The Art of Building Reusable UI

AnDevCon Boston 2015

Stephen Barnes
@smbarne

github.com/smbarne/AndroidReusableUI

Who is this guy?

Senior iOS Developer @Fitbit, previously Senior Mobile Developer @Raizlabs

@smbarne

smbarne

engineeringart.io

Resources

This class assumes you are comfortable with using Android UI components and editing layout xml files.

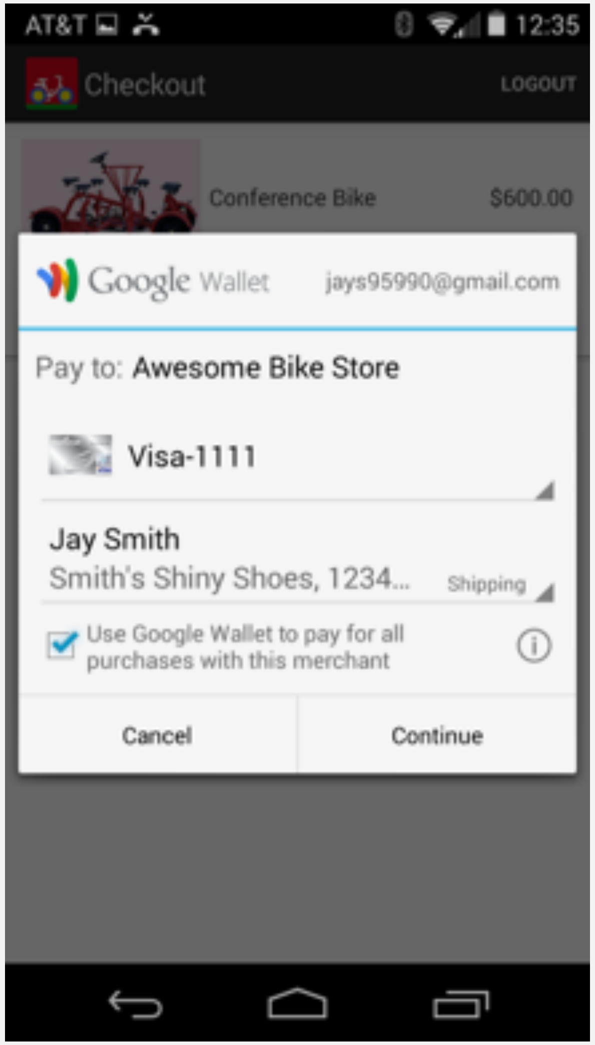
Slides and sample code can be found on Github at github.com/smbarne/AndroidReusableUI

What is Reusable UI

A Reusable UI element is an encapsulated component with a user facing element.

- Responsibilities for a reusable component can be large or small.
- Reusable UI elements are typically extensions of system components.
- Good Reusable UI is opinionated: it describes a narrow situation.

Large, Activity Level Components



AT&T 12:35

Checkout LOGOUT

Conference Bike $600.00

Google Wallet jays95990@gmail.com

Pay to: Awesome Bike Store

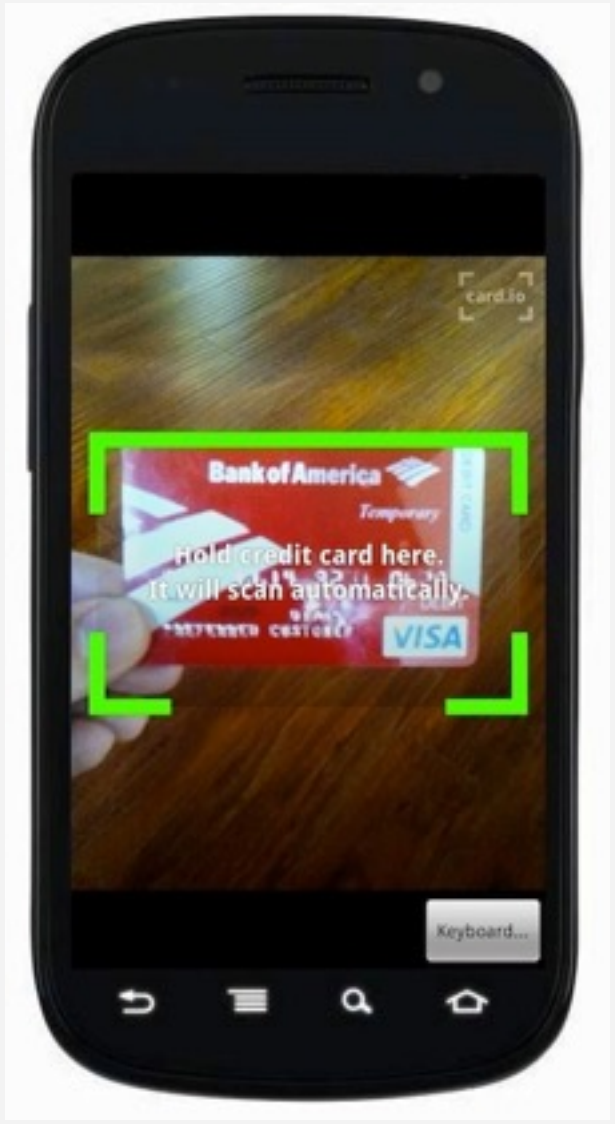
Visa-1111

Jay Smith
Smith's Shiny Shoes, 1234... Shipping

Use Google Wallet to pay for all purchases with this merchant

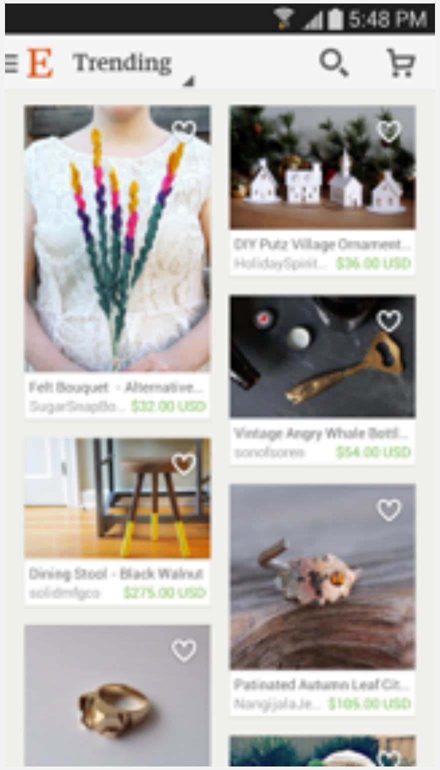
Cancel | Continue

Google Wallet



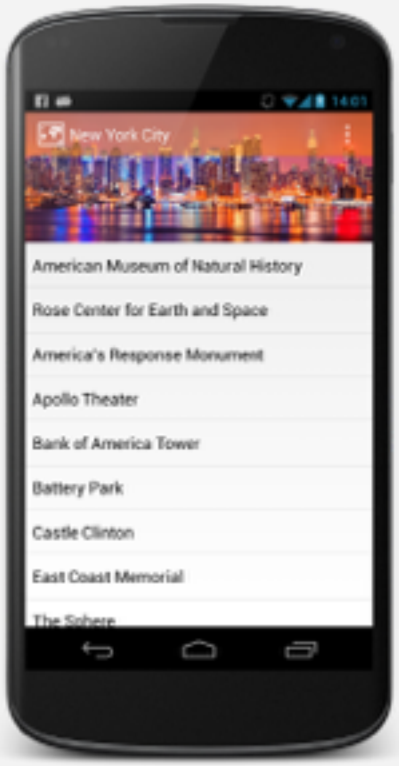
Card.io

Small, Modular Reusable UI Components


5:48 PM
Trending
DIY Putz Village Ornament...
HolidaySpirit... $36.00 USD
Felt Bouquet - Alternative...
SugarSnapBo... $32.00 USD
Vintage Angry Whale Bottl...
sonofsoren $54.00 USD
Dining Stool - Black Walnut
solidmfgco $275.00 USD
Patinated Autumn Leaf Cit...
NangijalaJe... $105.00 USD

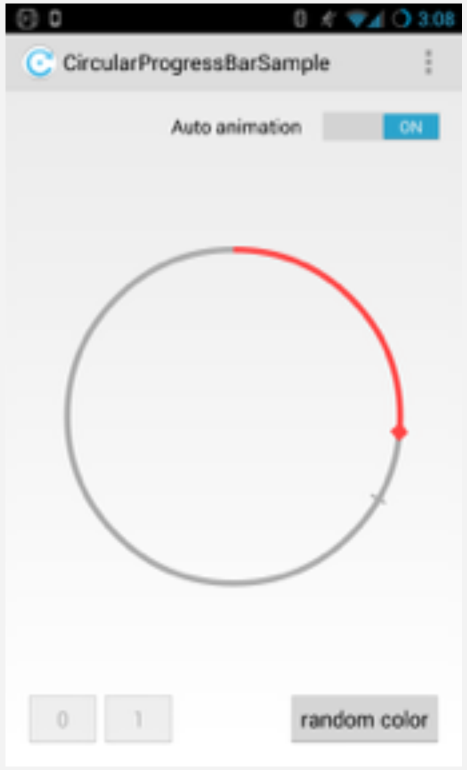
StaggeredGrid

github



FadingActionBar

github



CircularProgressBar

github

“Before software can be reusable it first has to be usable.”

- Ralph Johnson

Outline for Creating Great Reusable UI

User Happiness: Design UI that provides a good user experience and lasts the tests of time.

Developer Happiness: Create a clear, friendly API and make it easy to use.

