



# Android Development

## Lecture 2

### Android Graphical User Interface 1



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).

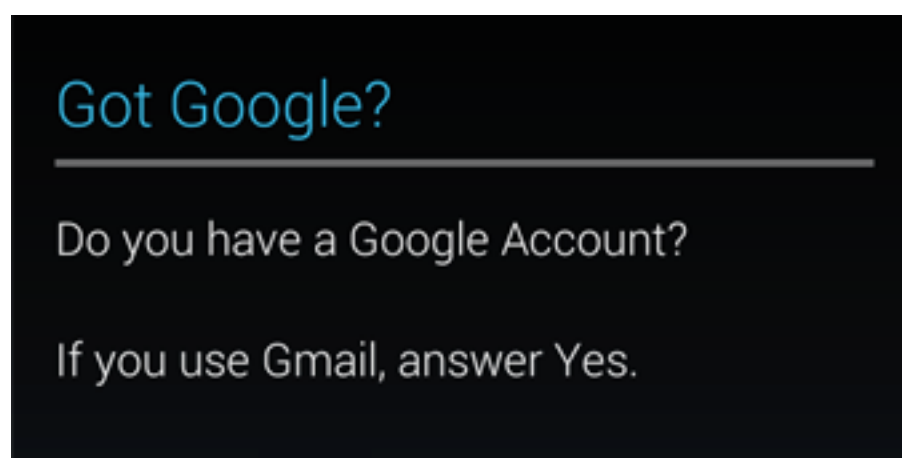
# Lecture Summary

- Design Principles
- Material Design
- Linear Layout
- Relative Layout
- Frame Layout
- Fragments



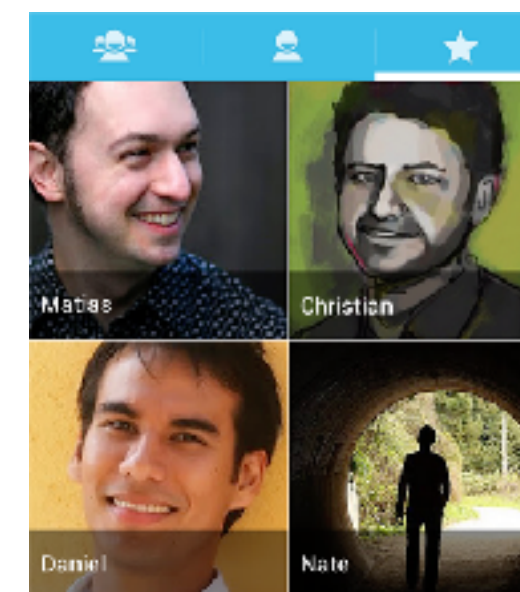
# Official Design Principles

## *Keep it brief*



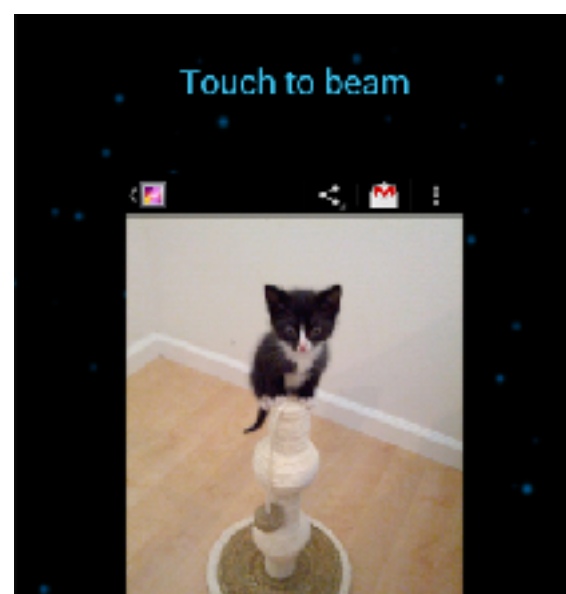
“Use short phrases with **simple words**”

## *Pictures are faster than words*



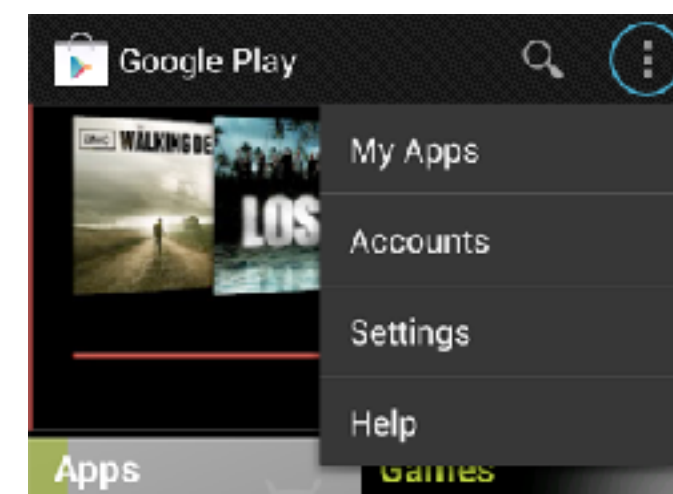
“Consider using **pictures to explain ideas**. They get people's attention and can be much more efficient than words.”

## *Decide for me but let me have the final say*



“Take your best guess and act rather than asking first. **Too many choices and decisions make people unhappy**. Just in case you get it wrong, **allow for 'undo'**”.

## *Only show what I need when I need it*

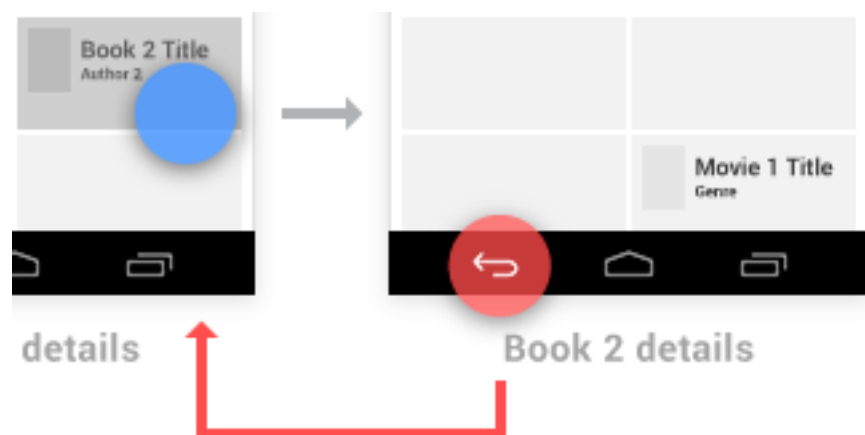


“People get overwhelmed when they see too much at once. Break tasks and information into small, digestible chunks. **Hide options that aren't essential at the moment, and teach people as they go.**”

Source: <https://developer.android.com/design/get-started/principles.html>

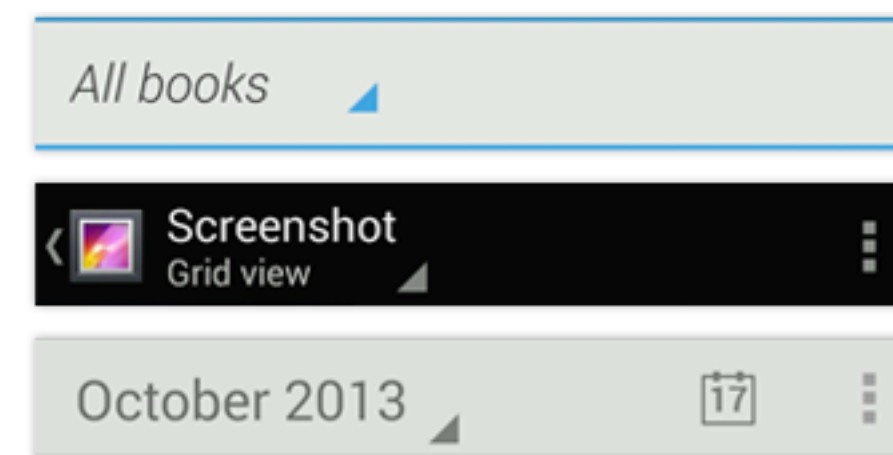
# Official Design Principles

*I should always know where I am*



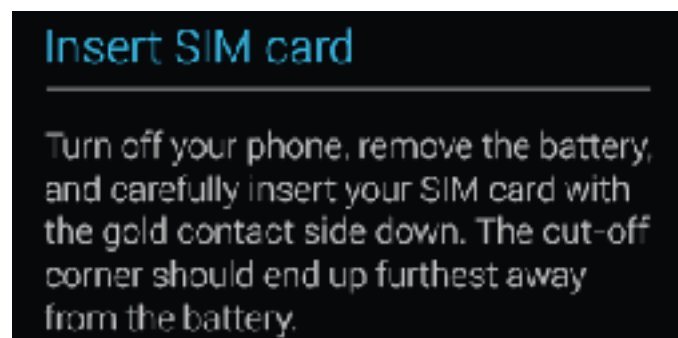
“Give people confidence that they know their way around. Make places in your app look distinct and use transitions to show relationships among screens. Provide feedback on tasks in progress.”

*If it looks the same, it should act the same*



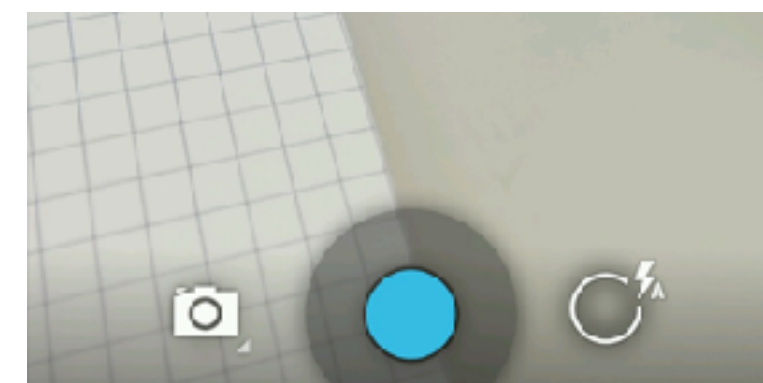
“Avoid modes, which are places that look similar but act differently on the same input.”

*It's not my fault*



“Be gentle in how you prompt people to make corrections. **They want to feel smart when they use your app.** If something goes wrong, give clear recovery instructions but spare them the technical details. If you can fix it behind the scenes, even better.”

*Make important things fast*

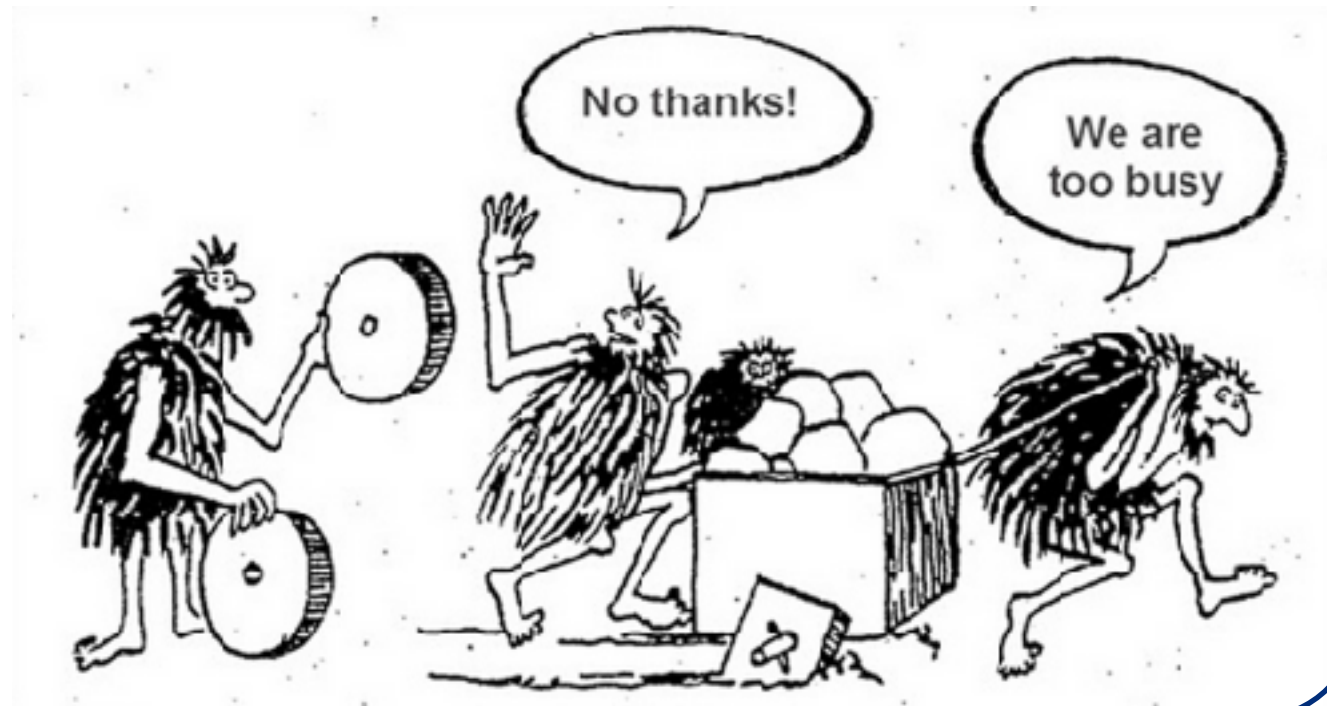


“**Not all actions are equal.** Decide what's most important in your app and make it easy to find and fast to use.”

Source: <https://developer.android.com/design/get-started/principles.html>

# Additional Design Principles

*Do not reinvent the wheel !*

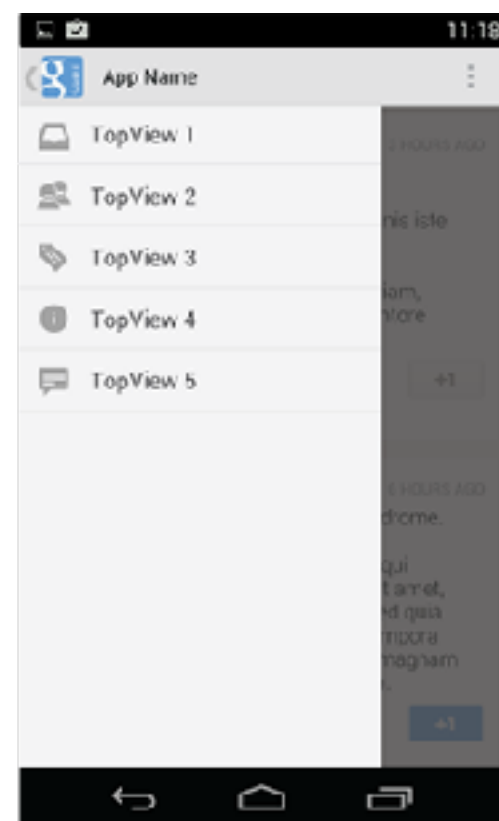
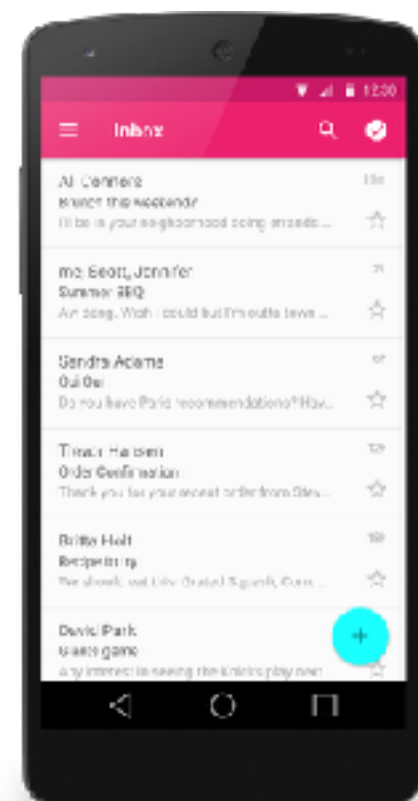
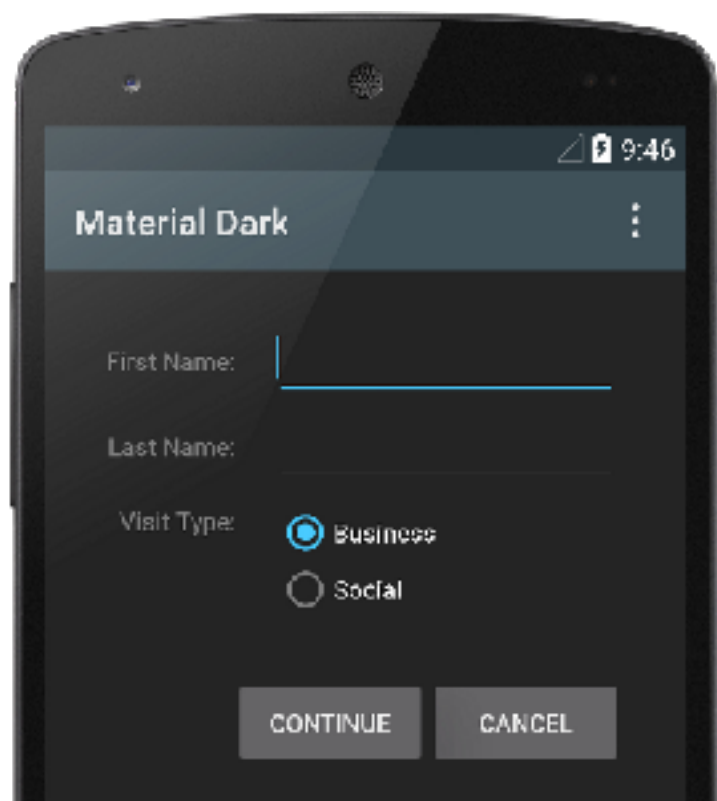


**You're not alone !**

At any given moment, somewhere in the world someone face the same software/design problems.

