Mobile Application Development

By,

Prof. Himanshu H Patel,

Prof. Hiten M Sadani

U. V. Patel College of Engineering, Ganpat University

Basic Widgets

Basic Widgets

Text View:

TextView

Added in API lev

Kotlin | J

```
open class TextView : View, ViewTreeObserver.OnPreDrawListener
```

kotlin.Any
↳ android.view.View
↳ android.widget.TextView

Known Direct Subclasses
Button, CheckedTextView, Chronometer, DigitalClock, EditText, TextClock

Known Indirect Subclasses
AutoCompleteTextView, CheckBox, CompoundButton, ExtractEditText, MultiAutoCompleteTextView, RadioButton, Switch, ToggleButton

Basic Widgets

Text View:

```
<LinearLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="match_parent"
      android:layout_height="match_parent">
     <TextView
         android:id="@+id/text_view_id"
         android:layout_height="wrap_content"
         android:layout_width="wrap_content"
         android:text="@string/hello" />
  </LinearLayout>
```

This code sample demonstrates how to modify the contents of the text view defined in the previous XML layout:

```
public class MainActivity extends Activity {

     protected void onCreate(Bundle savedInstanceState) {
          super.onCreate(savedInstanceState);
          setContentView(R.layout.activity_main);
          final TextView helloTextView = (TextView) findViewById(R.id.text_view_id);
          helloTextView.setText(R.string.user_greeting);
      }
  }
```

Basic Widgets

Text View:

TextView

- **TextView** is like a label in android.
- TextView is typically used to display a caption.
- •TextViews are *not* editable, therefore they take no input.

Basic Widgets

Text View:

Example


```
<TextView
android:id="@+id/myTextView1"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:background="#ff0000ff"
android:padding="3px"
android:text="Enter User Name"
android:textSize="16sp"
android:textStyle="bold"
android:gravity="center"
</TextView>
```

Basic Widgets

Button:

Button

Added in A

Kotlin

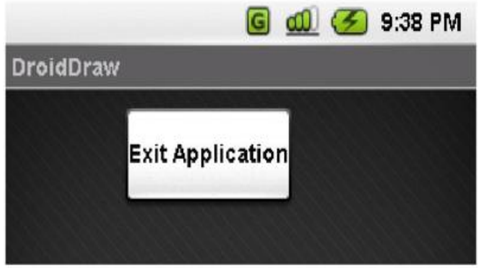
```
public class Button
extends TextView
```

java.lang.Object
↳ android.view.View
↳ android.widget.TextView
↳ android.widget.Button

Known direct subclasses
CompoundButton

Known indirect subclasses
CheckBox, RadioButton, Switch, ToggleButton

Buttons



- A **Button** widget allows thesimulation of a clicking action on a GUI.
- Button is a subclass of TextView. Therefore formatting a Button's face is similar to the setting of a TextView.

Basic Widgets

Button:

Button

Added in A

Kotlin

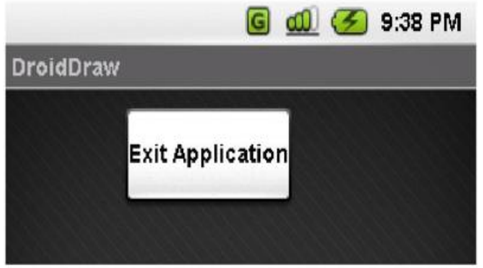
```
public class Button
extends TextView
```

java.lang.Object
↳ android.view.View
↳ android.widget.TextView
↳ android.widget.Button

Known direct subclasses
CompoundButton

Known indirect subclasses
CheckBox, RadioButton, Switch, ToggleButton

Buttons

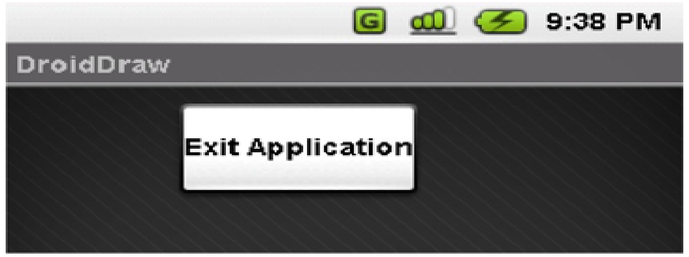


- A **Button** widget allows thesimulation of a clicking action on a GUI.
- Button is a subclass of TextView. Therefore formatting a Button's face is similar to the setting of a TextView.

