Android Programming

Module Number: 02

Module Name: UI Layouts and UI Elements

UI Layouts and UI Elements

Aim

To equip students in designing the desired User Interface for their Android applications.

Objectives

The Objectives of this module are:

- Explain various views with respect to layout design.
- Explain the various components under views on Android Platform.

Outcome

At the end of this module, you are expected to:

- Outline the use of views in Android Platform
- Compare the use of various layout design in Android platforms

UI Layouts and UI Elements

Content

1. **Layout**
   1. Linear Layouts
   2. Relative Layout
   3. Table Layout
   4. Constraint Layout

2. **View**
   1. TextView
   2. EditText
   3. Button
   4. Events & Listeners
   5. Image Button
   6. floating Button

(Continued) Content

7. AutoCompleteTextView
8. RadioButton
9. RadioGroup
10. ToogleButton
11. CheckBox
12. Spinner
13. ProgressBar
14. Toast
15. Alert Dialogs
16. Custom Alert Dialog.

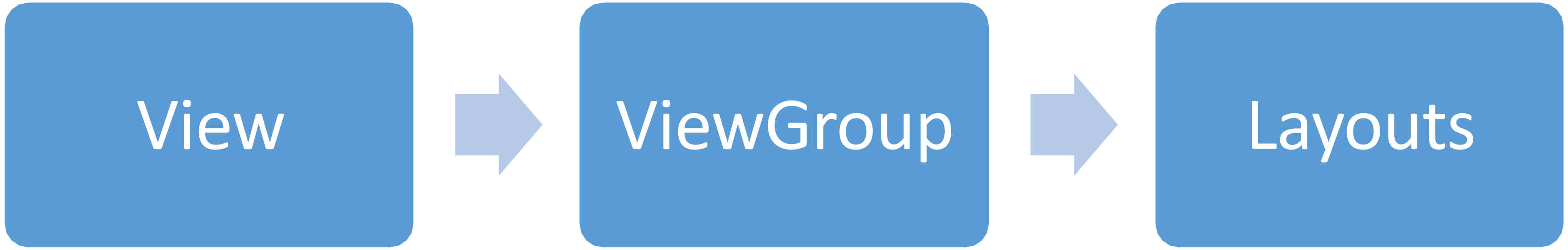
UI Layouts and UI Elements

Introduction

The basic building block for user interface is a **View** object which is created from the View class and occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for widgets, which are used to create interactive UI components like buttons, text fields, etc.

The **ViewGroup** is a subclass of **View** and provides invisible container that holds other Views or other ViewGroups and define their layout properties.

At third level, we have different **Layouts** which are subclasses of ViewGroup class and a typical layout defines the visual structure for an Android user interface and can be created either at run time using **View/ViewGroup** objects or you can declare your layout using simple XML file **main_layout.xml** which is located in the res/layout folder of your project.