You know you don't want to reinvent the wheel (or a flat tire), so look around and focus on relevant aspect of your application !

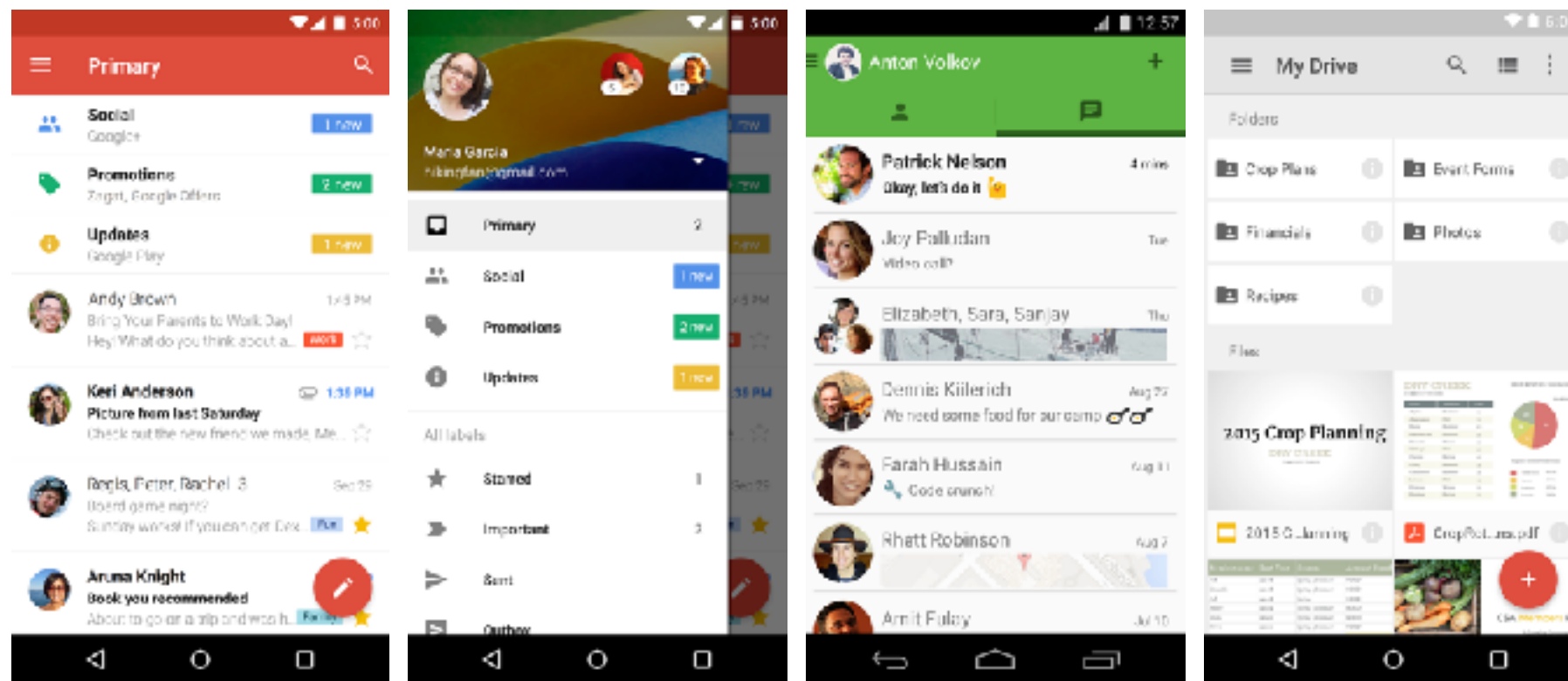
*Use native components and themes*



A rich and complete Mobile platform like Android provides many useful, tested and **well known native components**. You should use them during design and development phases.

# Additional Design Principles

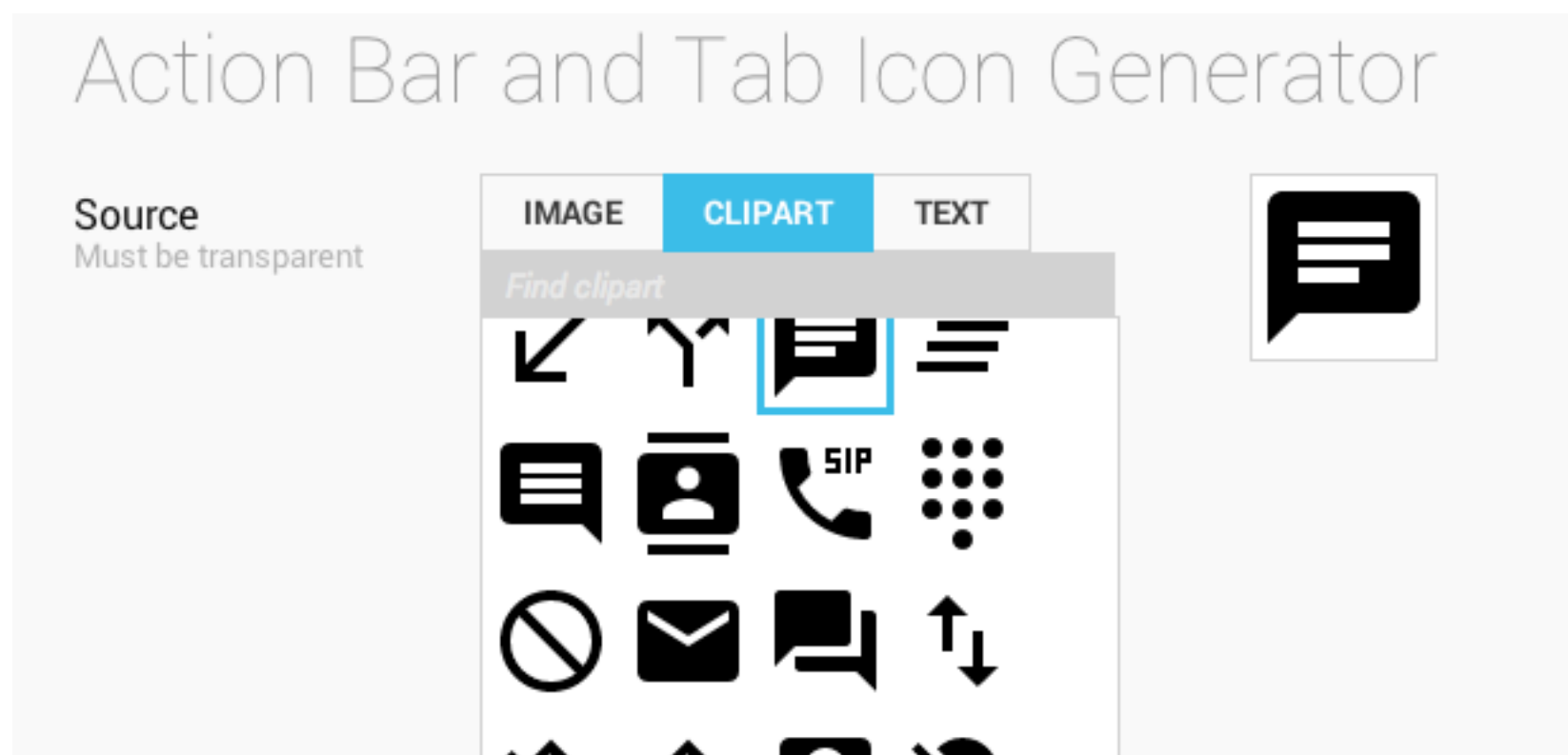
*Get Inspired ! ( Sometime Copy :) )*



Look around and learn from existing applications !

If you don't know how to design your application or a specific section, have a look at well known and official applications in order to understand how they are designed and composed.

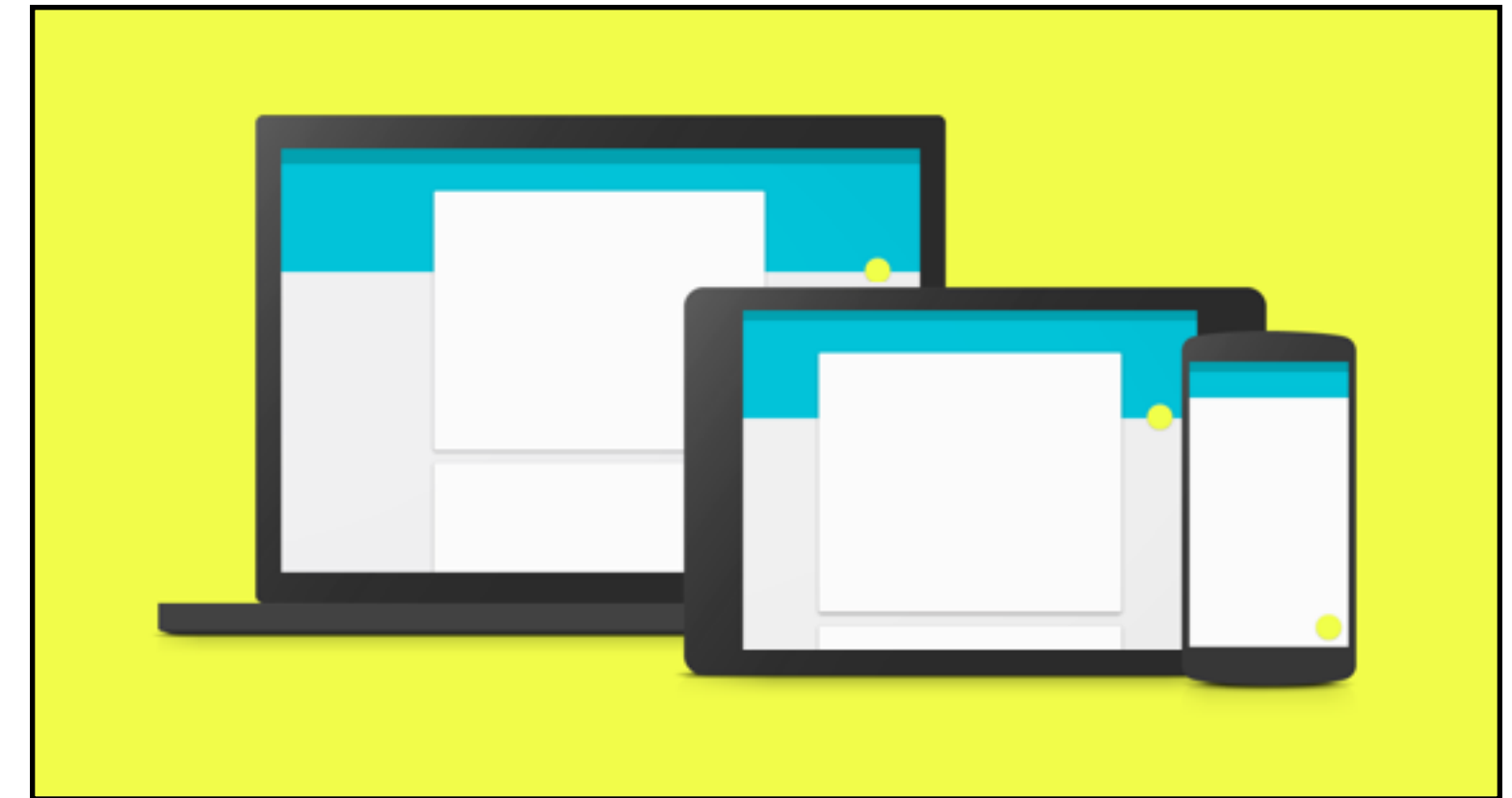
*Use online free tools & resources*



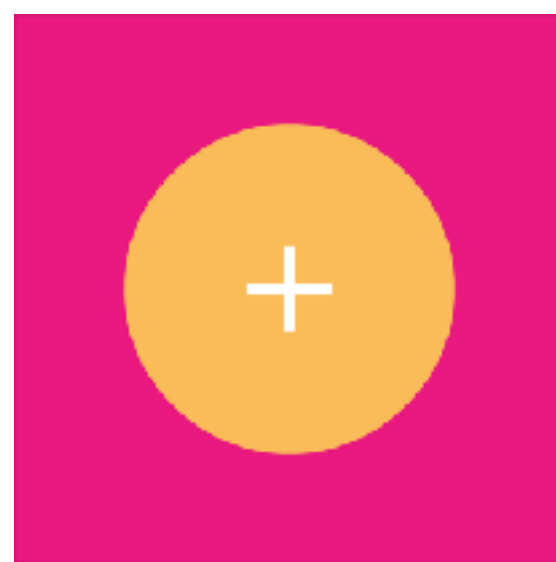
If you are not a designer, rely on existing tools and resources to improve and simplify the User Interface of your application.  
Keep it simple and clear !

# Material Design

- Material design is a comprehensive guide for visual, motion, and interaction design across platforms and devices.”
- Create a visual language that synthesizes classic principles of good design with the innovation and possibility of technology and science.”
- Develop a single underlying system that allows for a unified experience across platforms and device sizes. Mobile precepts are fundamental, but touch, voice, mouse, and keyboard are all first-class input methods.



