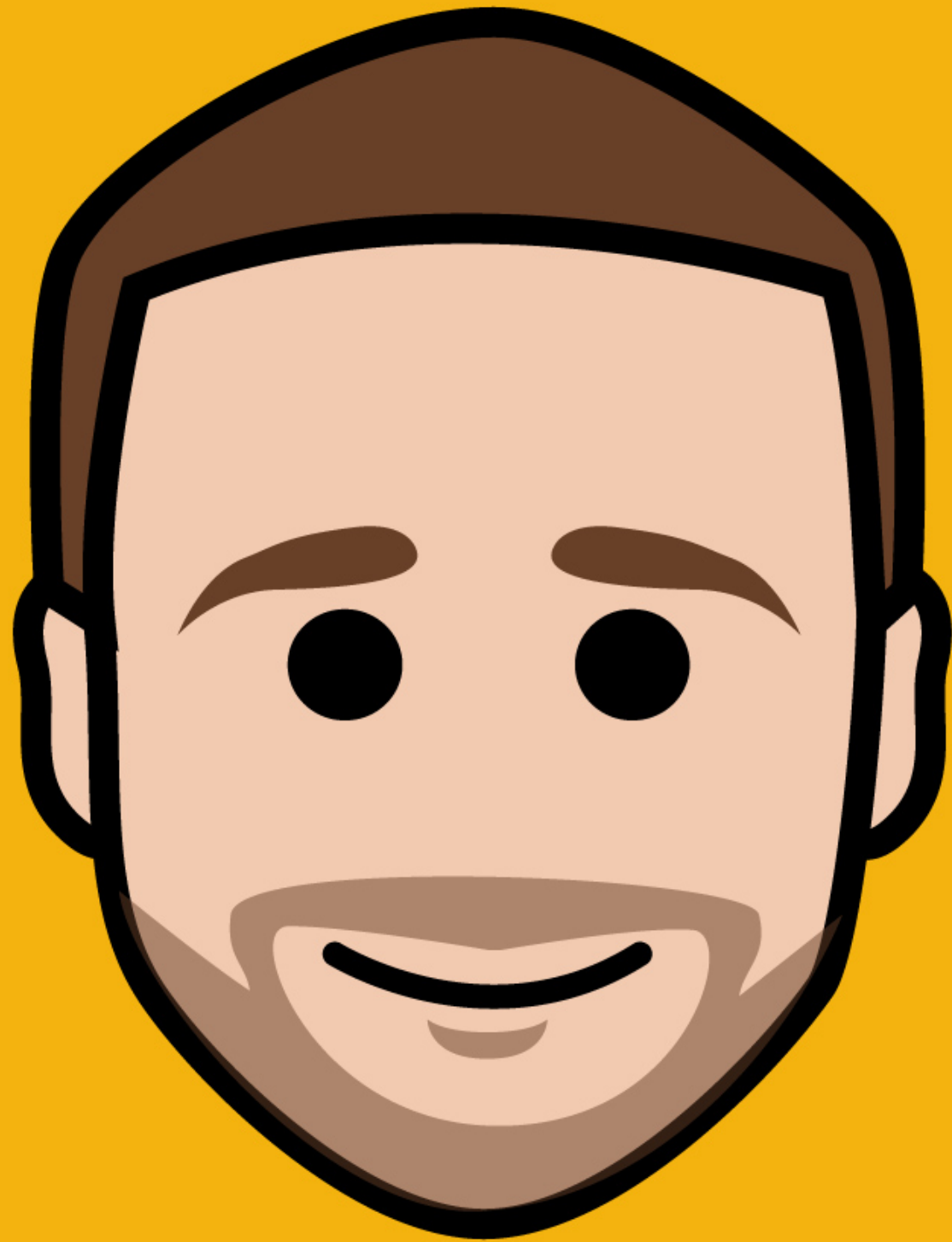


Android 101 for iOS Developers

Stephen Barnes - @smbarne

[360|iDev Min]



Who is this guy?

- Previously Senior Mobile Dev @Raizlabs, Now Senior iOS Dev @Fitbit
- Previous work includes Care.com, B&H Photography, and others for both Android and iOS
- Twitter: [@smbarne](#)
- Github: [github.com/smbarne](#)
- Misc: [engineeringart.io](#)

Overview

- A Word on UI Design
- Application Structure and Language
- Android Myths
- Building Blocks: Activities, Fragments, ListViews, ViewPagers, and more
- Android Lifecycle
- Android Layouts

Assumptions

- Working knowledge of Java
- Able to install and use the [Android Development Tools](#)
- You have written a mobile app before

Recommendations

Read through the [Building your first app](#) tutorial by Google when possible.

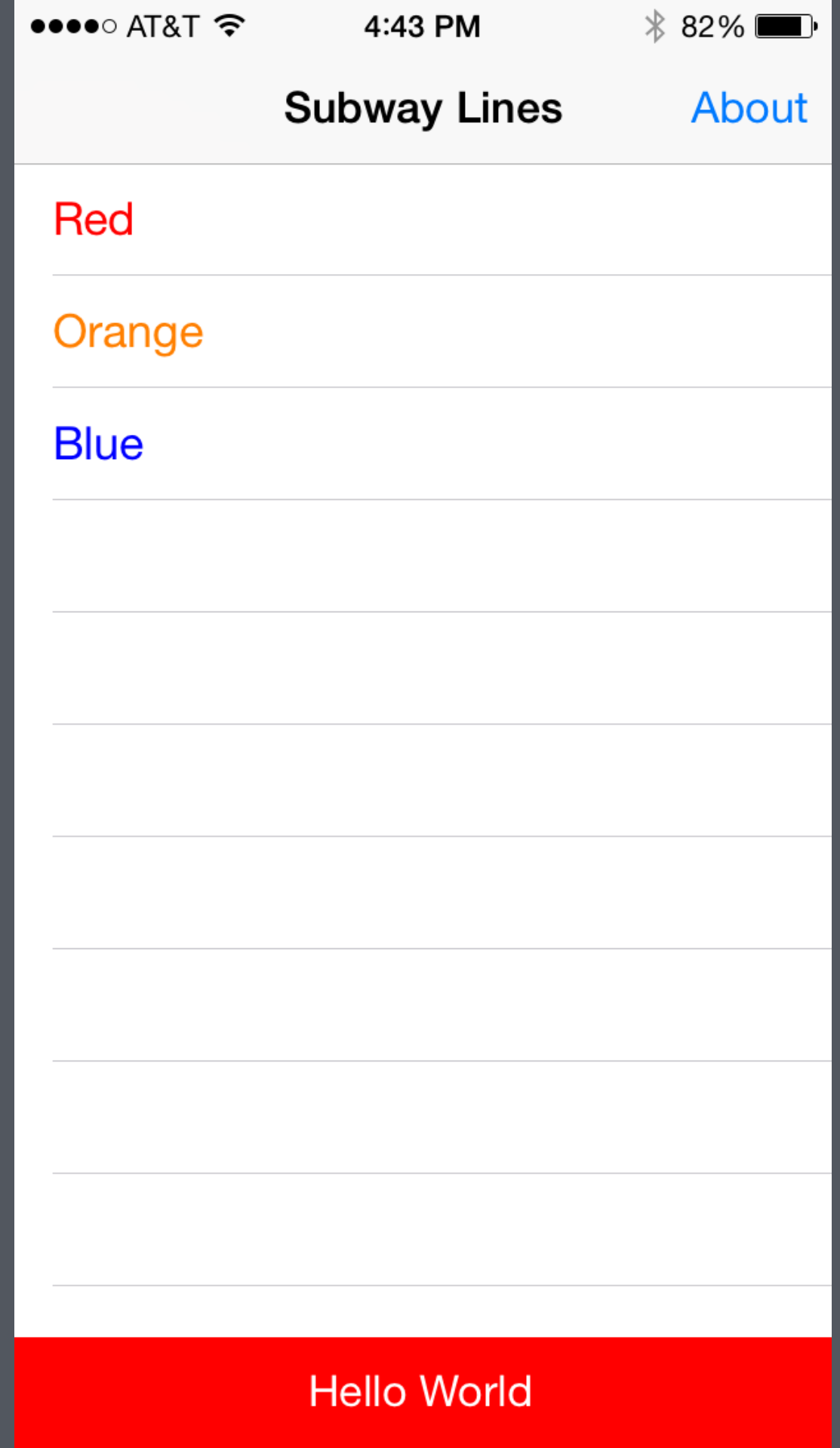
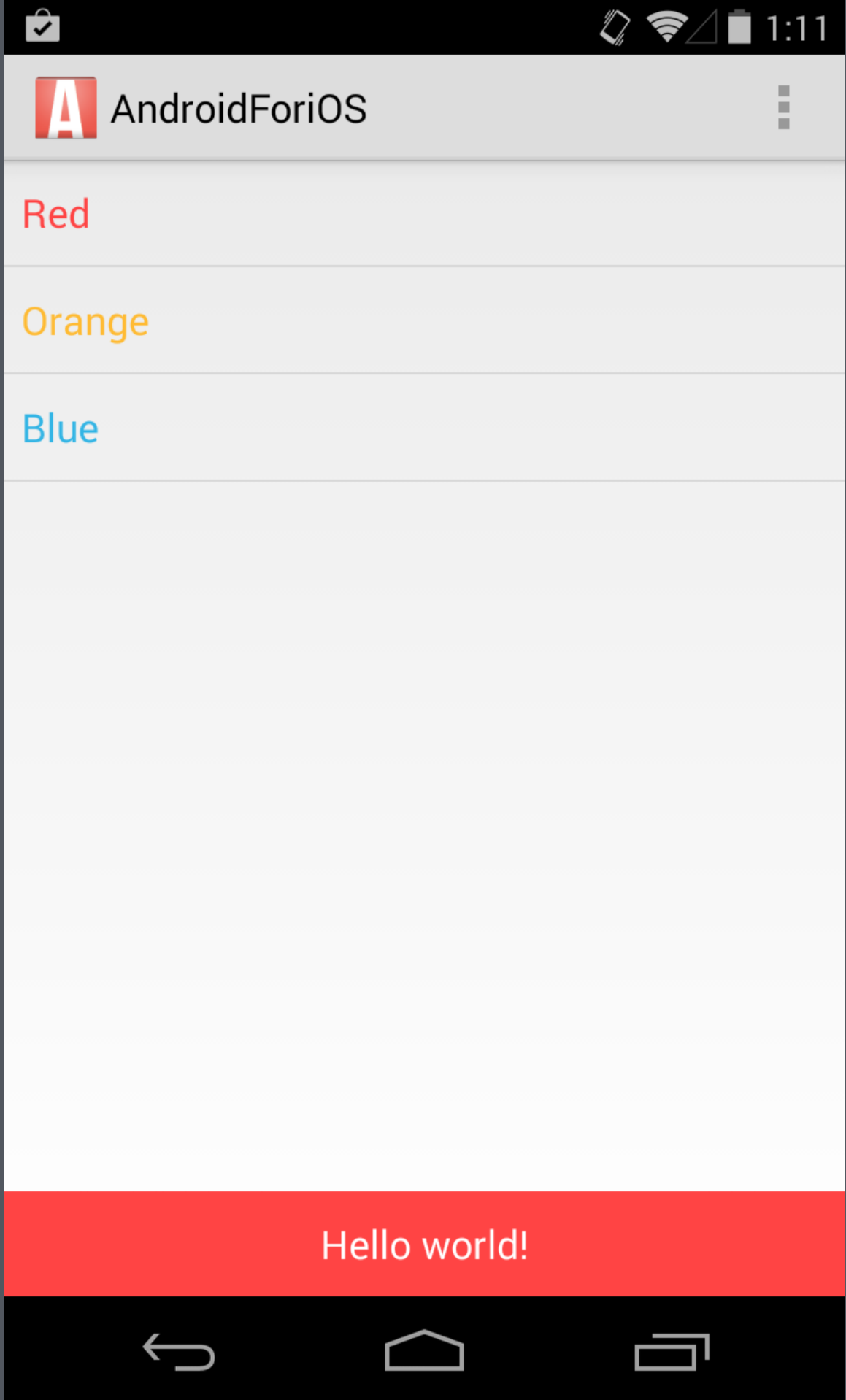
Resources

