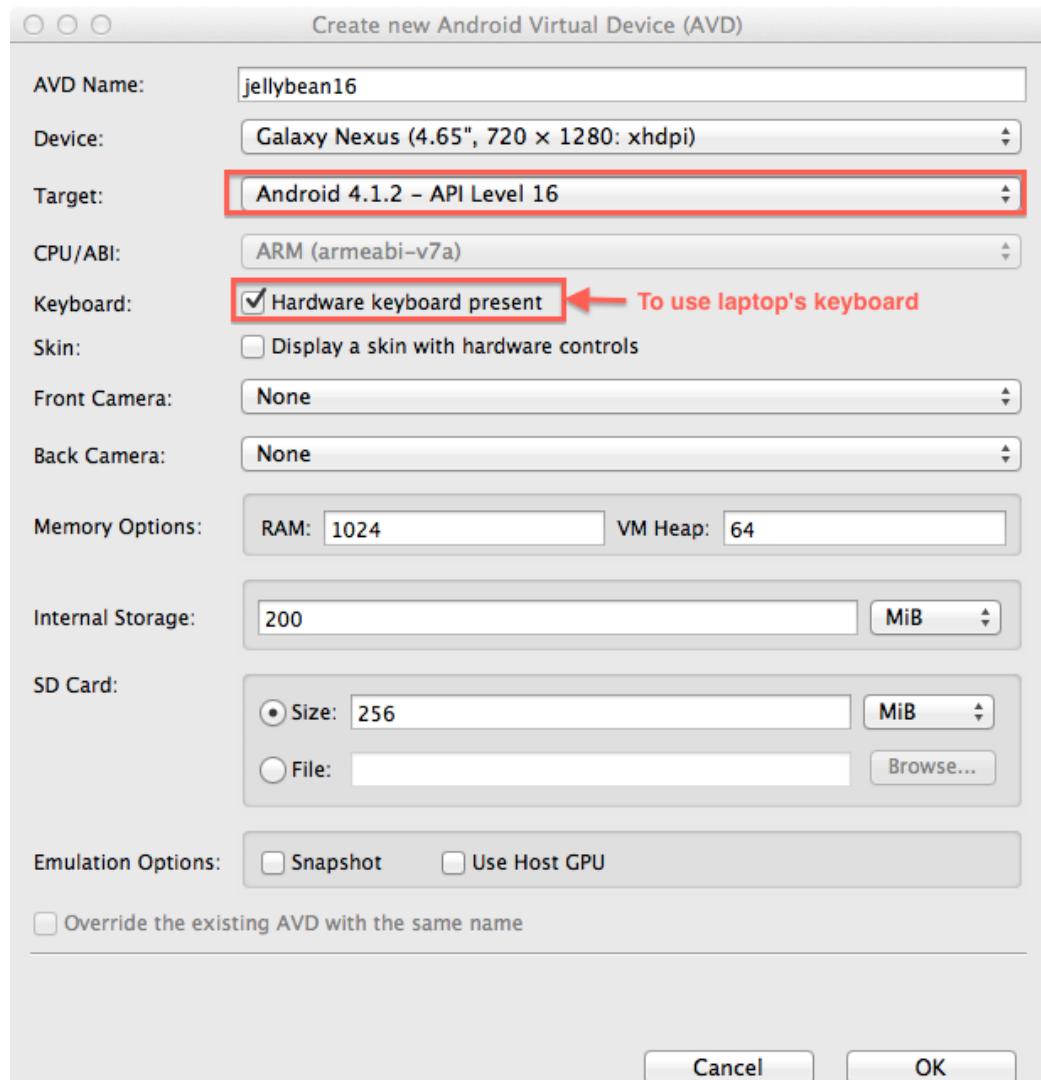
Allow installation from Unknown Sources

Setting up the emulator

> android avd

Click "New"

> emulator -avd
[avd name]

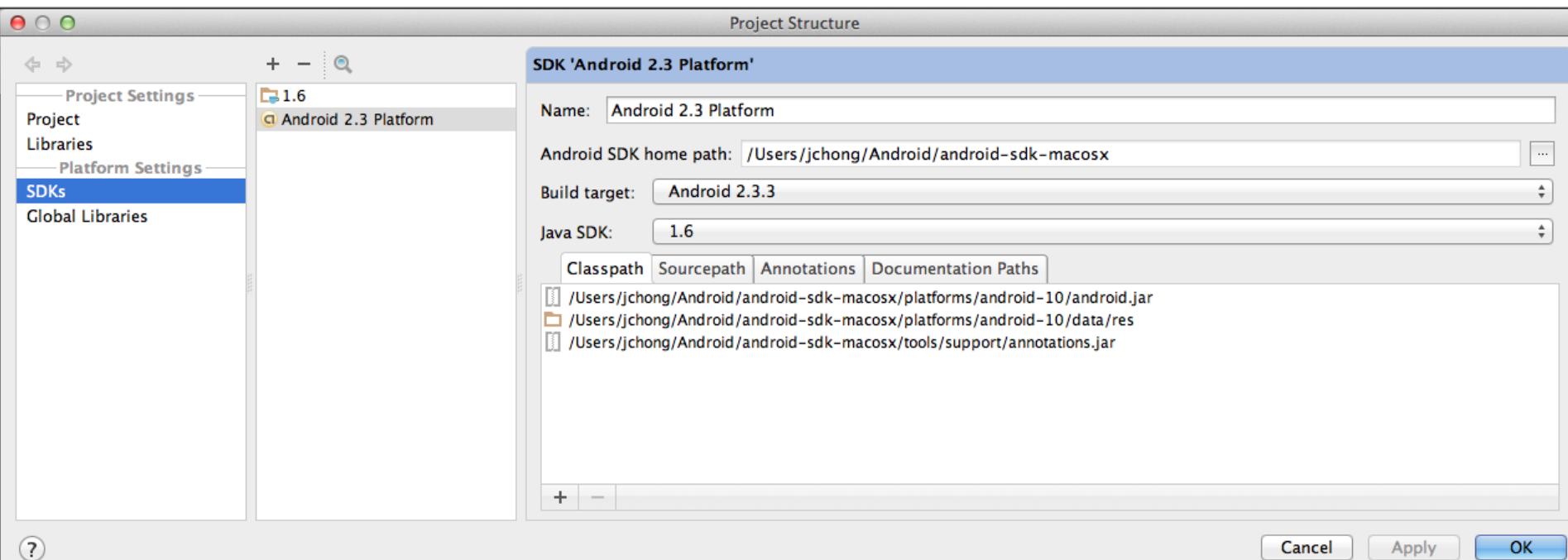


Adding the JDK and SDK paths

intellij: Configure > Project Defaults > Project Structure
JDK 1.6 path is (OS 10.7):

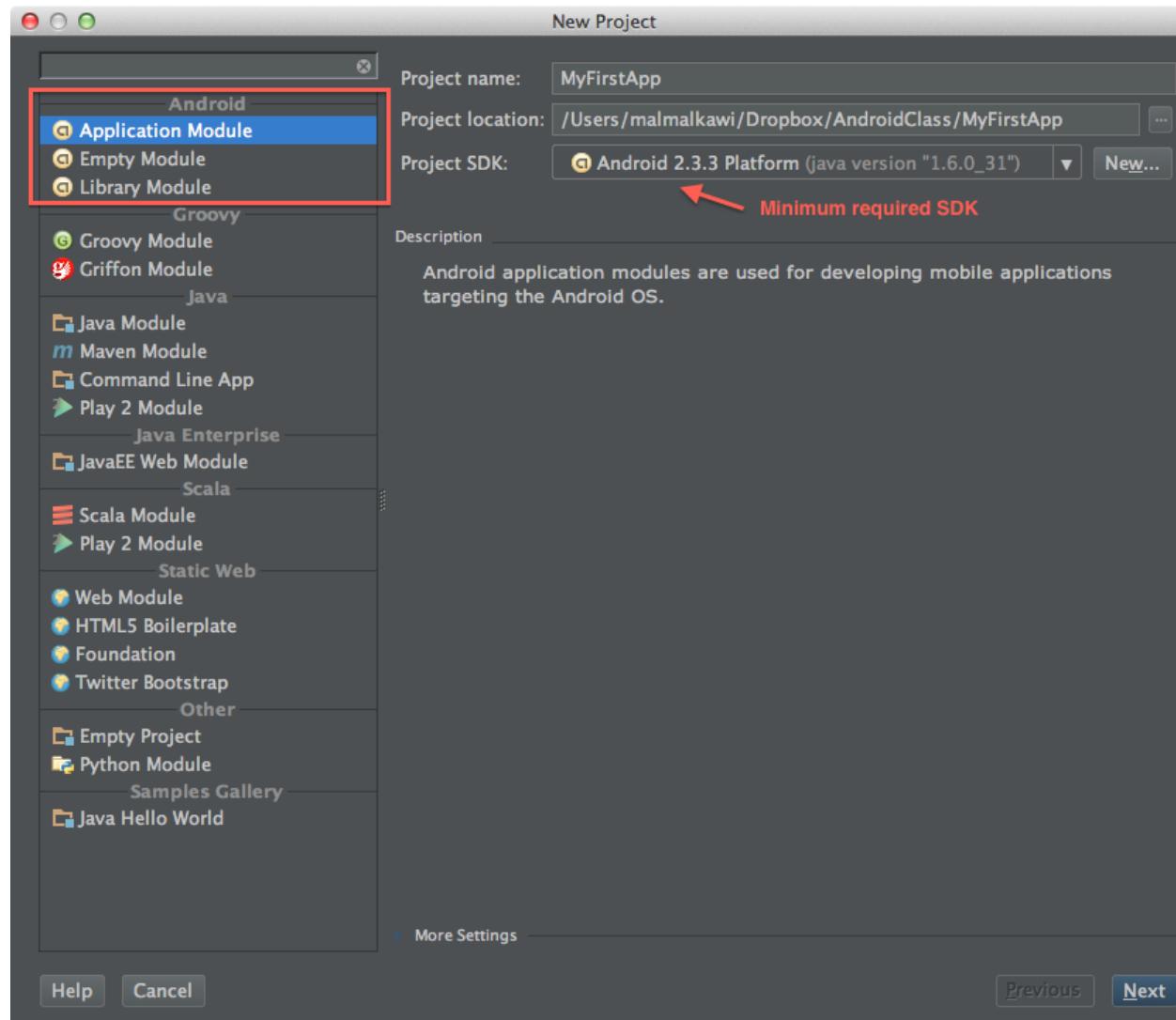
```
/System/Library/Frameworks/JavaVM.framework/Versions/1.6/Home
```

SDK: add the lowest API you want to support

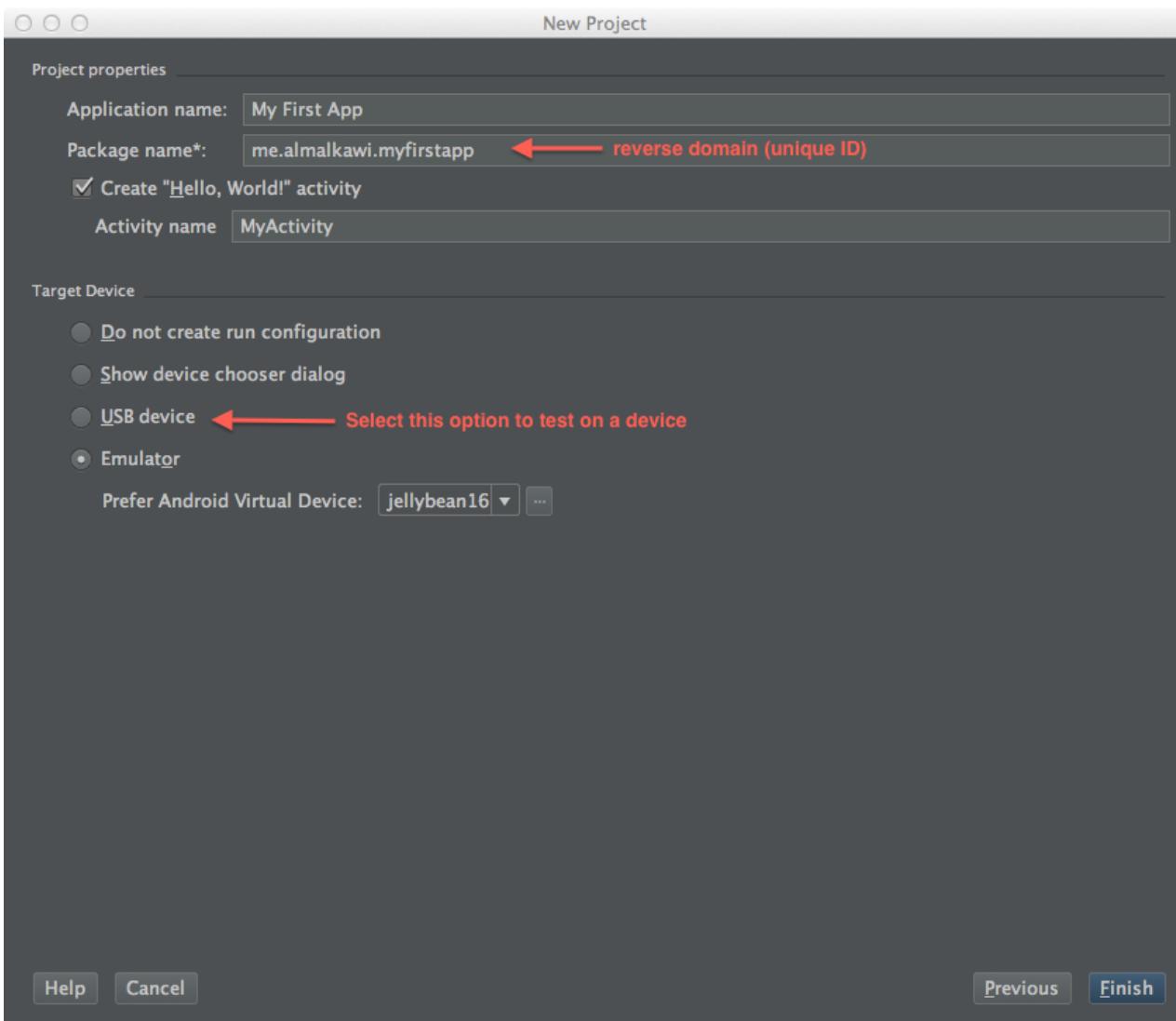


Create hello world project

File>New Project



New project properties



Default project structure

src/

- Implementation code (Java)

res/

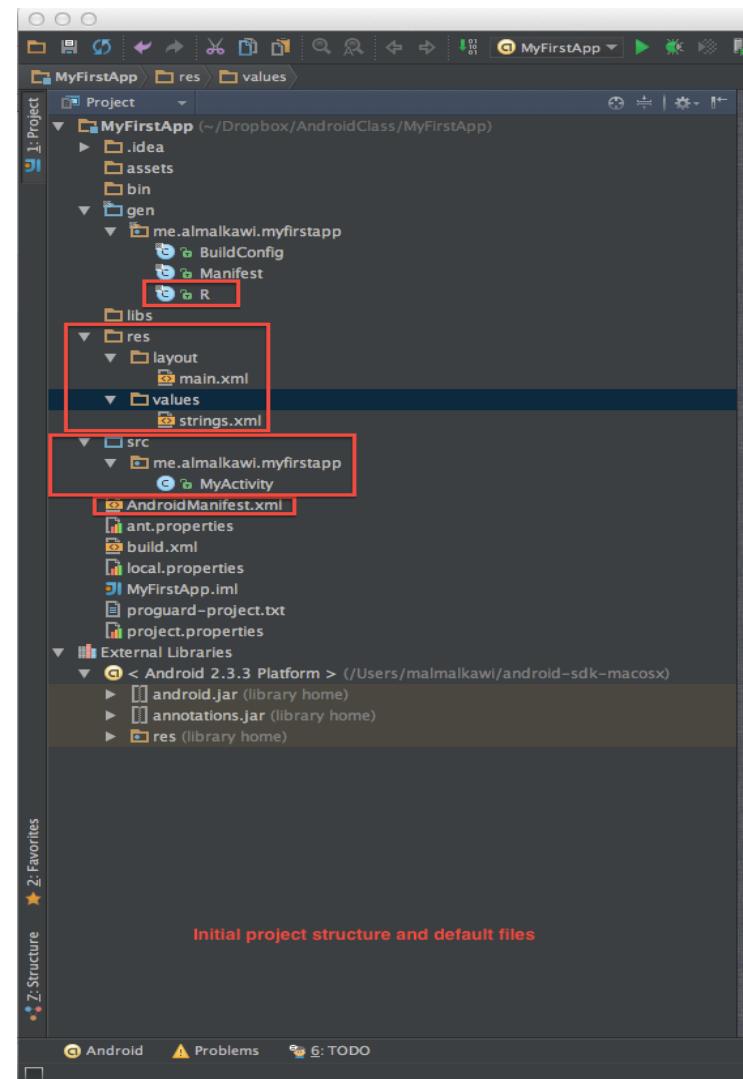
- Images
- XML layouts
- Strings
- Styles
- Constants

AndroidManifest.xml

- Information about the app

R.java file

- Autogenerated by build
- Contains resource IDs of res/ contents



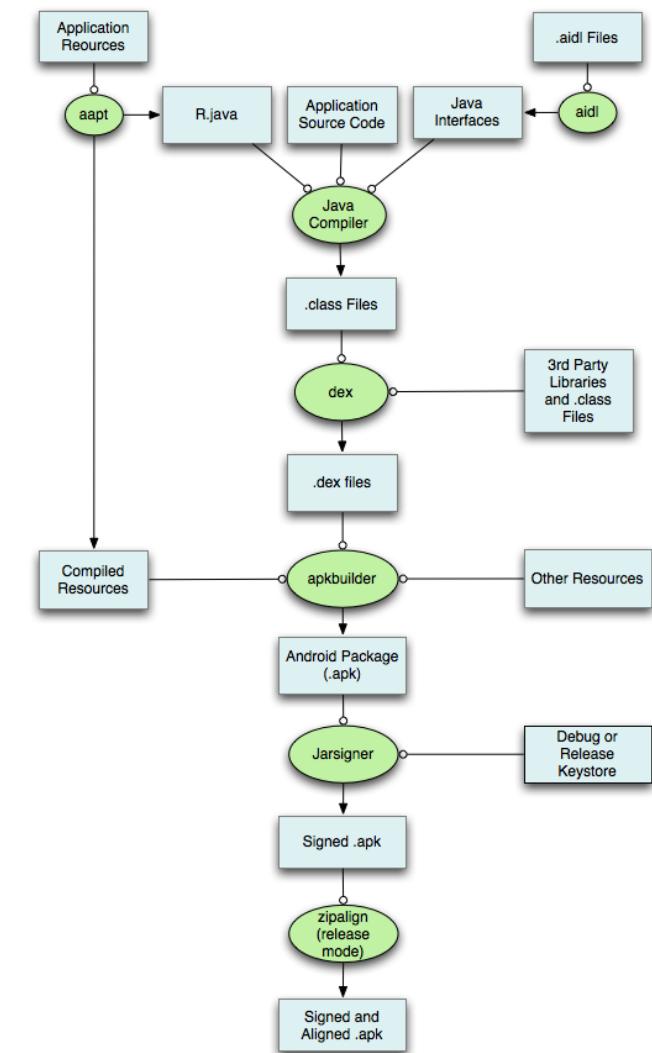
Building and Running Android app

IntelliJ: Shift + F10

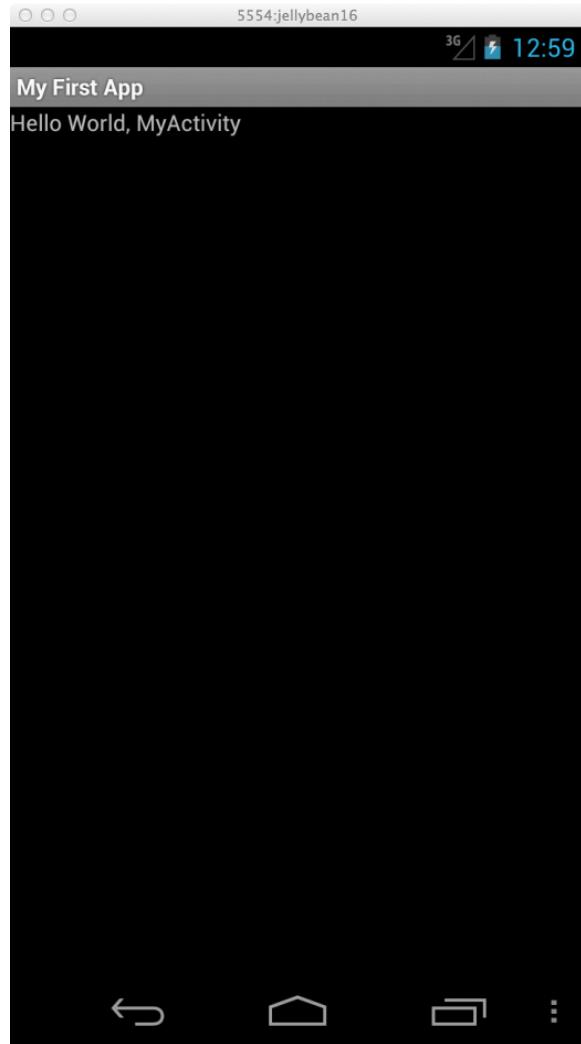
Eclipse: Shift + Command + F11

> ant debug install

Source:
<http://developer.android.com/images/build.png>



Hello World app running in emulator



APK (Application Package)

- Location:

IntelliJ: out/production/MyFirstApp/MyFirstApp.apk

Ant: bin/MyFirstApp-debug.apk

- APK is a zip archive of compiled app

```
> cp MyFirstApp.apk MyFirstApp.zip
```

```
> unzip -l MyFirstApp.zip
```

Length	Date	Time	Name
552	01-01-13	12:39	res/layout/main.xml
1444	01-01-13	12:39	AndroidManifest.xml
704	01-01-13	12:39	resources.arsc
2936	01-01-13	12:39	classes.dex
327	01-01-13	12:39	META-INF/MANIFEST.MF
380	01-01-13	12:39	META-INF/CERT.SF
776	01-01-13	12:39	META-INF/CERT.RSA

ADB (Android Debug Bridge)

```
> adb devices          # List connected devices  
  
> adb shell pm list packages    # List installed packages  
  
> adb shell pm list features    # List supported features  
  
> adb install <apk path>      # Install APK  
  
> adb uninstall <package-name> # Uninstall App  
  
> adb logcat            # View logs
```

Documentation

API Reference: <http://developer.android.com/reference/packages.html>

* Exercise:

- Add logging to Hello World app:

Add a debug log message in onCreate that outputs something when onCreate is called

HINT: Answer should only require you to add two lines to the code

* Steps:

- Search for "log" in API reference

- First hit is for class android.util.Log (method: Log.d).