View

ViewGroup

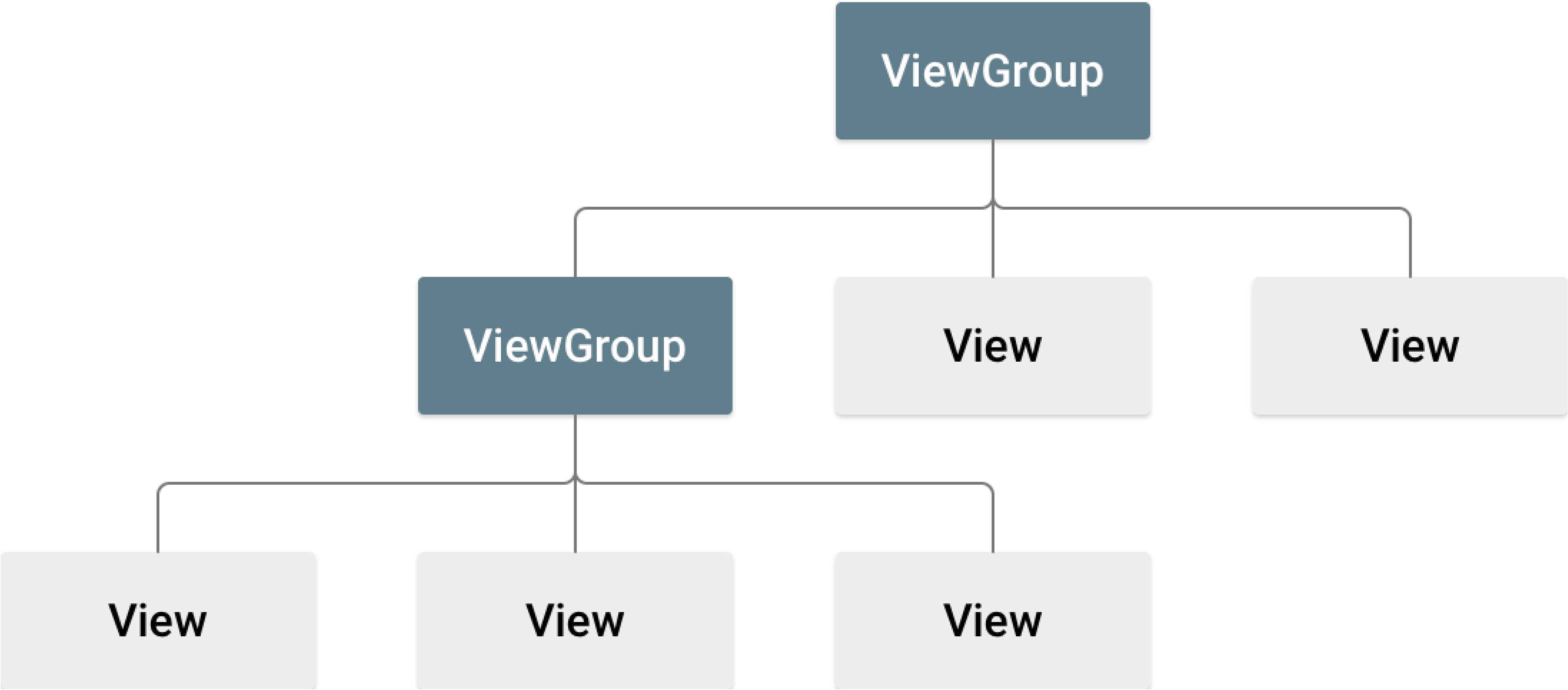
Layouts

Layout

- Layouts are subclasses of **ViewGroup** class.
- Layout comprises of **View** and **Viewgroups**.
- A typical layout defines the visual structure for an Android user interface.
- Layout is the third most important concept to be known after View and Viewgroups while dealing with User Interface.
- These different layouts are the subclasses of the **Viewgroup** class.
- The layout creates the visual structure for the user interface and can also be created during runtime using **View/ ViewGroup** objects.
- You can also declare the layout using simple XML file called **main_layout.xml** which is located in the **res/layout** folder of the new project that you are creating.

UI Layouts and UI Elements

Illustration of a view hierarchy, which defines a UI layout



UI Layouts and UI Elements

(Continued) Illustration of a view hierarchy, which defines a UI layout

The **View** objects are usually called "widgets" and can be one of many subclasses, such as **Button** or **TextView**. The **ViewGroup** objects are usually called "layouts" can be one of many types that provide a different layout structure, such as **LinearLayout** or **ConstraintLayout** .

You can declare a layout in two ways:

- **Declare UI elements in XML**. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.

You can also use Android Studio's Layout Editor to build your XML layout using a drag-and-drop interface.

- **Instantiate layout elements at runtime**. Your app can create View and ViewGroup objects (and manipulate their properties) programmatically.

UI Layouts and UI Elements

Android Layout Attributes

All the android layouts have a set of attributes that defines the visual properties of that layout. There are some common attributes with all the layouts and there are also some specific attributes for a specific layout type.

Attribute	**Description**
android:id	An ID that uniquely identifies the view
android:layout_width	Width of the layout
android:layout_height	Height of the layout
android:layout_marginTop	Specifies the extra space on the top side of the layout
android:layout_marginBottom	Specifies the extra space on the bottom side of the layout
android:layout_marginLeft	Specifies the extra space on the left side of the layout
android:layout_marginRight	Specifies the extra space on the right side of the layout.

UI Layouts and UI Elements

(Continued) Android Layout Attributes

Attribute	**Description**
android:layout_gravity	Position of the child Views
android:layout_weight	Specifies the extra space in the layout should be allocated to the View
android:layout_x	Specifies the x-coordinate of the layout
android:layout_y	Specifies the y-coordinate of the layout
android:paddingLeft	Left padding filled for the layout
android:paddingRight	Right padding filled for the layout
android:paddingTop	Top adding filled for the layout
android:paddingBottom	Bottom padding filled for the layout