Objc.io article: [objc.io/issue-11/
android_101_for_ios_developers.html](http://objc.io/issue-11/android_101_for_ios_developers.html)

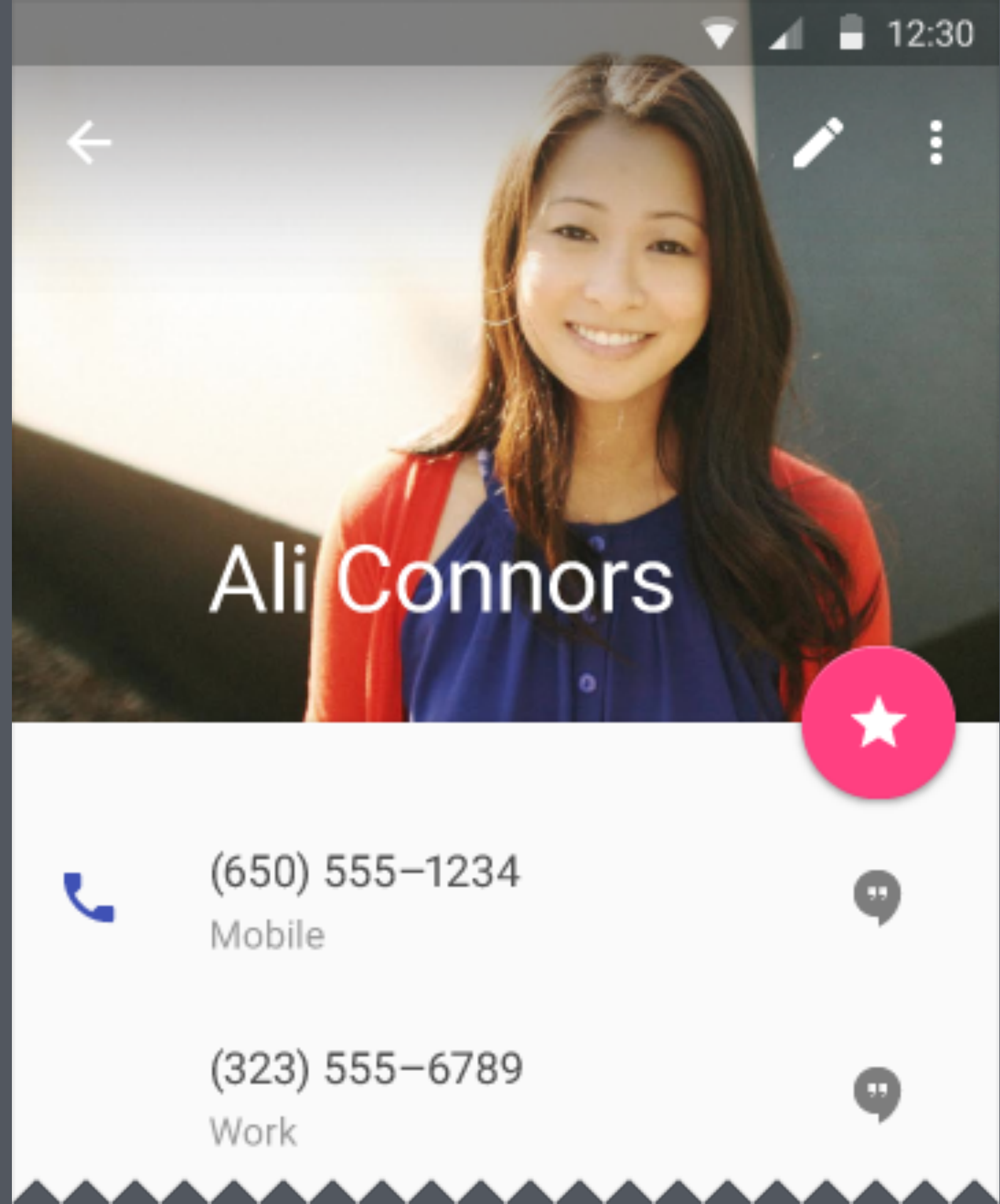
Code: github.com/smbarne/AndroidForiOS



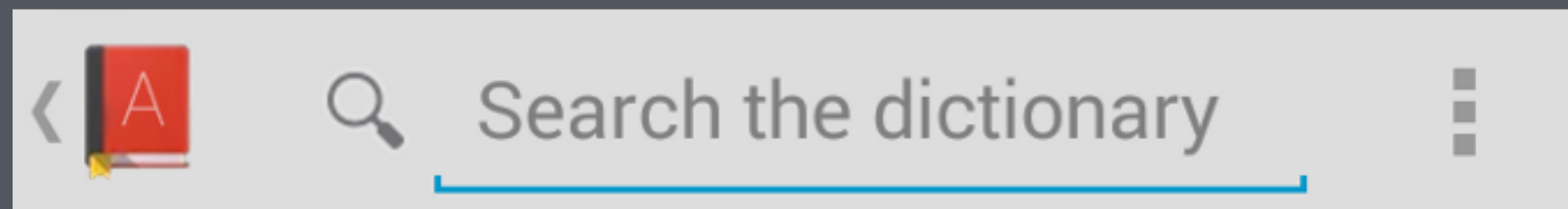
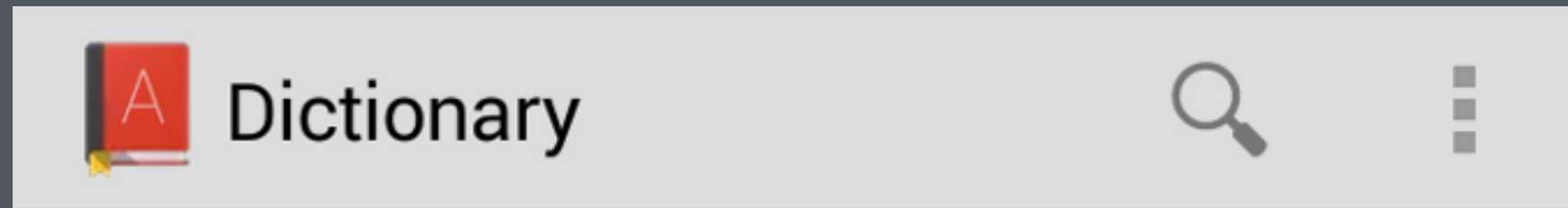
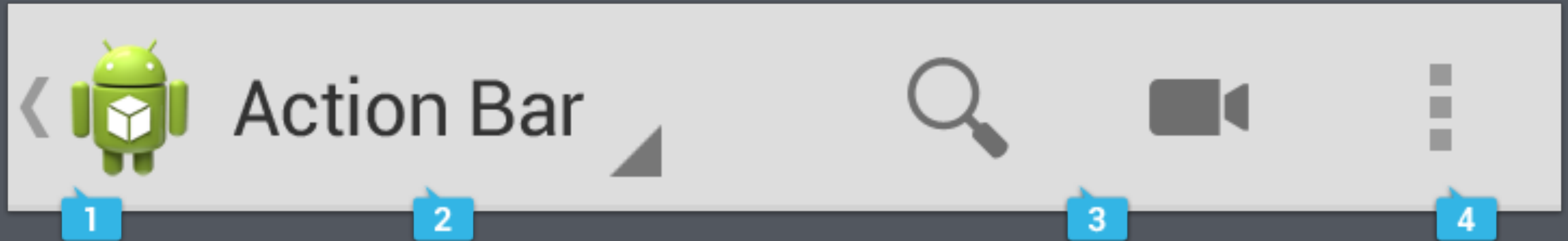
A Word on UI Design



