



Android Development

Lecture 4

Android Graphical User Interface 3



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

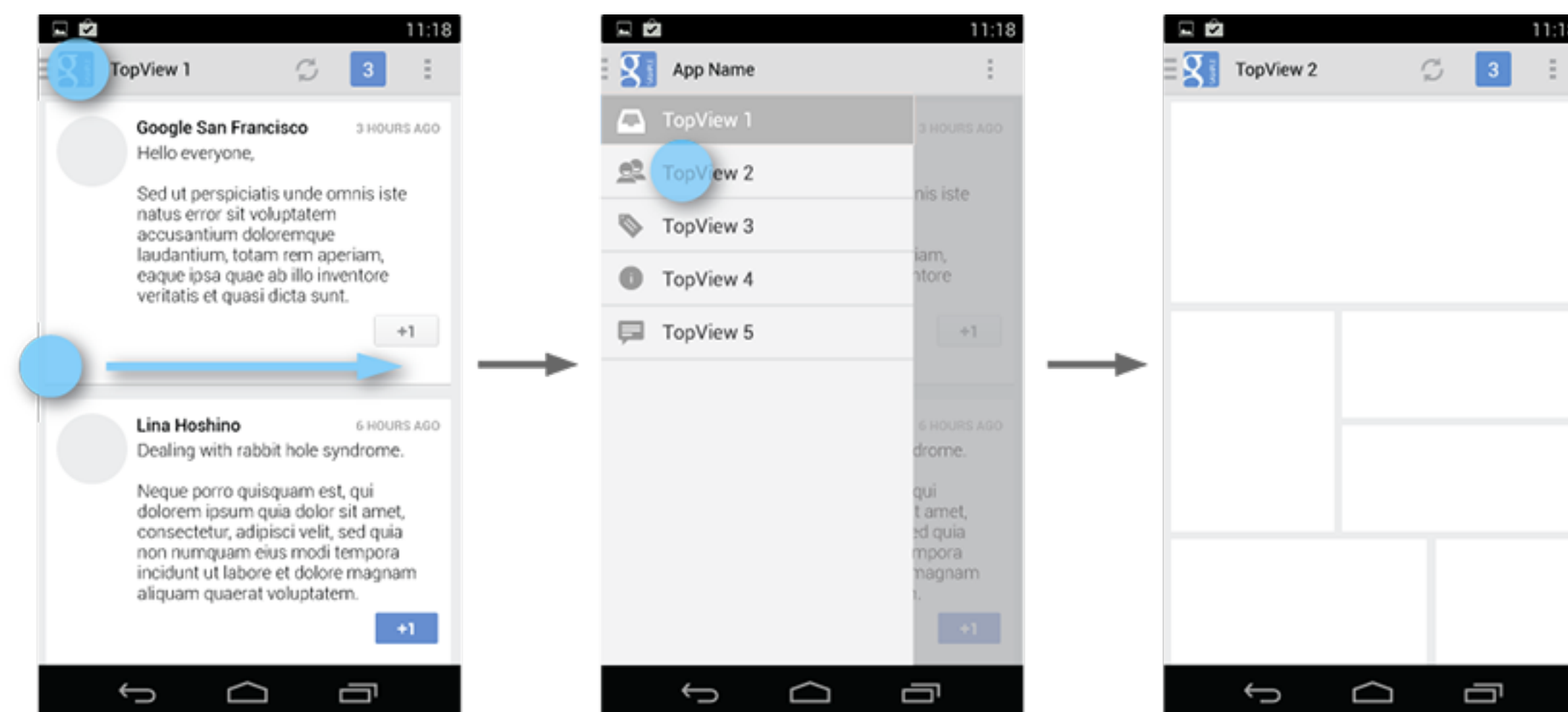
Lecture Summary

- Navigation Drawer
- View Pager
- Tabs
- Toast Notifications
- Dialogs



Navigation Drawer

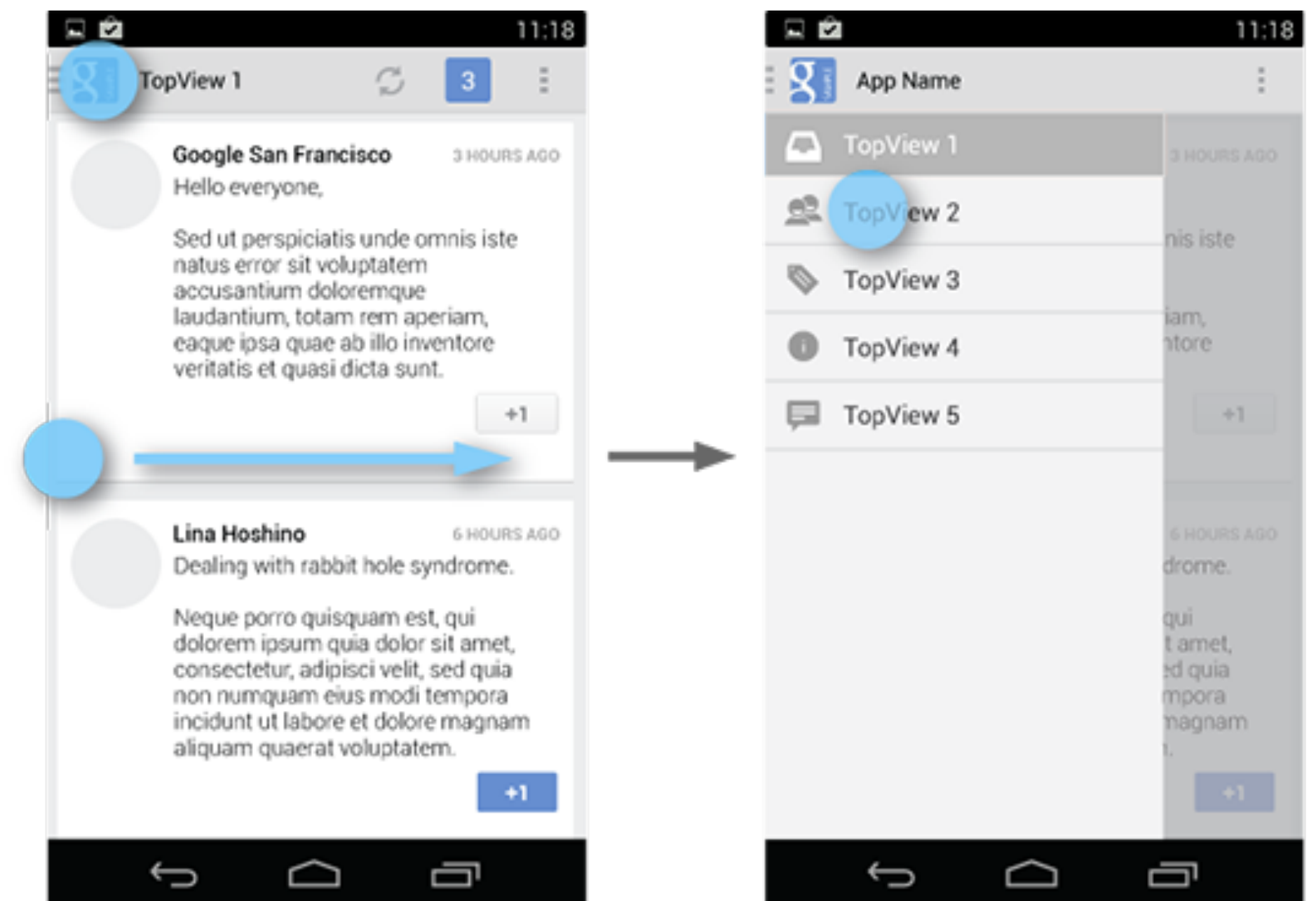
- The navigation drawer is a panel that transitions in from the left edge of the screen and displays the app's main navigation options.
- The user can bring the navigation drawer onto the screen by swiping from the left edge of the screen or by touching the application icon on the action bar.
- As the navigation drawer expands, it overlays the content but not the action bar. (If you are using a ToolBar you can change this behaviour and overlay also the ActionBar/ToolBar)
- The overflow menu with the standard action items for Settings and Help remains visible.