Basic Widgets

Button:

```
...
<Button
android:id="@+id/btnExitApp"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:padding="10px"
android:layout_marginLeft="5px"
android:text="Exit Application"
android:textSize="16sp"
android:textStyle="bold"
android:gravity="center"
android:layout_gravity="center_horizontal"/>
</Button>
```



Basic Widgets

Image View and Image Button:

Images

- **ImageView** and **ImageButton** are two Android widgets that allow embedding of images in your applications.
- Both are *image-based widgets analogue to TextView and Button, respectively.*
- Each widget takes an android:src or android:background attribute (in an XML layout) to specify what picture to use.
- Pictures are usually reference a *drawable resource.*
- ImageButton, is a subclass of ImageView. It adds the standard *Button behavior for responding to click events.*

Basic Widgets

Image View and Image Button:

```
...
<ImageButton
android:id="@+id/myImageBtn1"
android:src="@drawable/icon"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
>
</ImageButton>
<ImageView
android:id="@+id/myImageView1"
android:src="@drawable/microsoft_sunset"
android:layout_width="150px"
android:layout_height="120px"
android:scaleType="fitXY"/>
</ImageView>
```

ImageButton

Added in API level 1

Kotlin | Java

```
public class ImageButton
extends ImageView
```

java.lang.Object
↳ android.view.View
↳ android.widget.ImageView
↳ android.widget.ImageButton

Known direct subclasses
ZoomButton

Displays a button with an image (instead of text) that can be pressed or clicked by the user. By default, an ImageButton looks like a regular `Button`, with the standard button background that changes color during different button states. The image on the surface of the button is defined either by the `android:src` attribute in the `<ImageButton>` XML element or by the `ImageView.setImageResource(int)` method.

Basic Widgets

Image View and Image Button:

ImageButtons has push states, where as a clickable image does not. You also can't call setText for ImageButton, you can with a regular button.

They all derive from view, but looking at the following extends chain may help a little.

```
java.lang.Object
↳ android.view.View
   ↳ android.widget.ImageView
       ↳ android.widget.ImageButton
```

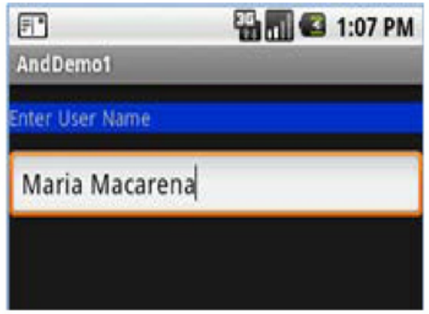
versus

```
java.lang.Object
↳ android.view.View
   ↳ android.widget.TextView
       ↳ android.widget.Button
```

Basic Widgets

Edit Text:

EditText



- The **EditText (or *textBox*)** *widget is an extension of* ***TextView*** *that allows updates.*
- The control configures itself to be *editable.*
- Important Java methods are: txtBox.setText("someValue") and txtBox.getText().toString()

In addition to the standard TextView properties EditText has many others features such as:

- android:autoText, (true/false) provides automatic spelling assistance
- android:capitalize, (words/sentences) *automatic capitalization*
- android:digits, *to configure the field to accept only certain digits*
- android:singleLine, *is the field for single-line / multiple-line input*
- android:password, *(true/false) controls field's visibility*
- android:numeric, *(integer, decimal, signed) controls numeric format*
- android:phonenumber, *(true/false) Formatting phone numbers*

Basic Widgets

Edit text:

EditText

Added in API level

Kotlin | Jav

```
public class EditText
extends TextView
```

java.lang.Object
↳ android.view.View
↳ android.widget.TextView
↳ android.widget.EditText

