

Android Development

Lecture 3

Android Graphical User Interface 2



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

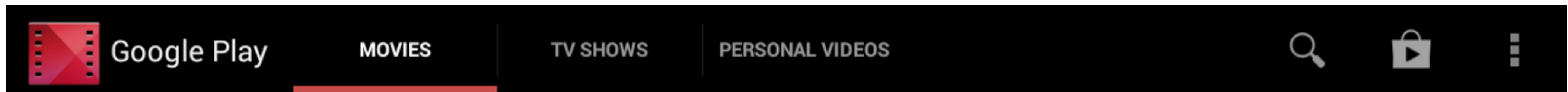
Lecture Summary

- The ActionBar
- Handling Actions with the ActionBar
- Navigation with the ActionBar
- ToolBar
- ActionBar -> ToolBar
- Lists & RecyclerView
- Handle Multiple View Types
- CardView



ActionBar

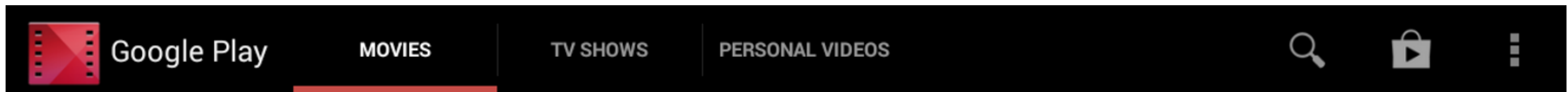
- The action bar is a window feature that allows to
 - identify the user location (current Activity/View)
 - provide user actions and navigation modes
- offer a familiar interface across applications that the system properly adapts for different screen configurations.
- support consistent navigation and view switching within apps (with tabs or drop-down lists).



UI Design Guide: <http://developer.android.com/design/patterns/actionbar.html>

ActionBar

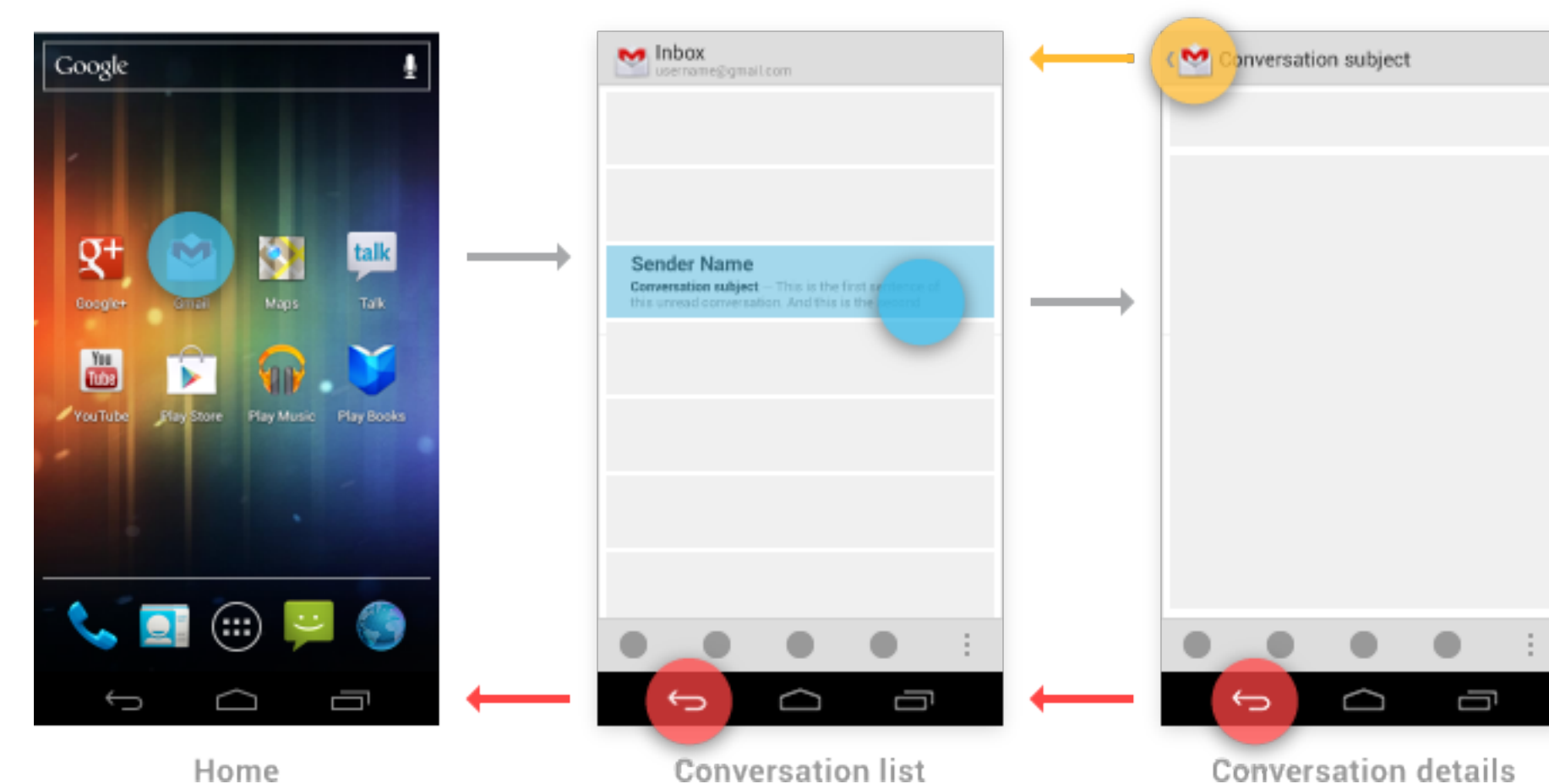
- The action bar is a window feature that allows to
 - identify the user location (current Activity/View)
 - provide user actions and navigation modes
 - offer a familiar interface across applications that the system properly adapts for different screen configurations.
 - support consistent navigation and view switching within apps (with tabs or drop-down lists).



UI Design Guide: <http://developer.android.com/design/patterns/actionbar.html>

ActionBar & Navigation

- “Consistent navigation is an essential component of the overall user experience.”
- With Android 2.3 and earlier OS versions the navigation was based on system Back button.
- With the introduction of action bars in Android 3.0, a second navigation mechanism appeared using the **Up button** (the app icon and a left-point caret).



ActionBar & Navigation Approach: <http://developer.android.com/design/patterns/navigation.html>

Adding the ActionBar

- In order to add and use the ActionBar to your project you must:
 - set up your project with the appcompat v7 support library.
 - Create your activity by extending ActionBarActivity.
 - Use (or extend) one of the Theme.AppCompat themes for your activity. For example:

```
<activity android:theme="@style/Theme.AppCompat.Light" ... >
```

- The right theme allow the OS to correctly load and create an ActionBar for your Activity. Some themes do not provides the support for the ActionBar and your will receive a null object in your Activity.

Action Bar Code

```
<application
    android:allowBackup="true"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/Theme.AppCompat">

    <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>
```

getSupportActionBar() if we have the support library v7 otherwise just getActionBar().

```
public class MainActivity extends ActionBarActivity{
```

```
...
```

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

```
// Set up the action bar.
```

```
ActionBar actionBar = getSupportActionBar();
```

```
// Specify that the Home/Up button should not be enabled,
//since there is no hierarchical parent.
```

```
actionBar.setHomeButtonEnabled(false);
```

```
//UnComment the following lines to hide the top part of the action bar
//actionBar.setDisplayShowTitleEnabled(false);
//actionBar.setDisplayHomeAsUpEnabled(false);
//actionBar.setDisplayShowHomeEnabled(false);
```

```
}
```

```
}
```

Methods to display or not some part of the ActionBar

ActionBar Actions

- The action bar provides users access to the most important action items relating to the app's current context.
- Those that appear directly in the action bar with an icon and/or text are known as **action buttons**. Actions that can't fit in the action bar or aren't important enough are **hidden in the action overflow**.
- The user can reveal a list of the other actions by pressing the overflow button on the right side (or the device Menu button, if available).
- When your activity starts, the system **populates the action items by calling your activity's `onCreateOptionsMenu()` method**. Use this method to inflate a menu resource that defines all the action items. For example, here's a menu resource defining a couple of menu items:

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
  <item android:id="@+id/action_search"
        android:icon="@drawable/ic_action_search"
        android:title="@string/action_search" />
  <item android:id="@+id/action_compose"
        android:icon="@drawable/ic_action_compose"
        android:title="@string/action_compose" />
</menu>
```

res/menu/main_activity_actions.xml

ActionBar Actions

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
  <item android:id="@+id/action_search"
        android:icon="@drawable/ic_action_search"
        android:title="@string/action_search" />
  <item android:id="@+id/action_compose"
        android:icon="@drawable/ic_action_compose"
        android:title="@string/action_compose" />
</menu>
```

res/menu/main_activity_actions.xml



XML inflated in the method
onCreateOptionsMenu(...)

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu items for use in the action bar
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.main_activity_actions, menu);
    return super.onCreateOptionsMenu(menu);
}
```

ActionBar Actions

- To request that an item appear directly in the action bar as an action button, include `showAsAction="ifRoom"` in the `<item>` tag.
- If there's not enough room for the item in the action bar, it will appear in the action overflow.

```
<menu xmlns:android="http://schemas.android.com/apk/res/android"
      xmlns:yourapp="http://schemas.android.com/apk/res-auto" >
  <item android:id="@+id/action_search"
        android:icon="@drawable/ic_action_search"
        android:title="@string/action_search"
        yourapp:showAsAction="ifRoom" />
  ...
</menu>
```

If your menu item use a title and an icon—with the title and icon attributes—then the action item shows only the icon by default. If you want to display the text title, add "withText" to the `showAsAction` attribute.