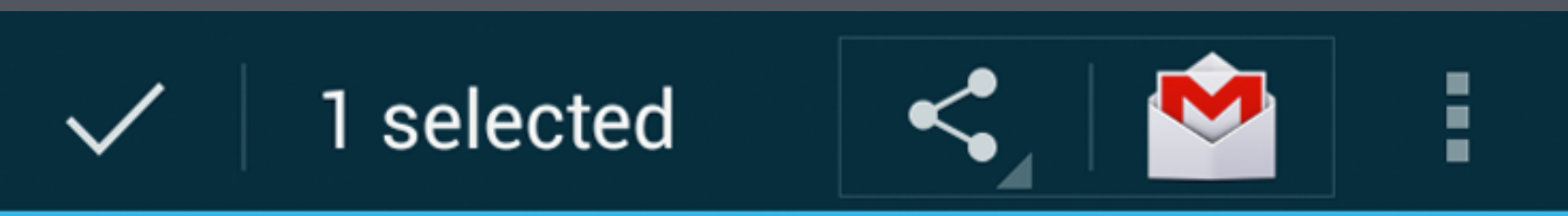
Material Design



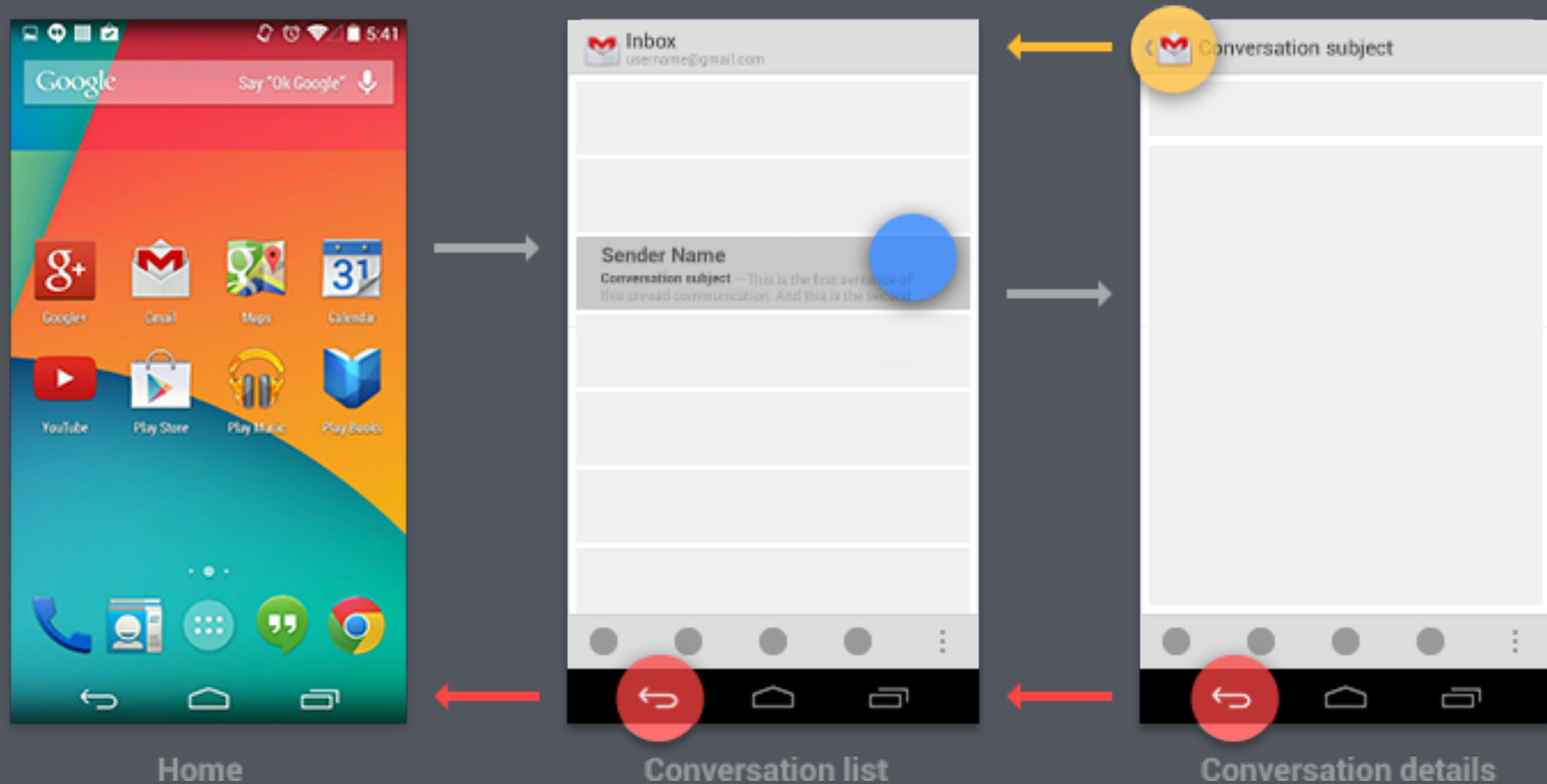
Actionbar



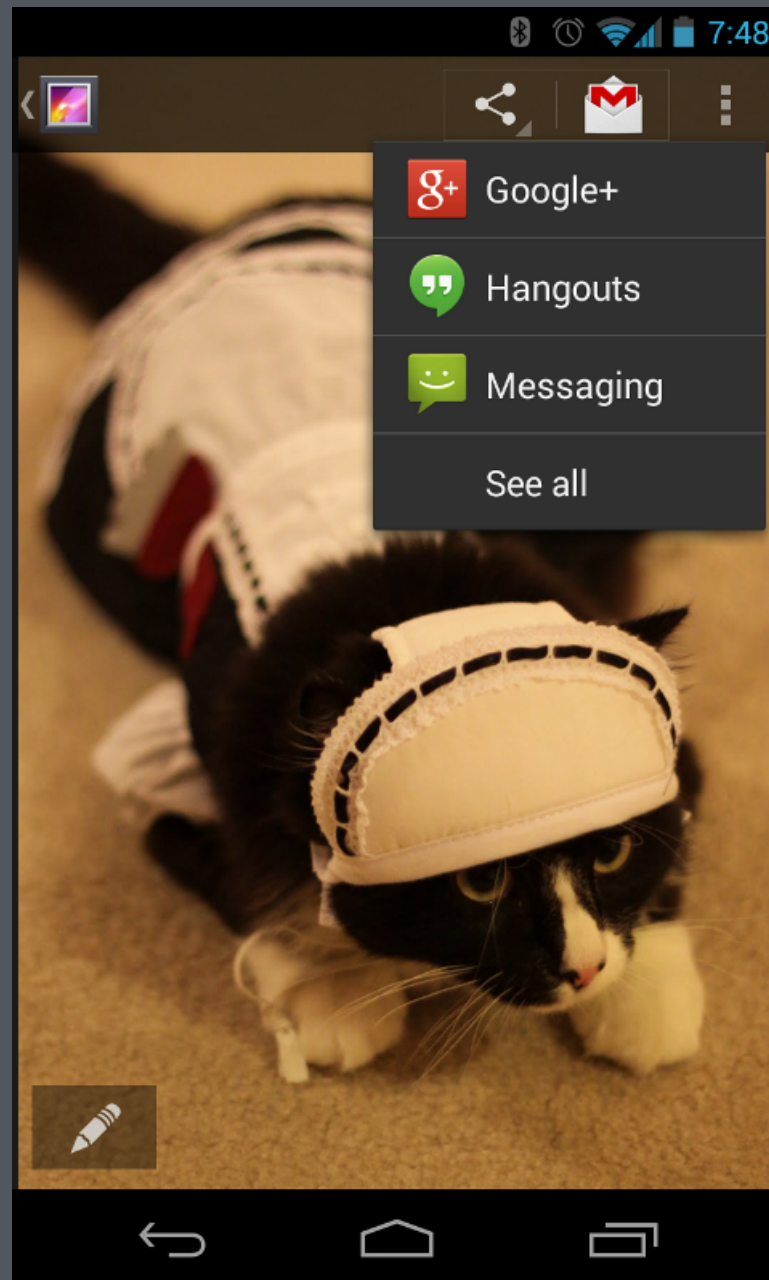
Actionbar Context



Back Button and Navigation

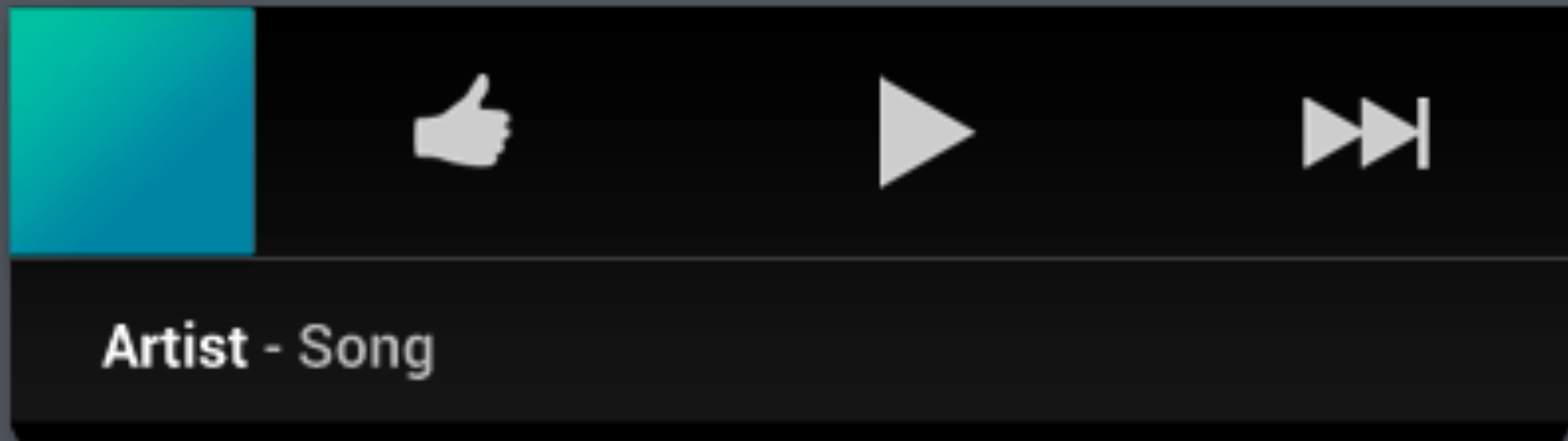


Overflow Menu



Widgets

Now on iOS 8!



Others

- Phone vs tablet¹
- Rotation handling
- Aspect ratios¹
- No overscroll

¹ iOS 8 now brings iOS to a similar architecture









































Code Styling and Patterns

- Leave class prefixes at home on Objective-C.
- Instance variables are prefixed with m, not _.
- Take advantage of JavaDoc **please**.
- Null check! Objective-C handles gracefully handles message sending to nil objects gracefully, but Java does not.
- Say goodbye to properties. You have to remember to call `getVariableName()` and `setVariableName()`.

Misc

- Goodbye provisioning!
- So long review cycle!

Project Structure

- ▼  src
 - ▼  com
 - ▼  costco.app.android
 - ▶  findastore
 - ▶  homenav
 - ▶  offers
 - ▶  onboarding
 - ▶  warehousedetails
 - ▶  warehouses
 - ▶  widget
 -  APIFormatUtils
