

# **CMSC436: Programming Handheld Systems**

# User Interface Classes

# Today's Topics

Views & View Events

View Groups, AdapterViews & Layouts

Menus & ActionBar

Dialogs

# Android User Interfaces

Activities usually display a user interface

Android provides many classes for constructing user interfaces

# View

Key building block for UI components

Occupies a rectangular space on screen

Responsible for drawing itself and for handling events

# Some Predefined Views

Button

ToggleButton

Checkbox

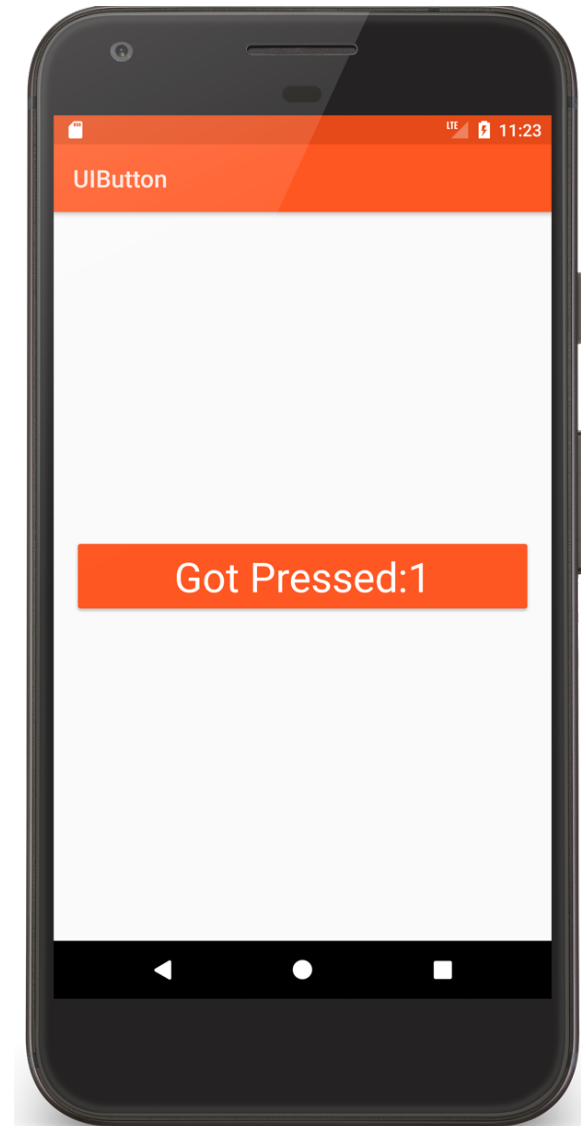
RatingBar

AutoCompleteTextView

# Button

View that can be clicked on to perform an action

# UIButton





# ButtonActivity.kt

```
class ButtonActivity : Activity() {  
    companion object {  
        private var mCount: Int = 0  
    }  
  
    private lateinit var mButton: Button  
  
    public override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.main)  
  
        // Get a reference to the Press Me Button  
        mButton = findViewById(R.id.button)
```

# ButtonActivity.kt

```
        // Reset Button Text if restarting
        savedInstanceState?.run {
            mButton.text = getString(R.string.got_pressed_string, mCount)
        }
    }

    // Set an OnClickListener on this Button
    // Called each time the user clicks the Button
    fun processClick(v: View) {

        // Maintain a count of user presses
        // Display count as text on the Button

        (v as Button).text =
            getString(R.string.got_pressed_string, ++mCount)
    }
}
```

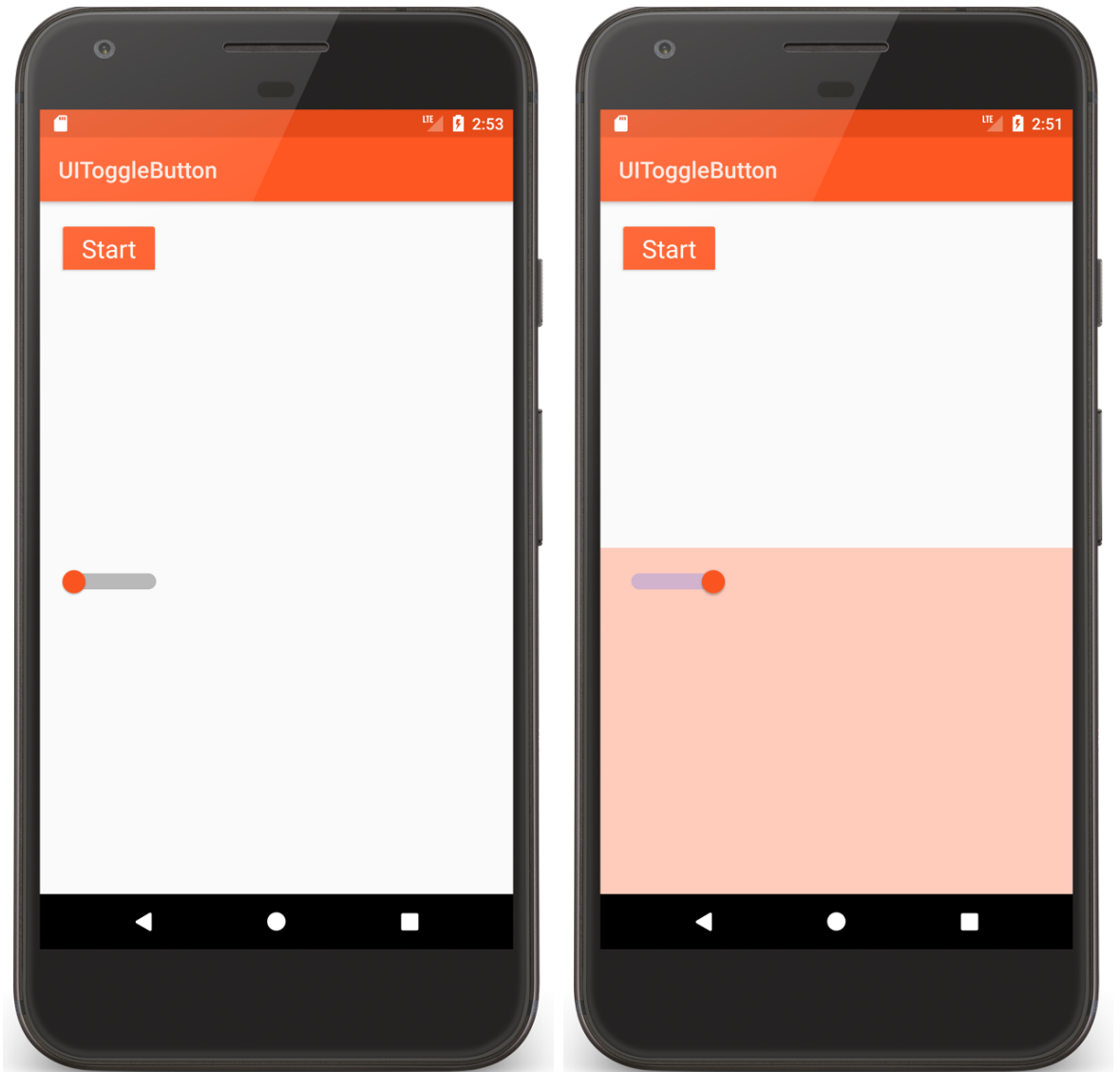
# ToggleButton

A 2-state Button

Checked/not checked state

Light indicator showing state

# UIToggleButton



# ToggleButtonActivity.kt

```
class ToggleButtonActivity : Activity() {  
    public override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.main)  
  
        // Set an setOnCheckedChangeListener on the ToggleButton  
        setListener(  
            findViewById<ToggleButton>(R.id.togglebutton),  
            findViewById<FrameLayout>(R.id.top_frame))  
  
        // Set an OnCheckedChangeListener on the Switch  
        setListener(findViewById<Switch>(R.id.switcher),  
            findViewById<FrameLayout>(R.id.bottom_frame))  
    }  
}
```

# ToggleButtonActivity.kt

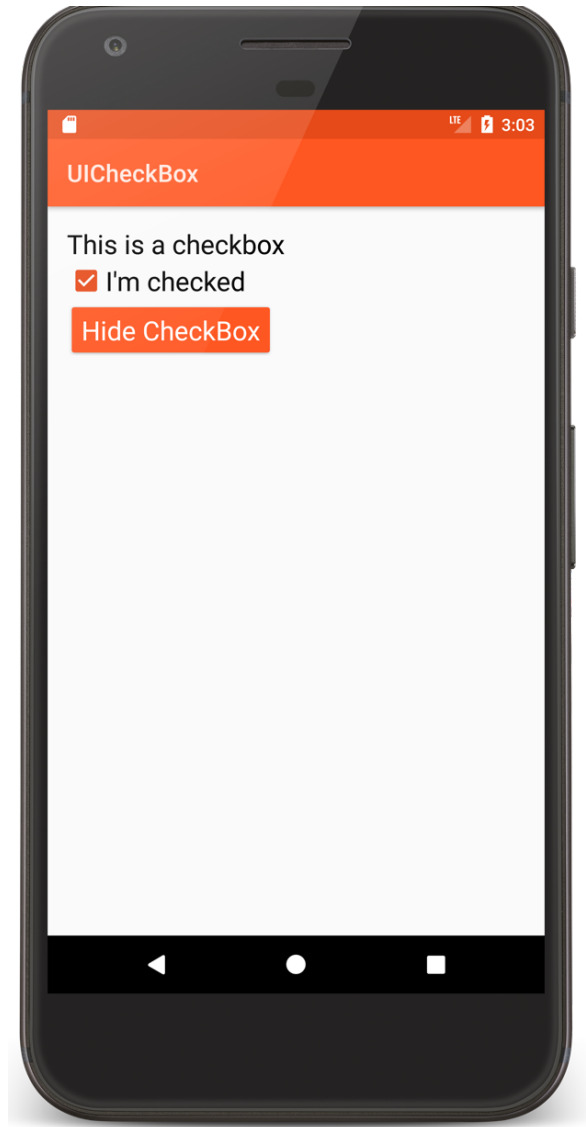
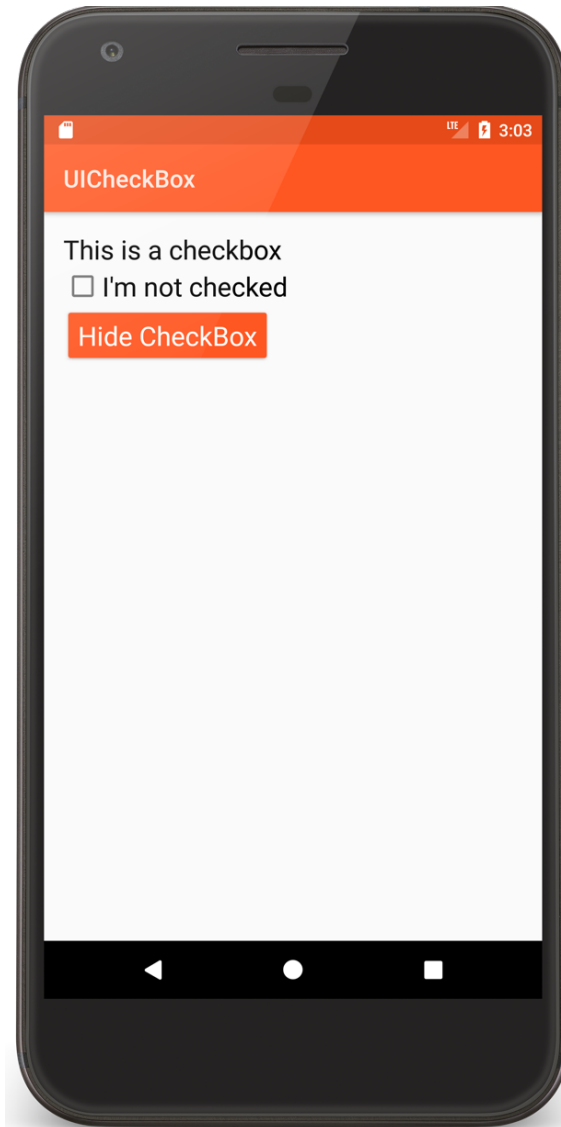
```
private fun setListener(button: CompoundButton, background: View) {
    button.setOnCheckedChangeListener { _, isChecked ->
        // Toggle the Background color between a light and dark color
        if (isChecked) {
            background.setBackgroundColor(
                resources.getColor(R.color.primary_light, null))
        } else {
            background.setBackgroundColor(Color.TRANSPARENT)
        }
    }
}
```