```
<item yourapp:showAsAction="ifRoom|withText" ... />
```

Handling Actions

- When the user presses an action, the system calls your activity's `onOptionsItemSelected()` method.
- Using the `MenuItem` passed to this method, you can identify the action by calling `getItemId()`.
- This returns the unique ID provided by the `<item>` tag's `id` attribute so you can perform the appropriate action.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle presses on the action bar items
    switch (item.getItemId()) {
        case R.id.action_search:
            openSearch();
            return true;
        case R.id.action_compose:
            composeMessage();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

ActionBar & Navigation

- Enabling the app icon as an Up button allows the user to navigate your app based on the hierarchical relationships between screens. For instance, if screen A displays a list of items, and selecting an item leads to screen B, then screen B should include the Up button, which returns to screen A.
- To enable the app icon as an Up button, call `setDisplayHomeAsUpEnabled()`.

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_details);

    ActionBar actionBar = getSupportActionBar();
    actionBar.setDisplayHomeAsUpEnabled(true);
    ...
}
```

Now the icon in the action bar appears with the *Up* caret (as shown in figure 4). However, it won't do anything by default. To specify the activity to open when the user presses *Up* button, you have two options:

ActionBar & Navigation - XML

- This is the best option when the parent activity is always the same. By declaring in the manifest which activity is the parent, the action bar automatically performs the correct action when the user presses the Upbutton.
- Beginning in Android 4.1 (API level 16), you can declare the parent with the `parentActivityName` attribute in the `<activity>` element.
- To support older devices with the support library, also include a `<meta-data>` element that specifies the parent activity as the value for `android.support.PARENT_ACTIVITY`.
- Once the parent activity is specified in the manifest like this and you enable the Up button with `setDisplayHomeAsUpEnabled()`, your work is done and the action bar properly navigates up.

```
<application ... >
...
<!-- The main/home activity (has no parent activity) -->
<activity
    android:name="com.example.myfirstapp.MainActivity" ...>
...
</activity>
<!-- A child of the main activity -->
<activity
    android:name="com.example.myfirstapp.DisplayMessageActivity"
    android:label="@string/title_activity_display_message"
    android:parentActivityName="com.example.myfirstapp.MainActivity" >
    <!-- Parent activity meta-data to support API level 7+ -->
    <meta-data
        android:name="android.support.PARENT_ACTIVITY"
        android:value="com.example.myfirstapp.MainActivity" />
    </activity>
</application>
```

ActionBar & Navigation - Code

- Define the navigation through code is appropriate when the parent activity may be different depending on how the user arrived at the current screen.
- The system calls `getSupportParentActivityIntent()` when the user presses the Up button while navigating your app.
- If the activity that should open upon up navigation differs depending on how the user arrived at the current location, then you should override this method to return the Intent that starts the appropriate parent activity.
- The system calls `onCreateSupportNavigateUpTaskStack()` for your activity when the user presses the Up button while your activity is running in a task that does not belong to your app. Thus, you must use the `TaskStackBuilder` passed to this method to construct the appropriate back stack that should be synthesized when the user navigates up.
- Even if you override `getSupportParentActivityIntent()` to specify up navigation as the user navigates your app, you can avoid the need to implement `onCreateSupportNavigateUpTaskStack()` by declaring "default" parent activities in the manifest file as shown above. Then the default implementation of `onCreateSupportNavigateUpTaskStack()` will synthesize a back stack based on the parent activities declared in the manifest.

Navigating Up with the App Icon

<http://developer.android.com/guide/topics/ui/actionbar.html>

ActionBar & Navigation - Code

@Override

```
public boolean onOptionsItemSelected(MenuItem item){  
  
    switch (item.getItemId()) {  
        // Respond to the action bar's Up/Home button  
        case android.R.id.home:  
            //NavUtils.navigateUpFromSameTask(this);  
            Intent intent = new Intent(this, MainActivity.class);  
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP | Intent.FLAG_ACTIVITY_NEW_TASK);  
            startActivityForResult(intent, 0);  
            return true;  
        }  
        return super.onOptionsItemSelected(item);  
    }  
}
```

ActionBar -> ToolBar

- With Android 5.0, Android introduces a new **Toolbar widget**.
- It is a generalization of the Action Bar pattern
- The ToolBar gives you much more control and flexibility. It is a view in your hierarchy just like any other, making it easier to interleave with the rest of your views, animate it, and react to scroll events.
- **You can also set it as your Activity's action bar, meaning that your standard options menu actions will be display within it.**
- Toolbar is fully supported in AppCompatActivity and has feature and API parity with the framework widget. In AppCompatActivity, Toolbar is implemented in the `android.support.v7.widget.Toolbar` class.
- The most important difference is that this **toolbar is now part of the layout** allowing us to work with it in any way we consider: animations, drawer overlay, etc.

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

ActionBar -> ToolBar

AppCompatActivity

- All of your Activities must extend from ~~ActionBarActivity~~, which extends from `FragmentActivity` from the v4 support library, so you can continue to use fragments.
- All of your themes (that want an Action Bar/Toolbar) must inherit from `Theme.AppCompat`.
- When inflating anything to be displayed on the action bar (such as a `SpinnerAdapter` for list navigation in the toolbar), make sure you use the action bar's themed context, retrieved via `getSupportActionBar().getThemedContext()`.
- You must use the static methods in `MenuItemCompat` for any action-related calls on a `MenuItem`.

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

ActionBar -> ToolBar (Theme)

```
<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:textColor">@color/primary</item>
  <!-- <item name="android:textColorSecondary">@color/primary</item> -->
  <!-- <item name="android:background">@color/primary</item> -->
  <item name="android:windowNoTitle">true</item>
  <item name="windowActionBar">>false</item>
</style>
```

```
<style name="AppTheme.NoActionBar">
  <item name="windowActionBar">>false</item>
  <item name="windowNoTitle">true</item>
</style>
```

Disable the traditional (and old) ActionBar in order to manually add and manage the new ToolBar.

Source: <http://antonioleiva.com/material-design-everywhere/>

ToolBar Layout XML

<LinearLayout

```
xmlns:android="http://schemas.android.com/apk/res/android"  
xmlns:tools="http://schemas.android.com/tools"  
android:layout_width="match_parent"  
android:layout_height="match_parent"  
android:orientation="vertical" >
```

<android.support.v7.widget.Toolbar

```
android:id="@+id/my_awesome_toolbar"  
android:layout_width="match_parent"  
android:layout_height="wrap_content"  
android:background="?attr/colorPrimary"  
android:minHeight="?attr/actionBarSize" />
```

<FrameLayout

```
android:id="@+id/container"  
android:layout_width="match_parent"  
android:layout_height="match_parent"  
tools:context="com.mobdev.multifragment.MainActivity"  
tools:ignore="MergeRootFrame" >
```

</FrameLayout>

</LinearLayout>

Source: <http://antonioleiva.com/material-design-everywhere/>

ToolBar Code

```
public class InfoActivity extends ActionBarActivity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.single_fragment_activity);  
  
        [...]  
  
        Toolbar toolbar = (Toolbar)findViewById(R.id.my_awesome_toolbar);  
        toolbar.setTitle("App Info");  
  
        setSupportActionBar(toolbar);  
  
        getSupportActionBar().setHomeButtonEnabled(true);  
        getSupportActionBar().setDisplayHomeAsUpEnabled(true);  
    }  
  
    [...]  
}
```