 -  BaseActivity
 -  BaseFragment
 -  Constants
 -  CostcoApplication
 -  DateRange
 -  GeneralPreferences
 -  IntentUtils
 -  IViewHolder
 -  LatLng
 -  LaunchActivity
 -  LocalizedString
 -  MainActivity
 -  MinuteUpdateViewHelper
 -  TimeManager
 -  TimeRange
 -  TimeUtils
 - ▼  raizlabs
 - ▶  fragments
 - ▶  widget
 -  BasePreferencesManager
 -  LooperThread
 -  AndroidManifest.xml
 -  build.gradle
 -  Costco.iml
 -  gradlew
 -  gradlew.bat
 -  ic_launcher-web.png
 -  proguard-project.txt
 -  project.properties

- Folder Structure based on package naming
- **AndroidManifest.xml** is required and similar to the **info.plist** on iOS
- **build.gradle** for each project

Resources (res folder)



Drawable

Drawables are images and any other renderable objects that you can define (think *XML* defined *CAShape*)

No more **@2x** or **@3x** - now you get many, many buckets!

drawable-mdpi, drawable-hdpi, drawable-xhdpi, drawable-xxhdpi, etc

Drawables are typically Images

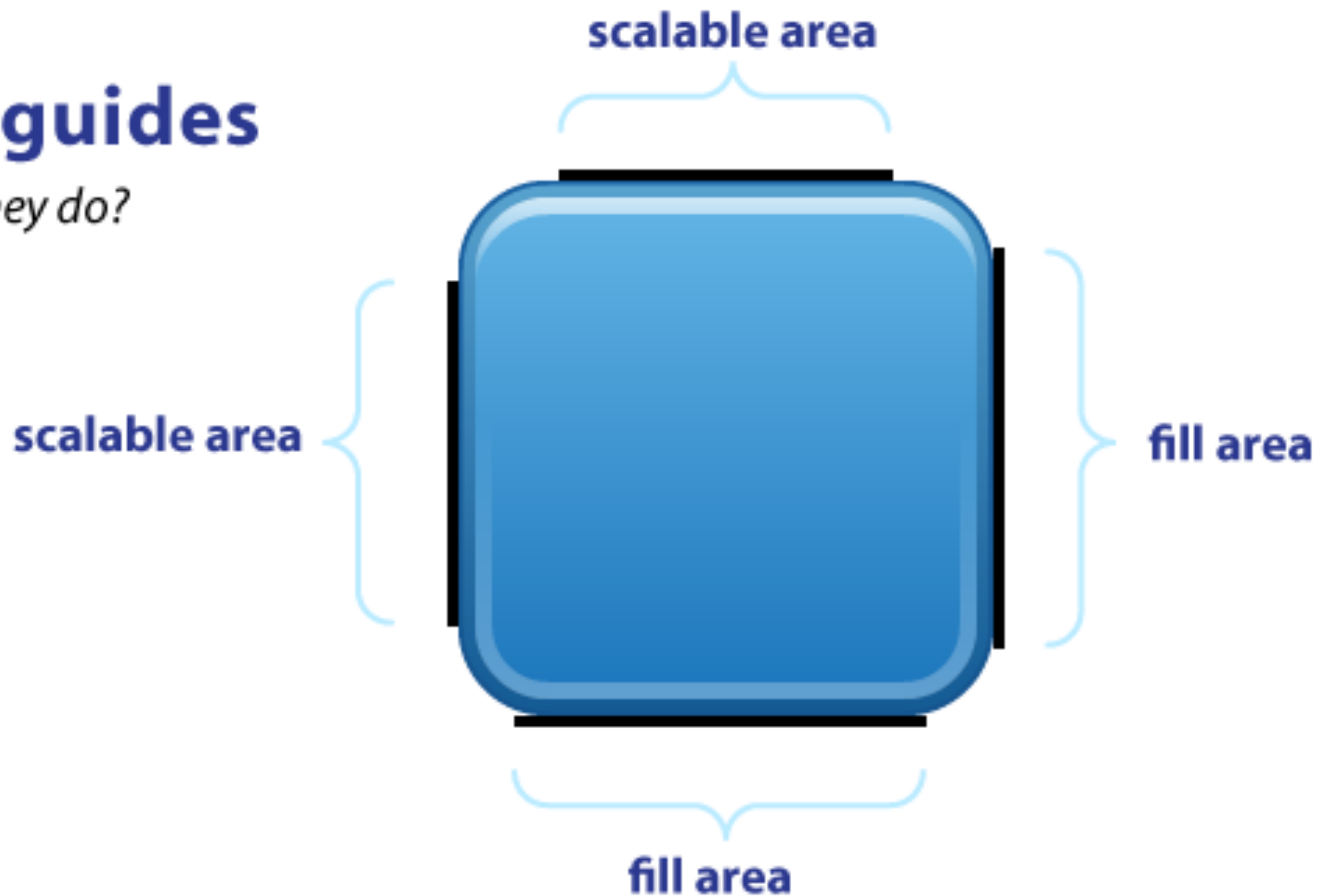
DPI (Dots Per Inch, or Density Independent Pixel)