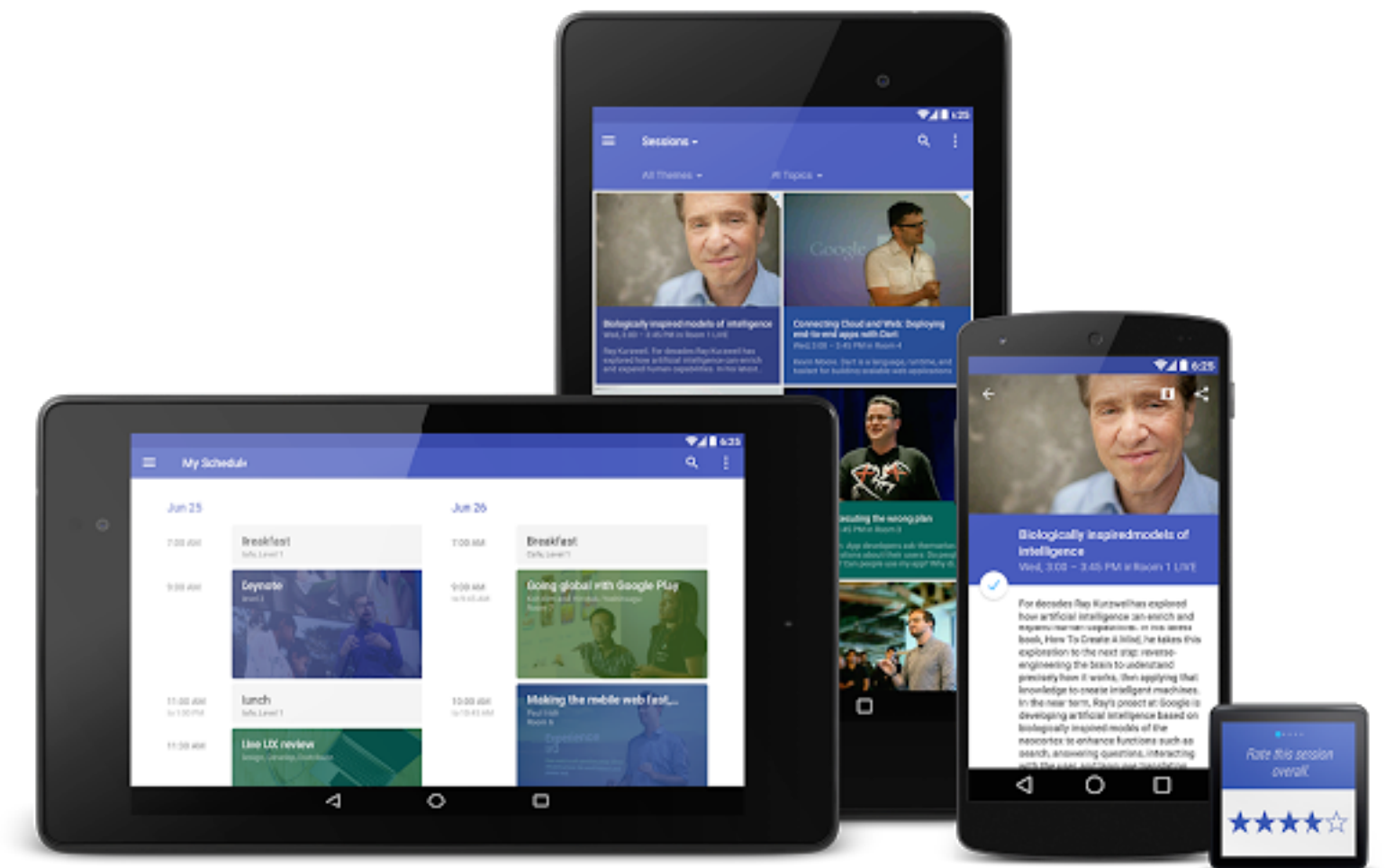
“Material is the metaphor”



“Bold, graphic, intentional”



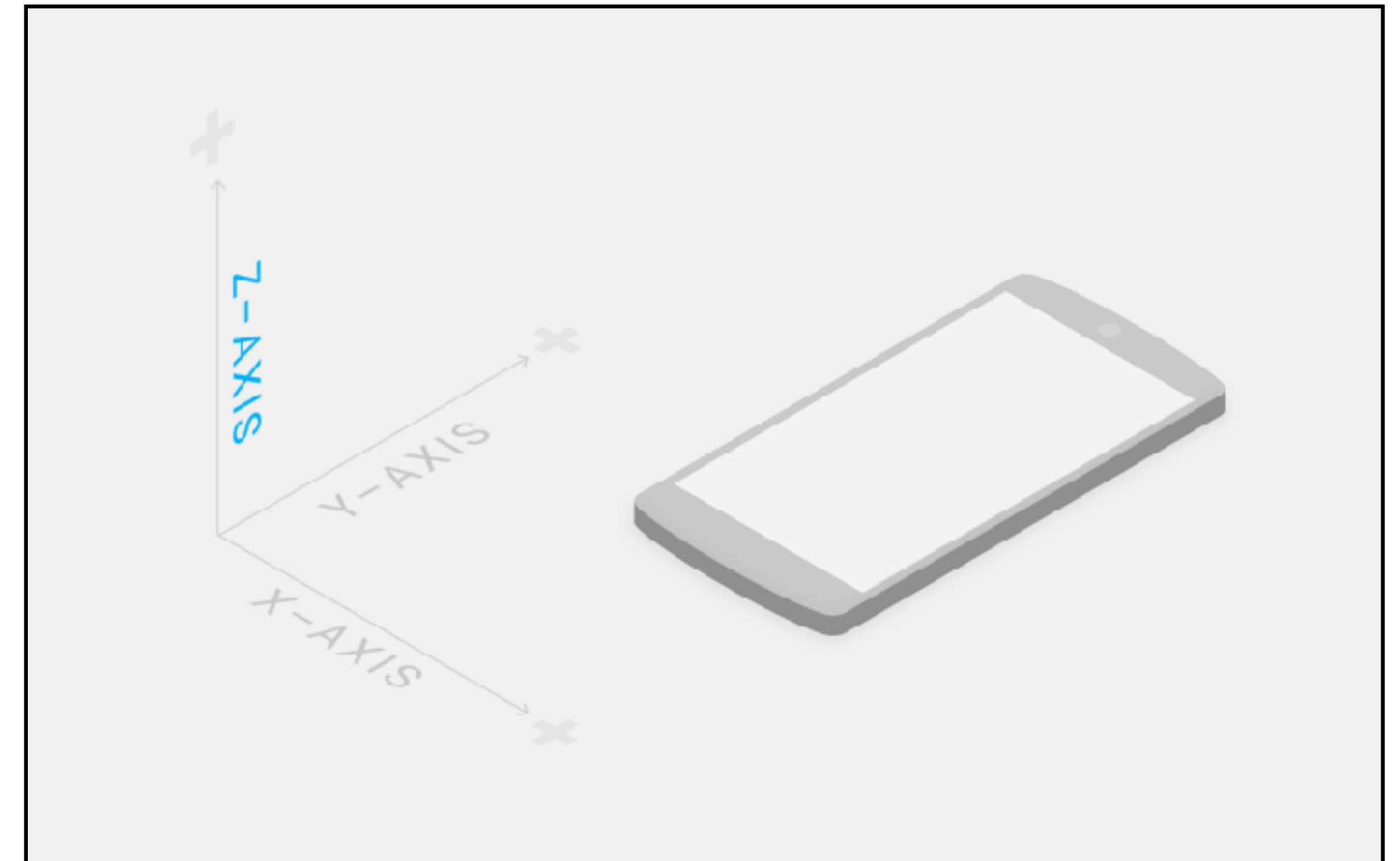
“Motion provides meaning”



Source: <http://www.google.com/design/spec/material-design/>

# Material Design - Environment

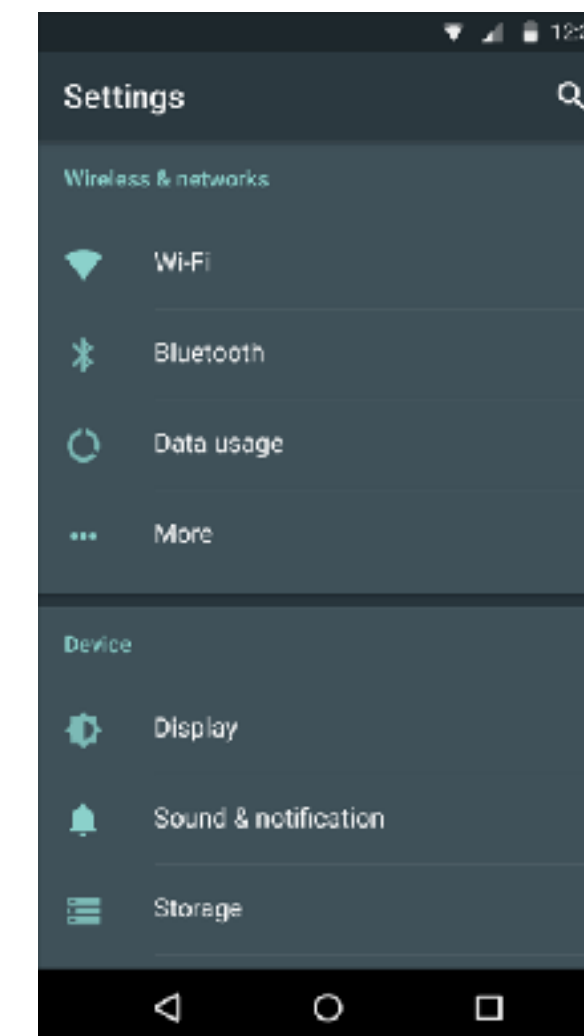
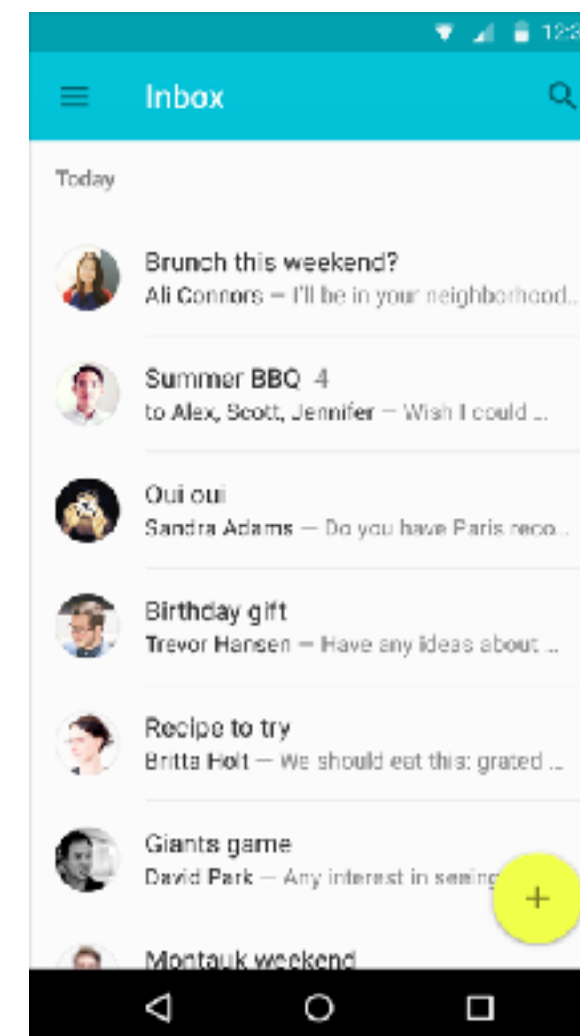
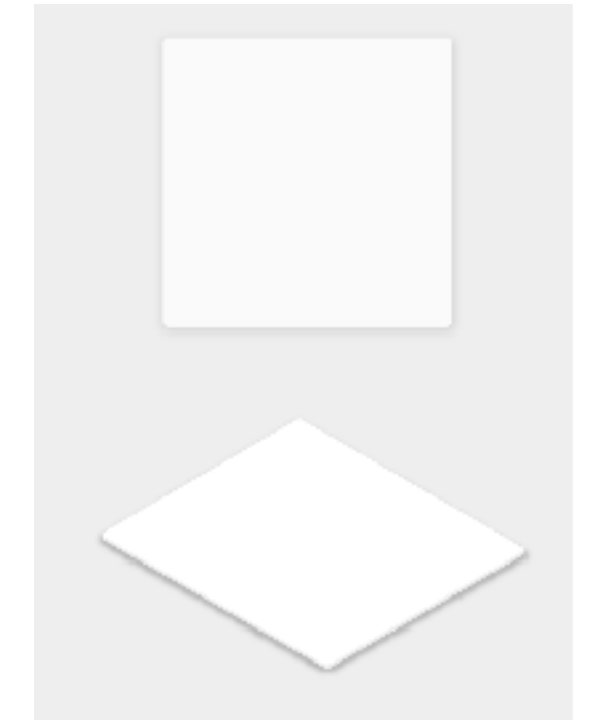
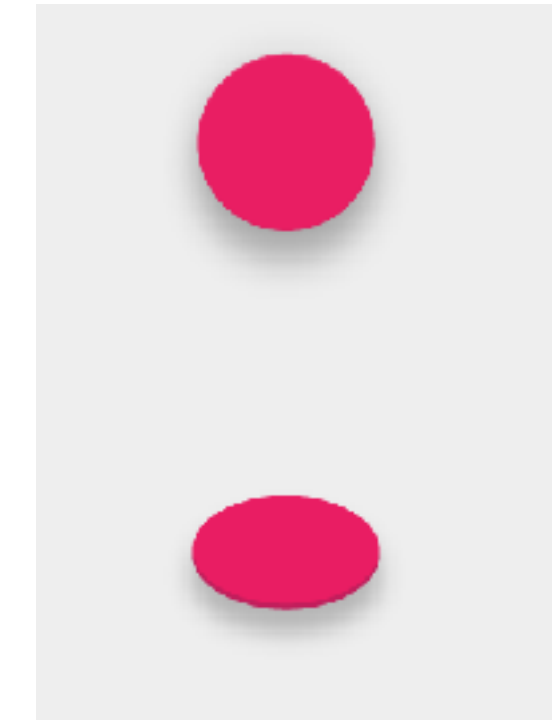
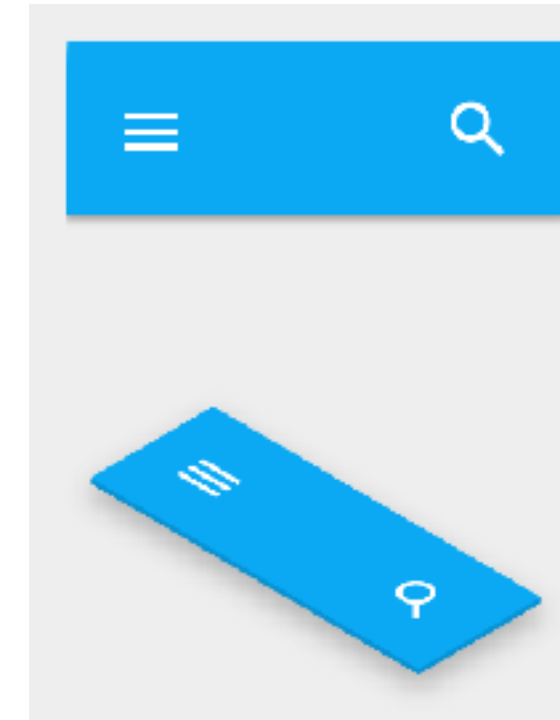
- The material environment is a 3D space
- All objects have x, y, and z dimensions
- The z-axis is perpendicularly aligned to the plane of the display, with the positive z-axis extending towards the viewer.
- Every sheet of material occupies a single position along the z-axis and has a standard 1dp thickness.
- Within the material environment, virtual lights illuminate the scene and allow objects to cast shadows.
- A key light creates directional shadows, while an ambient light creates consistent, soft shadows from all angles.