# Checkbox

Another kind of 2-state button

Checked/not checked

# UICheckBox





# main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    ...
    <CheckBox
        android:id="@+id/checkbox"
        ...
        android:onClick="checkBoxClickCallback"
    .../>

    <Button
        android:id="@+id/button"
        ...
        android:onClick="buttonClickCallback"
    .../>
</RelativeLayout>
```

# CheckBoxActivity.kt

```
// Set with android:onClick
fun checkBoxClickCallback(view: View) {
    setCheckedStateAndText()
}

private fun setCheckedStateAndText() {
    // Check whether CheckBox is currently checked
    // Set CheckBox text accordingly
    if (mCheckBox.isChecked) {
        mCheckBox.text = getString(R.string.im_checked_string)
    } else {
        mCheckBox.text = getString(R.string.im_not_checked_string)
    }
}
```

# CheckBoxActivity.kt

```
// Set with android:onClick
fun buttonClickCallback(view: View) {

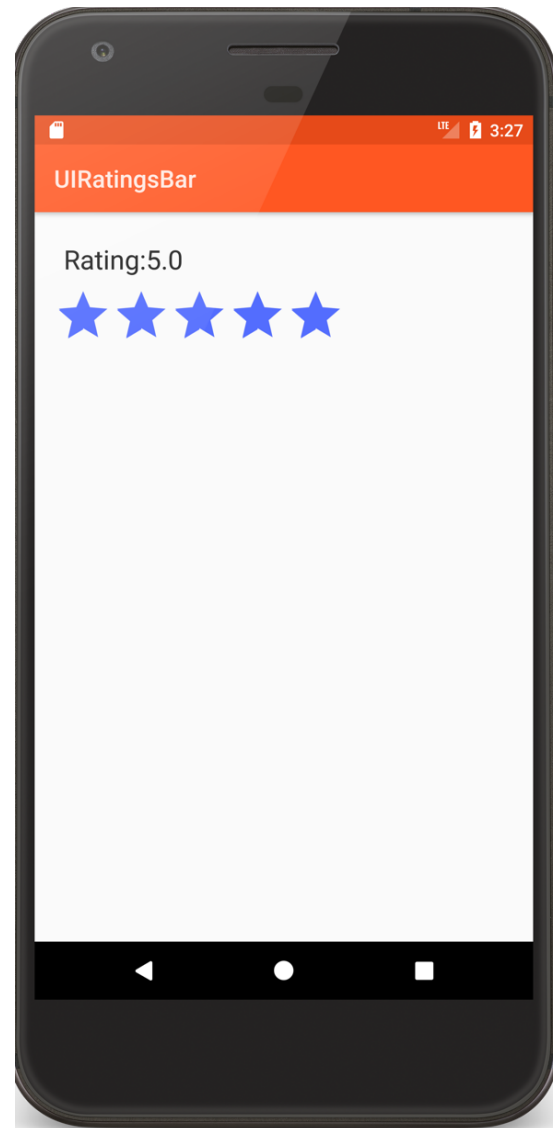
    // Toggle the CheckBox's visibility state
    // Set the Button text accordingly
    if (mCheckBox.isShown) {
        mCheckBox.visibility = View.INVISIBLE
        mButton.text = getString(R.string.unhide_checkbox_string)
    } else {
        mCheckBox.visibility = View.VISIBLE
        mButton.text = getString(R.string.hide_checkbox_string)
    }
}
```

# RatingBar

A view comprising a row of stars

The user can click or drag the stars to highlight some number of them

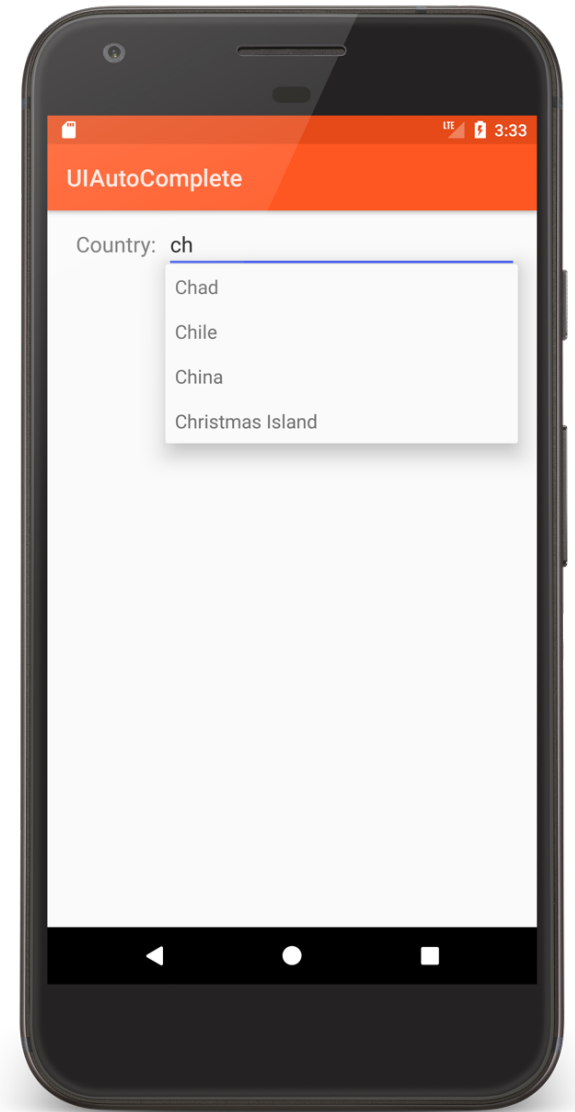
# UIRatingBar



# AutoCompleteTextView

An editable text field that provides completion suggestions as the user types in text

# UIAutoComplete TextView



# Common View Operations

Set visibility: Show or hide View

Set checked state: Checked or not checked

Set listeners: Code that will be executed when specific events occur

Set properties: Opacity, background, rotation

Manage input focus: Allow View to take focus, request focus, etc.



# View Event Sources

## User interaction

- Touch

- Keyboard/trackball/D-pad

## System control

- Lifecycle changes

# Handling View Events

Will often handle events using listeners

Many Listener interfaces defined by View class

# View Listener interfaces

`OnClickListener.onClick()`

View has been clicked

`OnLongClickListener.onLongClick()`

View has been pressed & held

# View Listener interfaces

`OnFocusChangeListener.onFocusChange()`

View has received or lost focus

`OnKeyListener.onKey()`

View is about to receive a hardware key press

# Displaying Views

Views within a UI are organized as a tree

Displaying/refreshing the UI has multiple steps

- Measure – get dimensions of each View

- Layout – Position each View

- Draw – Draw each view

# Handling View Events

Typically, create View subclasses

Then, override View methods

# Handling View Events

`onMeasure()`

Determine the size of this View and its children

`onLayout()`

Assign a size and position to all View's children

`onDraw()`

Render View content

# Handling View Events

`onFocusChanged()`

Called when View's focus state has changed

`onKeyUp()`, `onKeyDown()`

Called when a hardware key event has occurred

`onWindowVisibilityChanged()`

Window containing view has changed its visibility status



# ViewGroup

An invisible View that contains other Views

Used for grouping & organizing a set of Views

Base class for View containers and Layouts

# Some Predefined ViewGroups

RadioGroup

TimePickerFragment

DatePickerFragment

WebView

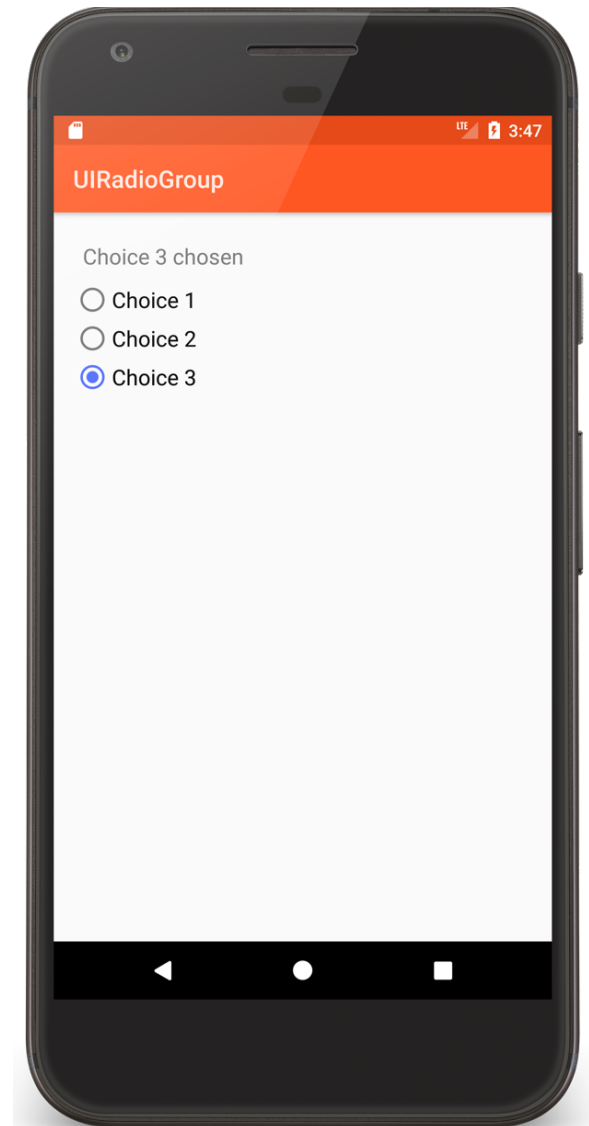
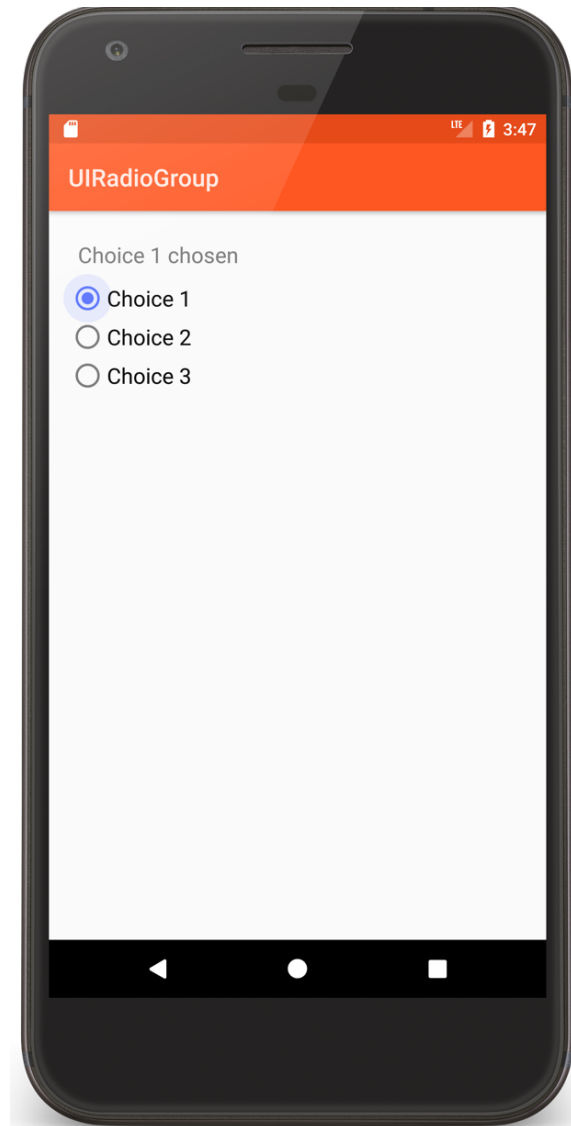
MapView

# RadioGroup

A ViewGroup containing a set of Radio Buttons

Only one RadioButton can be selected at any one time

# UIRadioGroup



# main.xml