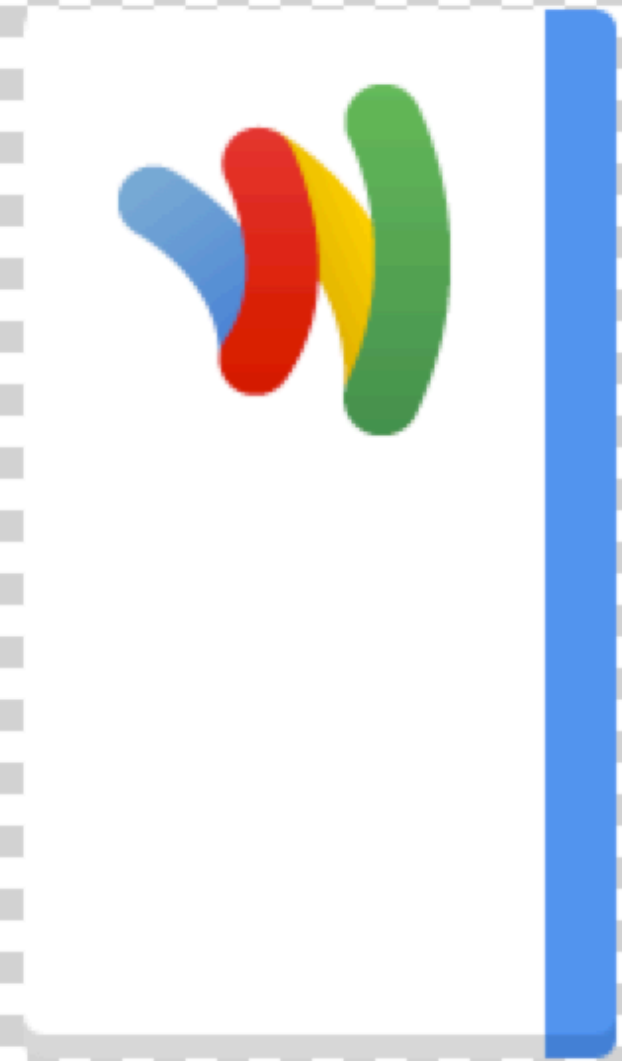
- Each device has an internal DPI bucket
- Android picks the resource from the corresponding bucket or the closet one and scales it

9-Patches (Stretchable Images)

9-patch guides

what do they do?





Dimens and Strings

```
<dimen name="Margin.Standard">5dp</dimen>
```

```
<string name="LocationServices.DisabledPrompt.Title">Location services disabled</string>
```

Styles

```
<style name="Button" parent="@style/TextStyle.Standard">  
    <item name="android:textSize">@dimen/TextSize.Large</item>  
    <item name="android:background">@color/Panel.White.Transparent</item>  
    <item name="android:minWidth">@dimen/MinTouchableSize</item>  
    <item name="android:minHeight">@dimen/MinTouchableSize</item>  
</style>
```

You can have different values for different API levels!

Tools

Build Tools

- Gradle
- Ant

IDEs

- Android Studio (*built by IntelliJ*)
- Eclipse

Android Myths

Fragmentation

Use Google's Support Library

Android Apps Crash More

Not True

Android Users Don't Use Their Devices As Much

Somewhat. The Android userbase is **huge. Really** huge. Some users don't use their devices much, but some use them a lot. The user segment you target means a lot.

Building Blocks

Activities, Fragments, ListViews, ViewPagers,
and more

Activities <-> UIViewControllers

Activities are the basic unit of an Android app.

An application usually consists of multiple activities that are loosely bound to each other.

One activity in an application is specified as the "main" activity.

Activities can start and return information

- Activities can register that they handle common data such as images
- Activities can also receive and send specific data, such as an item ID

Example: An Activity Launching
Another Activity and Responding
When It Is Done

```
protected void startNextActivity() {
    Intent nextActivityIntent = NextActivity.getIntent(this);
    startActivityResult(nextActivityResult, REQUEST_CODE_NEXT_ACTIVITY);
}

@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
    switch (requestCode) {
        case REQUEST_CODE_NEXT_ACTIVITY:
            if (resultCode == RESULT_OK) {
                Toast.makeText(this, "Result OK!", Toast.LENGTH_SHORT).show();
                // We can also do something with returnObject within data here
            }
            return;
        }
    super.onActivityResult(requestCode, resultCode, data);
}
```

Next Activity Finishing with Data

```
public static final String activityResultString = "activityResultString";

/*
 * On completion, place the object ID in the intent and finish with OK.
 * @param returnObject that was processed
 */
private void onActivityResult(Object returnObject) {
    Intent data = new Intent();
    if (returnObject != null) {
        data.putExtra(activityResultString, returnObject.uniqueId);
    }

    setResult(RESULT_OK, data);
    finish();
}
```

Fragments

Mini controllers that can be
instantiated to populate activities

Fragments are the "new", accepted way to
build Android apps²

² And they're awesome

There can be **one** or **multiple** fragments within a single activity.

Fragments:

- Store state
- Contain view logic
- Do not have a context, the activity has the context
- Must be tied to an activity

A

Albert Gianni



Alex Crook



Alex Diebenkorn



Alexander Jones



Alina Anderson



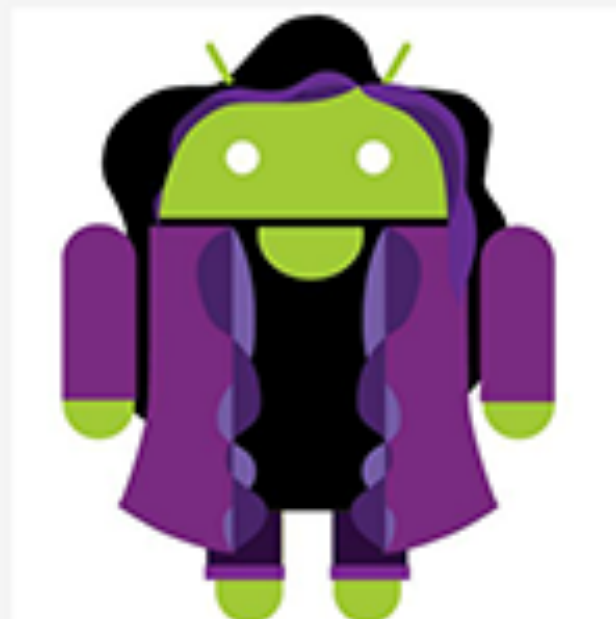
Amy Rihanna



Andrew Faustner



Andy Wagner



Amy Rihanna



I <3 Android

PHONE

+1 650-222-1234

MOBILE

EMAIL

amy@android.com

OTHER

WEBSITE

<http://www.google.com>

CONNECTIONS

+ Add connection

ADDRESS

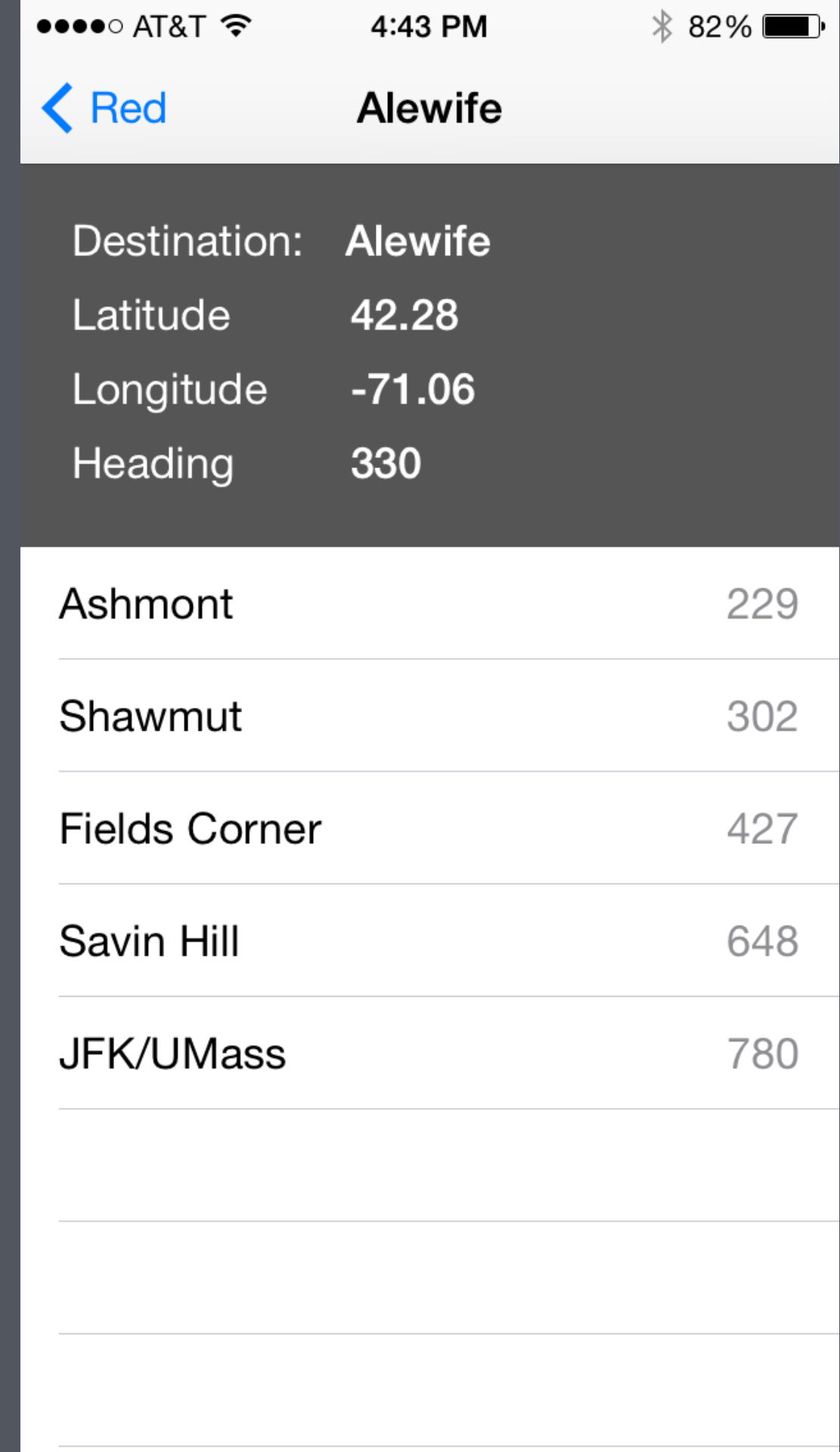
1600 Amphitheatre Parkway
Mountain View, CA 94043