UI Layouts and UI Elements

Android Layout Types

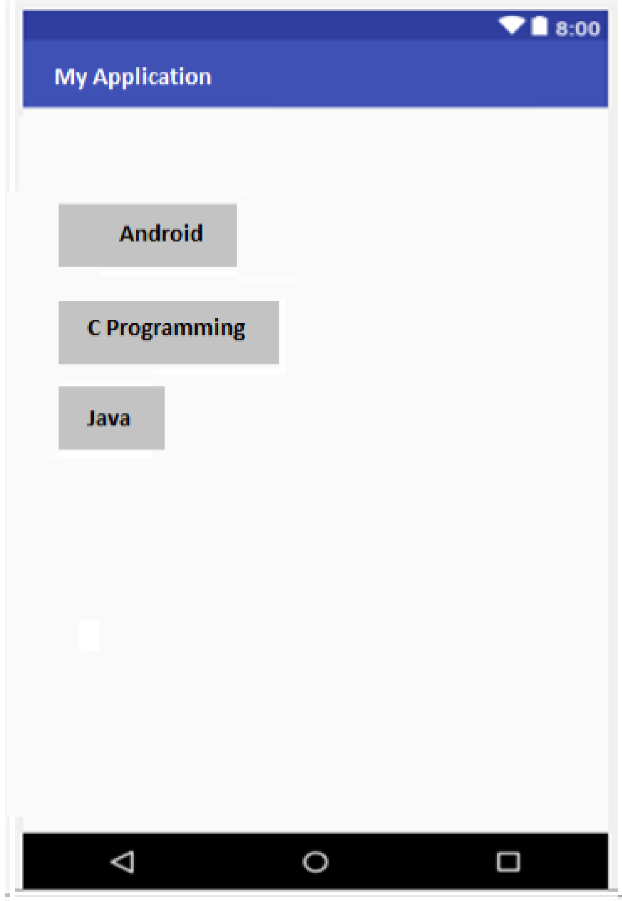
There are a number of layouts that are provided by Android.

- Linear Layout
- Relative Layout
- Table Layout
- Absolute Layout
- Frame Layout
- List View
- Grid View
- Constraint Layout

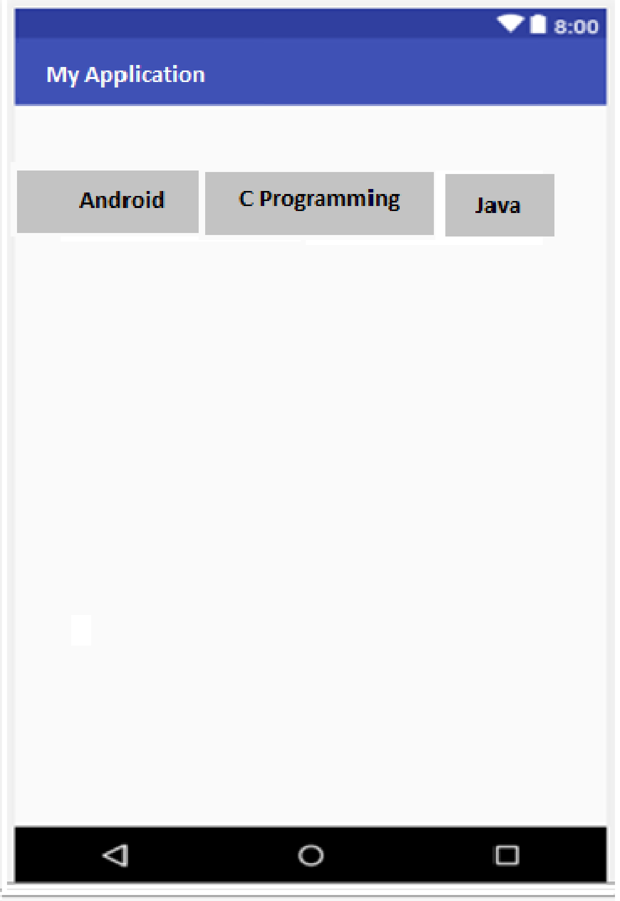
UI Layouts and UI Elements

Linear Layout

In this layout, all the children are aligned in a single direction either vertically or horizontally.



Horizontal



Vertical

UI Layouts and UI Elements

Code to define Linear Layout in Your XML layout

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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">
        <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:orientation="horizontal"
        tools:ignore="MissingConstraints">

            <Button android:id="@+id/android"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_margin="5dp"
                android:text="Android"/>
```

UI Layouts and UI Elements

(Continued) Code to define Linear Layout in Your XML layout

```
            <Button android:id="@+id/cprogramming"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_margin="5dp"
                android:text="C Programming"/>
            <Button android:id="@+id/java"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_margin="5dp"
                android:text="Java"/>