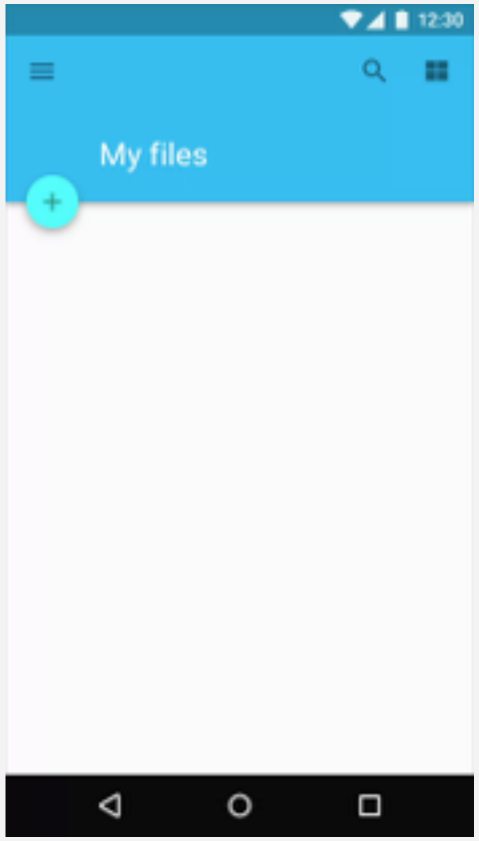
Operating System Happiness: Take advantage of OS capabilities such as utilizing XML properties in Android.

Let’s Build Something!

Problem

To provide a good UX, it is best to not block user interaction with a progress dialog when loading (the Android Design guidelines cover this here).

One approach is to use the 'Activity Circle' pattern. However, often, when loading data from the web, we need a tri-state ViewGroup.



The States

- Loading Data State
- Content State
- Empty State (Banana Peel*)

A “banana peel” is a Raizlabs term coined by Jon Green (@gildedsplinter) for a useful empty state graphic. The empty state is is something you can slip on, so be careful!

Agenda

1. Create an engaging, robust, design.
2. Create an initial API description
3. Core UI implementation
4. Create and use the control via code
5. Create and use the control via XML

Agenda Pt 2

6. Default component styling
7. Create a library
8. Integrate into another project
9. Customize styles via overridden XML
10. Customize via Subclassing

Practical UI Design

Define Primary Features

Three states

Can contain any content

Empty state needs a description and an image

Empty state needs to be tappable

A good reusable component can do one thing very well.

Cover The Details

Remember press states

Plan around transitions

Design for any aspect ratio

Your UI should adapt to the content

Thumb-friendly tap areas

Respect the OS: don't repurpose common UI elements or patterns for new ones (ex: don’t highjack the back or up buttons)

Get Inspired, But Stick to the Rules

Inspiration Resources

mobile-patterns.com/android

androidniceties.tumblr.com

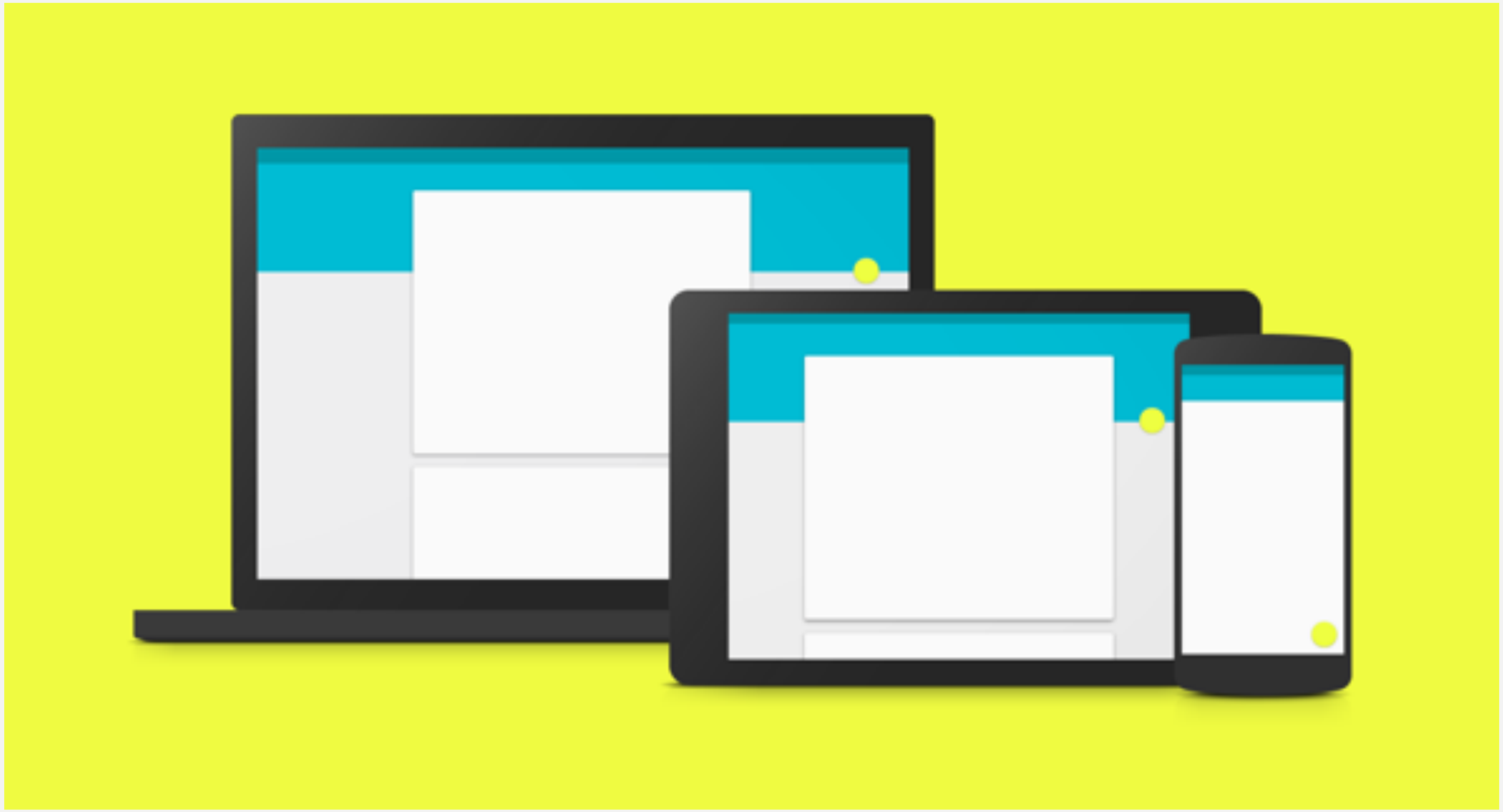
android.inspired-ui.com/

Design Rules

- developer.android.com/design
- petrnohejl.github.io/Android-Cheatsheet-For-Graphi
  Designers/

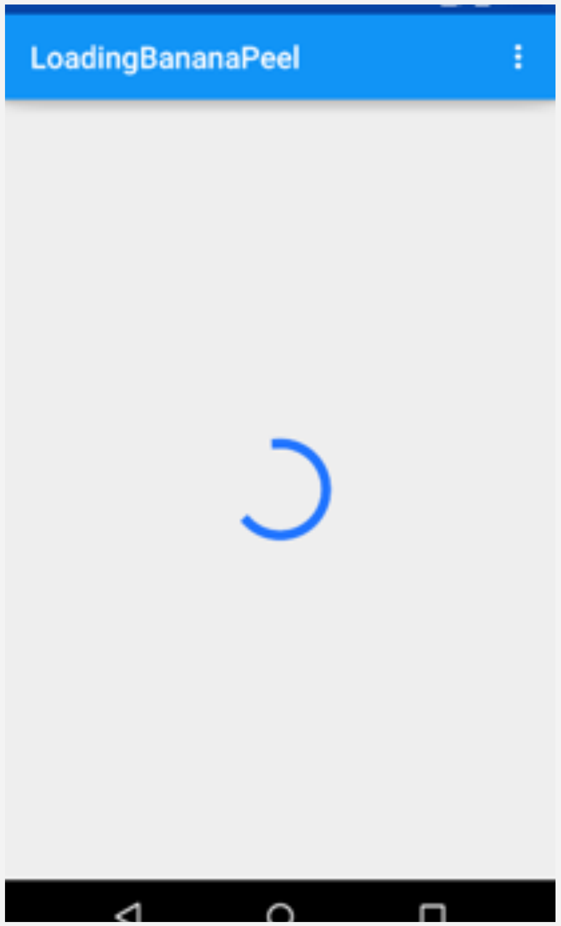
Material is Your Friend

The Material design language is powerful and well documented at developer.android.com/design. iOS has Apple's HIG (Human Interface Guidelines), Google has the Android Design Guidelines.

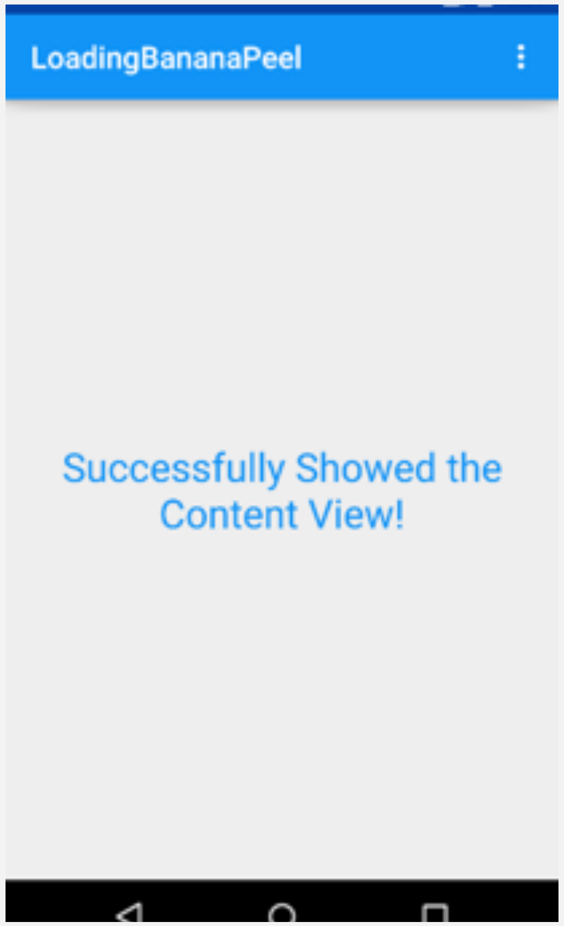


Designing Our View

Create a customizable, reusable ViewGroup with loading, content, and empty state.