- Add to MyActivity.java's onCreate method

Log.d("hackweek", "onCreate is called");

> adb logcat | grep hackweek

- Run app

Crashes

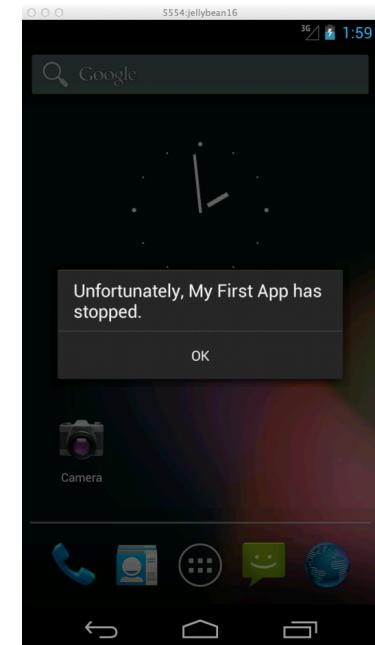
- Add to onCreate:

```
final ArrayList<String> list = null;  
list.add("Android"); // boom! NPE.
```

- Run (Shift+F10)

- Get crash call stack

```
> adb logcat *:E
```



```
E/AndroidRuntime( 635): FATAL EXCEPTION: main  
E/AndroidRuntime( 635): java.lang.RuntimeException: Unable to start activity ComponentInfo{me.almalkawi.myfirstapp/me.almalkawi.myfirstapp.MyActivity}: java.lang.NullPointerException  
E/AndroidRuntime( 635):     at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2059)  
E/AndroidRuntime( 635):     at android.app.ActivityThread.handleLaunchActivity(ActivityThread.java:2084)  
E/AndroidRuntime( 635):     at android.app.ActivityThread.access$600(ActivityThread.java:130)  
E/AndroidRuntime( 635):     at android.app.ActivityThread$H.handleMessage(ActivityThread.java:1195)  
E/AndroidRuntime( 635):     at android.os.Handler.dispatchMessage(Handler.java:99)  
E/AndroidRuntime( 635):     at android.os.Looper.loop(Looper.java:137)  
E/AndroidRuntime( 635):     at android.app.ActivityThread.main(ActivityThread.java:4745)  
E/AndroidRuntime( 635):     at java.lang.reflect.Method.invokeNative(Native Method)  
E/AndroidRuntime( 635):     at java.lang.reflect.Method.invoke(Method.java:511)  
E/AndroidRuntime( 635):     at com.android.internal.os.ZygoteInit$MethodAndArgsCaller.run(ZygoteInit.java:786)  
E/AndroidRuntime( 635):     at com.android.internal.os.ZygoteInit.main(ZygoteInit.java:553)  
E/AndroidRuntime( 635):     at dalvik.system.NativeStart.main(Native Method)  
E/AndroidRuntime( 635): Caused by: java.lang.NullPointerException  
E/AndroidRuntime( 635):     at me.almalkawi.myfirstapp.MyActivity.onCreate(MyActivity.java:18)  
E/AndroidRuntime( 635):     at android.app.Activity.performCreate(Activity.java:5008)  
E/AndroidRuntime( 635):     at android.app.Instrumentation.callActivityOnCreate(Instrumentation.java:1079)  
E/AndroidRuntime( 635):     at android.app.ActivityThread.performLaunchActivity(ActivityThread.java:2023)  
E/AndroidRuntime( 635):     ... 11 more
```

Android Monitor

IntelliJ: Tools->Android->Monitor (or > monitor)

The screenshot shows the Android Debug Monitor window in IntelliJ IDEA. The window has a toolbar at the top with various icons, followed by a menu bar with 'File', 'Edit', 'Tools', 'Android', 'Run', 'View', 'Help'. Below the menu is a tab bar with 'Threads', 'Heap', 'Allocation Tracker', 'Network Statistics', 'File Explorer' (which is selected and highlighted in blue), and 'Emulator Control'. The main area is divided into two panes. The left pane is a 'File Explorer' tree view showing the file structure of the emulator ('jellybean16 [emulator-5554]') and the host system. The right pane is a 'File Explorer' table view showing files and their details like name, size, date, time, permissions, and info. A red arrow points to the 'File Explorer' tab in the toolbar. Another red arrow points to the 'File Explorer' table view. A third red arrow points to the process 'me.alimalkawi.myfirstapp' in the table view, which is highlighted with a blue border. A fourth red arrow points to the 'Emulator Control' tab in the toolbar. A fifth red arrow points to the status bar at the bottom right which says '9M of 38M'.

Take a screen capture

Android Debug Monitor

Devices

Threads Heap Allocation Tracker Network Statistics File Explorer Emulator Control

Name

jellybean16 [emulator-5554] Online jellybean16...

com.android.email 509 8600

com.android.quicksearchbox 608 8601

com.android.calendar 339 8602

com.android.defcontainer 412 8603

com.android.exchange 527 8604

com.svox.picco 587 8605

com.android.systemui 227 8606

system_process 166 8607

com.android.contacts 438 8608

android.process.media 460 8609

com.android.phone 257 8610

com.android.providers.calendar 399 8611

com.android.settings 292 8612

android.process.acore 320 8613

com.android.deskclock 364 8614

com.android.mms 484 8615

com.android.launcher 270 8616

com.android.keychain 569 8617

com.android.inputmethod.latin 241 8618

me.alimalkawi.myfirstapp 1207 8619 / 8700

Our demo app's process

File Explorer

Emulator Control

Simulate phone call, SMS, location, ...etc

Name Size Date Time Permissions Info

acct 2013-01-01 13:54 drwxr-xr--x

cache 2013-01-01 00:44 drwxrwx---

config 2013-01-01 13:54 dr-x-----

d 2013-01-01 13:54 lrwxrwxrwx -> /sys/k...

data 2012-12-28 15:34 drwxrwxr-x

app 2013-01-01 14:13 drwxrwxr-x

app-asec 2012-12-28 15:33 drwx-----

app-private 2012-12-28 15:33 drwxrwxr-x

backup 2013-01-01 13:55 drwx-----

dalvik-cache 2013-01-01 14:13 drwxrwxr-x

data 2013-01-01 12:44 drwxrwxr-x

dontpanic 2012-12-28 15:33 drwxr-x---

drm 2012-12-28 15:33 drwxrwxr-x

local 2012-12-28 15:33 drwxr-xr-x

lost+found 2012-12-28 15:33 drwxrwxr---

misc 2012-12-28 15:33 drwxrwxr-t

nativebenchmark 2012-10-09 13:16 drwxrwxr-x

nativetest 2012-10-09 13:16 drwxrwxr-x

property 2013-01-01 13:56 drwx-----

resource-cache 2012-12-28 15:33 drwxrwxr-x

ssh 2012-12-28 15:33 drwxr-x---

system 2013-01-01 14:13 drwxrwxr-x

user 2012-12-28 15:33 drwx--x--x

default.prop 116 1969-12-31 16:00 -rw-f-r--r-

etc 2013-01-01 13:55 drwxr-xr-x

init 2013-01-01 13:54 lrwxrwxrwx -> /syste...

init.goldfish.rc 2344 1969-12-31 16:00 -rwxr-x---

init.rc 17057 1969-12-31 16:00 -rwxr-x---

init.trace.rc 1637 1969-12-31 16:00 -rwxr-x---

init.usb.rc 3915 1969-12-31 16:00 -rwxr-x---

mnt 2013-01-01 13:54 drwxrwxr-x

proc 1969-12-31 16:00 dr-xr-xr-x

root 2011-12-08 15:00 drwx-----

sbin 1969-12-31 16:00 drwxr-x---

sdcard 2013-01-01 13:54 lrwxrwxrwx -> /mnt/...

sys 1969-12-31 16:00 drwxr-xr-x

system 2012-10-09 13:20 drwxr-xr-x

ueventd.goldfish.rc 272 1969-12-31 16:00 -rw-f-r--r-

LogCat

Console

DDMS

9M of 38M

Hello Android!

Building basic UI

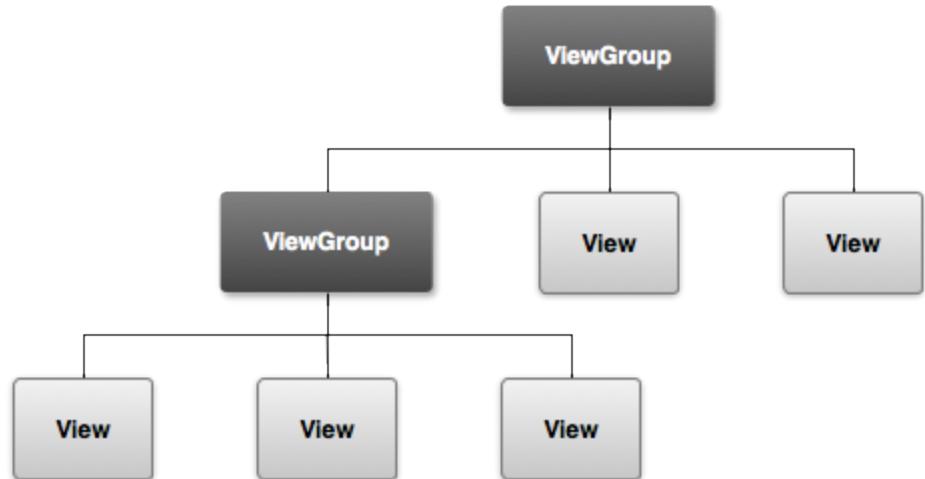
Activities

- Roughly analogous to Controller in MVC model
- On a basic level, you can think of an Android app as interacting with a series of Activities
- Only one activity has focus on an Android device at any one time
- Use *intents* to move between activities (more on that later)
- Usually, but not always, associated with a View (via an xml layout file)

UI: Hierarchy of Views

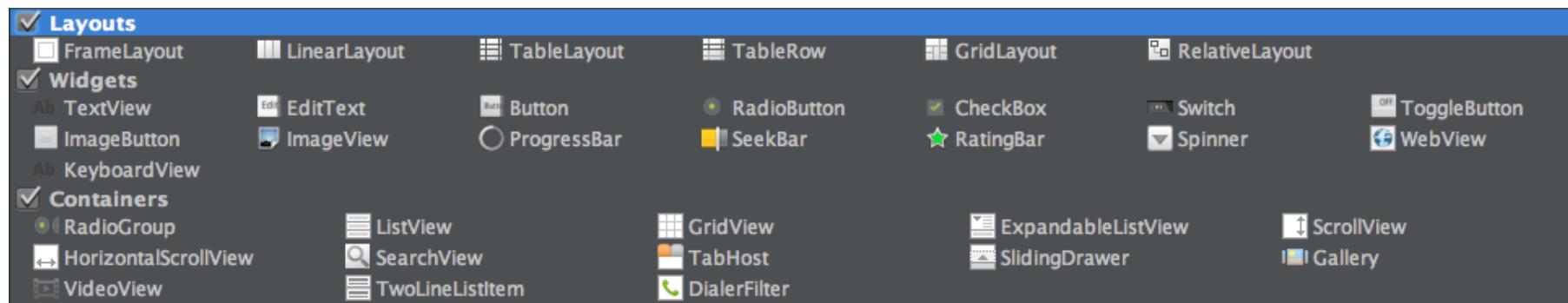
- ViewGroups:

Layouts and Containers
(e.g. LinearLayout)



- Views:

Widgets (e.g. Button)



Create New Project (SimpleUI)

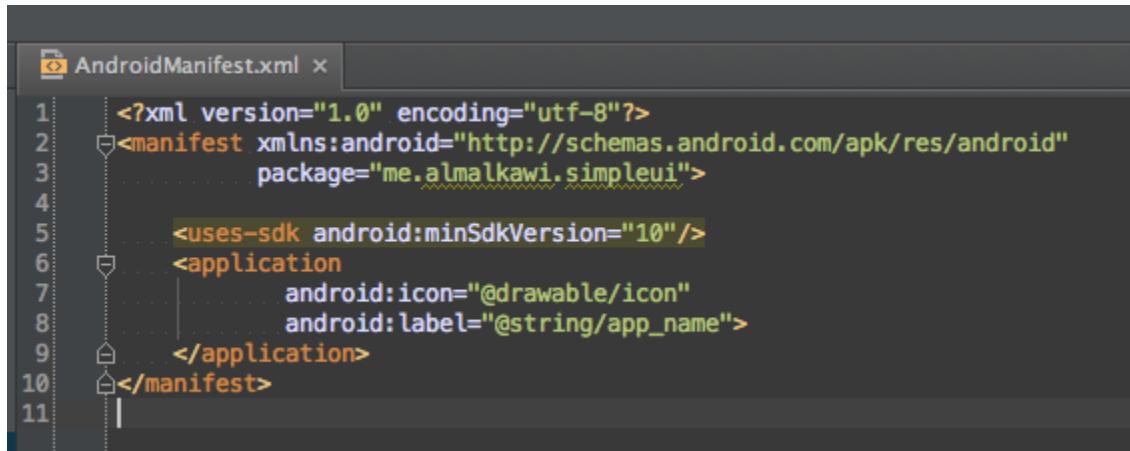
- Create new Android project:

Name: SimpleUI

Package: com.example.simpleui

Uncheck: "Create Hello World activity"

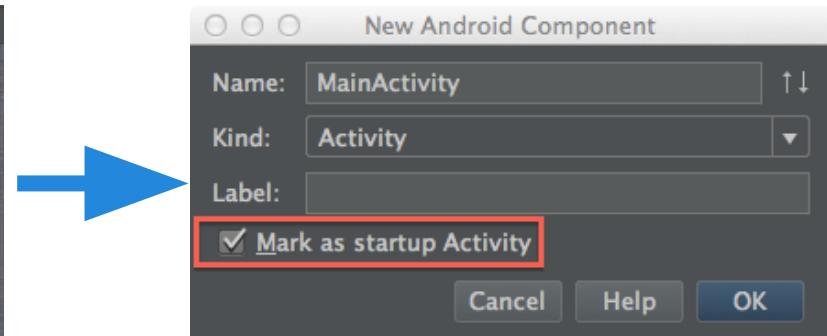
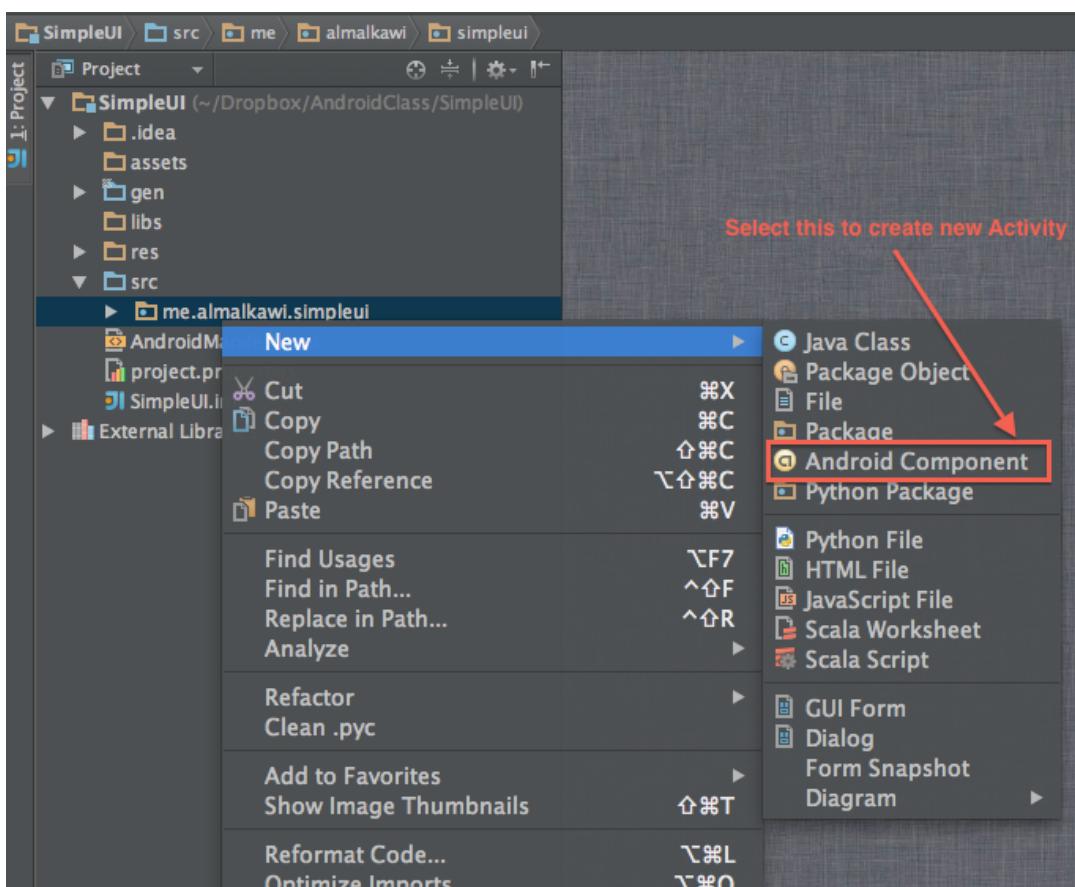
- Open the AndroidManifest.xml file:



A screenshot of an IDE showing the AndroidManifest.xml file. The code is displayed in a monospaced font with syntax highlighting. The XML structure includes a manifest tag with attributes for version and package, a uses-sdk tag for minSdkVersion, and an application tag with icon and label attributes.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="me.almalkawi.simpleui">
    <uses-sdk android:minSdkVersion="10"/>
    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
    </application>
</manifest>
```

Create startup Activity



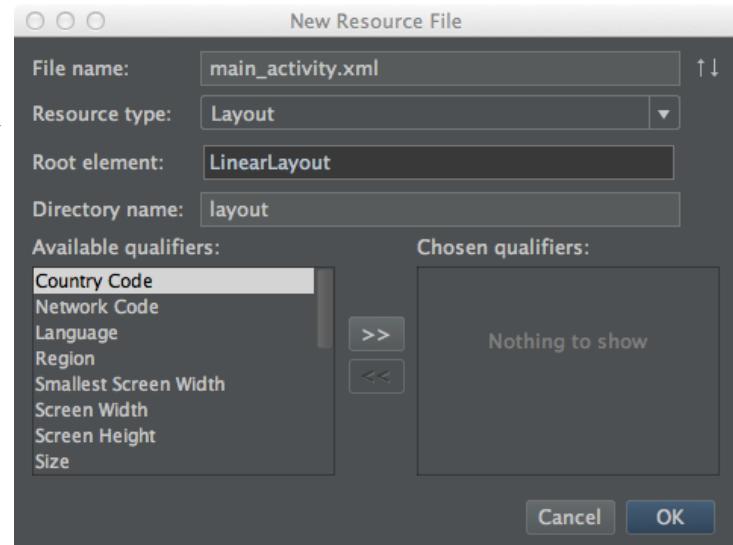
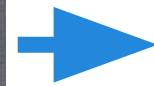
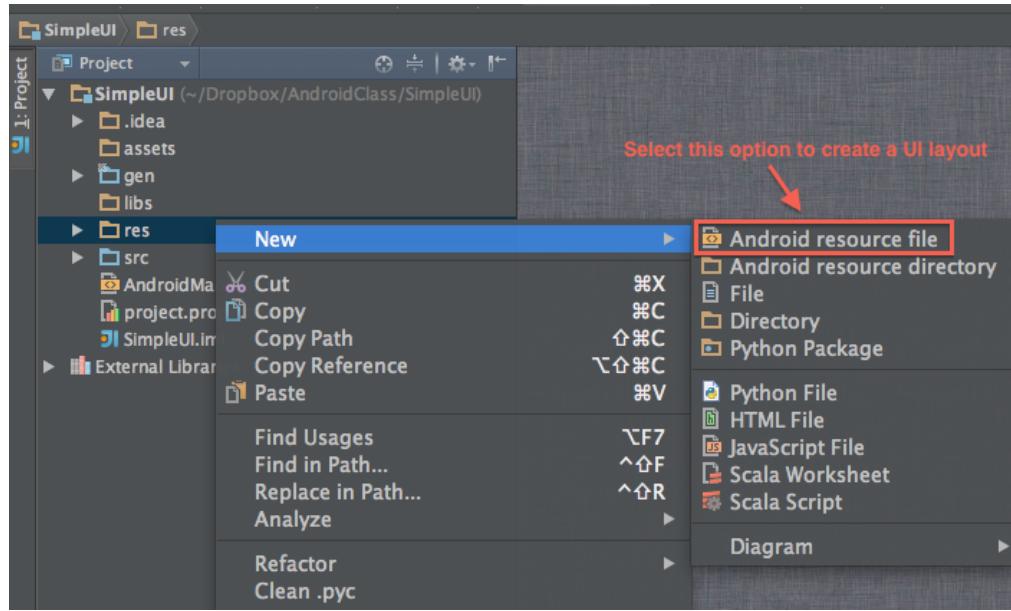
Activity in AndroidManifest.xml

- Every activity is declared in manifest
- Declaration added automatically by IDE

This is what makes it a startup activity.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="me.almalkawi.simpleui">
    <uses-sdk android:minSdkVersion="10"/>
    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
    </application>
</manifest>
```

Create UI Layout



UI Designer

The screenshot illustrates the Android UI Designer interface, specifically the Device Screen tab.

- Component Tree:** Shows the hierarchy of the layout. The root node is a **LinearLayout (vertical)**, highlighted with a red box and arrow.
- Properties Panel:** Displays the properties for the selected **LinearLayout**. Key properties highlighted with red boxes and arrows are:
 - layout_width: fill_parent**
 - layout_height: fill_parent**
 - style: orientation: vertical**

A note below the properties states: "These two properties are required for all views to specify their size.
- fill_parent is deprecated, change to match_parent.
- for views that need to occupy space enough for their content, use wrap_content."

Additional notes:
 - "LinearLayout can be either vertical or horizontal" is shown near the style property.
 - "Here you can toggle between design mode" is shown near the bottom left of the design area.
 - "Root of the layout (ViewGroup)" is labeled in the Component Tree panel.
 - "WYSIWYG" is labeled next to the preview window.
- Design View:** The main preview window showing a black rectangular screen labeled "SimpleUI".
- Toolbox:** A large palette on the right side containing categories like Layouts, Widgets, Containers, Date & Time, Expert, and Custom. Common widgets like **LinearLayout**, **EditText**, **Button**, and **Image** are highlighted with red boxes and arrows.
- Bottom Buttons:** Buttons for "Design" and "Text" modes, with "Design" currently selected.