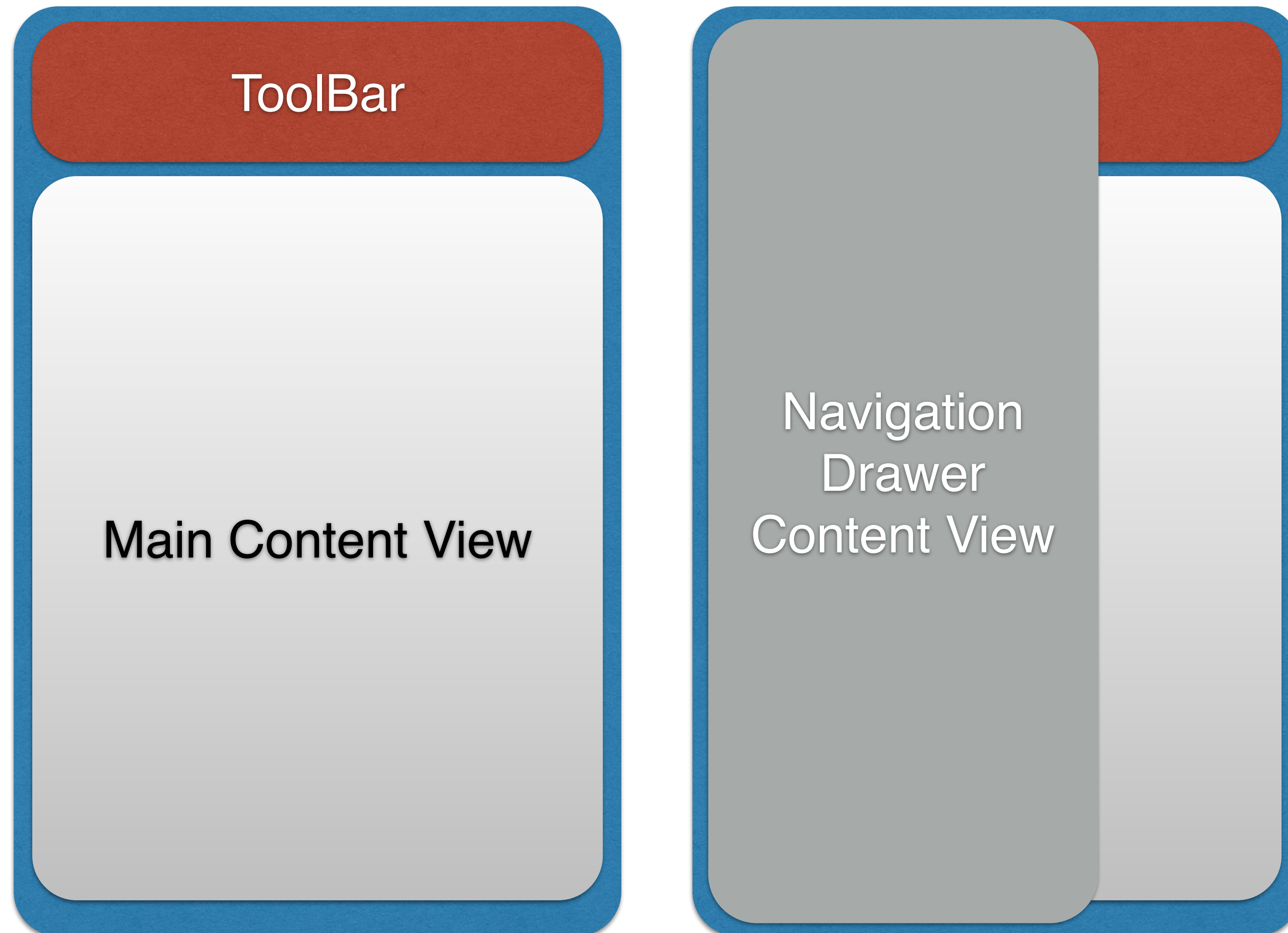
Source: <https://developer.android.com/design/patterns/navigation-drawer.html>

When to Use the Navigation Drawer

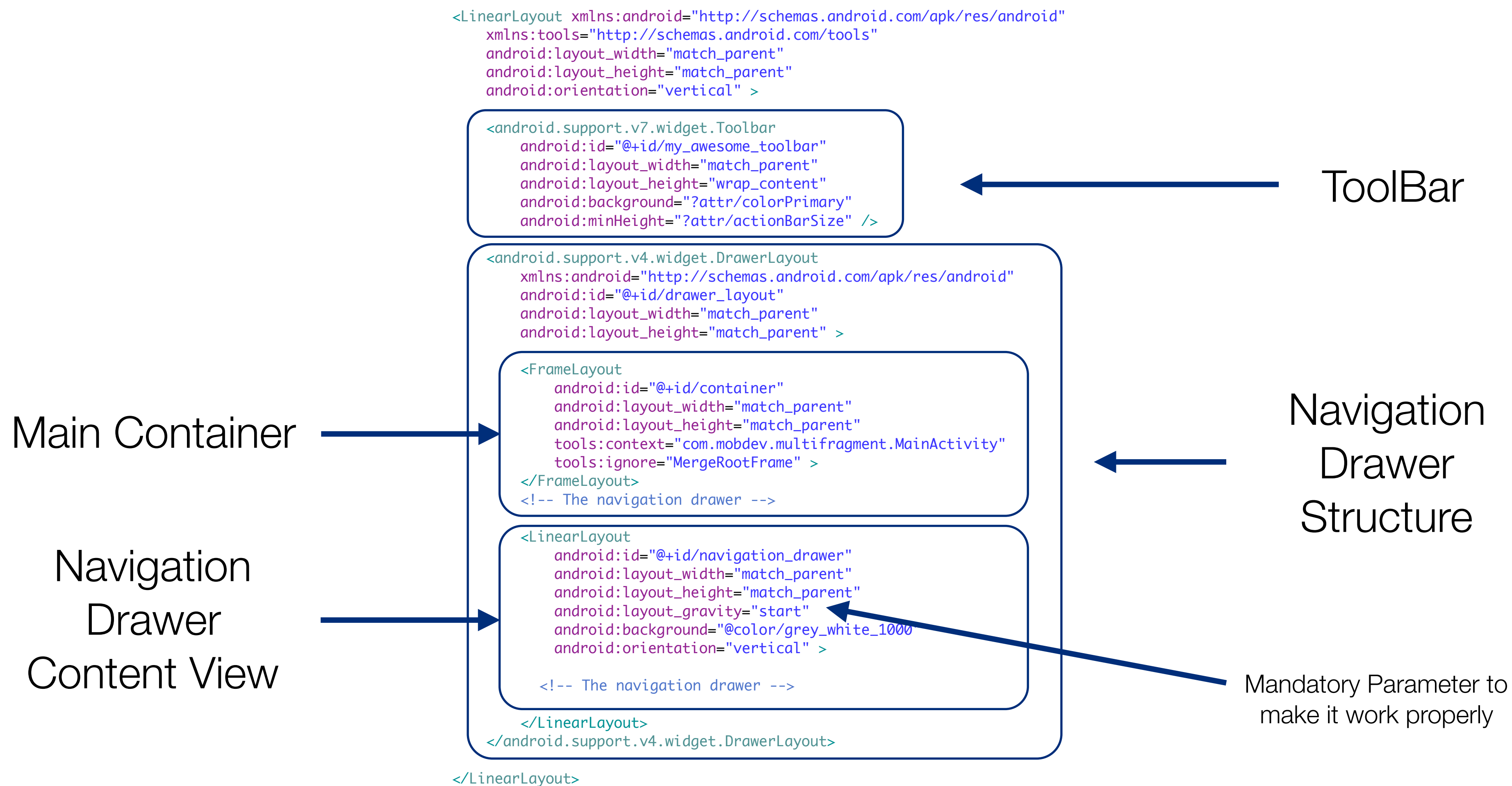
- The navigation drawer is not a general replacement for top-level navigation via spinners or tabs.
- The structure of your app should guide your choice of which pattern to use for top-level switching.
- A good example where navigation drawers work best is for example if you have more than 3 top-level views
- Navigation drawers are great for displaying a large number of navigation targets concurrently. Use the navigation drawer if you have more than 3 unique top-level views. If not, use fixed tabs for top-level organization to ease discovery and interaction.



Navigation Drawer Structure



Navigation Drawer Structure (XML)



Navigation Drawer Structure (Code)

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

    if (savedInstanceState == null) {
        randomFragment = new RandomNumberFragment();
        getSupportFragmentManager().beginTransaction().add(R.id.container, randomFragment).commit();
    }
}
```

```
Toolbar toolbar = (Toolbar)findViewById(R.id.my_awesome_toolbar);
setSupportActionBar(toolbar);
```

```
mDrawerLayout = (DrawerLayout) findViewById(R.id.drawer_layout);

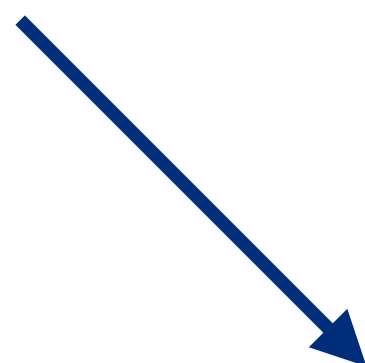
toggle = new ActionBarDrawerToggle(this, mDrawerLayout, toolbar, R.string.drawer_open, R.string.drawer_close);
toggle.setDrawerIndicatorEnabled(true);