    </LinearLayout>

</android.support.constraint.ConstraintLayout>
```

Relative Layout

- **RelativeLayout** is a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent **RelativeLayout** area (such as aligned to the bottom, left or center).

RelativeLayout Properties

- android:layout_alignParentTop

  If "true", makes the top edge of this view match the top edge of the parent.
- android:layout_centerVertical

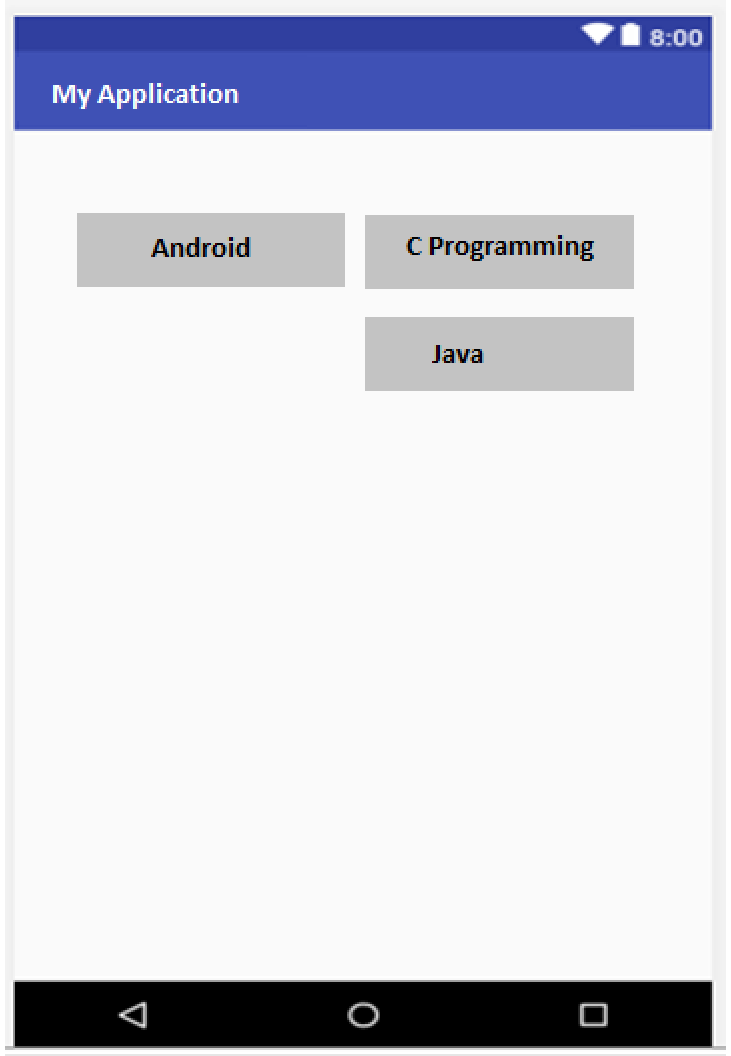
  If "true", centers this child vertically within its parent.
- android:layout_below

  Positions the top edge of this view below the view specified with a resource ID.
- android:layout_toRightOf

  Positions the left edge of this view to the right of the view specified with a resource ID.

UI Layouts and UI Elements

(Continued) Relative Layout



UI Layouts and UI Elements

Code to define RelativeLayout in Your XML layout

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

    <RelativeLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentStart="true"
        android:layout_alignParentTop="true"
        tools:ignore="MissingConstraints“>

        <Button
            android:id="@+id/android"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignParentTop="true"
            android:layout_marginStart="129dp"
            android:layout_marginTop="202dp"
            android:text="Android" />
```