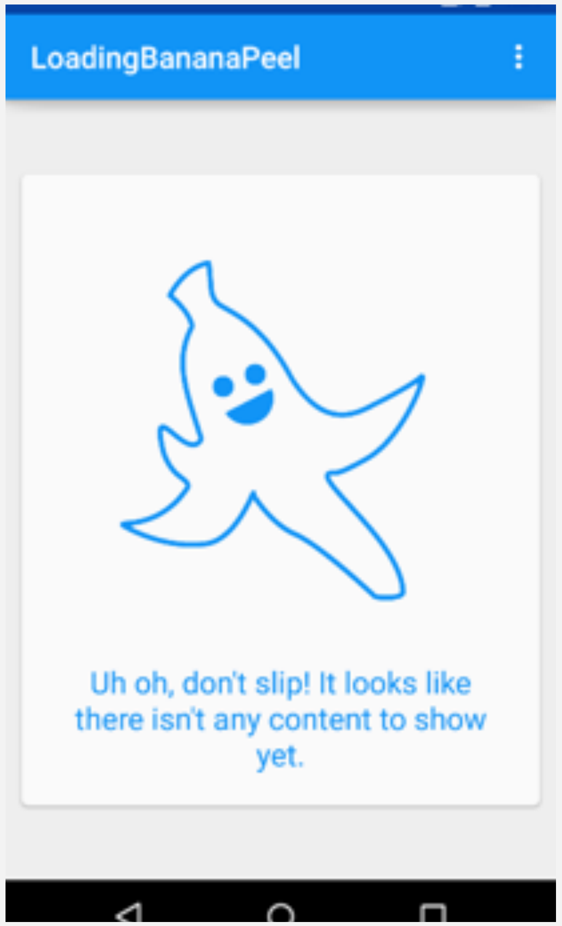


Loading



Successfully Showed the Content View!

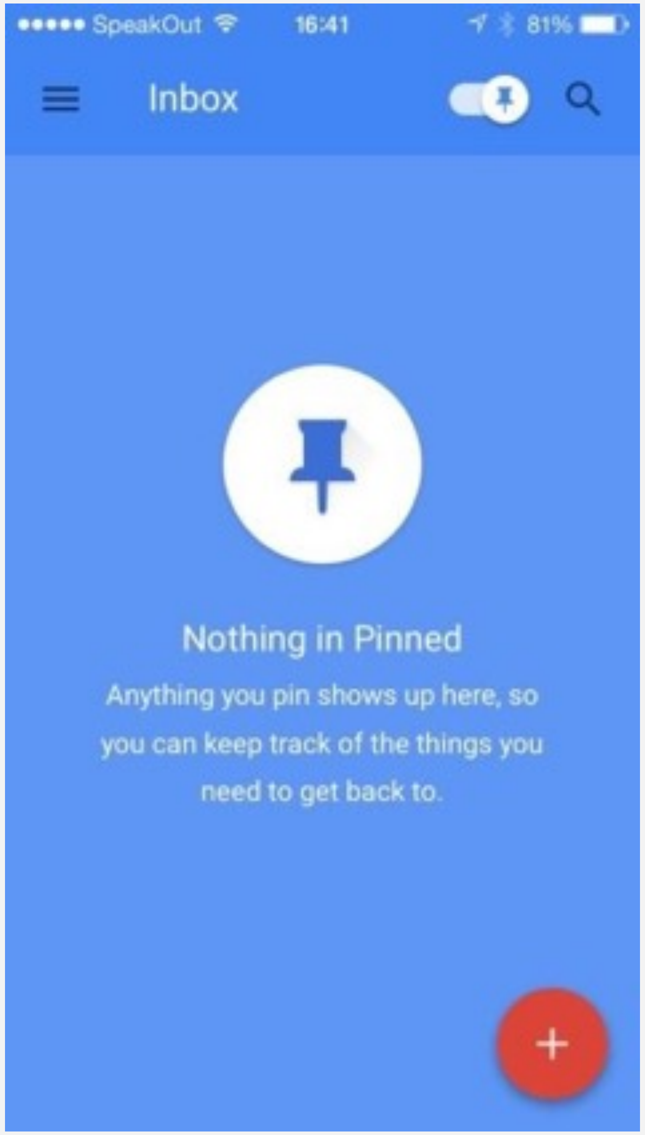
Content



Uh oh, don't slip! It looks like there isn't any content to show yet.

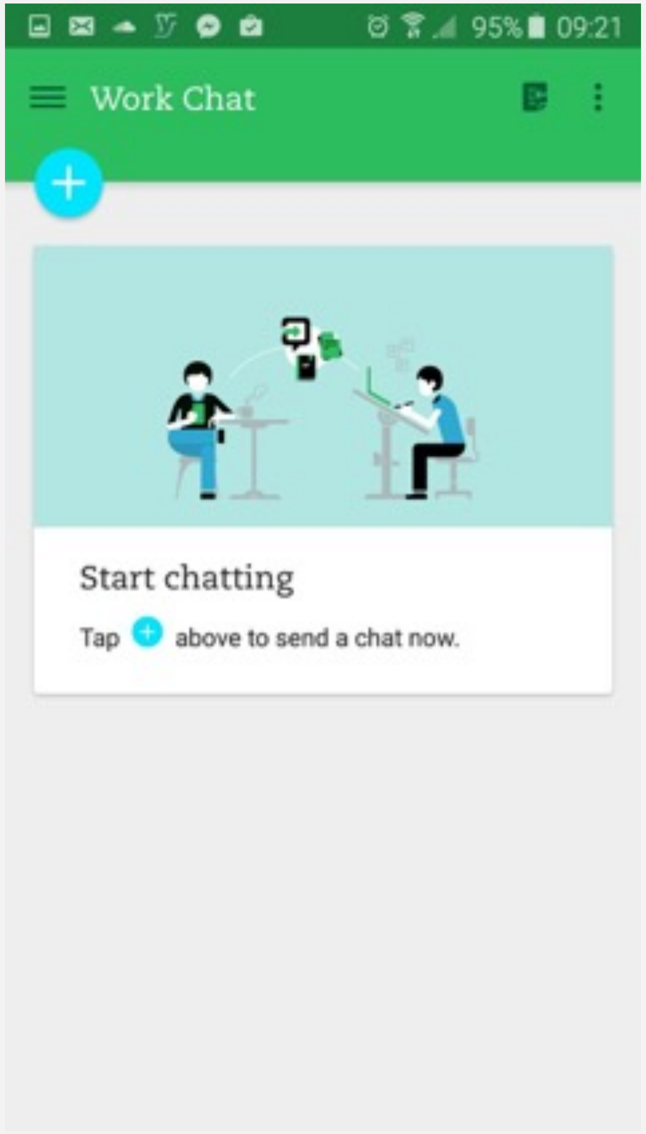
Empty State

Research: Examples of Empty States



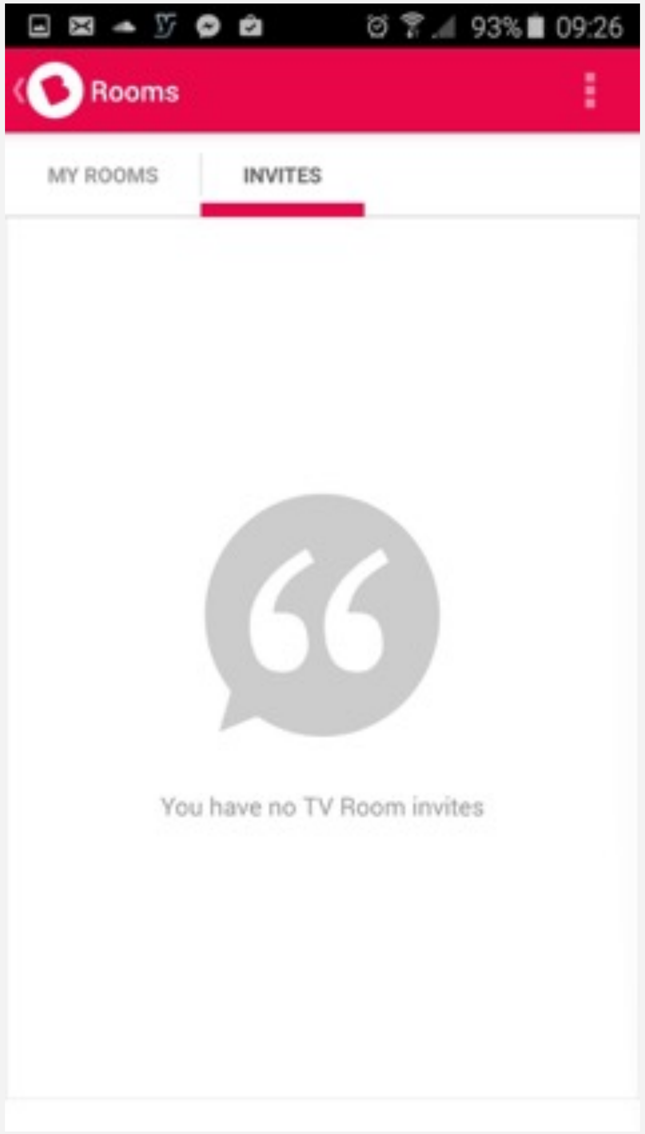
Nothing in Pinned

Anything you pin shows up here, so you can keep track of the things you need to get back to.