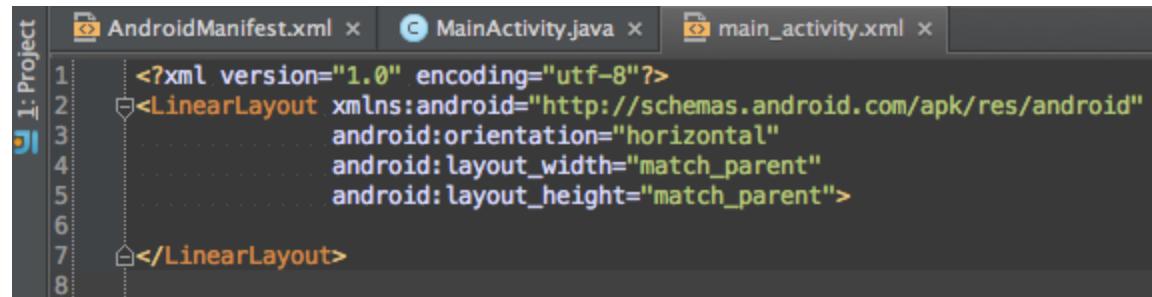
Basic View properties

The `layout_height` and `layout_width` can have the following values:

- `FILL_PARENT` - deprecated in API 8
- `MATCH_PARENT` - match the height or width of my View
the height or width of the parent View
- `WRAP_CONTENT` - make my View as large as it needs to
be to display the content inside
- `<numerical value>` - "30dp", can use px (pixels), dp
(density-independent pixels), sp (scaled pixels), in
(inches) or mm (millimeters)

UI Design (text mode)

- XML layout file (`main_activity.xml`) - The Real Thing!



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
</LinearLayout>
```

- Used to tweak things

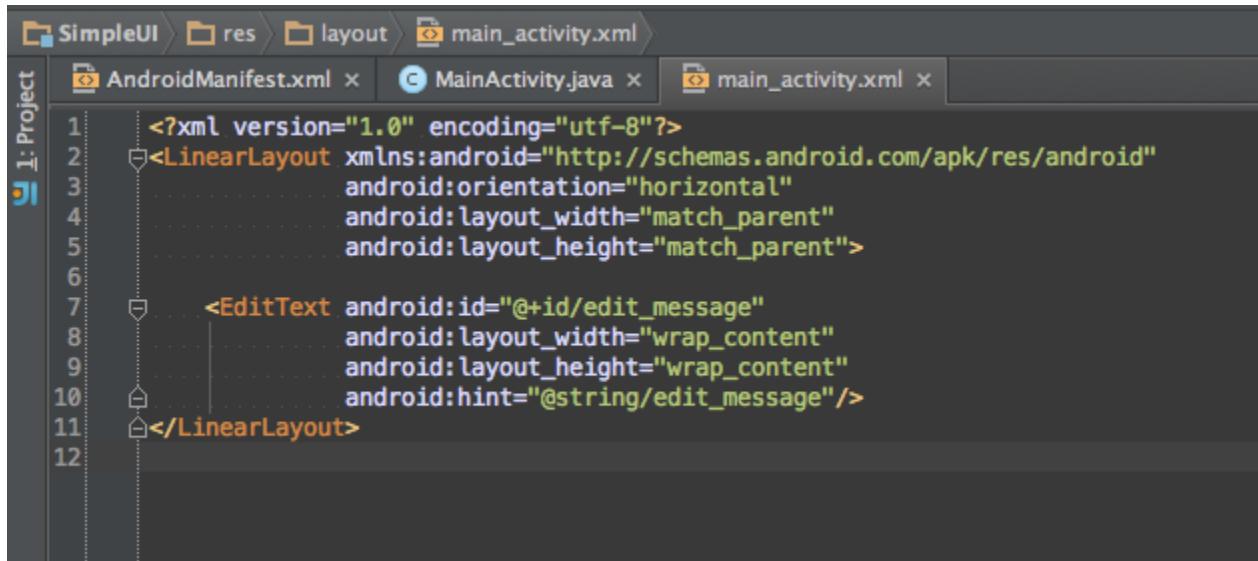
Add EditText View

The screenshot shows the Android Studio interface with the following components and annotations:

- Component Tree:** Shows the hierarchy: Device Screen > LinearLayout (horizontal) > edit_message: EditText.
- Properties Panel:** Shows the following properties:
 - layout_width: wrap_content
 - layout_height: wrap_content
 - hint: @string/edit_message (highlighted with a red box)
 - id: @+id/edit_message (highlighted with a red box)Annotations:
 - A red arrow points from the "edit_message" entry in the Component Tree to the "hint" field in the Properties panel.
 - A red arrow points from the "wrap_content" value in the "layout_width" and "layout_height" fields to the "Makes the view as big as needed" note.
 - A red arrow points from the "edit_message" entry in the Component Tree to the "id" field in the Properties panel.
 - A red arrow points from the "edit_message" entry in the Component Tree to the "String key" in the String Resource dialog.
 - A red arrow points from the "edit_message" entry in the Component Tree to the "String value" in the String Resource dialog.
 - A red arrow points from the "edit_message" entry in the Component Tree to the "File name" field in the New String Value Resource dialog.
 - A red arrow points from the "edit_message" entry in the Component Tree to the "Unique ID of the view, + means define" note.
- Preview:** Shows a black screen with a white text input field containing "Enter a message". A red arrow points from the "Preview of EditText" text to the input field.
- AndroidManifest.xml:** Shows the manifest file with tabs for Full, 2.7in QVGA, Portrait, Android 2.3.3, Any locale, Normal, Day time, and Theme.
- MainActivity.java:** Shows the Java code for the activity.
- main_activity.xml:** Shows the XML code for the layout.
- Palette:** Shows various UI components like Layouts, Widgets, Containers, etc.
- Resources Dialog:** Shows the "String" section with entries for "app_name" and "edit_message".
 - "edit_message" is highlighted with a red box.
 - A red arrow points from the "edit_message" entry to the "String key" in the New String Value Resource dialog.
 - A red arrow points from the "edit_message" entry to the "String value" in the New String Value Resource dialog.
- New String Value Resource Dialog:** Shows fields for Resource name, Resource value, File name (set to "strings.xml"), and Create the resource in directories (values).
 - "Resource name" and "Resource value" fields are empty.
 - "File name" is set to "strings.xml".
 - "Create the resource in directories" is checked.
 - "values" is selected under "English: res/values/strings.xml".
 - Buttons: Cancel, OK, and New Resource.

EditText in XML

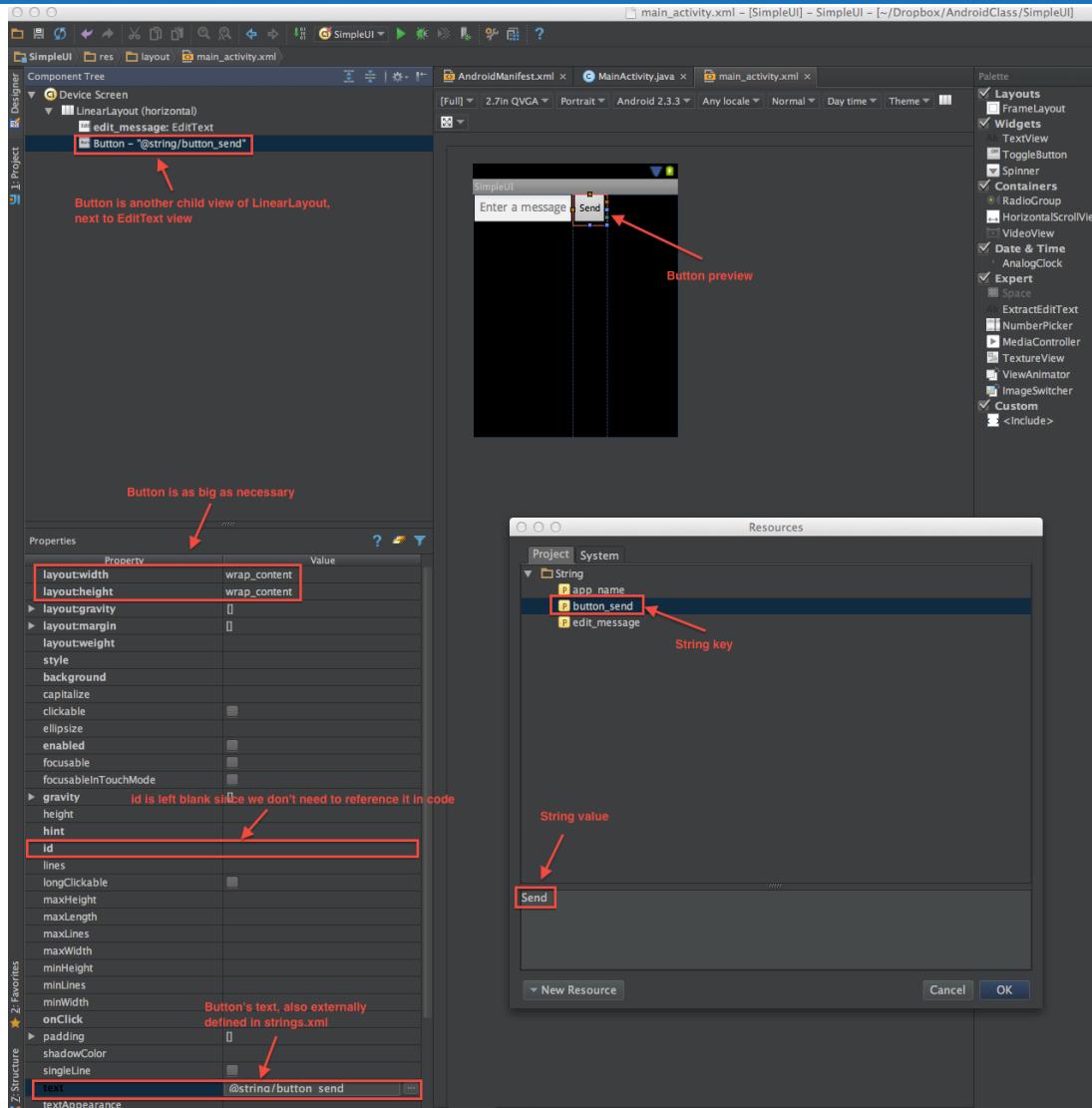
The updated XML after adding EditText:



The screenshot shows the Android Studio interface with the project navigation bar at the top. The main area displays the XML code for the `main_activity.xml` file. The code defines a horizontal LinearLayout containing an EditText field. The XML code is as follows:

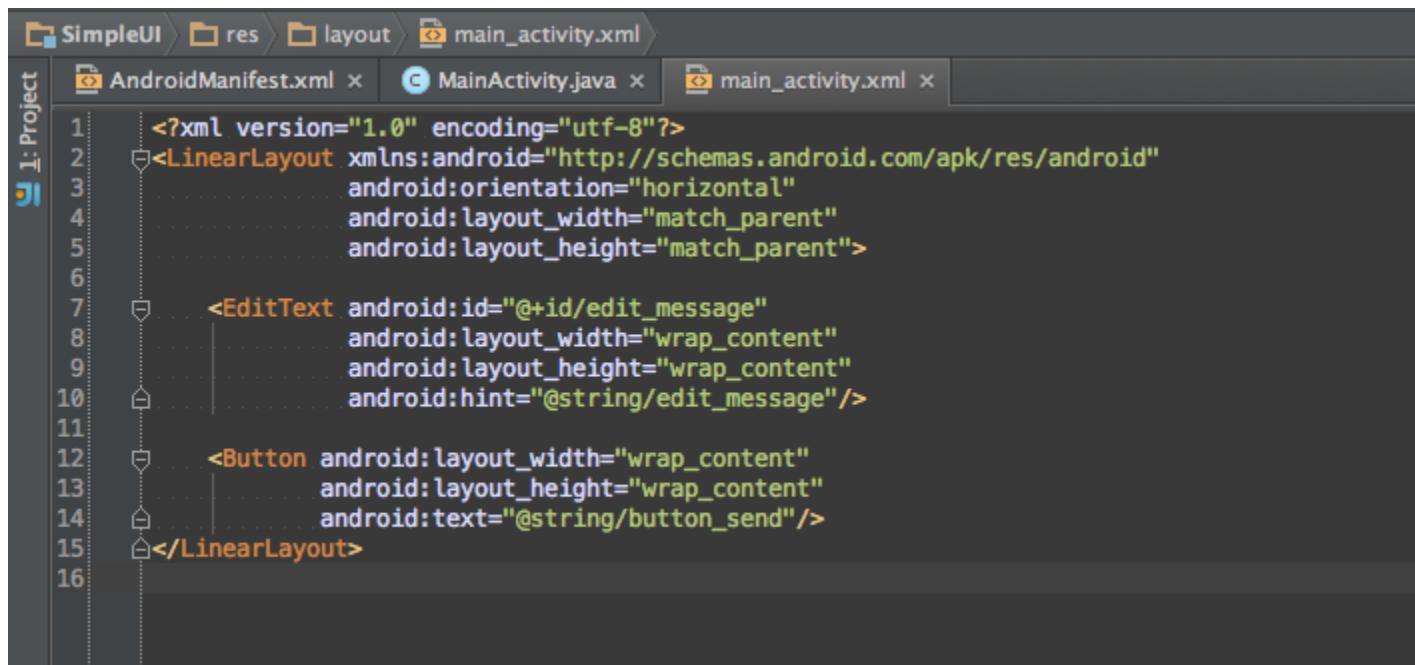
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <EditText android:id="@+id/edit_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message"/>
</LinearLayout>
```

Add Button View



Button in XML

The updated XML after adding Button:



A screenshot of the Android Studio interface showing the XML code for a layout. The code defines a horizontal LinearLayout containing an EditText and a Button. The Button has a placeholder text of "button_send".

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <EditText android:id="@+id/edit_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message"/>
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send"/>
</LinearLayout>
```

View weight

The screenshot shows the Android Studio Layout Editor interface. On the left, the Component Tree panel shows a hierarchy: Device Screen > LinearLayout (horizontal) > editText_message: EditText. The main area displays a preview of a screen titled "SimpleUI" with the text "Enter a message" in an EditText field and a "Send" button. A red arrow points from the text "Preview shows how EditText is using all unused screen width" to the right edge of the EditText's bounding box. In the bottom-left corner, the Properties panel is open, showing the following table:

Property	Value
layout:width	wrap_content
layout:height	wrap_content
layout:gravity	[]
layout:margin	[]
layout_weight	1
style	

A red box highlights the "layout_weight" row, and a red arrow points from the text "layout_weight of 1" to the value "1".

[Full] 2.7in QVGA Portrait Android 2.3.3 Any locale Normal Day

SimpleUI
Enter a message Send

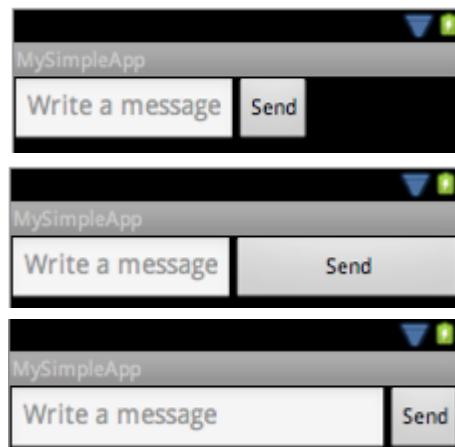
Preview shows how EditText is using all unused screen width

layout_weight of 1

layout_weight

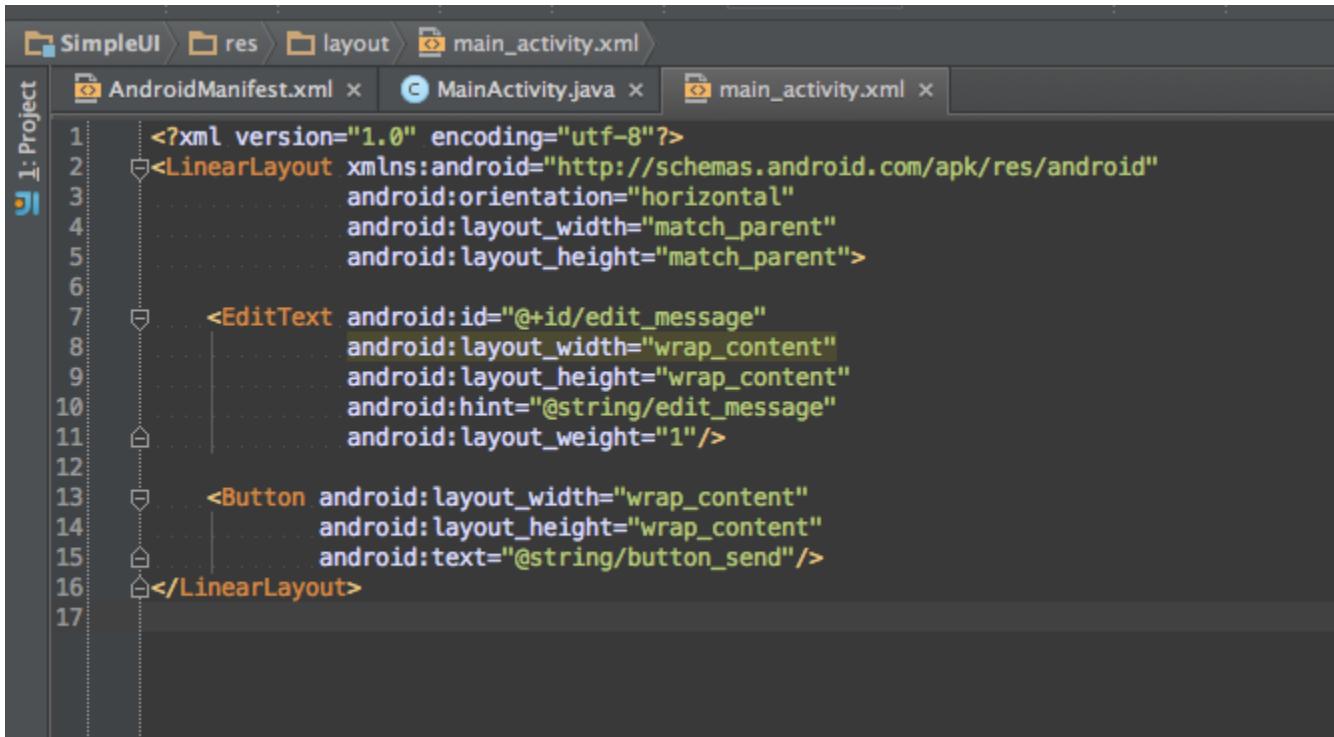
What does `layout_weight` do?

- Assigns the relative "importance" of a child view
- Determines which view gets any extra space available in a layout



EditText Weight	Button Weight
0	0
0	1
5	1

Completed UI (XML)

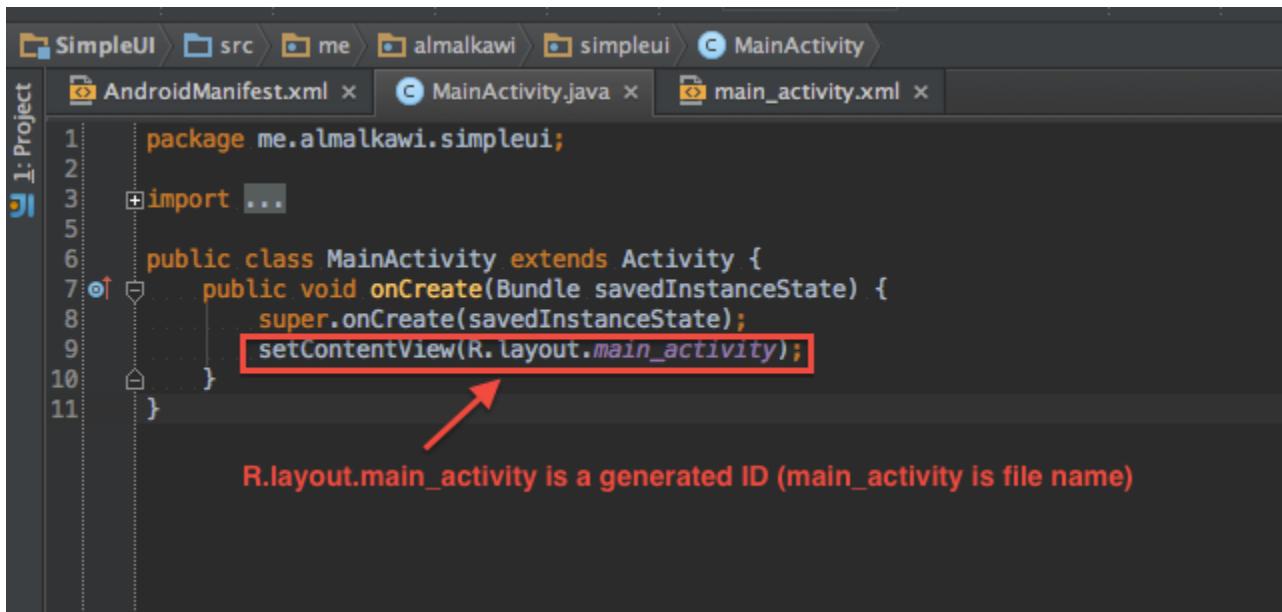


The screenshot shows the Android Studio interface with the project navigation bar at the top. The main area displays the XML code for the `main_activity.xml` layout file. The code defines a horizontal linear layout containing an edit text field and a button.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <EditText android:id="@+id/edit_message"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:hint="@string/edit_message"
        android:layout_weight="1"/>
    <Button android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/button_send"/>
</LinearLayout>
```

Loading UI resource in Activity

- Set layout in Activity (MainActivity.java, Command+N shortcut)

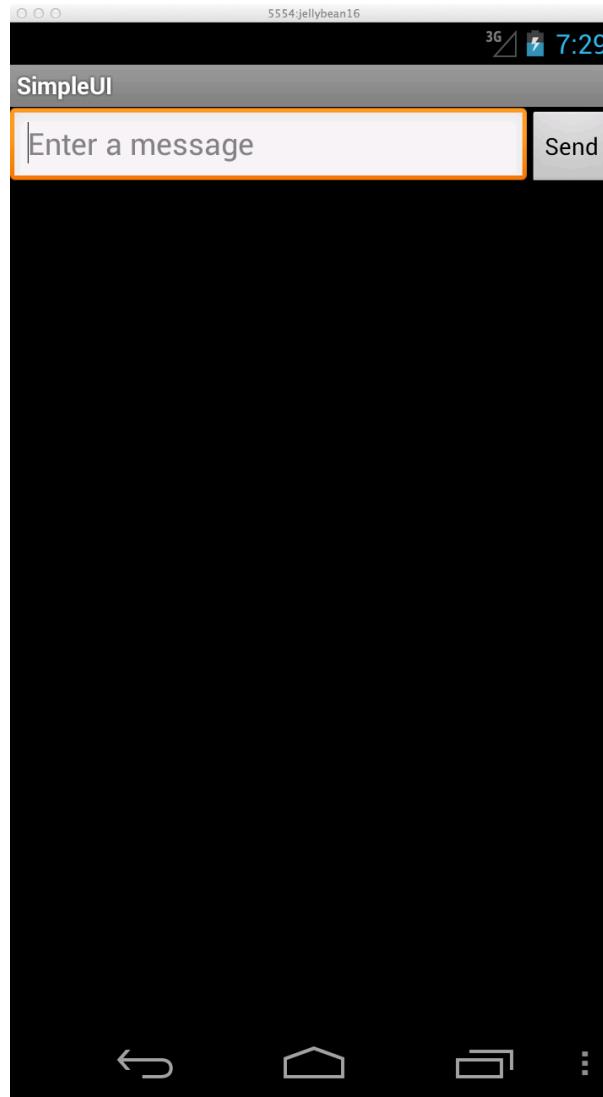


The screenshot shows the Android Studio interface with the project navigation bar at the top. Below it, the code editor displays MainActivity.java. The code is as follows:

```
1 package me.almalkawi.simpleui;
2
3 import ...
4
5
6 public class MainActivity extends Activity {
7     @Override
8     protected void onCreate(Bundle savedInstanceState) {
9         super.onCreate(savedInstanceState);
10        setContentView(R.layout.main_activity);
11    }
12}
```

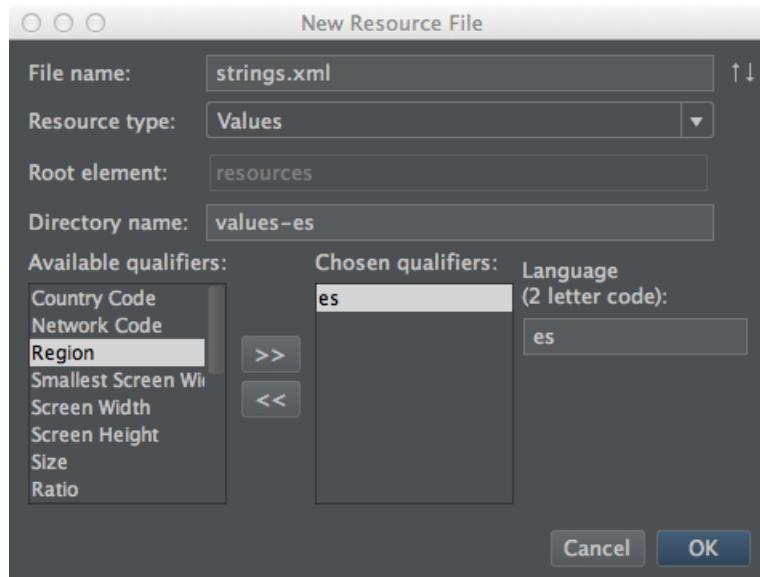
A red arrow points from the explanatory text below to the line of code `setContentView(R.layout.main_activity);`. The explanatory text reads: "R.layout.main_activity is a generated ID (main_activity is file name)".