(Continued) Code to define RelativeLayout in Your XML layout

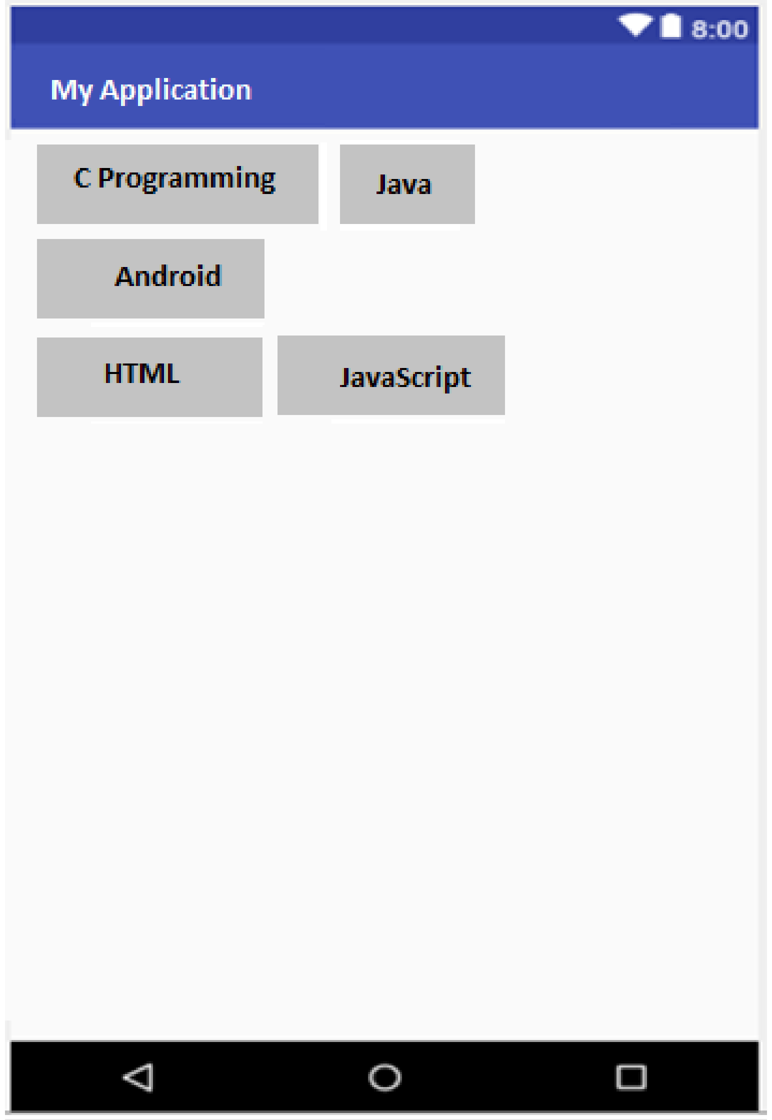
```
        <Button
            android:id="@+id/cprogramming"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignTop="@+id/android"
            android:layout_toEndOf="@+id/android"
            android:text="C Programming" />

        <Button
            android:id="@+id/java"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_alignEnd="@+id/cprogramming"
            android:layout_below="@+id/android"
            android:text="Java" />

    </RelativeLayout>

</RelativeLayout>
```

Table Layout

- TableLayout is a ViewGroup that displays child View elements in rows and columns.
- TableLayout positions its children into rows and columns.
- TableLayout containers do not display border lines for their rows, columns, or cells.
- The table will have as many columns as the row with the most cells.
- A table can leave cells empty.
- Cells can span multiple columns, as they can in HTML.
- You can span columns by using the span field in the TableRow.LayoutParams class.

UI Layouts and UI Elements

(Continued) Table Layout



UI Layouts and UI Elements

Code to define Table Layout in Your XML layout

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    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">
    <TableLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <TableRow
            android:layout_width="match_parent"
            android:layout_height="wrap_content ">
            <Button
                android:id="@+id/cprogramming"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="C Programming" />
```

UI Layouts and UI Elements

(Continued) Code to define Table Layout in Your XML layout

```
        <Button
            android:id="@+id/java"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Java" />
    </TableRow>
    <TableRow
        android:layout_width="match_parent"
        android:layout_height="match_parent">
        <Button
            android:id="@+id/android"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Android" />
    </TableRow>
```

UI Layouts and UI Elements

(Continued) Code to define Table Layout in Your XML layout

```
       <TableRow
            android:layout_width="match_parent"
            android:layout_height="match_parent">
            <Button
                android:id="@+id/html"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="HTML" />
            <Button
                android:id="@+id/javascripy"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="JavaScript" />
        </TableRow>
    </TableLayout>