Retrieve and configure
the ToolBar from the
Layout

Set the ToolBar as
ActionBar

Use classic ActionBar
methods

Source: <http://antonioleiva.com/material-design-everywhere/>

ToolBar Code - Navigation

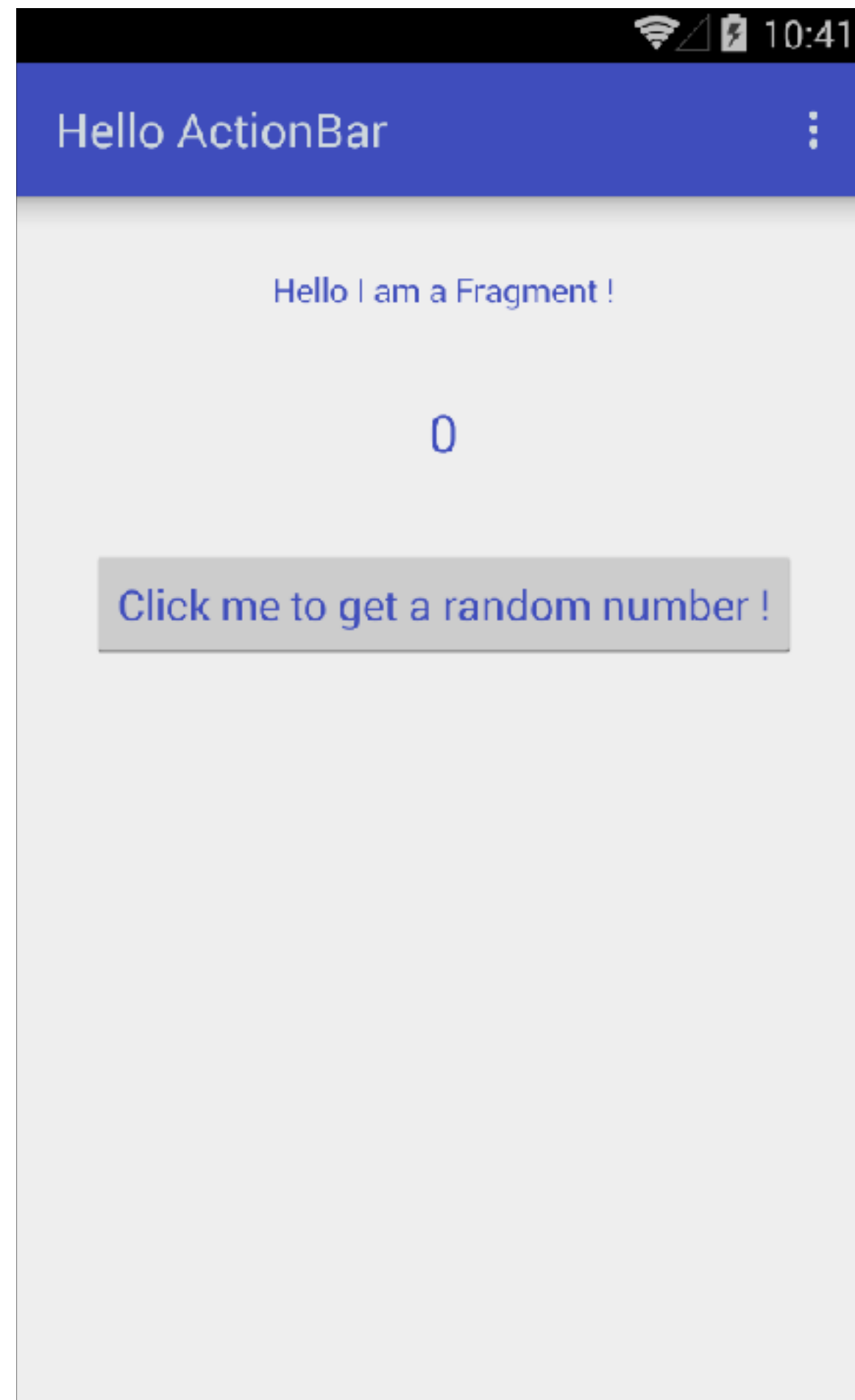
@Override

```
public boolean onOptionsItemSelected(MenuItem item){  
  
    switch (item.getItemId()) {  
        // Respond to the action bar's Up/Home button  
        case android.R.id.home:  
            //NavUtils.navigateUpFromSameTask(this);  
            Intent intent = new Intent(this, MainActivity.class);  
            intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP | Intent.FLAG_ACTIVITY_NEW_TASK);  
            startActivityForResult(intent, 0);  
            return true;  
        }  
        return super.onOptionsItemSelected(item);  
    }  
}
```

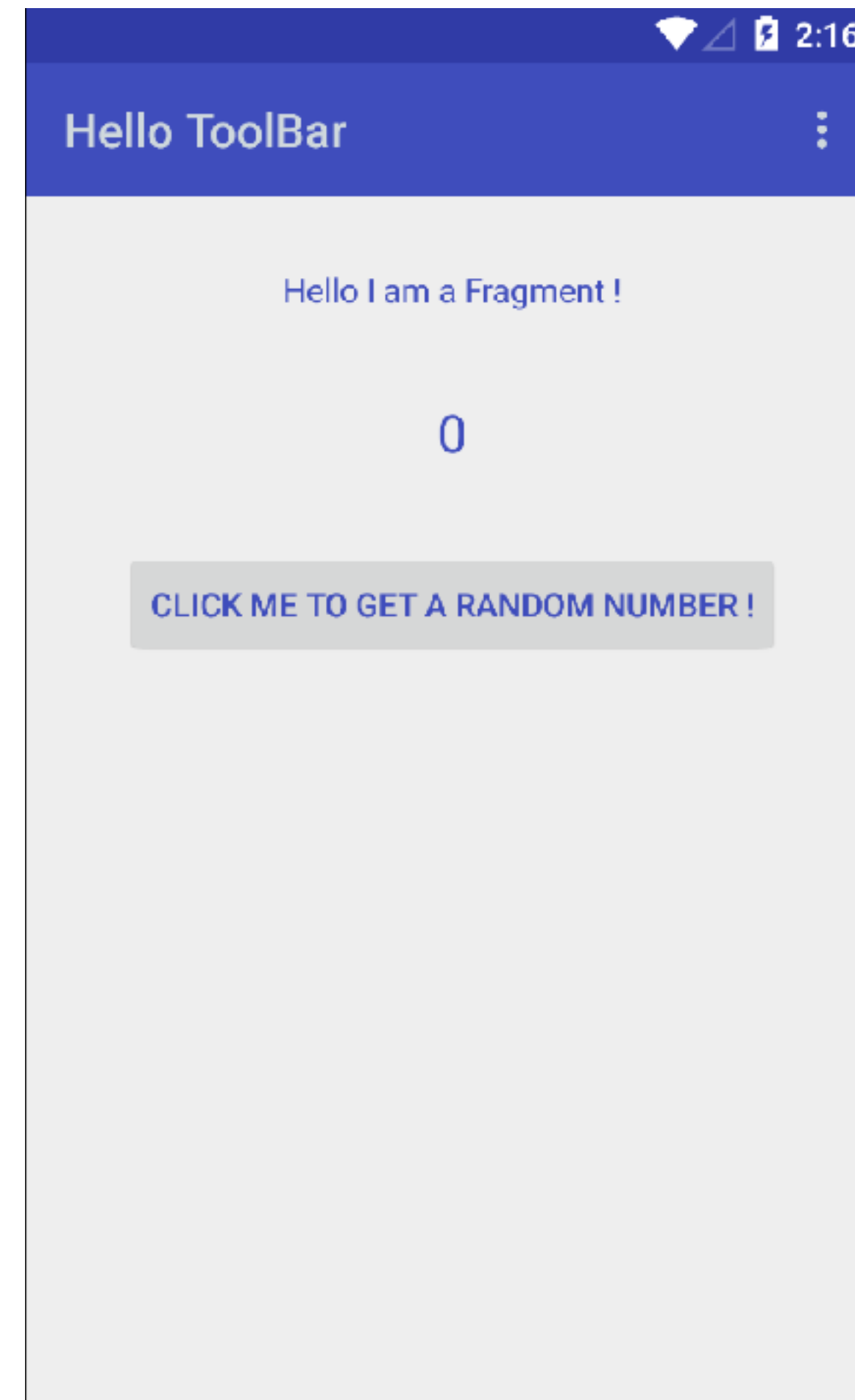
Same code and behaviour of the traditional ActionBar

ToolBar Result

Traditional
Action Bar on
4.4 Emulator



ToolBar on
5.0 Lollipop
Device



Android Application Icons

Icon Design

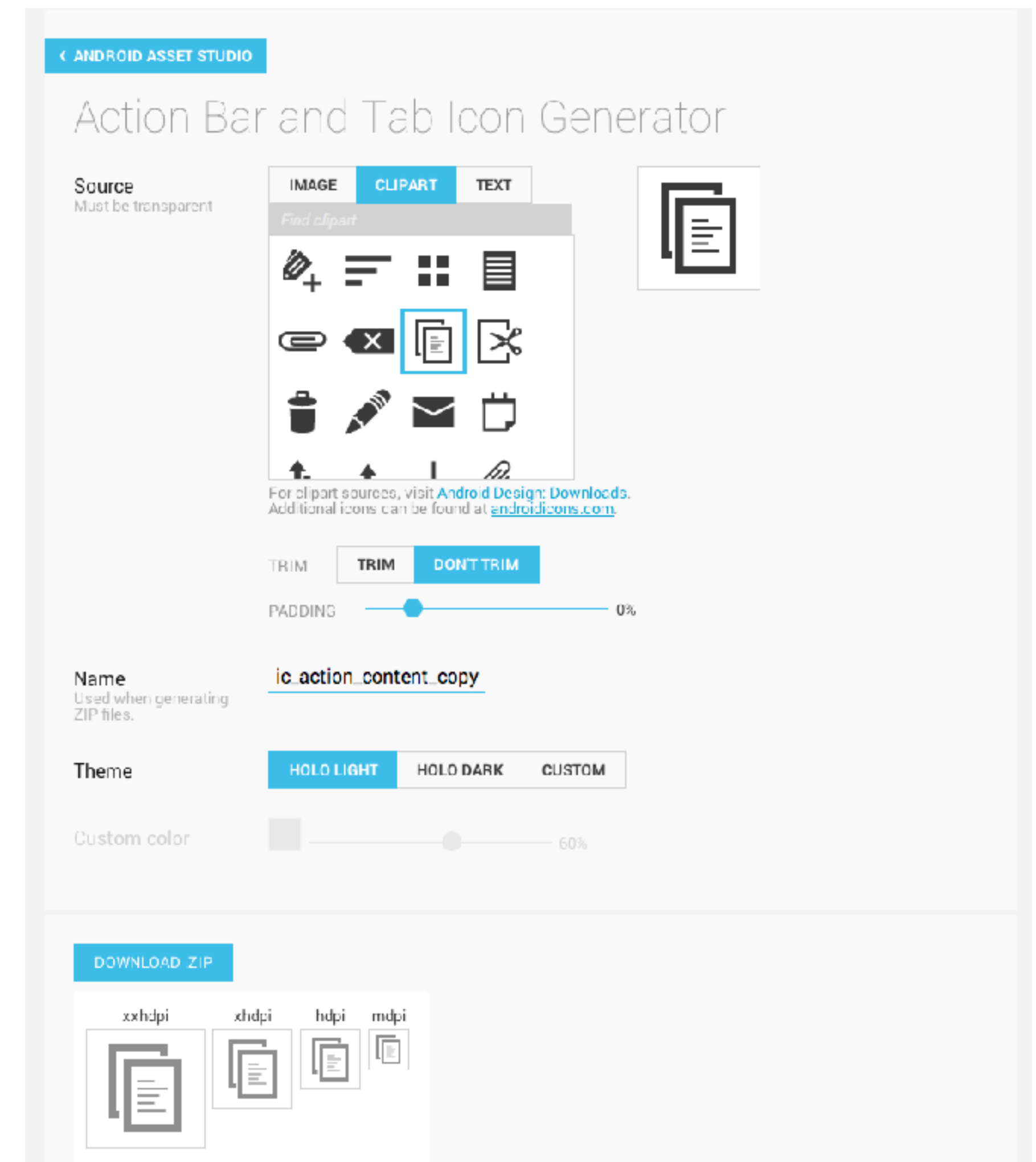
http://developer.android.com/guide/practices/ui_guidelines/icon_design.html