Start chatting

Tap above to send a chat now.



You have no TV Room invites

Friendly API Design

A Word on API Design

A Few API Requests:

- Modular
- Approachable
- Documented
- Platform Friendly

An API Should be as Minimal as Possible

The more options an API has, the harder it can be to learn. Have an opinion: implement things with authority. Mirror the primary features as methods. For example, a primary primary feature of our UI is the three UI states.

```
public void showLoading() { }
public void showContent() { }
public void showBananaPeel() { }

public View setContentView(int contentViewLayoutResourceId) { }
public View setBananaPeel(int messageResourceId, int 
imageDrawableResourceId, final BananaPeelActionListener listener) { }
```

Ready to Go, Out of the Box

Do not require additional methods to have a component work after initialization, but do allow for it.

```
/**
  * Create a banana peel with the given message and image drawable.
  * @param messageResourceId - The string resource id for the banana
peel text.
  * @param imageDrawableResourceId - The id for the drawable to be
displayed.
  * @param listener - (optional) An event listener for when the
banana peel is clicked.
 */
public void setBananaPeel(int messageResourceId, int
imageDrawableResourceId, final BananaPeelActionListener listener) {
    configureBananaPeel(imageDrawableResourceId, listener);
    setBananaPeelMessage(messageResourceId);
}
```

Ready to Go, Out of the Box

```
/**
  * Update the banana peel message text. If 0 is passed in for the
{@code messageResourceId}
  * then the image message textview will be set to visibility {@link
View#GONE}.
  * @param messageResourceId - The resource id for the banana peel
text.
  */
public void setBananaPeelMessage(int messageResourceId) {
```

```
/**
  * Update the banana peel image drawable.  If 0 is passed in for the
{@code imageDrawableResourceId}
  * then the image drawable will be set to visibility {@link
View#GONE}.
  * @param imageDrawableResourceId - The id for the drawable to be
displayed.
  */
public void setBananaPeelImage(int imageDrawableResourceId) {
```

API Documentation

Take advantage of JavaDoc and **document** your API. The method documentation and API structure should make it possible to use the control without another document.

```
/**
  * Create a banana peel with a card view with the given message and image
drawable.
  * @param messageResourceId - The string resource id for the banana peel
text.
  * @param imageDrawableResourceId - The id for the drawable to be
displayed.
  * @param listener - (optional) An event listener for when the banana
peel is clicked.
  */

public void setBananaPeel(int messageResourceId,
                          int imageDrawableResourceId,
                          final BananaPeelActionListener listener) {
    configureBananaPeel(imageDrawableResourceId, listener);
    setBananaPeelMessage(messageResourceId);
}
```

LoadingBananaPeel Implementation

LoadingBananaPeel Approach

Subclass ViewFlipper.

Expose the content view and the empty state setters.

Create and expose an interface for the empty state click listener.

Inflate and layout our views.

Make our public API thread-safe.

Overrides for Subclassing a View

```
public class MyView extends View {

    // Constructor required for in-code creation
    public MyView(Context context){
        super(context);
        init(context, null);
    }

    // Constructor required for inflation from resource file
    public MyView(Context context, AttributeSet attr){
        super(context,attr);
        init(context, attrs);
    }

    // Private common init method
    private void init(Context context, AttributeSet attrs) {
        // …
    }
}
```

Initializing with Default Layout XML

Often, if UI is non-trivial, it is best to create an XML file to inflate instead of creating elements manually. In this case, create a default XML layout file for your reusable UI (we'll cover overriding this later).

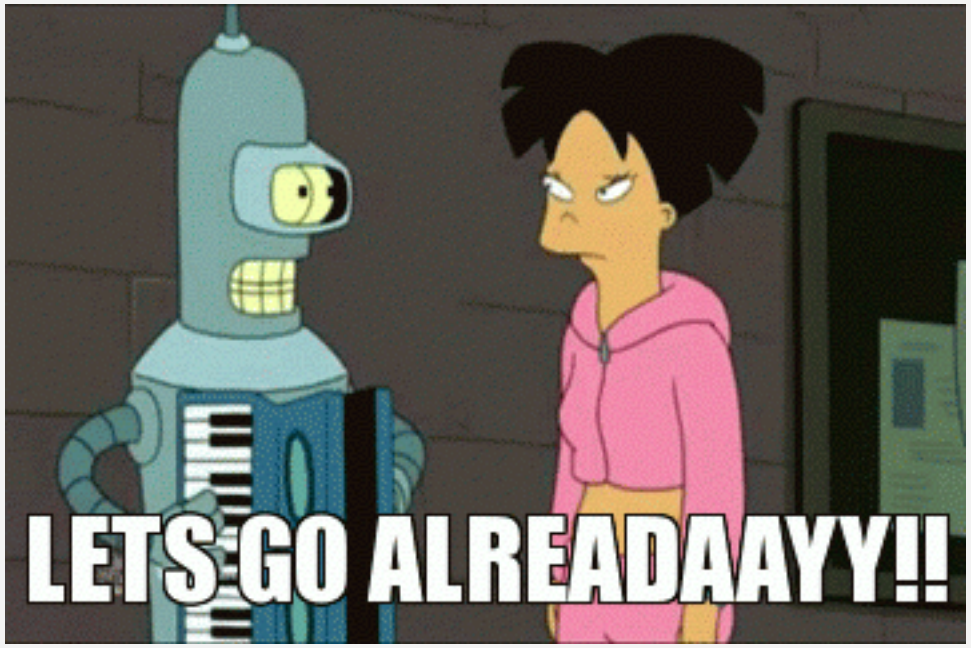
In our case, we want default layouts for each of our three states.

Inflating Our Default XML

Inflating and Adding the default banana peel view state.

```
@Override
protected void onFinishInflate() {
    super.onFinishInflate();
    bananaPeelView =
inflateAndAddResource(R.layout.view_default_banana_peel);
}

private View inflateAndAddResource(int layoutResourceId) {
    LayoutInflater inflater = LayoutInflater.from(getContext());
    View bananaPeel = inflater.inflate(layoutResourceId, this, false);
    addView(bananaPeel);
    cacheViewIndicies();
    return bananaPeel;
}
```



Let's Go Already!

< Coding Time />

Exposing Configuration via Code

State Management

It is important to expose configuration setters publicly. But remember!

Break out configuration into modular methods.

Use the same methods to initialize the component.

Update the component state after setting properties.

BananaPeel Code Example

Public API Convenience Method

```
/**
  * Create a banana peel with with the given message and image drawable.
  * @param messageResourceId - The string resource id for the banana peel
text.
  * @param imageDrawableResourceId - The id for the drawable to be
displayed.
  * @param listener - (optional) An event listener for when the banana
peel is clicked.
  */
public void setBananaPeel(int messageResourceId, int
imageDrawableResourceId, final BananaPeelActionListener listener) {
    configureBananaPeel(imageDrawableResourceId, listener);
    setBananaPeelMessage(messageResourceId);
}
```

BananaPeel Code Example

Public API Setter

```
/**
  * Update the banana peel message text. If 0 is passed in for the {@code
messageResourceId}
  * then the image message textview will be set to visibility {@link
View#GONE}.
  * @param messageResourceId - The resource id for the banana peel text.
  */
public void setBananaPeelMessage(int messageResourceId) {
    if (bananaPeelView != null) {
        TextView bananaTitle = (TextView)
bananaPeelView.findViewById(R.id.bananaPeel_TextView);
        if (messageResourceId != 0) {
            bananaTitle.setVisibility(VISIBLE);
            bananaTitle.setText(messageResourceId);
        } else {
            bananaTitle.setVisibility(GONE);
        }
    }
}
```

Exposing Configuration via XML

UI Configuration the Android Way

Often, we want to include our reusable UI in a layout XML file. Wouldn’t it be great if we could configure it in the XML file as well?

Styleable XML Attributes

First, we need to define styleable attributes in a XML file specific to our component.

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <declare-styleable name="LoadingBananaPeelView">
        <attr name="bananaPeelDefaultMessageStringResource"
format="reference" />
        <attr name="bananaPeelDefaultImageResource" format="reference" />
        <attr name="bananaPeelViewResource" format="reference" />
        <attr name="bananaPeelContentViewLayoutResource"
format="reference" />
    </declare-styleable>
</resources>
```

Valid Attribute Format Types

```
public enum AttributeFormat {
    Reference, String, Color, Dimension, Boolean, Integer, Float,
Fraction, Enum, Flag
}
```

Applying Attributes

When configuring your View, you can retrieve and apply the attributes at runtime.

```
public LoadingBananaPeelView(Context context, AttributeSet attrs) {
    super(context, attrs);
    init(attrs);
}
private void init(AttributeSet attrs) {
    hadAttrs = (attrs != null);
    if (attrs != null) {
       TypedArray styledAttributes =
getContext().getApplicationContext().obtainStyledAttributes(attrs,
R.styleable.LoadingBananaPeelView);
       bananaPeelViewResourceId =
styledAttributes.getResourceId(R.styleable.LoadingBananaPeelView_bananaPee
lViewResource, 0);
       styledAttributes.recycle();
    }
}
```

Using Styleable Attributes in Layouts

Lastly, specify values for any of your custom styled attributes in a layout XML file. Note the namespace!