mDrawerLayout.setDrawerListener(toggle);
```

```
setupDrawerUI();
```

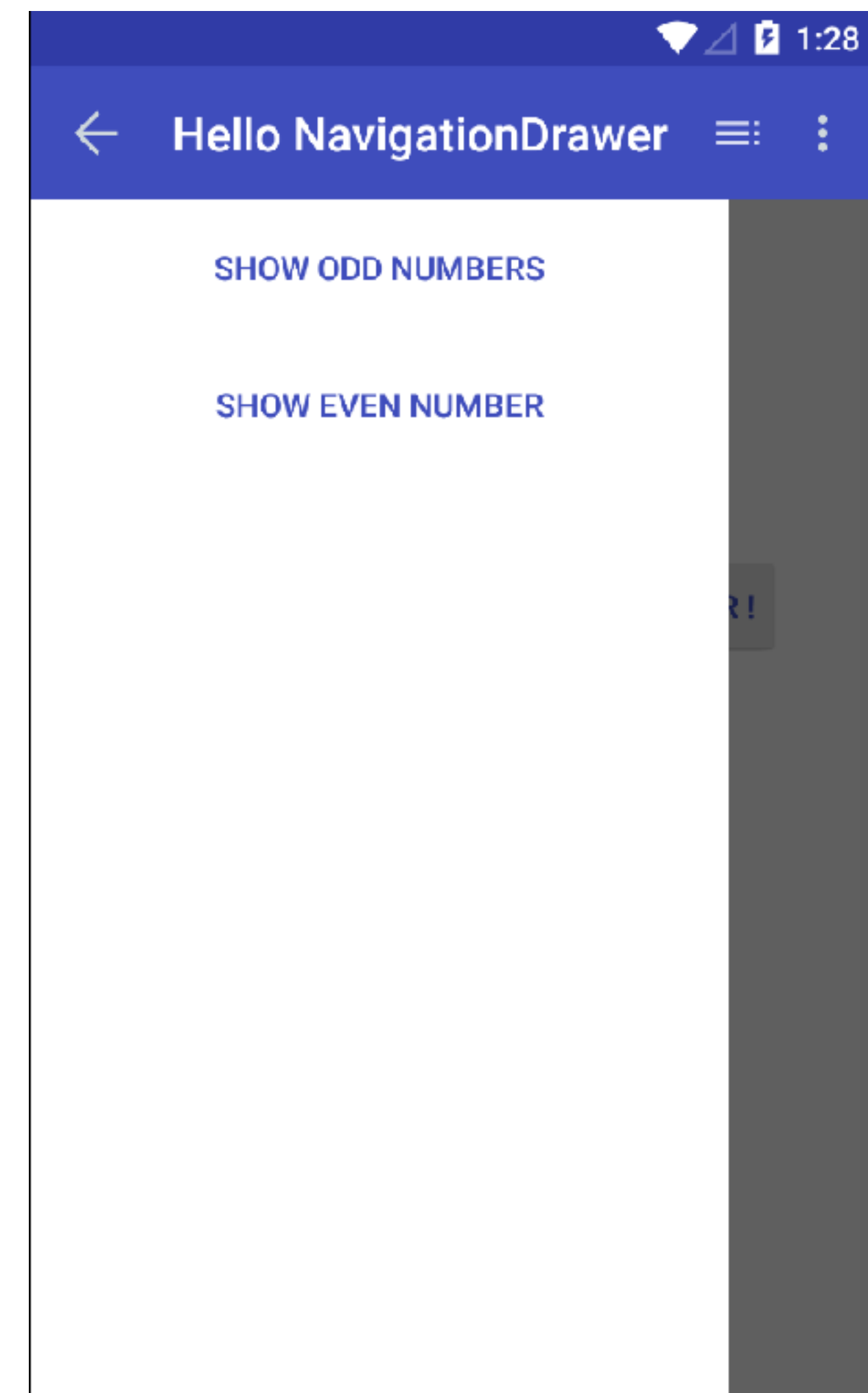
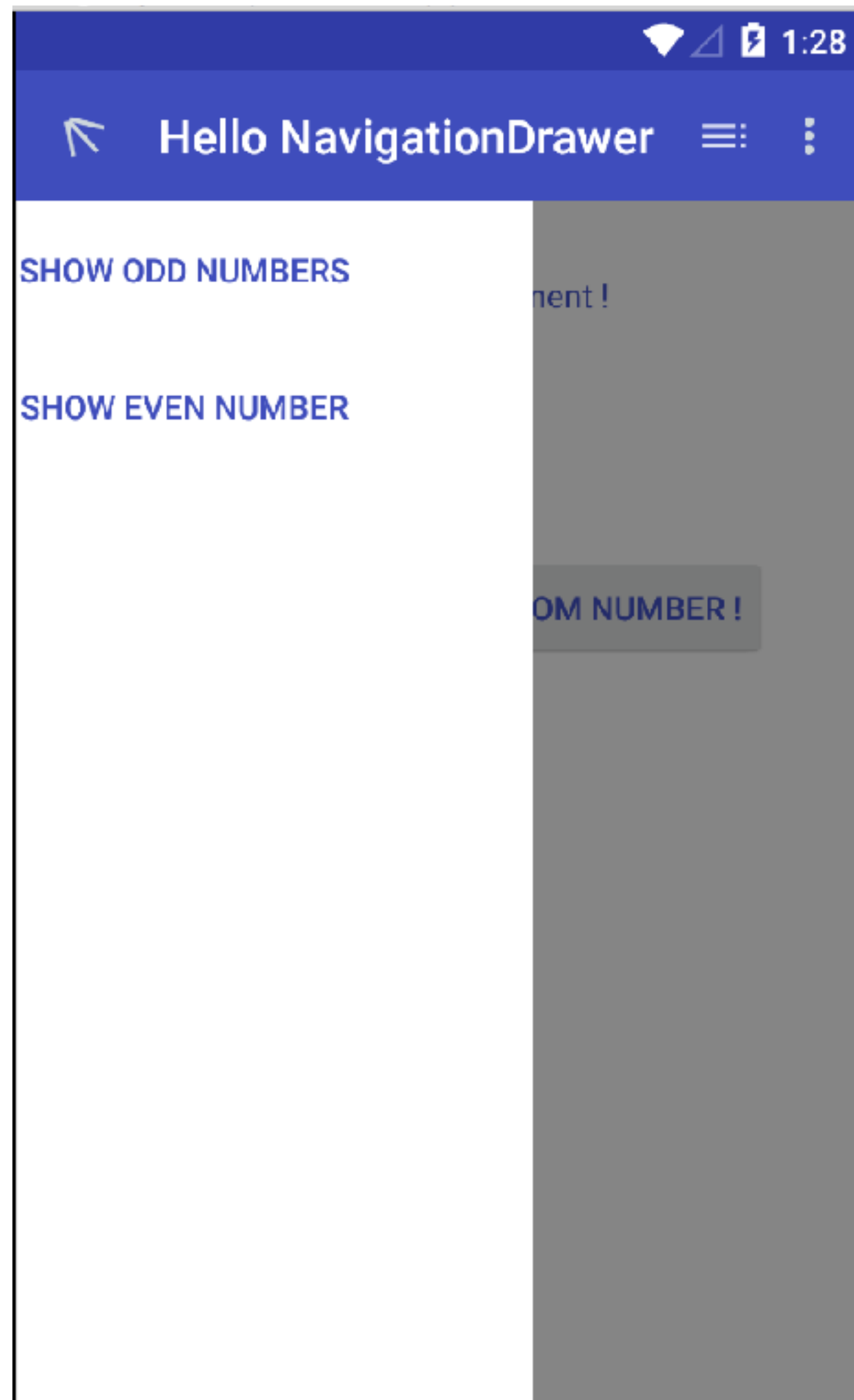
```
@Override
protected void onPostExecute(Bundle savedInstanceState) {
    super.onPostExecute(savedInstanceState);
    toggle.syncState();
}
```

Binding
between the
ToolBar and
the Drawer



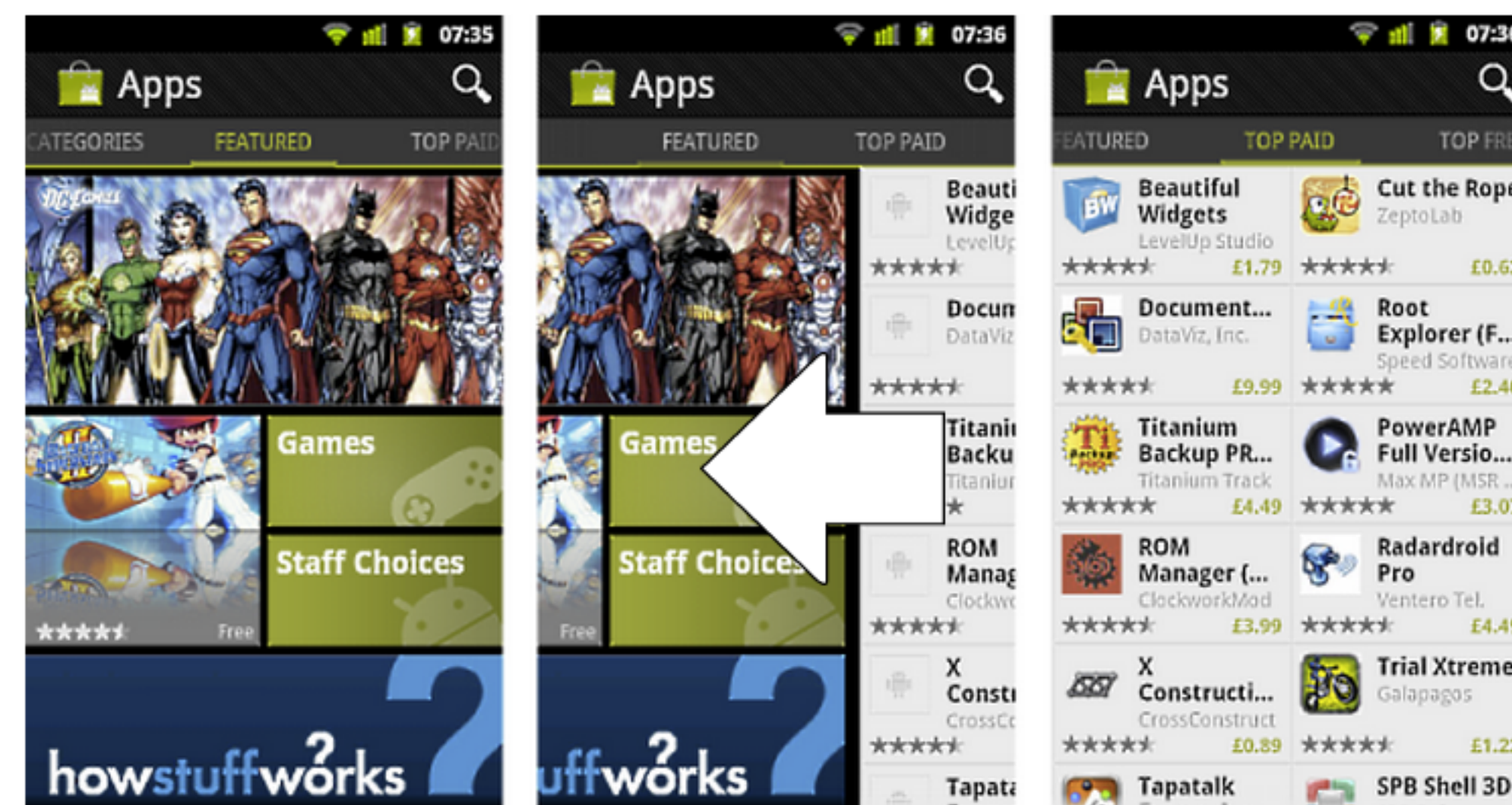
A custom and internal method
to setup the content of the
Drawer that has been defined in
the XML

Navigation Drawer



ViewPager

- ViewPagers have built-in swipe gestures to transition through pages.
- They display screen slide animations by default.
- ViewPagers use PagerAdapters as a supply for new pages to display.
- PagerAdapter will use the Fragments to define the content of each page.
- ViewPager is a ViewGroup that works in a similar way to AdapterViews (like ListView and Gallery)
- ViewPager was released as part of the Compatibility Package revision 3 and works with Android 1.6 upwards.
- **[Note]** If you use the ViewPager in an xml layout, be sure to use the full class reference.



Additional Info (E.g Customized Animation)

<http://developer.android.com/training/animation/screen-slide.html>

ViewPager - Implementation

- Sets the content view to be the layout with the ViewPager
- Creates a class that extends the **FragmentStatePagerAdapter** abstract class and implements the `getItem()` method to supply instances of `ScreenSlidePageFragment` as new pages. The pager adapter also requires that you implement the `getCount()` method, which returns the amount of pages the adapter will create (five in the example).
- Hooks up the `PagerAdapter` to the `ViewPager`.
- Handles the device's back button by moving backwards in the virtual stack of fragments. If the user is already on the first page, go back on the activity back stack.