The closest approximation for fragments in iOS is using child view controllers.

Explain it to me with code

Let's look at a sample **UITableViewController** and a sample **ListFragment** that show a list of prediction times for a subway trip curtesy of the [MBTA](#) (Massachusetts Bay Transportation Authority).

iOS Tableview Implementation



```
@interface MBTASubwayTripTableViewController ()
@property (assign, nonatomic) MBTATrip *trip;
@end

@implementation MBTASubwayTripTableViewController
- (instancetype)initWithTrip:(MBTATrip *)trip
{
    self = [super initWithStyle:UITableViewStylePlain];
    if (self) {
        _trip = trip;
        [self setTitle:trip.destination];
    }
    return self;
}

- (void)viewDidLoad
{
    [super viewDidLoad];

    [self.tableView registerClass:[MBTAPredictionCell class]
        forCellReuseIdentifier:[MBTAPredictionCell reuseId]];
    [self.tableView
        registerNib:[UINib nibWithNibName:NSStringFromClass([MBTATripHeaderView class]) bundle:nil]
        forCellReuseIdentifier:[MBTATripHeaderView reuseId]];
}
```

```
#pragma mark - Table view data source
```

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView {  
    return 1;  
}
```

```
- (NSInteger)tableView:(UITableView *)tableView numberOfRowsInSection:(NSInteger)section {  
    return [self.trip.predictions count];  
}
```

```
#pragma mark - UITableViewDelegate
- (CGFloat)tableView:(UITableView *)tableView heightForHeaderInSection:(NSInteger)section {
    return [MBTATripHeaderView heightWithTrip:self.trip];
}

- (UIView *)tableView:(UITableView *)tableView viewForHeaderInSection:(NSInteger)section {
    MBTATripHeaderView *headerView = [self.tableView
        dequeueReusableHeaderFooterViewWithIdentifier:[MBTATripHeaderView reuseIdentifier]];
    [headerView setFromTrip:self.trip];
    return headerView;
}

- (UITableViewCell *)tableView:(UITableView *)tableView cellForRowAtIndexPath:(NSIndexPath *)indexPath {
    UITableViewCell *cell = [tableView
        dequeueReusableCellWithIdentifier:[MBTAPredictionCell reuseIdentifier]
        forIndexPath:indexPath];

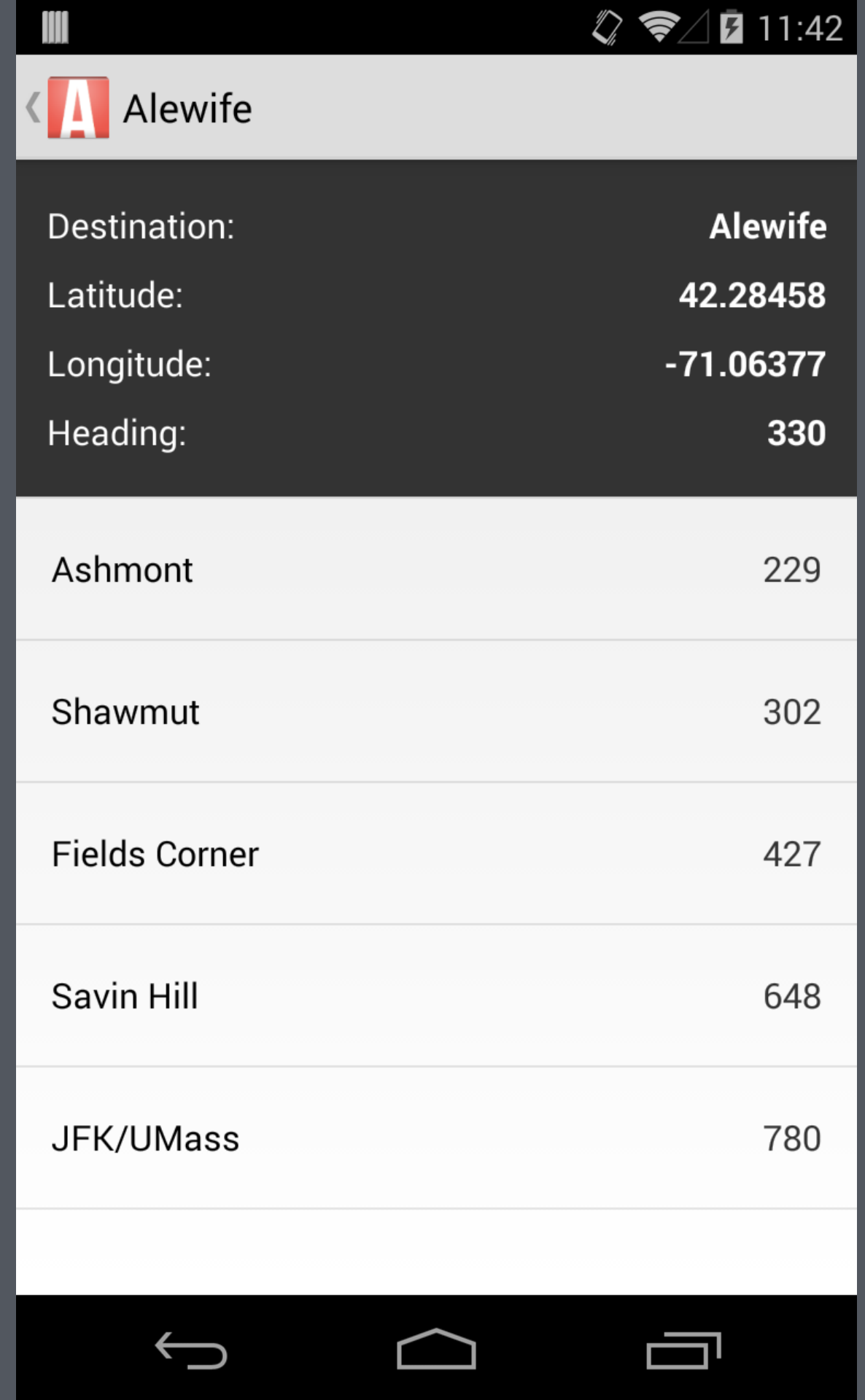
    MBTAPrediction *prediction = [self.trip.predictions objectAtIndex:indexPath.row];
    [(MBTAPredictionCell *)cell setFromPrediction:prediction];

    return cell;
}

- (BOOL)tableView:(UITableView *)tableView canEditRowAtIndexPath:(NSIndexPath *)indexPath {
    return NO;
}

- (void)tableView:(UITableView *)tableView didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    [tableView deselectRowAtIndexPath:indexPath animated:YES];
}
@end
```


ListFragment Implementation



```
public class TripDetailFragment extends ListFragment {
    /**
     * The configuration flags for the Trip Detail Fragment.
     */
    public static final class TripDetailFragmentState {
        public static final String KEY_FRAGMENT_TRIP_DETAIL = "KEY_FRAGMENT_TRIP_DETAIL";
    }

    protected Trip mTrip;

    /**
     * Use this factory method to create a new instance of
     * this fragment using the provided parameters.
     *
     * @param trip the trip to show details
     * @return A new instance of fragment TripDetailFragment.
     */
    public static TripDetailFragment newInstance(Trip trip) {
        TripDetailFragment fragment = new TripDetailFragment();
        Bundle args = new Bundle();
        args.putParcelable(TripDetailFragmentState.KEY_FRAGMENT_TRIP_DETAIL, trip);
        fragment.setArguments(args);
        return fragment;
    }

    public TripDetailFragment() { }
}
```

```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
                          Bundle savedInstanceState) {
    Prediction[] predictions =
        mTrip.predictions.toArray(new Prediction[mTrip.predictions.size()]);