```
<RadioGroup
    android:id="@+id/radio_group"
    ...>
    <RadioButton
        android:id="@+id/choice1"
        android:onClick="radioButtonClickCallback"
    .../>
    <RadioButton
        android:id="@+id/choice2"
        android:onClick="radioButtonClickCallback"
    .../>
    <RadioButton
        android:id="@+id/choice3"
        android:onClick="radioButtonClickCallback"
    .../>
</RadioGroup>
```

```
<RadioGroup
  android:id="@+id/radio_group"
  ... >
  <RadioButton
    android:id="@+id/choice1"
    android:onClick="radioButtonClickCallback"
    .../>
  <RadioButton
    android:id="@+id/choice2"
    android:onClick="radioButtonClickCallback"
    .../>
  <RadioButton
    android:id="@+id/choice3"
    android:onClick="radioButtonClickCallback"
    ... />
</RadioGroup>
```

# RadioGroupActivity.kt

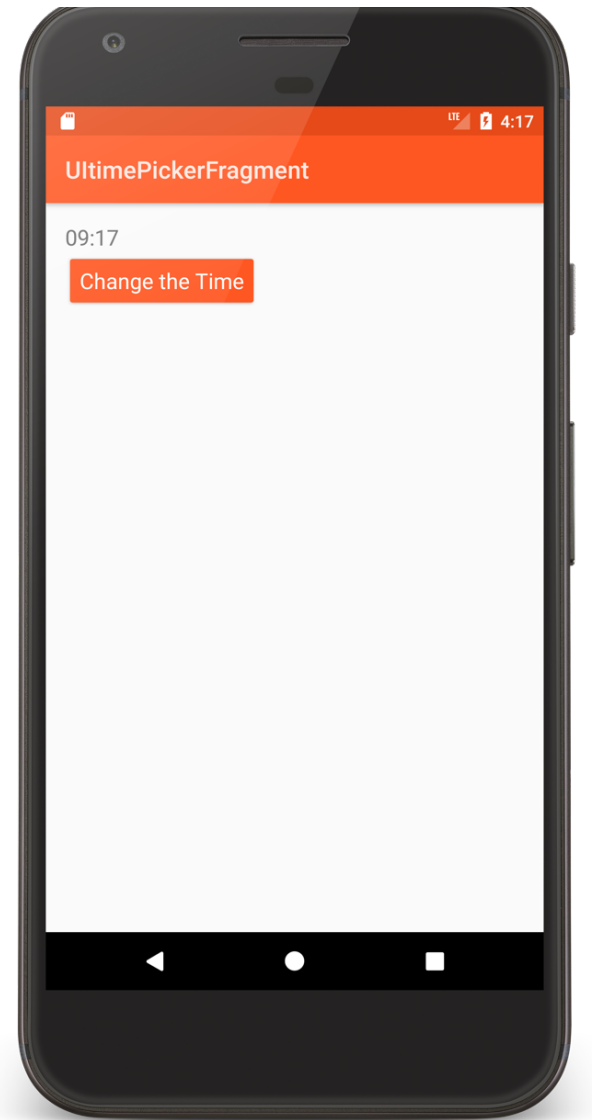
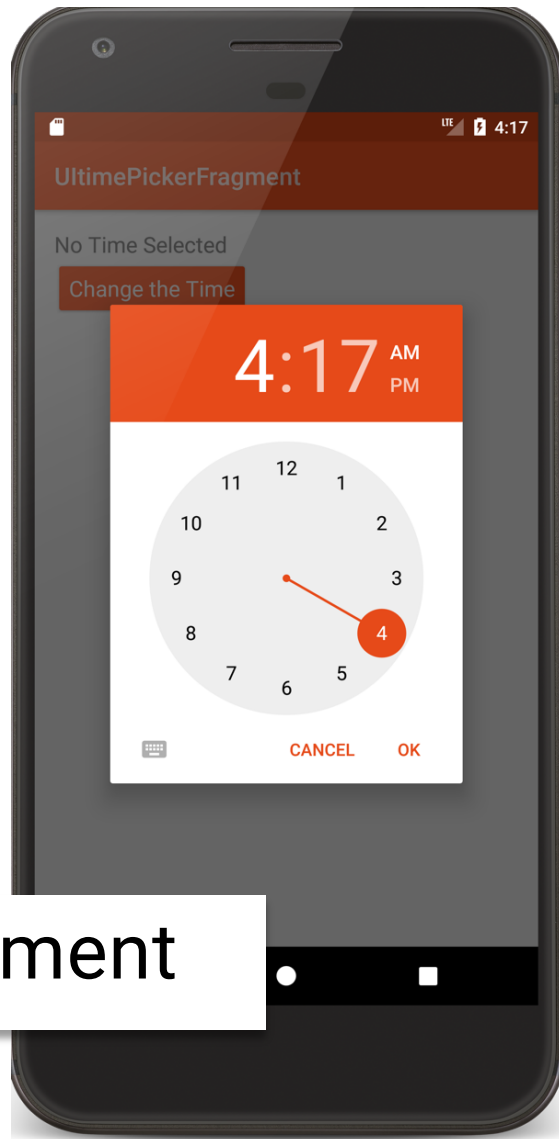
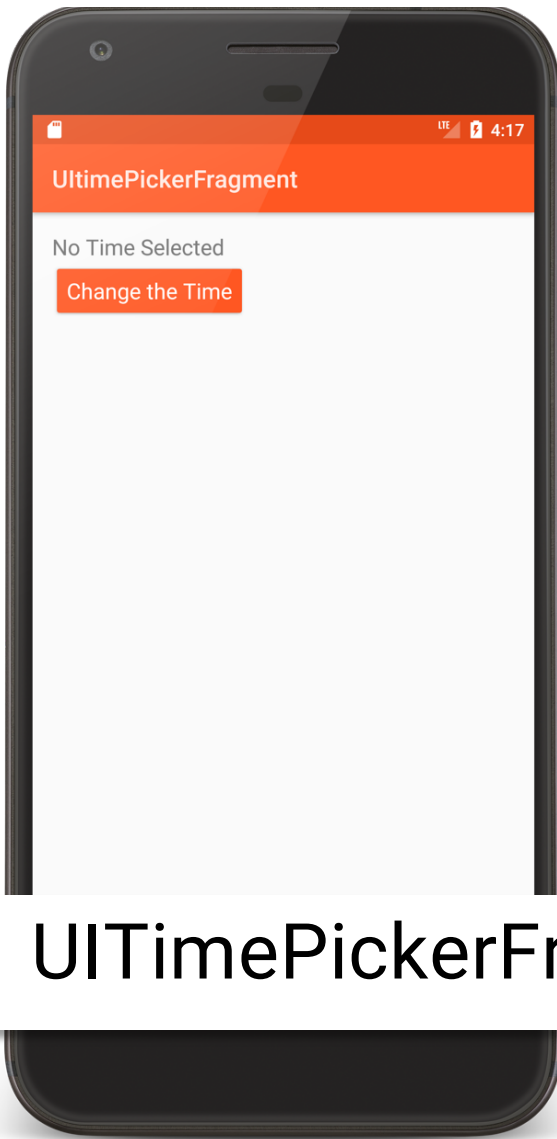
```
// Define a generic listener for all three RadioButtons in the RadioGroup
fun radioButtonClickCallback(v: View) {
    val rb = v as RadioButton

