# CS4131
# Mobile Application Development

Name: ________________________________ (        ) Date: _______________

# Chapter 2: Android Layouts, Views and View Groups

## 2.1 Introduction to Layouts, View Groups and Views

Within an Android app Activity, a layout defines the structure for a user interface within the same Activity. All elements in the layout are built using a ***hierarchy of View and ViewGroup objects***. A **View object is a visual object containing elements which the user can see and interact with**, whereas a **ViewGroup object is a container that defines the layout structure** for multiple View or other ViewGroup objects, as you can see below:

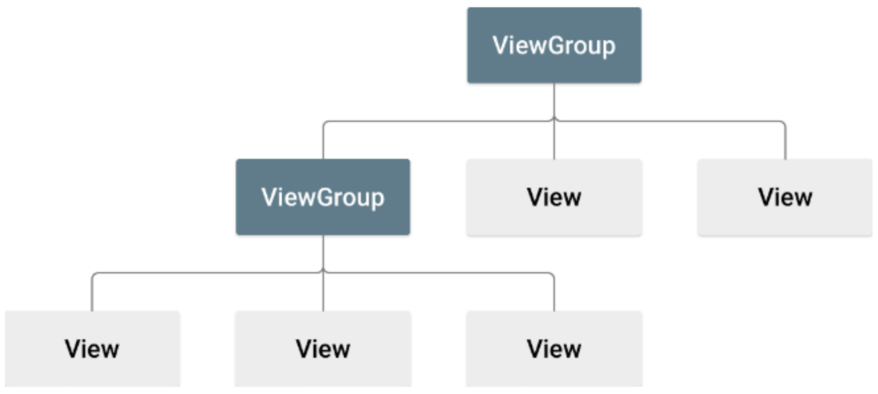


**Figure 1: Android View Hierarchy**

Hence, you can now define a hierarchy of ViewGroups and Views as a graph-like structure, or in the form of a component tree which you may see in Android Studio when editing your XML files in Design mode:

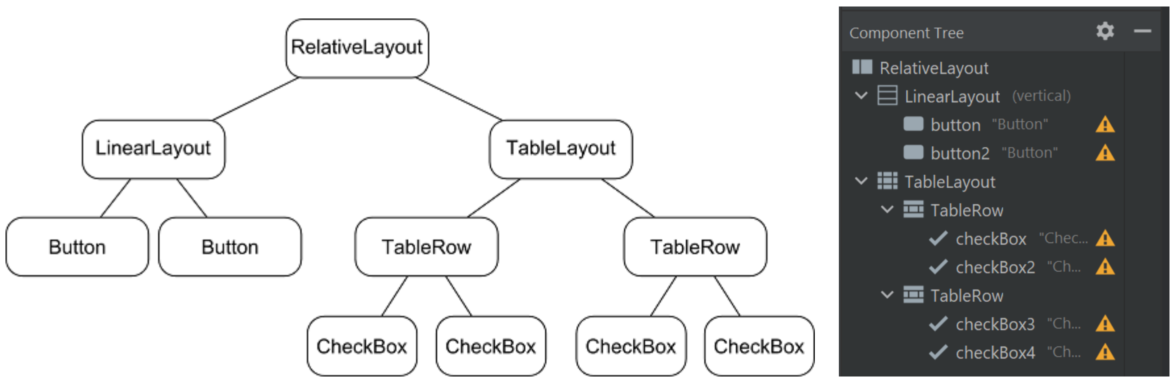
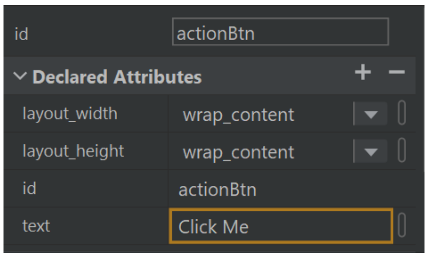


**Figure 2: Android View Hierarchy / Android Component Tree Example**

The subsequent sections will cover the different Android Layouts available and the more commonly used UI components (Views, Buttons, Widgets, etc) in Android apps.

## 2.2 View / ViewGroup Attributes in XML / Design Layout

Similar to JavaFX, each and every View or ViewGroup will have attributes which you may define to either edit some of its properties (like the textSize, layout attributes, etc). To edit the attributes, you will have to click on the individual View or View Group either in the component tree or the design layout window. When you insert a new View or ViewGroup, there will be a few standard attributes which you will immediately see under "Declared Attributes". There are other many other attributes you can actually "declare" and initialize under the "All Attributes" section, with the more common ones in the "Common Attributes" section, for each and every View and ViewGroup inserted.



| | |
|---|---|
| id | actionBtn |
| **Declared Attributes** | |
| layout_width | wrap_content |
| layout_height | wrap_content |
| id | actionBtn |
| text | Click Me |

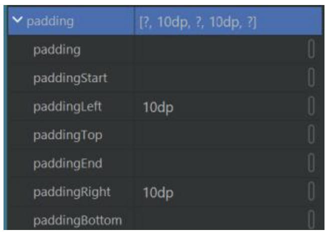
Some of the more important attributes you will have to take note of:

| **Attribute** | **Description** |
|---|---|
| *id* | The ID of the View / ViewGroup object. This same ID will also be the variable name of the View / ViewGroup object in your code. |
| *layout_width* | This attribute controls the width of the View / ViewGroup object.<br>There are 3 options, either:<br>• *wrap_content*:<br>• *match_parent*:<br>• Manual input of the pixels, which is in the units *dp* / *sp* (**density pixels / scalable pixels**), for example, 13sp.<br>  ○ Both the *sp* and *dp* is a virtual unit based on the physical density of the given screen. Hence, this means that on a higher resolution screen with a higher dpi (dots per inch), more pixels can be represented per 1dp or 1sp.<br>  ○ The *sp* measure is special in a way that it is adjustable depending on the user preference, for example, those who's eyesight is not very good may opt for font sizes in their phones to be bigger.<br>  ○ The *dp* measure is mainly for UI layout sizes while the sp measure is mainly for text-based UI components<br><br>The understanding of the dp/sp measure is important not just in the aspect of UI component sizing, but also in understanding why higher resolution screens are more sensitive to touch, and why developers use the same bitmap graphics of different resolutions in their resources (and hence why developers will prefer to use vector graphics instead.<br><br>You may want to read up on the following resource for a better understanding:<br>https://developer.android.com/training/multiscreen/screendensities |

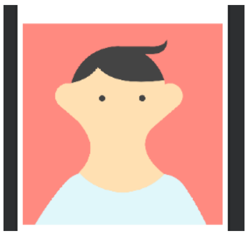
| | |
|---|---|
| *layout_height* | This attribute controls the height of the View / ViewGroup object.<br><br>The explanation for the options and the dp/sp measure will be the same as the above. |
| *text* | Some UI components require a text element such that the user using the UI component knows exactly what the components does intuitively, and it is good for such text to be informative yet concise which enhances the user's experience.<br><br>Note that it is recommended that such text is drawn from the string.xml file in the res folder through typing @string/... in the field. |

**Paddings**

Paddings provide some whitespace between different Views or ViewGroups which will be vital in improving the user experience such that each and every UI component appears o be distinct from each other. You can set the paddng by using the padding attribute:

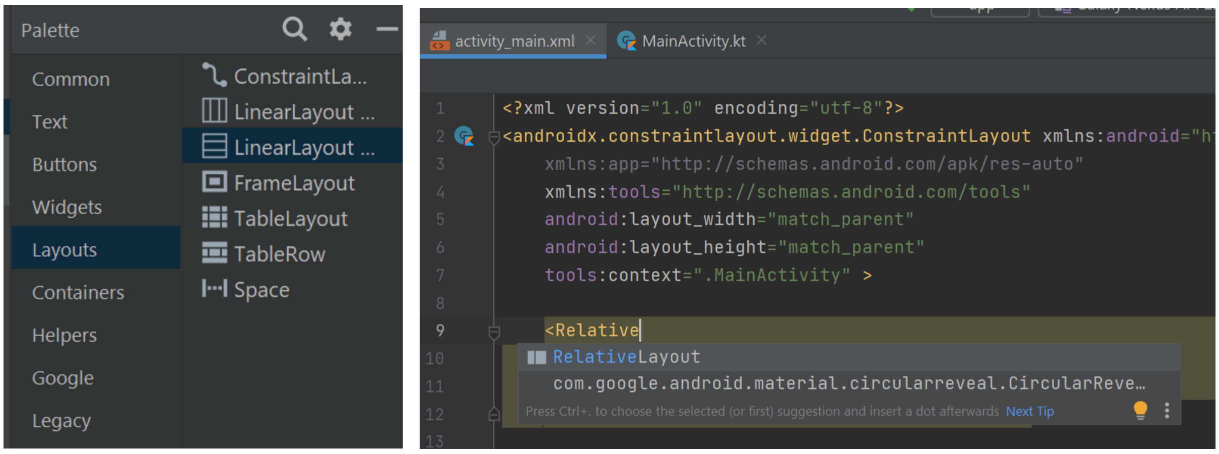


| padding | [?, 10dp, ?, 10dp, ?] |
|---|---|
| padding | |
| paddingStart | |
| paddingLeft | 10dp |
| paddingTop | |
| paddingEnd | |
| paddingRight | 10dp |
| paddingBottom | |



## 2.3 Layouts

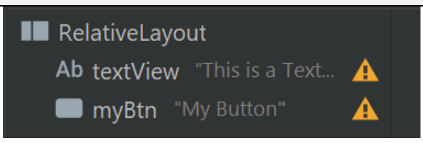
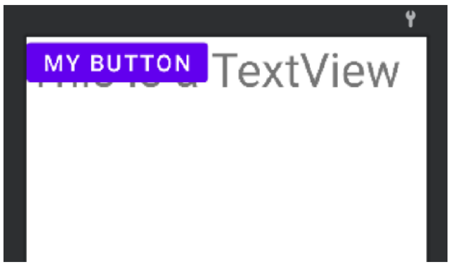
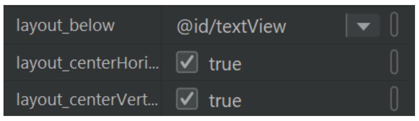
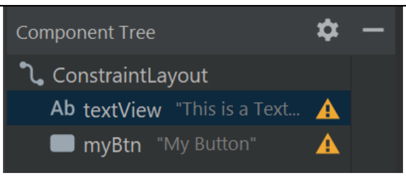
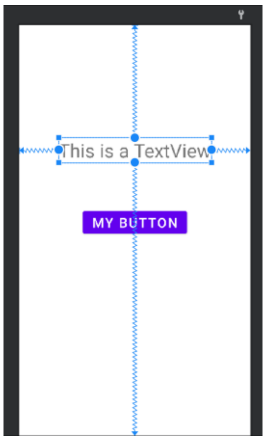
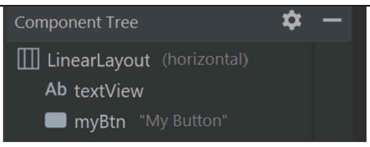
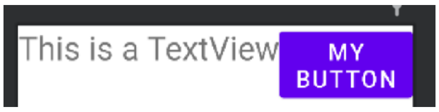
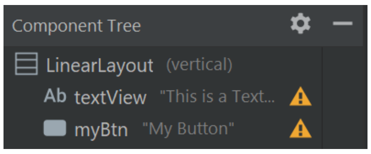
Android Studio provides the following layouts you may use as containers to store other ViewGroups and Views. Note that some of the Layouts cannot be seen in the Design Palette and will require a manual entry into the XML file, for example, in the below figure, the RelativeLayout cannot be found in the Design Palette, but if you edit the XML file directly to look for the RelativeLayout, you can find it as one of their suggestions:

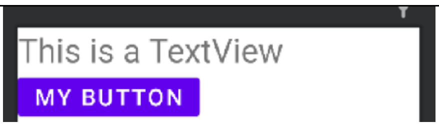
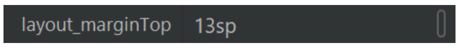
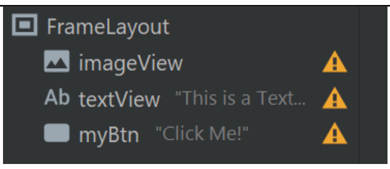
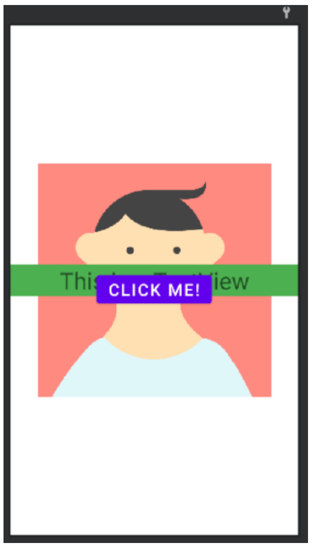
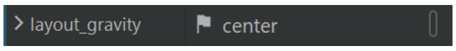
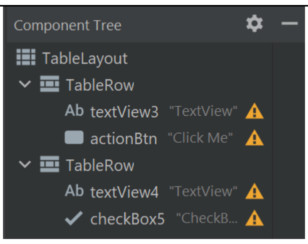
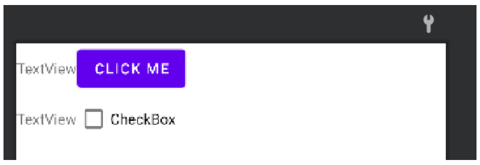


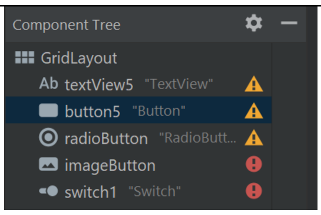
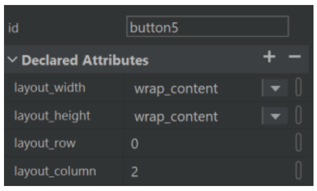
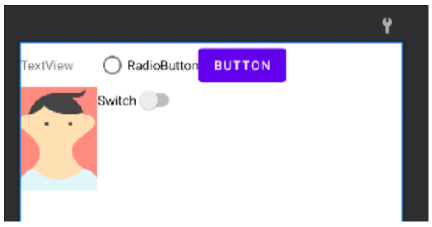
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout xmlns:android="h
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity" >