App running in emulator



Supporting multiple languages

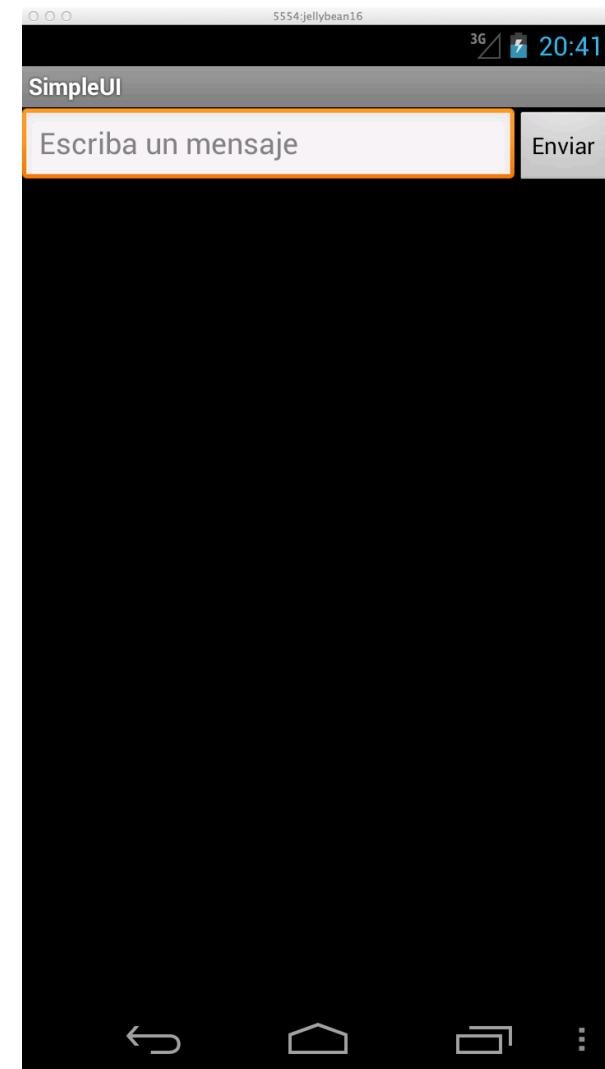
- Create new Android resource file for Spanish



```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">SimpleUI</string>
    <string name="edit_message">Escriba un mensaje</string>
    <string name="button_send">Enviar</string>
</resources>
```

Test Spanish localization

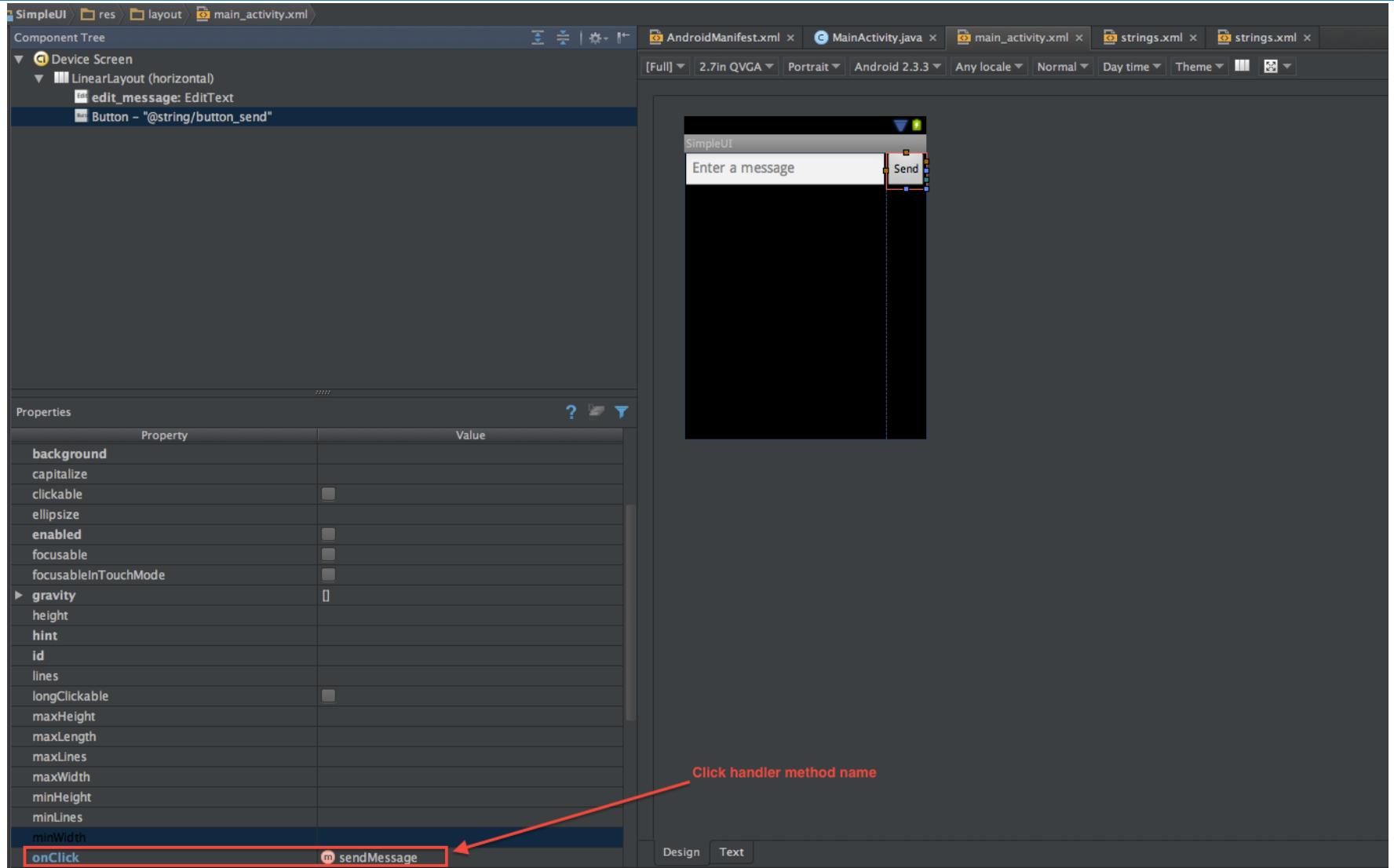
- Switch system language to Spanish
(Settings->Language & Input->Language)
- Build and install app (Shift+F10)
- To switch back to English:
(Configuration -> Idioma)



Hello Android!

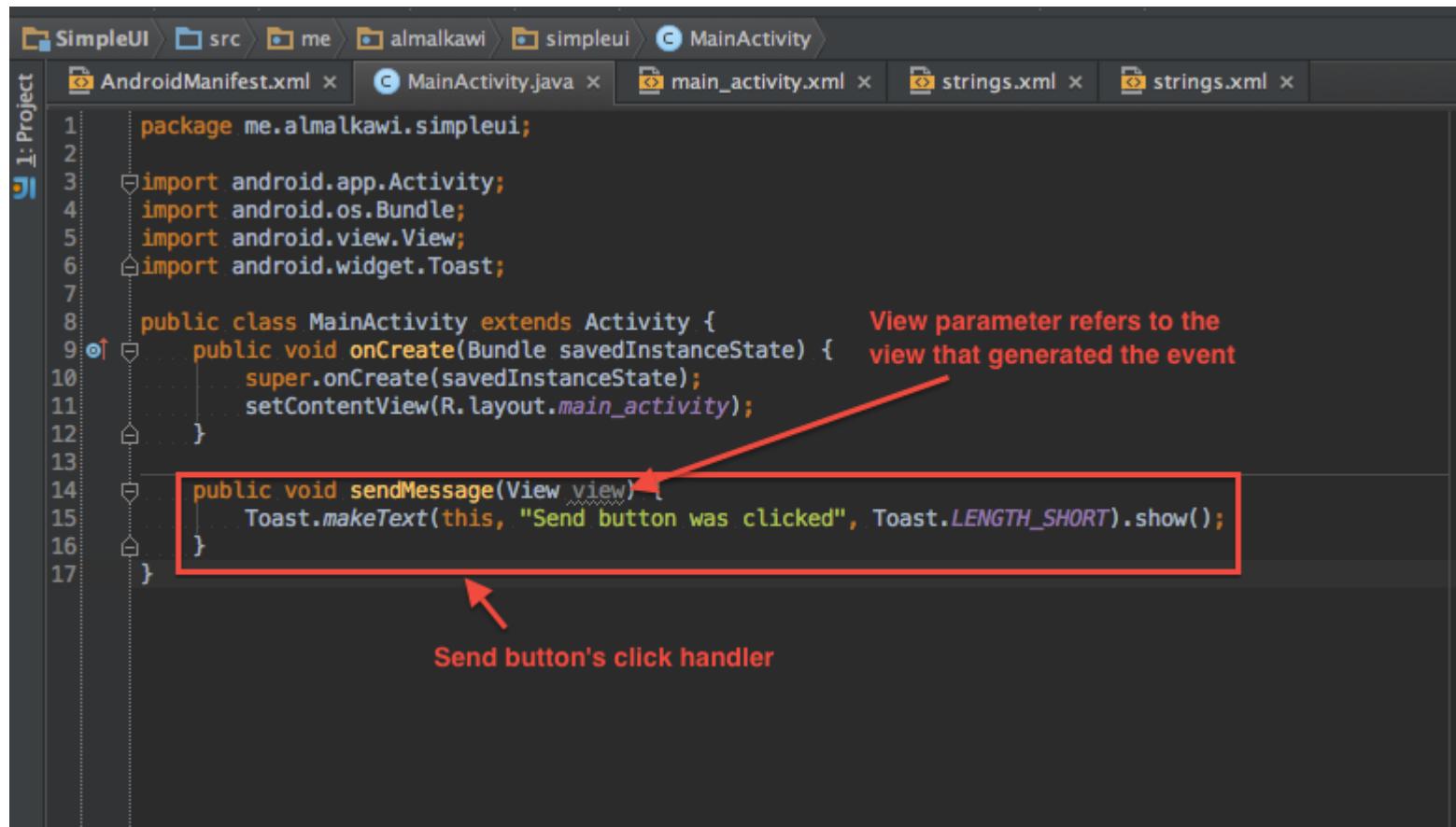
Event handlers

Specify click handler for Button



Implement click handler

Add click handler to MainActivity.java

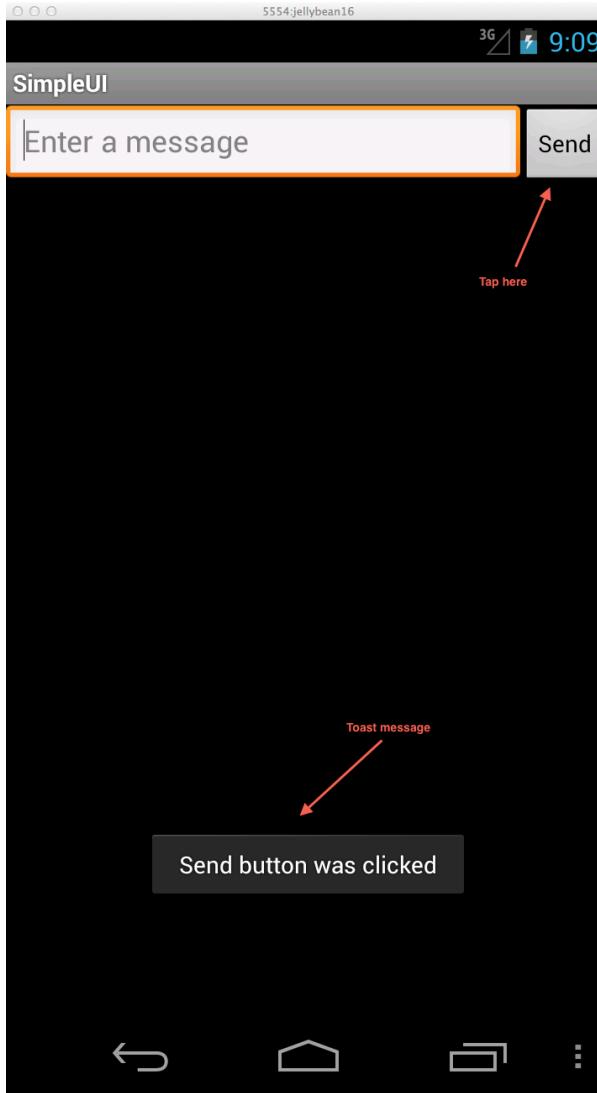


The screenshot shows the Android Studio interface with the project 'SimpleUI' open. The 'MainActivity.java' file is the active tab in the editor. The code implements a basic Activity with an onCreate method and a sendMessage method that uses a Toast to show a message when a button is clicked.

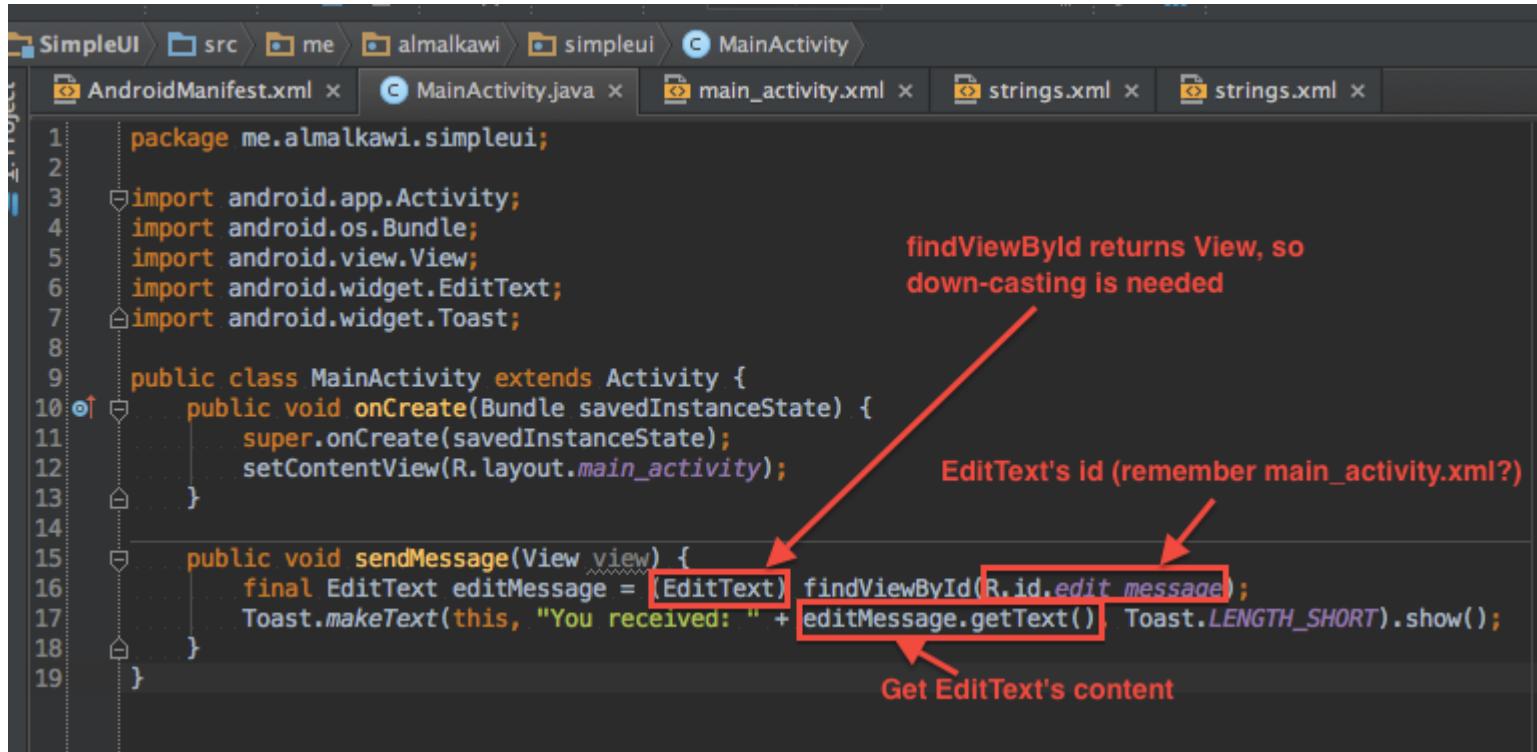
```
1 package me.almalkawi.simpleui;
2
3 import android.app.Activity;
4 import android.os.Bundle;
5 import android.view.View;
6 import android.widget.Toast;
7
8 public class MainActivity extends Activity {
9     @Override
10    public void onCreate(Bundle savedInstanceState) {
11        super.onCreate(savedInstanceState);
12        setContentView(R.layout.main_activity);
13    }
14
15    public void sendMessage(View view) {
16        Toast.makeText(this, "Send button was clicked", Toast.LENGTH_SHORT).show();
17    }
}
```

A red box highlights the `sendMessage` method, and a red arrow points from the text "Send button's click handler" to the start of this method. A red annotation box with the text "View parameter refers to the view that generated the event" has a red arrow pointing to the `view` parameter in the `sendMessage` method signature.

Test click handler



Show EditText content in Toast



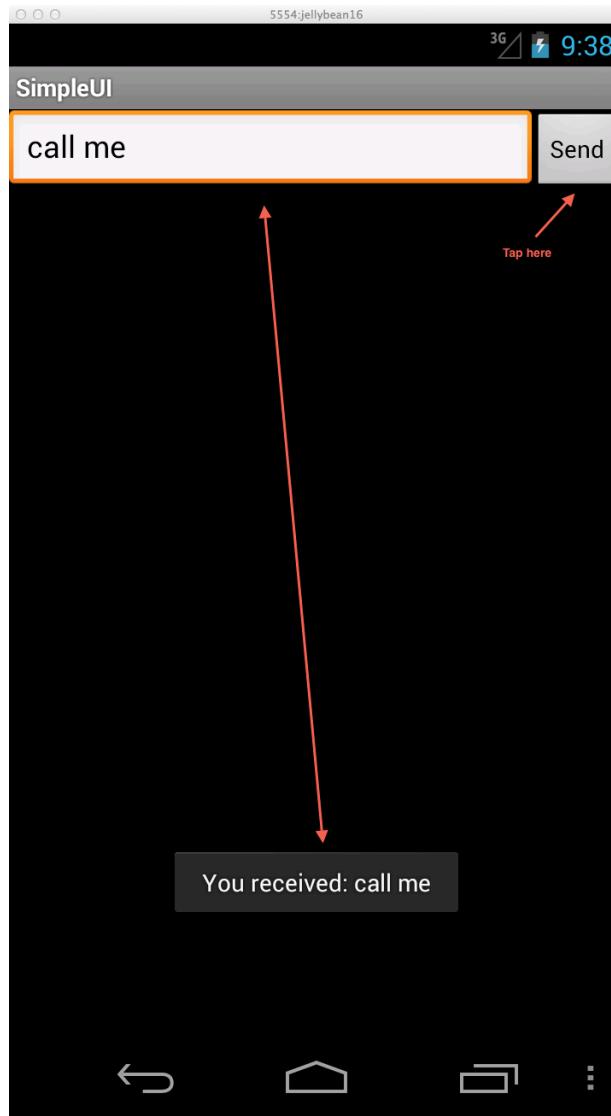
The screenshot shows the Android Studio interface with the project navigation bar at the top. Below it, the file tabs show `AndroidManifest.xml`, `MainActivity.java`, `main_activity.xml`, and `strings.xml`. The main code editor displays `MainActivity.java` with the following content:

```
1 package me.almalkawi.simpleui;
2
3 import android.app.Activity;
4 import android.os.Bundle;
5 import android.view.View;
6 import android.widget.EditText;
7 import android.widget.Toast;
8
9 public class MainActivity extends Activity {
10     @Override
11     protected void onCreate(Bundle savedInstanceState) {
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.main_activity);
14     }
15     public void sendMessage(View view) {
16         final EditText editMessage = (EditText) findViewById(R.id.edit_message);
17         Toast.makeText(this, "You received: " + editMessage.getText(), Toast.LENGTH_SHORT).show();
18     }
19 }
```

Annotations with arrows explain specific parts of the code:

- An arrow points to the line `final EditText editMessage = (EditText) findViewById(R.id.edit_message);` with the text "findViewByld returns View, so down-casting is needed".
- An arrow points to the line `R.id.edit_message` with the text "EditText's id (remember main_activity.xml?)".
- An arrow points to the line `editMessage.getText()` with the text "Get EditText's content".

Test reading EditText content

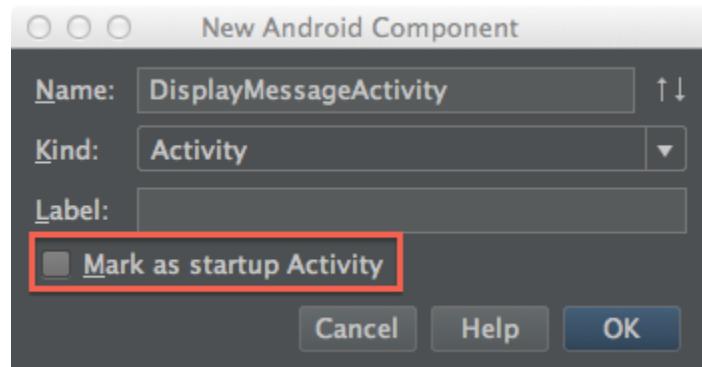


Hello Android!