```
<android.support.v4.view.ViewPager
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/pager"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

ViewPager - Implementation

```
public class AppSectionsPagerAdapter extends FragmentPagerAdapter {

    public AppSectionsPagerAdapter(FragmentManager fm) {
        super(fm);
    }

    /**
     * Returns the Fragment associated to the element in position 'i'
     */
    @Override
    public Fragment getItem(int i) {
        switch (i) {
            case 0:
                return new BookmarkGridFragment();
            case 1:
                return new BookmarkListFragment();
            default:
                return new BookmarkGridFragment();
        }
    }
}
```

```
/**
 * Returns the number of element in the ViewPager
 */
@Override
public int getCount() {
    return 2;
}

/**
 * Returns the title (as CharSequence) of the Page at position 'position'
 */
@Override
public CharSequence getPageTitle(int position) {
    switch (position) {
        case 0:
            return "Add";
        case 1:
            return "Top";
        case 2:
            return "My List";
        default:
            return "No Title !";
    }
}
}
```

ViewPager - Implementation

Create the ViewPager Adapter

```
mAppSectionsPagerAdapter = new AppSectionsPagerAdapter(getSupportFragmentManager());
```

```
// Set up the ViewPager, attaching the adapter and setting up a listener for when the  
// user swipes between sections.
```

```
mViewPager = (ViewPager) findViewById(R.id.pager);
```

```
mViewPager.setAdapter(mAppSectionsPagerAdapter);
```

Retrieve the Pager and set the adapter

```
mViewPager.addOnPageChangeListener(new ViewPager.OnPageChangeListener() {
```

```
@Override
```

```
public void onPageScrolled(int position, float positionOffset, int positionOffsetPixels)  
{..}
```

```
@Override
```

```
public void onPageSelected(int position) {..}
```

```
@Override
```

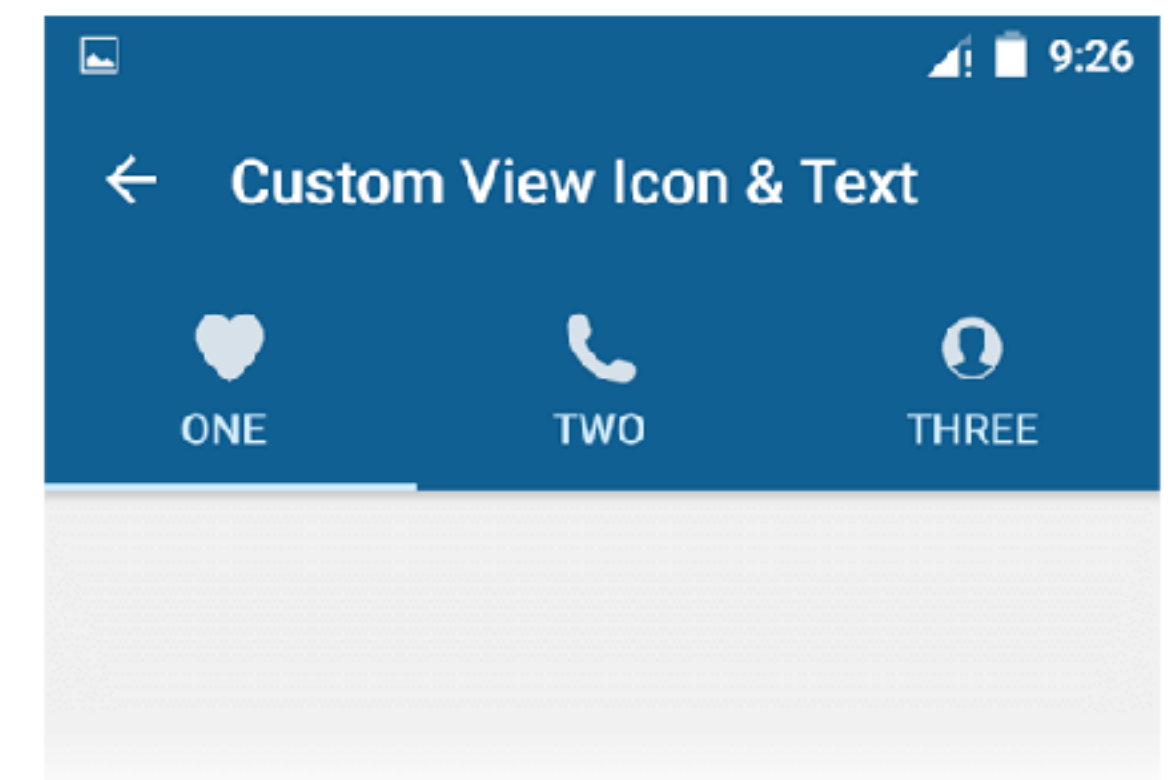
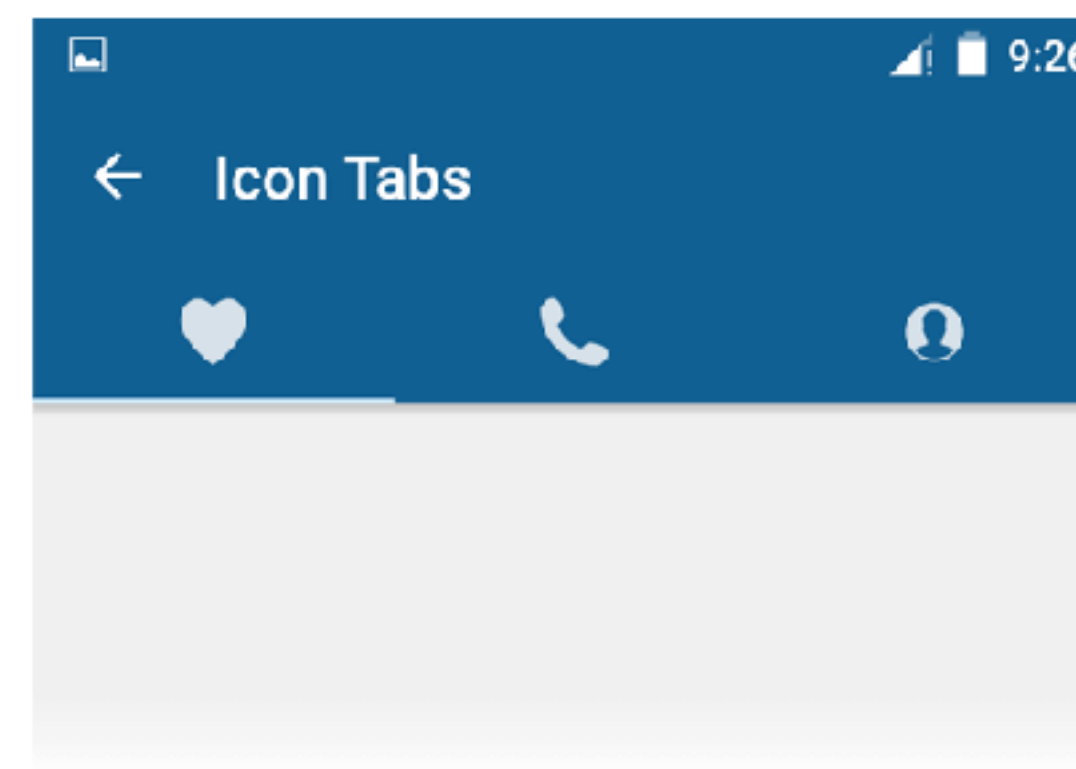
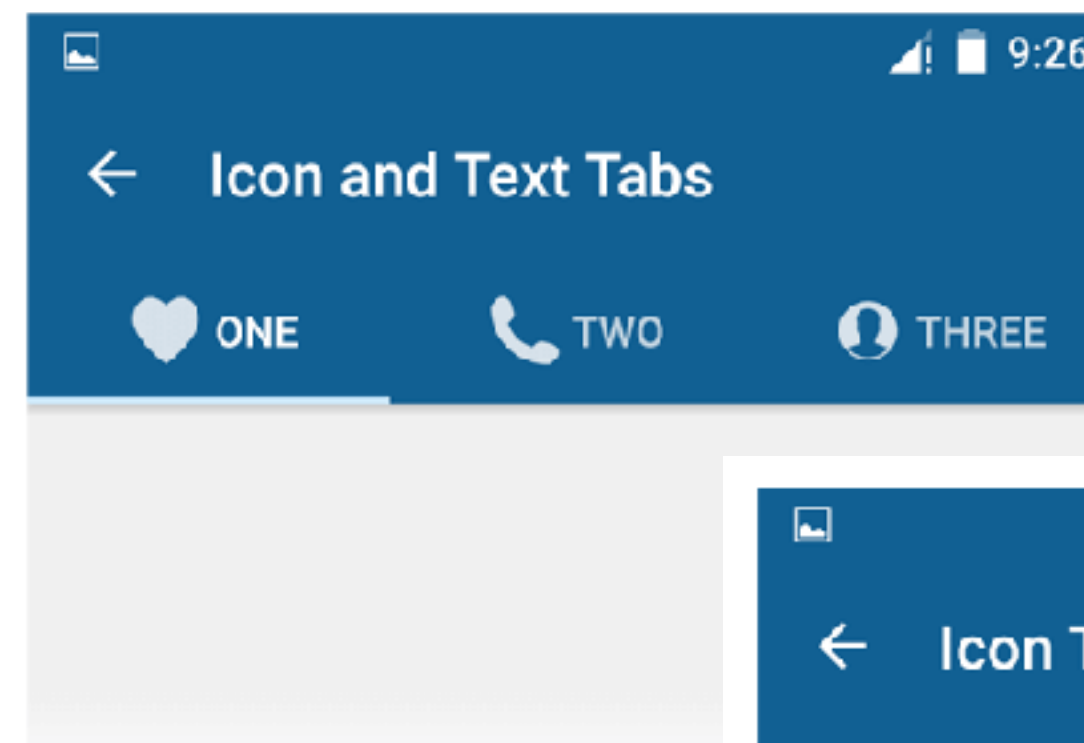
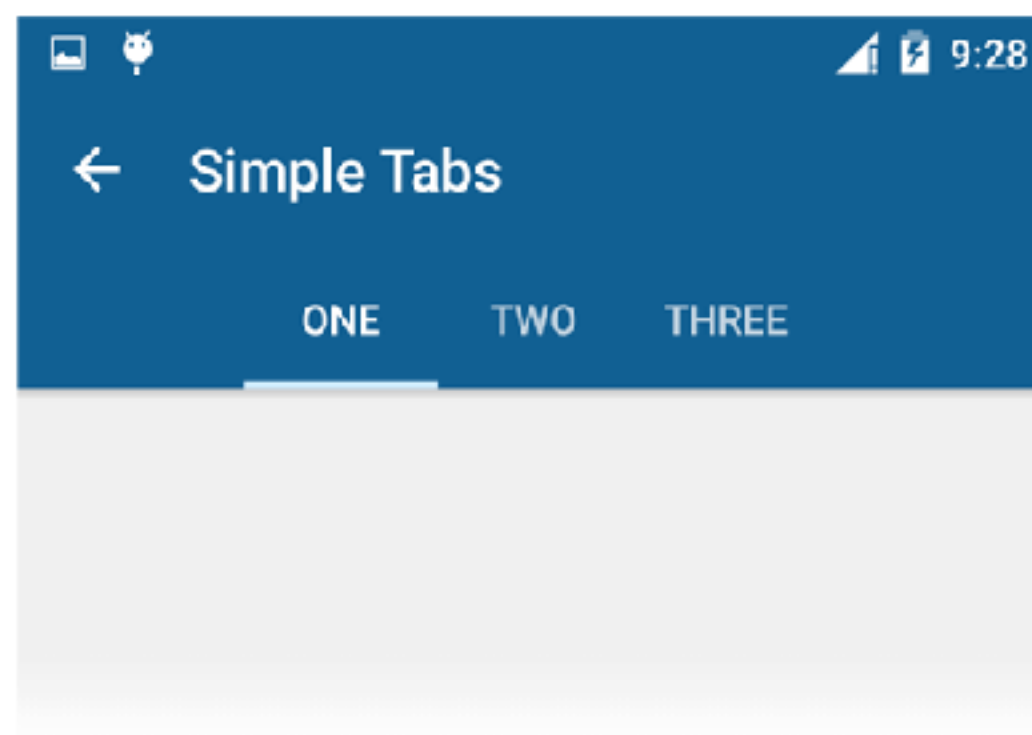
```
public void onPageScrollStateChanged(int state) {..}
```

```
});
```

