**Элементы пользовательского интерфейса**

Поле ввода, чекбокс, радио кнопки, спиннер, переключатель, всплывающие подсказки, всплывающие сообщения (pop-up message)

# User Interface & Navigation

Your app's user interface is everything that the user can see and interact with. Android provides a variety of pre-built UI components such as structured layout objects and UI controls that allow you to build the graphical user interface for your app. Android also provides other UI modules for special interfaces such as dialogs, notifications, and menus.

# EditText

**Kotlin** |[Java](https://developer.android.com/reference/android/widget/EditText)

open class EditText : [TextView](https://developer.android.com/reference/kotlin/android/widget/TextView)

|  |  |  |  |
| --- | --- | --- | --- |
| [kotlin.Any](https://kotlinlang.org/api/latest/jvm/stdlib/kotlin/-any/index.html) | | | |
| ↳ | [android.view.View](https://developer.android.com/reference/kotlin/android/view/View.html) | | |
|  | ↳ | [android.widget.TextView](https://developer.android.com/reference/kotlin/android/widget/TextView) | |
|  |  | ↳ | [android.widget.EditText](https://developer.android.com/reference/kotlin/android/widget/EditText) |
| Known Direct Subclasses  [AutoCompleteTextView](https://developer.android.com/reference/kotlin/android/widget/AutoCompleteTextView), [ExtractEditText](https://developer.android.com/reference/kotlin/android/inputmethodservice/ExtractEditText.html) | | | |

|  |
| --- |
| Known Indirect Subclasses  [MultiAutoCompleteTextView](https://developer.android.com/reference/kotlin/android/widget/MultiAutoCompleteTextView) |

A user interface element for entering and modifying text. When you define an edit text widget, you must specify the [android.R.styleable#TextView\_inputType](https://developer.android.com/reference/kotlin/android/R.styleable.html" \l "TextView_inputType:kotlin.Int) attribute. For example, for plain text input set inputType to "text":

<EditText  
      android:id="@+id/plain\_text\_input"  
      android:layout\_height="wrap\_content"  
      android:layout\_width="match\_parent"  
      android:inputType="text"/>

Choosing the input type configures the keyboard type that is shown, acceptable characters, and appearance of the edit text. For example, if you want to accept a secret number, like a unique pin or serial number, you can set inputType to "numericPassword". An inputType of "numericPassword" results in an edit text that accepts numbers only, shows a numeric keyboard when focused, and masks the text that is entered for privacy.

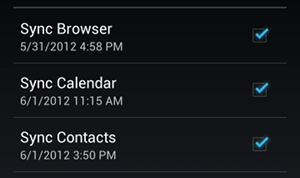
See the [Text Fields](https://developer.android.com/guide/topics/ui/controls/text.html) guide for examples of other [android.R.styleable#TextView\_inputType](https://developer.android.com/reference/kotlin/android/R.styleable.html" \l "TextView_inputType:kotlin.Int) settings.

You also can receive callbacks as a user changes text by adding a [android.text.TextWatcher](https://developer.android.com/reference/kotlin/android/text/TextWatcher.html) to the edit text. This is useful when you want to add auto-save functionality as changes are made, or validate the format of user input, for example. You add a text watcher using the [TextView#addTextChangedListener](https://developer.android.com/reference/kotlin/android/widget/TextView" \l "addTextChangedListener(android.text.TextWatcher)) method.

This widget does not support auto-sizing text.

Checkboxes

Checkboxes allow the user to select one or more options from a set. Typically, you should present each checkbox option in a vertical list.



To create each checkbox option, create a [CheckBox](https://developer.android.com/reference/android/widget/CheckBox) in your layout. Because a set of checkbox options allows the user to select multiple items, each checkbox is managed separately and you must register a click listener for each one.

A key class is the following:

* [CheckBox](https://developer.android.com/reference/android/widget/CheckBox)

Responding to Click Events

When the user selects a checkbox, the [CheckBox](https://developer.android.com/reference/android/widget/CheckBox) object receives an on-click event.

To define the click event handler for a checkbox, add the [android:onClick](https://developer.android.com/reference/android/R.attr" \l "onClick) attribute to the <CheckBox> element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The [Activity](https://developer.android.com/reference/android/app/Activity) hosting the layout must then implement the corresponding method.