Source: <http://www.google.com/design/spec/material-design/>

# Material Design Specifications

- Material design defines every aspect of your application design:
  - Style (Color, Icons, Imagery, Typography)
  - Layout
  - Components
  - Patterns (Data format, errors, gestures, etc ...)
  - Usability
  - Animation

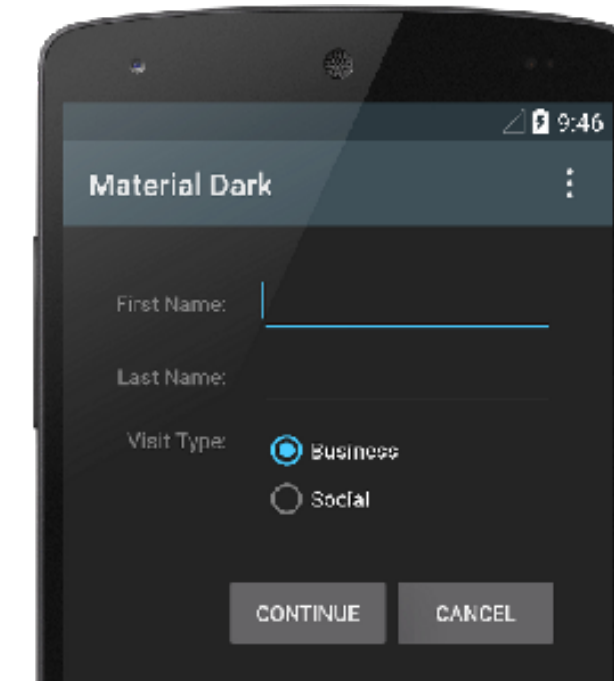


| Indigo       | Blue         |
|--------------|--------------|
| 500 #3F51B5  | 500 #2196F3  |
| 50 #E8EAF6   | 50 #E3F2FD   |
| 100 #C5CAE9  | 100 #BBDEFB  |
| 200 #9FA8DA  | 200 #90CAF9  |
| 300 #7986CD  | 300 #64B5F6  |
| 400 #5C6BC0  | 400 #42A5F5  |
| 500 #3F51B5  | 500 #2196F3  |
| 600 #3949AB  | 600 #1E88E5  |
| 700 #303F9F  | 700 #1976D2  |
| 800 #283593  | 800 #1565C0  |
| 900 #1A237E  | 900 #0D47A1  |
| A100 #8C9EFF | A100 #82B1FF |
| A200 #536DFE | A200 #448AFF |
| A400 #305AFE | A400 #2070FF |
| A700 #304FFE | A700 #2952FF |

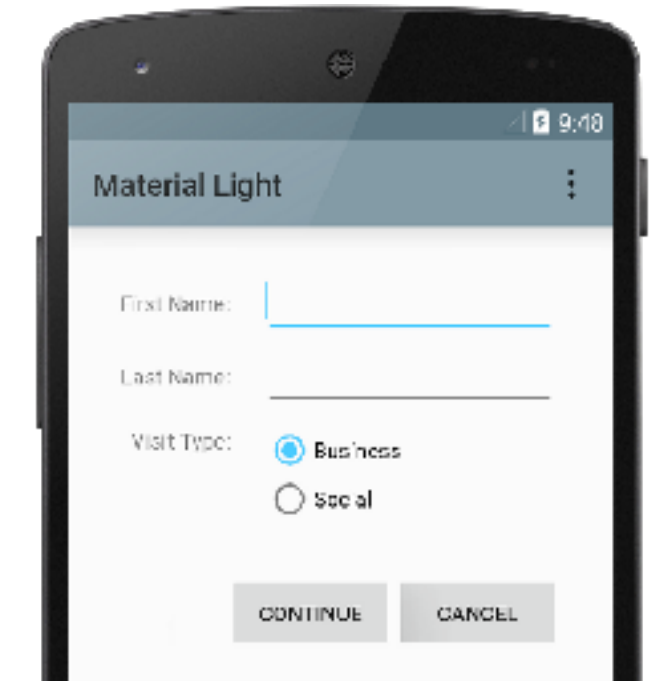
Source: <http://www.google.com/design/spec/material-design/>

# Material Design - Main Components and Features

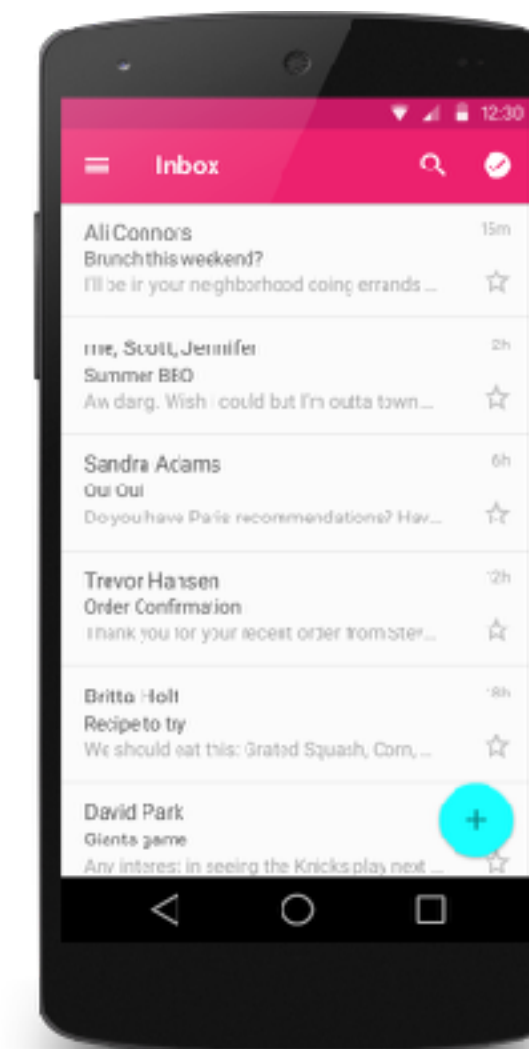
- Android provides the following elements for you to build material design apps:
- A new theme with the redesign and adaptation of existing Android key building blocks and patterns such as:
  - ActionBar/ToolBar
  - Tabs & ViewPager
  - Navigation Drawer
- New widgets for complex views
  - Lists
  - Cards
- New APIs for custom shadows and animations



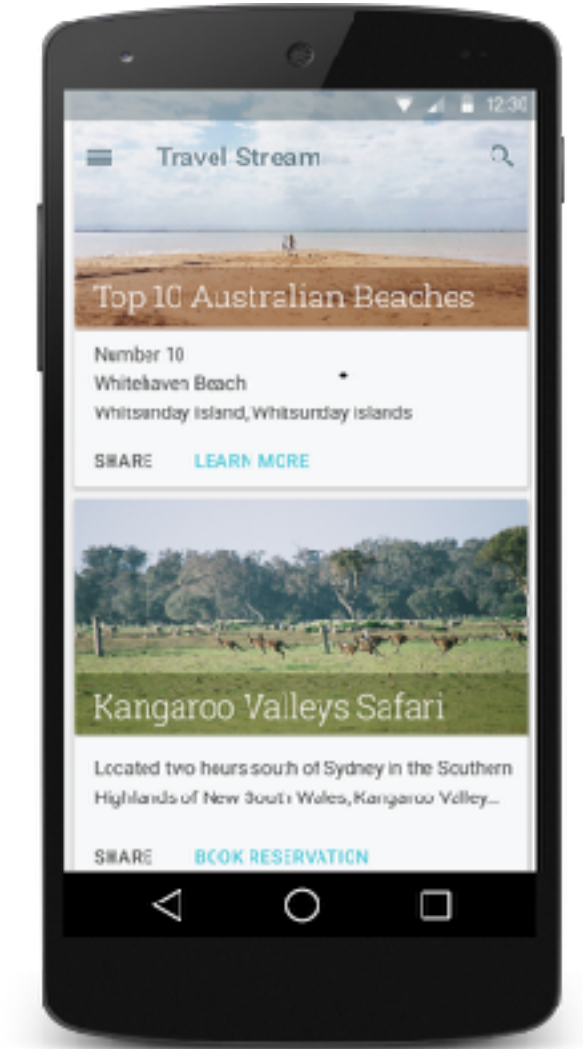
Dark Material Theme



Light Material Theme



Lists

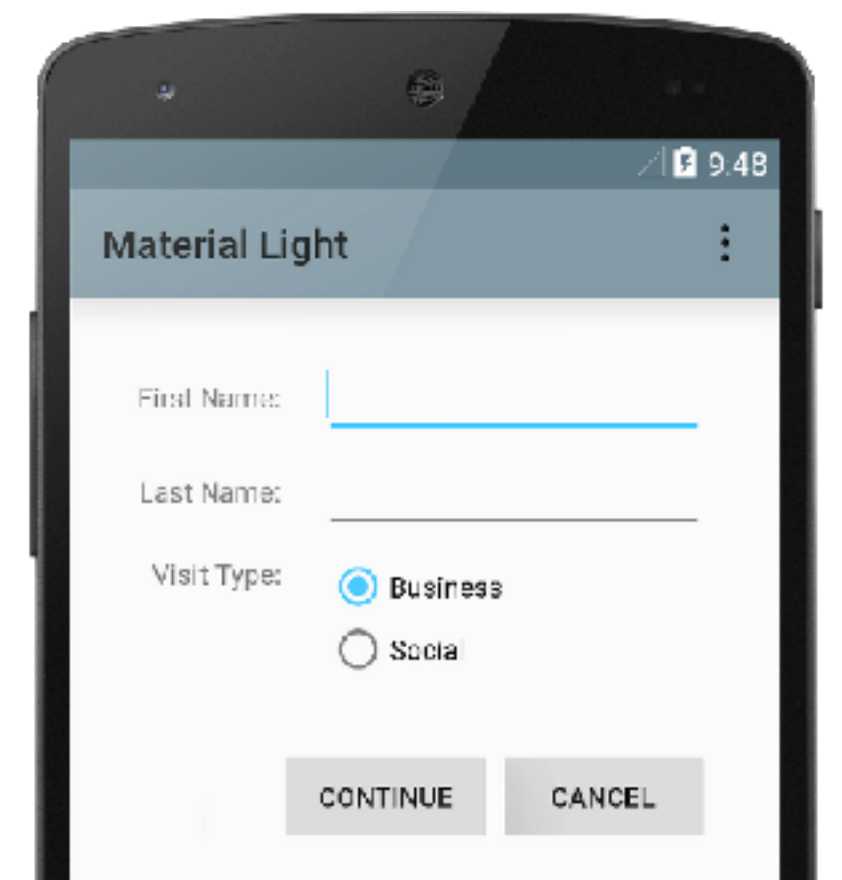
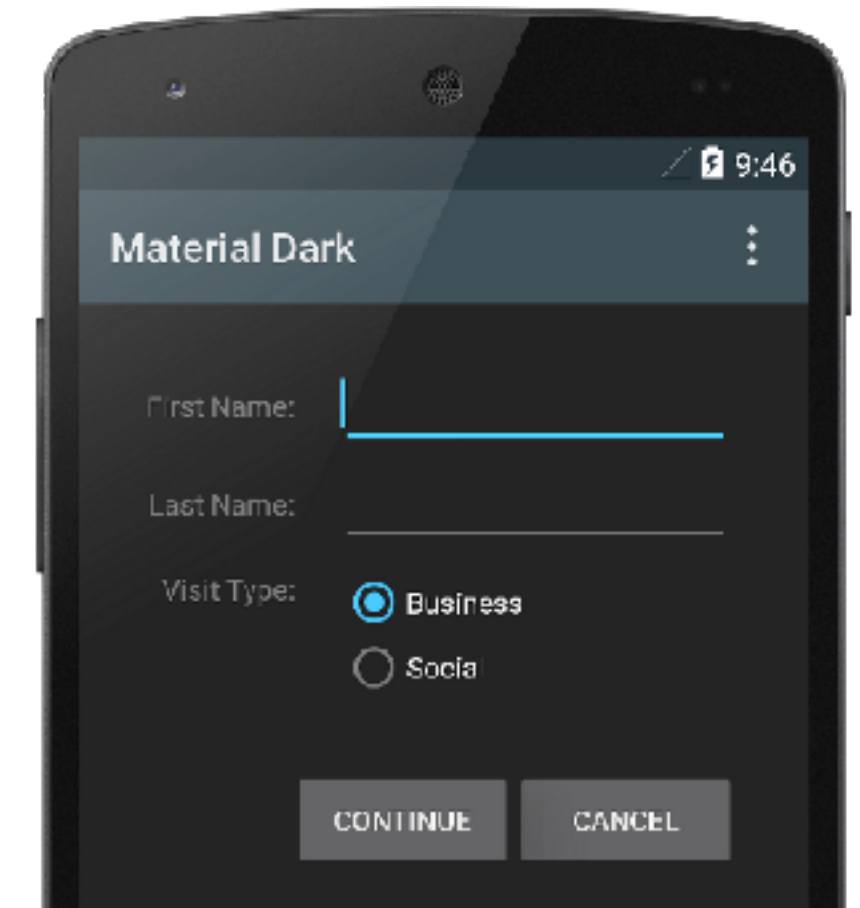


Cards

# Setup Material Design

- The material theme is defined as:
  - `@android:style/Theme.Material` (dark version)
  - `@android:style/Theme.Material.Light` (light version)
  - `@android:style/Theme.Material.Light.DarkActionBar`
- The material theme is only available in Android 5.0 (API level 21) and above.
- In order to include and use themes with material design styles on previous Android version you should use the **v7 Support Libraries**.

Dark Material Theme

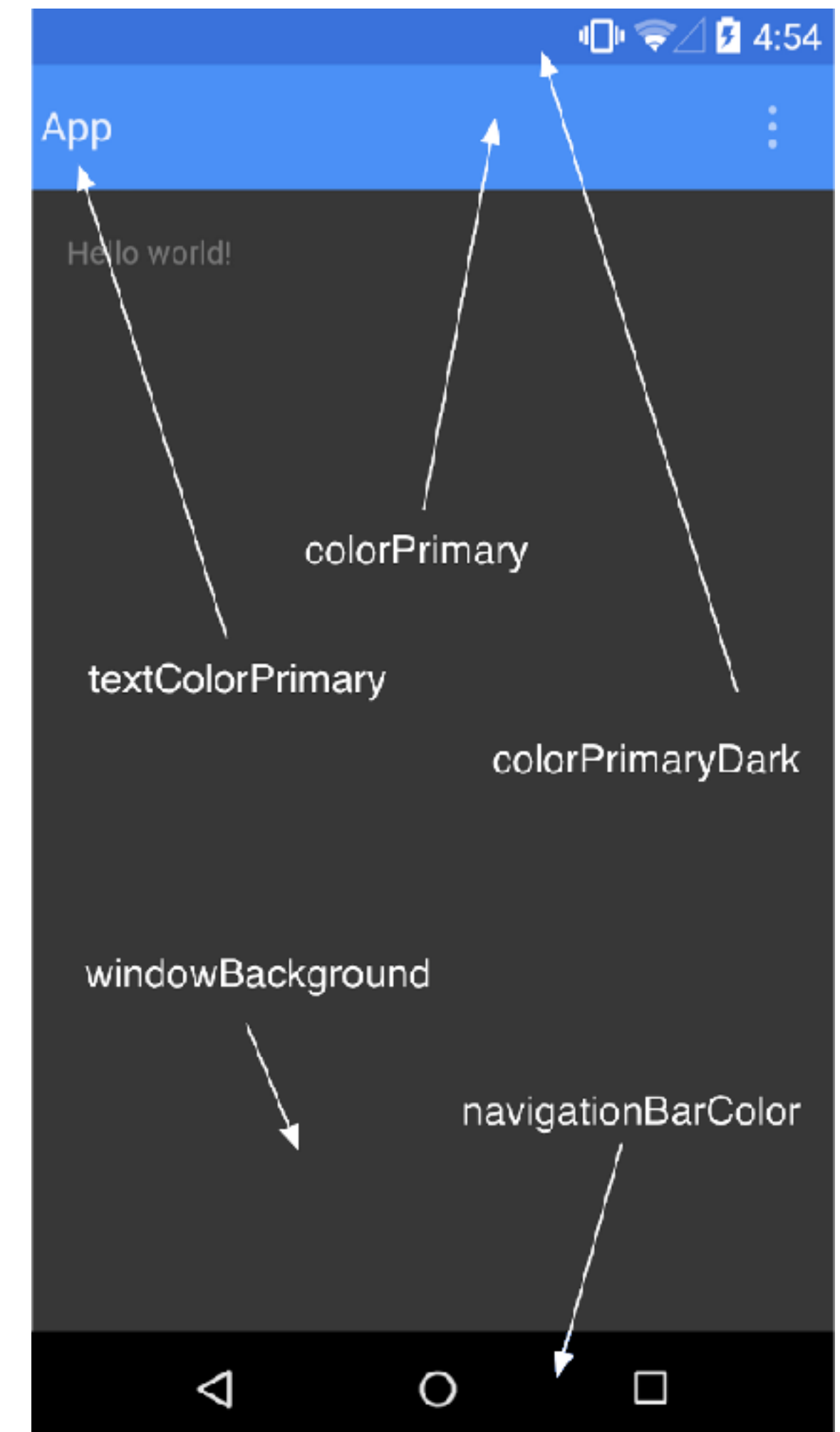


Light Material Theme

# Customize your Color Palette

- You can customize the theme's base colors to fit your brand or application preferences.
- You can specify your own custom colours using theme attributes in the **style.xml** file or directly a new theme inherited from material in the **themes.xml** file.

```
<resources>
  <!-- inherit from the material theme -->
  <style name="AppTheme" parent="android:Theme.Material">
    <!-- Main theme colors -->
    <!-- your app branding color for the app bar -->
    <item name="android:colorPrimary">@color/primary</item>
    <!-- darker variant for the status bar and contextual app bars -->
    <item name="android:colorPrimaryDark">@color/primary_dark</item>
    <!-- theme UI controls like checkboxes and text fields -->
    <item name="android:colorAccent">@color/accent</item>
  </style>
</resources>
```



# Setup Material Design - Maintaining Compatibility

- To enable you to bring your latest designs to older Android platforms Google has expanded the v7 Support Libraries (r21), including a major update to **AppCompat**, as well as new **RecyclerView**, **CardView** and **Palette** libraries.
- AppCompat was a backport of the Android 4.0 ActionBar API for devices running on Gingerbread, providing a common API layer on top of the backported implementation and the framework implementation.
- AppCompat v21 delivers an API and feature-set that is up-to-date with Android 5.0
- You can configure your app to use the material theme by:
  - Defining a theme that inherits from an older theme (like Holo) in **res/values/styles.xml**
  - Defining a theme with the same name that inherits from the material theme in **res/values-v21/styles.xml**
  - Setting this theme as your app's theme in the manifest file
- Lollipop transitions are not backwards compatible, so at the moment you can not be able to see smooth transitions in pre-21 devices.