Listener to react when a specific page has been selected by the user

Android & Tabs (TabLayout)

- TabLayout implements both fixed tabs as well as scrollable tabs, where the tabs are not a uniform size and can scroll horizontally.
- Tabs can be added programmatically.



TabLayout

```
<android.support.design.widget.TabLayout
    android:id="@+id/tabLayout"
    android:scrollbars="horizontal"
    android:layout_below="@+id/toolbar"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="?attr/colorPrimary" />
```



TabLayout XML Layout

```
TabLayout tabLayout = (TabLayout) findViewById(R.id.tabLayout);
tabLayout.setTabGravity(TabLayout.GRAVITY_FILL);
tabLayout.addTab(tabLayout.newTab().setText("Tab 1"));
tabLayout.addTab(tabLayout.newTab().setText("Tab 2"));
tabLayout.addTab(tabLayout.newTab().setText("Tab 3"));
tabLayout.addTab(tabLayout.newTab().setText("Tab 4"));
```



Retrieve the Object
reference and add 4
different Tabs reference

```
dependencies {
    compile 'com.android.support:support-v4:23.1.1'
    compile 'com.android.support:appcompat-v7:23.1.1'
    compile 'com.android.support:design:23.1.1'
}
```



Gradle Dependencies

TabLayout - Listener

```
tabLayout.setOnTabSelectedListener(new
TabLayout.OnTabSelectedListener() {
    @Override
    public void onTabSelected(TabLayout.Tab tab) {
        int position = tab.getPosition();
    }

    @Override
    public void onTabUnselected(TabLayout.Tab tab) {

    }

    @Override
    public void onTabReselected(TabLayout.Tab tab) {

    }
});
```



Handles different components callbacks.

For each method you can have information about the associated Tab and its position through the “Tab” parameter.

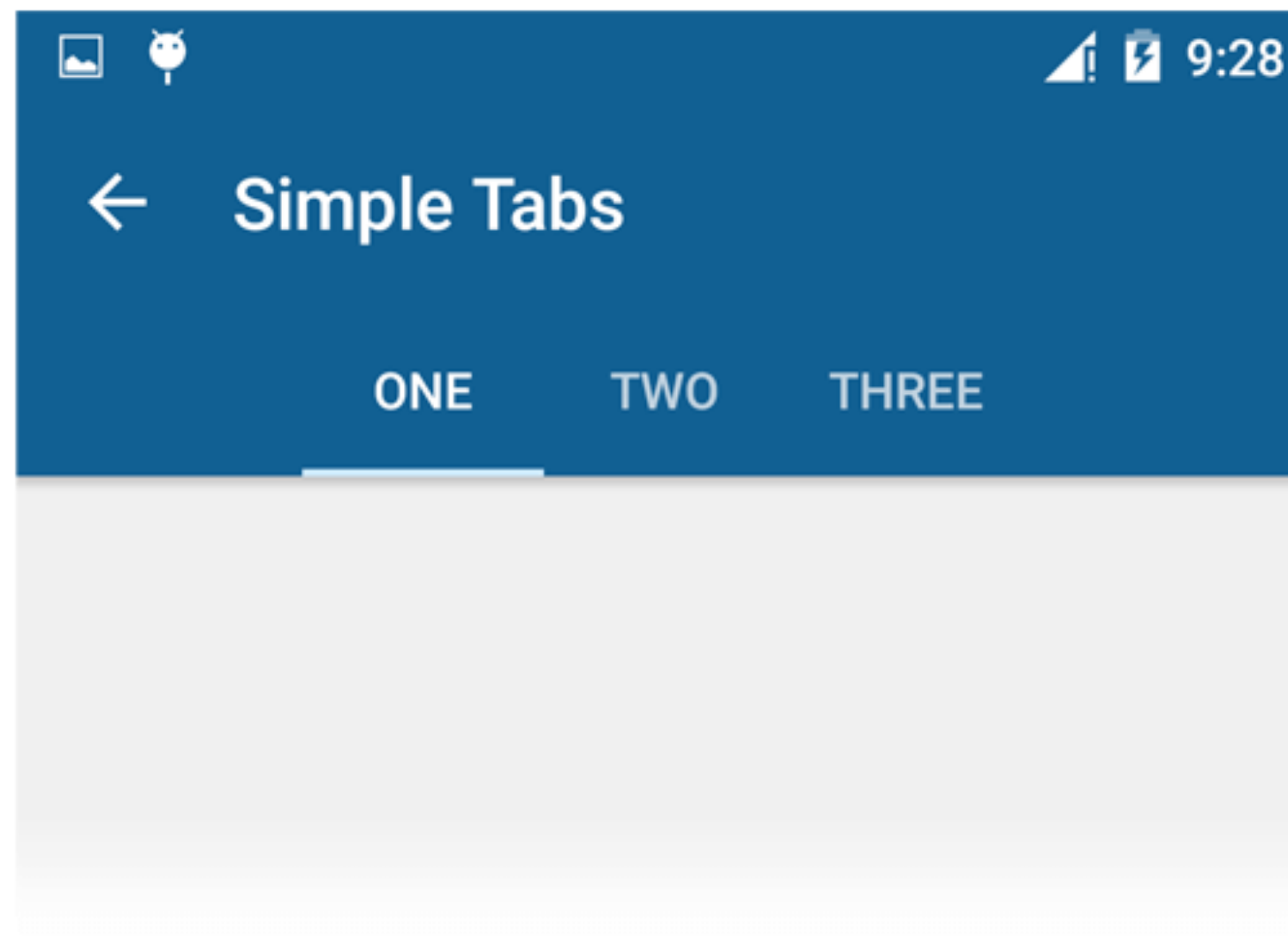
TabLayout & View Pager

//TabLayout with ViewPager

```
tabLayout = (TabLayout) findViewById(R.id.tabLayout);  
tabLayout.setTabGravity(TabLayout.GRAVITY_FILL);  
tabLayout.setupWithViewPager(mViewPager);
```