For example, here are a couple [CheckBox](https://developer.android.com/reference/android/widget/CheckBox) objects in a list:

<?xml version="1.0" encoding="utf-8"?>  
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:orientation="vertical"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="fill\_parent">  
    <CheckBox android:id="@+id/checkbox\_meat"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="@string/meat"  
        android:onClick="onCheckboxClicked"/>  
    <CheckBox android:id="@+id/checkbox\_cheese"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="@string/cheese"  
        android:onClick="onCheckboxClicked"/>  
</LinearLayout>

Within the [Activity](https://developer.android.com/reference/android/app/Activity) that hosts this layout, the following method handles the click event for both checkboxes:

[KOTLIN](https://developer.android.com/guide/topics/ui/controls/checkbox#kotlin)[JAVA](https://developer.android.com/guide/topics/ui/controls/checkbox#java)

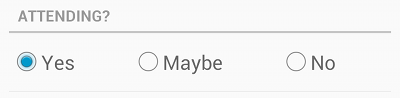
fun onCheckboxClicked(view: View) {  
    if (view is CheckBox) {  
        val checked: Boolean = view.isChecked  
  
        when (view.id) {  
            R.id.checkbox\_meat -> {  
                if (checked) {  
                    // Put some meat on the sandwich  
                } else {  
                    // Remove the meat  
                }  
            }  
            R.id.checkbox\_cheese -> {  
                if (checked) {  
                    // Cheese me  
                } else {  
                    // I'm lactose intolerant  
                }  
            }  
            // TODO: Veggie sandwich  
        }  
    }  
}

The method you declare in the [android:onClick](https://developer.android.com/reference/android/R.attr" \l "onClick) attribute must have a signature exactly as shown above. Specifically, the method must:

* Be public
* Return void
* Define a [View](https://developer.android.com/reference/android/view/View) as its only parameter (this will be the [View](https://developer.android.com/reference/android/view/View) that was clicked)

Radio Buttons

Radio buttons allow the user to select one option from a set. You should use radio buttons for optional sets that are mutually exclusive if you think that the user needs to see all available options side-by-side. If it's not necessary to show all options side-by-side, use a [spinner](https://developer.android.com/guide/topics/ui/controls/spinner) instead.



To create each radio button option, create a [RadioButton](https://developer.android.com/reference/android/widget/RadioButton) in your layout. However, because radio buttons are mutually exclusive, you must group them together inside a [RadioGroup](https://developer.android.com/reference/android/widget/RadioGroup). By grouping them together, the system ensures that only one radio button can be selected at a time.

Key classes are the following:

* [RadioButton](https://developer.android.com/reference/android/widget/RadioButton)
* [RadioGroup](https://developer.android.com/reference/android/widget/RadioGroup)

Responding to Click Events

When the user selects one of the radio buttons, the corresponding [RadioButton](https://developer.android.com/reference/android/widget/RadioButton) object receives an on-click event.

To define the click event handler for a button, add the [android:onClick](https://developer.android.com/reference/android/R.attr" \l "onClick) attribute to the <RadioButton> element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The [Activity](https://developer.android.com/reference/android/app/Activity) hosting the layout must then implement the corresponding method.

For example, here are a couple [RadioButton](https://developer.android.com/reference/android/widget/RadioButton) objects:

<?xml version="1.0" encoding="utf-8"?>  
<RadioGroup xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout\_width="match\_parent"  
    android:layout\_height="wrap\_content"  
    android:orientation="vertical">  
    <RadioButton android:id="@+id/radio\_pirates"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="@string/pirates"  
        android:onClick="onRadioButtonClicked"/>  
    <RadioButton android:id="@+id/radio\_ninjas"  
        android:layout\_width="wrap\_content"  
        android:layout\_height="wrap\_content"  
        android:text="@string/ninjas"  
        android:onClick="onRadioButtonClicked"/>  
</RadioGroup>

**Note:** The **[RadioGroup](https://developer.android.com/reference/android/widget/RadioGroup)** is a subclass of **[LinearLayout](https://developer.android.com/reference/android/widget/LinearLayout)** that has a vertical orientation by default.