    // RadioButtons report each click, even if the toggle state doesn't change
    if (rb.isChecked) {
        mTextView.text = getString(R.string.chosen_string, rb.text)
    }
}
```

# TimePickerFragment

A ViewGroup that allows the user to select a time

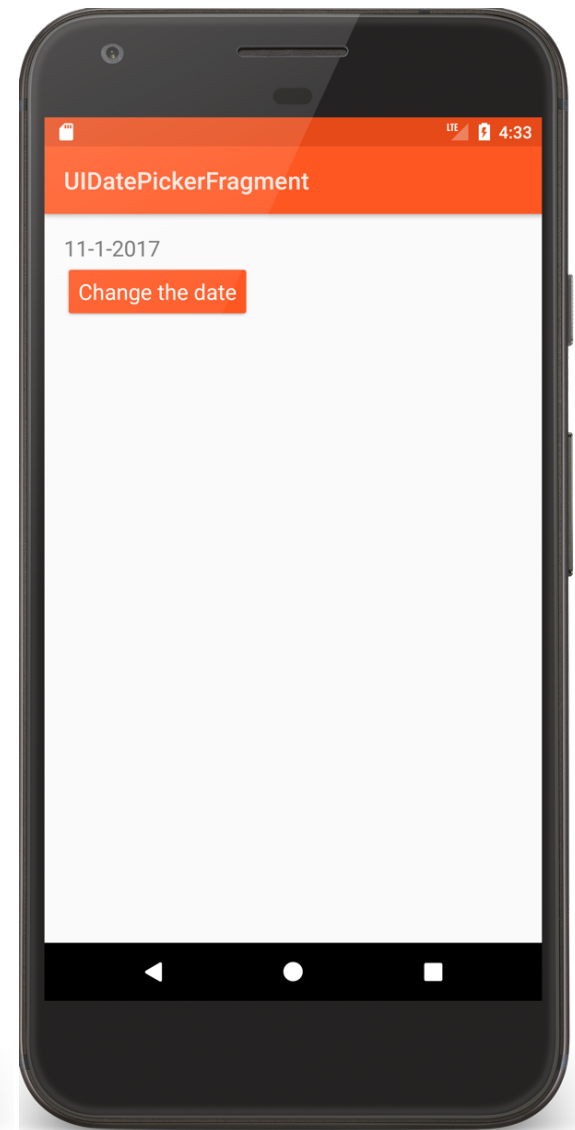
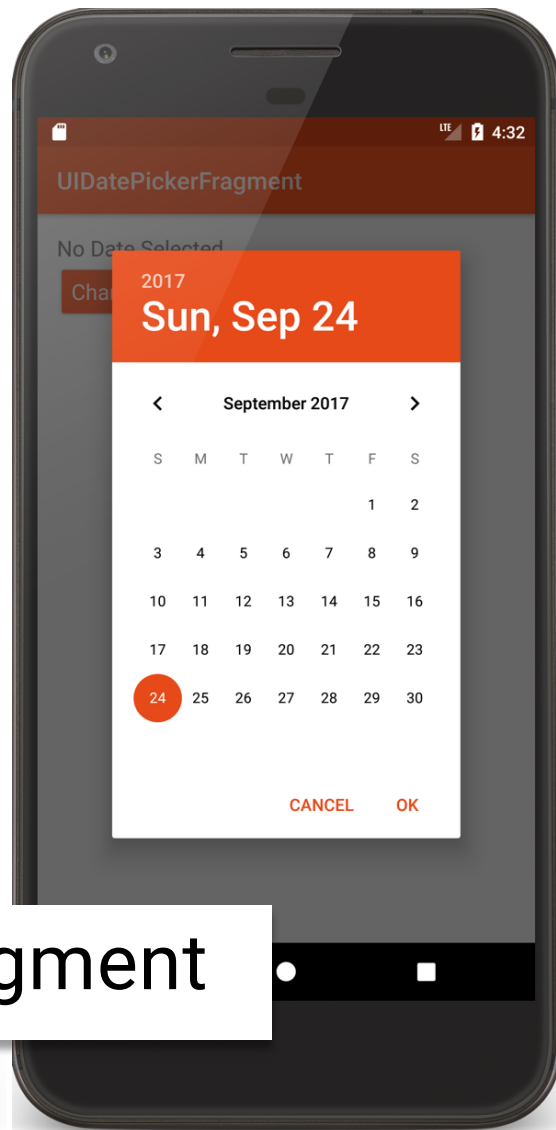
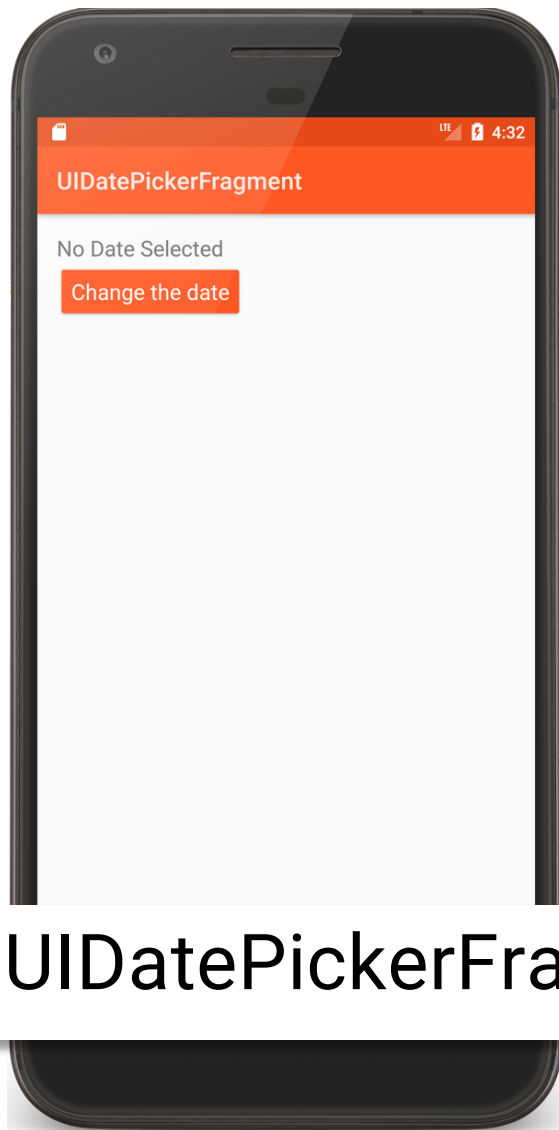




UTimePickerFragment

# DatePickerFragment

A ViewGroup that allows the user to select a date



UIDatePickerFragment

# WebView

A ViewGroup that displays a web page

UIWebView



# main.xml

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

    <WebView
        android:id="@+id/webview"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</RelativeLayout>
```

# WebViewActivity.kt

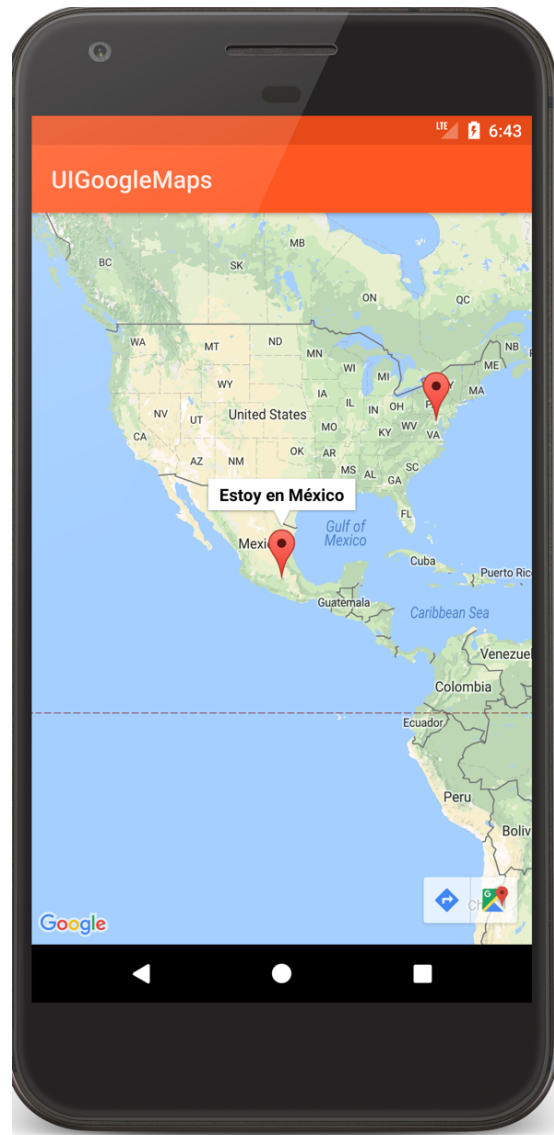
```
class WebViewActivity : Activity() {  
    ...  
    public override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.main)  
  
        mWebView = findViewById(R.id.webview)  
  
        // Set a kind of listener on the WebView so the WebView can intercept  
        // URL loading requests if it wants to  
        mWebView.webViewClient = HelloWebViewClient()  
  
        // Add Zoom controls to WebView  
        mWebView.settings.builtInZoomControls = true  
  
        mWebView.loadUrl("https://www.cs.umd.edu/~aporter/Tmp/bee.html")  
    }  
}
```

# MapView

A ViewGroup that displays a Map



# UIGoogleMaps



# Adapters & AdapterViews

AdapterViews are Views whose children and data are managed by an Adapter

Interaction pattern

Adapter manages the data and provides data Views to AdapterView

AdapterView displays the data Views

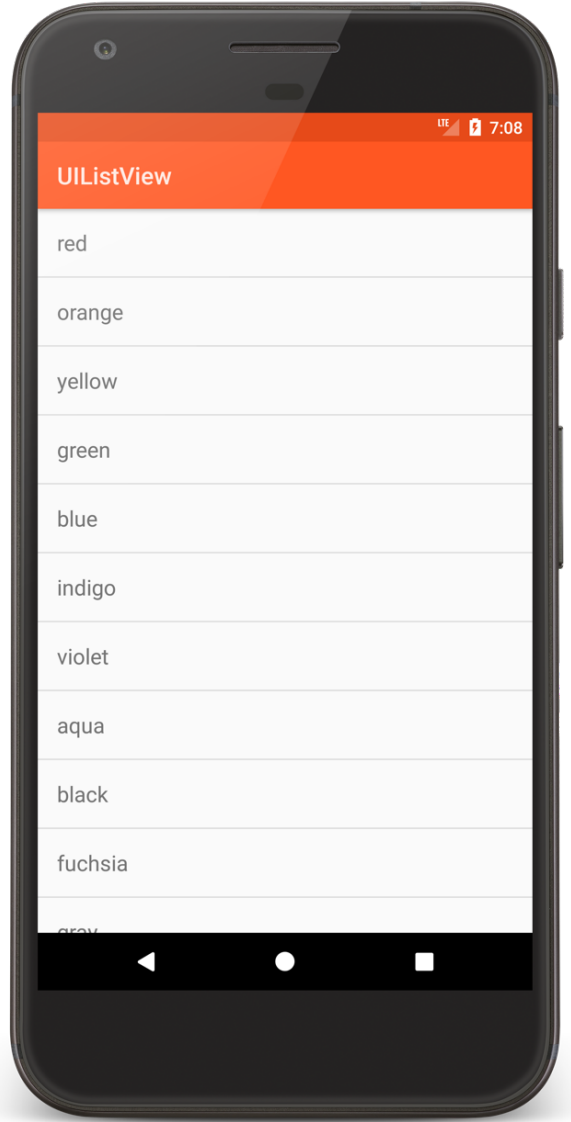
# ListView

An AdapterView that displays a scrollable list of selectable items

Data items managed by a ListAdapter

ListView can filter the list of items based on text input

# UITableView



# list\_item.xml

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:padding="@dimen/activity_margin"  
    android:textAppearance="@android:style/TextAppearance.Material.Medium" />
```

# ListViewActivity.kt

```
public override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    // Create a new Adapter containing a list of colors
    // Set the adapter on this ListActivity's built-in ListView
    listAdapter = ArrayAdapter(
        this, R.layout.list_item,
        resources.getStringArray(R.array.colors)
    )

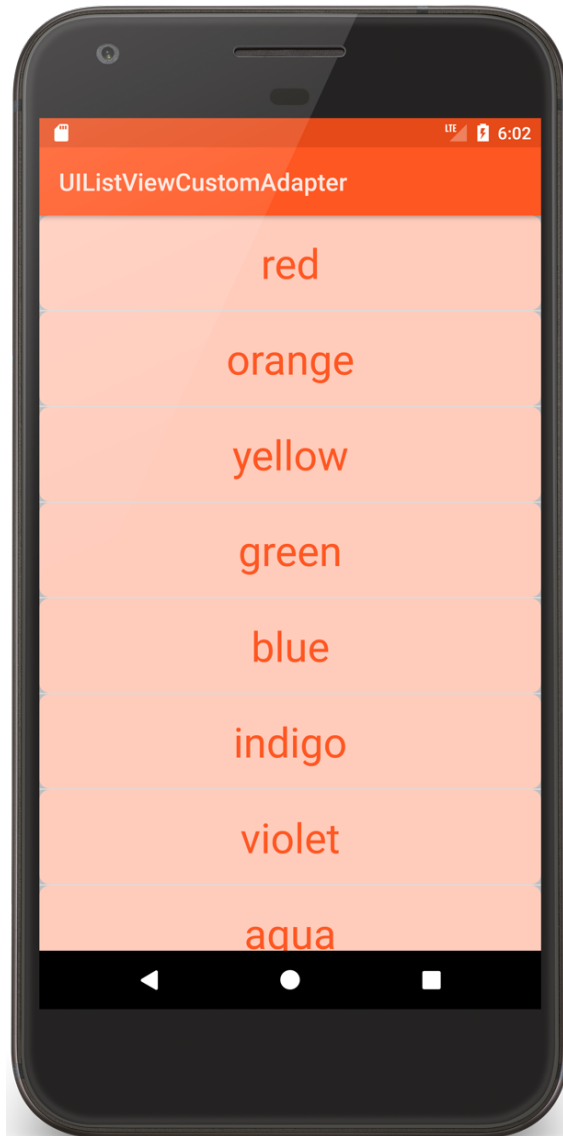
    val lv = listView

    // Enable filtering when the user types in the virtual keyboard
    lv.isTextFilterEnabled = true
}
```

# ListViewActivity.kt