    <Relative
     RelativeLayout
     com.google.android.material.circularreveal.CircularReve…
Press Ctrl+. to choose the selected (or first) suggestion and insert a dot afterwards Next Tip
```

**Figure 3: Editing the XML file directly to look for layouts.**

The following table shows some of the different layouts available in Android Studio

| Layouts | Images from Android Studio | Description |
|---|---|---|
| RelativeLayout<br>*(Need to manually type into XML)* |  RelativeLayout<br>Ab textView "This is a Text...<br>myBtn "My Button"<br> | The RelativeLayout allows you to display UI components relative to each other. If the alignment is not defined, the UI components will be stacked in the order they are place in the hierarchy.<br><br>To enforce an alignment, you must edit the following attributes:<br><br>layout_below @id/textView<br>layout_centerHori... ✓ true<br>layout_centerVert... ✓ true<br>Explore the layout attributes and how it can affect the different UI components. |
| ConstraintLayout |  Component Tree<br>ConstraintLayout<br>Ab textView "This is a Text...<br>myBtn "My Button"<br> | ConstraintLayout allows for creation of large and complex layouts with a flat-view hierarchy. Hence, in a way it is similar to the RelativeLayout such that you have to define the alignment relationships with the different Views and ViewGroups but it is more intuitive to set in the the relative positions  in the Layout Editor of Android Studio and enforce alignment between UI components.<br><br>More on the ConstraintLayout will be explored in the next section. |
| LinearLayout<br>(Vertical / Horizontal) |  Component Tree<br>LinearLayout (horizontal)<br>Ab textView<br>myBtn "My Button"<br> <br> Component Tree<br>LinearLayout (vertical)<br>Ab textView "This is a Text...<br>myBtn "My Button" | The LinearLayout will layout the UI components within it either vertically or horizontally depending on the *orientation* chosen (just need to right-click the LinearLayout in the component tree to change the orientation).<br><br>To adjust the spacing between the UI components, you can adjust the respective layout margin fields for |

| | | |
|---|---|---|
| |  | each UI component, for example,  layout_marginTop 13sp |
| FrameLayout |  FrameLayout<br>imageView<br>textView "This is a Text...<br>myBtn "Click Me!"  | The FrameLayout is designed to block out an area on the screen to display a single item. Generally, it should be used to hold a single child view,as it can be difficult to organize child views scalable to different screen sizes without overlapping each other. You can, however, add multiple children to a FrameLayout and control their position by assigning gravity to each child, with:  layout_gravity center<br><br>Note that the UI components are stacked with the latest component being the “front-most”. |
| TableLayout |  Component Tree<br>TableLayout<br>TableRow<br>textView3 "TextView"<br>actionBtn "Click Me"<br>TableRow<br>textView4 "TextView"<br>checkBox5 "CheckB...  | The TableLayout contains a certain number of TableRow objects (which have to be inserted in by you). Each TableRow object define each row of the table, where each row can contain columns depending on the number of View objects put into each row. For the case on the left, there are 3 columns (2 columns with View objects and one empty column) and 2 rows. Note that the table borderlines will not be shown in a TableLayout.<br><br>You can also define each column to be either shrinkable (with respect to the content in the cell) and stretchable by using the setColumnShrinkable() and setColumnStretchable() methods |

| | | |
|---|---|---|
| GridLayout<br>*(Need to manually type into XML)* | <br>Component Tree<br>GridLayout<br>Ab textView5 "TextView"<br>button5 "Button"<br>radioButton "RadioButt...<br>imageButton<br>switch1 "Switch"<br><br>id button5<br>Declared Attributes<br>layout_width wrap_content<br>layout_height wrap_content<br>layout_row 0<br>layout_column 2<br><br> | The GridLayout places all the View children in a rectangular grid, where you are able to define the row and column index within the grid, such that grid[0][0] is defined to be on the top-left corner of the layout itself. Note that it is possible for you to define the View children to occupy the same cell or same group of cells. However, there is no guarantee that the View children themselves overlap after the layout operation completes. |
| CoordinatorLayout<br>*(Need to manually type into XML)* | The CoordinatorLayout is a super-powered FrameLayout where the main appeal is its ability to coordinate animations and transitions of the Views within it.<br><br>The CoordinatorLayout uses a Behaviour object, which represents interactions between 2 or more Views in the same layout, to determine how child View items should be laid out and moved as the user interacts with your app. The main behaviours include ***layout-based behaviours*** and ***scroll-based behaviours***.<br><br>The CoordinatorLayout is typically used in scenarios involving the interaction between the Free Action Bar (FAB) and other widgets such as the Snackbar, CardViews, RecyclerView, ScrollViews, Collapsing Toolbar, Appbar, all of which will be explored in the next chapter. | |

## 2.4 More on ConstraintLayout

The ConstraintLayout is probably one of the more interesting and commonly used layouts as the API and the Layout Editor on Android Studio were specifically built for each other, which will hence allow the building of the layout with ConstraintLayout entirely by dragging-and-dropping instead of editing the XML.

To define a view's position in ConstraintLayout, you must add at least one horizontal and one vertical constraint for the view. Each constraint represents a connection or alignment to another view, the parent layout, or an invisible guideline. Each constraint defines the view's position along either the vertical or horizontal axis.

When you drop a view into the Layout Editor, it stays where you leave it even if it has no constraints. However, this is only to make editing easier; if a view has no constraints when you run your layout on a device, it is drawn at position (0,0) (the top-left corner).

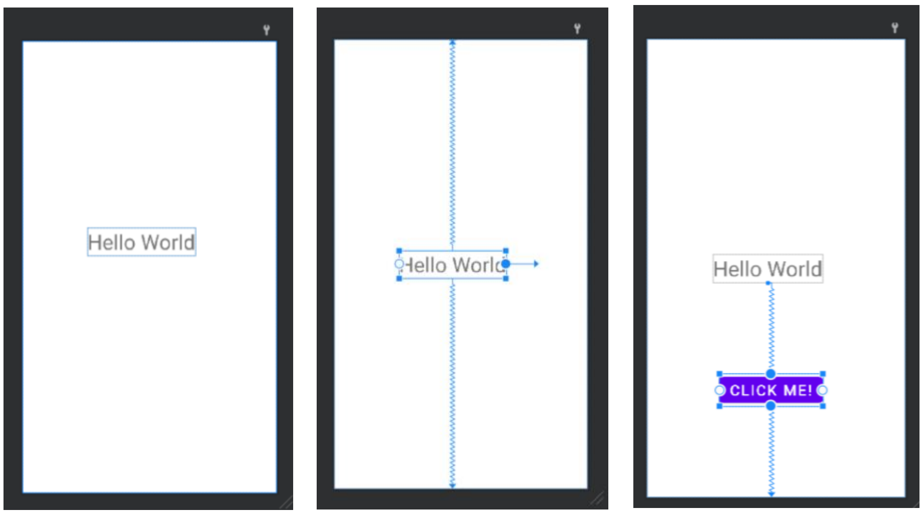
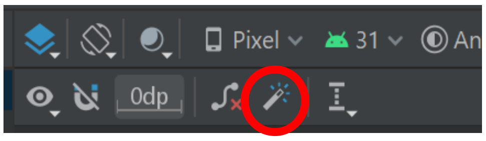
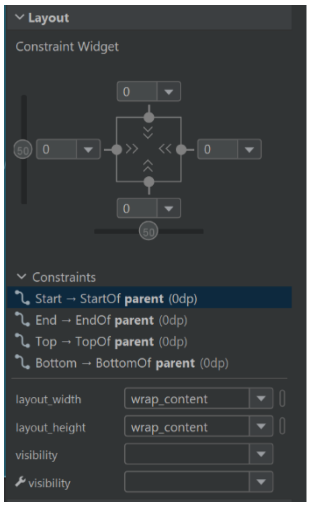


**Figure 4: Manipulating the Constraints**

In Figure 4, when the TextView is initially dragged into the Layout Editor, no **constraints (denoted by the "springs" you see above)** are inserted. To manipulate the constraints manually, you can click and drag the circular nodes on each view to either the edges of the layout they are confined within OR to other Views as you see on the rightmost diagram of Figure 4 where the Button is constrainted to the TextView.



If you wish to allow Android Studio, you can enable *Inference Mode*, denoted by the following button circled in the diagram on the right (Infer Constraints). Inference Mode uses a heuristic approach to automatically implement constraint connections after widgets have already been added to the layout.



Layout

Constraint Widget

Constraints
- Start → StartOf **parent** (0dp)
- End → EndOf **parent** (0dp)
- Top → TopOf **parent** (0dp)
- Bottom → BottomOf **parent** (0dp)

| | |
|---|---|
| layout_width | wrap_content |
| layout_height | wrap_content |
| visibility | |
| visibility | |

The ConstraintLayout attributes have special layout attributes on top of the standard few attributes any Layout will have. The circular nodes on the Constraint Widget defines the respective x and y positions as a percentage of the View within coordinates (0, 0) of the layout, which is defined to be on the top-left corner. What this means is that if both are set to 50, the View will be set to be exactly in the middle of the ConstraintLayout. You can also set a **bias** within the ConstraintLayout by setting the numbers (in the drop-down lists) on the four edges of the Constraint Widget, as seen below:

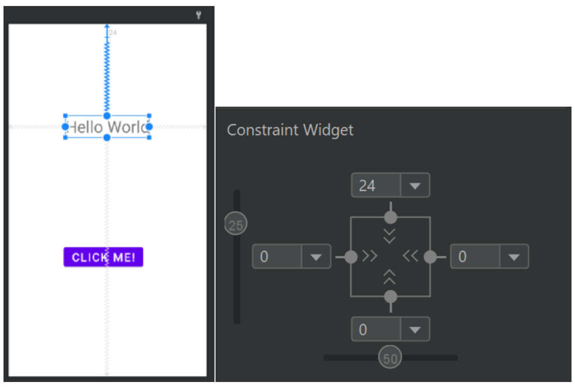
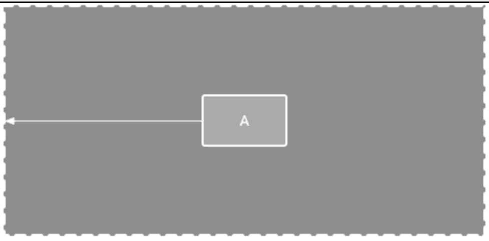
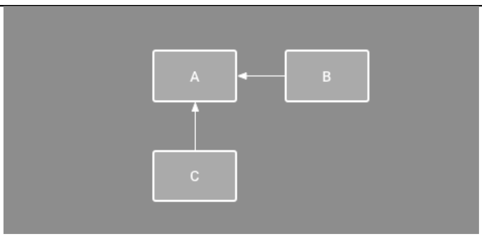
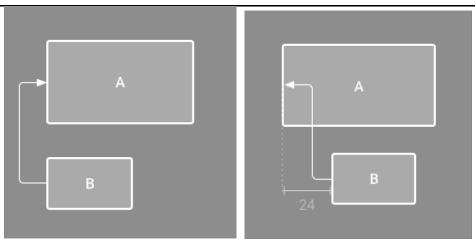
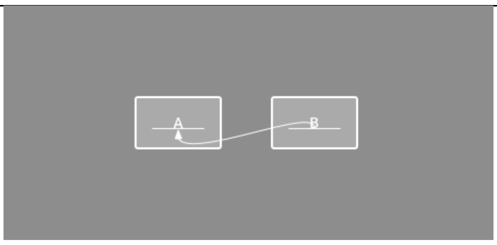
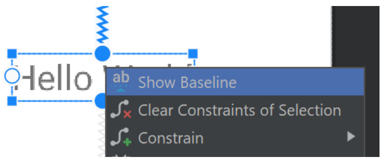
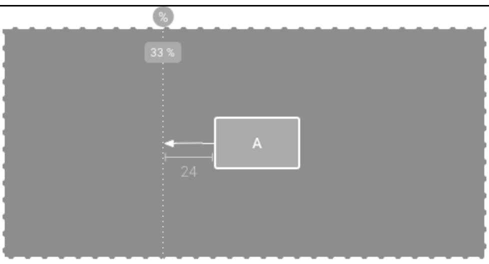
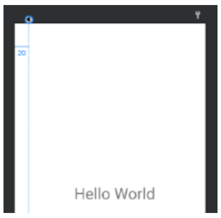
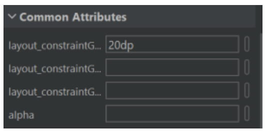
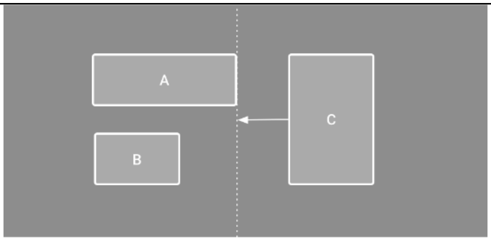
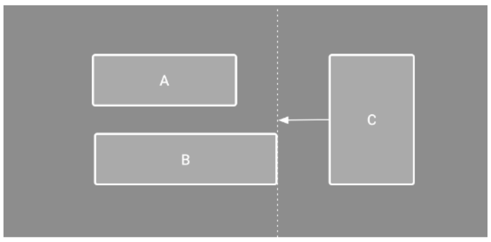
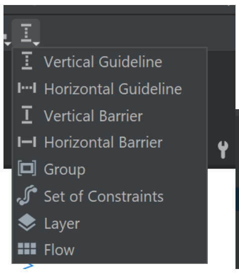
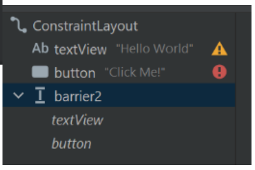
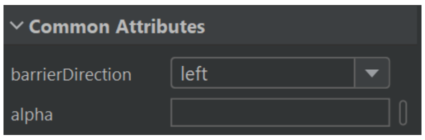


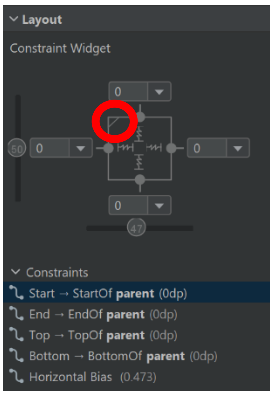
**Figure 5: Using the Constraint Widget to set Position of View and Bias**

Manipulating the constraints in the Views allows you to achieve different types of layout behaviour:

| **Layout Positions** | **Diagram** | **Description** |
|---|---|---|
| Parent Position |  | Constrain the side of a view to the corresponding edge of the layout. In the diagram, the left side of the view is connected to the left edge of the parent layout. You can define the distance from the edge with margin. |
| Order Position |  | Define the order of appearance for two Views (vertically or horizontally). In the diagram, B is constrained to always be right of A, and C below A. However, **these constraints do not imply alignment**, so B can still move up and down. |
| Alignment |  <br>**Left:** Horizontal alignment constraint<br>**Right:** Offset horizontal alignment constraint by 24dp | Align the edge of a View to the same edge of another View.<br>In the diagram, the left side of B is aligned to the left side of A. For central alignment, create a constraint on both sides.<br><br>You can offset the alignment by dragging the view inward from the constraint. In the diagram, B is set with a 24dp offset alignment. The offset is defined by the constrained View's margin.<br><br>You can also select all the views you want to align, and then click **Align** in the toolbar (same toolbar as where you find "Infer Constraints") |
| Baseline Alignment |  <br> Hello<br>Show Baseline<br>Clear Constraints of Selection<br>Constrain | Align the text baseline of a View to the text baseline of another View.<br><br>In the diagram, the first line of B is aligned with the text in A.<br><br>To create a baseline constraint, right-click the TextView you want to constrain and then click **Show Baseline**. Then click on the text baseline and drag the line to another baseline. |

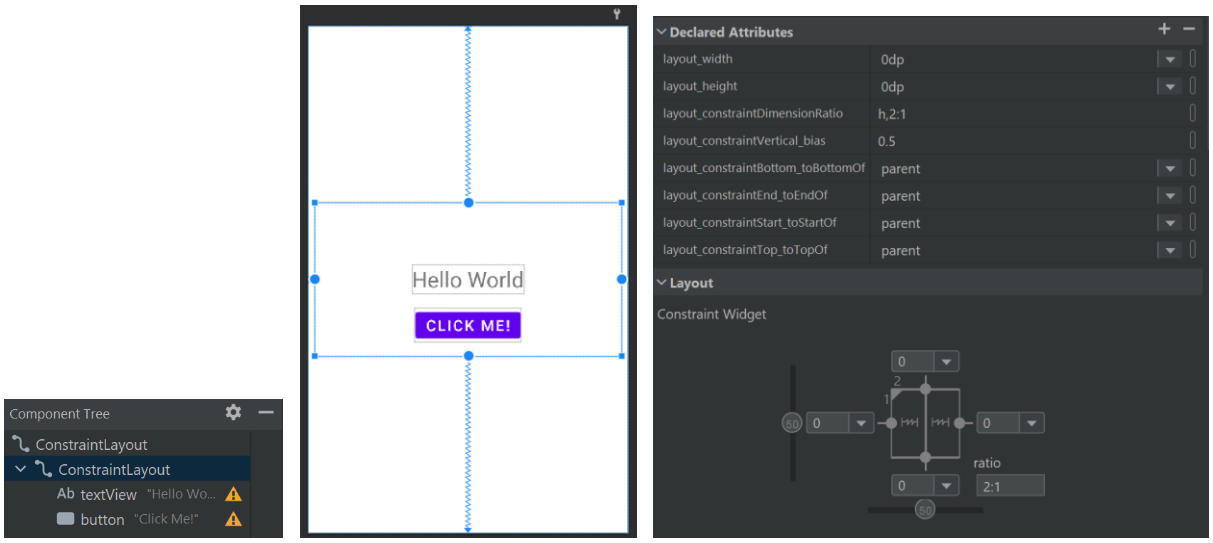
| | | |
|---|---|---|
| Constrain to a **Guideline** |    layout_constraintG... 20dp<br>layout_constraintG...<br>layout_constraintG...<br>alpha | Add guidelines to constrain Views. The guideline will be invisible to app users. You can position the guideline within the layout based on either dp units or percent, relative to the layout's edge.<br><br>To create a guideline, click **Guidelines** in the toolbar, and then click either **Add Vertical Guideline** or **Add Horizontal Guideline**. Drag the dotted line to reposition it and click the circle at the edge of the guideline to toggle the measurement mode. |
| Constrain to a **Barrier** |   View C is constrained to a barrier, which moves based on the relative position of itself to both View A and B.  Vertical Guideline<br>Horizontal Guideline<br>Vertical Barrier<br>Horizontal Barrier<br>Group<br>Set of Constraints<br>Layer<br>Flow  ConstraintLayout<br>Ab textView "Hello World"<br>button "Click Me!"<br>barrier2<br>textView<br>button  barrierDirection left<br>alpha | Like a guideline, a barrier is an invisible line that you can constrain Views to, except a barrier does not define its own position. The barrier position moves based on the position of Views contained within it. This is useful when you want to constrain a View to the a set of Views rather than to one specific View.<br><br>The diagram shows View C constrained to the right side of a barrier. The barrier is set to the "end" (right side in a left-to-right layout) of both View A and View B. The barrier moves depending on whether the right side of View A or View B is further right.<br><br>To create a barrier, follow these steps:<br>1. Click **Guidelines** in the toolbar, and then click **Add Vertical Barrier** or **Add Horizontal Barrier**.<br>2. Select the Views you want inside the barrier and drag them into the barrier component in the **Component Tree**<br>3. Select the barrier and set the barrierDirection attribute<br><br>You can also:<br>• Constrain Views inside the barrier to the barrier. This way, you can ensure all Views in the barrier align to each other<br>• Include guidelines inside barrier to ensure a "minimum" position for the barrier. |

## Working with Aspect Ratios


Layout
Constraint Widget
Constraints
Start → StartOf **parent** (0dp)
End → EndOf **parent** (0dp)
Top → TopOf **parent** (0dp)
Bottom → BottomOf **parent** (0dp)
Horizontal Bias (0.473)

ConstraintLayout ratios allow one dimension of a Layout to be sized relative to its other dimension. This concept of a ratio between dimensions is known as an **aspect ratio**. If the ConstraintLayout is **NOT the root layout**, you will notice that the Constraint Widget has a subtle difference on the top left corner of the widget diagram:

Clicking on the top-left corner triangle will allow you to edit the aspect ratio of the ConstraintLayout. Also. You may also notice there is also a "Horizontal / Vertical Bias" attribute within the layout constraints. This bias defines a "padding" between the root layout and the child ConstraintLayout. You may see an example below:



Component Tree
ConstraintLayout
ConstraintLayout
Ab textView "Hello Wo...
button "Click Me!"

Declared Attributes

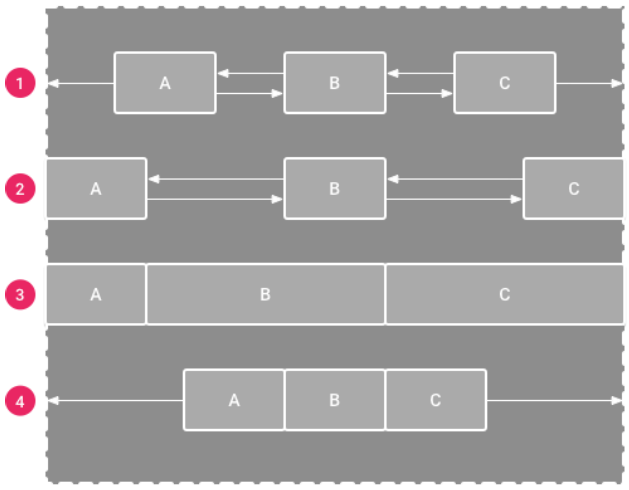
| | |
|---|---|
| layout_width | 0dp |
| layout_height | 0dp |
| layout_constraintDimensionRatio | h,2:1 |
| layout_constraintVertical_bias | 0.5 |
| layout_constraintBottom_toBottomOf | parent |
| layout_constraintEnd_toEndOf | parent |
| layout_constraintStart_toStartOf | parent |
| layout_constraintTop_toTopOf | parent |

Layout
Constraint Widget

**Figure 6: Setting the Aspect Ratio of a Child ConstraintLayout**

## Chaining Views

A chain is a group of Views that are linked to each other with bi-directional position constraints. The Views within a chain can be distributed either vertically or horizontally. The following shows the different chain styles that can be employed for multiple Views:



**1) Spread**: The Views are evenly distributed ***(default)***

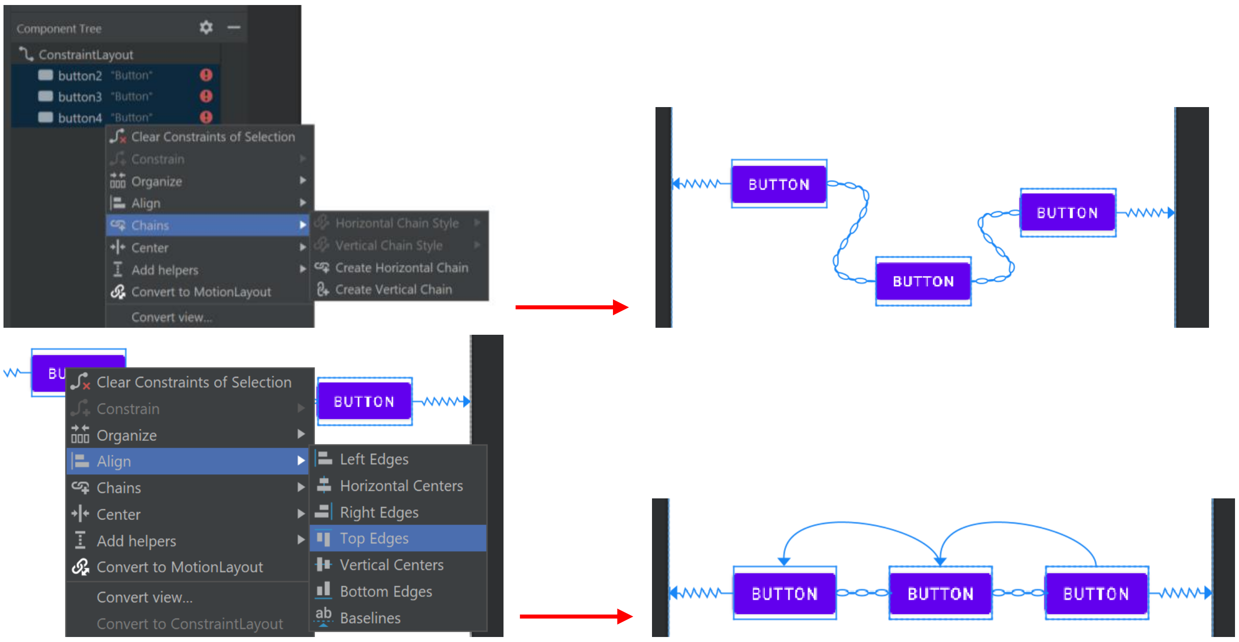
**2) Spread Inside**: The first and last view are affixed to the constraints on each end of the chain and the rest are evenly distributed

**3) Weighted**: You can assign a weight of importance to each view using the attributes:
layout_constraintHorizontal_weight
layout_constraintVertical_weight

The View with the highest weight gets the most amount of space and Views that have the same weight get the same amount of space.

**4) Packed**: The Views are packed together (after accounting for margins). You can then also adjust the whole chain's bias (left/right or up/down) by changing the chain's head view bias, where the head view is the left-most View in a horizontal chain or the top-most View in a vertical chain.

To create a chain, you can select all the Views to be included in the chain, right-click, select **Chains**, then select either **Center Horizontally** or **Center Vertically**. Note that chaining does not align the Views, hence, you need to align the Views which are chained after chaining them.



**It is strongly encouraged that you also study the changes in the XML code as you explore the editing the layouts in the Layout Editor so that you can see how the individual layout attributes are defined and their respective attribute names.**

## 2.5 Views, Controls and Buttons

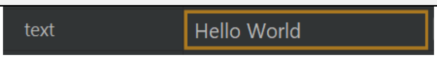
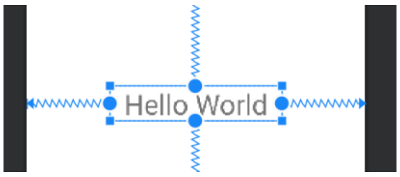
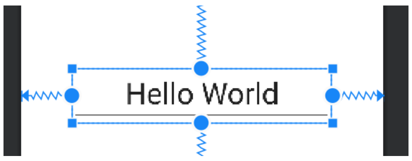
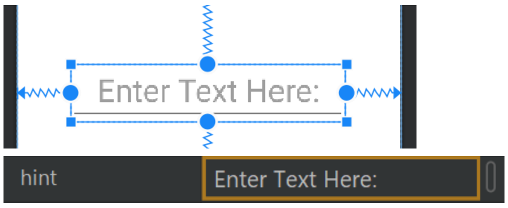
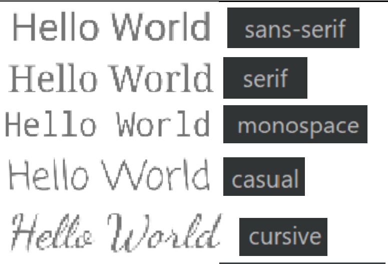
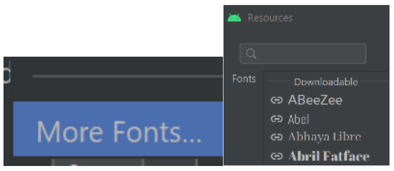
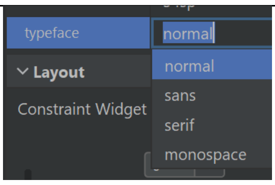
Just like JavaFX you have learnt, Android has a wide variety of UI elements you can use for user interaction with your app. As you may have realized by now, the UI controls as we know in JavaFX are known as Views in Android. Android also classifies the different UI elements as Views, Widgets and Buttons. This set of notes will cover the more commonly used UI elements.

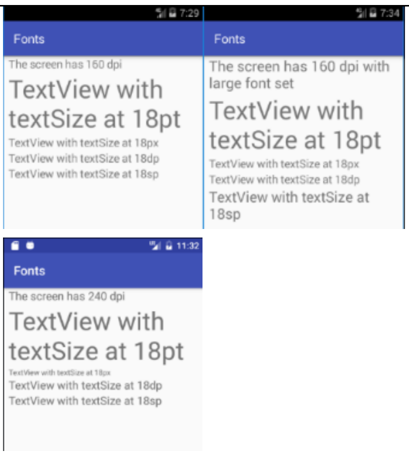
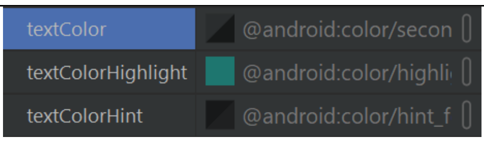
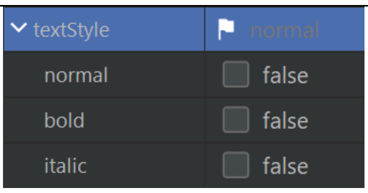
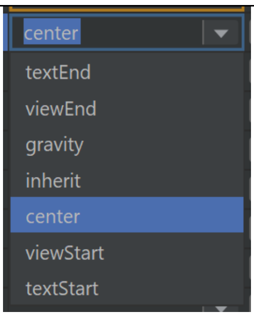
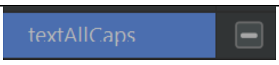
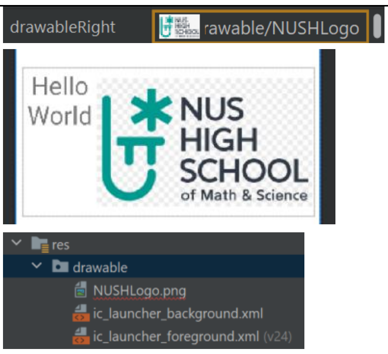
### <u>TextView / EditText</u>

The TextView displays text to the user and optionally allows the user to edit text. When it's a TextView, its only purpose is to display text, or images as well. When it is set to **EditText, which is a sub-class of TextView**, the user can edit text as well.

The TextView allows for displaying of both text (recall that it is recommended to use the *sp* unit of measure for text) and images (which are placed in the *drawable* folder inside the res folder).

The following describes how to work with some common attributes in the TextView / EditText:

| Attribute | Images | Description |
|---|---|---|
| *text* |  text: Hello World   To distinguish between EditText and TextView, EditText has a “line” underneath the said text. | When *text* attribute is edited in the TextView object, the text entered will display what is entered. As a good practice, it is preferred to refer to strings defined in strings.xml in the res folder. When it is the EditText object instead, editing the *text* attribute will display a “default” text in the editable text field. To give some kind of “hint” for the user to enter, you may edit the *hint* attribute:  hint: Enter Text Here: |
| *fontFamily* |   To include custom fonts which may be more familiar to you, like Arial or Times New Roman or Calibri, you may select “More Fonts…” and download fonts you may need. | The *fontFamily* attribute allows you to decide on the font face of the text to display. As a front-end designer, it is important to **know the difference between sans-serif and serif text**. The main difference is serif text have a small line attached to the larger end of the character stroke. An example of serif text will be Times New Roman. An example of sans-serif text will be Arial. You may realise that many apps today utilize a **sans-serif font face**, with the reason being sans-serif text **improve readability, in particular, for those who are dyslexic**, which is important in the aspect of inclusivity for the user experience. |
| *typeFace* |  typeface: normal | The *typeFace* attribute works in nearly the same way as the *fontFamily* attribute but with less options. |

| | | |
|---|---|---|
| *textSize* |  Fonts<br>The screen has 160 dpi<br>TextView with textSize at 18pt<br>TextView with textSize at 18px<br>TextView with textSize at 18dp<br>TextView with textSize at 18sp<br>Fonts<br>The screen has 160 dpi with large font set<br>TextView with textSize at 18pt<br>TextView with textSize at 18px<br>TextView with textSize at 18dp<br>TextView with textSize at 18sp<br>Fonts<br>The screen has 240 dpi<br>TextView with textSize at 18pt<br>TextView with textSize at 18px<br>TextView with textSize at 18dp<br>TextView with textSize at 18sp | The *textSize* attribute allows you to adjust the size of your text. As mentioned in the earlier discussion, it is recommended to use the "sp" units for text as it is adjustable depending on the user preference |
| *textColor* |  textColor @android:color/secon<br>textColorHighlight @android:color/highli<br>textColorHint @android:color/hint_f | The *textColor* attribute sets the colour of the text. Note that the text colour will either follow predefined android colours, tagged as *android:color*, or colours defined in the colors.xml file. All colours should be preceded by the @ operator |
| *textStyle* |  textStyle normal<br>normal false<br>bold false<br>italic false | The *textStyle* attribute sets the text to either normal, bold or italic. |
| *textAlignment* |  center<br>textEnd<br>viewEnd<br>gravity<br>inherit<br>center<br>viewStart<br>textStart | The *textAlignment* attribute sets the alignment of the text. viewStart, center, viewEnd sets the alignment with respect to the TextView object itself. You may notice there is a gravity option. A layout's gravity essentially sets the alignment such that if the layout's gravity is set to align left, all object within the layout set to a gravity alignment will follow the parent layout's gravity. |
| *textAllCaps* |  textAllCaps | The *textAllCaps* attribute will set the given text to all capital letters. Clicking on the ⊟ button will toggle between null, true, or false. The text will be set to all caps if it is set to true. |
| *drawable…* |  drawableRight rawable/NUSHLogo<br>Hello World<br>res<br>drawable<br>NUSHLogo.png<br>ic_launcher_background.xml<br>ic_launcher_foreground.xml (v24) | Note that the TextView is able to support **simple** image viewing. You have to insert your images into the drawable folder (under res) and utilize the respective drawable attribute to set the position of the image. |

The TextView class also has multiple subclasses used for various purposes other than EditText, for example, the Password class, Phone class, Email class, Date class, Number class and AutoCompleteTextView class. You may want to explore the features of each of the above-mentioned classes for their use.

Note that to obtain the String from the EditText (with variable name *textField*), or any other editable TextView object, you will have to use, for example:

```
val inputText = (findViewById<View>(R.id.textField) as EditText).text.toString()
```

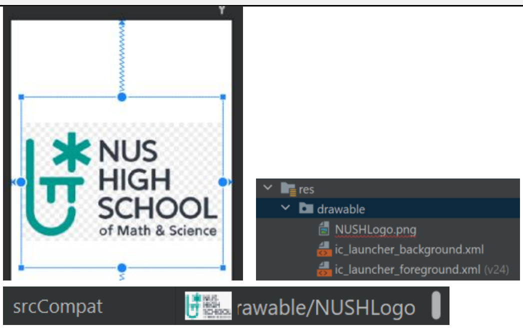
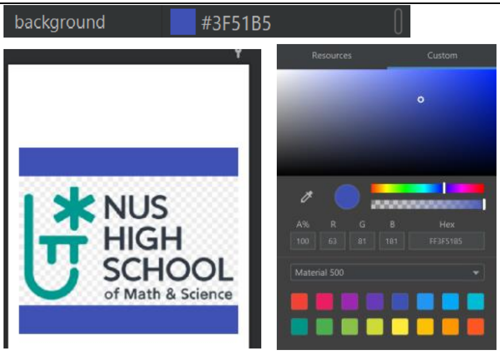
To change the text within the TextView (with variable name *resultView*) to some String *str* or some predefined string in the string.xml file,

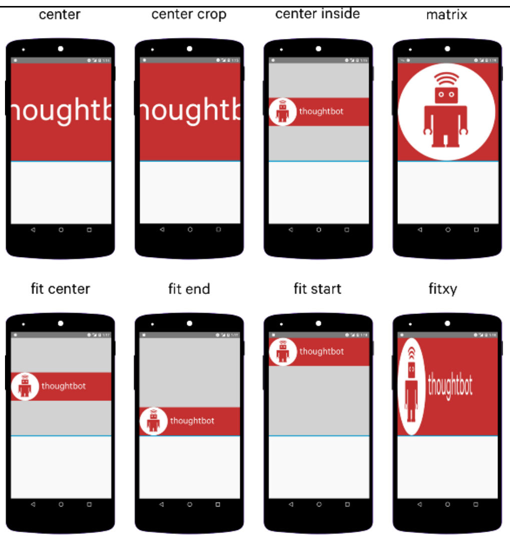
`resultView.text = str` OR
`resultView.text = getString(R.string.someResourceString)`

**<u>ImageView</u>**

The ImageView displays images to the user. The images are pulled from the *drawable* folder. While TextView can also display images, the ImageView can be used to do some minor image editing like applying tints to an image, image scaling, and also handling Bitmaps.

The following describes how to work with some common attributes in the ImageView:

| **Attribute** | **Images** | **Description** |
|---|---|---|
| *srcCompat* |  res<br>drawable<br>NUSHLogo.png<br>ic_launcher_background.xml<br>ic_launcher_foreground.xml (v24)<br>srcCompat rawable/NUSHLogo | The *srcCompat* attribute defines the source of the image. Note that all raw images need to be placed in the drawable folder (under res). |
| *background* |  background #3F51B5<br> Resources<br>Custom<br>A% R G B Hex<br>100 63 81 181 FF3F51B5<br>Material 500 | The *background* attribute defines any background colour you may want to define in the whitespaces of your ImageView.<br><br>You **do not** need to play with the hexadecimal codes to get the colour you need. Double clicking on the field will allow you to enter the colour palette to visualize and select your colours. Note that you can easily select predefined colours from your colors.xml file in your res folder by selecting the "Resources" tab. |

| | | |
|---|---|---|
| *scaleType* |  center<br>center crop<br>center inside<br>matrix<br>fit center<br>fit end<br>fit start<br>fitxy<br>thoughtbot | The *scaleType* attribute allows you to perform scaling of your image with respect to the size of your ImageView. You may see the left on the different types of scaling you may employ for the ImageView object.<br><br>Note that matrix will scale the image with a supplied Matrix class, and set using the setImageMatrix method. With the Matrix class, you can perform linear transformations like rotations to your image (you will need knowledge on Linear Algebra to do so)<br><br>Note that fitXY allows you to set the exact size of the image in your layout, however note that it does not maintain the aspect ratio. |

As there are many different screen sizes, resolutions and densities, many developers will place **<u>bitmap images</u> of multiple resolutions** for consistent good graphical qualities regardless of the pixel density of each device. If such is not done, what will happen is that Android will scale your image such that it occupies the same visible space on each screen, resulting in blurring.

Each density size corresponds to a different density qualifier as seen in the table below:

| **Density Qualifier** | **Description** | **Scale Factor** | **Pixels** |
|---|---|---|---|
| *ldpi* | low-density (*ldpi*) screens (~120dpi) | 0.75x | 1dp = 0.75px |
| *mdpi* | medium-density (*mdpi*) screens (~160dpi). (This is the baseline density) | 1.0x | 1dp = 1px |
| *hdpi* | high-density (*hdpi*) screens (~240dpi) | 1.5x | 1dp = 1.5px |
| *xhdpi* | extra-high-density (*xhdpi*) screens (~320dpi) | 2.0x | 1dp = 2px |
| *xxhdpi* | extra-extra-high-density (*xxhdpi*) screens (~480dpi) | 3.0x | 1dp = 3px |
| *xxxhdpi* | extra-extra-extra-high-density (*xxxhdpi*) uses (~640dpi) | 4.0x | 1dp = 4px |
| *nodpi* | All densities. These are density-independent resources. The system does not scale resources tagged with this qualifier, regardless of the current screen's density | NIL | NIL |
| *tvdpi* | Somewhere between *mdpi* and *hdpi* (~213dpi). This is not considered a "primary" density group. It | 1.33x | 1dp = 1.33px |

| | | | |
|---|---|---|---|
| | is mostly intended for televisions and most apps shouldn't need it—providing mdpi and hdpi resources is sufficient for most apps | | |

To create alternative bitmap drawables for different densities, you should follow the **3:4:6:8:12:16** scaling ratio between the six primary densities. For example, if you have a bitmap drawable that's 100x100 pixels for *mdpi,* on hdpi, you should have the same bitmap drawable of 150x150 pixels. A relative comparison is shown below:

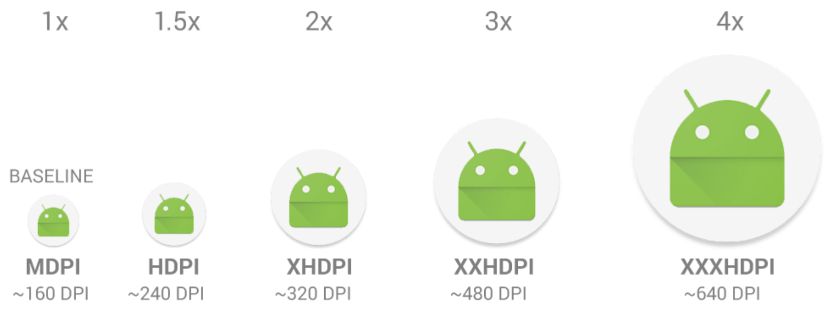
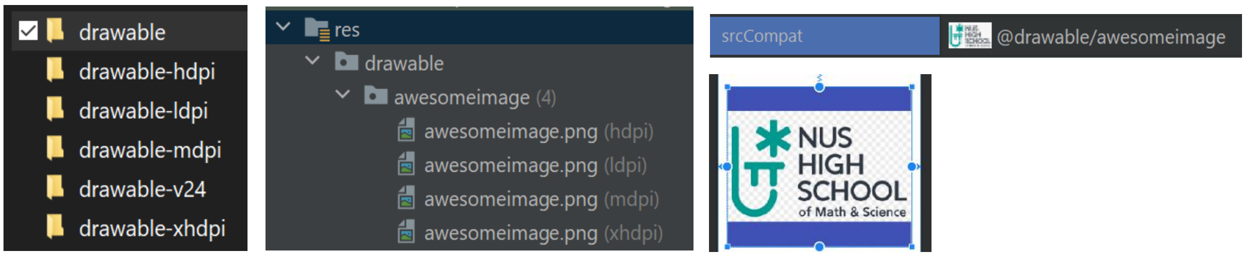


**Figure 7: Relative sizes for bitmaps at different density sizes**

The generated image files should then be placed in the appropriate subdirectory under res folder and the system will pick the correct one automatically based on the pixel density of the device your app is running on. For example, if you have copies of an image "*awesomeimage.png*" set for multiple densities, you will have to create multiple drawable folders, each containing a reference to the density qualifiers. You will then have to put the **image of the same name** into each of the folders. Now, Android Studio has a category of "*awesomeimage.png*" with the different density qualifiers and you may just reference awesomeimage.png, and the system will automatically set the corresponding image to the app's screen size in the artifact.



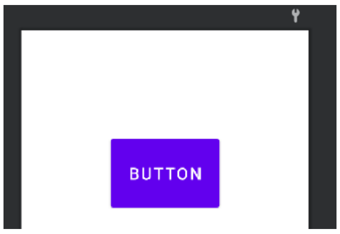
srcCompat @drawable/awesomeimage

**Figure 8: Setting Bitmaps of Different Qualifiers in Android Studio**

An alternative which most app designers go for instead of having multiple density-specific versions of an image is to create a **vector graphic**. Vector graphics create an image using XML to define paths and colors, instead of using pixel bitmaps. As such, vector graphics can scale to any size without scaling artifacts, though **they are usually best for illustrations such as icons, not photographs.**

Android uses their own "vector drawable" format for vector graphics instead of the more well known SVG or PSD formats. To perform the conversion from your existing, you will have to use the **Vector Asset Studio** within Android Studio. You may find out more in the link below:
https://developer.android.com/studio/write/vector-asset-studio

**<u>Button / ImageButton</u>**



The Button class is a subclass of the TextView. Other than the attributes inherited from TextView, the Button class has some other attributes like the *style*, *stateListAnimator* and *onClick*. Only the *onClick* attribute will be emphasized upon in the next section on Android Event Listeners while you explore the other attributes on your own.

Note that:

- ImageButton is a sub-class of the ImageView and will have both the features inherited from ImageView as well as Button's own unique attributes as mentioned above.
- Button / ImageButton objects follow the colour scheme specified in the values/themes xml files within the res folder, meaning that changing the background or text colour attributes will NOT affect how they look, and you will have to edit the themes.xml files directly.

## 2.6 Android Event Listeners and Framework

Events in Android can take a variety of different forms but are usually generated in response to an external action. The most common form of events, particularly for smart devices (handphones, tablets, etc), involve some form of interaction with the touch screen. Such events fall into the category of input events.

The Android framework maintains an event queue into which events are placed as they occur. Events are then removed from the stack on a first-in-first-out (FIFO) basis. In the case of an input event such as a touch on the screen, the event is passed to the View positioned at the location on the screen where the touch took place. A range of information (depending on the event type) about the nature of the event, such as the coordinates of the point of contact between the user's fingertip and the screen are also passed.

To handle the event that it has been passed, the View object must have in place an event listener. The Android View class contains a range of event listener interfaces, each of which contains an abstract declaration for a callback method. To respond to an event of a particular type, a view must register the appropriate event listener and implementing the corresponding callback. The event listeners available in the Android Framework and the associated callback method are in the table below:

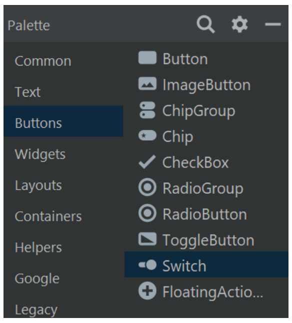
| **Listener** | **Description** |
|---|---|
| *onClick* | Called when the user either touches the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses the suitable "enter" key or presses down on the trackball. |
| *onLongClick* | Called when the user either touches and holds the item (when in touch mode), or focuses upon the item with the navigation-keys or trackball and presses and holds the suitable "enter" key or presses and holds down on the trackball (for one second). |
| *onTouch* | Called when the user performs an action qualified as a touch event, including a press, a release, or any movement gesture on the screen (within the bounds of the item). |

| | |
|---|---|
| *onCreateContextMenu* | Called when a Context Menu is being built (as the result of a sustained "long click"). See the discussion on context menus in the Menus developer guide. |
| *onFocusChange* | Detects when focus moves from the current view as the result of interaction with a track-ball or navigation key. |
| *onKey* | Called when the user is focused on the item and presses or releases a hardware key on the device. |

Note that there are many more listeners available in the View class and is not limited to what is presented. The next section will cover examples which primarily cover two key elements:

- The other Views available for use in your apps
- How to code the functionalities through associated methods or event listeners

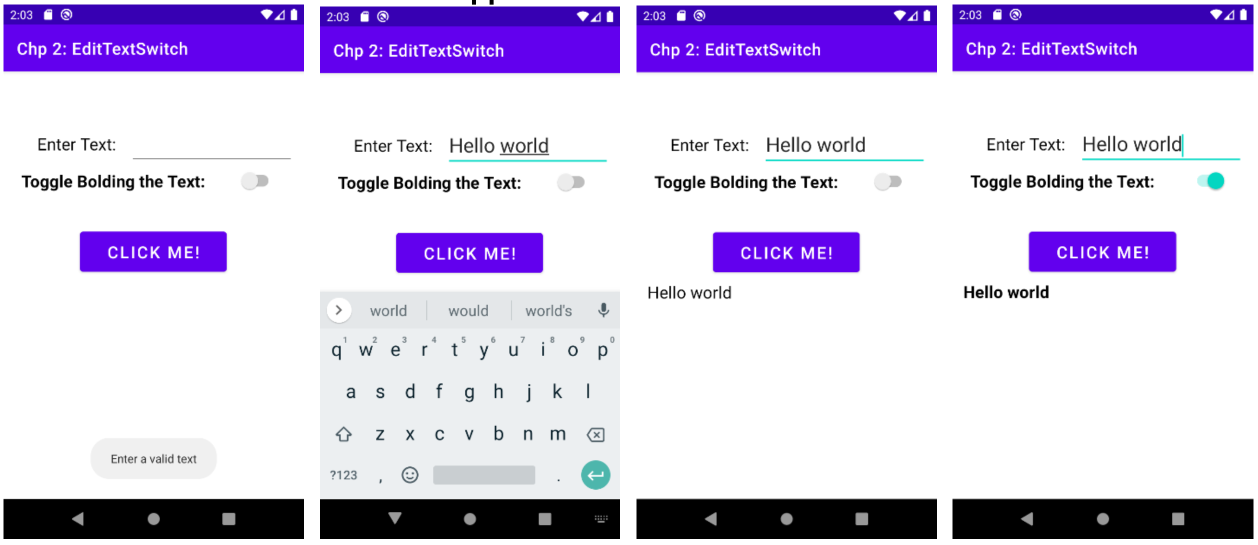
## 2.7 Example Apps of Views / Widgets / Features in Android



### Example 1: EditTextSwitch

A switch is a two-state toggle widget that can select between two options. In the following app, after the user enters the text in the EditText and clicks on the Button, the switch is used to toggle between bolding or un-bolding the text output below the Button. The Switch can be found under the Buttons section of the palette.

### Screenshots for EditTextSwitch app



strings.xml

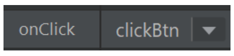
```
<resources>
    <string name="app_name">Chp 2: EditTextSwitch</string>
    <string name="enterText">Enter Text:</string>
    <string name="btnText">Click Me!</string>
    <string name="switchText">Toggle Bolding the Text: </string>
</resources>
```

MainActivity.kt