    PredictionArrayAdapter predictionArrayAdapter =
        new PredictionArrayAdapter(getActivity().getApplicationContext(), predictions);

    setListAdapter(predictionArrayAdapter);
    return super.onCreateView(inflater, container, savedInstanceState);
}
```

```
@Override
public void onViewCreated(View view, Bundle savedInstanceState) {
    super.onViewCreated(view, savedInstanceState);
    TripDetailsView headerView = new TripDetailsView(getActivity());
    headerView.updateFromTripObject(mTrip);
    getListView().addHeaderView(headerView);
}
}
```

WAT?

Let's break down some of that piece by piece...

Listviews and Adapters

UITableView, meet ListView

Both are structured around showing a linear list of Views smoothly

- Android doesn't have *cells* - instead any view can be used
- Reuse your views in a ListView! *This can be even more important to performance than on iOS*
- Default ListView views are available that you can populate just like iOS has default UITableViewCell

Android's **GridView** is similar to **UICollectionView**

Listviews are populated via **Adapters**

Goodbye datasources, hello adapters

Instead of having a datasource delegate, Android has **Adapters**

- An Adapter object acts as a bridge between an AdapterView and the underlying data for that view.
- The Adapter provides access to the data items.
- The Adapter is also responsible for making a View for each item in the data set.

```
public class PredictionArrayAdapter extends ArrayAdapter<Prediction> {  
    int LAYOUT_RESOURCE_ID = R.layout.view_three_item_list_view;  
  
    public PredictionArrayAdapter(Context context) {  
        super(context, R.layout.view_three_item_list_view);  
    }  
  
    public PredictionArrayAdapter(Context context, Prediction[] objects) {  
        super(context, R.layout.view_three_item_list_view, objects);  
    }  
}
```

```
@Override
public View getView(int position, View convertView, ViewGroup parent) {
    Prediction prediction = this.getItem(position);
    View inflatedView = convertView;
    if(convertView==null)
    {
        LayoutInflater inflater = (LayoutInflater)getContext()
            .getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        inflatedView = inflater.inflate(LAYOUT_RESOURCE_ID, parent, false);
    }

    TextView stopNameTextView = (TextView)inflatedView
        .findViewById(R.id.view_three_item_list_view_left_text_view);
    TextView middleTextView = (TextView)inflatedView
        .findViewById(R.id.view_three_item_list_view_middle_text_view);
    TextView stopSecondsTextView = (TextView)inflatedView
        .findViewById(R.id.view_three_item_list_view_right_text_view);

    stopNameTextView.setText(prediction.stopName);
    middleTextView.setText("");
    stopSecondsTextView.setText(prediction.stopSeconds.toString());

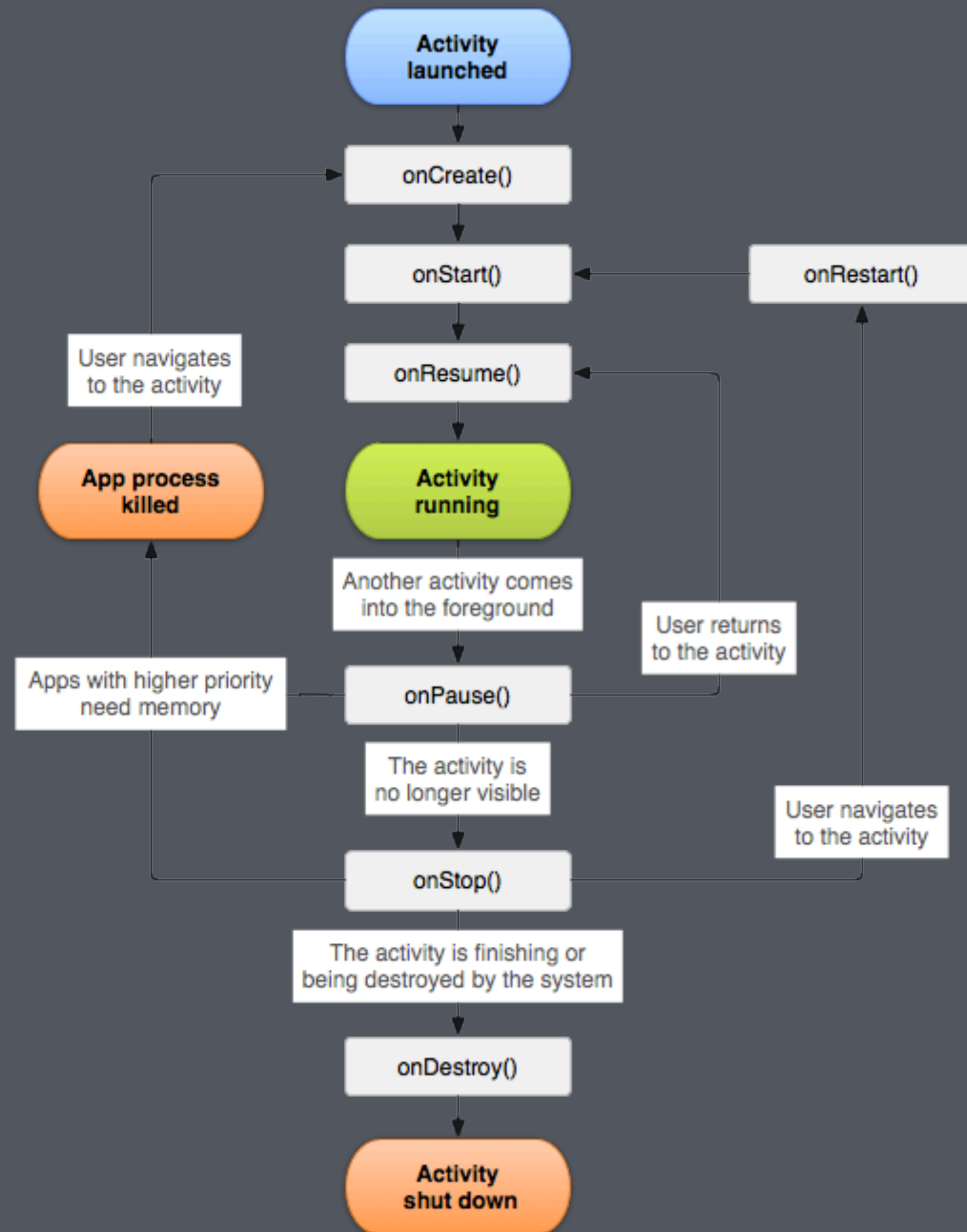
    return inflatedView;
}
}
```

AsyncTasks

```
private class DownloadFilesTask extends AsyncTask<URL, Integer, Long> {
    protected Long doInBackground(URL... urls) {
        int count = urls.length;
        long totalSize = 0;
        for (int i = 0; i < count; i++) {
            totalSize += Downloader.downloadFile(urls[i]);
            publishProgress((int)((i '/' (float) count) * 100));
            // Escape early if cancel() is called
            if (isCancelled()) {
                break;
            }
        }
        return totalSize;
    }

    protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }

    protected void onPostExecute(Long result) {
        showDialog("Downloaded " + result + " bytes");
    }
}
```

Activities must save their state

- Place information into the `savedInstanceState` that you wish to keep and restore it on *onCreate()*
- Activities can be **Destroyed** on rotation
 - Dont turn off rotation - it will just hide edge case lifecycle bugs if you do
- *Activities* and *Fragments* can be loosely coupled. On creation, look for initialized *Fragments* before creating them.

Example: Storing Activity Data

```
public static Intent getTripListActivityIntent(Context context,
                                              TripList.LineType lineType) {
    Intent intent = new Intent(context, TripListActivity.class);
    intent.putExtra(TripListActivityState.KEY_ACTIVITY_TRIP_LIST_LINE_TYPE,
                  lineType.getLineName());
    return intent;
}

public static final class TripListActivityState {
    public static final String KEY_ACTIVITY_TRIP_LIST_LINE_TYPE =
        "KEY_ACTIVITY_TRIP_LIST_LINE_TYPE";
}