</RelativeLayout>
```

Constraint Layout

- A ConstraintLayout is a ViewGroup which allows you to position and size widgets in a flexible way.
- Most preferred Layout.

Note: ConstraintLayout is available as a support library that you can use on Android systems starting with API level 9 (Gingerbread). As such, we are planning on enriching its API and capabilities over time. This documentation will reflect those changes.

(Continued) Constraint Layout

Various types of constraints that you can use:

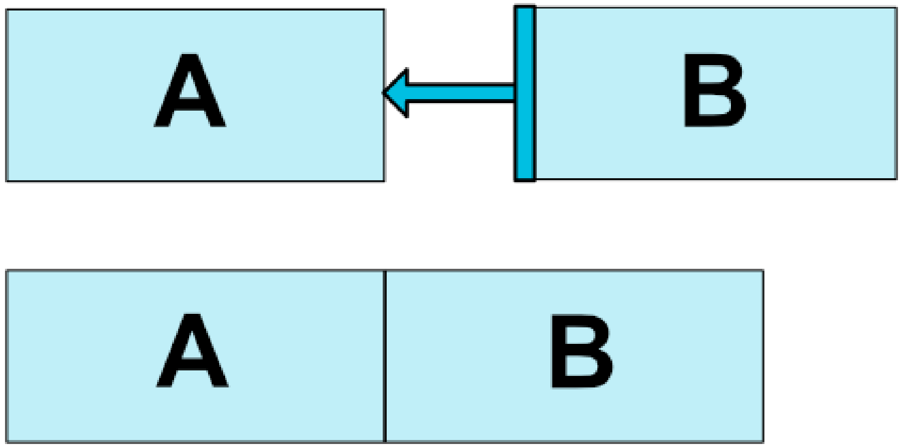
- Relative positioning
- Margins
- Centering positioning
- Circular positioning
- Visibility behavior
- Dimension constraints
- Chains
- Virtual Helpers objects
- Optimizer

(Continued) Constraint Layout

Relative positioning

- Relative positioning is one of the basic building block of creating layouts in ConstraintLayout. Those constraints allows you to position a given widget relative to another one. You can constrain a widget on the horizontal and vertical axis:
  - Horizontal Axis: left, right, start and end sides.
  - Vertical Axis: top, bottom sides and text baseline

The general concept is to constrain a given side of a widget to another side of any other widget.

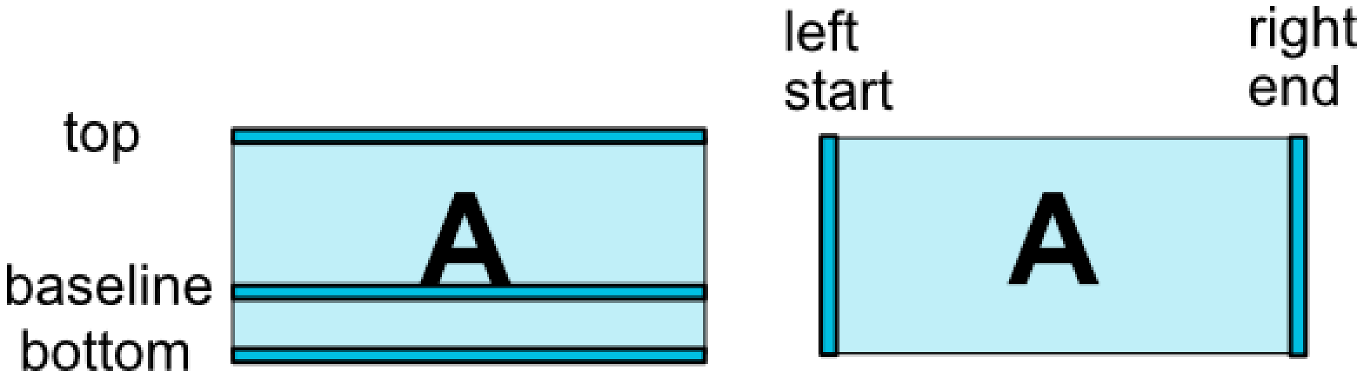


UI Layouts and UI Elements

(Continued) Constraint Layout