Within the [Activity](https://developer.android.com/reference/android/app/Activity) that hosts this layout, the following method handles the click event for both radio buttons:

[KOTLIN](https://developer.android.com/guide/topics/ui/controls/radiobutton#kotlin)[JAVA](https://developer.android.com/guide/topics/ui/controls/radiobutton#java)

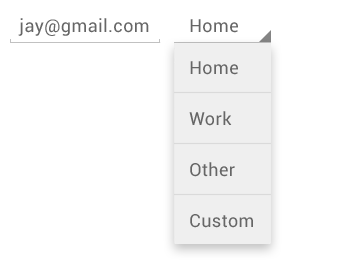
fun onRadioButtonClicked(view: View) {  
    if (view is RadioButton) {  
        // Is the button now checked?  
        val checked = view.isChecked  
  
        // Check which radio button was clicked  
        when (view.getId()) {  
            R.id.radio\_pirates ->  
                if (checked) {  
                    // Pirates are the best  
                }  
            R.id.radio\_ninjas ->  
                if (checked) {  
                    // Ninjas rule  
                }  
        }  
    }  
}

The method you declare in the [android:onClick](https://developer.android.com/reference/android/R.attr" \l "onClick) attribute must have a signature exactly as shown above. Specifically, the method must:

* Be public
* Return void
* Define a [View](https://developer.android.com/reference/android/view/View) as its only parameter (this will be the [View](https://developer.android.com/reference/android/view/View) that was clicked)

Spinners

Spinners provide a quick way to select one value from a set. In the default state, a spinner shows its currently selected value. Touching the spinner displays a dropdown menu with all other available values, from which the user can select a new one.



You can add a spinner to your layout with the [Spinner](https://developer.android.com/reference/android/widget/Spinner) object. You should usually do so in your XML layout with a <Spinner> element. For example:

<Spinner  
    android:id="@+id/planets\_spinner"  
    android:layout\_width="fill\_parent"  
    android:layout\_height="wrap\_content" />

To populate the spinner with a list of choices, you then need to specify a [SpinnerAdapter](https://developer.android.com/reference/android/widget/SpinnerAdapter) in your [Activity](https://developer.android.com/reference/android/app/Activity) or [Fragment](https://developer.android.com/reference/android/app/Fragment) source code.

Key classes are the following:

* [Spinner](https://developer.android.com/reference/android/widget/Spinner)
* [SpinnerAdapter](https://developer.android.com/reference/android/widget/SpinnerAdapter)
* [AdapterView.OnItemSelectedListener](https://developer.android.com/reference/android/widget/AdapterView.OnItemSelectedListener)

Populate the Spinner with User Choices

The choices you provide for the spinner can come from any source, but must be provided through an [SpinnerAdapter](https://developer.android.com/reference/android/widget/SpinnerAdapter), such as an [ArrayAdapter](https://developer.android.com/reference/android/widget/ArrayAdapter) if the choices are available in an array or a [CursorAdapter](https://developer.android.com/reference/android/widget/CursorAdapter) if the choices are available from a database query.

For instance, if the available choices for your spinner are pre-determined, you can provide them with a string array defined in a [string resource file](https://developer.android.com/guide/topics/resources/string-resource):

<?xml version="1.0" encoding="utf-8"?>  
<resources>  
    <string-array name="planets\_array">  
        <item>Mercury</item>  
        <item>Venus</item>  
        <item>Earth</item>  
        <item>Mars</item>  
        <item>Jupiter</item>  
        <item>Saturn</item>  
        <item>Uranus</item>  
        <item>Neptune</item>  
    </string-array>  
</resources>

With an array such as this one, you can use the following code in your [Activity](https://developer.android.com/reference/android/app/Activity) or [Fragment](https://developer.android.com/reference/android/app/Fragment) to supply the spinner with the array using an instance of [ArrayAdapter](https://developer.android.com/reference/android/widget/ArrayAdapter):

[KOTLIN](https://developer.android.com/guide/topics/ui/controls/spinner#kotlin)[JAVA](https://developer.android.com/guide/topics/ui/controls/spinner#java)

val spinner: Spinner = findViewById(R.id.spinner)  
// Create an ArrayAdapter using the string array and a default spinner layout  
ArrayAdapter.createFromResource(  
        this,  
        R.array.planets\_array,  
        android.R.layout.simple\_spinner\_item  
).also { adapter ->  
    // Specify the layout to use when the list of choices appears  
    adapter.setDropDownViewResource(android.R.layout.simple\_spinner\_dropdown\_item)  
    // Apply the adapter to the spinner  
    spinner.adapter = adapter  
}

The [createFromResource()](https://developer.android.com/reference/android/widget/ArrayAdapter" \l "createFromResource(android.content.Context, int, int)) method allows you to create an [ArrayAdapter](https://developer.android.com/reference/android/widget/ArrayAdapter) from the string array. The third argument for this method is a layout resource that defines how the selected choice appears in the spinner control. The [simple\_spinner\_item](https://developer.android.com/reference/android/R.layout" \l "simple_spinner_item) layout is provided by the platform and is the default layout you should use unless you'd like to define your own layout for the spinner's appearance.