TripList.LineType mLineType;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    mLineType = TripList.LineType.getLineType(
        getIntent().getStringExtra(TripListActivityState.KEY_ACTIVITY_TRIP_LIST_LINE_TYPE));
}
```

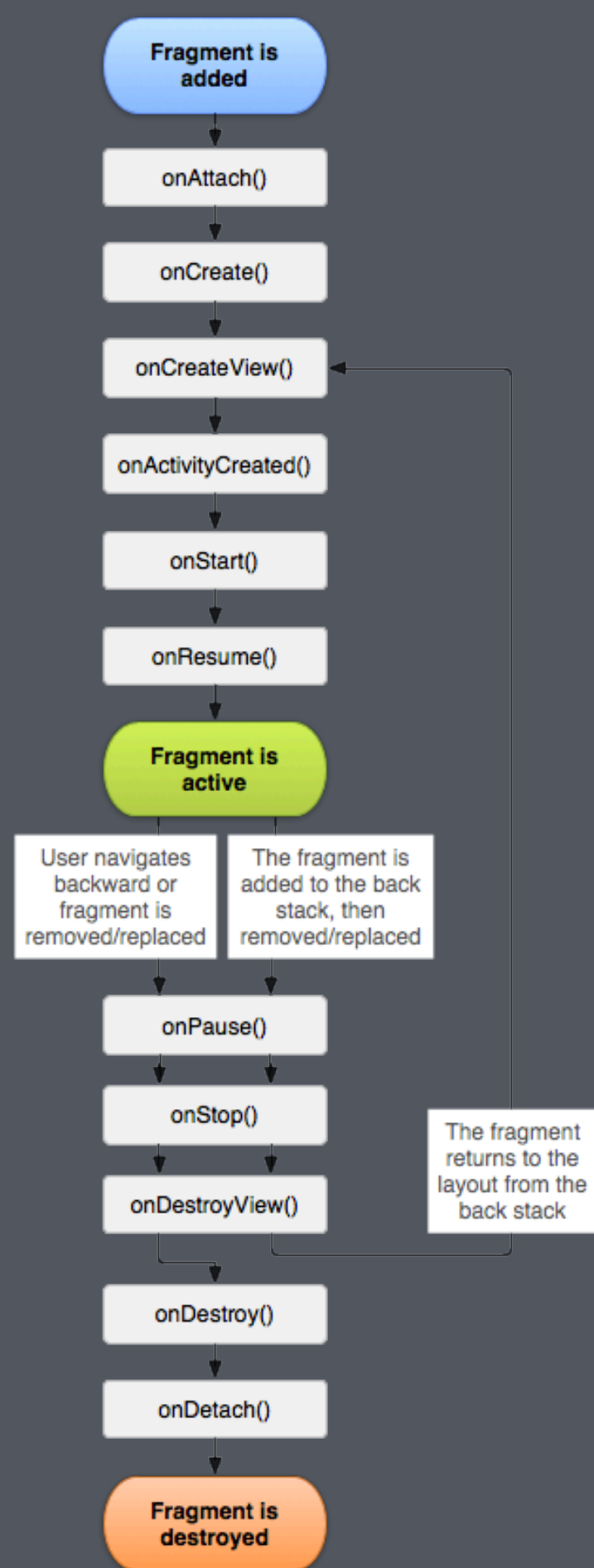
onResume()

Fragment is
active

Fragment Lifecycle

User navigates
backward or
fragment is
removed/replaced

The fragment is
added to the back
stack, then
removed/replaced



Fragments must save their state

- Fragments have their own `savedInstanceState` you must use as well.
- Fragments can have *Bundled Arguments* - this can be a good place to store state data.
- Fragments are *created **before** activities* sometimes.
 - Many fragments have *Listeners* (aka delegates) to their Activities. Don't try to use or reference these before *onActivityCreated()**

iOS is Heading More and More Towards
Android's Layout Structure

Layout Basics

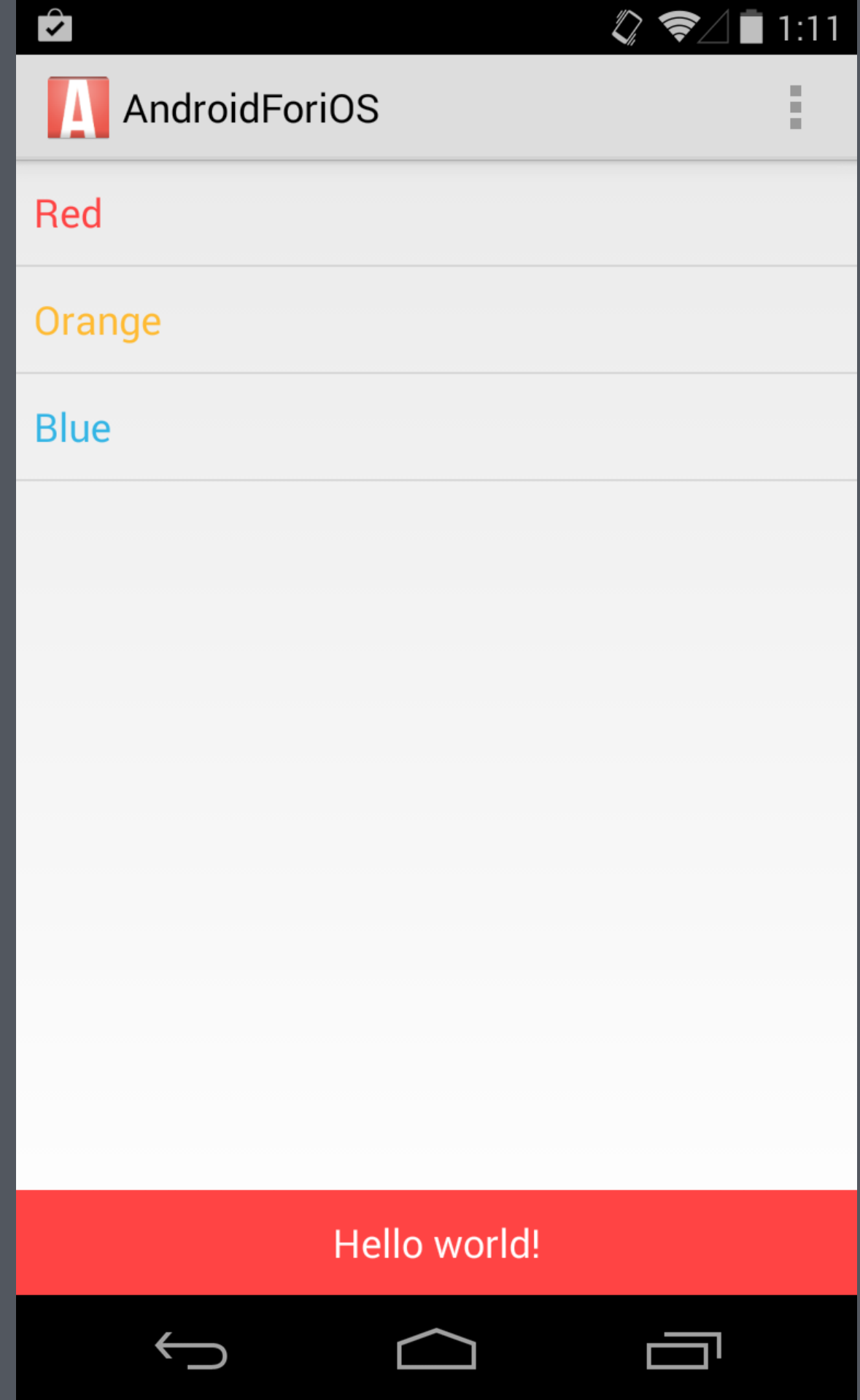
- Declare UI elements in XML
- Instantiate layout elements at runtime

Width and Height

All elements require a width and a height. Typically, either:

- `match_parent`
- `wrap_content`

Subway ListView Layout



```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.example.androidforios.app.activities.MainActivity$PlaceholderFragment">

    <ListView
        android:id="@+id/fragment_subway_list_listview"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:paddingBottom="@dimen/Button.Default.Height" />

    <Button
        android:id="@+id/fragment_subway_list_Button"
        android:layout_width="match_parent"
        android:layout_height="@dimen/Button.Default.Height"
        android:minHeight="@dimen/Button.Default.Height"
        android:background="@drawable/button_red_selector"
        android:text="@string/hello_world"
        android:textColor="@color/Button.Text"
        android:layout_alignParentBottom="true"
        android:gravity="center" />
</RelativeLayout>
```

Main Layout ViewGroups

- RelativeLayout
- LinearLayout
- FrameLayout

Layout Tips

Instead of Press States, You Have Selectors

```
<selector xmlns:android="http://schemas.android.com/apk/res/android">  
    <item android:state_pressed="true" android:drawable="@color/Button.Selected" />  
    <item android:drawable="@color/Button.Background" />  
</selector>
```

- Always work in dp (**Density-independent Pixels**) instead of pixels directly.
- Don't bother nudging items in the visual editor for layouts. Your best bet is to adjust the xml directly. Sorry, this is no interface builder.
- Design your layouts to handle different sizes!

Data

Options

- Shared Preferences <-> **NSUserDefaults**
- In memory objects
- Saving to and fetching from file structure via the **internal** or **external** file storage <-> saving to the documents directory
- **SQLite** <-> **Core Data**

Homework

Android Homework

- Learn the UI: Action Bar, Overflow Menu, and the Menu Button
- Cross App Data Sharing
- Intents and Responding to common OS actions
- Java's features: Generics, virtual methods and classes
- Google's Compatability Libraries
- The Android Emulator: Install the x86 HAXM plugin to make the emulator buttery smooth.

Popular Libraries

- **Volley <-> AFNetworking**
- **ActiveAndroid <-> MagicalRecord**
- **Picasso <-> SDWebImage**
- **Google Support Library**

Go Out and Build Something Cross Platform!