Tab Icon Design

http://developer.android.com/guide/practices/ui_guidelines/icon_design_tab.html

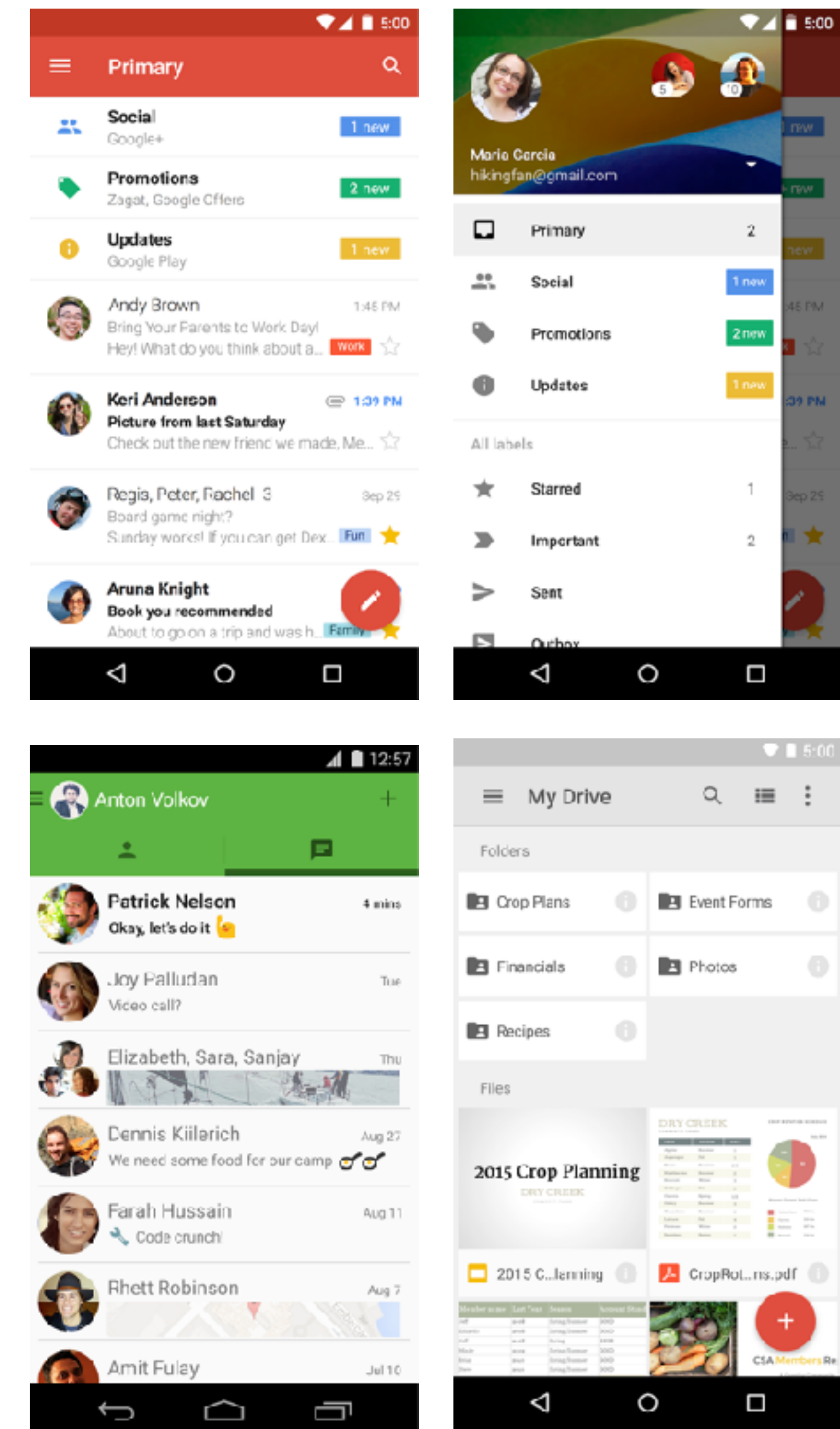
Android Icon Web Tool

<http://android-ui-utils.googlecode.com/hg/asset-studio/dist/index.html>



Lists

- Lists are a common and powerful way to visualise heterogenous contents inside your apps
- With material design in order to create complex lists you can use the **RecyclerView**
- **ListView** has been dismissed in behalf of RecyclerView and CardView (next lecture)
- RecyclerView is part of the support library.
- We had ListView for a very long time in Android and the concept of recycling views not new. In previous version (with ListView) the appearance, recycling and everything was tightly coupled. The new approach is more flexible with the new RecyclerView.



Source: <https://developer.android.com/training/material/lists-cards.html> Source: <http://www.grokkingandroid.com/first-glance-androids-recyclerview/>

RecyclerView

- The RecyclerView widget is a more advanced and flexible version of ListView
- It is a container for displaying large data sets that can be scrolled very efficiently by maintaining a limited number of views.
- Use the RecyclerView widget when you have data collections whose elements change at runtime based on user action or network events.
- The RecyclerView class simplifies the display and handling of large data sets by providing:
 - Layout managers for positioning items
 - Default animations for common item operations, such as removal or addition of items of invisible views around.

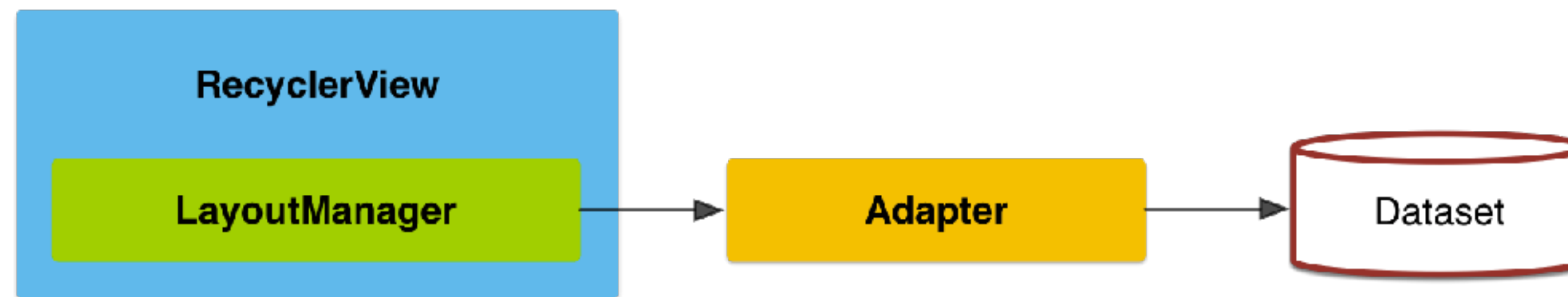
Source: <https://developer.android.com/training/material/lists-cards.html> Source: <http://www.grokkingandroid.com/first-glance-androids-recyclerview/>

RecyclerView

- The RecyclerView widget is a more advanced and flexible version of ListView
- It is a container for displaying large data sets that can be scrolled very efficiently by maintaining a limited number of views.
- Use the RecyclerView widget when you have data collections whose elements change at runtime based on user action or network events.
- The RecyclerView class simplifies the display and handling of large data sets by providing:
 - Layout managers for positioning items
 - Default animations for common item operations, such as removal or addition of items
- You also have the flexibility to define custom layout managers and animations for RecyclerView widgets.

Source: <https://developer.android.com/training/material/lists-cards.html> Source: <http://www.grokkingandroid.com/first-glance-androids-recyclerview/>