Known direct subclasses
AutoCompleteTextView, ExtractEditText

Known indirect subclasses
MultiAutoCompleteTextView

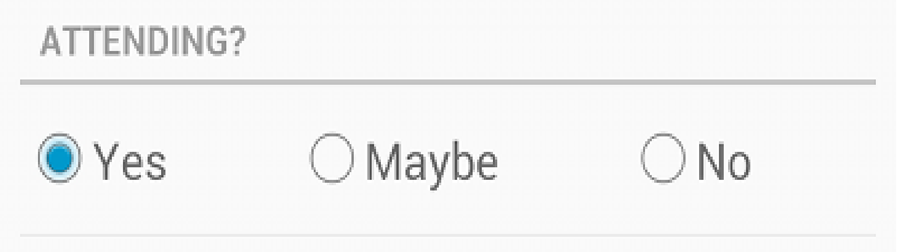
A user interface element for entering and modifying text. When you define an edit text widget, you must specify the `R.styleable.TextView_inputType` attribute. For example, for plain text input set inputType to "text":

Basic Widgets

Radio Button:

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 instead.



To create each radio button option, create a `RadioButton` in your layout. However, because radio buttons are mutually exclusive, you must group them together inside a `RadioGroup`. By grouping them together, the system ensures that only one radio button can be selected at a time.

Basic Widgets

Radio Button:

Key classes are the following:

- `RadioButton`
- `RadioGroup`

Responding to Click Events

When the user selects one of the radio buttons, the corresponding `RadioButton` object receives an on-click event.

To define the click event handler for a button, add the `android: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` hosting the layout must then implement the corresponding method.

Basic Widgets

Radio Button:

For example, here are a couple `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` is a subclass of `LinearLayout` that has a vertical orientation by default.

Basic Widgets

Radio Button:

Within the `Activity` that hosts this layout, the following method handles the click event for both radio buttons:

KOTLIN JAVA

```kotlin
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
                }
        }
    }
}
```

Basic Widgets

Radio Button:

Within the `Activity` that hosts this layout, the following method handles the click event for both radio buttons:

KOTLIN JAVA

```kotlin
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
                }
        }
    }
}
```

Basic Widgets

Radio Button:

RadioButtons

- A radio button is a two-states button that can be either *checked or unchecked.*
- When the radio button is unchecked, the user can press or click it to check it.
- Radio buttons are normally used together in a **RadioGroup.**
- When several radio buttons live inside a radio group, checking one radio button *unchecks all the others.*
- RadioButton inherits from ... TextView. Hence, all the standard TextView properties for *font face, style, color, etc. are available for controlling the look of radio buttons.*
- Similarly, you can call ***isChecked() on a RadioButton to see if it is selected, toggle() to select it, and so on, like you can with a CheckBox.***

Basic Widgets

Radio Group:

RadioGroup in Kotlin

Last Updated: 13-07-2020

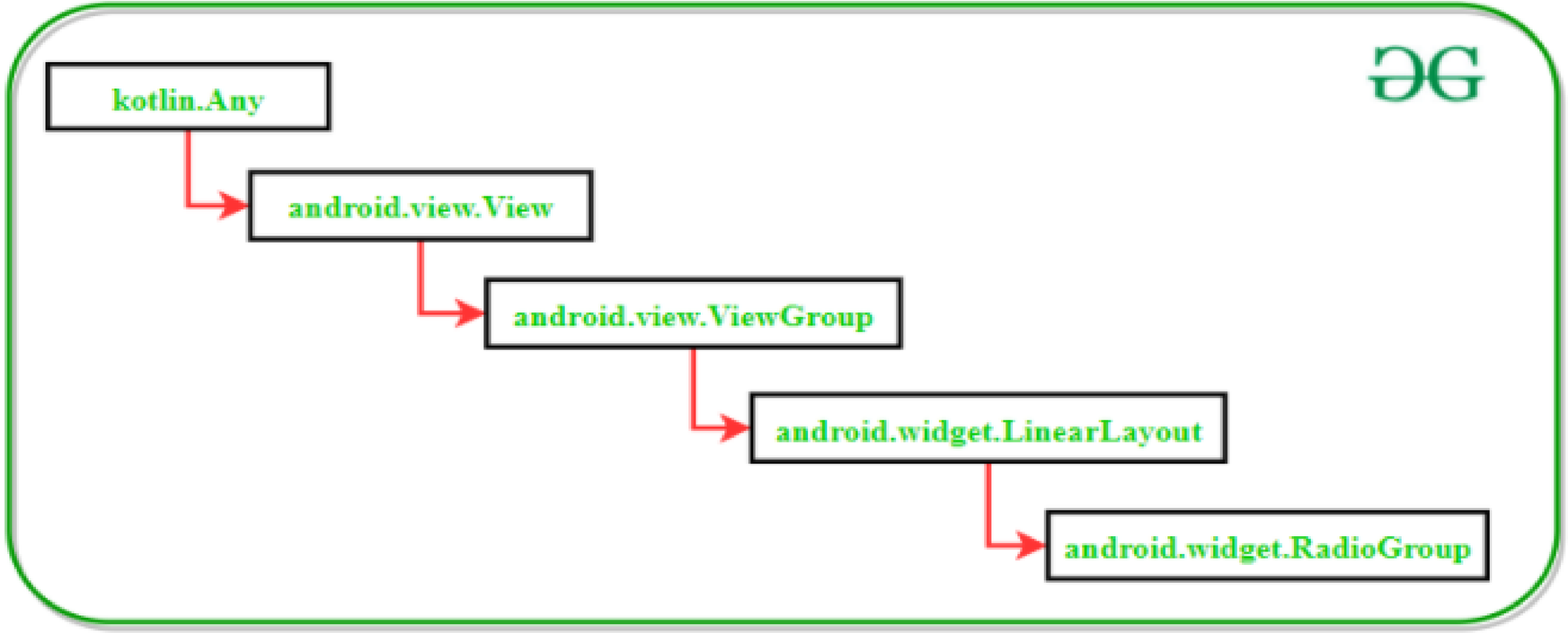
RadioGroup class of Kotlin programming language is used to create a container which holds multiple RadioButtons. The RadioGroup class is beneficial for placing a set of radio buttons inside it because this class adds **multiple-exclusion scope** feature to the radio buttons. This feature assures that the user will be able to check only one of the radio buttons among all which belongs to a RadioGroup class. If the user checks another radio button, the RadioGroup class unchecks the previously checked radio button. This feature is very important when the developer wants to have only one answer to be selected such as asking the gender of a user.