You should then call [setDropDownViewResource(int)](https://developer.android.com/reference/android/widget/ArrayAdapter" \l "setDropDownViewResource(int)) to specify the layout the adapter should use to display the list of spinner choices ([simple\_spinner\_dropdown\_item](https://developer.android.com/reference/android/R.layout" \l "simple_spinner_dropdown_item) is another standard layout defined by the platform).

Call [setAdapter()](https://developer.android.com/reference/android/widget/AdapterView" \l "setAdapter(T)) to apply the adapter to your [Spinner](https://developer.android.com/reference/android/widget/Spinner).

Responding to User Selections

When the user selects an item from the drop-down, the [Spinner](https://developer.android.com/reference/android/widget/Spinner) object receives an on-item-selected event.

To define the selection event handler for a spinner, implement the [AdapterView.OnItemSelectedListener](https://developer.android.com/reference/android/widget/AdapterView.OnItemSelectedListener)interface and the corresponding [onItemSelected()](https://developer.android.com/reference/android/widget/AdapterView.OnItemSelectedListener" \l "onItemSelected(android.widget.AdapterView<?>, android.view.View, int, long)) callback method. For example, here's an implementation of the interface in an [Activity](https://developer.android.com/reference/android/app/Activity):

[KOTLIN](https://developer.android.com/guide/topics/ui/controls/spinner#kotlin)[JAVA](https://developer.android.com/guide/topics/ui/controls/spinner#java)

class SpinnerActivity : Activity(), AdapterView.OnItemSelectedListener {  
  
    override fun onItemSelected(parent: AdapterView<\*>, view: View, pos: Int, id: Long) {  
        // An item was selected. You can retrieve the selected item using  
        // parent.getItemAtPosition(pos)  
    }  
  
    override fun onNothingSelected(parent: AdapterView<\*>) {  
        // Another interface callback  
    }  
}

The [AdapterView.OnItemSelectedListener](https://developer.android.com/reference/android/widget/AdapterView.OnItemSelectedListener) requires the [onItemSelected()](https://developer.android.com/reference/android/widget/AdapterView.OnItemSelectedListener" \l "onItemSelected(android.widget.AdapterView<?>, android.view.View, int, long)) and [onNothingSelected()](https://developer.android.com/reference/android/widget/AdapterView.OnItemSelectedListener" \l "onNothingSelected(android.widget.AdapterView<?>))callback methods.

Then you need to specify the interface implementation by calling [setOnItemSelectedListener()](https://developer.android.com/reference/android/widget/AdapterView" \l "setOnItemSelectedListener(android.widget.AdapterView.OnItemSelectedListener)):

[KOTLIN](https://developer.android.com/guide/topics/ui/controls/spinner#kotlin)[JAVA](https://developer.android.com/guide/topics/ui/controls/spinner#java)

val spinner: Spinner = findViewById(R.id.spinner)  
spinner.onItemSelectedListener = this

If you implement the [AdapterView.OnItemSelectedListener](https://developer.android.com/reference/android/widget/AdapterView.OnItemSelectedListener) interface with your [Activity](https://developer.android.com/reference/android/app/Activity) or [Fragment](https://developer.android.com/reference/android/app/Fragment) (such as in the example above), you can pass this as the interface instance.

Toggle Buttons

A toggle button allows the user to change a setting between two states.

You can add a basic toggle button to your layout with the [ToggleButton](https://developer.android.com/reference/android/widget/ToggleButton) object. Android 4.0 (API level 14) introduces another kind of toggle button called a switch that provides a slider control, which you can add with a [Switch](https://developer.android.com/reference/android/widget/Switch) object. [SwitchCompat](https://developer.android.com/reference/androidx/appcompat/widget/SwitchCompat) is a version of the Switch widget which runs on devices back to API 7.