```
<com.smb.loadingbananapeel.LoadingBananaPeelView
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:id="@+id/fragment_content_loading_view"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   app:bananaPeelDefaultImageResource="@drawable/ic_bananapeel_default"
   app:bananaPeelDefaultMessageStringResource="@string/
banana_peel_default_empty_message"
   app:contentViewLayoutResource="@layout/view_example_content" />
```

Namespacing

Prepend your styleable attributes!

There is no namespacing for styleable XML attributes. If you name them generically in your library, they can collide during the dexing process.

Exposing Properties via XML

< Coding Time />

Time to Get Opinionated: Default Styling

Zero Effort Pretty

Others are more likely to use the component you have spent your time on if it does not require any styling to make it look nice. Make sure that your UI looks presentable without any overrides.

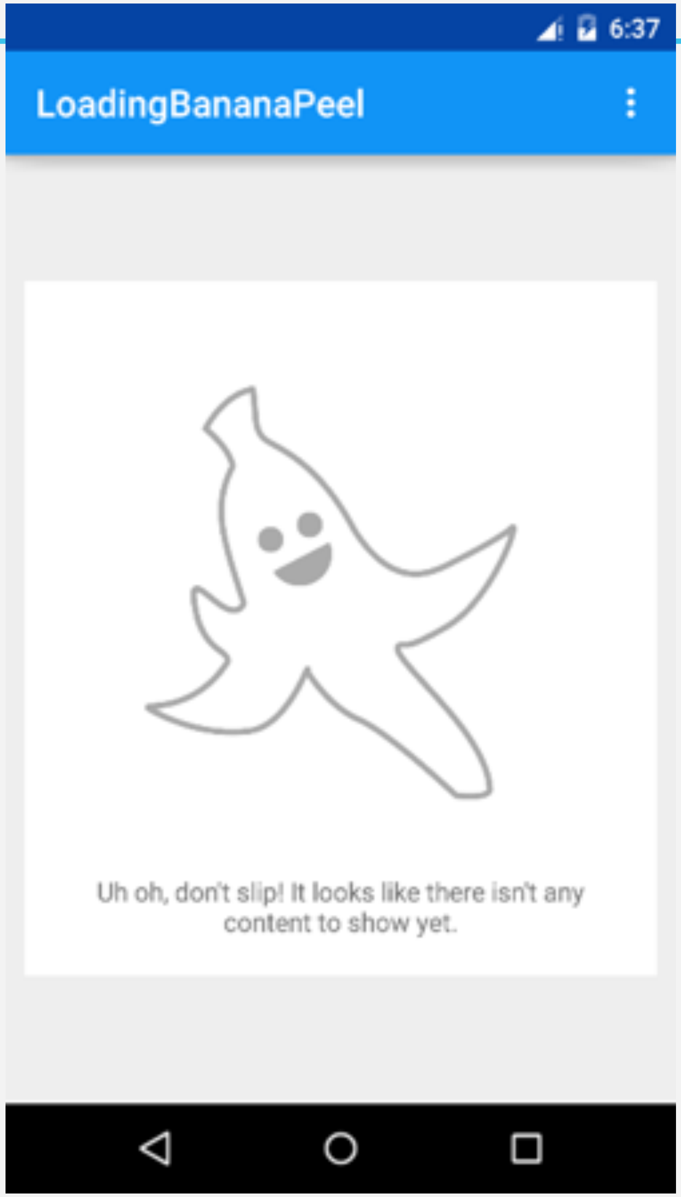
LoadingBananaPeel Default Styling

Place the banana peel image and message within a card.

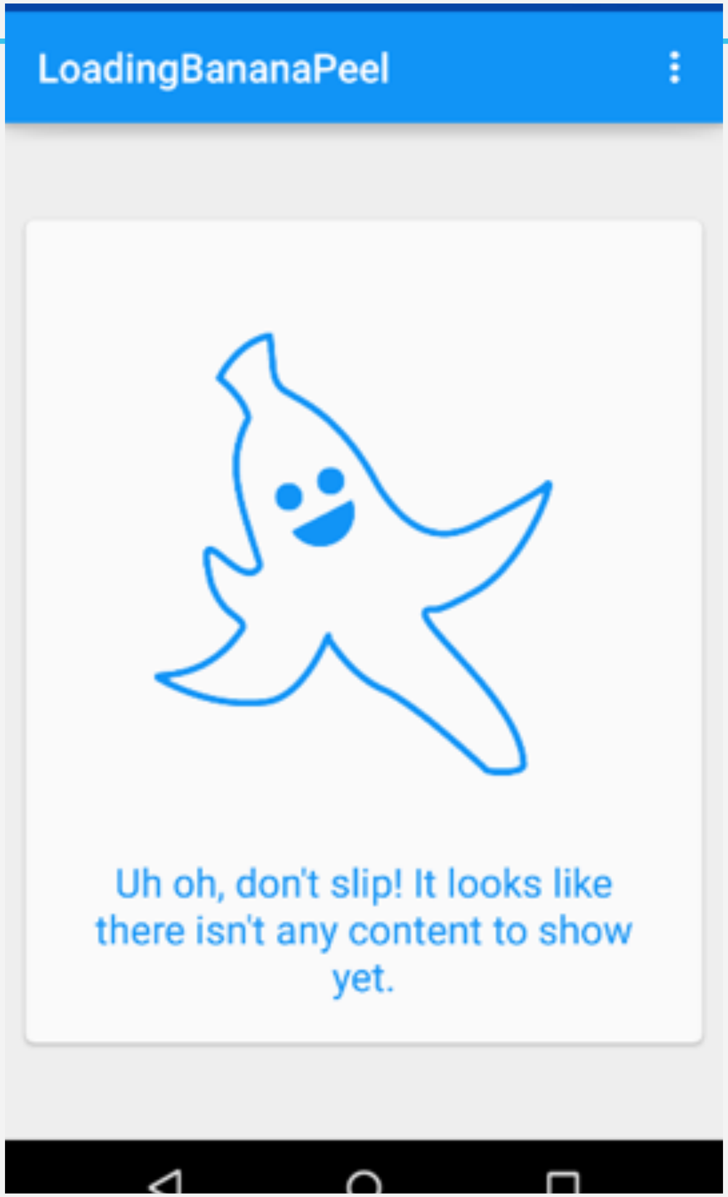
Give a default banana peel image.

Update the font size and color.

Match the Activity Theme’s primary color.



Uh oh, don't slip! It looks like there isn't any content to show yet.



Uh oh, don't slip! It looks like there isn't any content to show yet.

Create a Custom XML Files

For each of the following, create new files for your component.

Style XML File

Color XML File

Dimens XML File

Values XML File

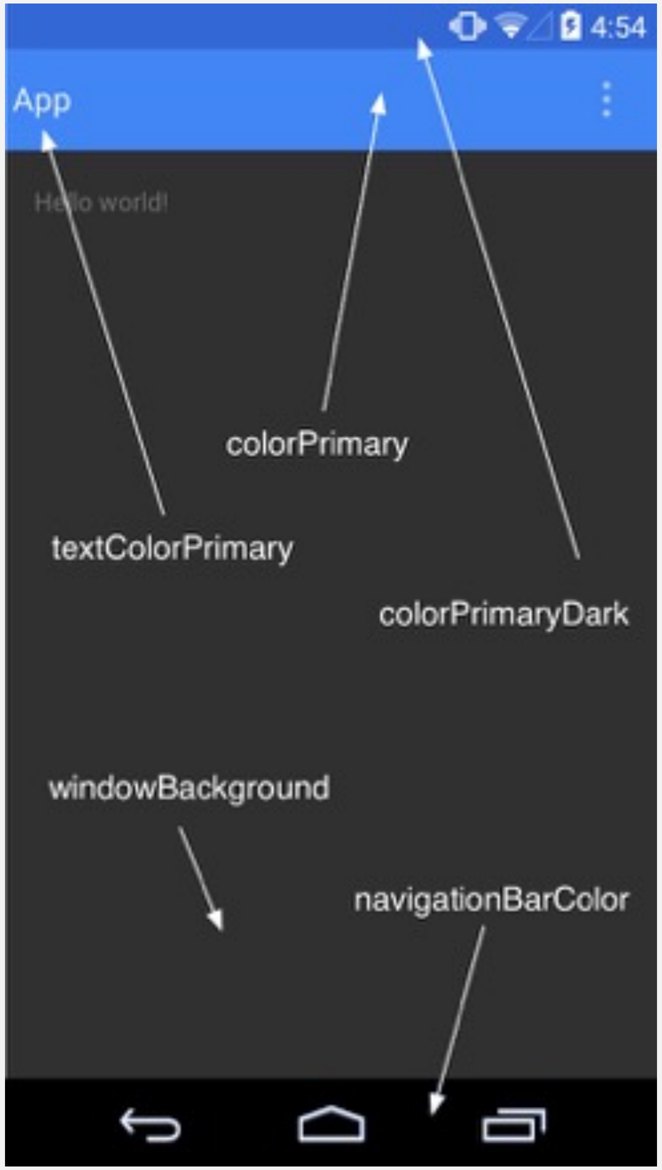
String XML File

Material Fun

Android L and Material design have introduced some great concepts for customizable UI components. Two of which are:

- Activity Configurable Colors
- Image Tinting via XML

Material Fun



source: http://treyrobinson.net/blog/android-l-tutorials-part-2-material-theme-colors/

Material Fun

```
<style name="AppTheme" parent="AppTheme.Base"/>

<!-- Base application theme. -->
<style name="AppTheme.Base" parent="Theme.AppCompat.Light.DarkActionBar" >
    <item name=“colorPrimary"> @color/Material.Blue </item>
    <item name=“colorPrimaryDark”> @color/Material.Blue.Dark </item>
    <item name=“colorAccent”> @color/Material.Blue.Accent </item>
    <item name=“android:textColorPrimary”> @color/Material.White </item>
</style>
```

Material Fun