```
// Set an setOnItemClickListener on the ListView
lv.setOnItemClickListener =.setOnItemClickListener { _, view, _, _ ->
    // Display a Toast message indicting the selected item
    Toast.makeText(applicationContext,
        (view as TextView).text,
        Toast.LENGTH_SHORT
    ).show()
}
}
```

# UIListViewWith CustomAdapter





# list\_item.xml

```
<TextView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/text"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="@drawable/list_background"
    android:gravity="center"
    android:padding="16dp"
    android:textAppearance=
        "@android:style/TextAppearance.Material.Display1"
    android:textColor="@color/divider">
</TextView>
```

# list\_background.xml

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"  
    android:shape="rectangle">  
    <solid android:color="@color/primary_light" />  
    <corners android:radius="8dp" />  
</shape>
```

# ListViewActivity.kt

```
public override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)

    // Create a new Adapter containing a list of colors
    // Set the adapter on this ListActivity's built-in ListView
    listAdapter = ListViewAdapter(
        this,
        R.layout.list_item,
        resources.getStringArray(R.array.colors)
    )
    ...
}
```

# ListViewAdapter.kt

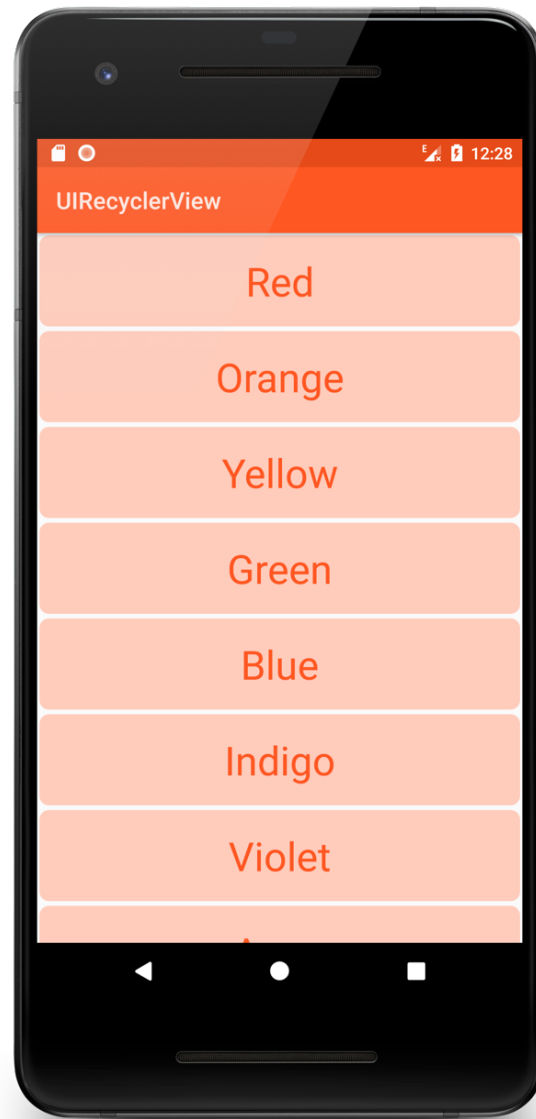
```
class ListViewAdapter(context: Context, resource: Int, objects: Array<String>):  
    ArrayAdapter<String>(context, resource, objects) {  
    ...  
    override fun getView(position: Int, convertView: View?,  
        parent: ViewGroup): View {  
        val newView: View  
        // Check for recycled View  
        if (null == convertView) {  
            // Not recycled. Create the View  
            newView = mLayoutInflater.inflate(R.layout.list_item, parent, false)  
            // Cache View information in ViewHolder Object  
            val viewHolder = ViewHolder()  
            newView.tag = viewHolder  
            viewHolder.textView = newView.findViewById(R.id.text)  
        } else {  
            newView = convertView  
        }  
    }  
}
```

# ListViewAdapter.kt

```
// Set the View's data
// Retrieve the viewHolder Object
val storedViewHolder = newView.tag as ViewHolder
//Set the data in the data View
storedViewHolder.textView.text = getItem(position)
return newView
}

// The ViewHolder class.
// See: http://developer.android.com/training/improving-layouts/smooth-scrolling.html#ViewHolder
internal class ViewHolder {
    lateinit var textView: TextView
}
}
```

# UIRecyclerView



# RecyclerViewActivity.kt

```
public override fun onCreate(savedInstanceState: Bundle?) {  
    ...  
    //Set up RecyclerView  
    val mRecyclerView = findViewById<RecyclerView>(R.id.list)  
    //Set the layout manager  
    mRecyclerView.layoutManager = LinearLayoutManager(this)  
    // Set up the adapter  
    val names = ArrayList<String>()  
    Collections.addAll(names, *resources.getStringArray(R.array.colors))  
    val mAdapter = MyRecyclerViewAdapter(names, R.layout.list_item)  
    mRecyclerView.adapter = mAdapter  
}
```

# MyRecyclerAdapter.kt

```
internal class MyRecyclerViewAdapter(private val mName: List<String>?,
    private val mRowLayout: Int) :
    RecyclerView.Adapter<MyRecyclerViewAdapter.ViewHolder>() {

    // Create ViewHolder which holds a View to be displayed
    override fun onCreateViewHolder(viewGroup: ViewGroup, i: Int):ViewHolder {
        val v = LayoutInflater.from(viewGroup.context).
            inflate(mRowLayout, viewGroup, false)
        return ViewHolder(v)
    }

    // Binding: The process of preparing a child view to display data
    // corresponding to a position within the adapter.
    override fun onBindViewHolder(viewHolder: ViewHolder, i: Int) {
        viewHolder.mName.text = mName!![i]
    }
}
```



# MyRecyclerAdapter.kt

```
override fun getItemCount(): Int {
    return mNames?.size ?: 0
}

class ViewHolder internal constructor(itemView: View) :
    RecyclerView.ViewHolder(itemView),
    View.OnClickListener {
    internal val mName: TextView = itemView.findViewById(R.id.text)
    init {
        itemView.setOnClickListener(this)
    }
    override fun onClick(view: View) {
        // Display a Toast message indicting the selected item
        Toast.makeText(view.context, mName.text, Toast.LENGTH_SHORT).show()
    }
}
...
```

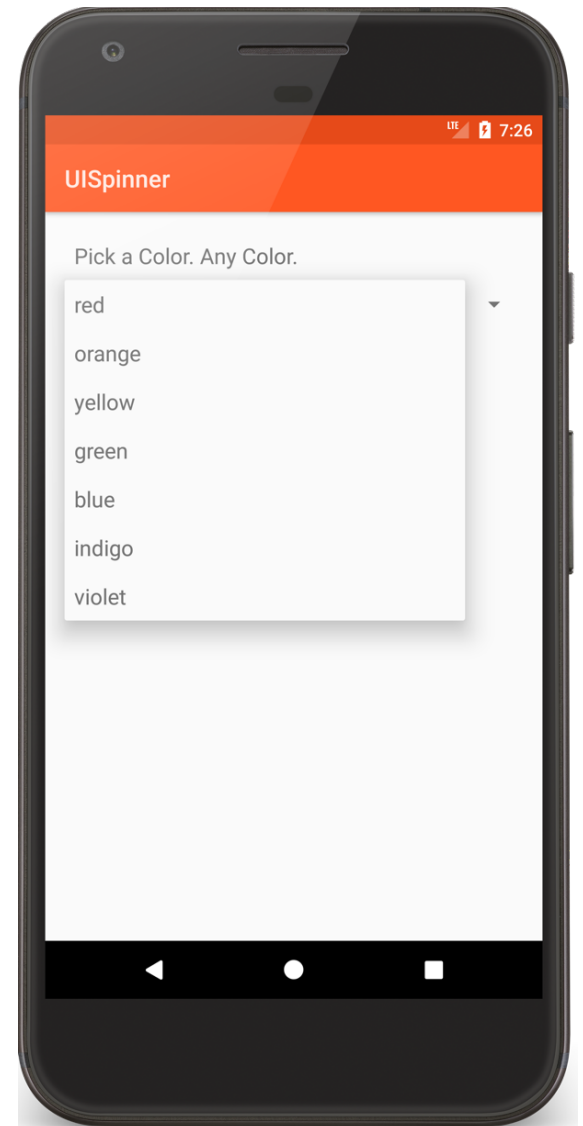
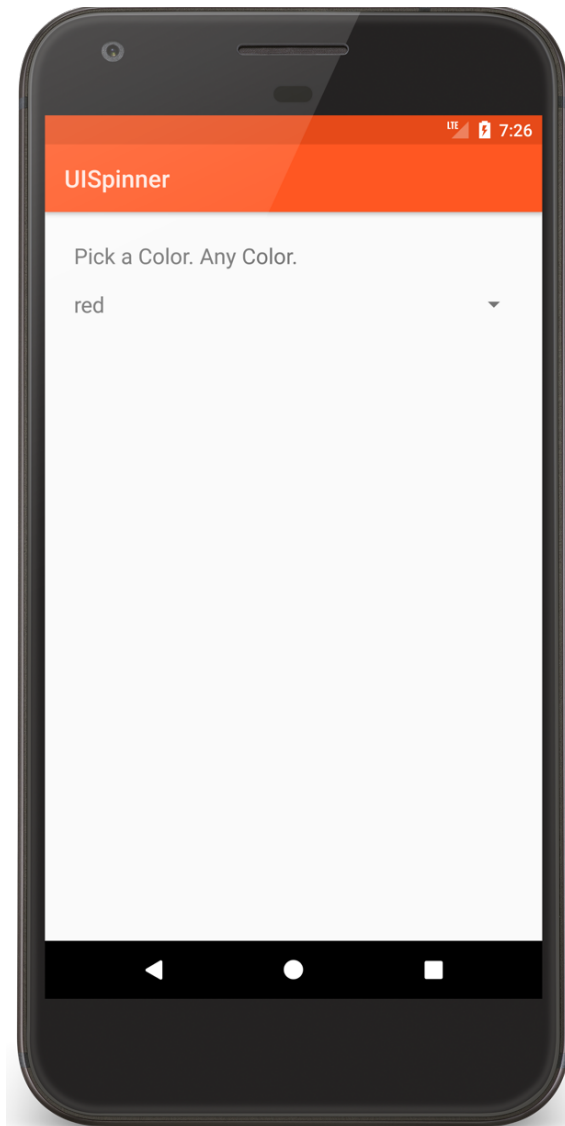
# Spinner