The class hierarchy of the RadioGroup class in Kotlin

Basic Widgets

Radio Group:

The class hierarchy of the RadioGroup class in Kotlin



Basic Widgets

Radio Group:

XML attributes of RadioGroup widget

XML ATTRIBUTES	DESCRIPTION
android:id	To uniquely identify the RadioGroup
android:background	To set a background colour
android:onClick	A method to perform certain action when RadioGroup is clicked
android:onClick	It's a name of the method to invoke when the radio button clicked.
android:visibility	Used to control the visibility i.e., visible, invisible or gone
android:layout_width	To set the width
android:layout_height	To set the height
android:contentDescription	To give a brief description of the view
android:checkedButton	Stores id of child radio button that needs to be checked by default within this radio group
android:orientation	To fix the orientation constant of the view

Basic Widgets

Radio Group:

```
<RadioGroup
    android:id="@+id/radio_group"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:background="#dbeceb"
    android:padding="15dp">
    <TextView
        android:id="@+id/title"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Which is your favorite color?"
        android:textStyle="bold"
        android:textSize="20sp"/>
    <RadioButton
        android:id="@+id/red"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="RED"
        android:onClick="radio_button_click"/>
    <RadioButton
        android:id="@+id/green"
        android:layout_width="wrap_content"
```

Basic Widgets

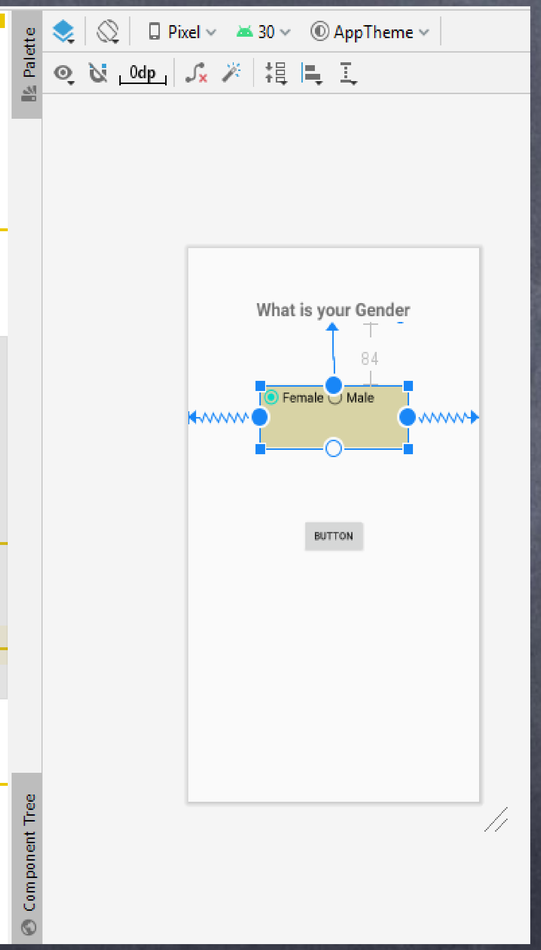
Radio Group:

```
// Get the selected radio button text using radio button on click listener
fun radio_button_click(view: View){
    // Get the clicked radio button instance
    val radio: RadioButton = findViewById(radio_group.checkedRadioButtonId)
    Toast.makeText(applicationContext, text: "On click : ${radio.text}",
        Toast.LENGTH_SHORT).show()
}
```