```
1   import android.content.Context
2   import android.graphics.Typeface
3   import androidx.appcompat.app.AppCompatActivity
4   import android.os.Bundle
5   import android.view.MotionEvent
6   import android.view.View
7   import android.view.inputmethod.InputMethodManager
8   import android.widget.EditText
9   import android.widget.Switch
10  import android.widget.TextView
11  import android.widget.Toast
12
13  class MainActivity : AppCompatActivity() {
14      override fun onCreate(savedInstanceState: Bundle?) {
15          super.onCreate(savedInstanceState)
16          setContentView(R.layout.activity_main)
17          val dispText : TextView = findViewById(R.id.dispTextView)
18          val sw : Switch = findViewById(R.id.boldSwitch)
19          sw.setOnCheckedChangeListener { view, isChecked ->
20              if (isChecked) {
21                  dispText.setTypeface(null, Typeface.BOLD)
22              } else {
23                  dispText.setTypeface(null, Typeface.NORMAL)
24              }
25          }
26      }
27
28      // The method will force the touch keyboard to hide when user touches anywhere on screen
29      override fun onTouchEvent(event: MotionEvent?): Boolean {
30          val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
31          if(imm.isAcceptingText) imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
32          return true
33      }
34
35      fun clickBtn(view: View){
36          val inputText = (findViewById<View>(R.id.editText) as EditText).text.toString()
37          val dispText : TextView = findViewById(R.id.dispTextView)
38          if (inputText == "")
39              Toast.makeText(this, "Enter a valid text",Toast.LENGTH_LONG).show()
40          else {
41              dispText.text = inputText
42              Toast.makeText(this, "Text Displayed!", Toast.LENGTH_LONG).show()
43              val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
44              imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
45          }
46      }
47  }
```

Explanation:

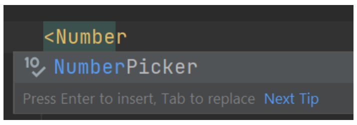
| | |
|---|---|
| Lines 18 to 25 | The code implements a OnCheckedChangeListener on the switch to listen for any changes on the switch. Note that a ***lambda expression*** *(similar to what you learnt in OOP2)* is used to code the listener's functionalities, such that view and isChecked, of type View and Boolean respectively, are parameters for the overridden method OnCheckedChangeListener. The listener code sets the respective typeface (bold or normal) in accordance to the switch's state. |
| Lines 28 to 33 | This code forces the keyboard (if it is currently on the screen), to disappear when the user taps anywhere on the screen. |

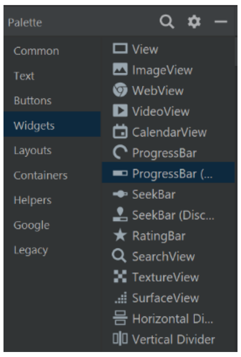
| | |
|---|---|
| Lines 35 to 46 | Another way to add a View functionality is to link a method to its on… attribute. For this case, the method clickBtn is linked to the Button’s onClick attribute, as you see in the XML file below, or if you are on design mode, you will have to set the following:  onClick clickBtn . If the inputText retrieved from the EditText is empty, a Toast (a message displayed at the bottom in the first screenshot) will reflect that the message is invalid, otherwise, output the inputText in the TextView below the button.<br><br>Note certain differences between Kotlin and Java. Similar to Python, you can now compare strings simply with an “==” instead of an equalsTo() method. Also, in line 36 and 41, in general, if the object has both an accessor and mutator method, both can be accessed through just its associated attribute. For example, in line 41, instead of dispText.setText(inputText), you can simply write dispText.text = inputText.<br><br>In line 36, notice how object explicit type casting is done with the “as” keyword to cast a View object to its descendent, the EditText. |

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/button"
        android:layout_width="176dp"
        android:layout_height="60dp"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        android:layout_weight="1"
        android:onClick="clickBtn"
        android:text="@string/btnText"
        android:textSize="20sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.497"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.412" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:text="@string/enterText"
        android:textColor="@color/black"
        android:textSize="20sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.127"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.15" />

    <EditText
        android:id="@+id/editText"
        android:layout_width="197dp"
        android:layout_height="48dp"
        android:layout_marginStart="16dp"
        android:layout_weight="1"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="@+id/textView2"
        app:layout_constraintHorizontal_bias="0.693"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintStart_toEndOf="@+id/textView2"
        app:layout_constraintTop_toTopOf="@+id/textView2"
        app:layout_constraintVertical_bias="0.368" />

    <Switch
        android:id="@+id/boldSwitch"
        android:layout_width="307dp"
        android:layout_height="49dp"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        android:layout_marginBottom="24dp"
        android:text="@string/switchText"
        android:textSize="20sp"
        android:textStyle="bold"
        app:layout_constraintBottom_toTopOf="@+id/button"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.365"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.941" />

    <TextView
        android:id="@+id/dispTextView"
        android:layout_width="335dp"
        android:layout_height="215dp"
        android:layout_marginStart="8dp"
        android:layout_marginEnd="8dp"
        android:textColor="@color/black"
        android:textSize="20sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.56"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/button"
        app:layout_constraintVertical_bias="0.078" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

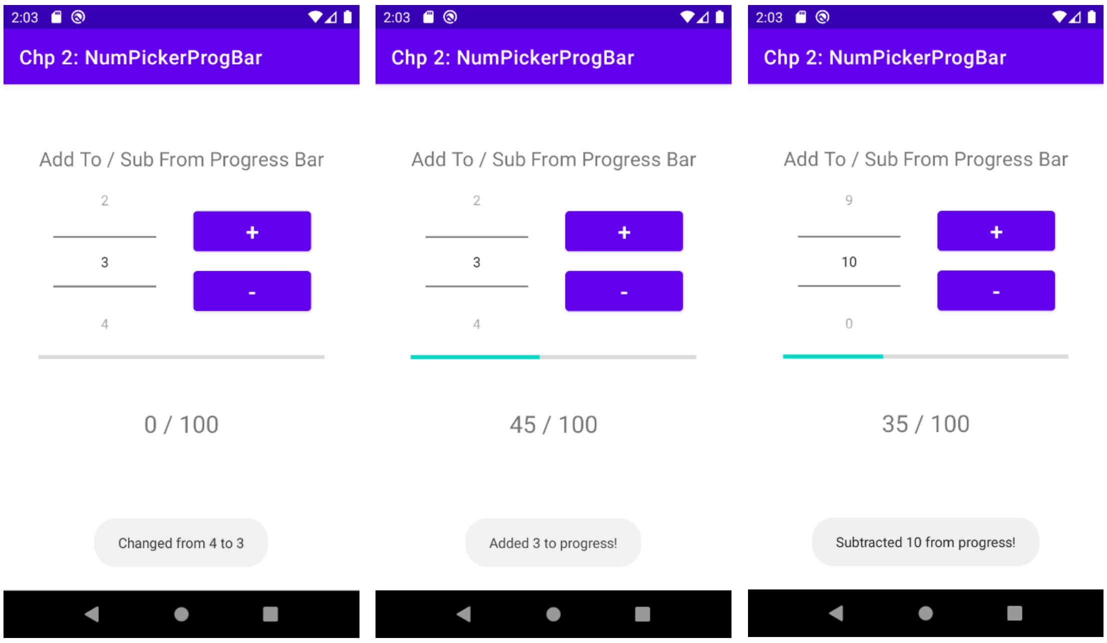
## Example 2: NumberPickerProgressBar





This app introduces another two commonly used widgets, which are the NumberPicker and the ProgressBar. The ProgressBar can be found in the Widgets section of the Palette in Design mode, whereas the NumberPicker is one such example of a widget which is NOT readily available in the Palette and needs to be retrieved through manually editing the XML code itself. This app will allow the user to interact with the number spinner then add / subtract the progress accordingly with the + or – buttons. The addition / subtraction of progress will be reflected in the progress bar.

### Screenshots for NumberPickerProgressBar app



strings.xml