Source: <http://android-developers.blogspot.it/2014/10/appcompat-v21-material-design-for-pre.html>

Source: <http://antonioleiva.com/material-design-everywhere/> Source: <https://chris.banes.me/2014/10/17/appcompat-v21/>

# Setup Material Design - Maintaining Compatibility

## (res/values/style.xml)

```
<resources>

    <!--
        Base application theme, dependent on API level. This theme is replaced
        by AppBaseTheme from res/values-vXX/styles.xml on newer devices.
    -->
    <style name="AppBaseTheme" parent="Theme.AppCompat.Light">
        <!--
            Theme customizations available in newer API levels can go in
            res/values-vXX/styles.xml, while customizations related to
            backward-compatibility can go here.
        -->
    </style>

</resources>
```

# Setup Material Design - Maintaining Compatibility

## (res/values/themes.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="AppTheme" parent="AppTheme.Base"/>

    <style name="AppTheme.Base" parent="Theme.AppCompat.Light">
        <item name="colorPrimary">@color/primary</item>
        <item name="colorPrimaryDark">@color/primary_dark</item>
        <item name="android:textColorPrimary">@color/blue_grey_100</item>
        <item name="android:windowNoTitle">true</item>
        <item name="windowActionBar">true</item>
    </style>
</resources>
```

Source: <http://www.murrayc.com/permalink/2014/10/28/android-changing-the-toolbars-text-color-and-overflow-icon-color/>

# Setup Material Design - Maintaining Compatibility

## (res/values-21/themes.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
<style name="AppTheme" parent="AppTheme.Base">
    <item name="android:windowContentTransitions">true</item>
    <item name="android:windowAllowEnterTransitionOverlap">true</item>
    <item name="android:windowAllowReturnTransitionOverlap">true</item>
    <item name="android:windowSharedElementEnterTransition">@android:transition/move</item>
    <item name="android:windowSharedElementExitTransition">@android:transition/move</item>
</style>
</resources>
```

# Setup Material Design - Maintaining Compatibility

## (/AndroidManifest.xml)

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.mobdev.hellomaterial"
    android:versionCode="1"
    android:versionName="1.0" >

    [...]

    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme" >
        <activity
            android:name=".MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```

# Setup Material Design - Some Colors :)

## (/res/values/colors.xml)

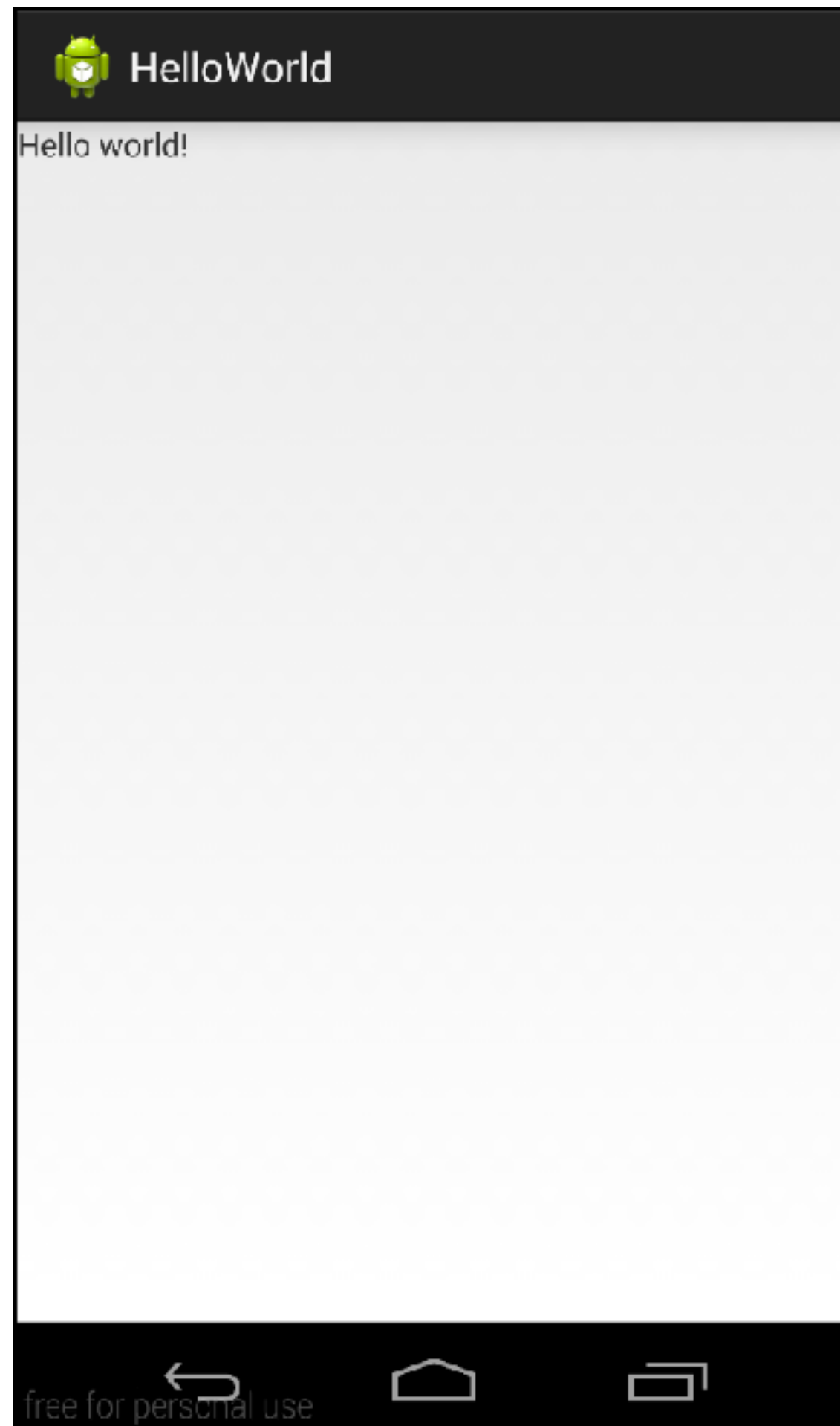
```
<?xml version="1.0" encoding="utf-8"?>

<resources>
    <color name="red_50">#fde0dc</color>
    <color name="red_100">#f9bdbb</color>
    <color name="red_200">#f69988</color>
    <color name="red_300">#f36c60</color>
    <color name="red_400">#e84e40</color>
    <color name="red_500">#e51c23</color>
    <color name="red_600">#dd191d</color>
    <color name="red_700">#d01716</color>
    <color name="red_800">#c41411</color>
    <color name="red_900">#b0120a</color>
    <color name="red_A100">#ff7997</color>
    <color name="red_A200">#ff5177</color>
    <color name="red_A400">#ff2d6f</color>
    <color name="red_A700">#e00032</color>

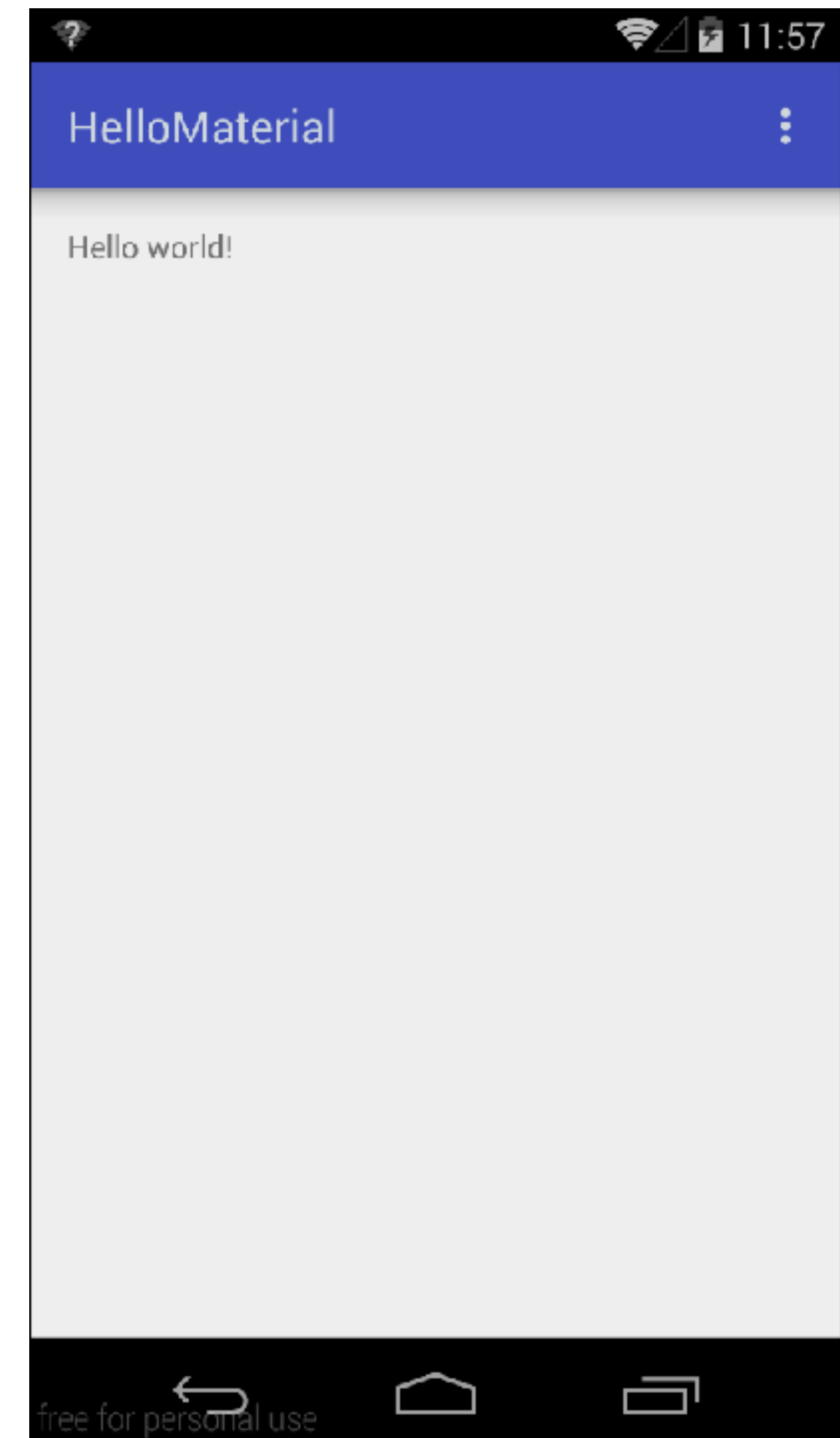
    [...]

</resources>
```

# Hello Material ! :)



styles.xml  
themes.xml  
colors.xml  
AndroidManifest.xml



# Activity

- An Activity is both a unit of user interaction (typically associated to a View) and a unit of execution.
- Inside the override method `onCreate(...)` the developer can set the layout View associated to the Activity calling `setContentView(R.layout.<layout_file>)` using the shared R object to specify the resource file containing the layout.
- When the root view has been set it is possible to retrieve objects associated to the UI elements through the method `findViewById(R.id.<resource_id>)`.
- A View object like `TextView` can be used to update the user interface, to get values from input fields or to set Event listeners using provided get and set methods.

## View/Layout

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

    <TextView
        android:id="@+id/mainTextView"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/hello" />

</LinearLayout>
```

## Controller

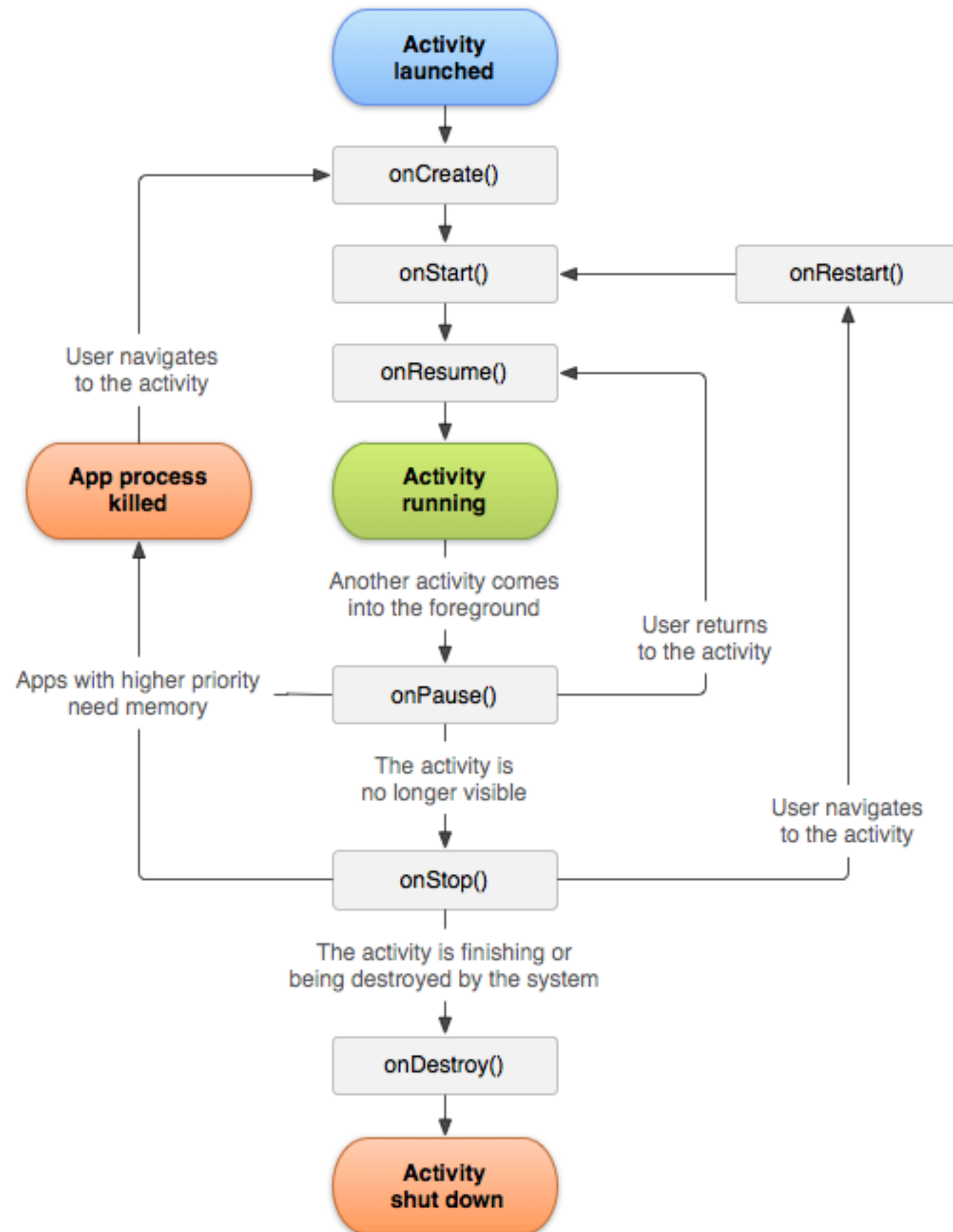
```
public class HelloWorldActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

Binding between Layout (View) and Activity (Controller)

UI Element access and modification

```
TextView mainTextView = (TextView)findViewById(R.id.mainTextView);
mainTextView.setText("Hello World ! :D");
```