Intents

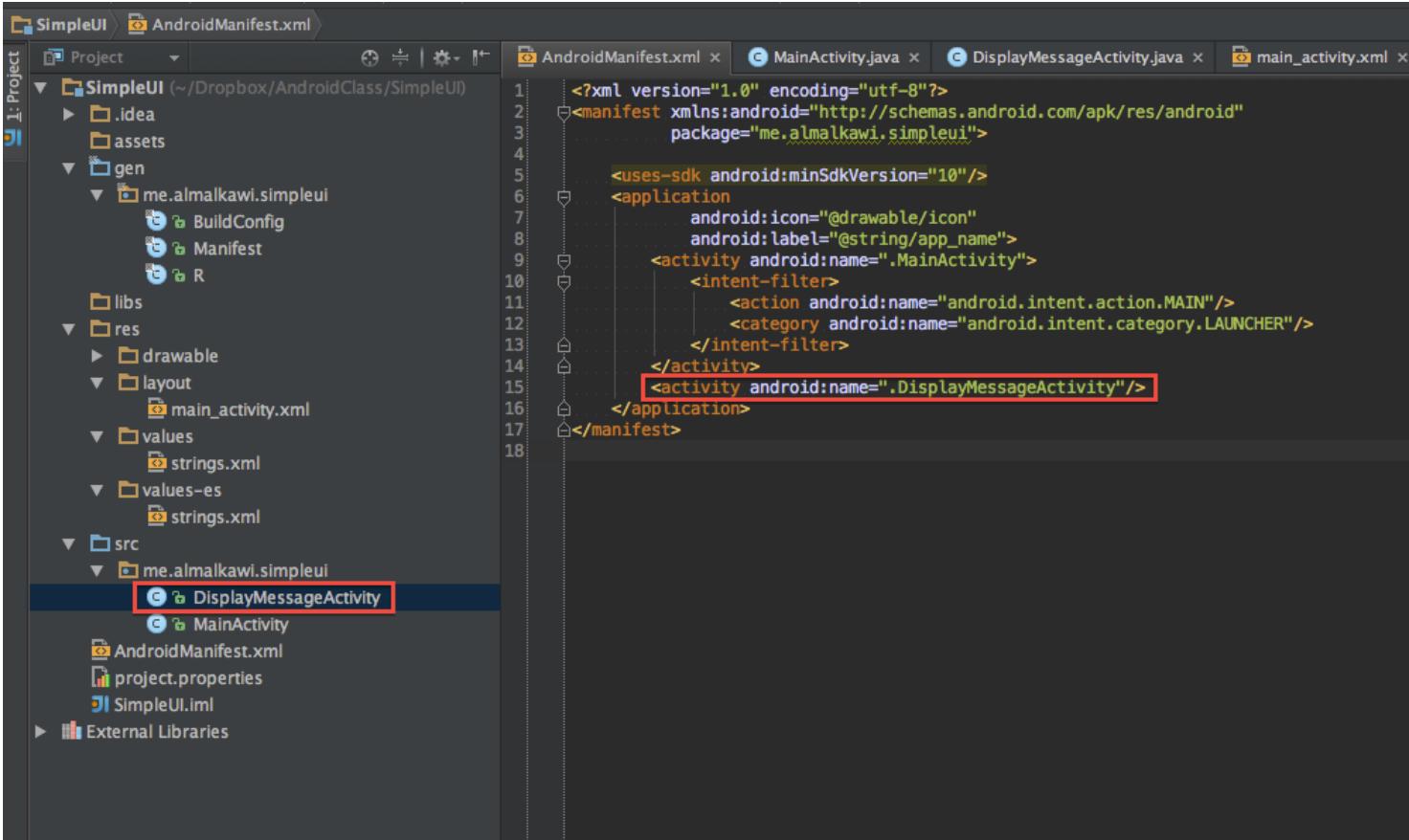
Add second Activity to SimpleUI

- Create a second Activity (DisplayMessageActivity)



Second Activity declared in manifest

New Activity is automatically added to the manifest.



The screenshot shows the Android Studio interface with the following details:

- Project View:** Shows the project structure under "SimpleUI". It includes folders for .idea, assets, gen, libs, res (with drawable, layout, values, and values-es), and src (with me.almalkawi.simpleui containing MainActivity and DisplayMessageActivity). The "DisplayMessageActivity" file is currently selected and highlighted with a red border.
- Manifest Editor:** Shows the XML code for the AndroidManifest.xml file. The code defines a package named "me.almalkawi.simpleui" with a minimum SDK version of 10. It contains an application tag with an icon and label. Two activity tags are present: one for "MainActivity" and one for "DisplayMessageActivity". The "DisplayMessageActivity" tag is highlighted with a red box.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="me.almalkawi.simpleui">
    <uses-sdk android:minSdkVersion="10"/>
    <application
        android:icon="@drawable/icon"
        android:label="@string/app_name">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>
        <activity android:name=".DisplayMessageActivity"/>
    </application>
</manifest>
```

Create Intent

Create Intent to start the DisplayMessageActivity activity

```
androidManifest.xml x MainActivity.java x DisplayMessageActivity.java x main_activity.xml x strings.xml x strings.xml x
package me.almalkawi.simpleui;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class MainActivity extends Activity {
    public static final String EXTRA_MESSAGE = "SimpleUI_MESSAGE";

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

    public void sendMessage(View view) {
        final EditText editText = (EditText) findViewById(R.id.edit_message);
        final Intent intent = new Intent(this, DisplayMessageActivity.class);
        intent.putExtra(EXTRA_MESSAGE, editText.getText().toString());
        startActivity(intent);
    }
}
```

The OS receives this call and starts an instance of DisplayMessageActivity and passes it the intent.

Defines the Extra name as a public constant is needed to allow the activity receiving the intent to query for the extra by name.

Intent represents intent to do something

Second parameter is the class of the activity that should be started to receive the intent

key-value pair of data (the payload of the intent)

Reads EditText's content

Second Activity's Implementation

Activity receiving the Intent Extra

```
AndroidManifest.xml x MainActivity.java x DisplayMessageActivity.java x main_activity.xml x strings.xml x strings.x
package me.almalkawi.simpleui;

import ...

public class DisplayMessageActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);      Get the Intent that started the Activity
        // Get the message from the intent
        final Intent intent = getIntent();      Extract the string Extra delivered with the Intent
        final String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

        // Create the text view
        final TextView textView = new TextView(this);
        textView.setTextSize(40);
        textView.setText(message);

        // Set the text view as the activity layout
        setContentView(textView);      Makes the TextView the root View of the UI
    }
}
```

The screenshot shows the Android Studio interface with several tabs at the top: AndroidManifest.xml, MainActivity.java, DisplayMessageActivity.java, main_activity.xml, strings.xml, and strings.xml. The DisplayMessageActivity.java tab is active, showing the following Java code:

```
package me.almalkawi.simpleui;

import ...

public class DisplayMessageActivity extends Activity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);      Get the Intent that started the Activity
        // Get the message from the intent
        final Intent intent = getIntent();      Extract the string Extra delivered with the Intent
        final String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);

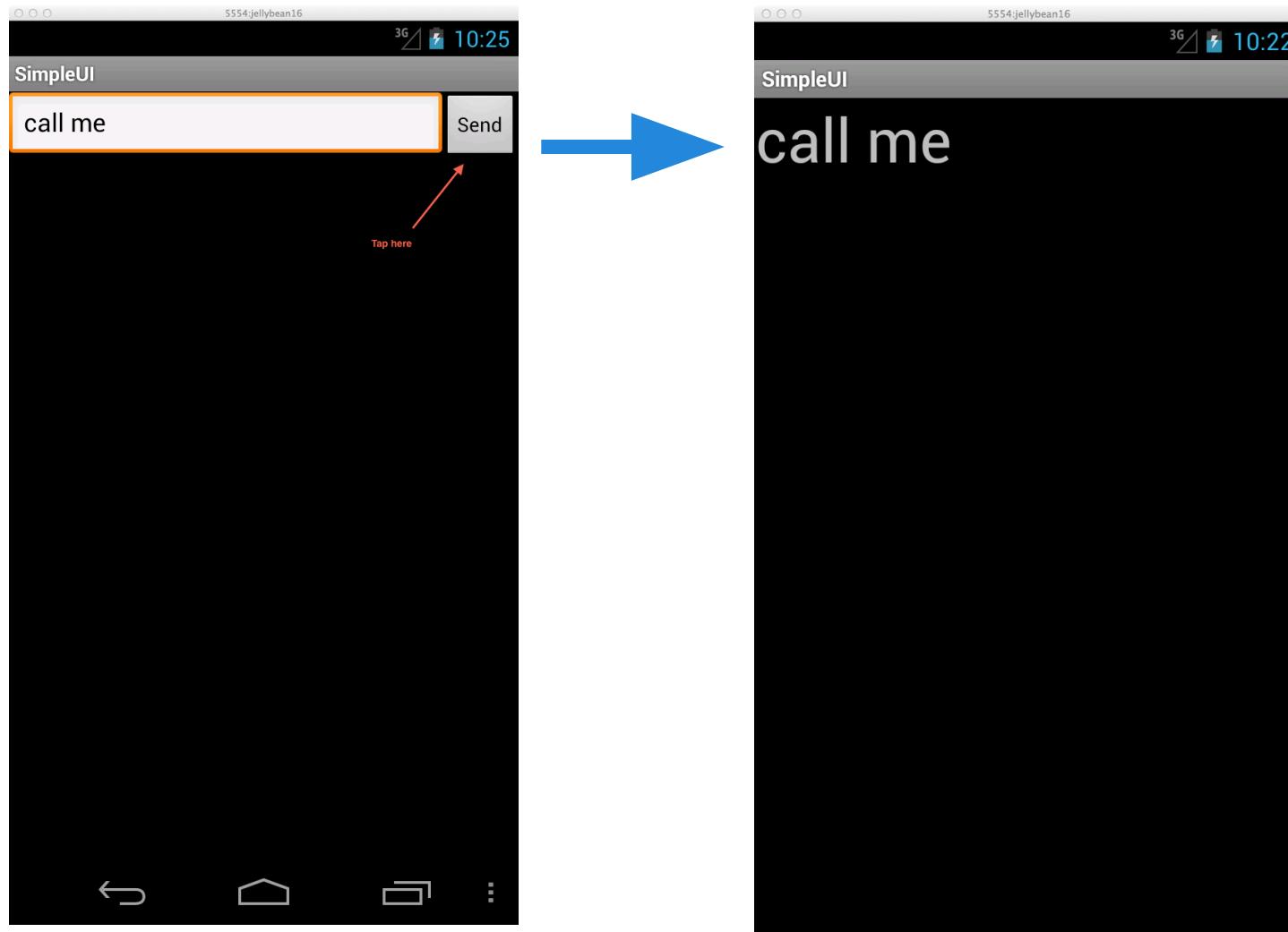
        // Create the text view
        final TextView textView = new TextView(this);
        textView.setTextSize(40);
        textView.setText(message);

        // Set the text view as the activity layout
        setContentView(textView);      Makes the TextView the root View of the UI
    }
}
```

Annotations with red arrows explain the code:

- An arrow points to the line `final Intent intent = getIntent();` with the text "Get the Intent that started the Activity".
- An arrow points to the line `final String message = intent.getStringExtra(MainActivity.EXTRA_MESSAGE);` with the text "Extract the string Extra delivered with the Intent".
- An arrow points to the line `setContentView(textView);` with the text "Makes the TextView the root View of the UI".

First Activity starting Second Activity

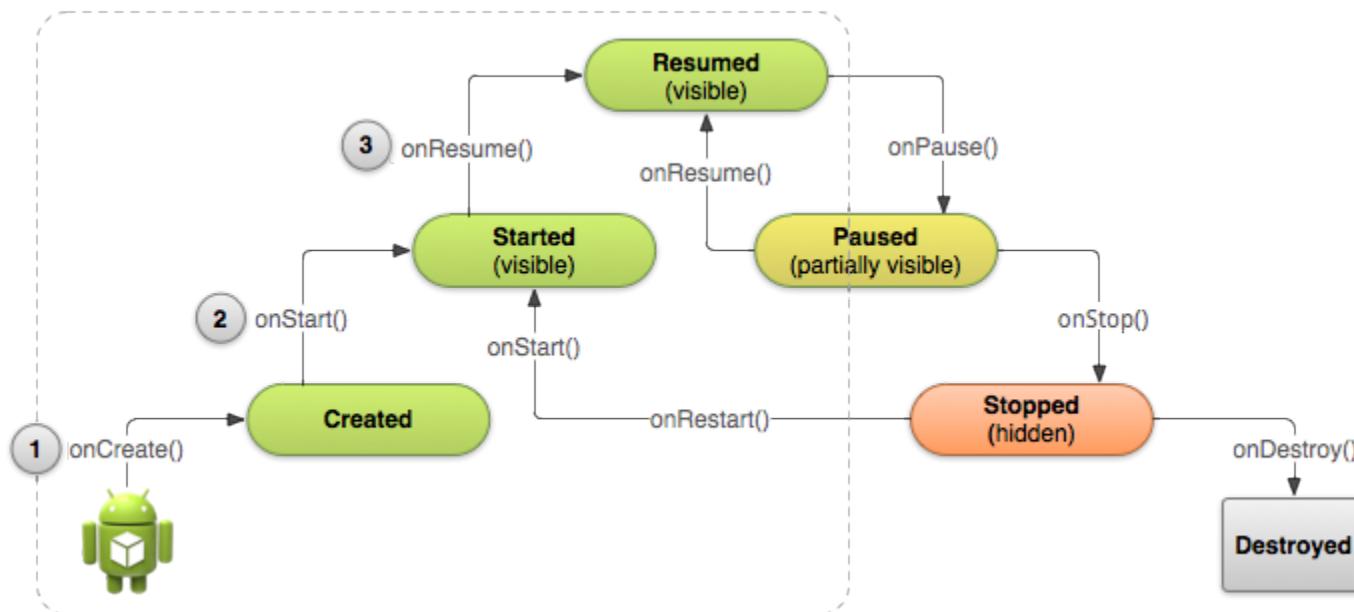


Hello Android!

Activity lifecycle

Activity Lifecycle

Activity instances transition between different states in their lifecycle



Activity lifecycle callbacks

The screenshot shows the Android Studio interface with the project 'ActivityLifecycle' open. The main window displays the code for `MainActivity.java`. Red arrows and annotations explain the purpose of each lifecycle method:

- `onCreate(Bundle savedInstanceState)`:
 - Annotations: "Bundle is a blob of key-value pairs representing previous instance state." and "If null, then Android is creating a new instance of the activity, instead of restoring a previous one that was destroyed."
 - Description: "onCreate is for performing application one-time startup logic (e.g. wiring-up UI view elements)"
- `onRestoreInstanceState(Bundle savedInstanceState)`:
 - Description: "Bundle passed to this method is the same one that was passed to onSaveInstanceState."
- `onRestart()`:
 - Description: "Only called if activity is coming from paused state (generally not used; use use onStart instead as a counterpart of onStop)"
- `onStart()`:
 - Description: "Always called after onCreate"
- `onResume()`:
 - Description: "Always called after onStart. Resumed state means the Activity is in foreground"
- `onSaveInstanceState(Bundle outState)`:
 - Description: "Used save additional state (e.g. member variables that track user's progress in a game)"
 - Description: "Only called if the activity instance might be re-instantiated in future."
- `onPause()`:
 - Description: "Called before entering the paused state as soon as the Activity is partially obscured (guaranteed to be called). Use it to Persist unsaved information that should be saved. Avoid performing CPU-intensive work."
- `onStop()`:
 - Description: "Called after onPause, when activity is in background"
- `onDestroy()`:
 - Description: "Very last callback. Used for cleaning long-running resources (e.g. method tracing). Not guaranteed to be called."
 - Description: "Happens when: a. User presses the Back button b. Calling finish() c. To recover memory on low resources."

A note at the bottom states: "Warning: local class references are destroyed before onDestroy (so most cleanup should be done in onPause and onStop)."

Activity lifecycle demo

Code: <https://github.com/almalkawi/AndroidClass/tree/master/ActivityLifecycle>

Run the following scenarios in emulator:

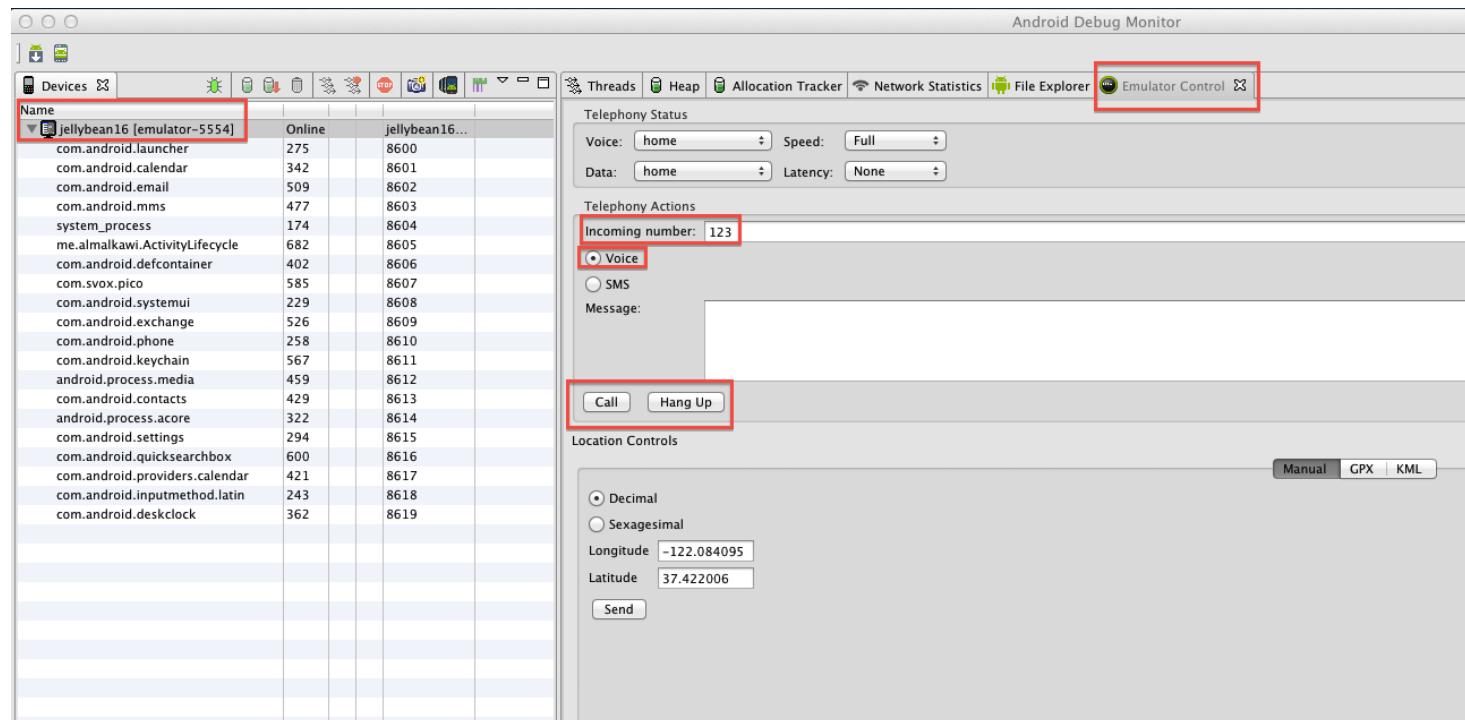
1. Start app for first time: onCreate (null Bundle), onStart, onResume
2. Stop app by clicking on home button: onSaveInstanceState, onPause, onStop
3. Re-start the app again (from Launcher): onRestart, onStart, onResume
4. Destroy the app by clicking on Back: onPause, onStop, onDestroy
5. Start the app again (from Launcher): onCreate (null Bundle), onStart, onResume
6. Press the “Finish” button: onPause, onStop, onDestroy
7. Rotate phone

onSaveInstanceState, onPause, onStop, onDestroy, onCreate (non-null bundle), onStart, onRestoreInstanceState, onResume

Activity lifecycle demo (phone calls)

Two more scenarios:

1. Receive call, then hangup quickly: onSaveInstanceState, onPause, onResume
2. Receive call, then answer: onSaveInstanceState, onPause (+ onStop when answered)



Hello Android!

AdapterViews and Adapters

AdapterViews and Adapters

AdapterView:

- View whose children are determined by an Adapter.
- Populated with data by binding it to an Adapter

Adapter:

- Retrieves data from external source and creates a View for each entry.
- Common external sources of data are arrays in memory and databases

ArrayAdapter and SimpleCursorAdapter

Android provides subclasses of Adapter for retrieving data and building views. Common ones:

- ArrayAdapter
- SimpleCursorAdapter

ArrayAdapter is used when data comes from an array.

SimpleCursorAdapter is used when data comes from a Cursor (DB).

Demo

ListView is an AdapterView that displays a list of scrollable items.

Example of ListView is tweets home timeline.

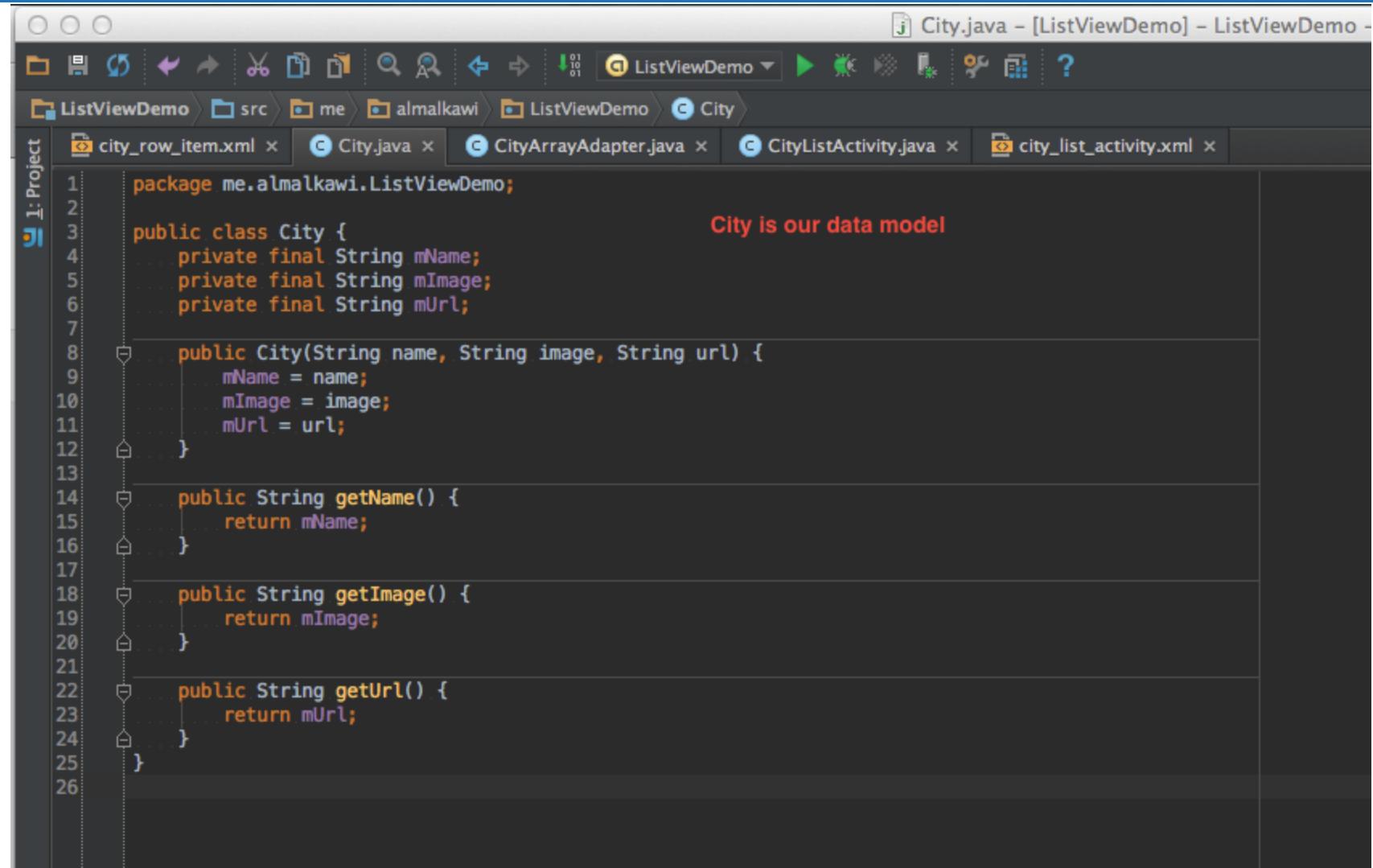
Create new Project (ListViewDemo)

Code Walkthroughs is in next set of slides

Images: <https://github.com/almalkawi/AndroidClass/tree/master/ListViewDemo/res/drawable>

Complete Code: <https://github.com/almalkawi/AndroidClass/tree/master/ListViewDemo>

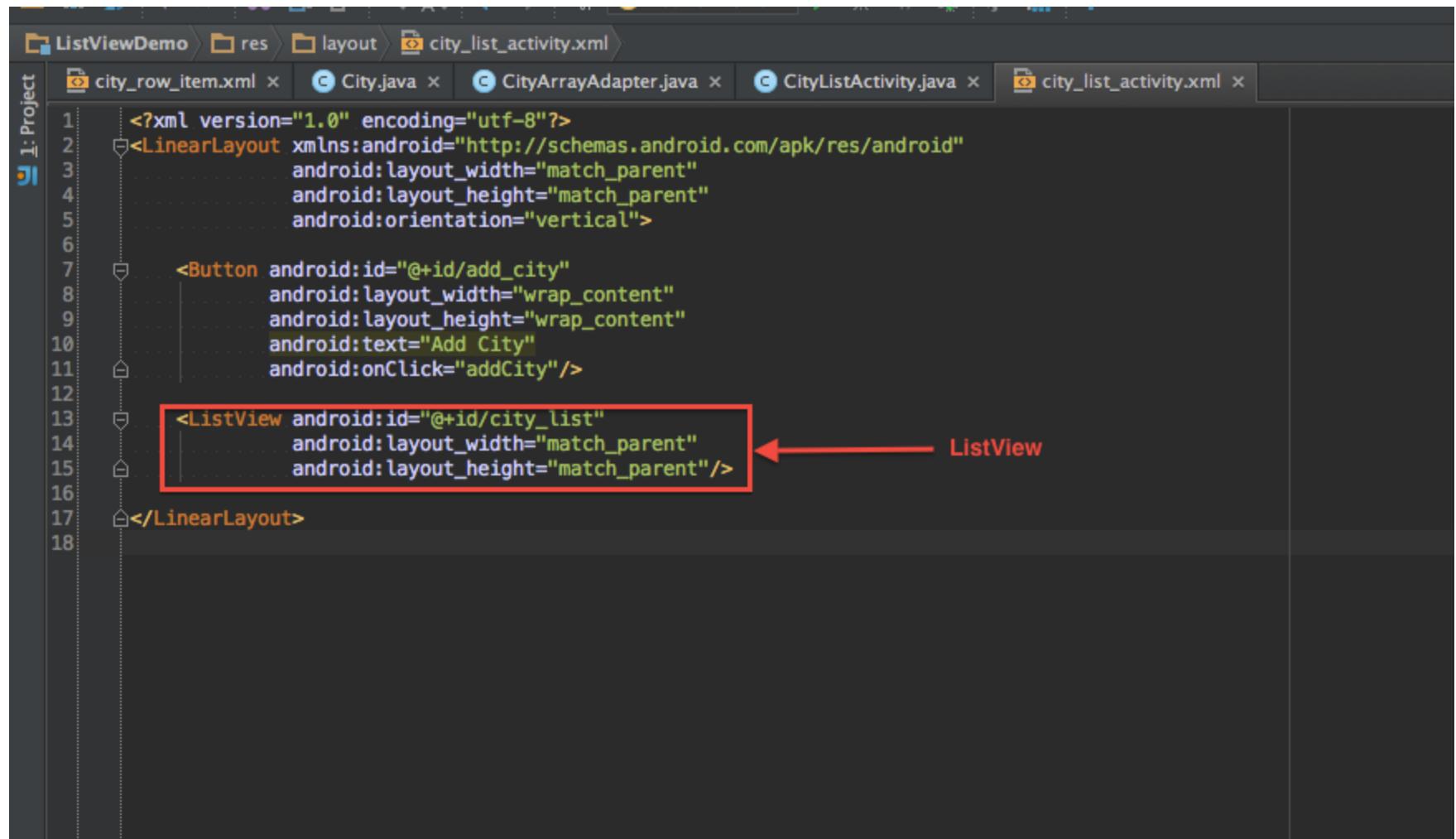
City Data Model (City.java)



The screenshot shows the Android Studio interface with the City.java file open in the main editor. The code defines a City class with private fields for name, image, and url, and public getters for each. A red annotation "City is our data model" is placed next to the class definition.