```
<ImageView
            android:id="@+id/bananaPeel_ImageView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:contentDescription="Banana Peel: Empty State"
            android:tint="?android:colorPrimary"/>
<TextView
            android:id="@+id/bananaPeel_TextView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:gravity="center"
            android:textColor="?android:attr/colorPrimary"
            android:textSize="20sp"/>
```

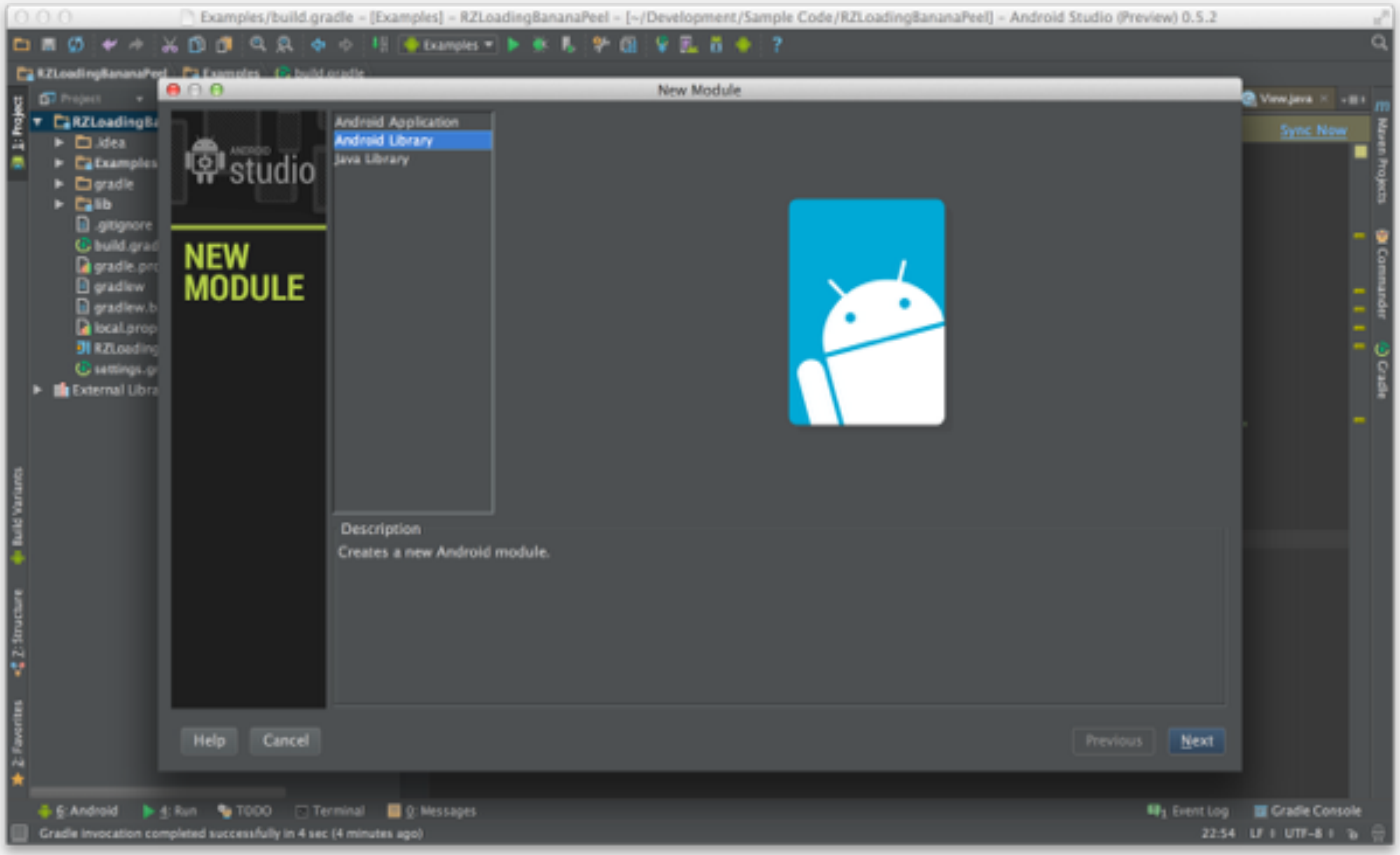
Make All the Things Pretty

< Coding Time />

Make it Truly Reusable: Build a Library

Creating a Library

Start by going to File -> New Module. Select Android Library.


Examples/build.gradle - [Examples] - RZLoadingBananaPeel - [~/Development/Sample Code/RZLoadingBananaPeel] - Android Studio (Preview) 0.5.2
New Module
NEW MODULE
Android Application
Android Library
Java Library
Help
Cancel
Previous
Next

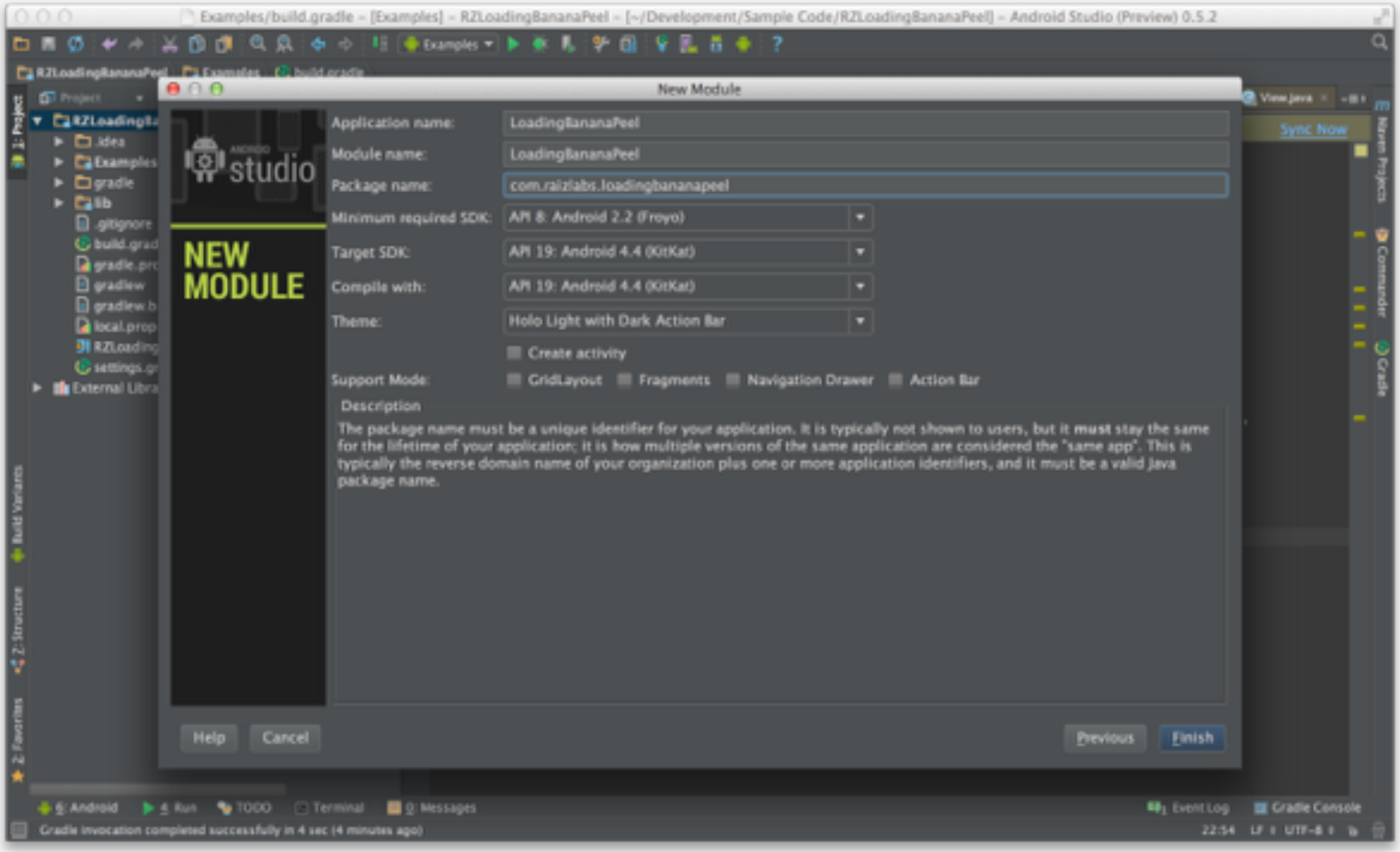
Description

Creates a new Android module.

Gradle invocation completed successfully in 4 sec (4 minutes ago)

Creating a Library

Configure your Module



Examples/build.gradle - [Examples] - RZLoadingBananaPeel - [~/Development/Sample Code/RZLoadingBananaPeel] - Android Studio (Preview) 0.5.2