Basic Widgets

Radio Group:

```
<RadioGroup
    android:id="@+id/rg1"
    android:layout_width="209dp"
    android:layout_height="83dp"
    android:layout_marginTop="84dp"
    android:background="#D8D3A5"
    android:orientation="horizontal"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView">
    <RadioButton
        android:id="@+id/rb_female"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Female"
        android:textSize="18sp"
        android:checked="true"/>
    <RadioButton
        android:id="@+id/rb_male"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Male"
        android:textSize="18sp" />
</RadioGroup>
```



Basic Widgets

Radio Group:

```
class MainActivity2 : AppCompatActivity() {
    var radioGroup:RadioGroup?=null
    lateinit var radioButton: RadioButton
    private lateinit var button:Button
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main2)

         radioGroup=findViewById(R.id.rg1)
        button=findViewById(R.id.button3)
        button.setOnClickListener(View.OnClickListener { it: View!
            val selectedOption:Int=radioGroup!!.checkedRadioButtonId
            radioButton=findViewById(selectedOption)
            Toast.makeText( context: this,radioButton.text,Toast.LENGTH_LONG).show()
        })
    }
}
```

Basic Widgets

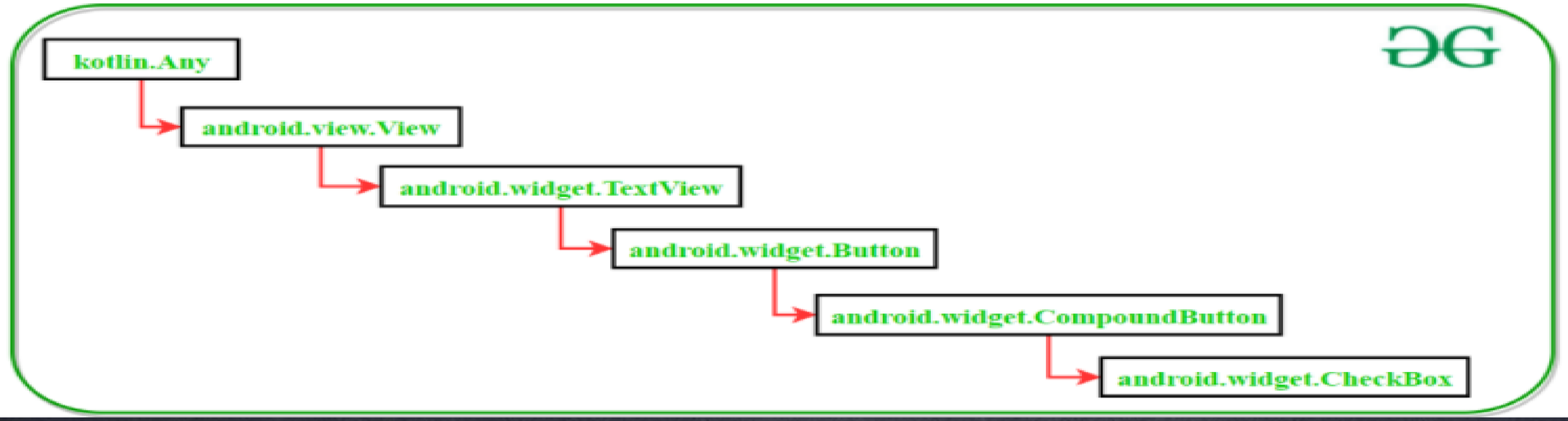
Check Box:

CheckBox in Kotlin

Last Updated: 13-07-2020

A CheckBox is a special kind of button in Android which has two states either checked or unchecked. The Checkbox is a very common widget to be used in Android and a very good example is the **"Remember me"** option in any kind of Login activity of an app which asks the user to activate or deactivate that service. There are many other uses of the CheckBox widget like offering a list of options to the user to choose from and the options are mutually exclusive i.e., the user can select more than one option. This feature of the CheckBox makes it a better option to be used in designing multiple-choice questions application or survey application in android.