An AdapterView that provides a scrollable list of items

User can select one item from the list

Items managed by a SpinnerAdapter

# UISpinner



# ViewPager

A ViewGroup showing a horizontally scrolling list

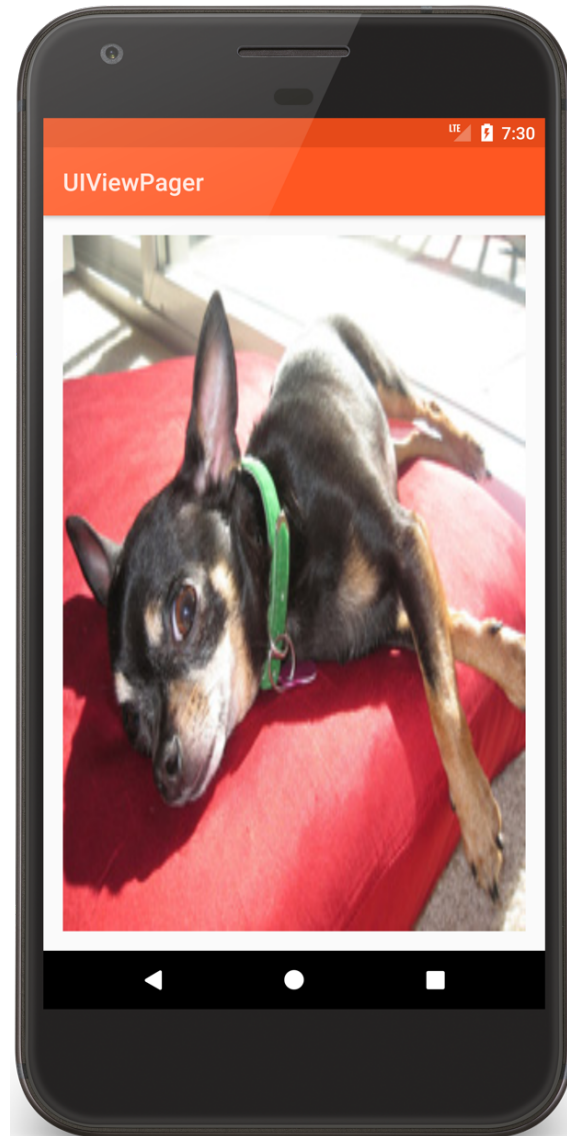
Items managed by a PagerAdapter

Two builtin PagerAdapters using Fragments

FragmentPagerAdapter

FragmentStatePagerAdapter

UIViewPager



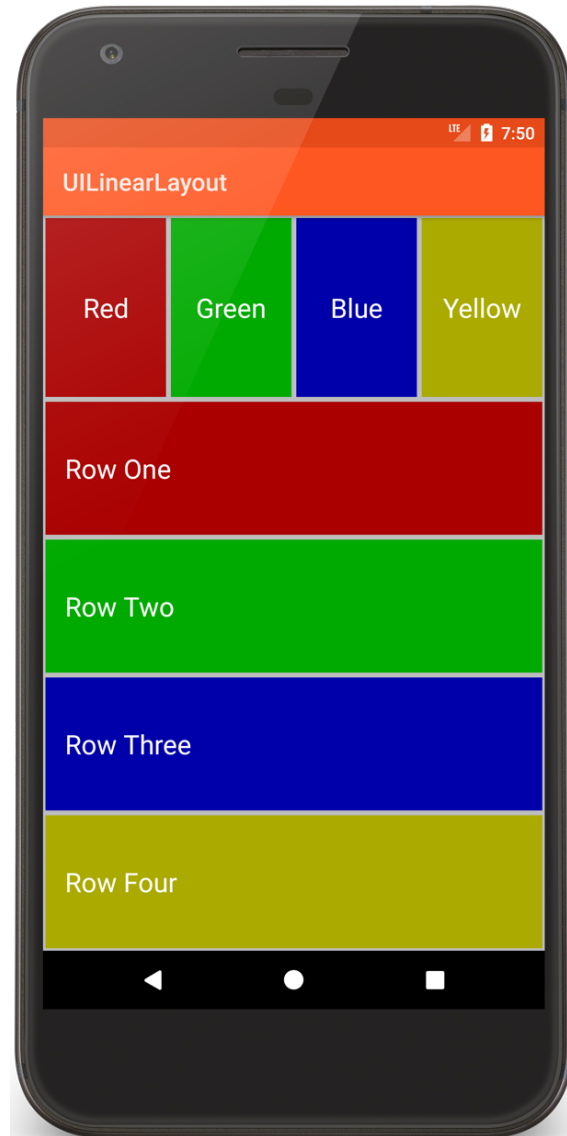
# Layouts

A generic Viewgroup that defines a structure/rules for positioning the Views it contains

# LinearLayout

Child Views arranged in a single horizontal or vertical row

# LinearLayout





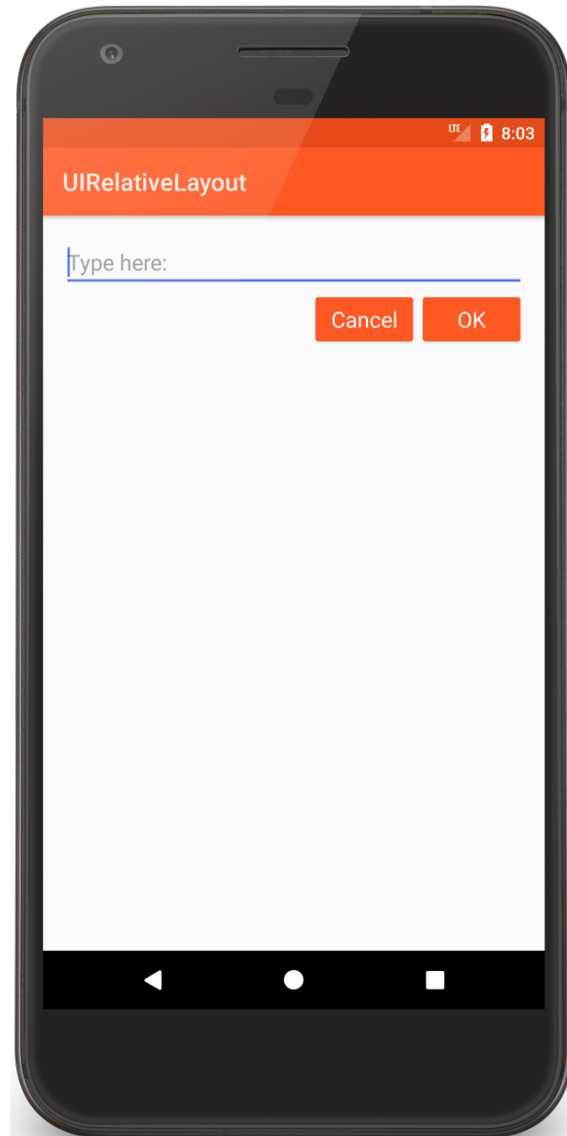
# main.xml

```
<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:background="@color/divider"
    android:orientation="vertical">
    <!--
    Inner LinearLayout with horizontal orientation
    and layout weight of 1 out of 4
    -->
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="0dp"
        android:layout_weight="1"
        android:orientation="horizontal">
        ...
```

# RelativeLayout

Child Views are positioned relative to each other and to parent View

# UIRelativeLayout



# main.xml

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"  
    android:layout_width="match_parent"  
    android:layout_height="match_parent"  
    android:padding="@dimen/activity_margin">
```

```
<!-- Note the use of android:hint to put explanatory text in the  
    EditText -->
```

```
<EditText  
    android:id="@+id/entry"  
    android:layout_width="match_parent"  
    android:layout_height="wrap_content"  
    android:hint="@string/type_here_string"  
    android:textAppearance=  
        "@android:style/TextAppearance.Material.Medium"  
    android:inputType="textPersonName"  
    android:autofillHints="username" />
```

# main.xml

