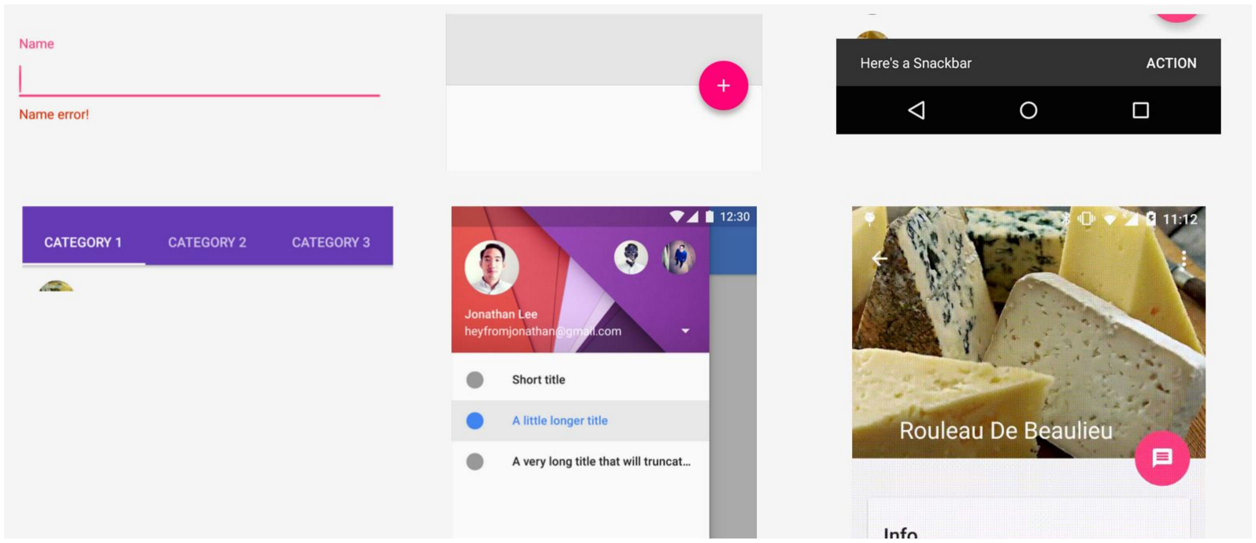
Material Design Codelab

Created by: onome@
Last update: onome@, 8/22/2015

Summary	In this codelab, you'll learn how to build an Android app using Material Design principles.
Material Design Spec	https://www.google.com/design/spec/material-design/introduction.html
Category	Android
Status	Final Draft
Feedback Link	https://docs.google.com/forms/d/16K77iRQIB3r0jJPzcEJqzYg9_JEDqRKpbjrsHctYwIY

Overview

Duration: 0:15



Material Design is a visual language that synthesizes the classic principles of good design with the innovation and possibility of technology and science. In this codelab, you'll learn the principles of this design language by building a sample Android app.

What you'll learn

- ✓ Material Design principles:
  - Pseudo-physical and tangible surfaces
  - Bold, graphic and intentional elements
  - Appropriate, authentic and meaningful motion
- ✓ Creating an Android app using the principles outlined above.

What you'll need

- Experience developing Android Apps
- A development machine with Android Studio version 1.0+ and JDK 7+
- A test device with Android 5.0 (Lollipop, API Level 21)*, or an Emulator with Lollipop

* Android devices running Android 2.3.3 (Gingerbread, API Level 10) or higher may be used, but, some Material Design effects will not be visible.

Setup

Duration: 0:45
Environment: Web
https://docs.google.com/document/d/17EBeztg5iHHXtE9cX4Bp_dyB0YFJ_IovG-1Af0bbogA/edit

Lessons

Duration: 1:45
Environment: Android Studio

Lesson 1 - Color and Typography

Duration: 0:15

Material Design Principles covered:

- Print-like Design

Let's jump right into one of the key features of Material Design: Themes and Color!

1. **Customize the Theme and Color Palette** -
   Themes let you apply a consistent tone to an app, and developers can choose between dark or light theme (see Figure 1 and Figure 2).