RecyclerView

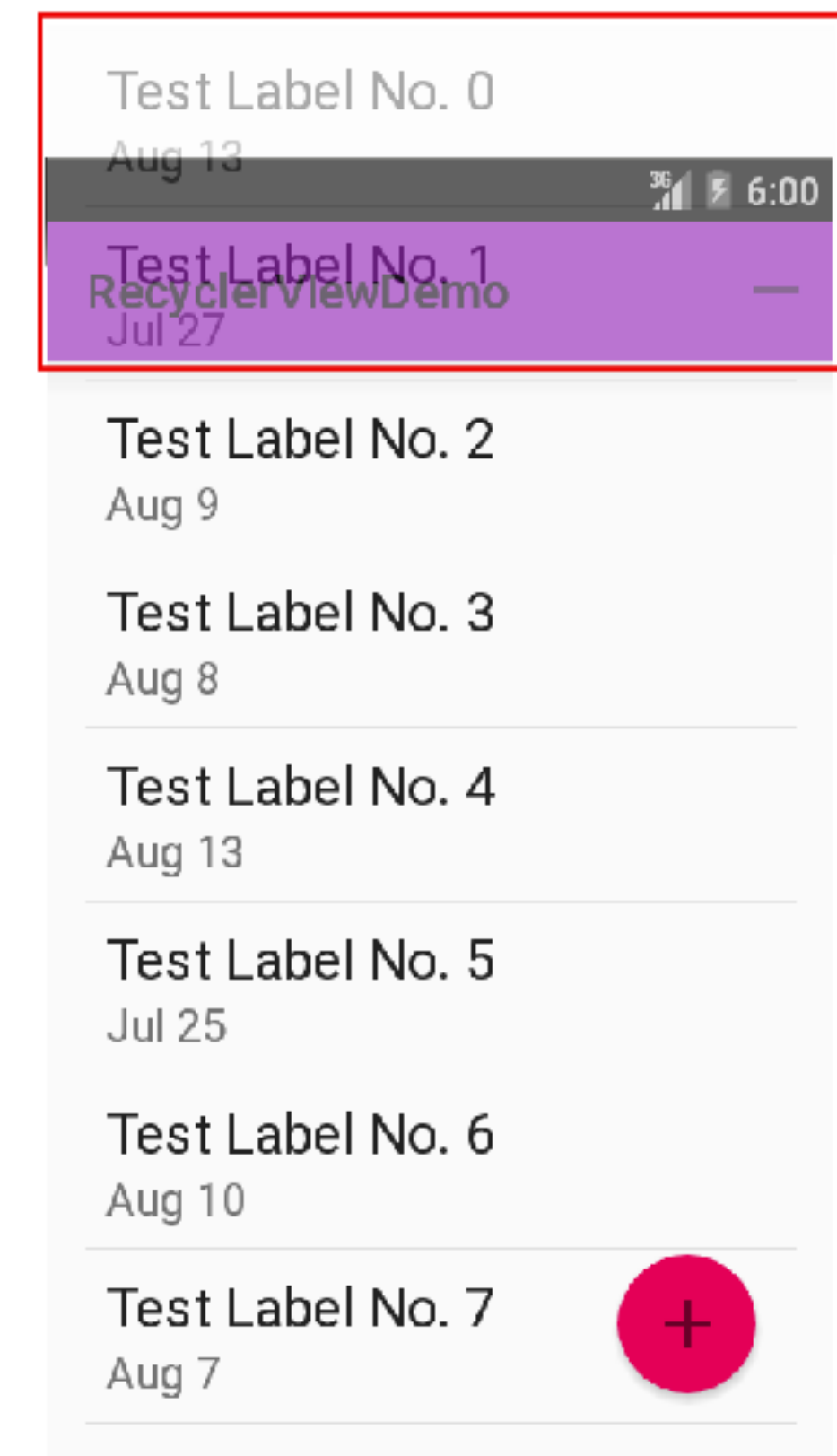
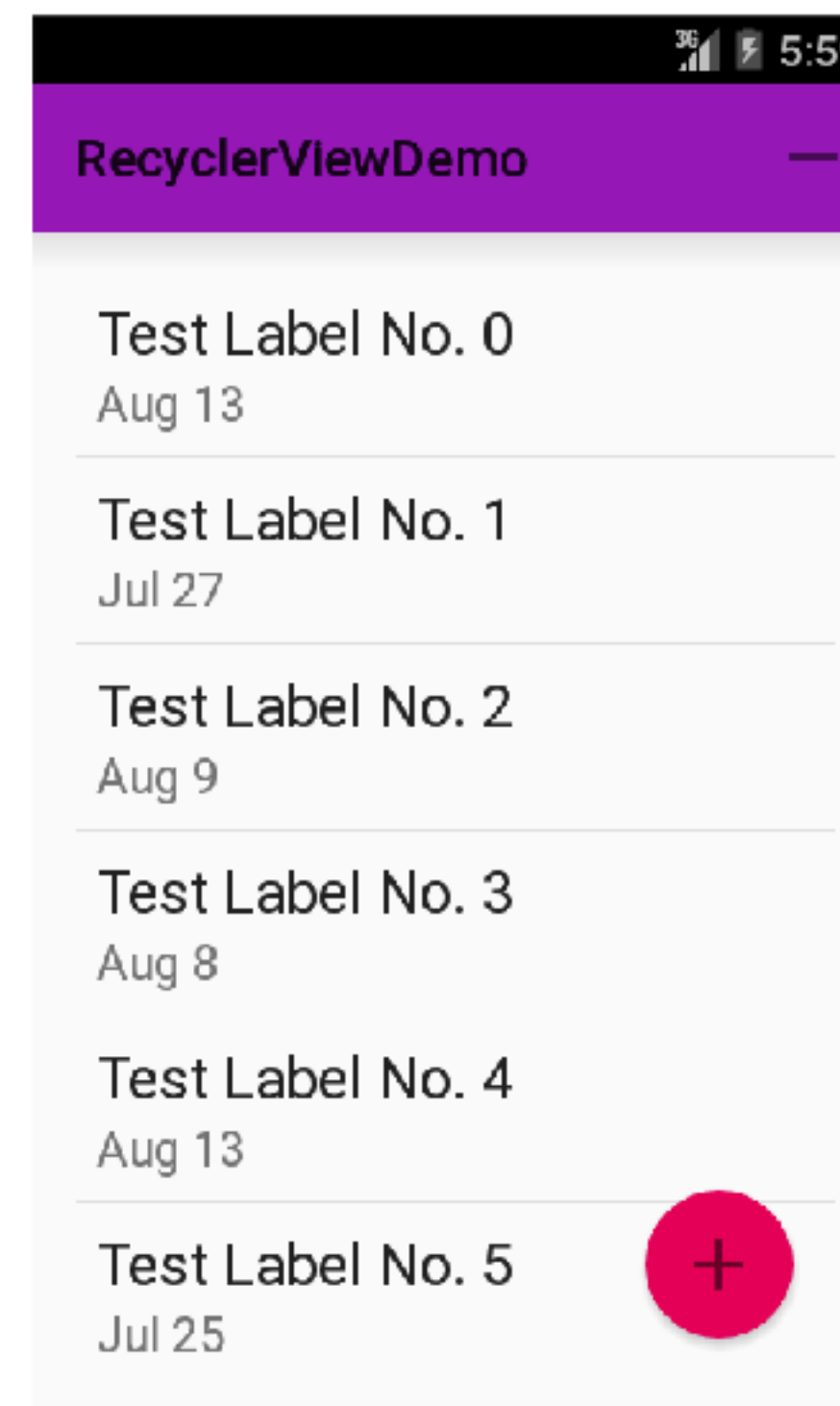


- The RecyclerView widget is based on an adapter and a layout manager.
- An Adapter is the bridge between the list and the data that fill the view.
 - The adapter is used to provide to the LayoutManager information about how many elements should be visualized (**getItemCount()**) and the actual View to use
- To create an adapter, extend the RecyclerView.Adapter class
- The details of the implementation depend on the specifics of your dataset and the type of views

Source: <https://developer.android.com/training/material/lists-cards.html> Source: <http://www.grokkingandroid.com/first-glance-androids-recyclerview/>

Layout Manager

- A layout manager positions item views inside a RecyclerView and determines when to reuse item views that are no longer visible to the user.
- To reuse (or recycle) a view, a layout manager may ask the adapter to replace the contents of the view with a different element from the dataset.
- Recycling views in this manner improves performance by avoiding the creation of unnecessary views or performing expensive findViewById() lookups.