```
<resources>
    <string name="app_name">Chp 2: NumPickerProgBar</string>
    <string name="textView">Add To / Sub From Progress Bar</string>
    <string name="add">+</string>
    <string name="sub">-</string>
</resources>
```

MainActivity.kt

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.NumberPicker
import android.widget.ProgressBar
import android.widget.TextView
import android.widget.Toast

class MainActivity : AppCompatActivity() {
    private var progValue : Int = 0

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
15          val picker : NumberPicker = findViewById(R.id.numberPicker)
16          picker.maxValue = 10
17          picker.minValue = 0
18          picker.wrapSelectorWheel = true
19          picker.setOnValueChangedListener{ _, oldVal, newVal ->
20              val toastText = "Changed from $oldVal to $newVal"
21              Toast.makeText(this@MainActivity, toastText, Toast.LENGTH_SHORT).show()
22          }
23      }
24      fun addSubVal(view : View) {
25          val pickerNum = (findViewById<NumberPicker>(R.id.numberPicker)).value
26          val progressBar: ProgressBar = findViewById(R.id.progressBar)
27          val dispProg : TextView = findViewById(R.id.progressScore)
28          val toastText: String
29          if (view.id == R.id.addBtn && (progValue + pickerNum) <= 100){
30              progValue += pickerNum
31              progressBar.progress = progValue
32              dispProg.text = "$progValue / 100"
33              toastText = "Added $pickerNum to progress!"
34          } else if (view.id == R.id.subBtn && (progValue - pickerNum) >= 0){
35              progValue -= pickerNum
36              progressBar.progress = progValue
37              dispProg.text = "$progValue / 100"
38              toastText = "Subtracted $pickerNum from progress!"
39          } else {
40              toastText = "Invalid add / subtract done"
41          }
42          Toast.makeText(this@MainActivity, toastText, Toast.LENGTH_SHORT).show()
43      }
44  }
```