```
<Button
    android:id="@+id/ok_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentEnd="true"
    android:layout_below="@id/entry"
    .../>
<Button
    android:id="@+id/cancel_button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignTop="@id/ok_button"
    android:layout_toStartOf="@id/ok_button"
    .../>
</RelativeLayout>
```

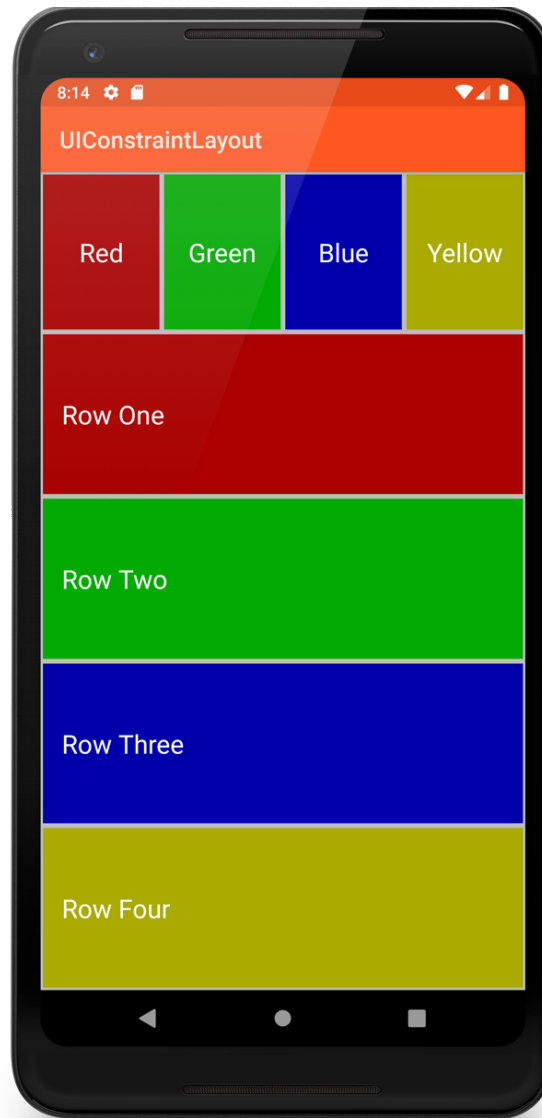
# ConstraintLayout

Combines features of LinearLayout and RelativeLayout

Avoids deeply nested layout structures with goal of improving drawing performance

Considered default UI layout for Android going forward

# UIConstraintLayout

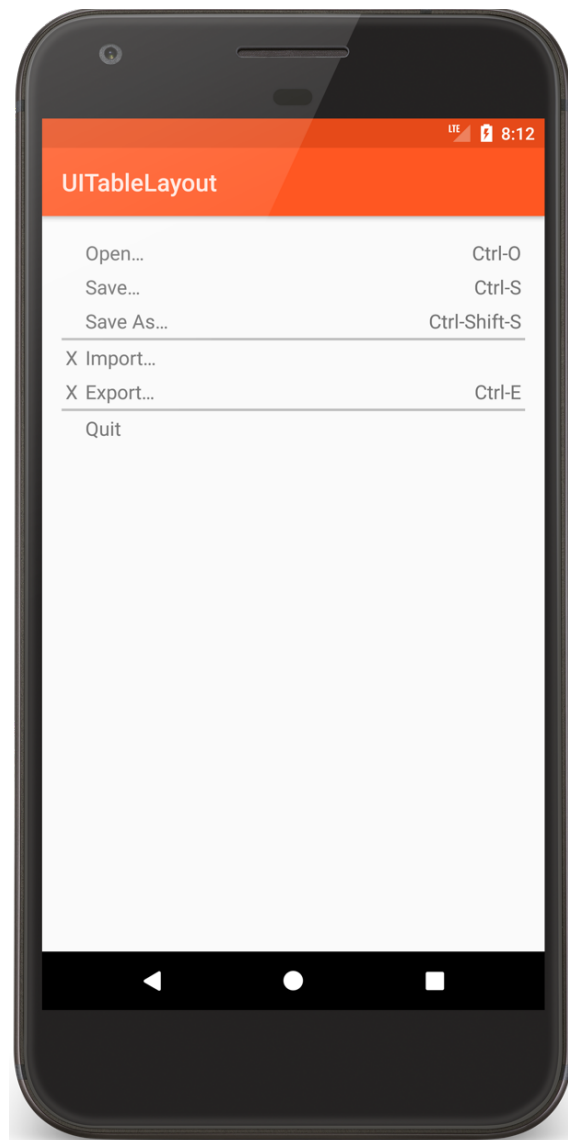


# TableLayout

Child views arranged into rows & columns



# UITableView



# Menus and ActionBar

Activities support menus

Activities can

- Add items to a menu

- Handle clicks on the menu items

# Menu Types

## Options

Menu shown when user presses the menu button

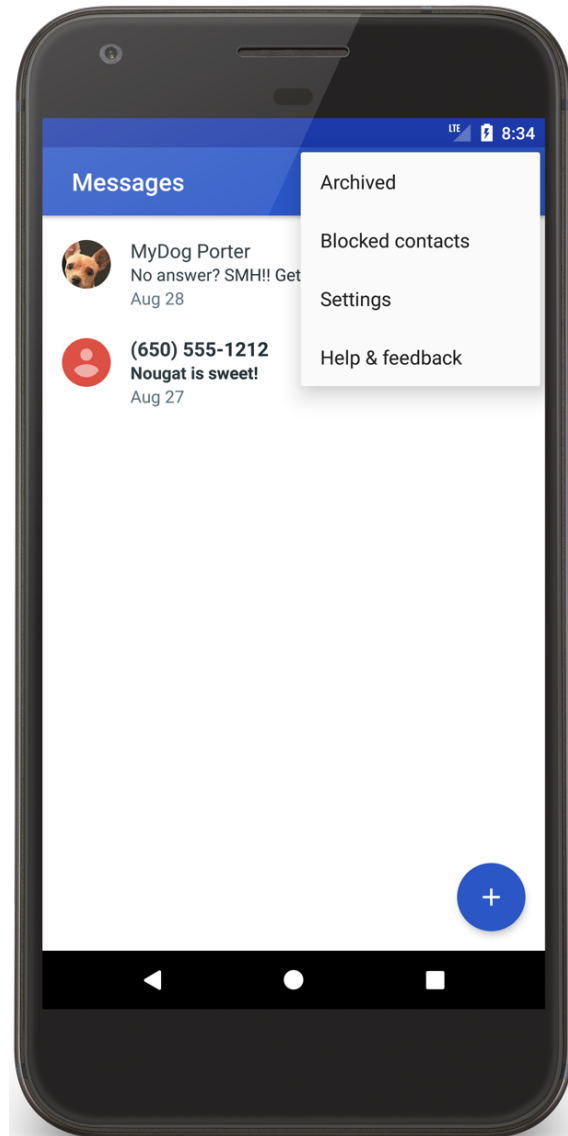
## Context

View-specific menu shown when user touches and holds the View

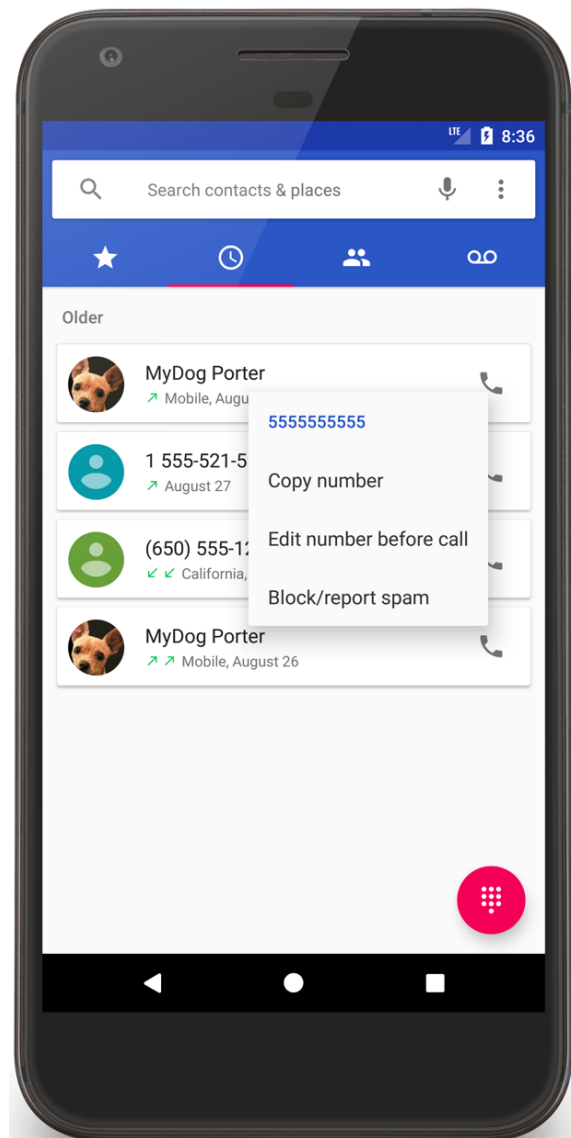
## Submenu

A menu activated when user touches a visible menu item

# Options Menus



# Context Menus



# Creating Menus

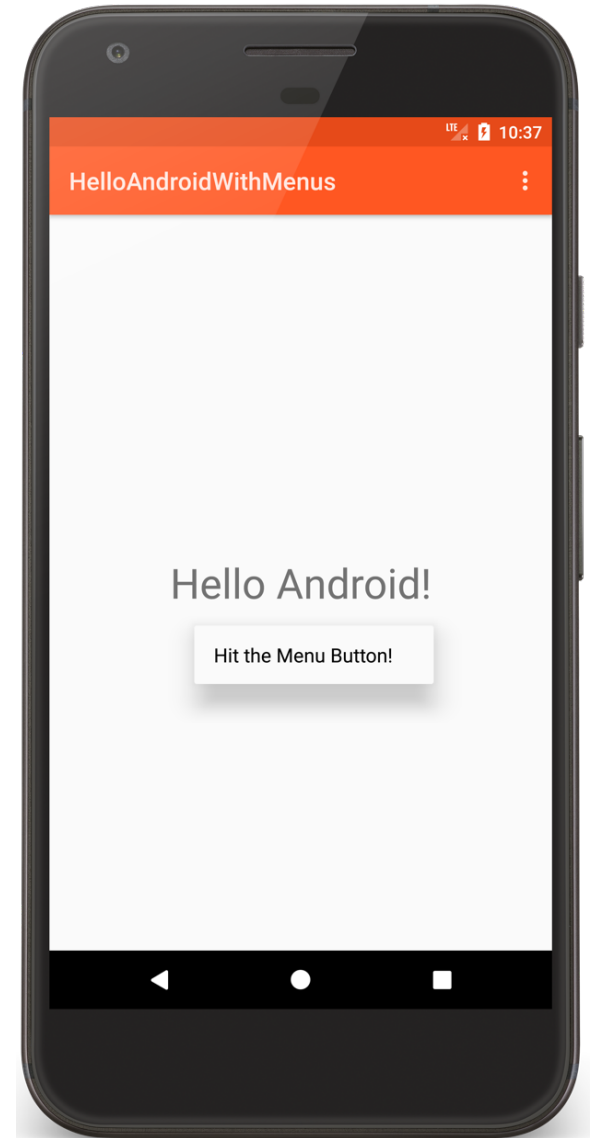
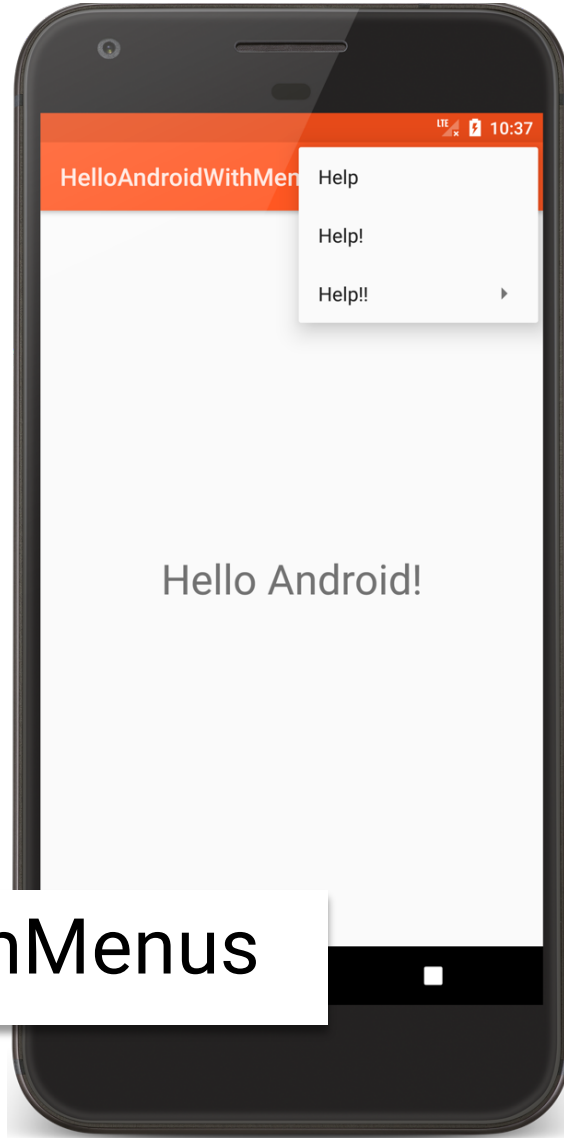
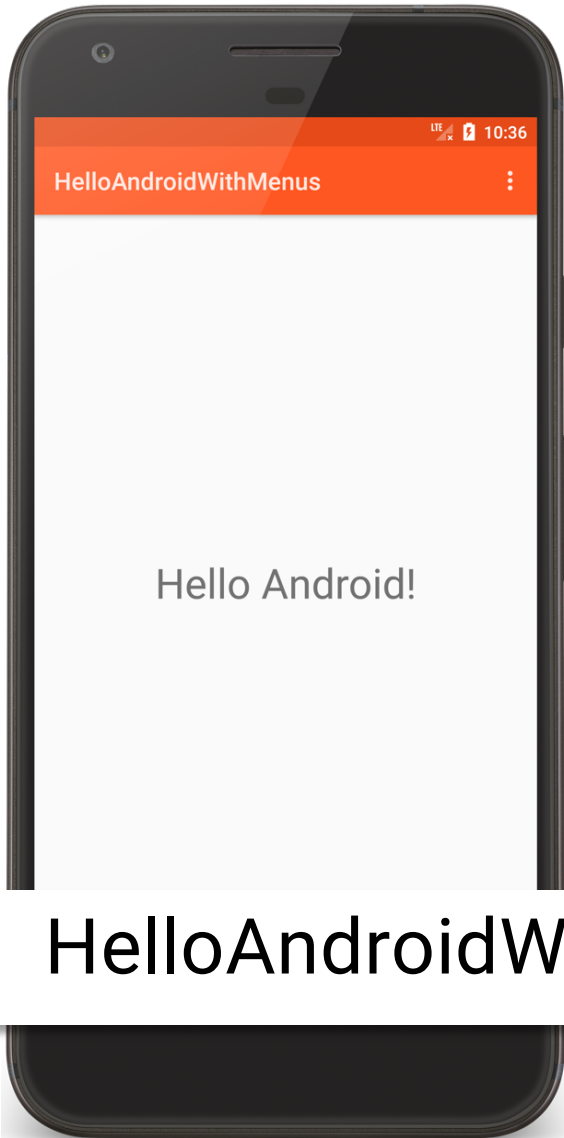
Define menu resource in XML file

Store in res/menu/filename.xml

# Creating Menus

Inflate menu resource using Menu Inflater in  
`onCreate{Options,Context}Menu()` methods

Handling item selection in appropriate  
`on{Options,Context}ItemsSelected()` methods



# HelloAndroidWithMenus



# HelloAndroidWithMenuActivity.kt