```
1 package me.almalkawi.ListViewDemo;
2
3 public class City {                                City is our data model
4     private final String mName;
5     private final String mImage;
6     private final String mUrl;
7
8     public City(String name, String image, String url) {
9         mName = name;
10        mImage = image;
11        mUrl = url;
12    }
13
14    public String getName() {
15        return mName;
16    }
17
18    public String getImage() {
19        return mImage;
20    }
21
22    public String getUrl() {
23        return mUrl;
24    }
25
26}
```

ListView in XML (city_list_activity.xml)



The screenshot shows the Android Studio interface with the project 'ListViewDemo' open. The 'city_list_activity.xml' file is selected in the navigation bar. The XML code defines a linear layout containing a button and a list view. The list view is highlighted with a red box and labeled 'ListView' with a red arrow.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <Button android:id="@+id/add_city"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Add City"
        android:onClick="addCity"/>

    <ListView android:id="@+id/city_list"
        android:layout_width="match_parent"
        android:layout_height="match_parent"/>

</LinearLayout>
```

City ListView Activity (CityListActivity.java)

```
1 package me.almalkawi.ListViewDemo;
2
3 import ...
4
5 public class CityListActivity extends Activity {
6     private ListView mListview;
7     private CityArrayAdapter mCityArrayAdapter;
8     private ArrayList<City> cities;
9
10    public void onCreate(Bundle savedInstanceState) {
11        super.onCreate(savedInstanceState);
12        setContentView(R.layout.city_list_activity);
13
14        // Data
15        cities = new ArrayList<City>();
16        cities.add(new City("San Francisco", "sf", "http://en.wikipedia.org/wiki/San_Francisco"));
17        cities.add(new City("Seattle", "seattle", "http://en.wikipedia.org/wiki/Seattle"));
18        cities.add(new City("Chicago", "chicago", "http://en.wikipedia.org/wiki/Chicago"));
19        cities.add(new City("London", "london", "http://en.wikipedia.org/wiki/London"));
20        cities.add(new City("Mumbai", "mumbai", "http://en.wikipedia.org/wiki/Mumbai"));
21        cities.add(new City("Moscow", "moscow", "http://en.wikipedia.org/wiki/Moscow"));
22        cities.add(new City("Sydney", "sydney", "http://en.wikipedia.org/wiki/Sydney"));
23
24        mListview = (ListView) findViewById(R.id.city_list);
25        mCityArrayAdapter = new CityArrayAdapter(this, R.layout.city_row_item, cities);
26        mListview.setAdapter(mCityArrayAdapter);
27        mListview.setOnItemClickListener(new AdapterView.OnItemClickListener() {
28            @Override
29            public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
30                final City city = (City) parent.getItemAtPosition(position);
31                final Intent intent = new Intent(Intent.ACTION_VIEW);
32                intent.setData(Uri.parse(city.getUrl()));
33                startActivity(intent);
34            }
35        });
36    }
37
38    public void addCity(View view) {
39        cities.add(new City("Tokyo", "tokyo", "http://en.wikipedia.org/wiki/Tokyo"));
40        mCityArrayAdapter.notifyDataSetChanged();
41    }
42}
```

Initial data for the ArrayAdapter

Create an ArrayAdapter:
- Specify layout of each row (city_row_item)
- Specify data array (cities)

Binds the ListView to the ArrayAdapter

Get the city that was clicked

Browser URI

AddCity button click handler

This changes the underlying data that is read by the ArrayAdapter

This is necessary to notify the attached ListView that the data has been changed and that it should refresh itself.

Layout of ListView rows (city_row_item.xml)

The screenshot shows the Android Studio interface with the project 'ListViewDemo' open. The 'city_row_item.xml' file is selected in the layout editor. The code is as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">

    <ImageView android:id="@+id/city_image"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"/>

    <TextView android:id="@+id/city_name"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:gravity="center_vertical"
        android:paddingLeft="10dp"
        android:textSize="20sp"/>

</LinearLayout>
```

A red box highlights the `<ImageView>` tag, and a red arrow points from it to the explanatory text on the right.

imageView is for displaying an image (duh)

Layout of each row in the ListView

City ArrayAdapter (CityArrayAdapter.java)

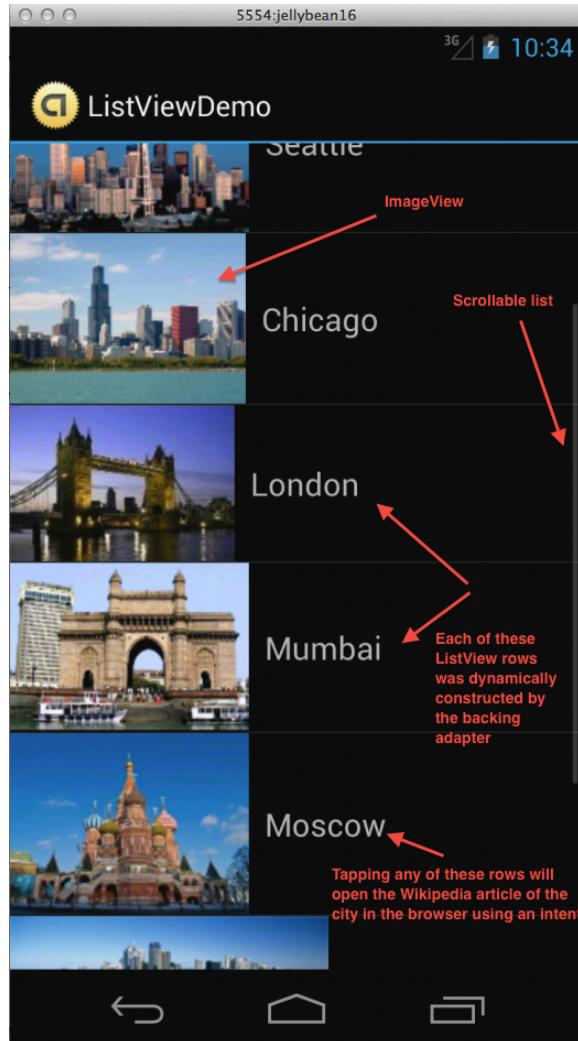
The screenshot shows the Android Studio interface with the project structure and code editor. The project structure on the left shows a folder named 'drawable' containing several PNG files: chicago.png, icon.png, london.png, moscow.png, mumbai.png, seattle.png, sf.png, sydney.png, and tokyo.png. The code editor on the right contains the Java code for the CityArrayAdapter class.

```
1 package me.almarkawi.ListViewDemo;
2
3 import android.content.Context;
4 import android.content.res.Resources;
5 import android.view.LayoutInflater;
6 import android.view.View;
7 import android.view.ViewGroup;
8 import android.widget.ArrayAdapter;
9 import android.widget.ImageView;
10 import android.widget.TextView;
11
12 import java.util.List;
13
14 public class CityArrayAdapter extends ArrayAdapter {
15     private int mResource;
16     private LayoutInflater mInflater;
17     private Context mContext;
18
19     public CityArrayAdapter(Context context, int resource, List objects) {
20         super(context, resource, objects);
21
22         mInflater = LayoutInflater.from(context); // used to create a view of the layout
23         mContext = context; // need it to access resources
24         mResource = resource; // the custom layout ID
25     }
26
27     @Override
28     public View getView(int position, View convertView, ViewGroup parent) {
29         if (convertView == null) { // Create a new view and inflate it in the row.
30             convertView = mInflater.inflate(mResource, null);
31         }
32
33         // Extract the city object
34         final City city = (City) getItem(position);
35
36         // Set the city name
37         final TextView cityName = (TextView) convertView.findViewById(R.id.city_name);
38         cityName.setText(city.getName());
39
40         // Set the city drawable
41         final Resources res = mContext.getResources();
42         final int id = res.getIdentifier(city.getImage(), "drawable", mContext.getPackageName());
43         final ImageView cityImage = (ImageView) convertView.findViewById(R.id.city_image);
44         cityImage.setImageDrawable(res.getDrawable(id));
45
46         return convertView;
47     }
48 }
49
```

Annotations in the code:

- A red box highlights the 'drawable' folder in the project structure, with a red arrow pointing to the corresponding line of code in the adapter's constructor where `mResource` is set to the custom layout ID.
- A callout bubble points to the `@Override` annotation on line 27 with the text: "Creates a view for each item in the ListView".
- A callout bubble points to the `if (convertView == null)` condition on line 29 with the text: "For perf reasons, Android recycles views and passes a reference to the recycled view in convertView. So, only create new view if null".
- A red box highlights the code for setting the city image on line 43, with a red arrow pointing to the corresponding line of code in the `getView` method where `cityImage.setImageDrawable(res.getDrawable(id))` is called.

ListView Demo in emulator



Hello Android!

Lab exercise

TO-DO List

- Create a TO-DO list app.
- Concepts to incorporate:
 - Activity
 - EditText and KeyListener
 - ListView
 - ArrayAdapter
- UI Mockup is on next slide

TO-DO App UI Mockup



Hello Android!

Dynamic UIs using Fragments

Fragments

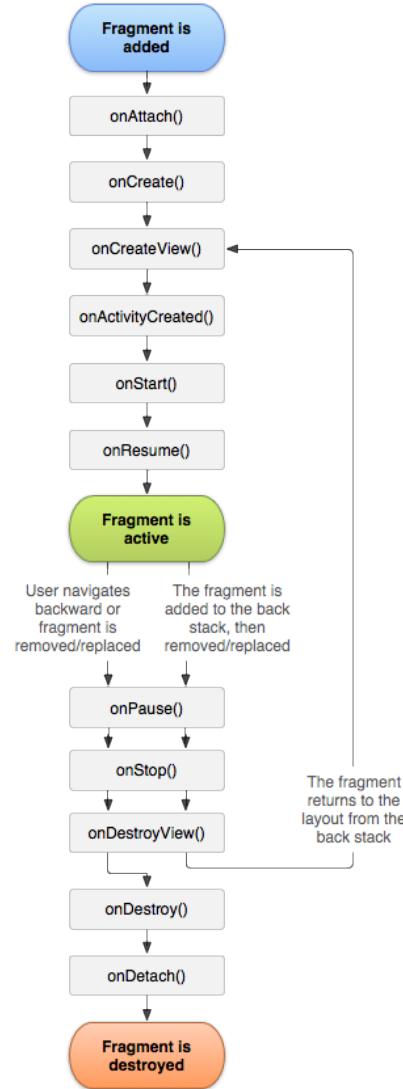
Fragments are:

- Reusable, modular UI components.
- For creating dynamic and multi-pane interfaces.
- Can be swapped into/out of activities based on things like screen size.

Each instance of a Fragment class must be associated with a parent Activity.

If Activity is a controller, then Fragments are mini-controllers.

Fragment Lifecycle



Associating Fragment with Activity

Fragment can be associated with parent activity:

- Using the activity's layout XML file.
- Added dynamically at run-time

Fragments do nt need to be registered in the manifest.
(because they can ***not*** exist on their own. They are
embedded in other activities).

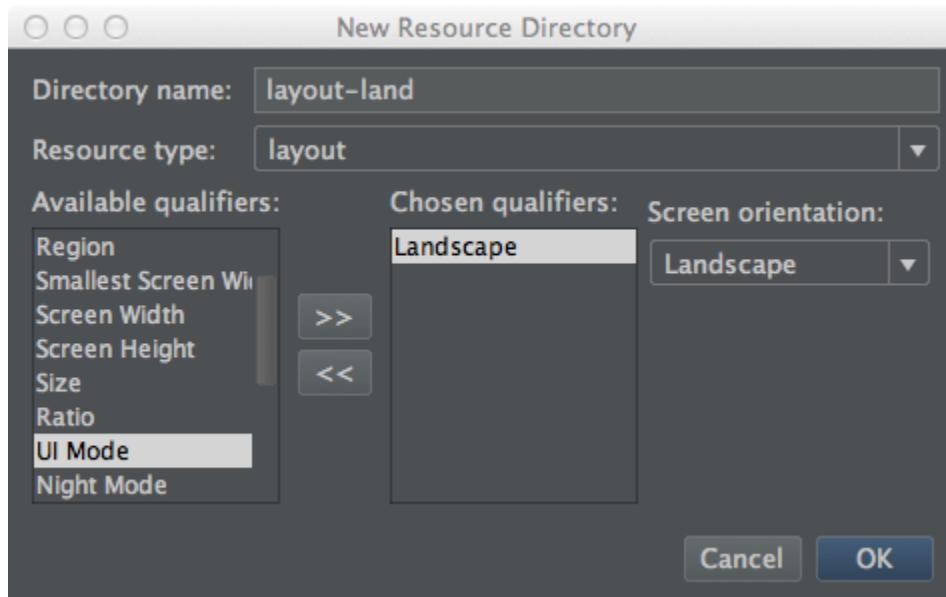
Demo: Fragments using XML

Source: <https://github.com/almalkawi/AndroidClass/tree/master/FragmentsXML>

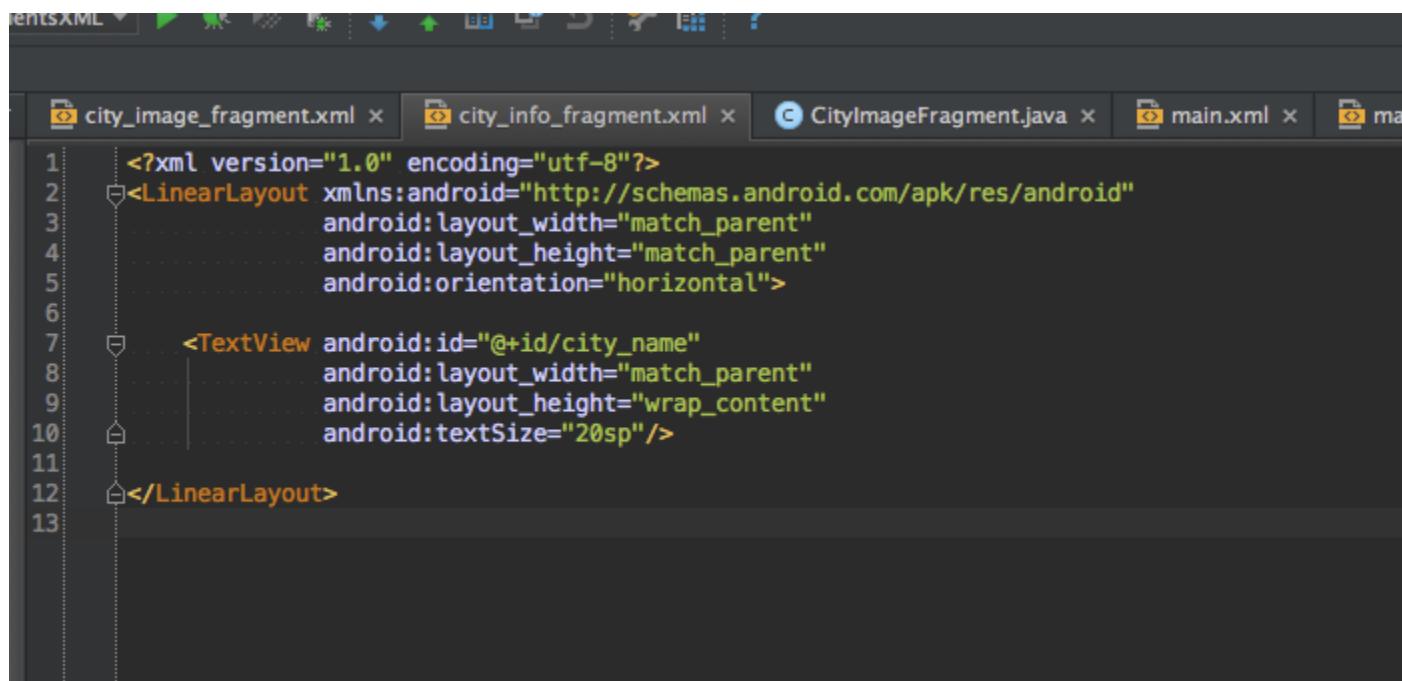
Demo demonstrates handling different screen sizes/orientation.

Run project in emulator and observe lifecycle of Activity and Fragment through Toast messages.

Create landscape layout



CityInfo Fragment UI (city_info_fragment.xml)



The screenshot shows the Android Studio interface with the code editor open. The tab bar at the top includes 'city_image_fragment.xml', 'city_info_fragment.xml' (which is the active tab), 'CityImageFragment.java', 'main.xml', and 'main.java'. The code editor displays the following XML layout:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
    <TextView android:id="@+id/city_name"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:textSize="20sp"/>
</LinearLayout>
```