New Module

NEW MODULE

Application name: LoadingBananaPeel

Module name: LoadingBananaPeel

Package name: com.raizlabs.loadingbananapeel

Minimum required SDK: API 8: Android 2.2 (Froyo)

Target SDK: API 19: Android 4.4 (KitKat)

Compile with: API 19: Android 4.4 (KitKat)

Theme: Holo Light with Dark Action Bar

☐ Create activity

Support Mode: ☐ GridLayout ☐ Fragments ☐ Navigation Drawer ☐ Action Bar

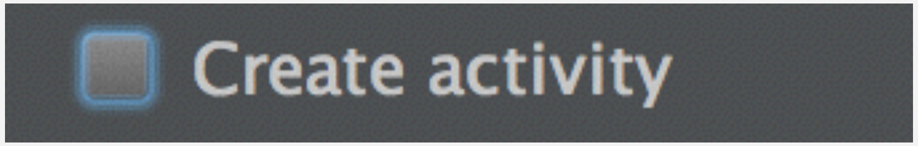
Description

The package name must be a unique identifier for your application. It is typically not shown to users, but it must stay the same for the lifetime of your application; it is how multiple versions of the same application are considered the "same app". This is typically the reverse domain name of your organization plus one or more application identifiers, and it must be a valid Java package name.

Gradle invocation completed successfully in 4 sec (4 minutes ago)

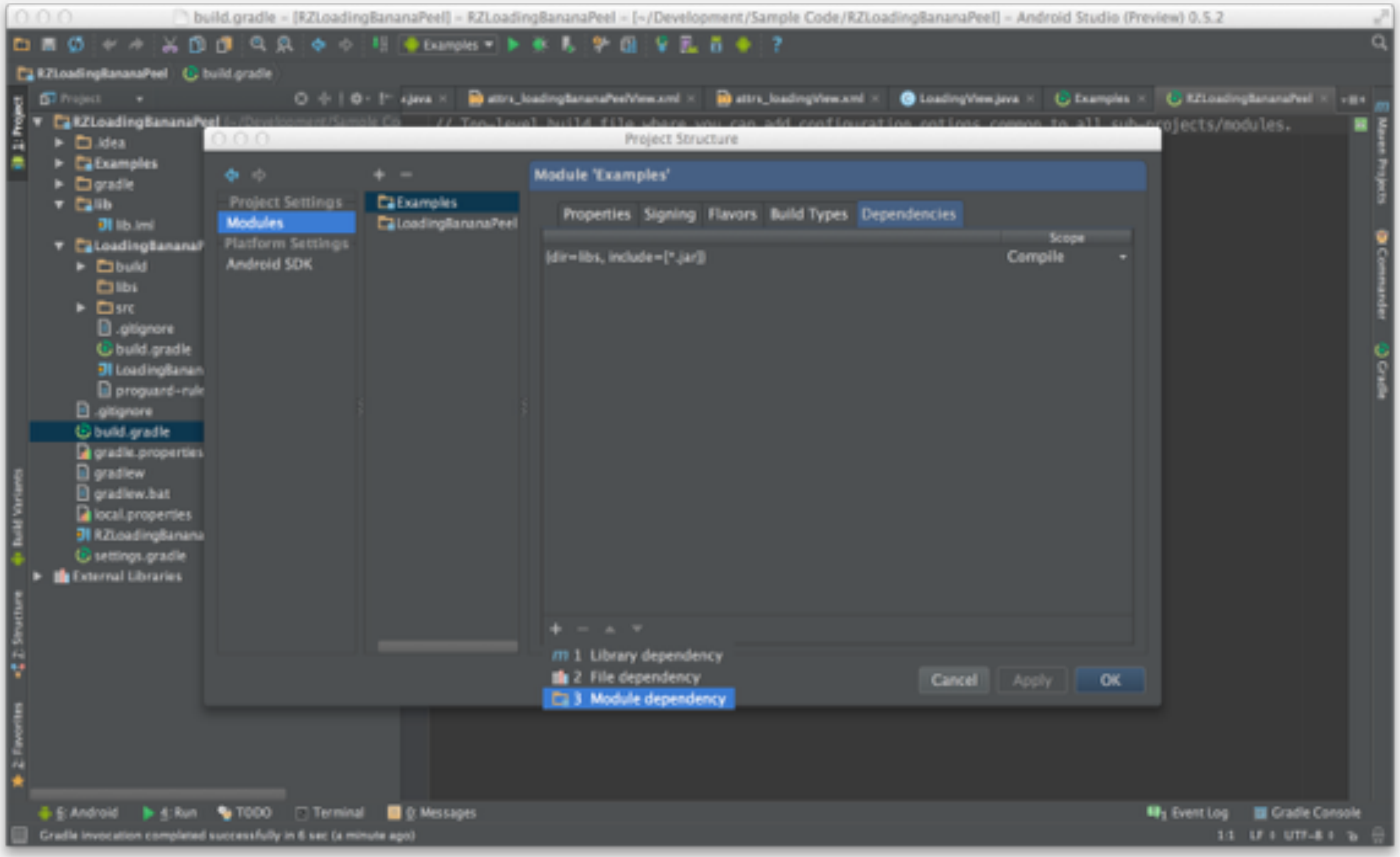
Creating a Library

Don’t forget to unselect ‘Create Activity’



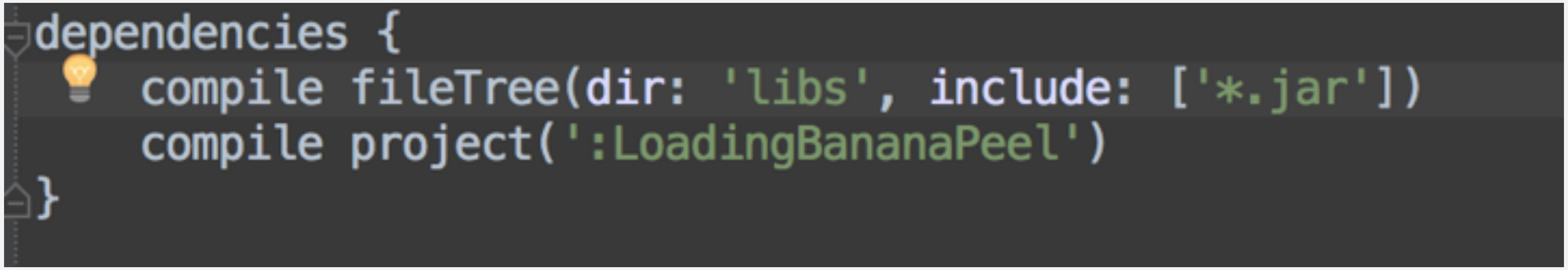
Creating a Library

Add your library as a dependency to the main project.



Creating a Library

Note that your build.gradle will update for your primary project.