If you need to change a button's state yourself, you can use the [CompoundButton.setChecked()](https://developer.android.com/reference/android/widget/CompoundButton#setChecked(boolean)) or[CompoundButton.toggle()](https://developer.android.com/reference/android/widget/CompoundButton#toggle()) method.

https://developer.android.com/images/ui/togglebutton.png

*Toggle buttons*

https://developer.android.com/images/ui/switch.png

*Switches (in Android 4.0+)*

Key classes are the following:

* [ToggleButton](https://developer.android.com/reference/android/widget/ToggleButton)
* [Switch](https://developer.android.com/reference/android/widget/Switch)
* [SwitchCompat](https://developer.android.com/reference/androidx/appcompat/widget/SwitchCompat)
* [CompoundButton](https://developer.android.com/reference/android/widget/CompoundButton)

Responding to Button Presses

To detect when the user activates the button or switch, create an [CompoundButton.OnCheckedChangeListener](https://developer.android.com/reference/android/widget/CompoundButton.OnCheckedChangeListener)object and assign it to the button by calling [setOnCheckedChangeListener()](https://developer.android.com/reference/android/widget/CompoundButton" \l "setOnCheckedChangeListener(android.widget.CompoundButton.OnCheckedChangeListener)). For example:

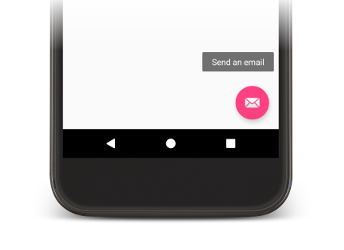
[KOTLIN](https://developer.android.com/guide/topics/ui/controls/togglebutton#kotlin)[JAVA](https://developer.android.com/guide/topics/ui/controls/togglebutton#java)

val toggle: ToggleButton = findViewById(R.id.togglebutton)  
toggle.setOnCheckedChangeListener { \_, isChecked ->  
    if (isChecked) {  
        // The toggle is enabled  
    } else {  
        // The toggle is disabled  
    }  
}

# Tooltips

A tooltip is a small descriptive message that appears near a view when users long press the view or hover their mouse over it. This is useful when your app uses an icon to represent an action or piece of information to save space in the layout. This page shows you how to add these tooltips on Android 8.0 (API level 26) and higher.

Some scenarios, such as those in productivity apps, require a descriptive method of communicating ideas and actions. You can use tooltips to display a descriptive message, as shown in figure 1.



**Figure 1.** Tooltip displayed in an Android app.

Some standard widgets display tooltips based on the content of the title or content description properties. Starting in Android 8.0, you can specify the text displayed in the tooltip regardless of the value of other properties.

## Setting the tooltip text

You can specify the tooltip text in a [View](https://developer.android.com/reference/android/view/View) by calling the [setTooltipText()](https://developer.android.com/reference/android/view/View" \l "setTooltipText(java.lang.CharSequence)) method. You can set the tooltipText property using the corresponding XML attribute or API.

To specify the tooltip text in your XML files, set the [android:tooltipText](https://developer.android.com/reference/android/R.styleable" \l "View_tooltipText) attribute, as shown in the following example:

<android.support.design.widget.FloatingActionButton  
        android:id="@+id/fab"  
        android:tooltipText="Send an email" />

To specify the tooltip text in your code, use the [setTooltipText(CharSequence)](https://developer.android.com/reference/android/view/View" \l "setTooltipText(java.lang.CharSequence)) method, as shown in the following example:

[KOTLIN](https://developer.android.com/guide/topics/ui/tooltips#kotlin)[JAVA](https://developer.android.com/guide/topics/ui/tooltips#java)

val fab: FloatingActionButton = findViewById(R.id.fab)  
fab.tooltipText = "Send an email"

The API also includes a [getTooltipText()](https://developer.android.com/reference/android/view/View" \l "getTooltipText()) method that you can use to retrieve the value of the tooltipTextproperty.

Android displays the value of the tooltipText property when users hover their mouse over the view or long press the view.

Android code snippets for android app developer.