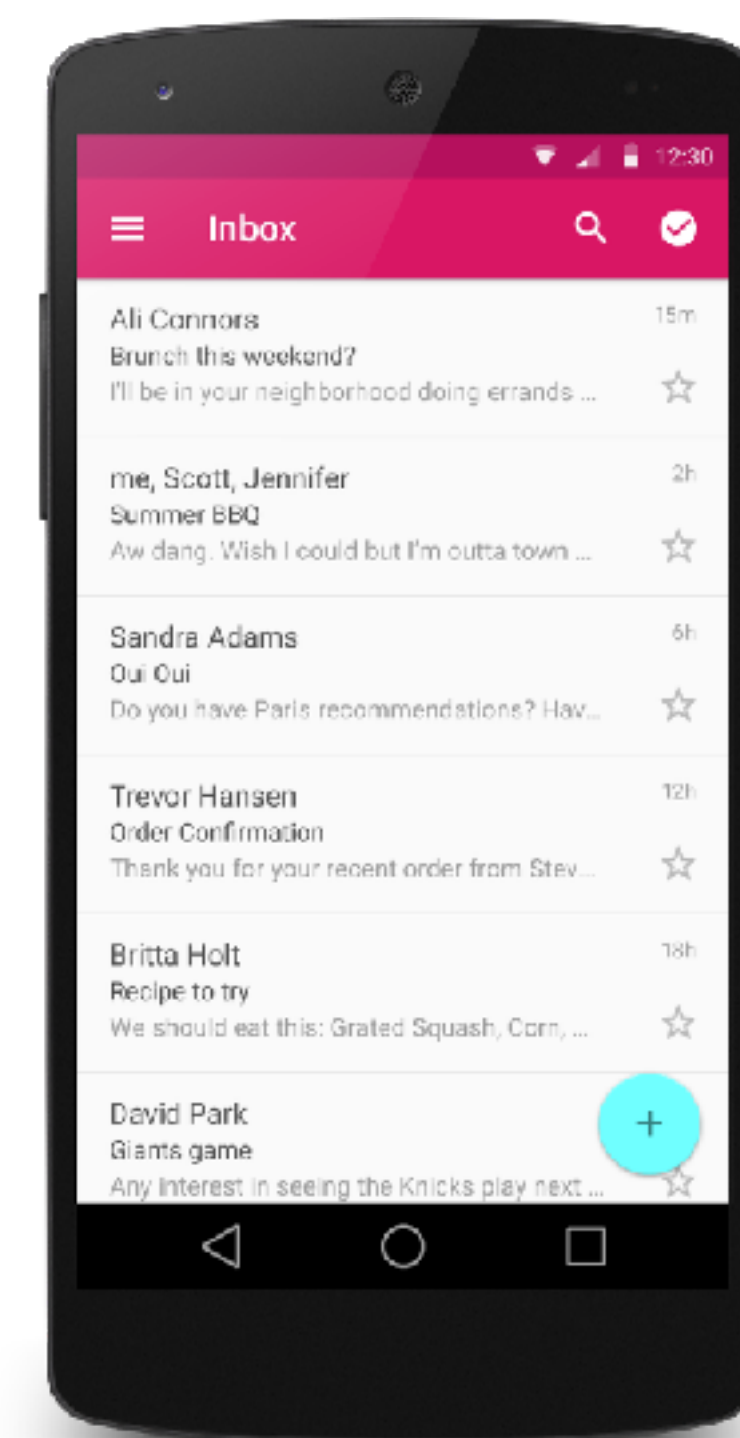


Source: <https://developer.android.com/training/material/lists-cards.html>

Source: <http://www.grokkingandroid.com/first-glance-androids-recyclerview/>

Layout Manager

- RecyclerView provides these built-in layout managers:
 - LinearLayoutManager shows items in a vertical or horizontal scrolling list
 - GridLayoutManager shows items in a grid
 - StaggeredGridLayoutManager shows items in a staggered grid
- To create a custom layout manager, extend the RecyclerView.LayoutManager class
- Recycling of views is a very useful approach. It saves CPU resources in that you do not have to inflate new views all the time and it saves memory in that it doesn't keep plent



Source: <https://developer.android.com/training/material/lists-cards.html>

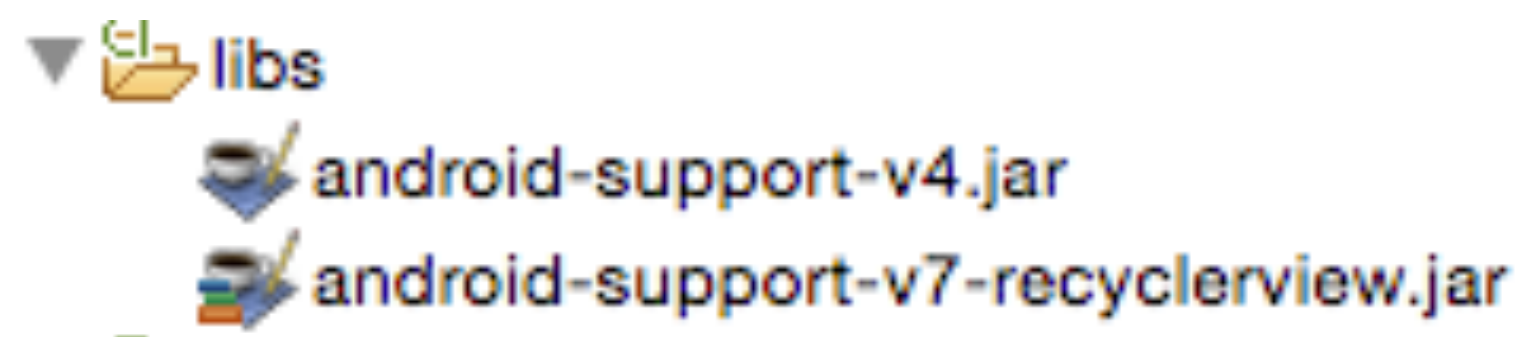
Source: <http://www.grokkingandroid.com/first-glance-androids-recyclerview/>

RecyclerView Dependencies

AndroidStudio

```
dependencies {  
    ...  
    compile 'com.android.support:recyclerview-v7:21.0.+'  
}
```

Eclipse



Copy the following Jar to the libs folder of your project

`$SDK_HOME$/extras/android/support/v7/recyclerview/libs/android-support-v7-recyclerview`

Recycler View XML

```
<android.support.v7.widget.RecyclerView  
    android:id="@+id/my_recycler_view"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:scrollbars="vertical" />
```

When you have added a RecyclerView widget to your layout you can obtain a handle to the object, connect it to a layout manager, and attach an adapter for the data to be displayed.

Recycler View Code

```
mRecyclerView = (RecyclerView)rootView.findViewById(R.id.my_recycler_view);
```

Here we are in a
Fragment :)

```
// use a linear layout manager
```

```
mLayoutManager = new LinearLayoutManager(getActivity());
```

```
mLayoutManager.setOrientation(LinearLayoutManager.VERTICAL);
```

```
mLayoutManager.scrollToPosition(0);
```

Scroll to a
specific position

```
mRecyclerView.setLayoutManager(mLayoutManager);
```

```
// use this setting to improve performance if you know that changes
```

```
// in content do not change the layout size of the RecyclerView
```

```
mRecyclerView.setHasFixedSize(true);
```

Your Layout
Manager

```
// specify an adapter (see also next example)
```

```
mAdapter = new MyAdapter(myDataSet);
```

```
mRecyclerView.setAdapter(mAdapter);
```

Your Adapter
Implementation

Recycler Adapter

- The adapter provides:
 - access to the items in your data set
 - creates views for items
 - replaces the content of some of the views with new data items when the original item is no longer visible

```
public class MyAdapter extends RecyclerView.Adapter<MyAdapter.ViewHolder> {  
  
    private ArrayList<Double> mDataset;  
  
    // Provide a reference to the views for each data item  
    // Complex data items may need more than one view per item, and  
    // you provide access to all the views for a data item in a view holder  
    public class ViewHolder extends RecyclerView.ViewHolder {  
        [...]  
    }  
  
    // Provide a suitable constructor (depends on the kind of dataset)  
    public MyAdapter(ArrayList<Double> myDataset) {  
        mDataset = myDataset;  
    }  
  
    // Create new views (invoked by the layout manager)  
    @Override  
    public MyAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {  
        [...]  
    }  
  
    // Replace the contents of a view (invoked by the layout manager)  
    @Override  
    public void onBindViewHolder(ViewHolder holder, final int position) {  
        [...]  
    }  
  
    // Return the size of your dataset (invoked by the layout manager)  
    @Override  
    public int getItemCount() {  
        return mDataset.size();  
    }  
}
```

Recycler Adapter

```
// Create new views (invoked by the layout manager)
@Override
public MyAdapter.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

    // create a new view
    View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.listview_element, parent, false);
    // set the view's size, margins, paddings and layout parameters

    ViewHolder vh = new ViewHolder(v);

    return vh;
}

// Replace the contents of a view (invoked by the layout manager)
@Override
public void onBindViewHolder(ViewHolder holder, final int position) {
    // - get element from your dataset at this position
    // - replace the contents of the view with that element
    holder.setText(""+mDataset.get(position));
}
```

View Holder

ViewHolders are basically caches of your View objects.

```
public class ViewHolder extends RecyclerView.ViewHolder {
```

```
    private View v = null;
```

```
    public ViewHolder(View v) {  
        super(v);  
        this.v = v;
```

View Inflated and provided
through the method
onCreateViewHolder()

```
        v.setOnLongClickListener(new OnLongClickListener() {
```

```
            @Override
```

```
            public boolean onLongClick(View v) {
```

```
                int position = getPosition();
```

```
                NumberManager.getInstance().removeNumber(position);
```

```
                notifyItemRemoved(position);
```

```
                return false;
```

```
            }
```

```
        });
```

```
    }
```

```
    public void setText(String text){
```

```
        TextView tView = (TextView)v.findViewById(R.id.myTextView);
```

```
        tView.setText(text);
```

```
    }
```

```
}
```

Click Listener for the View.

You can use both
setOnLongClickListener or
setOnClickListener

Animation

- The ItemAnimator class helps the RecyclerView with animating individual items. ItemAnimators deal with three events:
 - An item gets added to the data set
 - An item gets removed from the data set
 - An item moves as a result of one or more of the previous two operations
- Luckily a default implementation exists and is called DefaultItemAnimator. If you do not set a custom ItemAnimator, RecyclerView uses an instance of DefaultItemAnimator.
- The RecyclerView.Adapter class contains plenty of notifyXyz() methods. The two most specific are:
 - `public final void notifyItemInserted(int position)`
 - `public final void notifyItemRemoved(int position)`

Animation

Insert Animation



```
addButton.setOnClickListener(new OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        Log.d(MainActivity.TAG, "Add Button Clicked !");  
  
        Random rand = new Random();  
        int number = rand.nextInt((1000 - 0) + 1);  
        NumberManager.getInstance().addNumberToHead(Double.valueOf(number));  
        mAdapter.notifyItemInserted(0);  
  
        mLayoutManager.scrollToPosition(0);  
    }  
});
```

Remove Animation



```
v.setOnLongClickListener(new OnLongClickListener() {  
    @Override  
    public boolean onLongClick(View v) {  
        int position = getPosition();  
        NumberManager.getInstance().removeNumber(position);  
        notifyItemRemoved(position);  
        return false;  
    }  
});
```

Handle Multiple View Types

- The RecyclerView Adapter allows you to define and handle multiple and heterogeneous View types for your list
- Override the method `getItemViewType()` in order to return the view type of the item at position for the purposes of view recycling
- The default implementation of this method returns 0, making the assumption of a single view type for the adapter
- Unlike ListView adapters, types need not be contiguous. Consider using id resources to uniquely identify item view types
- The combination of `getItemViewType()` with multiple ViewHolder allows you to handle and create lists with heterogeneous elements

```
@Override  
public int getItemViewType(int position) {  
    [...]  
}
```