```
<Button android:id="@+id/buttonA" ... />
        <Button android:id="@+id/buttonB" ...
               app:layout_constraintLeft_toRightOf="@+id/buttonA" />
```

This tells the system that we want the left side of button B to be constrained to the right side of button A. Such a position constraint means that the system will try to have both sides share the same location.



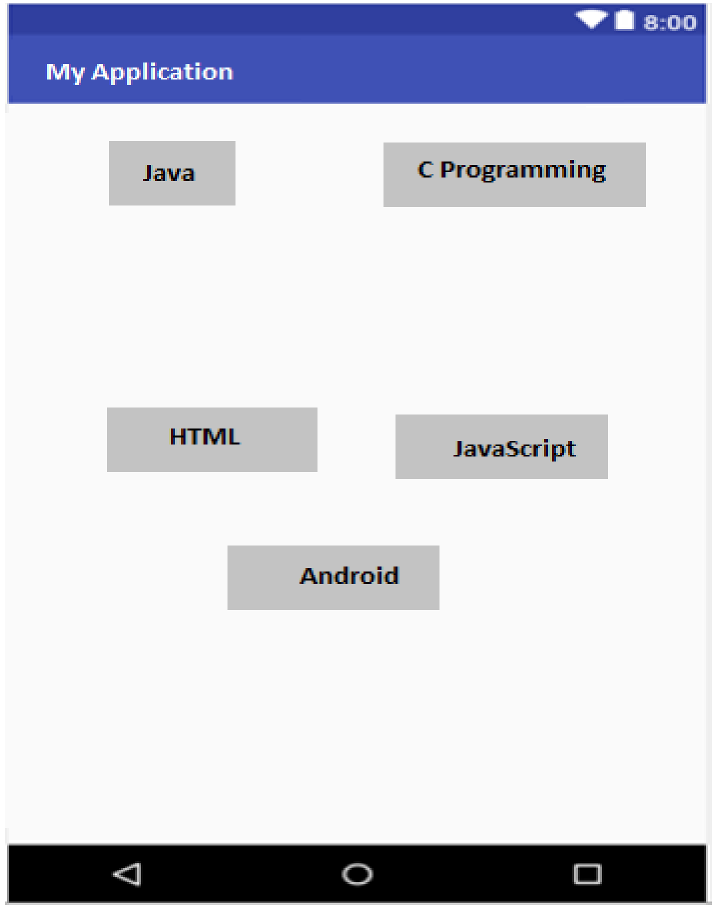
(Continued) Constraint Layout

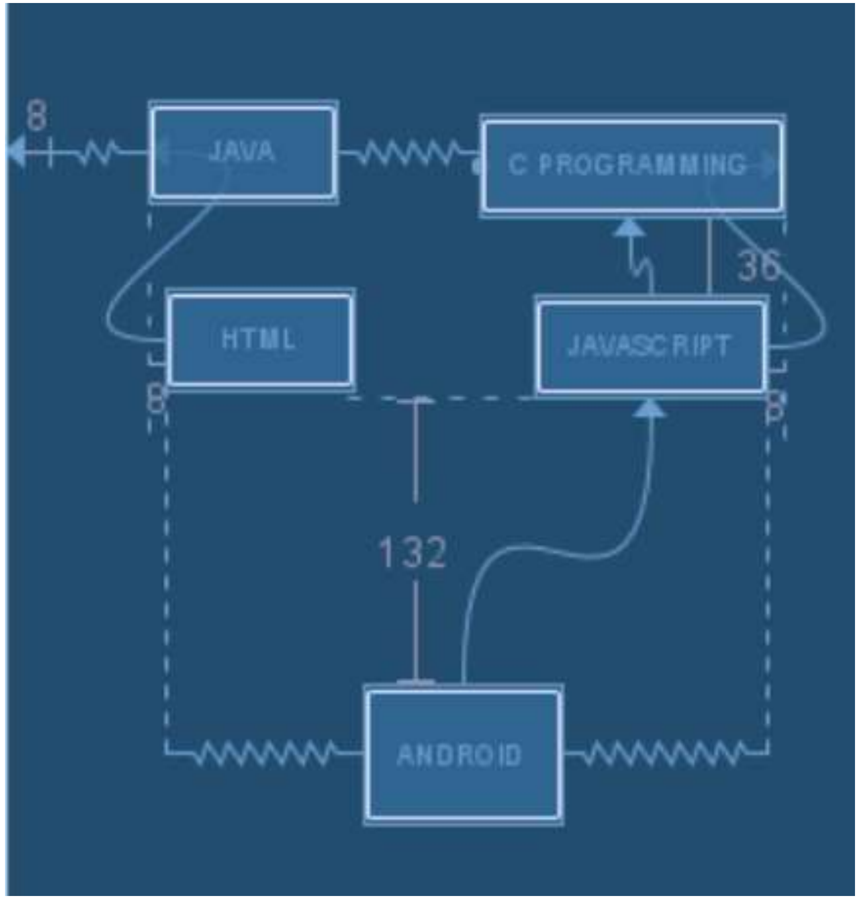
List of available constraints

- layout_constraintLeft_toLeftOf
- layout_constraintLeft_toRightOf
- layout_constraintRight_toLeftOf
- layout_constraintRight_toRightOf
- layout_constraintTop_toTopOf
- layout_constraintTop_toBottomOf
- layout_constraintBottom_toTopOf
- layout_constraintBottom_toBottomOf
- layout_constraintBaseline_toBaselineOf
- layout_constraintStart_toEndOf
- layout_constraintStart_toStartOf
- layout_constraintEnd_toStartOf
- layout_constraintEnd_toEndOf

UI Layouts and UI Elements

(Continued) Constraint Layout





UI Layouts and UI Elements

Code to define Constraint Layout in Your XML layout

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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/cprogramming"
        android:layout_width="142dp"
        android:layout_height="wrap_content"
        android:text="C Programming"
        tools:layout_editor_absoluteX="221dp"
        tools:layout_editor_absoluteY="55dp" />
    <Button
        android:id="@+id/java"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="8dp"
        android:text="Java"
        app:layout_constraintEnd_toStartOf="@+id/cprogramming"
        app:layout_constraintStart_toStartOf="parent"
        tools:ignore="MissingConstraints"
        tools:layout_editor_absoluteY="48dp" />
```