Explanation:

| | |
|---|---|
| Lines 15 to 22 | After the picker is initialized, you will have to set the various attributes of the picker itself, then set a OnValueChangedListener such that a Toast will be output to reflect the numerical changes in the spinner.<br><br>Note on line 19, in Kotlin, it is possible for you to substitute any **unused** parameters in the ***lambda expression*** with an underscore. |
| Lines 25 to 43 | Both + and – buttons share the same addSubVal method and you are able to detect specifically which View it is by checking the view's id property (note the view is the parameter passed into the method containing information about the View object calling the method).<br><br>Note on lines 32, 33, 37 and 38, how it is possible in Kotlin to include variables into strings instead of doing endless concatenations in Java. We use the $ symbol to signify that it is a variable and if the variable is an attribute / method call, we can instead use ${…} and put the variable into the curly braces. |

activity_main.xml

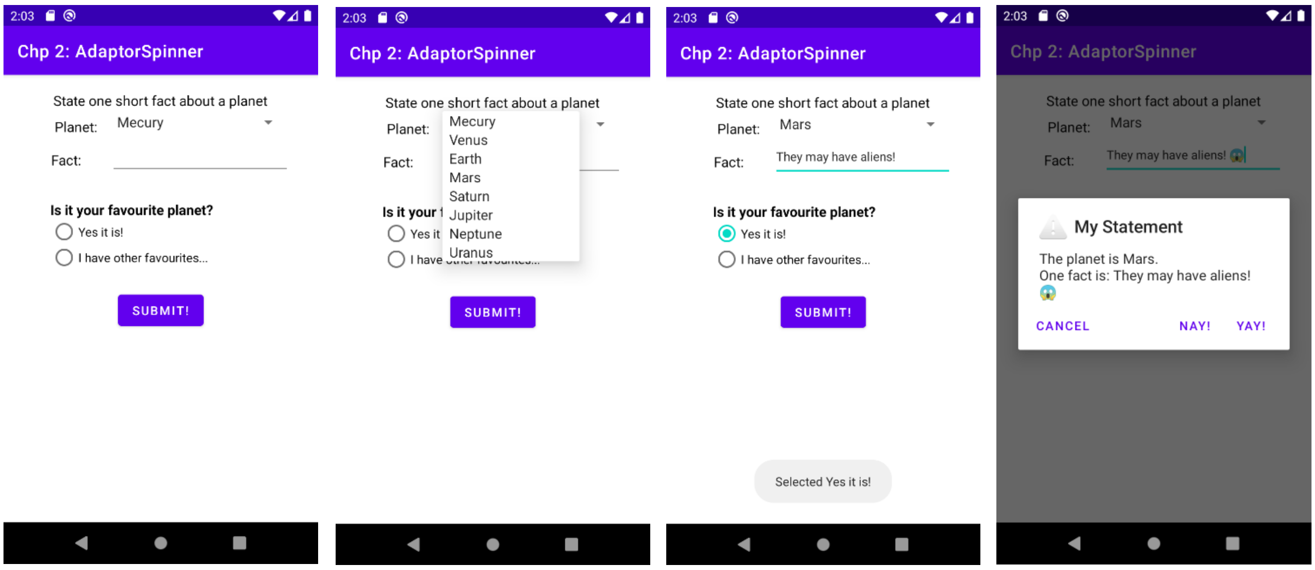
```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginStart="8dp"
        android:layout_marginTop="16dp"
        android:layout_marginEnd="8dp"
        android:text="@string/textView"
        android:textSize="20sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.504"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.097" />

    <NumberPicker
        android:id="@+id/numberPicker"
        android:layout_width="104dp"
        android:layout_height="187dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0.198"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.264" />

    <LinearLayout
        android:layout_width="143dp"
        android:layout_height="188dp"
        android:gravity="center_horizontal|center_vertical"
        android:orientation="vertical"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.407"
        app:layout_constraintStart_toEndOf="@+id/numberPicker"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.263">

        <Button
            android:id="@+id/addBtn"
            android:layout_width="119dp"
            android:layout_height="wrap_content"
            android:layout_marginBottom="8sp"
            android:onClick="addSubVal"
            android:text="@string/add"
            android:textSize="24sp" />

        <Button
            android:id="@+id/subBtn"
            android:layout_width="119dp"
            android:layout_height="wrap_content"
            android:onClick="addSubVal"
            android:text="@string/sub"
            android:textSize="24sp" />
    </LinearLayout>

    <ProgressBar
        android:id="@+id/progressBar"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="289dp"
        android:layout_height="55dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.544" />

    <TextView
        android:id="@+id/progressScore"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="0 / 100"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/progressBar"
        app:layout_constraintVertical_bias="0.134" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

## Example 3: AdaptorSpinner

This app introduces other commonly present Widgets, such as the Spinner (found inside **Containers** in the Palette), which allows the user to pick an option from a predefined list, and the RadioGroup and RadioButton (found inside **Buttons** in the Palette). The app also shows the use of a Dialog, which is a pop-up window which will only disappear when the user acknowledges one of the options. In this app, information will be retrieved from the user's selection of options then upon clicking on "Submit!", the Dialog will display the consolidated information.

You may also want to play around with the EditText, which is in fact, a MultiAutoCompleteTextView, which allows the user to type multiple lines in the TextView and gives options to complete the sentence, much like many of the text messaging functions of many apps today.

### Screenshots for AdaptorSpinner app



Planet: Mars
Fact: They may have aliens!
Selected Yes it is!

My Statement
The planet is Mars.
One fact is: They may have aliens! 😱

strings.xml