# Activity Life Cycle



| Method                   | Description                                                                                                                                                                                                                                                                                                                        | Killable after? | Next                                                 |
|--------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------|------------------------------------------------------|
| <code>onCreate()</code>  | Called when the activity is first created. This is where you should do all of your normal static set up – create views, bind data to lists, and so on.                                                                                                                                                                             | No              | <code>onStart()</code>                               |
| <code>onRestart()</code> | Called after the activity has been stopped, just prior to it being started again.                                                                                                                                                                                                                                                  | No              | <code>onStart()</code>                               |
| <code>onStart()</code>   | Called just before the activity becomes visible to the user.                                                                                                                                                                                                                                                                       |                 | <code>onResume()</code> or <code>onStop()</code>     |
| <code>onResume()</code>  | Called just before the activity starts interacting with the user. At this point the activity is at the top of the activity stack, with user input going to it.                                                                                                                                                                     | No              | <code>onPause()</code>                               |
| <code>onPause()</code>   | Called when the system is about to start resuming another activity. This method is typically used to commit unsaved changes to persistent data, stop animations and other things that may be consuming CPU, and so on. It should do whatever it does very quickly, because the next activity will not be resumed until it returns. | Yes             | <code>onResume()</code> or <code>onStop()</code>     |
| <code>onStop()</code>    | Called when the activity is no longer visible to the user. This may happen because it is being destroyed, or because another activity (either an existing one or a new one) has been resumed and is covering it.                                                                                                                   | Yes             | <code>onRestart()</code> or <code>onDestroy()</code> |
| <code>onDestroy()</code> | Called before the activity is destroyed. This is the final call that the activity will receive. It could be called either because the activity is finishing, or because the system is temporarily destroying this instance of the activity to save space.                                                                          | Yes             | nothing                                              |

Source: <http://developer.android.com/guide/topics/fundamentals/activities.html>

# Your Activity Class

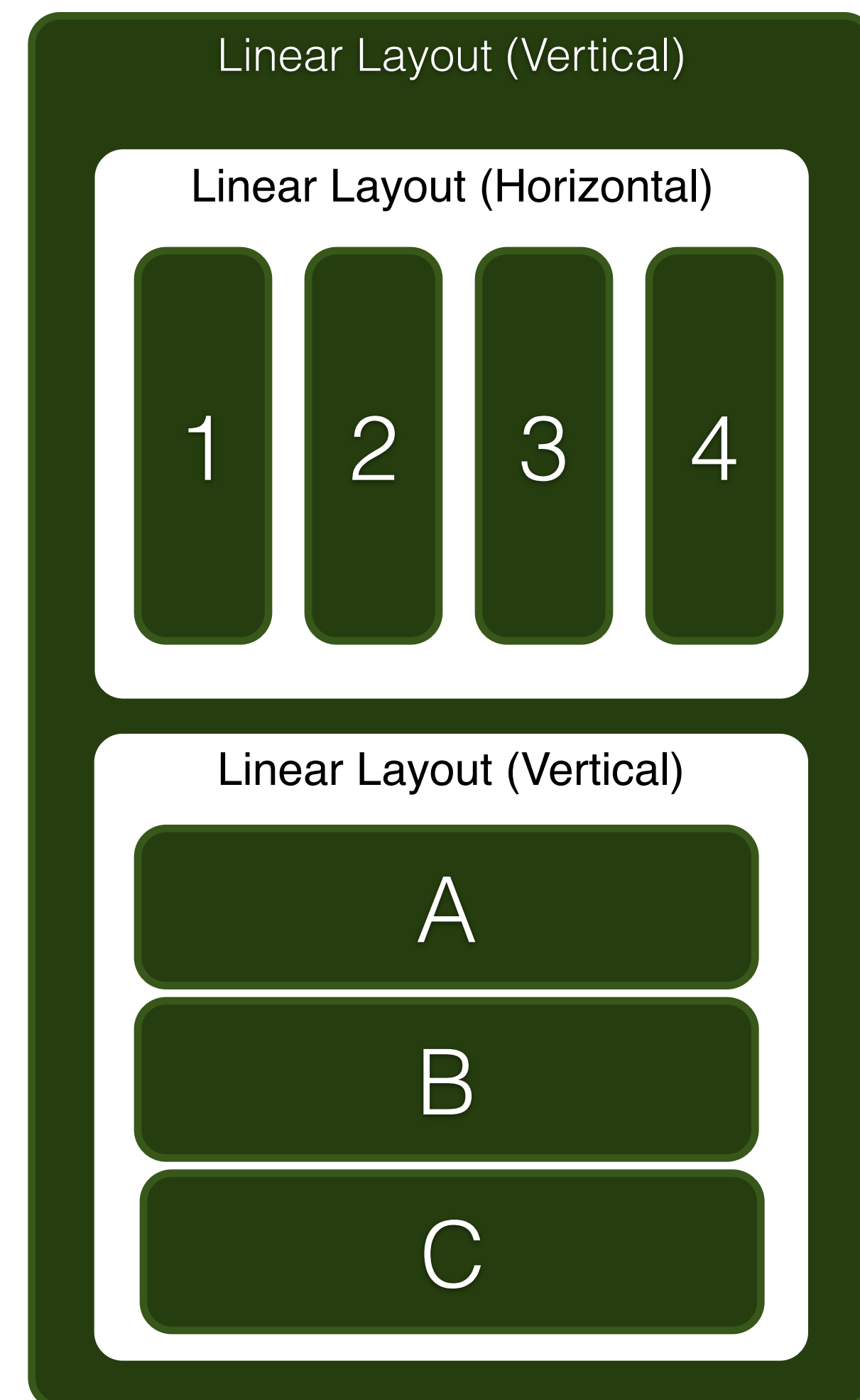
```
public class HelloWorldActivity extends Activity {  
    /** Called when the activity is first created. */  
    @Override  
    public void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.main);  
    }  
}
```

Set the Activity Layout



# Linear Layout

- A Layout that arranges its children in a single column or a single row.
- Available space is divided among layout children
- The direction of the row can be set by calling `setOrientation()` or through XML using “`android:orientation="vertical"`”.
- You can also specify gravity, which specifies the alignment of all the child elements by calling `setGravity()` or specify that specific children grow to fill up any remaining space in the layout by setting the weight member of `LinearLayout.LayoutParams`.
- The default orientation is horizontal.



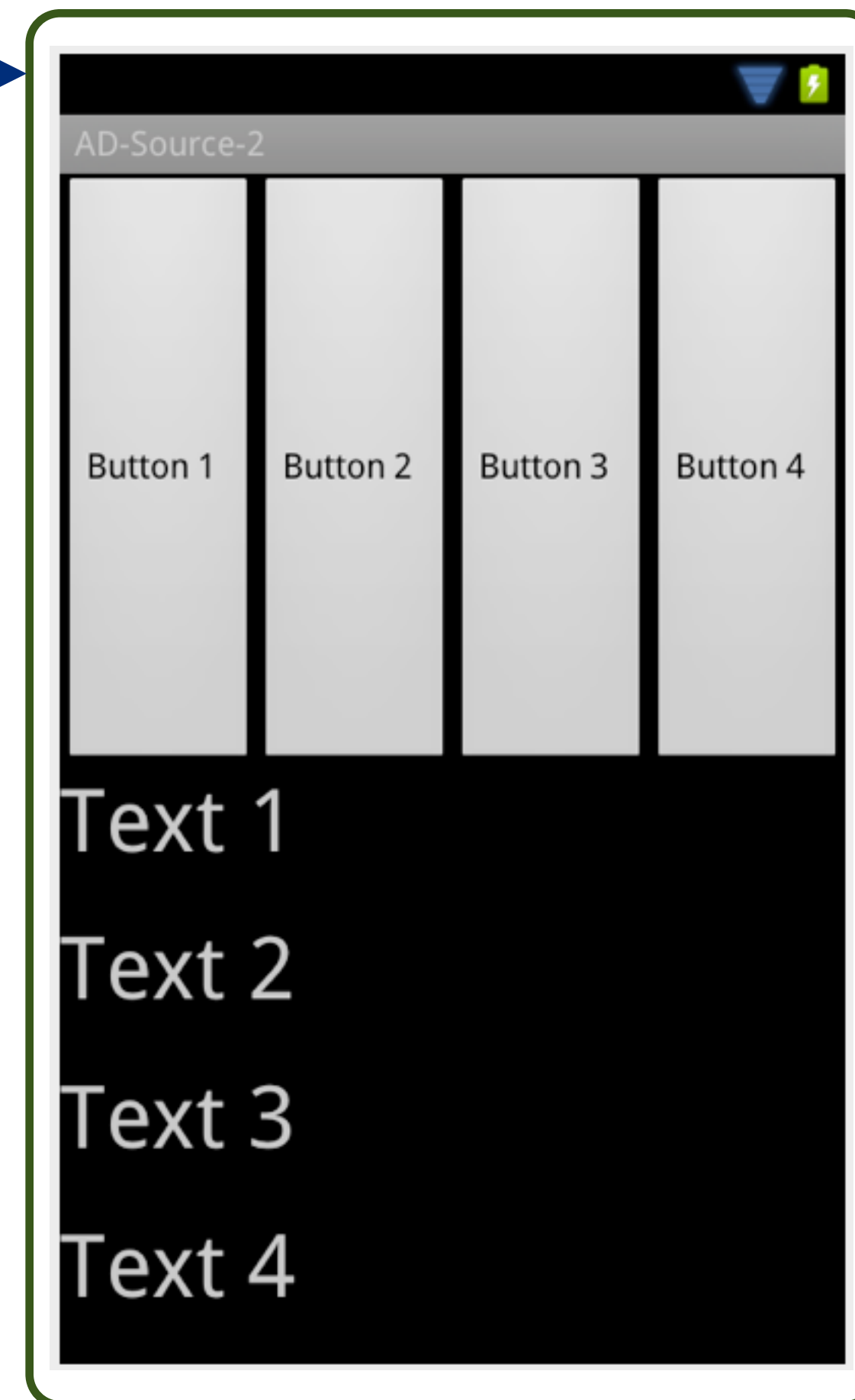
# Linear Layout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
```

```
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="1">
        <Button
            android:text="Button 1"
            android:gravity="center_horizontal"
            android:layout_width="wrap_content"
            android:layout_height="fill_parent"
            android:layout_weight="1"
        />
        <Button ... />
        <Button ... />
        <Button ... />
    </LinearLayout>
```

```
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:layout_weight="1">
        <TextView
            android:text="Text 1"
            android:textSize="15pt"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"/>
        <TextView ... />
        <TextView ... />
        <TextView ... />
        <TextView ... />
    </LinearLayout>
```

```
</LinearLayout>
```



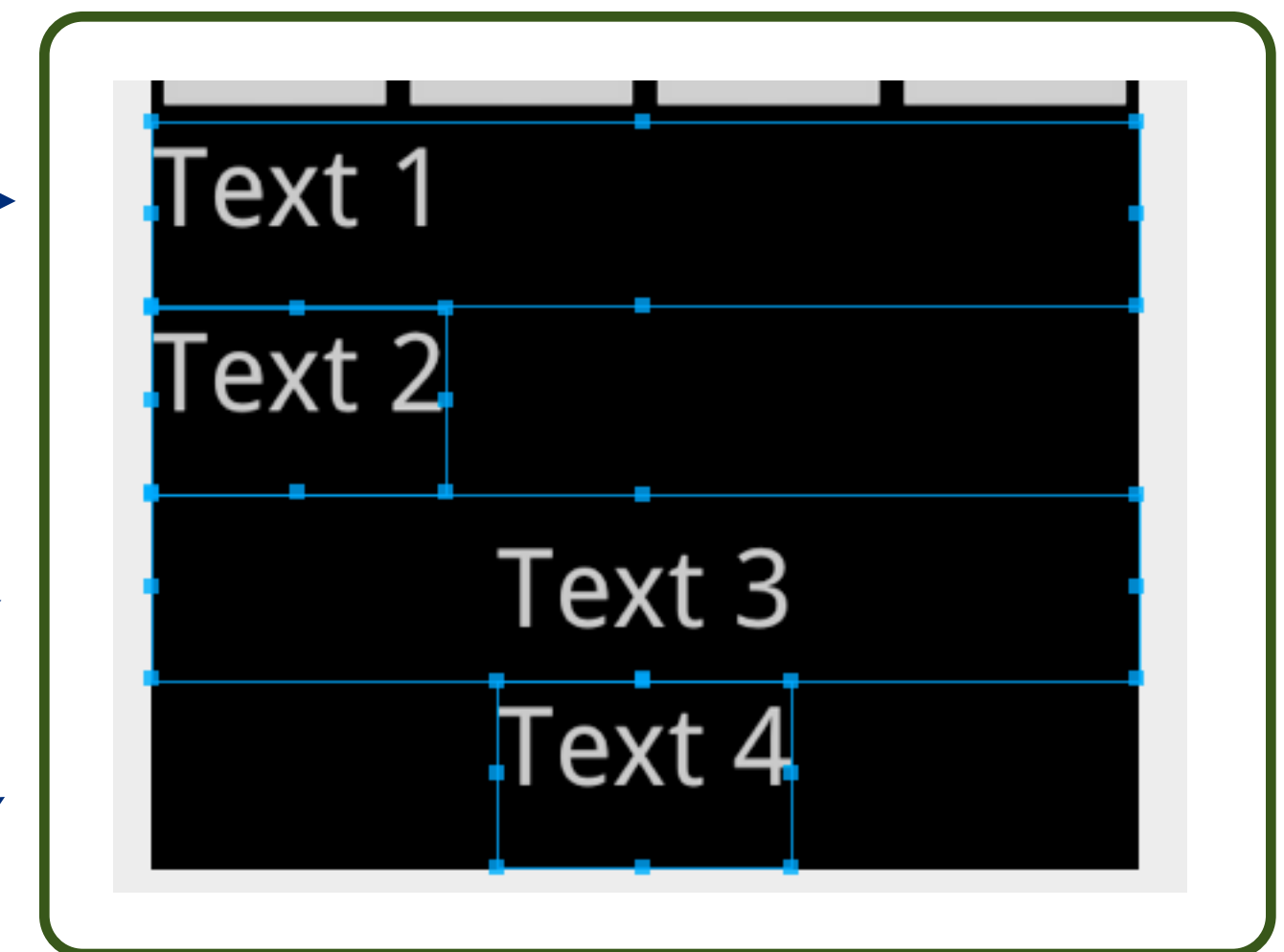
# Layout Parameters