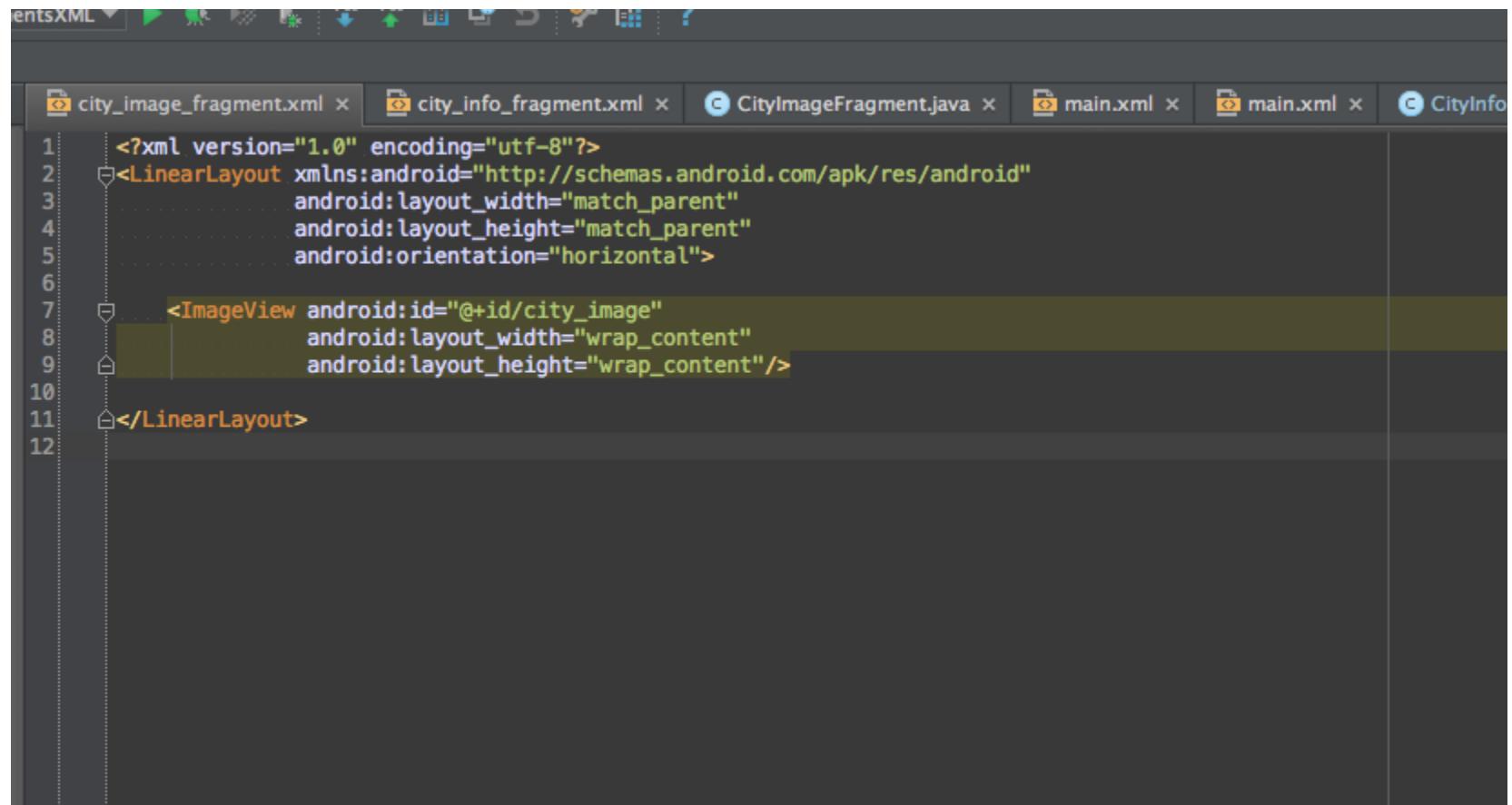
CityInfo Fragment (CityInfoFragment.java)

The screenshot shows the Android Studio interface with the CityInfoFragment.java file open in the code editor. The code defines a Fragment that overrides several lifecycle methods to show Toast messages indicating the current state. Red arrows and text annotations explain each method's purpose:

- onAttach(Activity activity):** Called when Fragment is attached to its parent Activity (usually used to get reference to parent).
- onCreate(Bundle savedInstanceState):** Used to create Fragment's UI.
- onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState):** Inflate UI.
- onActivityCreated(Bundle savedInstanceState):** Called once the parent Activity and the Fragment's UI have been created.
- onStart():** Same meaning as in Activity lifecycle.
- onResume():** Same meaning as in Activity lifecycle.
- onPause():** Equivalent to Activity's onDestroy.
- onStop():** Called when the Fragment has been detached from its parent Activity.
- onDestroyView():** Called when the Fragment has been detached from its parent Activity.
- onDetach():** Called when the Fragment has been detached from its parent Activity.

```
12 public class CityInfoFragment extends Fragment {
13     @Override
14     public void onAttach(Activity activity) {
15         super.onAttach(activity);
16         Toast.makeText(getActivity(), "Fragment: onAttach", Toast.LENGTH_SHORT).show();
17     }
18
19     @Override
20     public void onCreate(Bundle savedInstanceState) {
21         super.onCreate(savedInstanceState);
22         Toast.makeText(getActivity(), "Fragment: onCreate", Toast.LENGTH_SHORT).show();
23     }
24
25     @Override
26     public View onCreateView(LayoutInflater inflater, ViewGroup container,
27                             Bundle savedInstanceState) {
28         final View ui = inflater.inflate(R.layout.city_info_fragment, container, false);
29         final TextView cityName = (TextView) ui.findViewById(R.id.city_name);
30         cityName.setText("Chicago");
31
32         return ui; // If the fragment has no UI, we return null.
33     }
34
35     @Override
36     public void onActivityCreated(Bundle savedInstanceState) {
37         super.onActivityCreated(savedInstanceState);
38         Toast.makeText(getActivity(), "Fragment: onActivityCreated", Toast.LENGTH_SHORT).show();
39     }
40
41     @Override
42     public void onStart() {
43         super.onStart();
44         Toast.makeText(getActivity(), "Fragment: onStart", Toast.LENGTH_SHORT).show();
45     }
46
47     @Override
48     public void onResume() {
49         super.onResume();
50         Toast.makeText(getActivity(), "Fragment: onResume", Toast.LENGTH_SHORT).show();
51     }
52
53     @Override
54     public void onPause() {
55         super.onPause();
56         Toast.makeText(getActivity(), "Fragment: onPause", Toast.LENGTH_SHORT).show();
57     }
58
59     @Override
60     public void onStop() {
61         super.onStop();
62         Toast.makeText(getActivity(), "Fragment: onStop", Toast.LENGTH_SHORT).show();
63     }
64
65     @Override
66     public void onDestroyView() {
67         super.onDestroyView();
68         Toast.makeText(getActivity(), "Fragment: onDestroyView", Toast.LENGTH_SHORT).show();
69     }
70
71     @Override
72     public void onDetach() {
73         super.onDetach();
74         Toast.makeText(getActivity(), "Fragment: onDetach", Toast.LENGTH_SHORT).show();
75     }
76 }
```

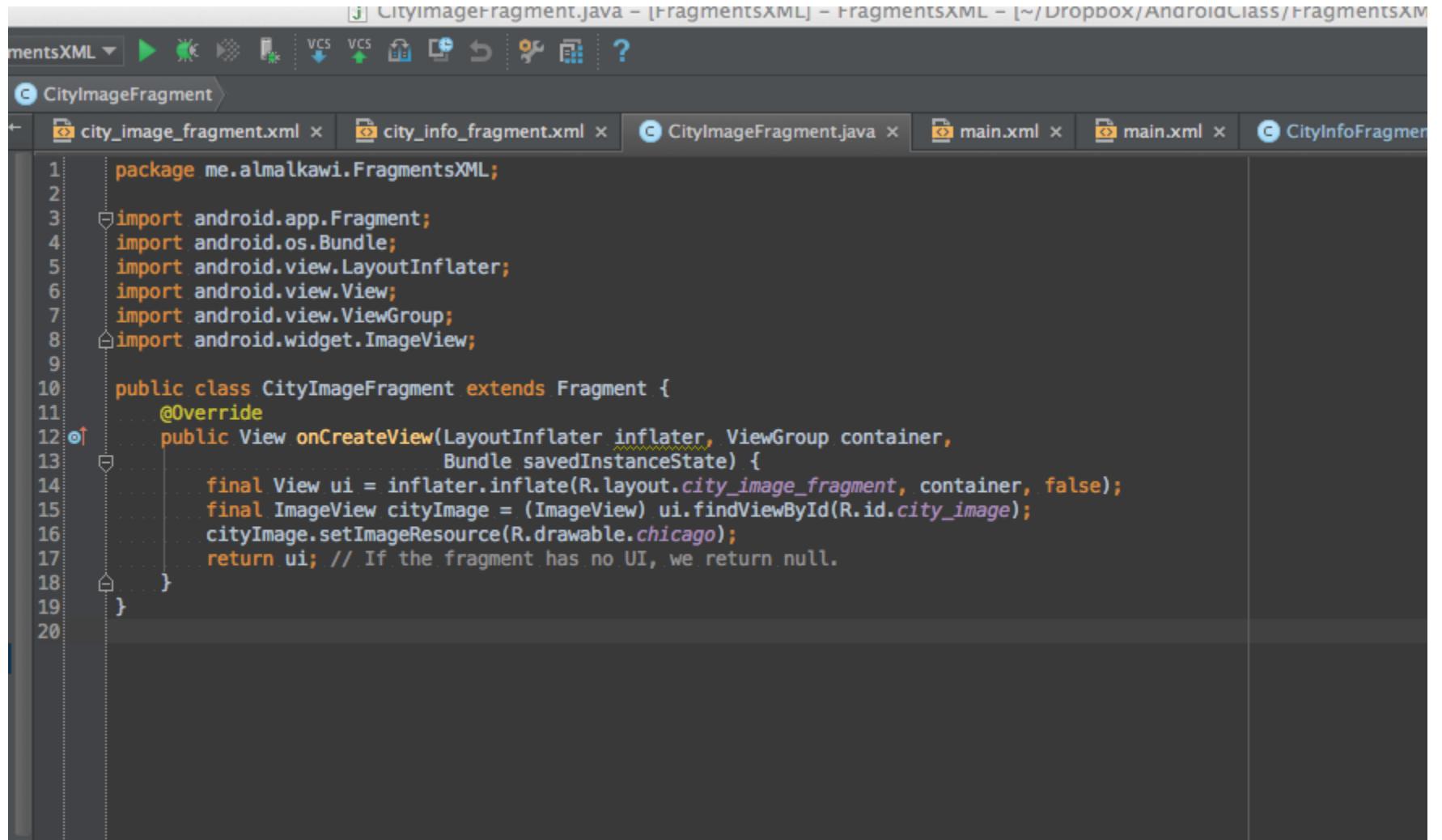
CityImage Fragment UI (city_image_fragment.xml)



The screenshot shows the Android Studio interface with the XML editor open. The title bar has tabs for 'city_image_fragment.xml' (which is the active tab), 'city_info_fragment.xml', 'CityImageFragment.java', 'main.xml', 'main.xml', and 'CityInfo'. The code editor displays the following XML:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal">
    <ImageView android:id="@+id/city_image"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"/>
</LinearLayout>
```

CityImage Fragment (CityImageFragment.java)



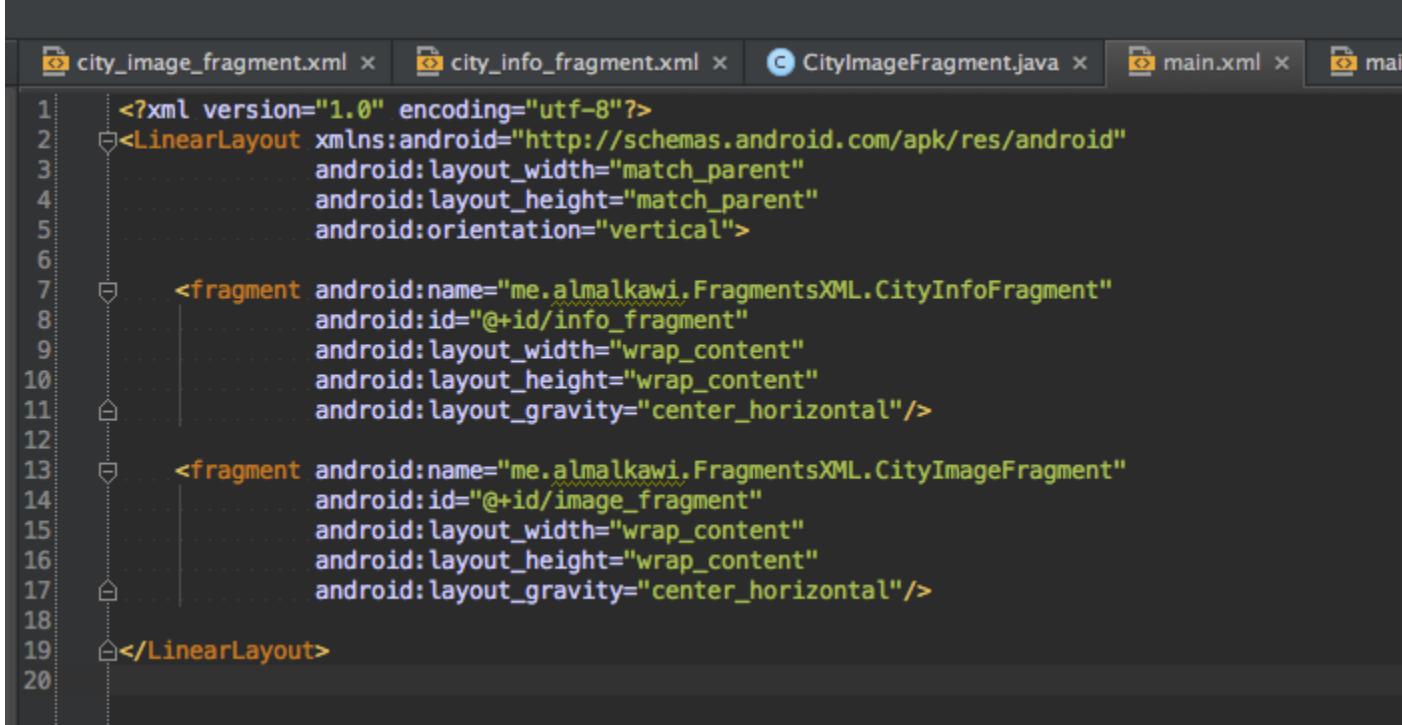
The screenshot shows the Android Studio interface with the code editor open to the file `CityImageFragment.java`. The title bar indicates the current file is `CityImageFragment.java`. The code editor displays Java code for a fragment that inflates a layout and sets an image resource for an ImageView.

```
1 package me.almalkawi.FragmentsXML;
2
3 import android.app.Fragment;
4 import android.os.Bundle;
5 import android.view.LayoutInflater;
6 import android.view.View;
7 import android.view.ViewGroup;
8 import android.widget.ImageView;
9
10 public class CityImageFragment extends Fragment {
11     @Override
12     public View onCreateView(LayoutInflater inflater, ViewGroup container,
13                             Bundle savedInstanceState) {
14         final View ui = inflater.inflate(R.layout.city_image_fragment, container, false);
15         final ImageView cityImage = (ImageView) ui.findViewById(R.id.city_image);
16         cityImage.setImageResource(R.drawable.chicago);
17         return ui; // If the fragment has no UI, we return null.
18     }
19 }
20
```

MainActivity.java

```
>MainActivity
city_image_fragment.xml × city_info_fragment.xml × CityImageFragment.java × main.xml ×
1 package me.almalkawi.FragmentsXML;
2
3 import android.app.Activity;
4 import android.os.Bundle;
5 import android.widget.Toast;
6
7 public class MainActivity extends Activity {
8     @Override
9     protected void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         Toast.makeText(this, "Activity: onCreate", Toast.LENGTH_SHORT).show();
12         setContentView(R.layout.main);
13     }
14
15     @Override
16     protected void onStart() {
17         super.onStart();
18         Toast.makeText(this, "Activity: onStart", Toast.LENGTH_SHORT).show();
19     }
20
21     @Override
22     protected void onRestart() {
23         super.onRestart();
24         Toast.makeText(this, "Activity: onRestart", Toast.LENGTH_SHORT).show();
25     }
26
27     @Override
28     protected void onResume() {
29         super.onResume();
30         Toast.makeText(this, "Activity: onResume", Toast.LENGTH_SHORT).show();
31     }
32
33     @Override
34     protected void onPause() {
35         super.onPause();
36         Toast.makeText(this, "Activity: onPause", Toast.LENGTH_SHORT).show();
37     }
38
39     @Override
40     protected void onStop() {
41         super.onStop();
42         Toast.makeText(this, "Activity: onStop", Toast.LENGTH_SHORT).show();
43     }
44
45     @Override
46     protected void onDestroy() {
47         super.onDestroy();
48         Toast.makeText(this, "Activity: onDestroy", Toast.LENGTH_SHORT).show();
49     }
50 }
```

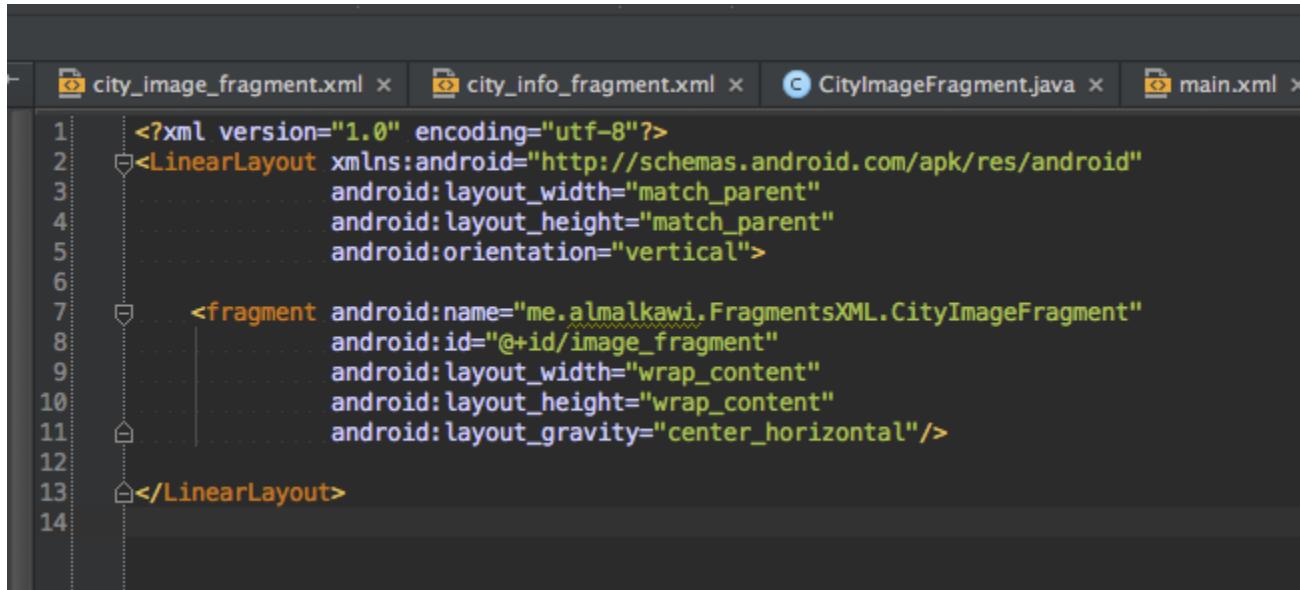
res/layout/main.xml



The screenshot shows the Android Studio code editor with the main.xml layout file open. The tab bar at the top includes files like city_image_fragment.xml, city_info_fragment.xml, CityImageFragment.java, main.xml, and main.java. The main.xml code is as follows:

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
3      android:layout_width="match_parent"
4      android:layout_height="match_parent"
5      android:orientation="vertical">
6
7      <fragment android:name="me.almalkawi.FragmentsXML.CityInfoFragment"
8          android:id="@+id/info_fragment"
9          android:layout_width="wrap_content"
10         android:layout_height="wrap_content"
11         android:layout_gravity="center_horizontal"/>
12
13     <fragment android:name="me.almalkawi.FragmentsXML.CityImageFragment"
14         android:id="@+id/image_fragment"
15         android:layout_width="wrap_content"
16         android:layout_height="wrap_content"
17         android:layout_gravity="center_horizontal"/>
18
19 </LinearLayout>
20
```

res/layout-land/main.xml



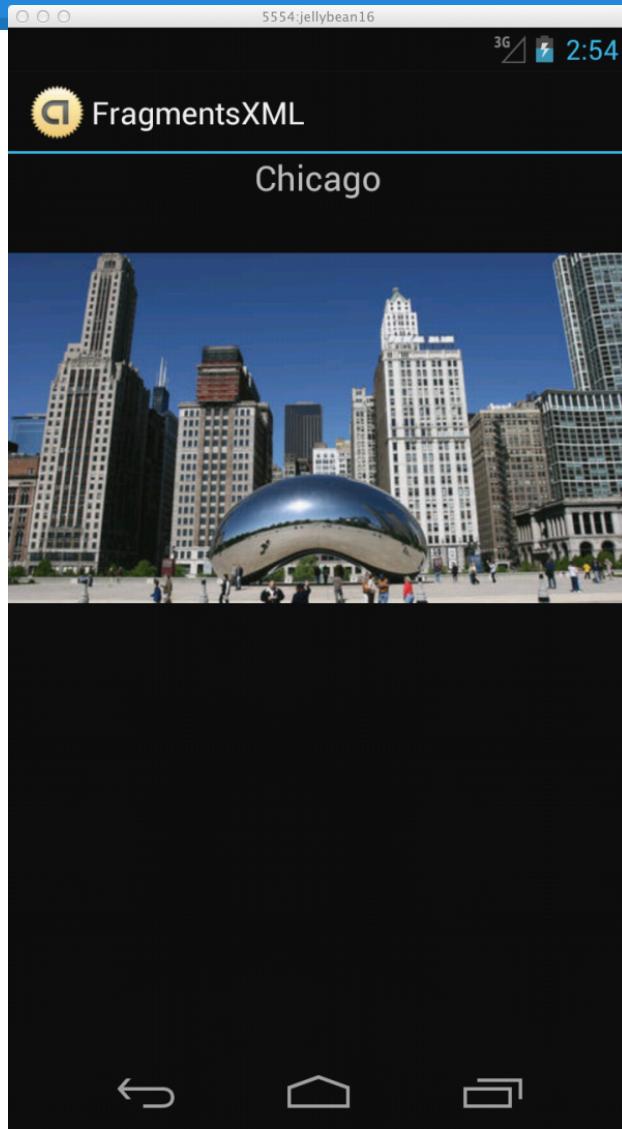
The screenshot shows the Android Studio code editor with the tab bar at the top containing four tabs: "city_image_fragment.xml", "city_info_fragment.xml", "CityImageFragment.java", and "main.xml". The "main.xml" tab is currently selected. The code editor displays the XML content for "main.xml" as follows:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