Class hierarchy of CheckBox class in Kotlin



Basic Widgets

Check Box:

XML attributes of CheckBox widget

XML ATTRIBUTES	DESCRIPTION
android:id	Used to uniquely identify a CheckBox
android:checked	To set the default state of a CheckBox as checked or unchechek
android:background	To set the background color of a CheckBox
android:text	Used to store a text inside the CheckBox
android:fontFamily	To set the font of the text of the CheckBox
android:textSize	To set the CheckBox text size
android:layout_width	To set the CheckBox width
android:layout_height	To set the CheckBox height
android:gravity	Used to adjust the CheckBox text alignment
android:padding	Used to adjust the left, right, top and bottom padding of the CheckBox

Basic Widgets

Check Box:

Responding to Click Events

When the user selects a checkbox, the `CheckBox` object receives an on-click event.

To define the click event handler for a checkbox, add the `android: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` hosting the layout must then implement the corresponding method.

For example, here are a couple `CheckBox` objects in a list:

```xml
<?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"/>
```

Basic Widgets

Check Box:

KOTLIN JAVA

```kotlin
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
        }
    }
}
```

Basic Widgets

Chronometer:

Android **ChronoMeter** is user interface control which shows timer in the view. We can easily start up or down counter with base time using the chronometer widget. By default, **start()** method can assume base time and starts the counter.

Generally, we can create use **ChronoMeter** widget in XML layout but we can do it programmatically also.

Basic Widgets

Chronometer:

Chronometer

Added in API leve

Kotlin | Ja

```
public class Chronometer
extends TextView
```

java.lang.Object
↳ android.view.View
↳ android.widget.TextView
↳ android.widget.Chronometer

Class that implements a simple timer.

You can give it a start time in the `SystemClock#elapsedRealtime` timebase, and it counts up from that, or if you don't give it a base time, it will use the time at which you call `start()`.

The timer can also count downward towards the base time by setting `setCountDown(boolean)` to true.

By default it will display the current timer value in the form "MM:SS" or "H:MM:SS", or you can use `setFormat(String)` to format the timer value into an arbitrary string.

Basic Widgets

Chronometer:

Nested classes	
interface	Chronometer.OnChronometerTickListener A callback that notifies when the chronometer has incremented on its own.

XML attributes	
android:countDown	Specifies whether this Chronometer counts down or counts up from the base.
android:format	Format string: if specified, the Chronometer will display this string, with the first "%s" replaced by the current timer value in "MM:SS" or "H:MM:SS" form.

Inherited XML attributes
From class android.widget.TextView
From class android.view.View

Inherited constants
From class android.widget.TextView
From class android.view.View

Basic Widgets

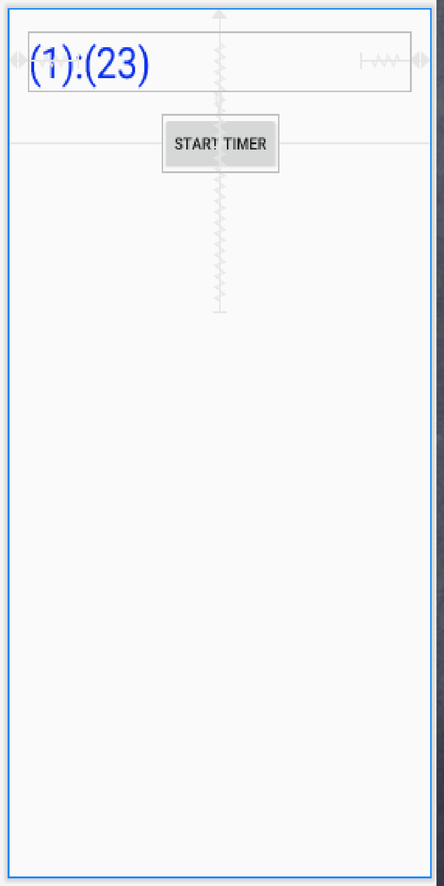
Chronometer:

XML ATTRIBUTES	DESCRIPTION
android:id	Used to specify the id of the view.
android:textAlignment	Used to the text alignment in the dropdown list.
android:background	Used to set the background of the view.
android:padding	Used to set the padding of the view.
android:visibilty	Used to set the visibility of the view.
android:gravity	Used to specify the gravity of the view like center, top, bottom etc
android:format	Used to define the format of the string to be displayed.
android:countDown	Used to define whether the chronometer will count up or count down.

Basic Widgets

Chronometer:

```
<Chronometer
    android:id="@+id/c_meter"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="20dp"
    android:layout_marginStart="68dp"
    android:layout_marginTop="256dp"
    android:layout_marginEnd="68dp"
    android:layout_marginBottom="28dp"
    android:textAppearance="@style/TextAppearance.AppCompat.Large"
    android:textColor="#092FEC"
    android:textSize="36sp"
    app:layout_constraintBottom_toTopOf="@+id/btn"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent" />
```



Basic Widgets

Chronometer:

```
btn?.setOnClickListener(object : View.OnClickListener {
    var isWorking = false
    override fun onClick(v: View) {
        if (!isWorking) {
            c_meter.start()
            isWorking = true
        } else {
            c_meter.stop()
            isWorking = false
        }
        btn.setText(if (isWorking) R.string.start else R.string.stop)
        Toast.makeText( context: this@Chronometer, getString(
            if (isWorking)
                R.string.working
            else
                R.string.stopped),
            Toast.LENGTH_SHORT).show()
    }
})
```

Basic Widgets

Progress Bar:

ProgressBar

Added in API level 1

Kotlin | Java

```
open class ProgressBar : View
```

kotlin.Any
↳ android.view.View
↳ android.widget.ProgressBar

Known Direct Subclasses
AbsSeekBar

Known Indirect Subclasses
RatingBar, SeekBar

A user interface element that indicates the progress of an operation. Progress bar supports two modes to represent progress: determinate, and indeterminate. For a visual overview of the difference between determinate and indeterminate progress modes, see Progress & activity. Display progress bars to a user in a non-interruptive way. Show the progress bar in your app's user interface or in a notification instead of within a dialog.

Basic Widgets

Progress Bar:

Indeterminate Progress

Use indeterminate mode for the progress bar when you do not know how long an operation will take. Indeterminate mode is the default for progress bar and shows a cyclic animation without a specific amount of progress indicated. The following example shows an indeterminate progress bar:

```
<ProgressBar
        android:id="@+id/indeterminateBar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        />
```

Basic Widgets

Progress Bar:

Determinate Progress

Use determinate mode for the progress bar when you want to show that a specific quantity of progress has occurred. For example, the percent remaining of a file being retrieved, the amount records in a batch written to database, or the percent remaining of an audio file that is playing.

To indicate determinate progress, you set the style of the progress bar to `android.R.style#Widget_ProgressBar_Horizontal` and set the amount of progress. The following example shows a determinate progress bar that is 25% complete:

```
<ProgressBar
        android:id="@+id/determinateBar"
        style="@android:style/Widget.ProgressBar.Horizontal"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:progress="25"/>
```

Basic Widgets

Progress Bar:

You can update the percentage of progress displayed by using the `setProgress(int)` method, or by calling `incrementProgressBy(int)` to increase the current progress completed by a specified amount. By default, the progress bar is full when the progress value reaches 100. You can adjust this default by setting the `android:max` attribute.

Other progress bar styles provided by the system include:

- `Widget.ProgressBar.Horizontal`
- `Widget.ProgressBar.Small`
- `Widget.ProgressBar.Large`
- `Widget.ProgressBar.Inverse`
- `Widget.ProgressBar.Small.Inverse`
- `Widget.ProgressBar.Large.Inverse`

The "inverse" styles provide an inverse color scheme for the spinner, which may be necessary if your application uses a light colored theme (a white background).