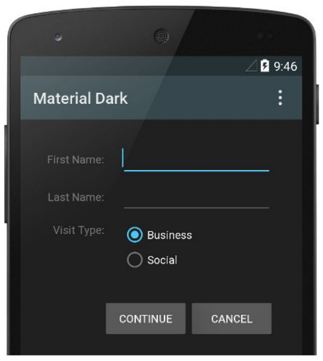


Figure 1. Dark material theme

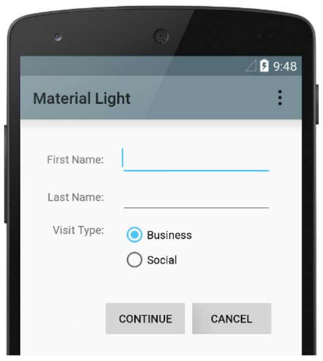


Figure 2. Light material theme

Custom colors can also be defined using theme attributes which are then automatically used by the app for different components e.g colorPrimaryDark for the Status Bar and colorPrimary for the App Bar (see Figure 3).

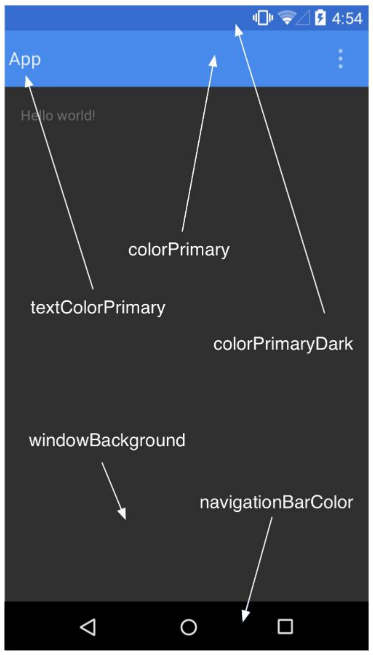


Figure 3. Customizing the material theme.

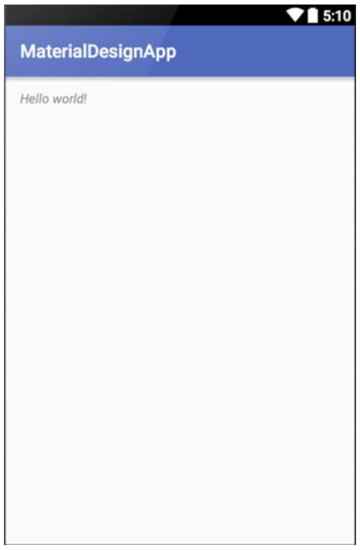
a. Add the Light theme to our app and customize some of the colors in res/values/styles.xml(v21)
b. Set typeface to Roboto and change the density and orientation of our app text in activity_main.xml

a. *res/values/styles.xml (v21)*

```
<resources>
   <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
       <!-- Customize your theme here. -->
       <item name="colorPrimary">#3F51B5</item>  <!-- Light Indigo -->
       <item name="colorPrimaryDark">#303F9F</item>  <!-- Dark Indigo -->
       <item name="colorAccent">#E91E63</item>  <!-- Magenta -->
   </style>
 </resources>
```

b. *res/layout/activity_main.xml*

```
<TextView
   android:text="@string/hello_world"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:textStyle="bold|italic"
   android:fontFamily="sans-serif-thin"/>  <!-- Roboto Typeface -->
```

App should now look like this: 

Extra Credit : If you have some time left at the end, play around with the typeface and color as you add more content to your app.

Lesson 2 - Layout and Animation

Duration: 0:45

Material Design Principles covered:

- Tangible Surfaces
- Bold Elements
- Meaningful Motion

1. Add a Top and Bottom ToolBar

a. Create a file called menu_bottom.xml containing 3 standard menu items (copy, cut and microphone), and place it under res/menu/ folder
b. In activity_main.xml, add a top and bottom ToolBar around the TextView
c. In MainActivity.java, inflate the menu items from (a) into the bottom ToolBar