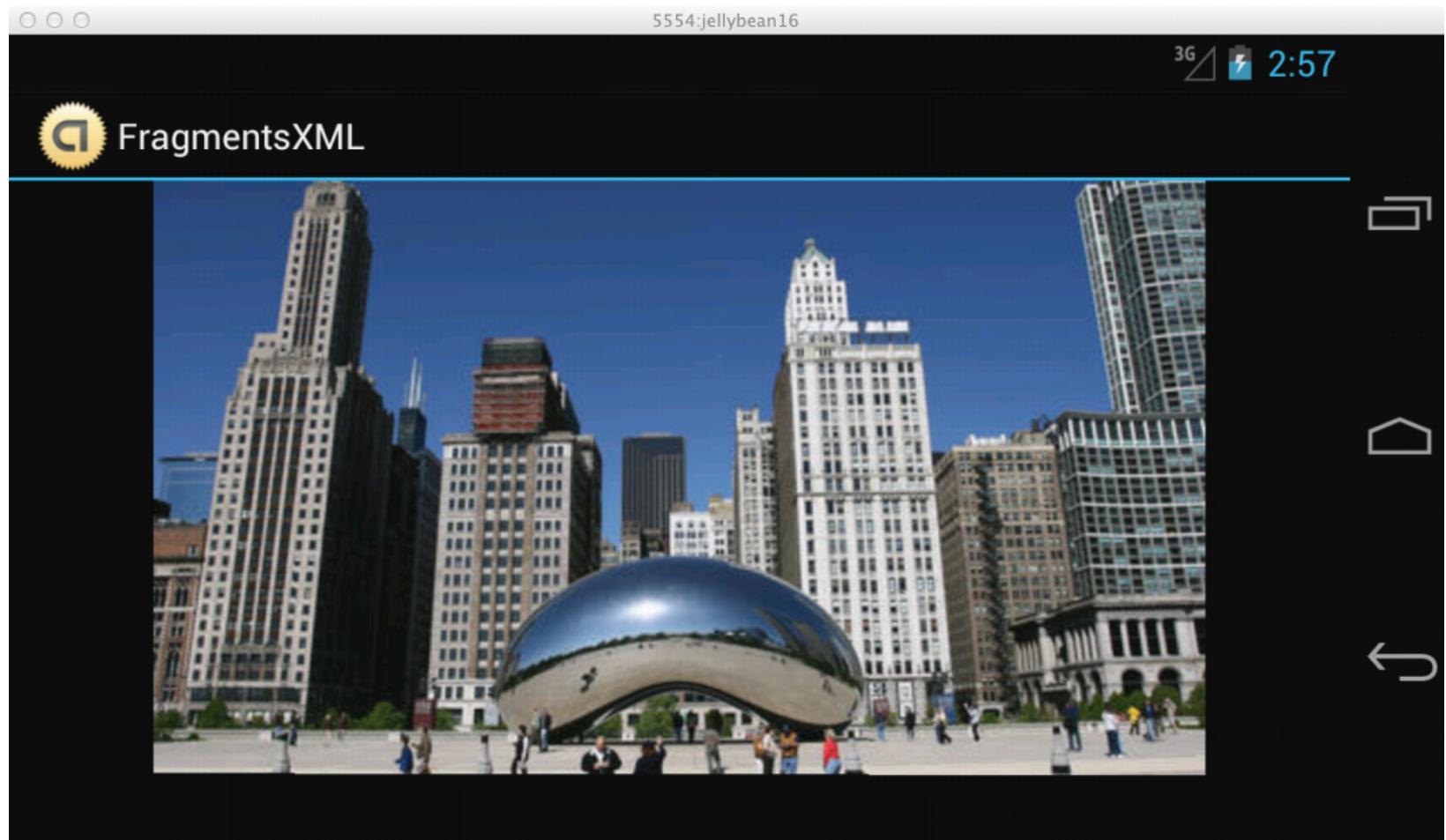
    <fragment android:name="me.almalkawi.FragmentsXML.CityImageFragment"
        android:id="@+id/image_fragment"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"/>

</LinearLayout>
```

Running project (Portrait)



Running project (Landscape)



Demo: Adding/Removing Fragments at runtime

Source: <https://github.com/almalkawi/AndroidClass/tree/master/DynamicFragments>

These are the same as in previous demo:

CityInfoFragment.java

city_image_fragment.xml

CityImageFragment.java

city_image_fragment.xml

main.xml

The screenshot shows the Android Studio code editor with the main.xml file open. The code defines a linear layout containing a fragment and several buttons. A red arrow points from a callout box to the <fragment> element, which is highlighted in yellow.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical">

    <fragment android:name="me.almalkawi.DynamicFragments.CityInfoFragment"
        android:id="@+id/image_fragment"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"/>

    <LinearLayout android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:layout_gravity="center_horizontal">

        <Button android:id="@+id/show_city"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:text="Add fragment"
            android:enabled="true"
            android:onClick="showCity"/>

        <Button android:id="@+id/hide_city"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_gravity="center_horizontal"
            android:text="Del fragment"
            android:enabled="false"
            android:onClick="hideCity"/>
    </LinearLayout>

    <FrameLayout android:id="@+id/image_container"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="center_horizontal"/>
</LinearLayout>
```

Container in which fragment can be placed at runtime

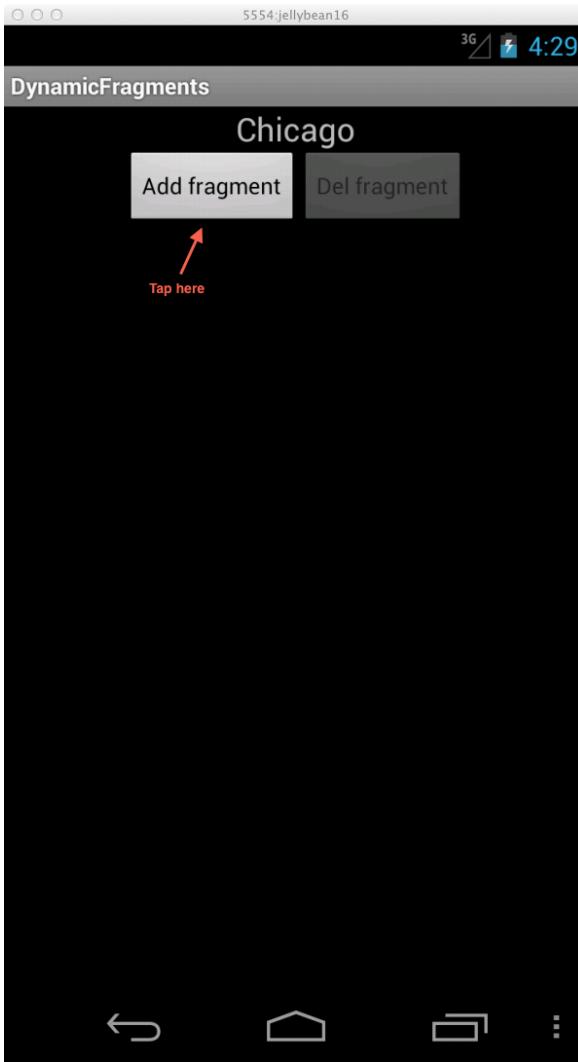
MainActivity.java

The screenshot shows the Android Studio code editor for `MainActivity.java`. The code implements a simple UI where pressing a button shows or hides a fragment. Annotations with arrows explain specific parts of the code:

- An annotation points to the line `setContentView(R.layout.main);` with the text "Fragment manager manages fragments (find, add, remove, replace)".
- An annotation points to the line `fragmentTransaction.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN);` with the text "Animation for adding/removing fragment".
- An annotation points to the line `fragmentTransaction.addToBackStack(null);` with the text "Makes pressing the Back button reverse the transaction".

```
1 package me.almalkawi.DynamicFragments;
2
3 import ...
4
5 public class MainActivity extends Activity {
6     private Button mShowButton;
7     private Button mHideButton;
8
9     @Override
10    public void onCreate(Bundle savedInstanceState) {
11        super.onCreate(savedInstanceState);
12        setContentView(R.layout.main);           Fragment manager manages fragments
13
14        mShowButton = (Button) findViewById(R.id.show_city);
15        mHideButton = (Button) findViewById(R.id.hide_city);
16    }
17
18    public void showCity(View view) {
19        final FragmentManager fragmentManager = getFragmentManager();
20        final FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
21
22        // Add fragment
23        fragmentTransaction.add(R.id.image_container, new CityImageFragment());
24
25        // Apply a transition animation
26        fragmentTransaction.setTransition(FragmentTransaction.TRANSIT_FRAGMENT_OPEN);
27
28        // Add the fragment transaction to the back stack
29        fragmentTransaction.addToBackStack(null);      Animation for adding/
30
31        fragmentTransaction.commit();
32        mShowButton.setEnabled(false);
33        mHideButton.setEnabled(true);
34
35    }
36
37    public void hideCity(View view) {
38        final FragmentManager fragmentManager = getFragmentManager();
39
40        // Remove fragment
41        final Fragment fragment = fragmentManager.findFragmentById(R.id.image_container);
42        if (fragment != null) {
43            final FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();
44            fragmentTransaction.remove(fragment);
45            fragmentTransaction.commit();
46        }
47
48        mHideButton.setEnabled(false);
49        mShowButton.setEnabled(true);
50
51    }
52
53
54
55
56}
```

Running project (initial and after removing fragment)



Running project (after tapping "Add fragment")



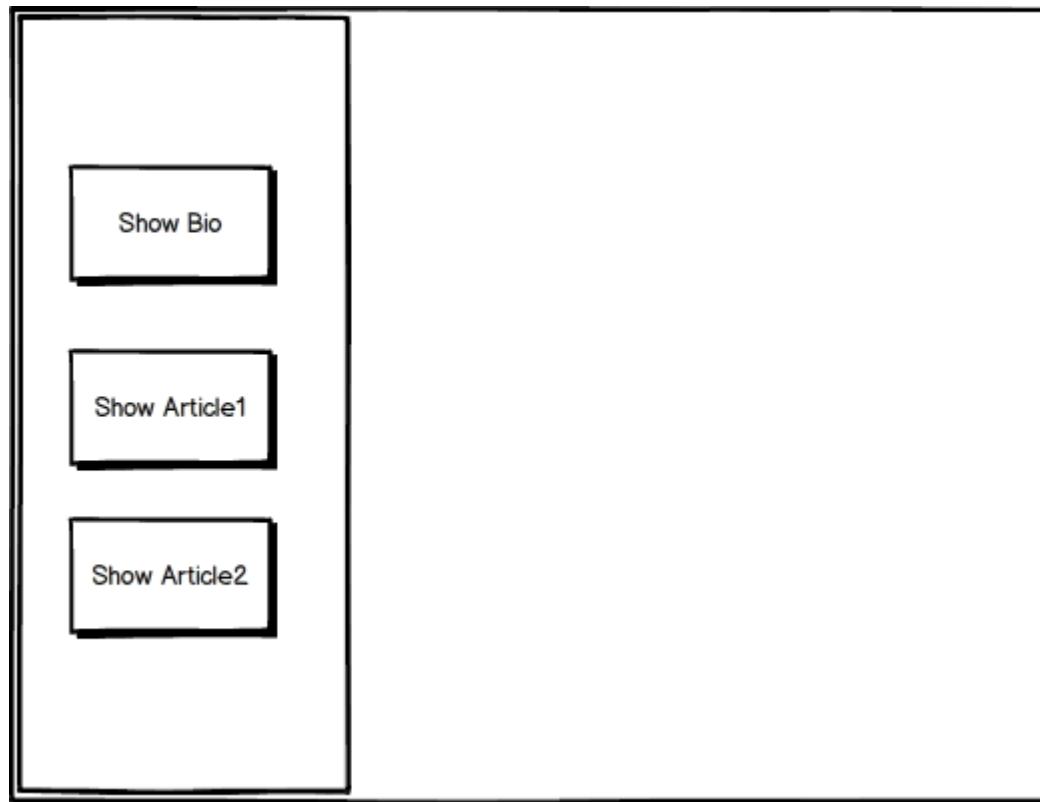
Hello Android!

Fragments Exercise

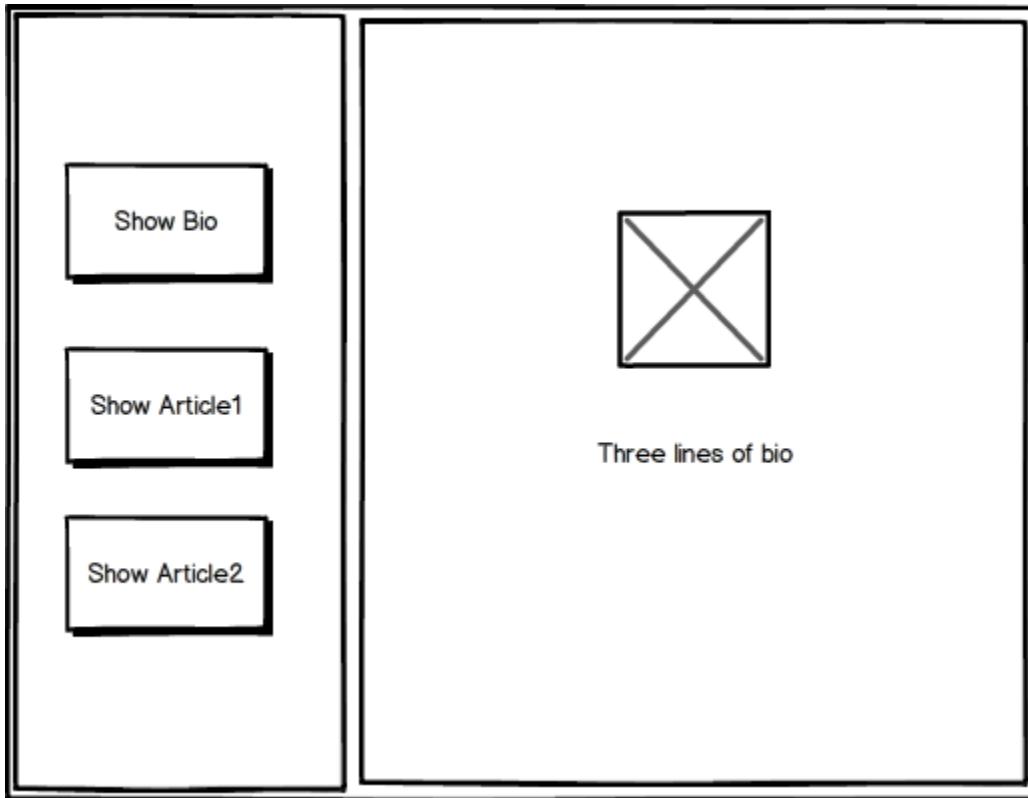
Exercise: Reader App

- Use one static Fragment for left pane
- Use one dynamic Fragment for bio
- Use one dynamic Fragment for article
- Mockups are in next slides

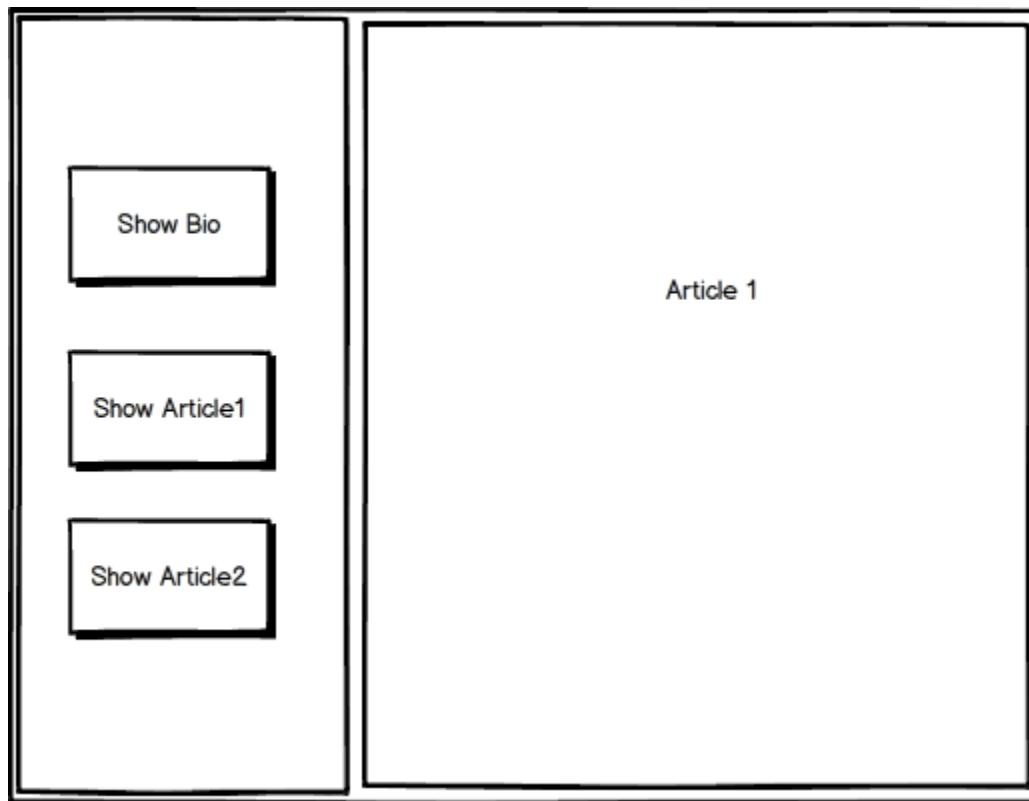
Exercise: Initial State



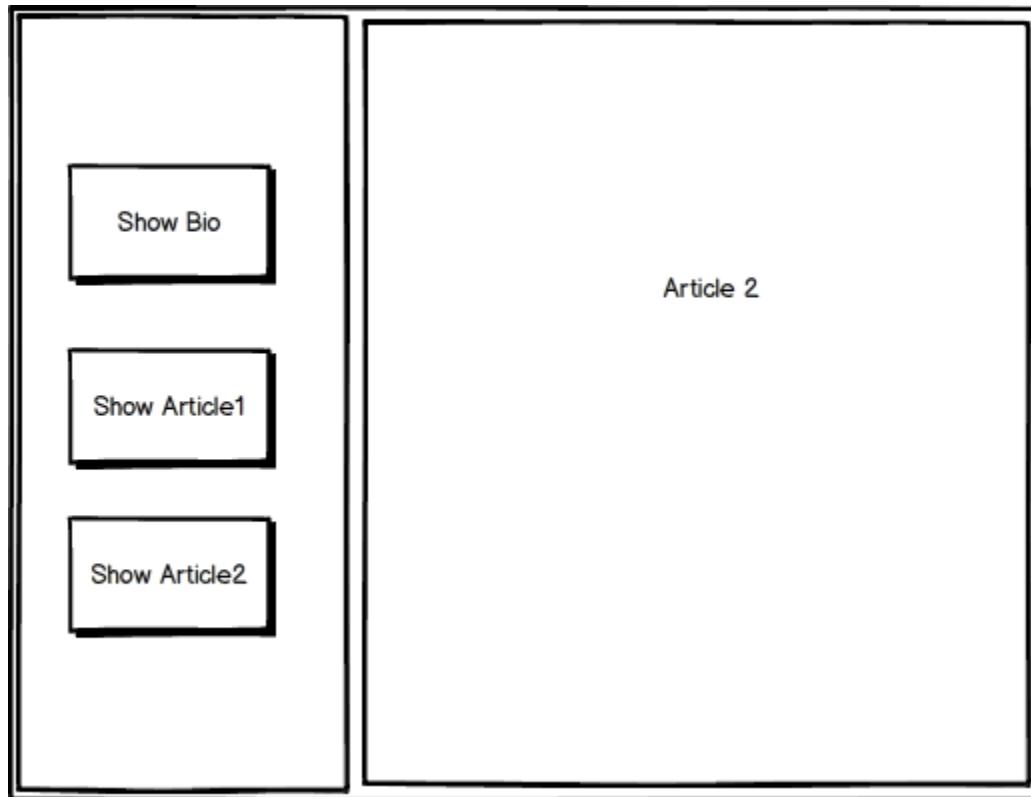
Exercise: Showing Bio Fragment



Exercise: Showing Article1



Exercise: Showing Article2



Hello Android!

Data Persistence

SQLite tools

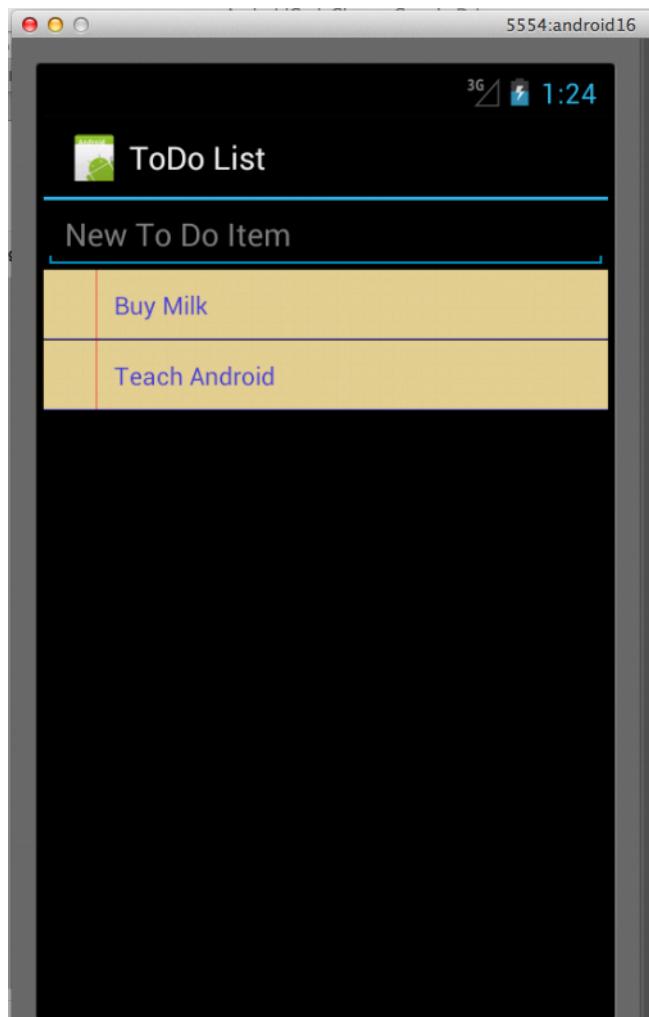
Command line:

```
> sqlite3
```

GUI:

- SQLite Manager Firefox plugin *free*
- Base 2 (<http://menial.co.uk/base/>) *favorite*
- Navicat for SQLite

Enhanced TODO App



Demo

GitHub Repo: <https://github.com/almalkawi/AndroidClass.git>

Project: PersistedToDoList
[\(https://github.com/almalkawi/AndroidClass/tree/master/PersistedToDoList\)](https://github.com/almalkawi/AndroidClass/tree/master/PersistedToDoList)

Pull DB from phone

- App data: /data/data/com.example.todolist/databases
- Copy DB to /sdcard
- adb pull the db
- Open DB using Base2

Demo

GitHub Repo: <https://github.com/almalkawi/AndroidClass.git>

Project: ContactsList
(<https://github.com/almalkawi/AndroidClass/tree/master/ContactsList>)

Hello Android!

Data Persistence Exercise

Exercise

- Add Checkbox next to every item in ToDo App
- Add handler to the checkboxes to remove checked items from the DB and ListView once checkbox is checked.