```
<resources>
    <string name="app_name">Chp 2: AdaptorSpinner</string>
    <string name="title">State one short fact about a planet</string>
    <string name="planet">Planet: </string>
    <string name="favourite">Is it your favourite planet?</string>
    <string name="yesFav">Yes it is!</string>
    <string name="noFav">I have other favourites...</string>
    <string name="fact">Fact: </string>
    <string name="submit">Submit!</string>
    <string-array name="planetList">
        <item>Mecury</item>
        <item>Venus</item>
        <item>Earth</item>
        <item>Mars</item>
        <item>Saturn</item>
        <item>Jupiter</item>
        <item>Neptune</item>
        <item>Uranus</item>
    </string-array>
    <string name="alertTitle">My Statement</string>
</resources>
```

MainActivity.kt

```
import android.content.Context
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.MotionEvent
import android.view.View
import android.view.inputmethod.InputMethodManager
import android.widget.*
import androidx.appcompat.app.AlertDialog

class MainActivity : AppCompatActivity() {
    private lateinit var planet : String
    private lateinit var fact : String
    private var fav : Boolean = false

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        val spinner : Spinner = findViewById(R.id.spinner)
        val radioGrp = findViewById<RadioGroup>(R.id.radio_group)
        val planetList = resources.getStringArray(R.array.planetList)
        val adapter = ArrayAdapter(this, android.R.layout.simple_spinner_item, planetList)
        spinner.adapter = adapter
        spinner.onItemSelectedListener = object : AdapterView.OnItemSelectedListener{
            override fun onItemSelected(parent: AdapterView<*>, view:View,
                                        position:Int, id:Long) {
                closeKeyboard()
                planet = planetList[position]
                Toast.makeText(this@MainActivity,
                    planetList[position] + " has been selected!", Toast.LENGTH_SHORT).show()
            }
            override fun onNothingSelected(parent: AdapterView<*>?) {
                closeKeyboard()
                Toast.makeText(this@MainActivity,
                    "No planet selected! Select a planet.", Toast.LENGTH_SHORT).show()
            }
        }
        radioGrp.setOnCheckedChangeListener {_, checkedId ->
            val radio: RadioButton = findViewById(checkedId)
            closeKeyboard()
            fav = if (checkedId == R.id.yesFav)
                radio.text.equals(R.string.yesFav) else radio.text.equals(R.string.noFav)
            Toast.makeText(this@MainActivity,
                "Selected ${radio.text}", Toast.LENGTH_SHORT).show()
        }
    }
46      // The method will force the touch keyboard to hide when user touches anywhere on screen
47      override fun onTouchEvent(event: MotionEvent?): Boolean {
48          closeKeyboard()
49          return true
50      }
51      fun closeKeyboard(){
52          val imm = getSystemService(Context.INPUT_METHOD_SERVICE) as InputMethodManager
53          if(imm.isAcceptingText) imm.hideSoftInputFromWindow(currentFocus!!.windowToken, 0)
54      }
55      fun submitBtn(view: View){
56          closeKeyboard()
57          fact = (findViewById<AutoCompleteTextView>(R.id.fact)).text.toString()
58          var start = ""
59          start += if (fav) "My favourite planet is " else "The planet is "
60
61          val builder = AlertDialog.Builder(this)
62          val diagText = "$start$planet.\nOne fact is: $fact"
63          builder.setTitle(R.string.alertTitle)
64          builder.setMessage(diagText)
65          builder.setIcon(android.R.drawable.ic_dialog_alert)
66
67          // Performing positive action
68          builder.setPositiveButton("Yay!"){_, _ ->
69              Toast.makeText(applicationContext,"Clicked Yay!",Toast.LENGTH_LONG).show()
70          }
71          // Performing cancel action
72          builder.setNeutralButton("Cancel"){_, _ ->
73              Toast.makeText(applicationContext,"Clicked Cancel",Toast.LENGTH_LONG).show()
74          }
75
76          // Performing negative action
77          builder.setNegativeButton("Nay!"){_, _ ->
78              Toast.makeText(applicationContext,"Clicked Nay!",Toast.LENGTH_LONG).show()
79          }
80          // Create the AlertDialog
81          val alertDialog: AlertDialog = builder.create()
82          // Set other dialog properties
83          alertDialog.setCancelable(false)
84          alertDialog.show()
85      }
86  }
```

Explanation:

| | |
|---|---|
| Lines 20 to 36 | The spinner needs to use of an array Adapter to store the options (in this case, it will be the planets information. To do that we can define a **string array** in strings.xml, pull the resource from the XML file, then store the resource into an ArrayAdapter object. This ArrayAdapter will be part of the adapter attribute of the Spinner object and the onItemSelectedListener will be setup to retrieve the user selected options. ***Note that as there is more than one method which needs to be overridden, a lambda expression cannot be written.***<br><br>To obtain the string from the user selected option, you can simply call planets[position], such that the position is the index of the item in the adapter. |
| Lines 37 to 43 | The OnCheckedChangedListener needs to be setup for the RadioGroup such that the user selected option will be indicated by the checkedId parameter. **Note that both radio buttons MUST have different ids.** A Toast will then be generated containing the option the user selected. |

| | |
|---|---|
| | Note lines 40 and 41, where Kotlin allows an if statement (or when statement) to be written after the assignment operator, effectively shortening the if-else statement. |
| Lines 55 to 85 | This method is linked to the onClick attribute of the “Submit!” button, and the AlertDialog Builder object will be called, where different attributes are subsequently set. You can also set the different buttons of the dialog to have customized texts and actions (the PositiveButton and NegativeButton). Finally, do not forget to actually create the AlertDialog and show it.<br><br>A Dialog can be quite a powerful feature you may want to use in your apps and can be used in ways more than just a simple AlertDialog. You can view more applications in the link below:<br>https://developer.android.com/guide/topics/ui/dialogs |

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="59dp"
        android:layout_height="26dp"
        android:text="@string/planet"
        android:textAlignment="center"
        android:textColor="@android:color/black"
        android:textSize="16sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0.178"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView3"
        app:layout_constraintVertical_bias="0.019" />

    <TextView
        android:id="@+id/textView2"
        android:layout_width="59dp"
        android:layout_height="26dp"
        android:layout_marginTop="8dp"
        android:text="@string/fact"
        android:textAlignment="viewStart"
        android:textColor="@android:color/black"
        android:textSize="16sp"
        app:layout_constraintBottom_toBottomOf="@+id/fact"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.181"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView"
        app:layout_constraintVertical_bias="0.6" />

    <TextView
        android:id="@+id/textView3"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:text="@string/title"
        android:textColor="@android:color/black"
        android:textSize="16sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintHorizontal_bias="0.496"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.022" />

    <Spinner
        android:id="@+id/spinner"
        android:layout_width="205dp"
        android:layout_height="29dp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.227"
        app:layout_constraintStart_toEndOf="@+id/textView"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.082" />

    <RadioGroup
        android:id="@+id/radio_group"
        android:layout_width="283dp"
        android:layout_height="109dp"
        android:layout_marginTop="16dp"
        android:padding="15dp"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView2">

        <TextView
            android:id="@+id/fav"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="@string/favourite"
            android:textColor="@color/black"
            android:textSize="16sp"
            android:textStyle="bold" />

        <RadioButton
            android:id="@+id/yesFav"
            android:layout_width="95dp"
            android:layout_height="29dp"
            android:text="@string/yesFav" />

        <RadioButton
            android:id="@+id/noFav"
            android:layout_width="192dp"
            android:layout_height="30dp"
            android:text="@string/noFav" />

    </RadioGroup>

    <MultiAutoCompleteTextView
        android:id="@+id/fact"
        android:layout_width="206dp"
        android:layout_height="48dp"
        android:textSize="14sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.214"
        app:layout_constraintStart_toEndOf="@+id/textView2"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.145" />

    <Button
        android:id="@+id/submit"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="8dp"
        android:onClick="submitBtn"
        android:text="@string/submit"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/radio_group"
        app:layout_constraintVertical_bias="0.0" />

</androidx.constraintlayout.widget.ConstraintLayout>
```

## [Reference]

[1] Android Layout and Views https://data-flair.training/blogs/android-layout-and-views/
[2] Build a UI with Layout Editor https://developer.android.com/studio/write/layout-editor
[3] Constraint Layout https://developer.android.com/training/constraint-layout
[4] TextView https://developer.android.com/reference/kotlin/android/widget/TextView
[5] Good Fonts for Dyslexia (Rello, Baeza-Yates, 2013): http://dyslexiahelp.umich.edu/sites/default/files/good_fonts_for_dyslexia_study.pdf
[5] ImageView https://developer.android.com/reference/android/widget/ImageView
[6] Screen Densities of Multiple Screens: https://developer.android.com/training/multiscreen/screendensities
[7] Android Event Handling https://www.tutorialspoint.com/android/android_event_handling.htm