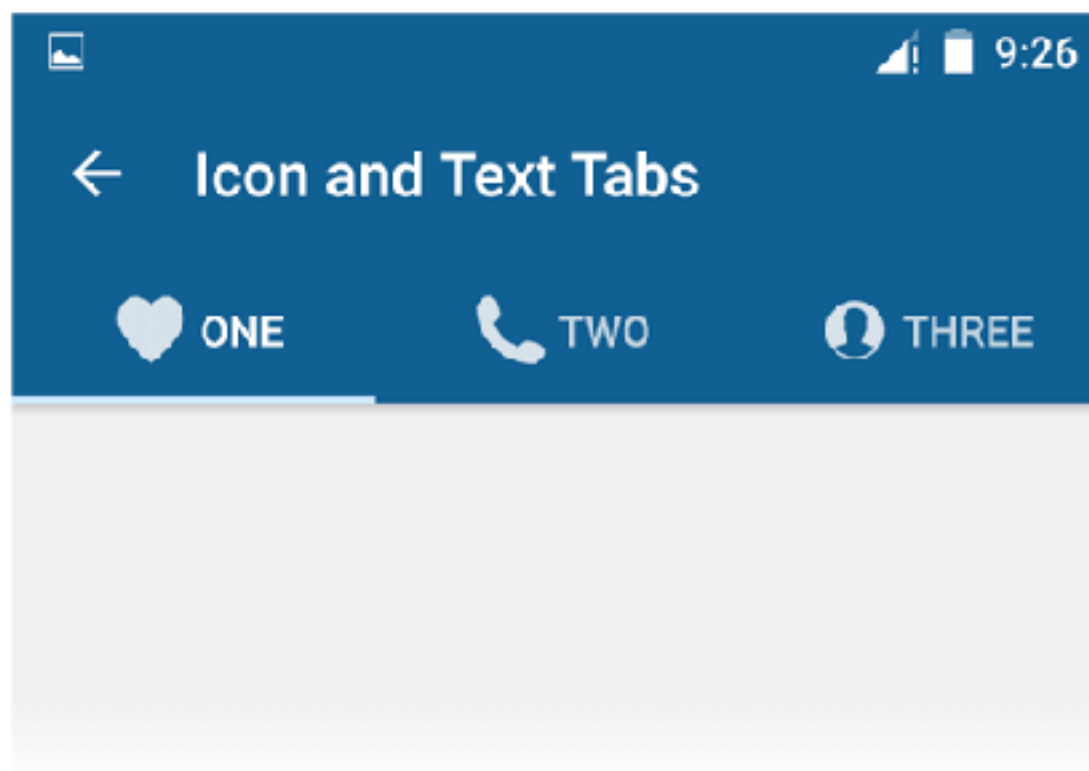


Create an association between ViewPager and TabLayout.



The library automatically retrieves Page names and count through the Pager Adapter and handles event both on the adapter and the tabs.

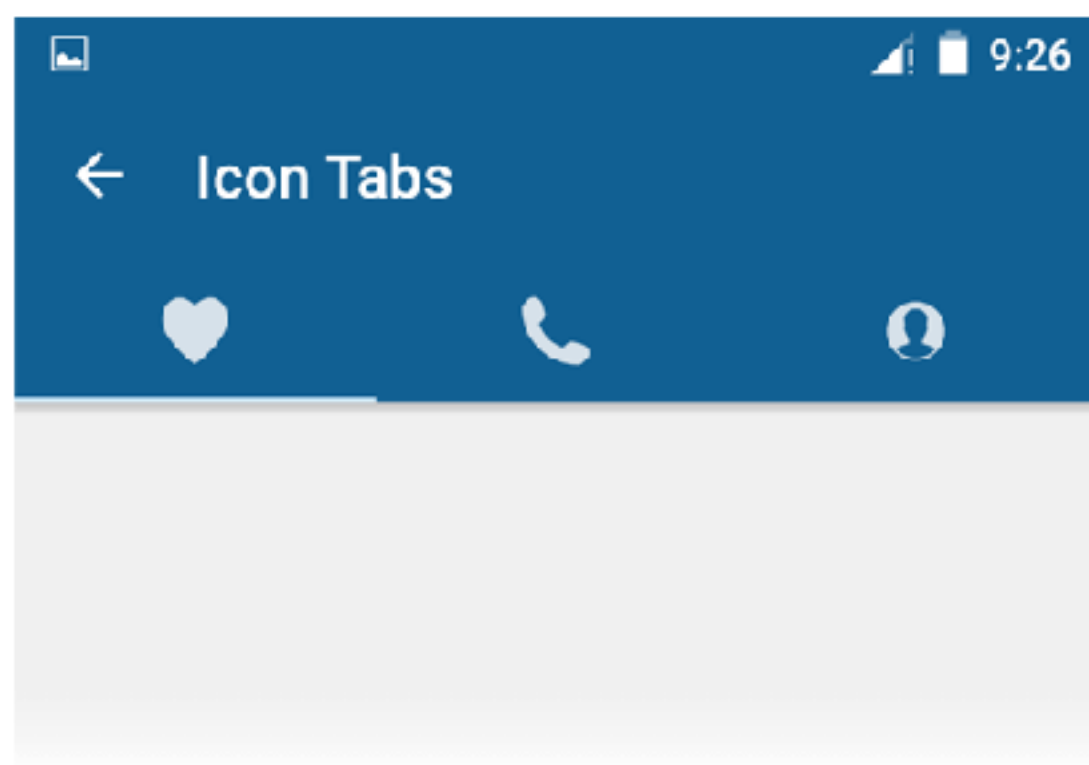
TabLayout Customization



Tab Layout Code

```
tabLayout.getTabAt(0).setIcon(tabIcons[0]);
```

➔ Icon & Text



Tab Layout Code

```
tabLayout.getTabAt(0).setIcon(tabIcons[0]);
```

In the ViewPagerAdapter

```
@Override  
public CharSequence getPageTitle(int position) {  
  
    // return null to display only the icon  
    return null;  
}
```

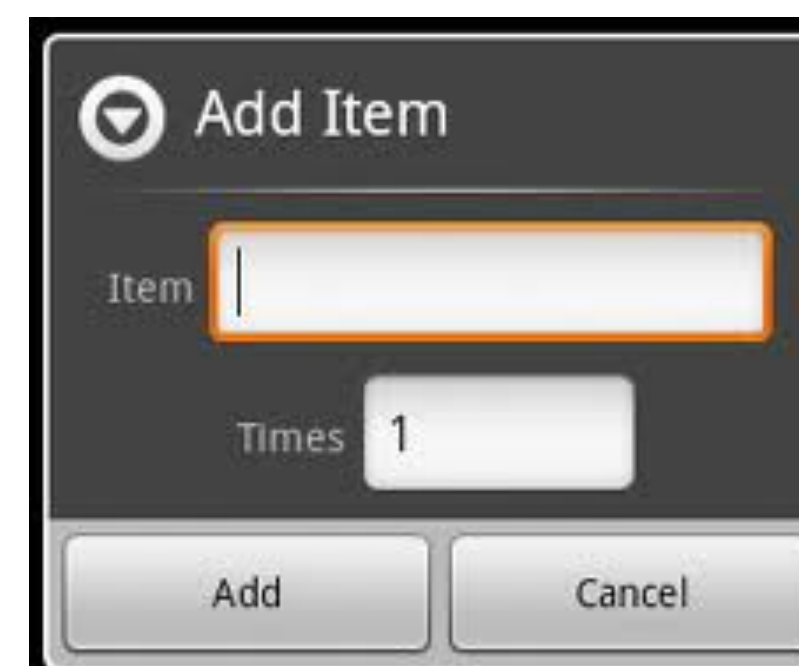
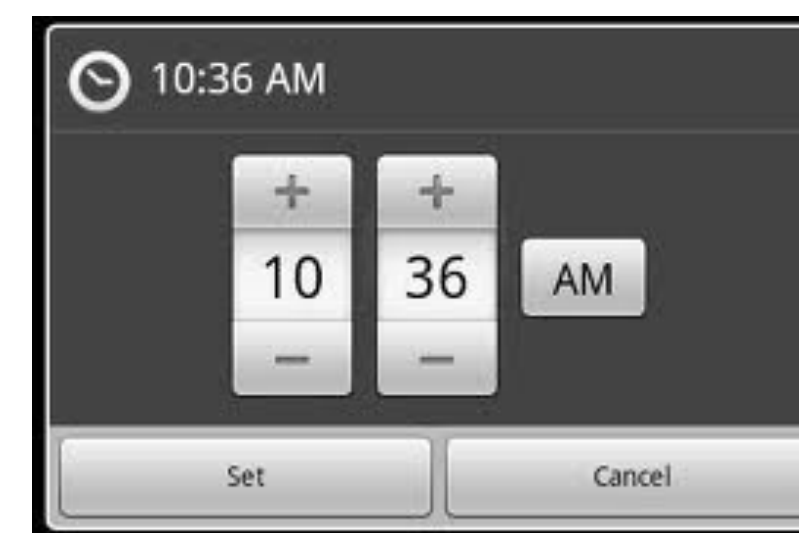
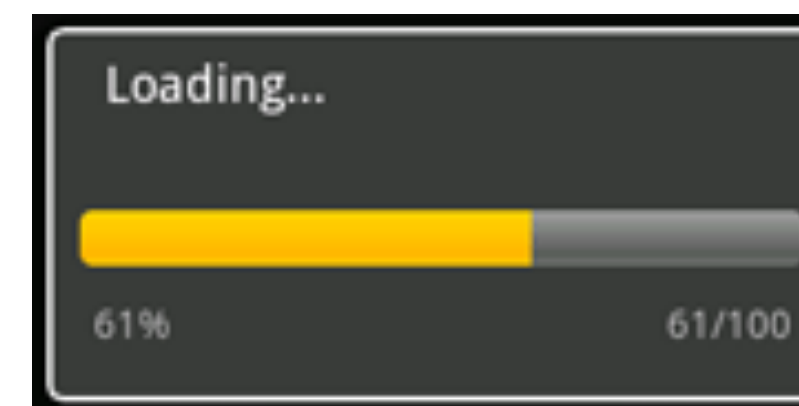
➔ Only Icon

Additional Customization at the following link

<http://www.androidhive.info/2015/09/android-material-design-working-with-tabs/>

Dialogs

- A dialog is a small window that appears in front of the current Activity.
- The underlying Activity loses focus and the dialog accepts all user interaction. Dialogs are used for notifications that should interrupt the user and to perform short tasks that directly relate to the application in progress (such as a progress bar or a login prompt).
- The Dialog class is the base class for creating dialogs. You can also use one of the following subclasses:
 - AlertDialog
 - ProgressDialog
 - DatePickerDialog
 - TimePickerDialog
- If you would like to customize your own dialog, you can extend the base Dialog object or any of the subclasses listed above and define a new layout.