```
class HelloAndroidWithMenuActivity : Activity() {  
    public override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
  
        setContentView(R.layout.main)  
  
        // Long presses on TextView invoke Context Menu  
        registerForContextMenu(findViewById<TextView>(R.id.text_view))  
    }  
}
```

# HelloAndroidWithMenuActivity.kt

```
// Create Options Menu
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    val inflater = menuInflater
    inflater.inflate(R.menu.top_menu, menu)
    return true
}
```

# HelloAndroidWithMenuActivity.kt

```
// Process clicks on Options Menu items
override fun onOptionsItemSelected(item: MenuItem): Boolean {
    return when (item.itemId) {
        R.id.help -> {
            showToast(R.string.helped_string)
            true
        }
        R.id.more_help -> {
            showToast(R.string.helped_more_string)
            true
        }
        R.id.even_more_help -> true
        else -> false
    }
}
```

# HelloAndroidWithMenuActivity.kt

```
// Create Context Menu
override fun onCreateContextMenu(
    menu: ContextMenu, v: View,
    menuInfo: ContextMenuInfo?
) {
    super.onCreateContextMenu(menu, v, menuInfo)
    val inflater = menuInflater
    inflater.inflate(R.menu.context_menu, menu)
}
```

# HelloAndroidWithMenuActivity.kt

```
// Process clicks on Context Menu Items
override fun onContextItemSelected(item: MenuItem): Boolean {
    return when (item.itemId) {
        R.id.help_guide -> {
            showToast(R.string.context_menu_shown_string)
            true
        }
        else -> false
    }
}
```

# Menus

Many other features supported

- Grouping menu items

- Binding shortcut keys to menu items

- Binding Intents to menu items

# ActionBar

Similar to Application Bar in many desktop applications

Enables quick access to common operations

# FragmentManagerDynamicLayoutWithActionBar

Shows play titles and one quote from selected play

Provides actions for the ActionBar

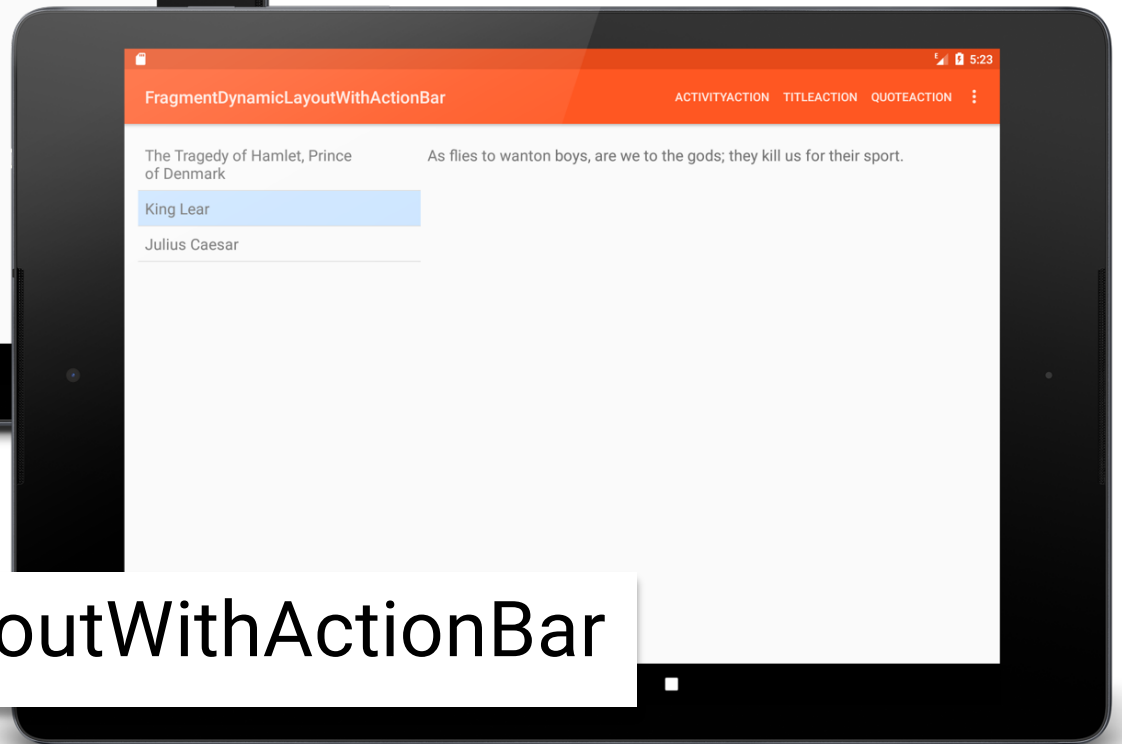
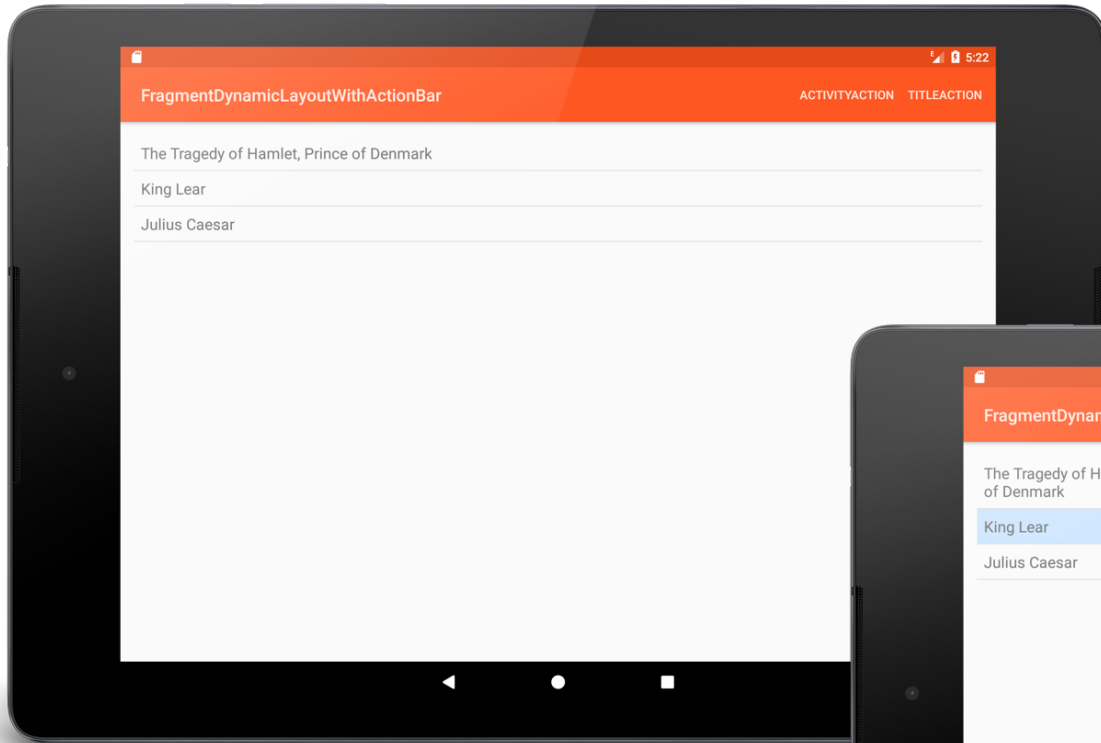
Three main objects

- QuoteViewerActivity

- TitleFragment

- QuoteFragment





FragmentDynamicLayoutWithActionBar

# Dialogs

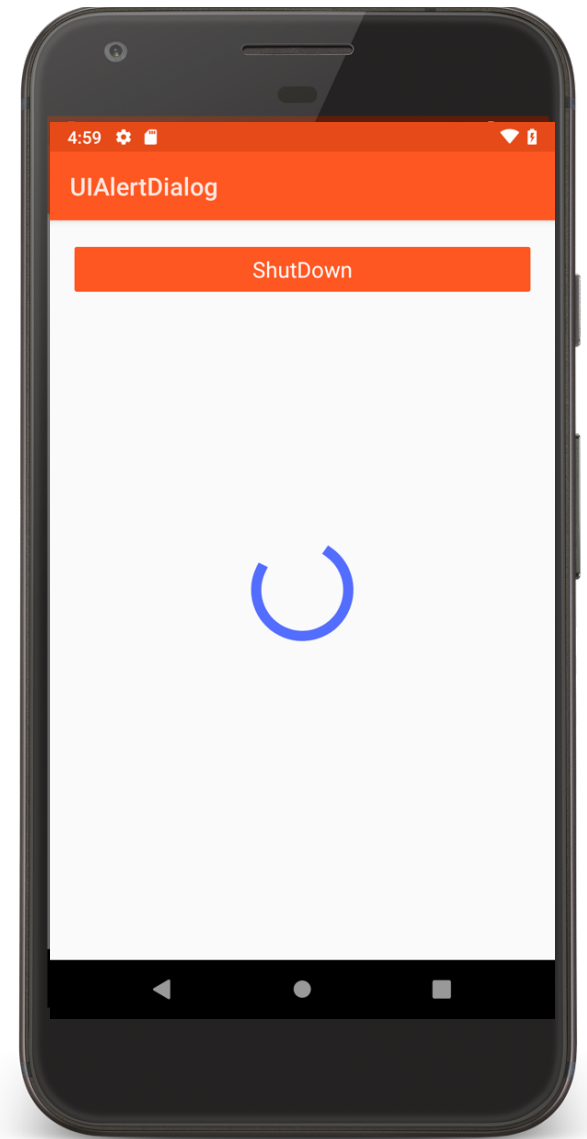
Independent subwindows used by Activities to communicate with user

# Dialog Subclasses

AlertDialog

DatePickerDialog

TimePickerDialog



UIAlertDialog

Next

The BroadcastReceiver class

# Example Applications

UIButton

UIToggleButton

UICheckbox

UIRatingBar

UIAutoCompleteTextView

UIRadioGroup

UITimePickerFragment

UIDatePickerFragment

UIWebView

UIGoogleMaps

UIListView

UIListViewWithCustom  
Adapter

UIRecyclerView

UISpinner

UIViewPager

UILinearLayout

UIRelativeLayout

UIConstraintLayout

UITableLayout

UIGridView

HelloAndroidWithMenus

UIAlertDialog