b. res/layout/activity_main.xml

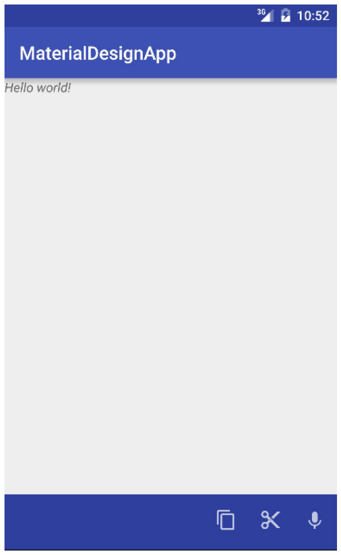
```
<RelativeLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto">

   <android.support.v7.widget.Toolbar
       android:id="@+id/toolbar"
       android:layout_width="match_parent"
```

```
        android:layout_height="?attr/actionBarSize"
        app:title="@string/app_name"
        android:background="?attr/colorPrimaryDark"
        android:theme="@style/ThemeOverlay.AppCompat.Dark"/>
    <!-- TextView -->
    <android.support.v7.widget.Toolbar
        android:id="@+id/bottom_toolbar"
        android:layout_height="?attr/actionBarSize"
        android:layout_width="match_parent"
        android:theme="@style/Base.ThemeOverlay.AppCompat.Dark"
        android:background="?attr/colorPrimaryDark"
        android:layout_alignParentBottom="true"/>
</RelativeLayout>
```

c. java/.../MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        /* Top toolbar */
        Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);

        /* Bottom toolbar. */
        Toolbar bottomToolbar = (Toolbar) findViewById(R.id.bottom_toolbar);
        bottomToolbar.inflateMenu(R.menu.menu_bottom);
    }
...
}
```

App should now look like this:



Extra credit:

2. Add a RecyclerView and Cards with Ripples

RecyclerView is a container for displaying large data sets that can be scrolled very efficiently by maintaining a limited number of views.
Our data set for this code lab are empty cards which are pieces of paper that serve as an

entry point to more information.

Let's add a RecyclerView and some cards to our app:

a. Create a file defining a card called cardview.xml under res/layout/. Take a look at the individual attributes to see what customizations are available
b. In activity_main.xml, replace the Text View with a CoordinatorLayout (enables components to trigger actions based on interactions) containing a RecyclerView
c. In MainActivity.java, create and customize the RecyclerView by adding the cards using the view defined in (a) via a RecyclerView.Adapter.
    i. Add a scroll animation to the cards with the RecyclerView
    ii. Add a ripple (press and release) animation to the cards

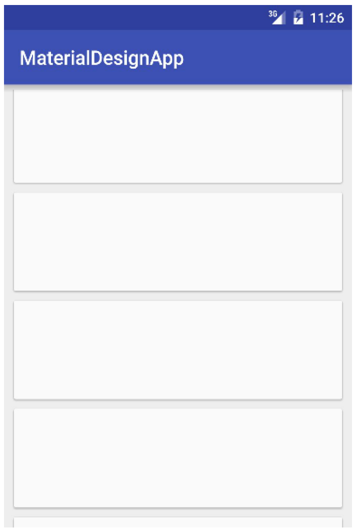
b. res/layout/activity_main.xml

```xml
<RelativeLayout
   <!-- Top Toolbar -->

   <android.support.design.widget.CoordinatorLayout
       android:id="@+id/main_content"
       android:layout_width="match_parent"
       android:layout_height="match_parent"
       android:layout_marginTop="?attr/actionBarSize"
       android:layout_marginBottom="?attr/actionBarSize"
       android:layout_alignParentTop="true">
       <android.support.v7.widget.RecyclerView
           android:id="@+id/my_recycler_view"
           android:scrollbars="vertical"
           android:layout_width="match_parent"
           android:layout_height="match_parent"
           android:paddingTop="8dp"
           android:paddingBottom="8dp"
           android:clipToPadding="false"
           app:layout_behavior="@string/appbar_scrolling_view_behavior"/>
   </android.support.design.widget.CoordinatorLayout>
   <!-- Bottom Toolbar -->
</RelativeLayout>
```

c. java/.../MainActivity.java

```
public class MainActivity extends AppCompatActivity {
   @Override
   protected void onCreate(Bundle savedInstanceState) {
       /* Top toolbar. */
       /* Bottom toolbar. */
       /* Create and customize RecyclerView. */
       RecyclerView recyclerView = (RecyclerView) findViewById(R.id.my_recycler_view);
       recyclerView.setHasFixedSize(true);
       recyclerView.setLayoutManager(new LinearLayoutManager(this));
       // Add 8 cards
       MyAdapter adapter = new MyAdapter(new String[8]);
       recyclerView.setAdapter(adapter);
   }