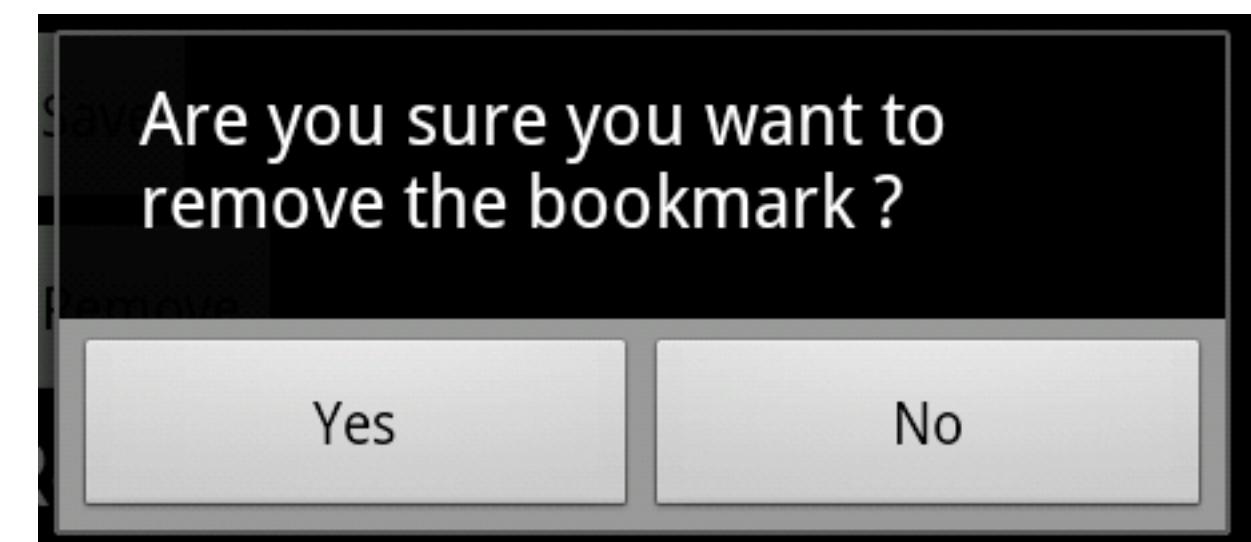


Alert Dialog

- A dialog that can manage zero, one, two, or three buttons, and/or a list of selectable items that can include checkboxes or radio buttons. The AlertDialog is capable of constructing most dialog user interfaces and is the suggested dialog type.

```
AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setMessage("Are you sure you want to exit?")
    .setCancelable(false)
    .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            MyActivity.this.finish();
        }
    })
    .setNegativeButton("No", new DialogInterface.OnClickListener() {
        public void onClick(DialogInterface dialog, int id) {
            dialog.cancel();
        }
    });
AlertDialog alert = builder.create();

alert.show();
```

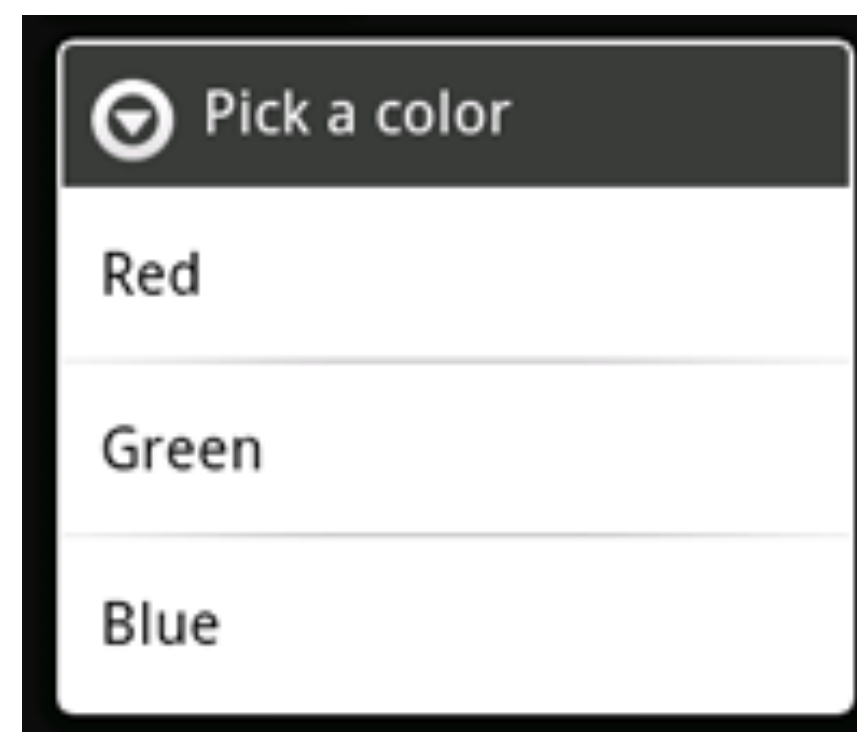


Alert Dialog

- It is also possible to create an AlertDialog with a list of selectable items using the method `setItems()`.

```
final CharSequence[] items = {"Red", "Green", "Blue"};

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick a color");
builder.setItems(items, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int item) {
        ...
    }
});
AlertDialog alert = builder.create();
```



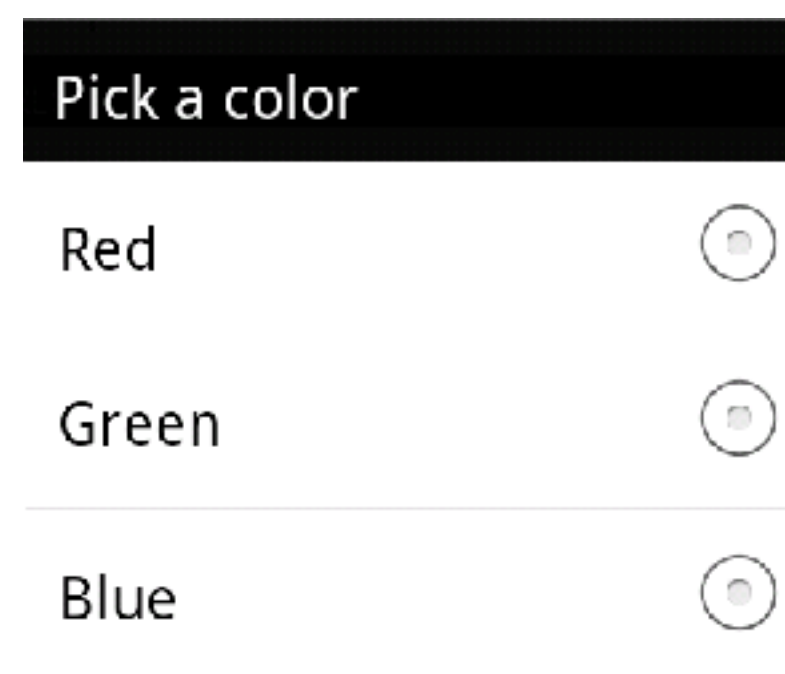
Alert Dialog

- Using the setMultiChoiceItems() and setSingleChoiceItems() methods, it is possible to create a list of multiple-choice items (checkboxes) or single-choice items (radio buttons) inside the dialog.

```
final CharSequence[] items = {"Red", "Green", "Blue"};

AlertDialog.Builder builder = new AlertDialog.Builder(this);
builder.setTitle("Pick a color");
builder.setSingleChoiceItems(items, -1, new DialogInterface.OnClickListener() {
    public void onClick(DialogInterface dialog, int item) {

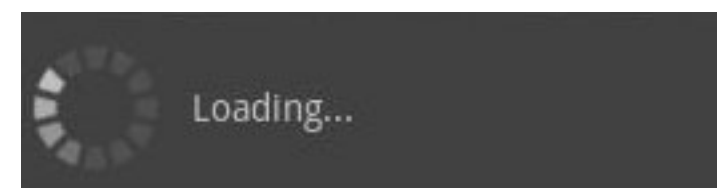
    }
});
AlertDialog alert = builder.create();
```



Progress Dialog

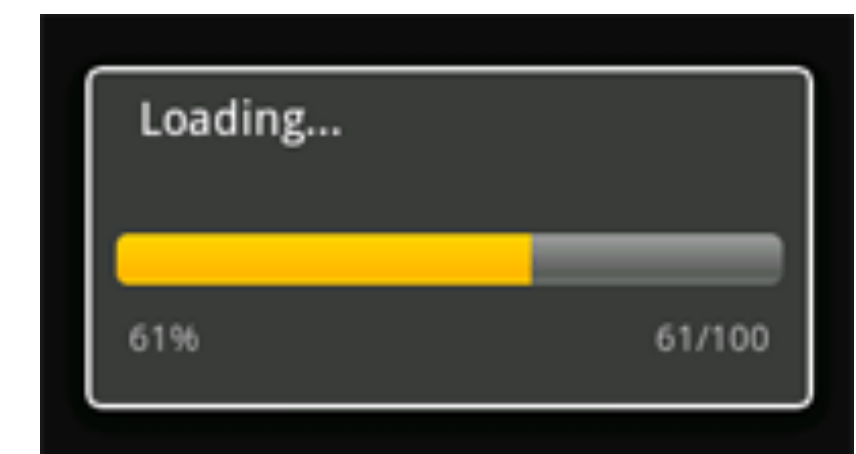
- A ProgressDialog is an extension of the AlertDialog class that can display a progress animation in the form of a spinning wheel, for a task with progress that's undefined, or a progress bar, for a task that has a defined progression.

```
ProgressDialog dialog = ProgressDialog.show(MyActivity.this, "", "Loading. Please wait...", true);
```



- If you want to create a progress bar that shows the loading progress with granularity you can :

```
ProgressDialog progressDialog;  
progressDialog = new ProgressDialog(mContext);  
progressDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);  
progressDialog.setMessage("Loading...");  
progressDialog.setCancelable(false);  
...  
progressDialog.show();
```



- You can increment the amount of progress displayed in the bar by calling either setProgress(int) with a value for the total percentage completed so far or incrementProgressBy(int) with an incremental value to add to the total percentage completed so far. You can also specify the maximum value in specific cases where the percentage view is not useful.