```
<LinearLayout
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:layout_weight="1">
  <TextView
    android:text="Text 1"
    android:textSize="15pt"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_weight="1"/>
  <TextView
    android:text="Text 2"
    android:textSize="15pt"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_weight="1"/>
  <TextView
    android:text="Text 3"
    android:textSize="15pt"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:gravity="center"
    android:layout_weight="1"/>
  <TextView
    android:text="Text 4"
    android:textSize="15pt"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_weight="1"/>
</LinearLayout>
```

`android:layout_width="wrap_content"`

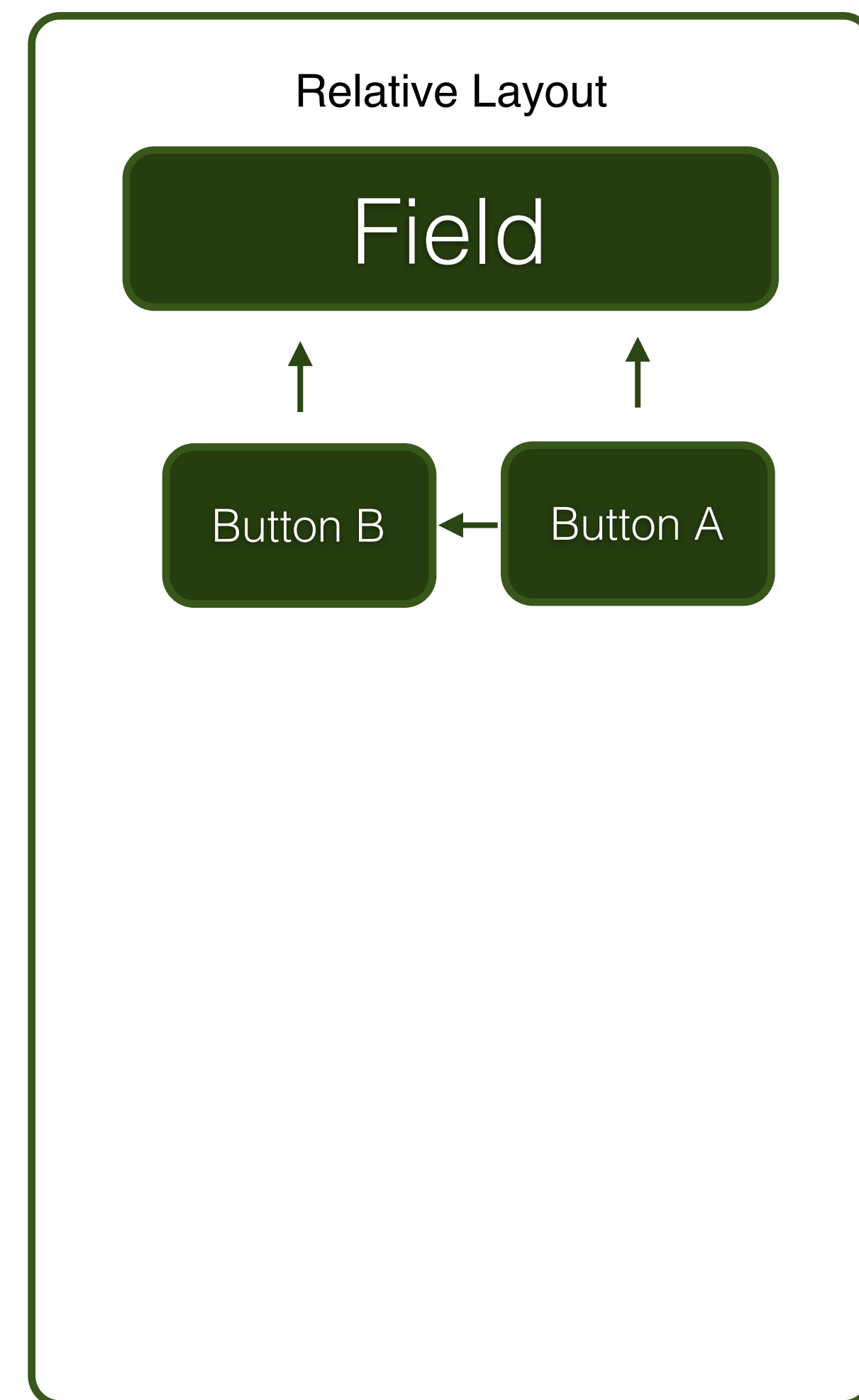
`android:gravity="center"`

`android:layout_width="wrap_content"`  
`android:layout_gravity="center"`

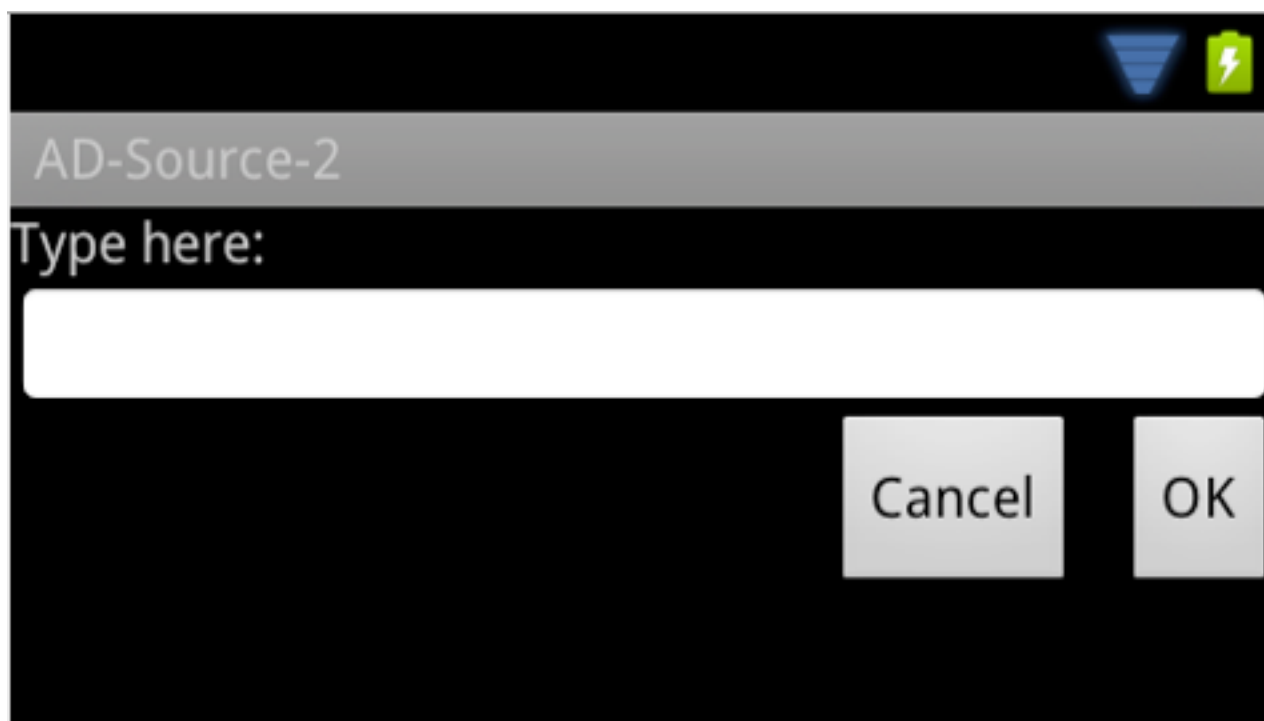


# Relative Layout

- RelativeLayout is a ViewGroup that displays child View elements in relative positions.
- The **position** of a View can be specified as **relative to sibling elements** (such as to the left-of or below a given element) or in positions relative to the RelativeLayout area (such as aligned to the bottom, left of center).
- A RelativeLayout is a very powerful utility for designing a user interface because it can eliminate nested ViewGroups. If you find yourself using several nested LinearLayout groups, you may be able to replace them with a single RelativeLayout.
- When using a RelativeLayout, you can use **android:layout\_\*** attributes, such as **layout\_below**, **layout\_alignParentRight**, and **layout\_toLeftOf** to describe how you want to position each View. Each one of these attributes define a different kind of relative position. Some attributes use the resource ID of a sibling View to define its own relative position.



# Relative Layout



`android:layout_below="@id/label"`

`android:layout_below="@id/entry"`  
`android:layout_alignParentRight="true"`

`android:layout_toLeftOf="@id/ok"`  
`android:layout_alignTop="@id/ok"`

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Type here:"/>
    <EditText
        android:id="@+id/entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/label"/>
    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
        android:text="OK" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="Cancel" />
</RelativeLayout>
```

# Frame Layout

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/container"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context="com.mobdev.hellofragment.MainActivity"
    tools:ignore="MergeRootFrame" />
```

- FrameLayout is designed to block out an area on the screen to display a **single item**.
- Generally, FrameLayout should be used to hold a single child view
- With multiple childrens could be used to overlap them to each other.
- It is really useful as generic container for a Fragment without the need to define multiple xml files. (Fragment will be introduced in the next slides)

# Button

<Button

```
    android:id="@+id/randomButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/infoTextView"  
    android:layout_centerHorizontal="true"  
    android:text="Click me to get a random number !"  
    android:textColor="@color/primary" />
```

```
//Retrieve a Button UI Object and set an inline listener  
Button randomButton = (Button)findViewById(R.id.randomButton);
```



Inside an Activity it means this.findViewById(...)

# Input Events

- Views/widgets and generally UI are used not only to show contents and information but have the important role to allow the user to interact with the application performing actions. To be informed of user input events, the developer needs to do one of two things:
  - **Define an event listener and register it with the View.** The View class contains a collection of nested interfaces named On<something>Listener, each with a callback method called On<something>().
    - For example, View.OnClickListener (for handling "clicks" on a View), View.OnTouchListener (for handling touch screen events in a View), and View.OnKeyListener (for handling device key presses within a View).
    - In order to be notified when the view is "clicked" (such as when a button is selected), implement OnClickListener and define its onClick() callback method (where you perform the action upon click), and register it to the View with setOnClickListener().
  - **Override an existing callback method for the View.** This is what you should do when you've implemented your own View class and want to listen for specific events that occur within it.

```
// Create an anonymous implementation of OnClickListener
private OnClickListener mCorkyListener = new OnClickListener() {
    public void onClick(View v) {
        // do something when the button is clicked
    }
};

protected void onCreate(Bundle savedInstanceState) {
    // Capture our button from layout
    Button button = (Button)findViewById(R.id.corky);
    // Register the onClick listener with the implementation above
    button.setOnClickListener(mCorkyListener);
}
```

# Button Events

```
public class MyActivity extends Activity {  
    protected void onCreate(Bundle icle) {  
        super.onCreate(icle);  
  
        setContentView(R.layout.content_layout_id);  
  
        final Button button = (Button) findViewById(R.id.button_id);  
        button.setOnClickListener(new View.OnClickListener() {  
            public void onClick(View v) {  
                // Perform action on click  
            }  
        });  
    }  
}
```

- Android is massively based (e.g. UListener, Location Management) on the principle/design patter of **Callback**.
- Callback methods are useful to handle events that can occur at an unspecified time (asynchronous events)
- A callback method is a method that is passed as an argument of another method to be later invoked
- In Java, Callback methods are implemented through **Interfaces**

Look and refer to the “Introduction to Java” Lecture and example source code available on the official website.



# ImageView

## <ImageView

```
android:id="@+id/imageView"  
android:layout_width="wrap_content"  
android:layout_height="wrap_content"  
android:layout_below="@+id/textView1"  
android:layout_centerHorizontal="true"  
android:layout_marginTop="14dp"  
android:src="@drawable/ic_action_android" />
```



```
ImageView imageView = (ImageView)findViewById(R.id.imageView);  
imageView.setImageDrawable(androidDrawable);  
imageView.setImageBitmap(bm);  
imageView.setImageResource(R.drawable.ic_action_android);
```

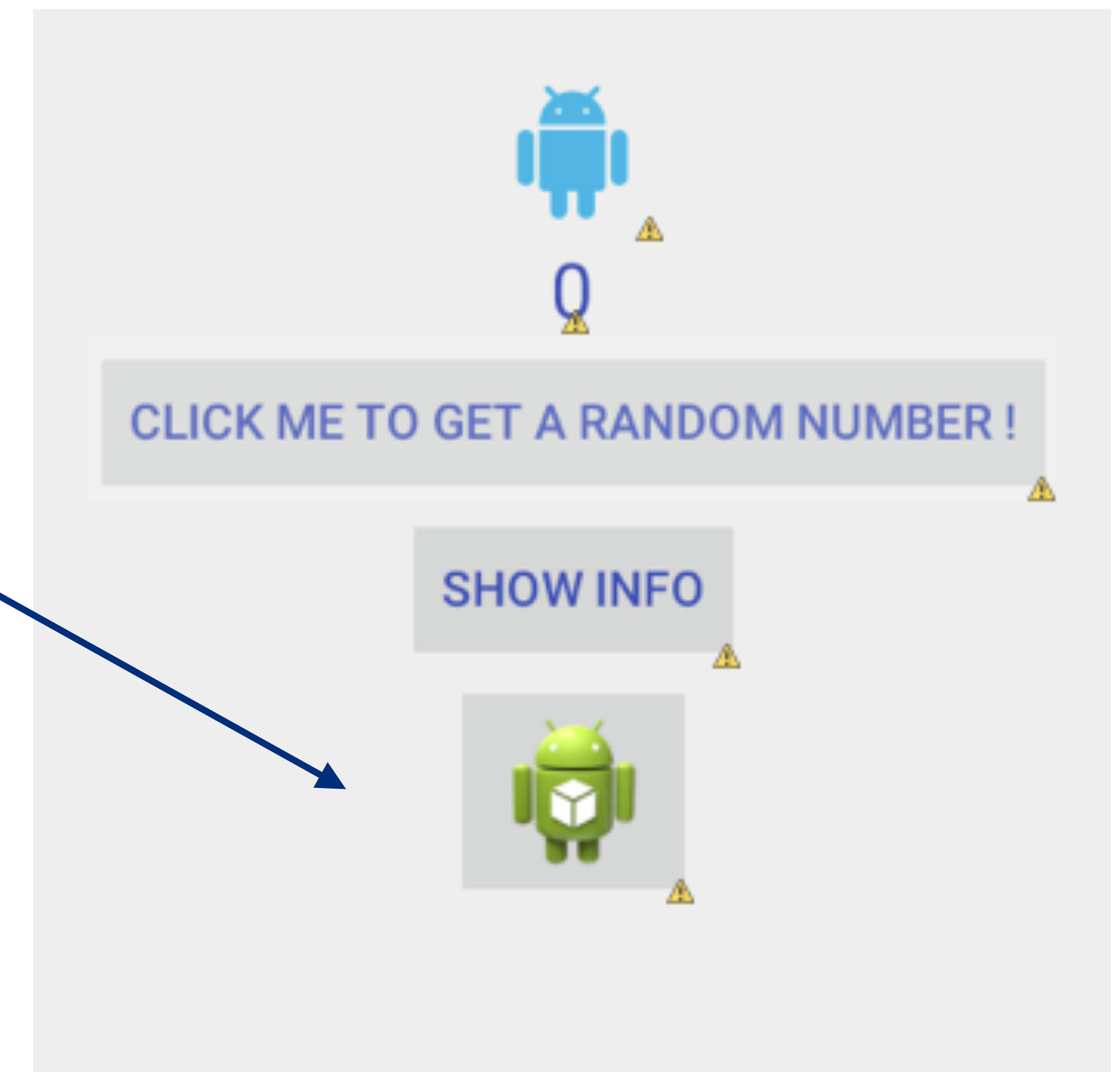
# Image Button

```
<ImageButton  
    android:id="@+id/imageButton"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_below="@+id/infoButton"  
    android:layout_centerHorizontal="true"  
    android:src="@drawable/ic_launcher" />
```

```
ImageButton imageButton = (ImageButton)findViewById(R.id.imageButton);  
imageButton.setImageDrawable(androidDrawable);  
imageButton.setImageBitmap(bm);  
imageButton.setImageResource(R.drawable.ic_launcher);
```

```
imageButton.setOnClickListener(new OnClickListener() {
```

```
    @Override  
    public void onClick(View v) {  
        ...  
    }  
});
```



# Launch / Finish Activity

- The **startActivity(Intent)** method is used to start a new activity, which will be placed at the top of the activity stack. It takes a single argument, an Intent, which describes the activity to be executed