   /* Create RecylcerView Adapter. */
   public static class MyAdapter extends RecyclerView.Adapter<MyAdapter.ViewHolder> {
       private String[] mDataset;

       public static class ViewHolder extends RecyclerView.ViewHolder {
           public View view;
           public TextView title;
           public ViewHolder(View v) {
               super(v);
               view = v;
               title = (TextView) v.findViewById(R.id.card_title);
           }
       }

       public MyAdapter(String[] myDataset) {
           mDataset = myDataset;
       }

       @Override
       public MyAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {
           View cardview = LayoutInflater.from(parent.getContext())
                   .inflate(R.layout.cardview, parent, false);
           return new ViewHolder(cardview);
       }

       @Override
       public void onBindViewHolder(ViewHolder holder, int position) {
            holder.title.setText("Card " + (position + 1));
       }

       @Override
       public int getItemCount() {
           return mDataset.length;
```

```
        }
    }
}
```

App should now look like this: 

Extra Credit : Add Lift on Touch animation.

3. Add an Image to a CollapsingToolbar

A Collapsing Toolbar provides visual transitions by collapsing a toolbar as the user scrolls down the app.

Let's go ahead and add a Collapsing Toolbar:

a. Download this image or any image of your choice and add it under res/drawable folder as image.png
b. In activity_main.xml, add an AppBarLayout (enables components react to scrolling) containing a CollapsingToolbar which consists of:
    i. An ImageView of the image from (a)
    ii. A ToolBar that the image collapses into

   These should be added within the CoordinatorLayout created in (2) above
c. In MainActivity.java, create a Palette instance from the bitmap of the image referred to by the ImageView
    i. Using the Palette created, asynchronously set the collapsed toolbar color

b. res/layout/activity_main.xml

```
<RelativeLayout
 <!-- Top Toolbar -->
 <android.support.design.widget.CoordinatorLayout
     android:id="@+id/main_content"
```

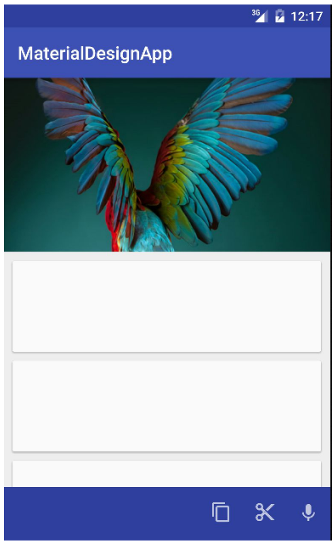
```
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginBottom="?attr/actionBarSize"
        android:layout_alignParentTop="true">
        <android.support.design.widget.AppBarLayout
            android:id="@+id/appbar"
            android:layout_width="match_parent"
            android:layout_height="192dp">
            <android.support.design.widget.CollapsingToolbarLayout
                android:id="@+id/collapsing_toolbar"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:theme="@style/ThemeOverlay.AppCompat.Dark"
                app:layout_scrollFlags="scroll|exitUntilCollapsed">
                <ImageView
                    android:id="@+id/image"
                    android:layout_width="match_parent"
                    android:layout_height="match_parent"
                    android:background="@drawable/image"
                    android:fitsSystemWindows="true"
                    android:scaleType="centerCrop"
                    app:layout_collapseMode="parallax" />
                <android.support.v7.widget.Toolbar
                    android:id="@+id/toolbar"
                    android:layout_width="match_parent"
                    android:layout_height="?attr/actionBarSize"
                    app:popupTheme="@style/ThemeOverlay.AppCompat.Light"
                    app:layout_collapseMode="pin" />
            </android.support.design.widget.CollapsingToolbarLayout>
        </android.support.design.widget.AppBarLayout>
        <!-- RecyclerView -->
    </android.support.design.widget.CoordinatorLayout>
    <!-- Bottom Toolbar -->
</RelativeLayout>
```

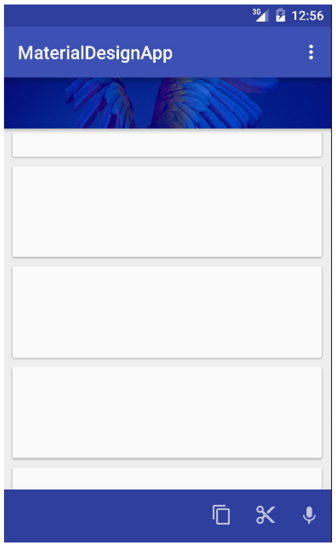
c. java/.../MainActivity.java