Basic Widgets

Progress Bar:

```
private var progressBar: ProgressBar? = null
private var i = 0
private var txtView: TextView? = null
private val handler = Handler()
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    progressBar = findViewById<ProgressBar>(R.id.progress_Bar) as ProgressBar
    txtView = findViewById(R.id.text_view) as TextView
    val btn = findViewById(R.id.show_button) as Button
    btn.setOnClickListener {
        i = progressBar!!.progress
        Thread(Runnable {
            while (i < 100) {
                i += 5
                // Update the progress bar and display the current value
                handler.post(Runnable {
                    progressBar!!.progress = i
                    txtView!!.text = i.toString() + "/" + progressBar!!.max
                })
                try {
                    Thread.sleep(100)
                } catch (e: InterruptedException) {
                    e.printStackTrace()
                }

            }
        }).start()
    }
}
```

Basic Widgets

Spinner :

Spinner

Added in API level 1

Kotlin | Java

```
open class Spinner : AbsSpinner, DialogInterface.OnClickListener
```

kotlin.Any
↳ android.view.View
↳ android.view.ViewGroup
↳ android.widget.AdapterView<android.widget.SpinnerAdapter>
↳ android.widget.AbsSpinner
↳ android.widget.Spinner

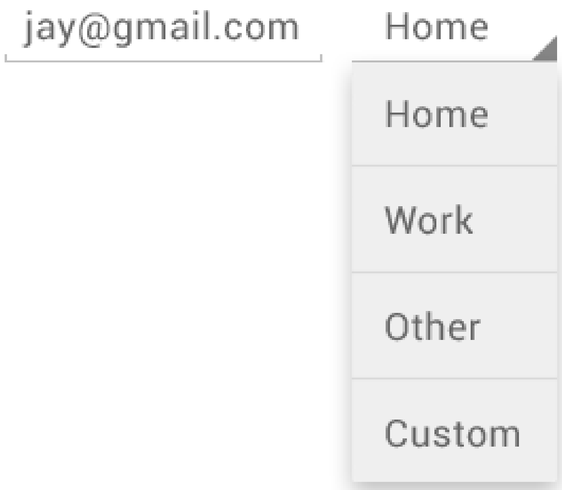
A view that displays one child at a time and lets the user pick among them. The items in the Spinner come from the `Adapter` associated with this view.

Basic Widgets

Spinner :

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.



Basic Widgets

Spinner :

You can add a spinner to your layout with the `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="match_parent"
    android:layout_height="wrap_content" />
```

To populate the spinner with a list of choices, you then need to specify a `SpinnerAdapter` in your `Activity` or `Fragment` source code.

Key classes are the following:

- `Spinner`
- `SpinnerAdapter`
- `AdapterView.OnItemSelectedListener`

Basic Widgets

Spinner :

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`, such as an `ArrayAdapter` if the choices are available in an array or a `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:

```xml
<?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>
```

Basic Widgets

Spinner :

With an array such as this one, you can use the following code in your `Activity` or `Fragment` to supply the spinner with the array using an instance of `ArrayAdapter`:

KOTLIN JAVA

```kotlin
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
}
```

Basic Widgets

Spinner :

Responding to User Selections

When the user selects an item from the drop-down, the `Spinner` object receives an on-item-selected event.

To define the selection event handler for a spinner, implement the `AdapterView.OnItemSelectedListener` interface and the corresponding `onItemSelected()` callback method. For example, here's an implementation of the interface in an `Activity`:

KOTLIN JAVA

```kotlin
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
    }
}
```

Basic Widgets

Spinner :

```
val languages = resources.getStringArray(R.array.Languages)

// access the spinner
val spinner = findViewById<Spinner>(R.id.spinner)
if (spinner != null) {
    val adapter = ArrayAdapter( context: this,
        android.R.layout.simple_spinner_item, languages)
    spinner.adapter = adapter

    spinner.onItemSelectedListener = object :
        AdapterView.OnItemSelectedListener {
        override fun onItemSelected(parent: AdapterView<*>,
                                    view: View, position: Int, id: Long) {
            Toast.makeText( context: this@Spinner1,
                text: getString(R.string.selected_item) + " " +
                        "" + languages[position], Toast.LENGTH_SHORT).show()
        }

        override fun onNothingSelected(parent: AdapterView<*>) {
            // write code to perform some action
        }
```