```
dependencies {
    compile fileTree(dir: 'libs', include: ['*.jar'])
    compile project(':LoadingBananaPeel')
}
```

Settings.gradle

Include all of your module definitions in settings.gradle

```
include ':app', ':LoadingBananaPeel'
```

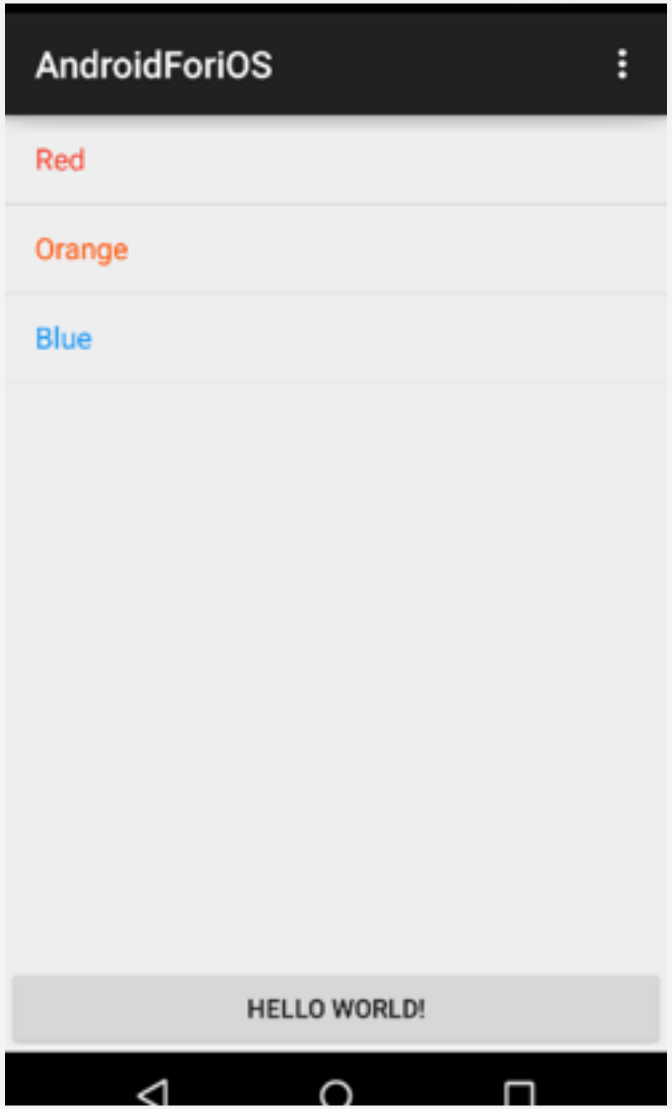
Library Integration

< Coding Time />

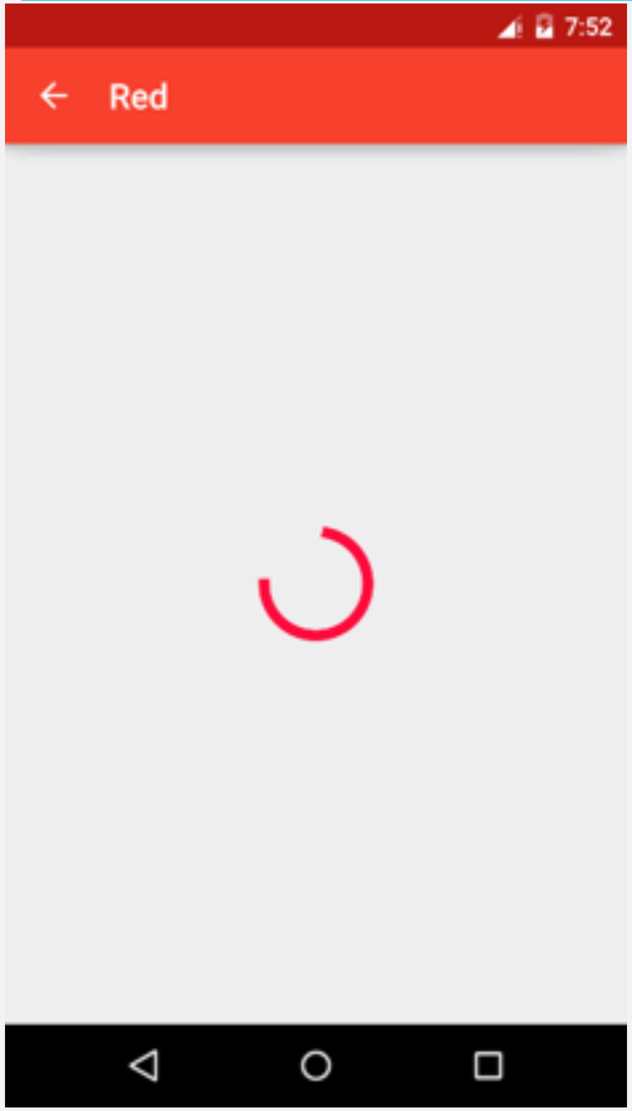
Integration into Another Project

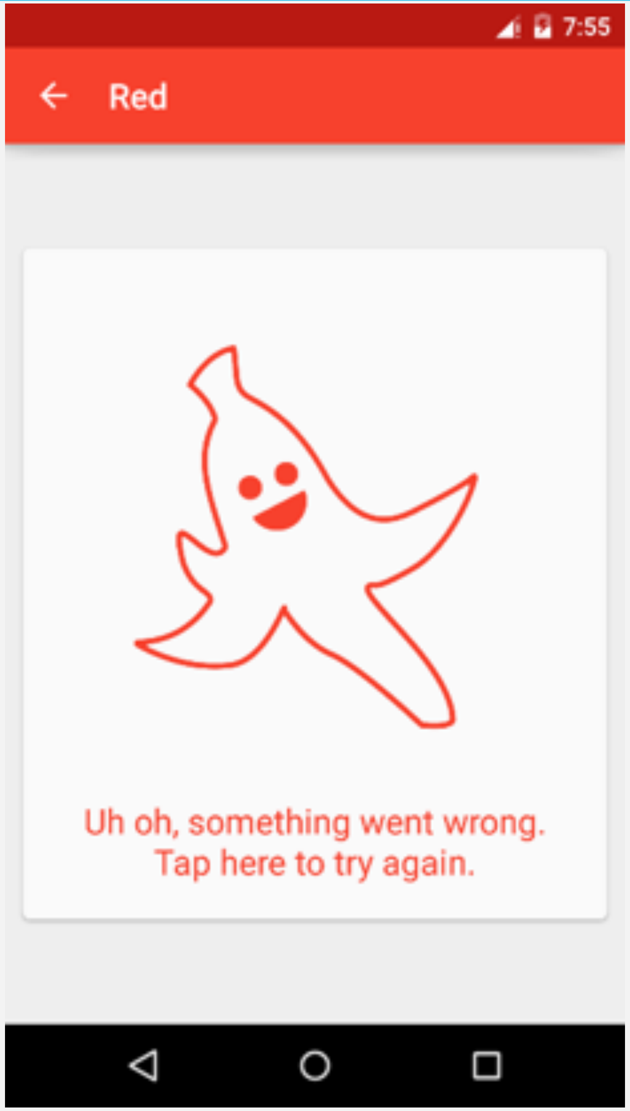
Let’s integrate the LoadingBananaPeel into another project.

Let’s use a small sample app that loads MBTA train data.

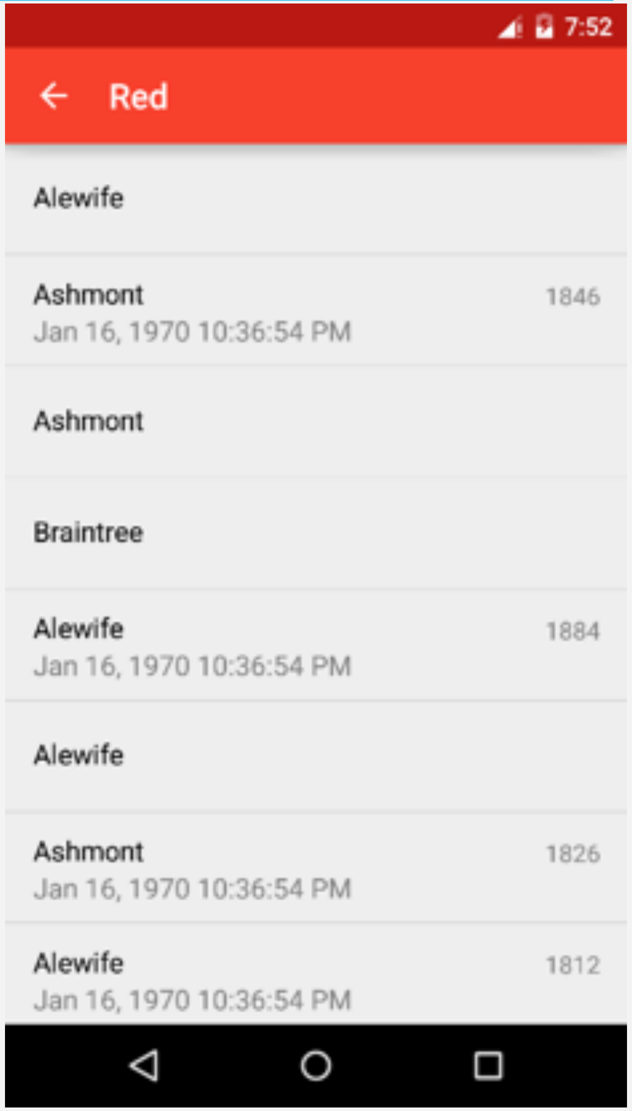


You can grab the code from:
github.com/smbarne/AndroidForiOS





Uh oh, something went wrong.
Tap here to try again.



Alewife

Ashmont 1846
Jan 16, 1970 10:36:54 PM

Ashmont

Braintree

Alewife 1884
Jan 16, 1970 10:36:54 PM

Alewife

Ashmont 1826
Jan 16, 1970 10:36:54 PM

Alewife 1812
Jan 16, 1970 10:36:54 PM

Library Integration Pt 2

< Coding Time />

Better than Vanilla: Customizing your Custom UI.

Overriding Default XML Styles

Overriding XML Files

XML files In the parent project override XML files in library projects. For a quick win, create an alternate implementation of view_default_banana_peel.xml with different coloring coloring, spacing, and typography.

Override Specific Values

You can also override specific values such as specific padding dimensions declared in your library.

```
<dimen name=“BananaPeel.DefaultMargin”>30dp</dimen>
```

Subclassing Your Reusable UI

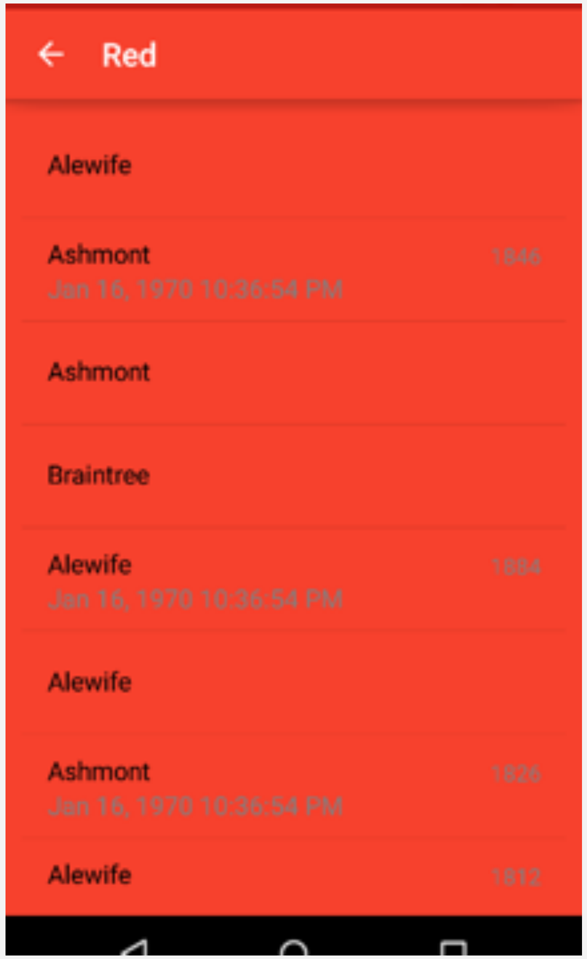
When and Why to Subclass

Congratulations, you can now use your wonderful, reusable UI in all of your projects. But what about special changes?

When you need to make special changes for one specific project - **subclass** instead of than inserting one-off code into the base component.

Subclassing Example

Let’s say we now want our content views to be inset and have a red background on them.



Alewife

Ashmont 1846
Jan 16, 1970 10:36:54 PM

Ashmont

Braintree

Alewife 1884
Jan 16, 1970 10:36:54 PM

Alewife

Ashmont 1826
Jan 16, 1970 10:36:54 PM

Alewife 1812

Subclassing

```
public class RedContentBananaPeel extends LoadingBananaPeelView {
   @Override
    public View setContentView(View contentView) {
        View inflatedView = super.setContentView(contentView);

        // Modify the inflated content view (hideously)
       inflatedView.setBackgroundColor(getResources().getColor
                                               (R.color.Material_Red));
        int padding =
          getResources().getDimensionPixelSize(R.dimen.Padding_Large);
        inflatedView.setPadding(padding, padding, padding, padding);

        return inflatedView;
    }
}
```

Subclassing

< Coding Time />

Thanks!

Slides and sample code can be found on Github at github.com/smbarne/AndroidReusableUI

@smbarne