I started this android app development tutorials blog in 2015. At the begging, I wrote tutorials for beginners about how to develop an android app. Most of my tutorials are very clear to read and easy to understand and simple to implement any existing project. Last 5 years, I covered many android app development topics such as layouts, widgets, bitmap, drawable, button, text view, alert dialog, card view, recycler view, API, date time picker, spinner, list view, grid view, theme, style, design, data binding, room database, volley network library, kotlin syntax, java syntax, notification, navigation drawer, wifi, Bluetooth, system permissions, storage, media, seek bar, progress bar, web view, animation, transition, toolbar, google map, screen, jetpack, work manager, android ktx, fragment, audio manager, activity, action bar and many more.

## Monday, February 12, 2018

### **android kotlin - Popup window example**

MainActivity.kt

package com.cfsuman.kotlinexamples

**import** android.support.v7.app.AppCompatActivity

**import** android.os.Bundle

**import** android.view.LayoutInflater

**import** kotlinx.android.synthetic.main.activity\_main.\*

**import** android.content.Context

**import** android.graphics.Color

**import** android.os.Build

**import** android.transition.Slide

**import** android.transition.TransitionManager

**import** android.view.Gravity

**import** android.widget.\*

**class** MainActivity : AppCompatActivity() {

override fun **onCreate**(savedInstanceState: Bundle?) {

super.onCreate(savedInstanceState)

setContentView(R.layout.activity\_main)

*// Set a click listener for button widget*

button.setOnClickListener{

*// Initialize a new layout inflater instance*

val inflater:LayoutInflater = getSystemService(Context.LAYOUT\_INFLATER\_SERVICE) as LayoutInflater

*// Inflate a custom view using layout inflater*

val view = inflater.inflate(R.layout.another\_view,null)

*// Initialize a new instance of popup window*

val popupWindow = PopupWindow(

view, *// Custom view to show in popup window*

LinearLayout.LayoutParams.WRAP\_CONTENT, *// Width of popup window*

LinearLayout.LayoutParams.WRAP\_CONTENT *// Window height*

)

*// Set an elevation for the popup window*

**if** (Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.LOLLIPOP) {

popupWindow.elevation = 10.0F

}

*// If API level 23 or higher then execute the code*

**if**(Build.VERSION.SDK\_INT >= Build.VERSION\_CODES.M){

*// Create a new slide animation for popup window enter transition*

val slideIn = Slide()

slideIn.slideEdge = Gravity.TOP

popupWindow.enterTransition = slideIn

*// Slide animation for popup window exit transition*

val slideOut = Slide()

slideOut.slideEdge = Gravity.RIGHT

popupWindow.exitTransition = slideOut

}

*// Get the widgets reference from custom view*

val tv = view.findViewById<TextView>(R.id.text\_view)

val buttonPopup = view.findViewById<Button>(R.id.button\_popup)

*// Set click listener for popup window's text view*

tv.setOnClickListener{

*// Change the text color of popup window's text view*

tv.setTextColor(Color.RED)

}

*// Set a click listener for popup's button widget*

buttonPopup.setOnClickListener{

*// Dismiss the popup window*

popupWindow.dismiss()

}

*// Set a dismiss listener for popup window*

popupWindow.setOnDismissListener {

Toast.makeText(applicationContext,"Popup closed",Toast.LENGTH\_SHORT).show()

}

*// Finally, show the popup window on app*

TransitionManager.beginDelayedTransition(root\_layout)

popupWindow.showAtLocation(

root\_layout, *// Location to display popup window*

Gravity.CENTER, *// Exact position of layout to display popup*

0, *// X offset*

0 *// Y offset*

)

}

}

}

activity\_main.xml

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

<LinearLayout

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

xmlns:tools="http://schemas.android.com/tools"

android:id="@+id/root\_layout"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:padding="16dp"

android:background="#f8fdf8"

>

<Button

android:id="@+id/button"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Show Popup Window"

android:textAllCaps="false"

/>

</LinearLayout>

another\_view.xml

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

<LinearLayout

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

android:id="@+id/linear\_layout"

android:layout\_width="match\_parent"

android:layout\_height="match\_parent"

android:orientation="vertical"

android:background="#c7bf98"

android:padding="20dp"

>

<TextView

android:id="@+id/text\_view"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Click me to change my text color"

android:textAppearance="@style/Base.TextAppearance.AppCompat.Large"

/>

<Button

android:id="@+id/button\_popup"

android:layout\_width="wrap\_content"

android:layout\_height="wrap\_content"

android:text="Close Me"

/>

</LinearLayout>