```
startActivity(new Intent(getApplicationContext(), MyActivity.class));
```

- Sometimes you want to get a result back from an activity when it ends. For example, you may start an activity that lets the user pick a person in a list of contacts; when it ends, it returns the person that was selected. To do this, you call the **startActivityForResult(Intent, int)** version with a second integer parameter identifying the call. The result will come back through your **onActivityResult(int, int, Intent)** method.
- When an activity exits, it can call **setResult(int)** to return data back to its parent. It must always supply a result code, which can be the standard results **RESULT\_CANCELED**, **RESULT\_OK**, or any custom values starting at **RESULT\_FIRST\_USER**. In addition, it can optionally return back an Intent containing any additional data it wants. All of this information appears back on the parent's **Activity.onActivityResult()**, along with the integer identifier it originally supplied.
- When the activity is done the developer can call the method **finish()** to close it. The **ActivityResult** is propagated back to whoever launched you via **onActivityResult()**.

# Start Activity with Parameters

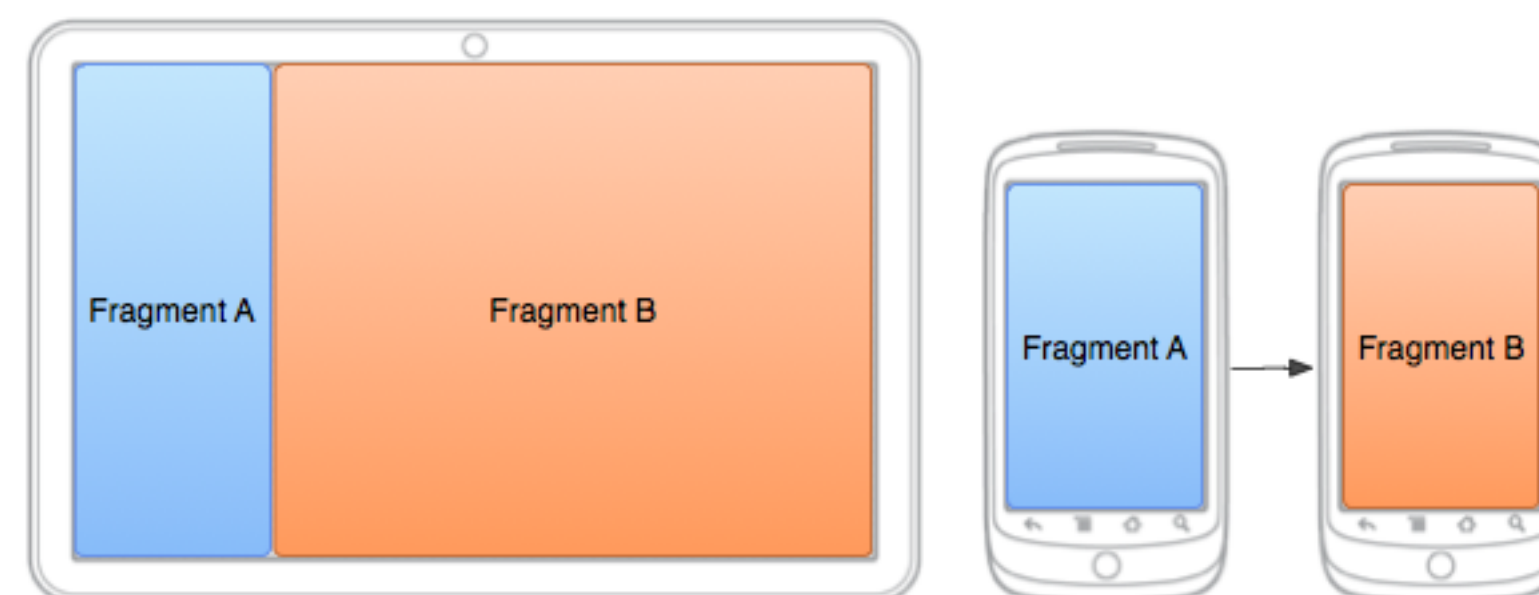
```
Bundle bundle = new Bundle();  
bundle.putInt("bookmarkPosition", position);
```

```
Intent newIntent = new Intent(getApplicationContext(), EditBookmarkActivity.class);  
newIntent.putExtras(bundle);  
startActivityForResult(newIntent);
```

- Using the method `startActivity(Intent)` or `startActivityForResult(Intent, result)` you can start a new activity, which will be placed at the top of the activity stack.
- With the second version you can specify an additional integer parameter identifying the call. The result will come back through your `onActivityResult(int, int, Intent)` method.
- The Bundle class is a map **Key<->Value** that allows to specify parameters associated to an Intent (through the method `putExtras(bundle)`) that you want to send to the new Activity.
- In the target Activity you can retrieve the incoming Bundle and extract sent values using the right Key.

```
Bundle bundle = this getIntent().getExtras();  
bookmarkPosition = bundle.getInt("bookmarkPosition");
```

# Fragments



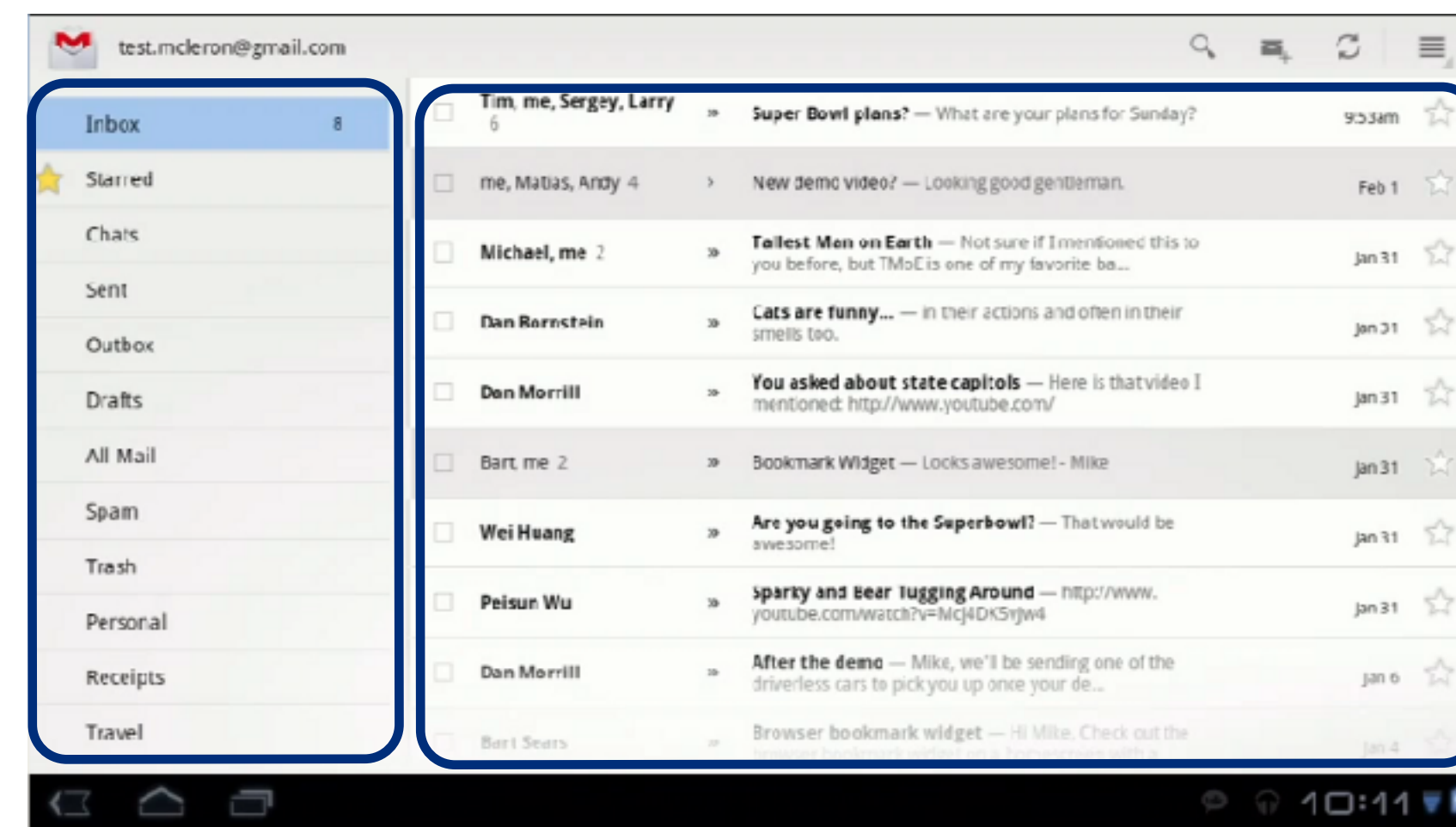
- A Fragment is a portion of the user interface in an Activity with a specific and potentially different behavior.
- It is possible to combine multiple fragments in a single activity in order to create a UI with multiple panels and reuse a fragment in multiple activities.
- Essentially a Fragment can be viewed as a modular section of an activity that
  - has its own lifecycle
  - receives its own input events
  - which can be added or removed while the activity is running

<http://developer.android.com/guide/components/fragments.html>

# Fragments Philosophy

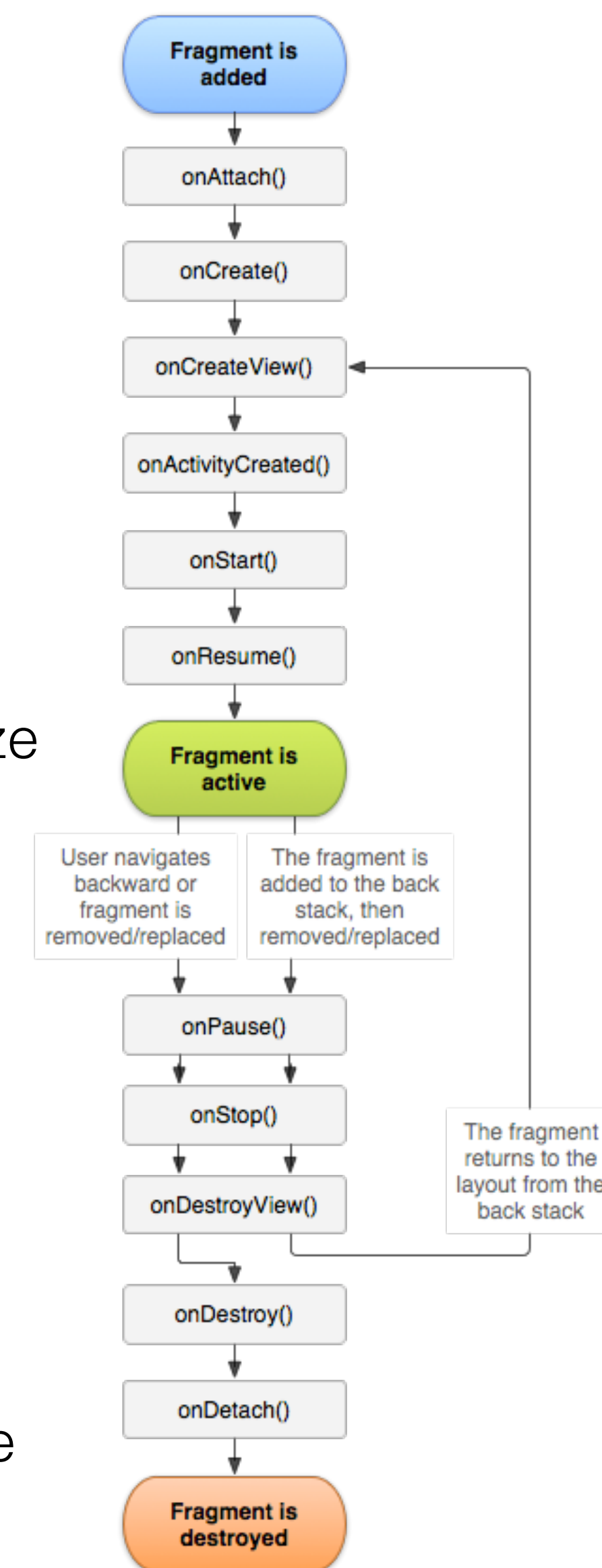
## – E.g:

- an email application can use one fragment to show the list of received e-mails on the left and another fragment to display the selected e-mail on the right
- both fragments appear in one activity, side by side
- each fragment has its own set of lifecycle callback methods and handle their own user input events.
- Using the fragments the user can in the same activity refresh the list of e-mails while he/she is reading an already selected message without changing the UI.



# Fragment LifeCycle

- To create a fragment, you must create a subclass of Fragment (or an existing subclass of it).
- The Fragment class looks appears like an Activity. It contains callback methods similar to an activity, such as onCreate(), onStart(), onPause(), and onStop().
- You can implement at least the following lifecycle methods to define your behavior:
  - onCreate(): The system calls this when creating the fragment. You should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.
  - onCreateView(): Called by the System when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a View from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.
  - onPause(): Method called by the System as a first feedback that the user is leaving the fragment (though it does not always mean the fragment is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session.



# Fragment Code

To define a layout for a fragment, you must implement the `onCreateView()` callback method called by the system when it's time for the fragment to draw its layout.

In order to return a Layout/View from the method `onCreateView()`, you can inflate it from a layout resource defined in XML. To help you the method provides a `LayoutInflater` object already created.

```
public static class ExampleFragment extends Fragment {  
    @Override  
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)  
    {  
        // Inflate the layout for this fragment  
        View view = inflater.inflate(R.layout.example_fragment, container, false);  
        return view;  
    }  
}
```

The container parameter represents the parent ViewGroup (from the activity's layout) in which the fragment layout will be inserted.

In order to return a Layout/View from the method `onCreateView()`, you can inflate it from a layout resource defined in XML. To help you the method provides a `LayoutInflater` object already created.

# Access UI Components (Activity vs Fragment)

```
<Button
    android:id="@+id/randomButton"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
/>
```

Activity



```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.single_fragment_activity);
    //Retrieve a Button UI Object
    Button randomButton = (Button)findViewById(R.id.randomButton);
}
```

Fragment



```
@Override
public View onCreateView(LayoutInflater inflater, ViewGroup container,
    Bundle savedInstanceState) {

    View view = inflater.inflate(R.layout.random_number_fragment, container, false);
    Button randomButton = ((Button)view.findViewById(R.id.randomButton));
    return view;
}
```

↑  
We directly access the View object to retrieve UI Elements

# Adding a fragment to an Activity

```
<?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">
    <fragment android:name="com.example.news.ArticleListFragment"
        android:id="@+id/list"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
    <fragment android:name="com.example.news.ArticleReaderFragment"
        android:id="@+id/viewer"
        android:layout_weight="2"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
</LinearLayout>
```

# Adding a fragment to an Activity

While your activity is running, you can add one or more fragments to your existing activity layout. You simply need to specify a ViewGroup in which to place the fragment. The management and the interaction between an Activity and a Fragment is defined and controlled by the class `FragmentManager`.

To make fragment transactions in your activity (add, remove, or replace a fragment), you must use APIs from `FragmentTransaction`.

```
FragmentManager fragmentManager = getFragmentManager()  
FragmentTransaction fragmentTransaction = fragmentManager.beginTransaction();  
  
ExampleFragment fragment = new ExampleFragment();  
fragmentTransaction.add(R.id.fragment_container, fragment);  
fragmentTransaction.commit();
```

You can then add a fragment using the `add()` method, specifying the fragment to add and the view in which to insert it. The first argument passed to `add()` is the ViewGroup in which the fragment should be placed, specified by resource ID, and the second parameter is the fragment to add. Once you've made your changes with `FragmentTransaction`, you must call `commit()` for the changes to take effect.

# Application communication with Fragments

- Although a Fragment is implemented as an object that is independent from an Activity and can be used inside multiple activities, a given instance of a fragment is directly tied to the activity that contains it.
- Specifically, the fragment can access the Activity instance with `getActivity()` and easily perform tasks such as find a view in the activity layout.

```
View listView = getActivity().findViewById(R.id.list);
```

- At the same an activity can call methods in the fragment by acquiring a reference to the Fragment from `FragmentManager`, using `findFragmentById()` or `findFragmentByTag()`.

```
ExampleFragment fragment = (ExampleFragment)  
getFragmentManager().findFragmentById(R.id.example_fragment);
```

# Fragments & Android Compatibility

- Android 3.0, or Honeycomb, came with some fundamental user interface changes, most notably in the form of the Fragment API.
- In order to support previous Android versions Google has released a library called the Android Compatibility package. (You should download and add the library to your project)
- This package provides support for the Fragment API as well as other key new features to devices as far back as Android 1.6.
- To support Fragment in previous version you should
  - Change the **Activity** to extend the **FragmentActivity** class when you want to use them
  - Change the call to the **getFragmentManager()** method to the **getSupportFragmentManager()** method
  - Organize or update the imports and the code (`import android.support.v4.app.Fragment;`  
(`android.support.v4.app.FragmentTabHost`)

# Access Context Reference

- You can access to your Application's Bundle Resources by using the Context object
- Activity, Service, BroadcastReceiver and ContentProvider extend the Context class.
- The Fragment class does not extend Context.

Inside an Activity you can do something like that:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.single_fragment_activity);
    Context context = this;
}
```

In a Fragment you should refer to the parent Activity in order to get the Context reference

```
//Context context = this.getActivity();
Context context = getActivity();
```

# Access Resources

- When you have a reference to the Context of your application you can retrieve available and indexed resources such as Strings, Drawable, Colors, Assets, Configuration, Layouts etc ...

## Drawable (e.g. Image)

```
Context context = getActivity(); //If for example we are in a Fragment
Drawable androidDrawable = context.getResources().getDrawable(R.drawable.ic_action_android);
imageView.setImageDrawable(androidDrawable);
```

## String (e.g. Text)

```
Context context = getActivity(); //If for example we are in a Fragment
String helloWorldString = context.getResources().getString(R.string.hello_world);
textView.setText(helloWorldString);
```

# Android Development

## Lecture 2

### Android Graphical User Interface 1



This work is licensed under a [Creative Commons Attribution-NonCommercial-ShareAlike 4.0 International License](https://creativecommons.org/licenses/by-nc-sa/4.0/).