Multiple View Holders

```
public class MyAdapter extends RecyclerView.Adapter<RecyclerView.ViewHolder> { ... }
```



Since we have multiple holders in our List use as reference a generic ViewHolder.

```
public class FirstViewHolder extends RecyclerView.ViewHolder {  
  
    private View v = null;  
  
    public EvenNumberViewHolder(View v) {  
        super(v);  
        this.v = v;  
  
        [...]  
    }  
  
    public void updateUI(String text){  
        [...]  
    }  
}
```

First ViewHolder

```
public class SecondViewHolder extends RecyclerView.ViewHolder {  
  
    private View v = null;  
  
    public OddNumberViewHolder(View v) {  
        super(v);  
        this.v = v;  
  
        [...]  
    }  
  
    public void updateUI(String text){  
        [...]  
    }  
}
```

Second ViewHolder

Manage Multiple ViewHolders

Inflate the
correct View
according to
the viewType
parameter.



```
@Override
public RecyclerView.ViewHolder onCreateViewHolder(ViewGroup parent, int viewType) {

    if(viewType == 0){
        View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.first_listview_element, parent, false);
        return new FirstViewHolder(v);
    }
    else if(viewType == 1){
        View v = LayoutInflater.from(parent.getContext()).inflate(R.layout.second_listview_element, parent, false);
        return new SecondViewHolder(v);
    }
}
```

Cast the
holder
according to
the type to
properly
handle and
update the UI



```
@Override
public void onBindViewHolder(ViewHolder holder, final int position) {

    if(getItemViewType(position) == 0){
        FirstViewHolder firstViewHolder = (FirstViewHolder)holder;
        firstViewHolder.updateUI(...);
    }
    else if(getItemViewType(position) == 1){
        SecondViewHolder secondViewHolder = (SecondViewHolder)holder;
        secondViewHolder.updateUI(...);
    }
}
```

Card View

- CardView extends the `FrameLayout` class and lets you show information inside cards that have a consistent look across the platform.
- CardView widgets can have shadows and rounded corners.
- To create a card with a shadow, use the `card_view:cardElevation` attribute. CardView uses real elevation and dynamic shadows on Android 5.0 (API level 21) and above and falls back to a programmatic shadow implementation on earlier versions.
- Use these properties to customize the appearance of the CardView widget:
 - To set the corner radius in your layouts, use the `card_view:cardCornerRadius` attribute.
 - To set the corner radius in your code, use the `CardView.setRadius` method.
 - To set the background color of a card, use the `card_view:cardBackgroundColor` attribute.



Card View Dependences

AndroidStudio

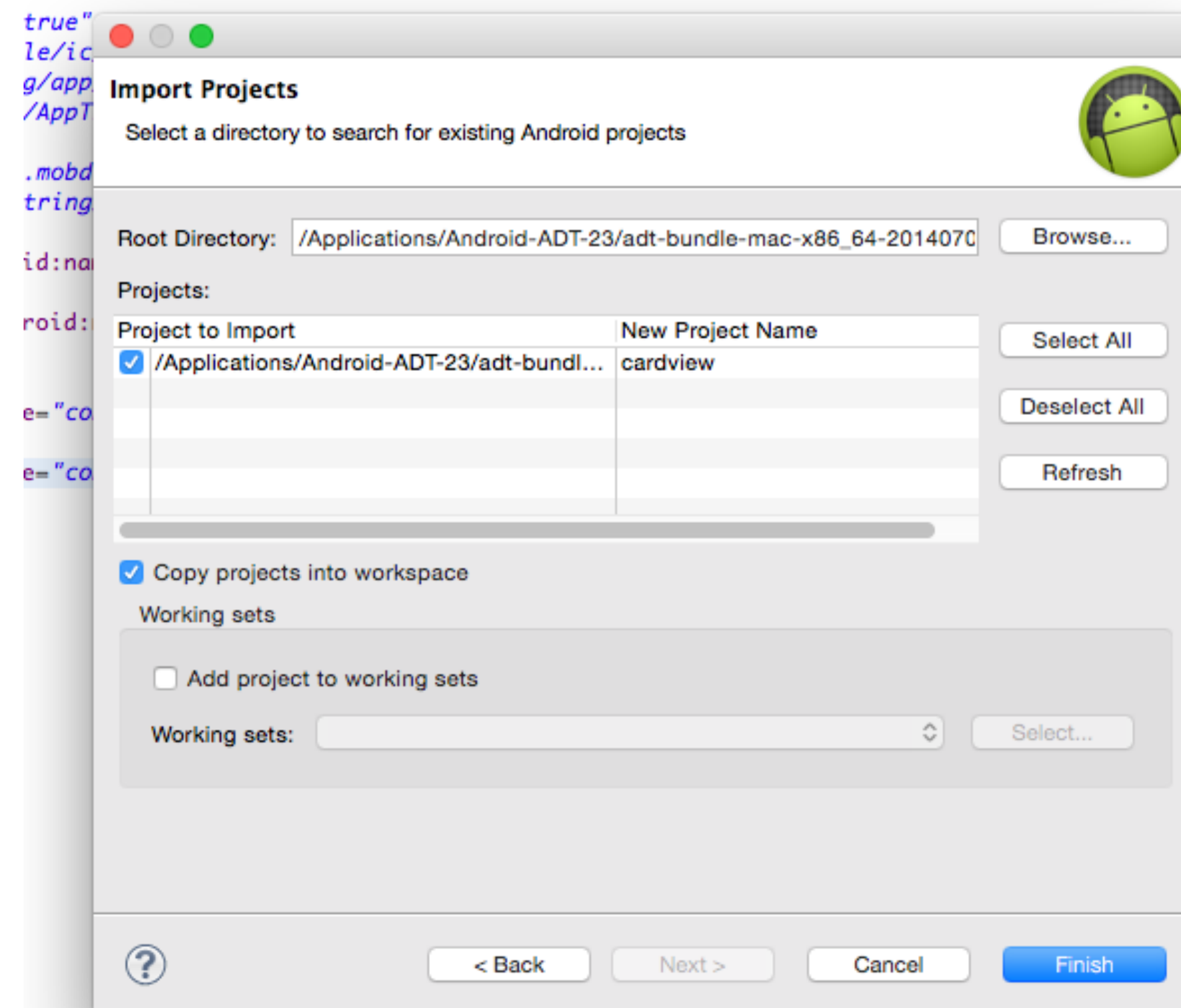
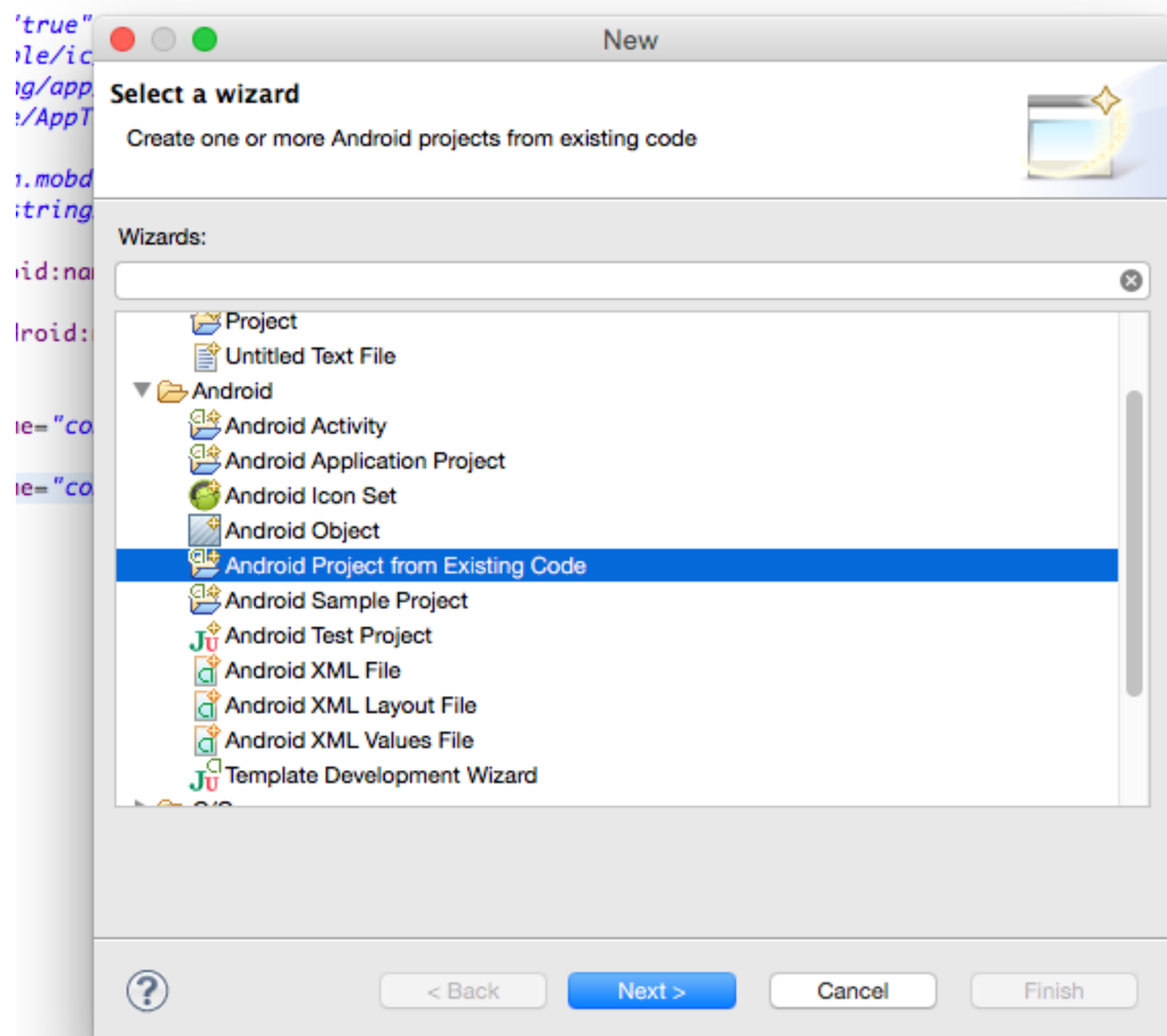
```
dependencies {  
    ...  
    compile 'com.android.support:cardview-v7:21.0.+'  
}
```