```
public class MainActivity extends AppCompatActivity {
   private CollapsingToolbarLayout ctb;
   private int mutedColor;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
        /* Top toolbar */
        /* Bottom toolbar */
        /* RecyclerView */
        /* Cards */
        /* Collapsing toolbar */
        ctb = (CollapsingToolbarLayout) findViewById(R.id.collapsing_toolbar);
        /* Define the image */
        ImageView image = (ImageView) findViewById(R.id.image);
```

```
            /* Decode bitmap from the image */
            Bitmap bitmap = BitmapFactory.decodeResource(getResources(), R.drawable.image);
            /* Generate palette from the image bitmap */
            Palette.from(bitmap).generate(new Palette.PaletteAsyncListener() {
                    @Override
                    public void onGenerated(Palette palette) {
                            mutedColor = palette.getMutedColor(R.attr.colorPrimary);
                            /* Set toolbar color from the palette */
                            ctb.setContentScrimColor(mutedColor);
                    }
            });
        }
    ...
    }
```

App should now look like this:

Before collapse: 

After collapse: 

Lesson 3 - Page Elements

Duration: 0:45

Material Design Principles covered:

- Tangible Surfaces
- Bold Elements

1. Add a NavigationDrawer

The navigation drawer slides in from the left. It is a common pattern found in Google apps and follows the keylines and metrics for lists.

Let's add a NavigationDrawer and set it to open when selected:

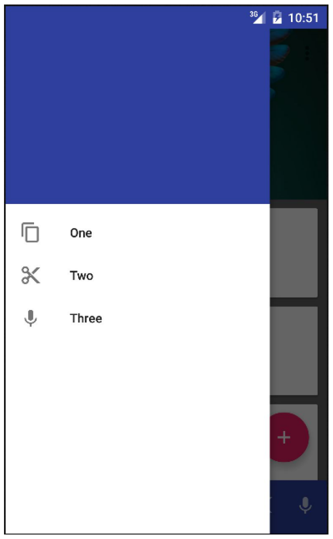
a. Create a file menu_navigation.xml defining the navigation items under res/menu folder
b. Create a file navheader.xml defining a Navigation Drawer material under res/layout/ folder
c. Create a file ic_menu_24dp.xml defining the navigation menu under res/drawable folder (to support versions of Android before Lollipop, download the icon from here)
d. In activity_main.xml:
    i. Encapsulate all the components within a DrawerLayout which enables interactive drawer views to be pulled out from the edge of the window.
    ii. Add a NavigationView outside the RelativeLayout
e. In MainActivity.java:
    i. Add navigation menu to the ActionBar
    ii. Enable open and close of NavigationView

c. res/layout/activity_main.xml

```
 <android.support.v4.widget.DrawerLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:id="@+id/drawer"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:fitsSystemWindows="true">
   <RelativeLayout
       android:layout_width="match_parent"
       android:layout_height="match_parent">
       .
       .
       .
   </RelativeLayout>
   <android.support.design.widget.NavigationView
       android:id="@+id/nav_view"
       android:layout_width="wrap_content"
       android:layout_height="match_parent"
       android:layout_gravity="start"
       android:background="#ffffff"
       android:clickable="true"
       app:headerLayout="@layout/navheader"
       app:menu="@menu/menu_navigation"
       app:itemBackground="?attr/selectableItemBackground"/>
</android.support.v4.widget.DrawerLayout>
```

c. java/.../MainActivity.java

```
public class MainActivity extends AppCompatActivity {
    private CollapsingToolbarLayout ctb;
    private int mutedColor;
    private DrawerLayout drawerLayout;
    private NavigationView navigationView;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        /* Top toolbar */
        /* Bottom toolbar */
        /* RecyclerView */
        /* Cards */
        /* Collapsing toolbar */
        /* NavigationView */
        navigationView = (NavigationView) findViewById(R.id.nav_view);
        drawerLayout = (DrawerLayout) findViewById(R.id.drawer);

         // On click of menu icon on toolbar
         toolbar.setNavigationIcon(R.xml.ic_menu_24dp);
         toolbar.setNavigationOnClickListener(new View.OnClickListener() {
             @Override
             public void onClick(View v) {
                 drawerLayout.openDrawer(GravityCompat.START);
             }
         });