Custom Dialog

- It is also possible to customize the design of a dialog. You can create your own layout for the dialog window with layout and widget elements.
- When the Dialog has been instantiated you can set your custom layout as the dialog's content view with setContentView(int), passing it the layout resource ID.
- Using the method findViewById(int) on the dialog object it is possible to retrieve and modify its content.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/layout_root"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    >
    <ImageView android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_marginRight="10dp"
        />
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:textColor="#FFF"
        />
</LinearLayout>
```

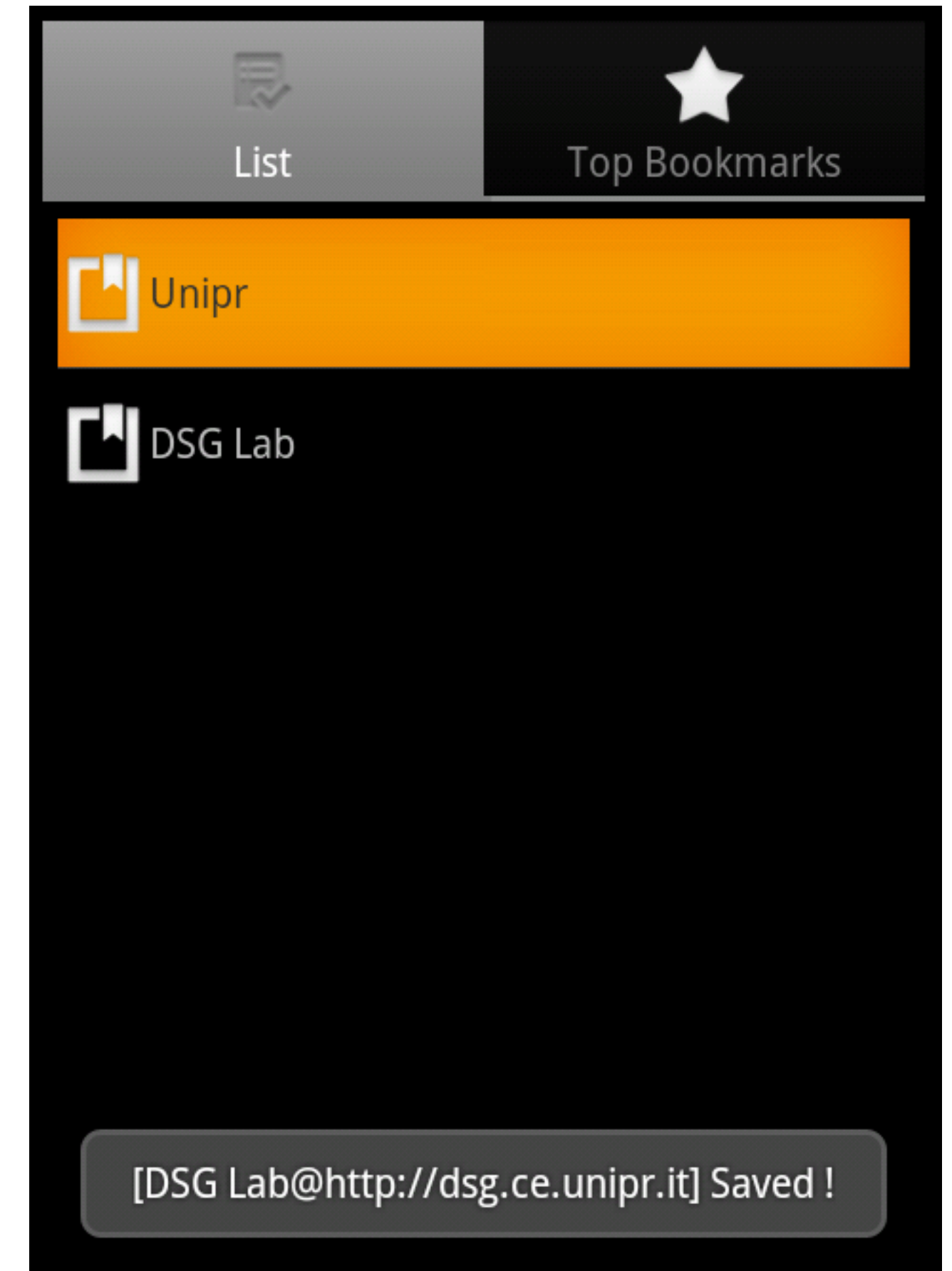
```
Context mContext = getApplicationContext();
Dialog dialog = new Dialog(mContext);

dialog.setContentView(R.layout.custom_dialog);
dialog.setTitle("Custom Dialog");

TextView text = (TextView)
dialog.findViewById(R.id.text);
text.setText("Hello, this is a custom dialog!");
ImageView image = (ImageView)
dialog.findViewById(R.id.image);
image.setImageResource(R.drawable.android);
```

Toast Notifications

- A toast notification is a message that pops up on the surface of the active view filling the amount of space required for the message.
- The current activity remains visible and interactive.
- The notification automatically fades in and out, and does not accept interaction events.
- A toast notification can be created and displayed from an Activity or Service. If you create a toast notification from a Service, it appears in front of the Activity currently in focus.



Toast Notifications

- First, instantiate a Toast object with one of the `makeText()` methods. This method takes three parameters: the application Context, the text message, and the duration for the toast.
- It returns a properly initialized Toast object and using the `show()` method you can display it.

```
Context context = getApplicationContext();  
CharSequence text = "Hello toast!";  
int duration = Toast.LENGTH_SHORT;  
  
Toast toast = Toast.makeText(context, text, duration);  
toast.show();
```

- The same result can be obtained in a short and easy way using a single line command

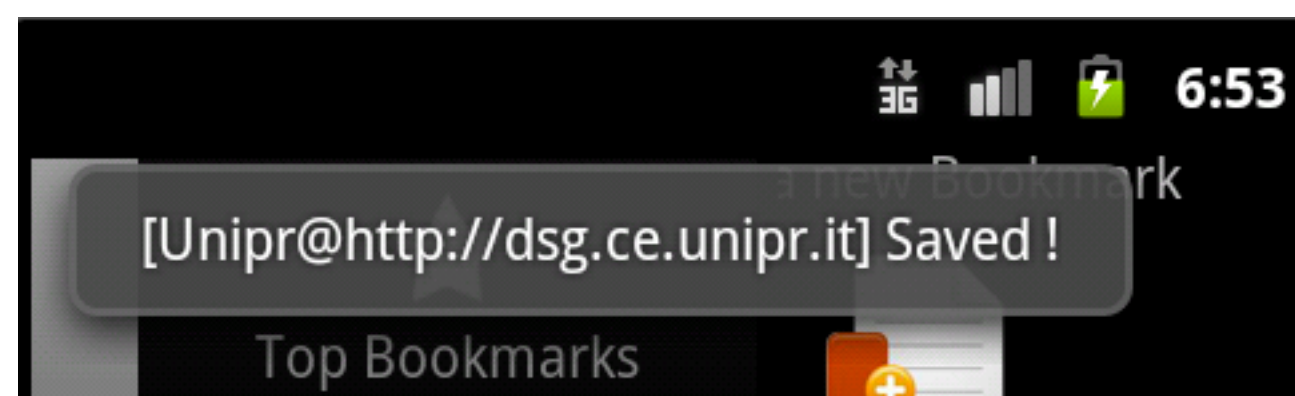
```
Toast.makeText(context, "Hello toast!", Toast.LENGTH_SHORT).show();
```

Toast Notifications - Position

- A standard toast notification appears near the bottom of the screen, centered horizontally. You can change this position with the `setGravity(int, int, int)` method. This accepts three parameters: a Gravity constant, an x-position offset, and a y-position offset.

```
toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0);
```

- For example, with the previous code you decide that the toast will appear in the top left corner.



Custom Toast Notification

- There is also the possibility to create a customized layout for your toast notification.
- As for previous UI element you can define custom layout by defining a View layout, in XML or in your application code, and pass the root View object to the `setView(View)` method.

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/
android"

    android:id="@+id/toast_layout_root"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:padding="10dp"
    android:background="#DAAA"
    >
    <ImageView android:id="@+id/image"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_marginRight="10dp"
        />
    <TextView android:id="@+id/text"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:textColor="#FFF"
        />
</LinearLayout>
```

- Notice that the ID of the `LinearLayout` element is `"toast_layout"`. You must use this ID to inflate the layout from the XML.

Custom Toast Notification

- To build a custom toast using a custom xml layout you can retrieve the LayoutInflater with `getLayoutInflater()` (or `getSystemService()`), and then inflate the layout from XML using `inflate(int, ViewGroup)`.
- The first parameter is the layout resource ID and the second is the root View.
- You can use this inflated layout to find more View objects in the layout, so now capture and define the content for the ImageView and TextView elements.
- Finally, create a new Toast with `Toast(Context)` and set some properties of the toast. Then call `setView(View)` and pass it the inflated layout. You can now display the toast with your custom layout by calling `show()`.

```
LayoutInflater inflater = getLayoutInflater();
View layout = inflater.inflate(R.layout.toast_layout, (ViewGroup) findViewById(R.id.toast_layout_root));

ImageView image = (ImageView) layout.findViewById(R.id.image);
image.setImageResource(R.drawable.android);
TextView text = (TextView) layout.findViewById(R.id.text);
text.setText("Hello! This is a custom toast!");

Toast toast = new Toast(getApplicationContext());
toast.setGravity(Gravity.CENTER_VERTICAL, 0, 0);
toast.setDuration(Toast.LENGTH_LONG);
toast.setView(layout);
toast.show();
```



Android Development

Lecture 4

Android Graphical User Interface 3



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