Eclipse

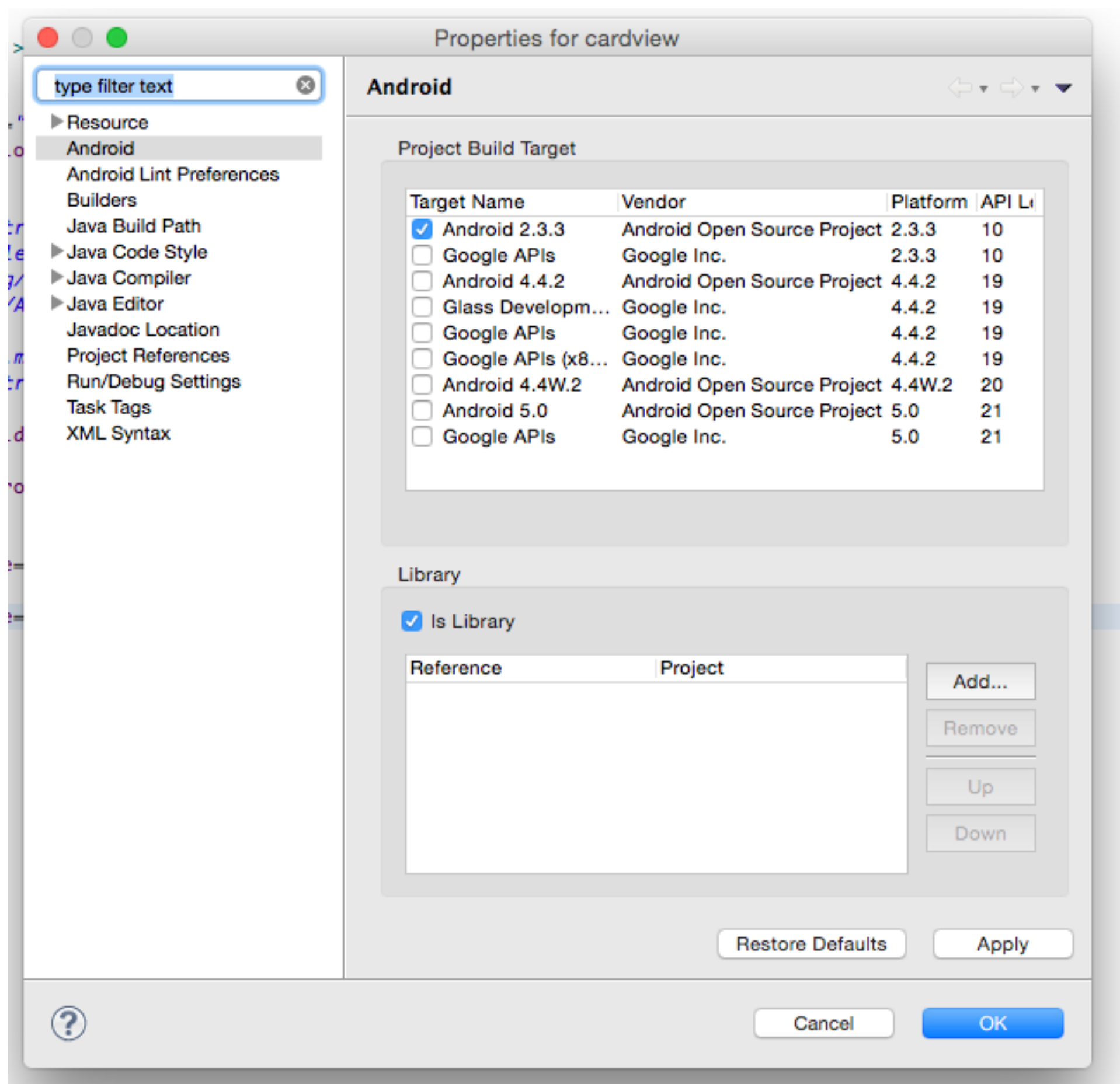
Import and Copy in to your workspace the following project and make it a Library. Then link the cardview library in your own projects.

`$SDK_HOME$/extras/android/support/v7/cardview`

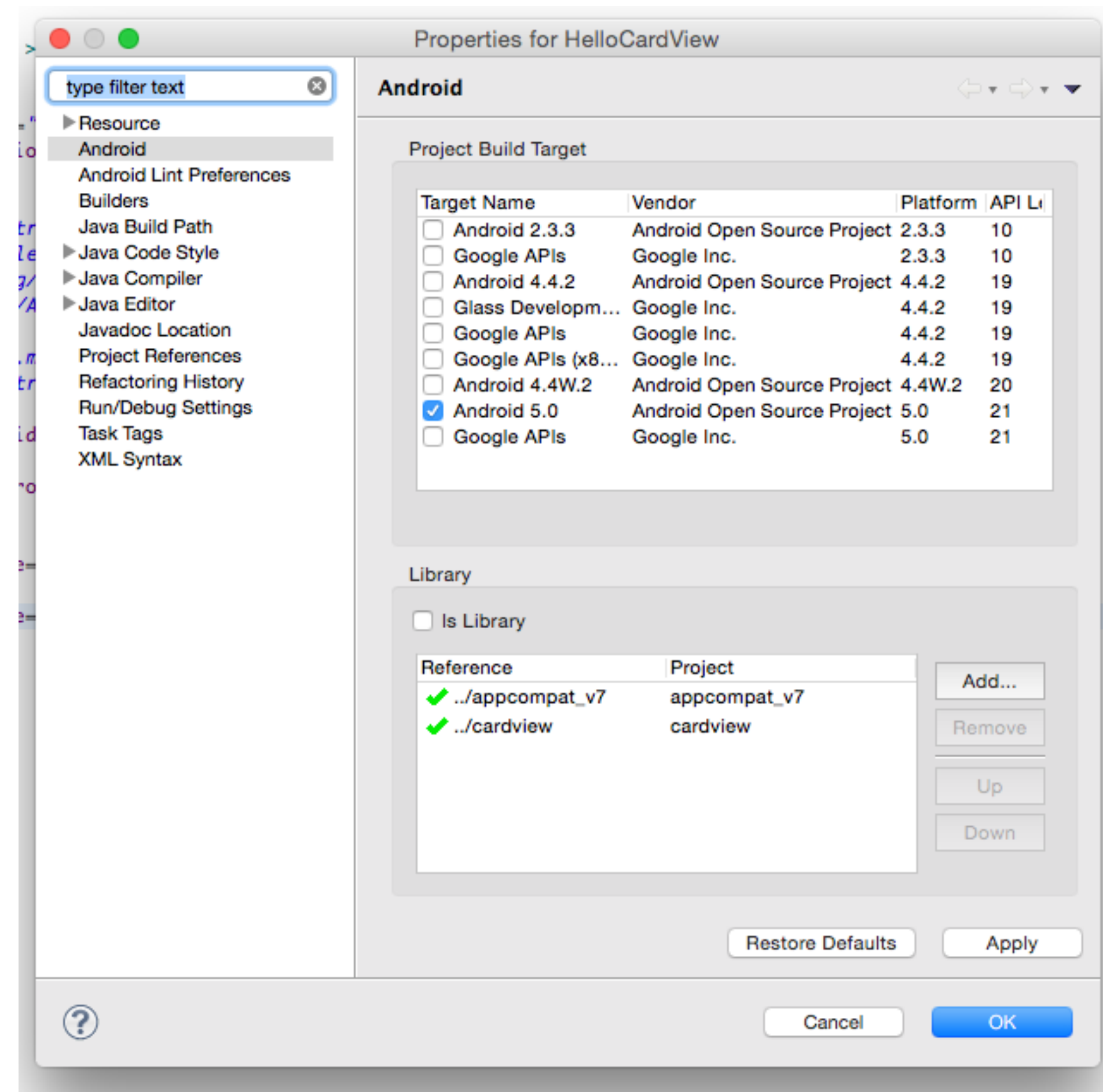
Card View Dependences



Card View Dependencies



CardView Project



Your Project with CardView

Card View In your Layout

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_marginLeft="10dp"
    android:layout_marginRight="10dp"
    android:background="@drawable/listview_selector"
    android:orientation="vertical" >
```

```
<android.support.v7.widget.CardView
    xmlns:card_view="http://schemas.android.com/apk/res-auto"
    android:id="@+id/card_view"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_margin="10dp"
    android:layout_gravity="center"
    android:elevation="10dp"
    card_view:cardCornerRadius="4dp" >

    <TextView
        android:id="@+id/myTextView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:text="Large Text"
        android:textAppearance="?android:attr/textAppearanceMedium" />
</android.support.v7.widget.CardView>
```

```
</LinearLayout>
```

←

Extends FrameLayout so is easier to use only a single child. For example a simple TextView or a ViewGroup Layout for more complex Cards.



Android Development

Lecture 3

Android Graphical User Interface 2



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