        // On click of the navigation menu
        navigationView.setNavigationItemSelectedListener(new
        NavigationView.OnNavigationItemSelectedListener() {
                // This method will trigger on item Click of navigation menu
                @Override
                public boolean onNavigationItemSelected(MenuItem menuItem) {
                        // Set item in checked state
                        menuItem.setChecked(true);

                        //TODO: handle navigation

                        //Closing drawer on item click
                        drawerLayout.closeDrawers();
                        return true;
                }
        });
    }
}
```

App should now look like this when menu is selected: 

2. Add a Floating Action Button (FAB) and trigger a SnackBar

Floating action buttons are used for a promoted action and are distinguished by a circled icon floating above the UI.

Let's create a FAB that triggers a SnackBar which provides lightweight feedback feedback about an operation by showing a brief message:

a. Download this icon and add it under res/drawable/ as ic_add.png
b. In activity_main.xml:
    i. Add a FloatingActionButton to the end of the CoordinatorLayout, after the RecyclerView.
    ii. Set the source(src attribute) of the FAB to the icon from (a)
c. In MainActivity.java, add a Listener to the FAB that creates a SnackBar onclick

c. res/layout/activity_main.xml

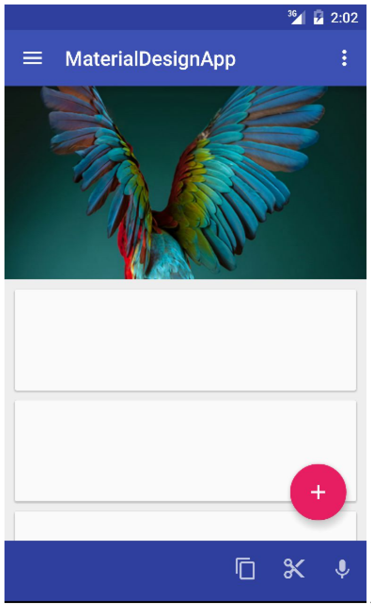
```
<android.support.v4.widget.DrawerLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:id="@+id/drawer"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:fitsSystemWindows="true">
   <RelativeLayout
       <android.support.design.widget.CoordinatorLayout
       .
       .
          <android.support.design.widget.FloatingActionButton
```

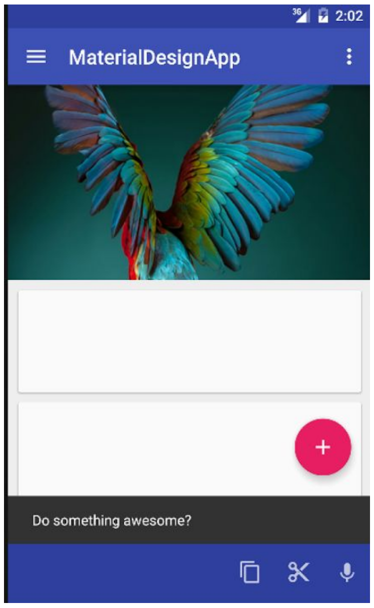
```
                android:id="@+id/fab"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginBottom="16dp"
                android:layout_marginRight="16dp"
                android:layout_above="@+id/bottom_toolbar"
                android:layout_gravity="right|bottom"
                android:tint="#ffffff"
                android:src="@drawable/ic_add"/>
        </android.support.design.widget.CoordinatorLayout>
        .
        .
   </RelativeLayout>
</android.support.v4.widget.DrawerLayout>
```

MainActivity.java

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    /* Top toolbar */
    /* Bottom toolbar */
    /* RecyclerView */
    /* Cards */
    /* Collapsing toolbar */
    /* NavigationView */

    /* Floating Action Button. */
    FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
    fab.setOnClickListener(new View.OnClickListener() {
       @Override
       public void onClick(View v) {
          Snackbar.make(v, "Do something awesome?",
               Snackbar.LENGTH_LONG).show();
       }
    });
 }
```

App should now look like this:

Before clicking FAB: 

After clicking FAB: 

Do something awesome?

Congratulations! You have finished the code lab!

Optional Lessons

Duration: 0:15 - ~

If you finish the codelab with time to spare, feel free to:

1. Trigger actions based on card selection in the RecyclerView
2. Trigger actions based on item selection in the NavigationView
3. Add more content to your app
4. Play around with the colors, themes and typography