UI Layouts and UI Elements

(Continued) Code to define Constraint Layout in Your XML layout

```
<Button
    android:id="@+id/android"
    android:layout_width="93dp"
    android:layout_height="66dp"
    android:layout_marginEnd="8dp"
    android:layout_marginLeft="8dp"
    android:layout_marginRight="8dp"
    android:layout_marginStart="8dp"
    android:layout_marginTop="132dp"
    android:text="Android"
    app:layout_constraintEnd_toEndOf="@+id/javascripy"
    app:layout_constraintStart_toStartOf="@+id/html"
    app:layout_constraintTop_toBottomOf="@+id/javascripy" />

<Button
    android:id="@+id/html"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="8dp"
    android:layout_marginStart="8dp"
    android:text="HTML"
    app:layout_constraintStart_toStartOf="@+id/java"
    tools:layout_editor_absoluteY="136dp" />
```

(Continued) Code to define Constraint Layout in Your XML layout

```
    <Button
        android:id="@+id/javascripy"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:layout_marginRight="8dp"
        android:layout_marginTop="36dp"
        android:text="JavaScript"
        app:layout_constraintEnd_toEndOf="@+id/cprogramming"
        app:layout_constraintTop_toBottomOf="@+id/cprogramming" />

</android.support.constraint.ConstraintLayout>
```

UI Layouts and UI Elements

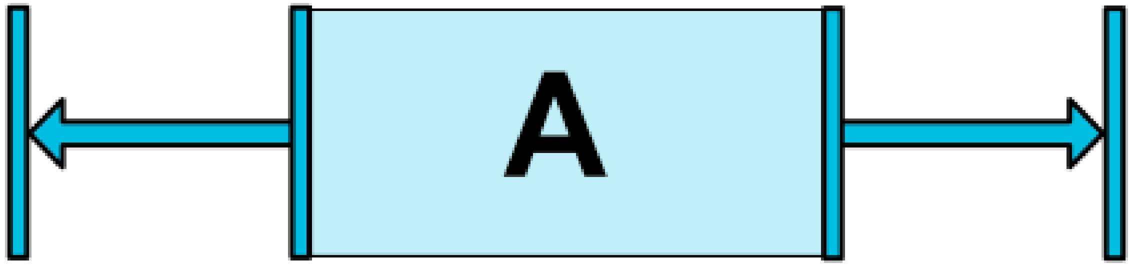
(Continued) Constraint Layout

Centering positioning and bias

A useful aspect of ConstraintLayout is in how it deals with "impossible" constrains. For example, if we have something like:

```
<android.support.constraint.ConstraintLayout ...>
        <Button android:id="@+id/button" ...
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent/>
     </>
```

Unless the ConstraintLayout happens to have the exact same size as the Button, both constraints cannot be satisfied at the same time (both sides cannot be where we want them to be).

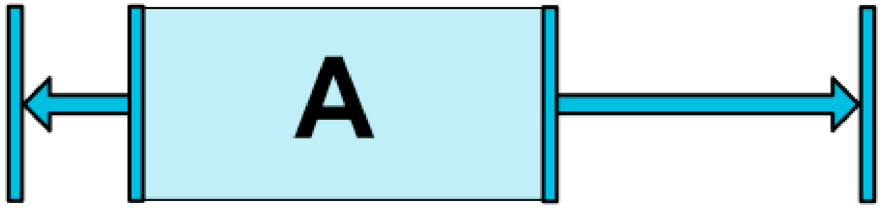


(Continued) Constraint Layout

Centering positioning with Bias

The default when encountering such opposite constraints is to center the widget; but you can tweak the positioning to favor one side over another using the bias attributes:

- **layout_constraintHorizontal_bias**
- **layout_constraintVertical_bias**



For example, the following will make the left side with a 30% bias instead of the default 50%, such that the left side will be shorter, with the widget

```
<android.support.constraint.ConstraintLayout ...>
        <Button android:id="@+id/button" ...
            app:layout_constraintHorizontal_bias="0.3"
            app:layout_constraintLeft_toLeftOf="parent"
            app:layout_constraintRight_toRightOf="parent/>
      </>
```

(Continued) Constraint Layout

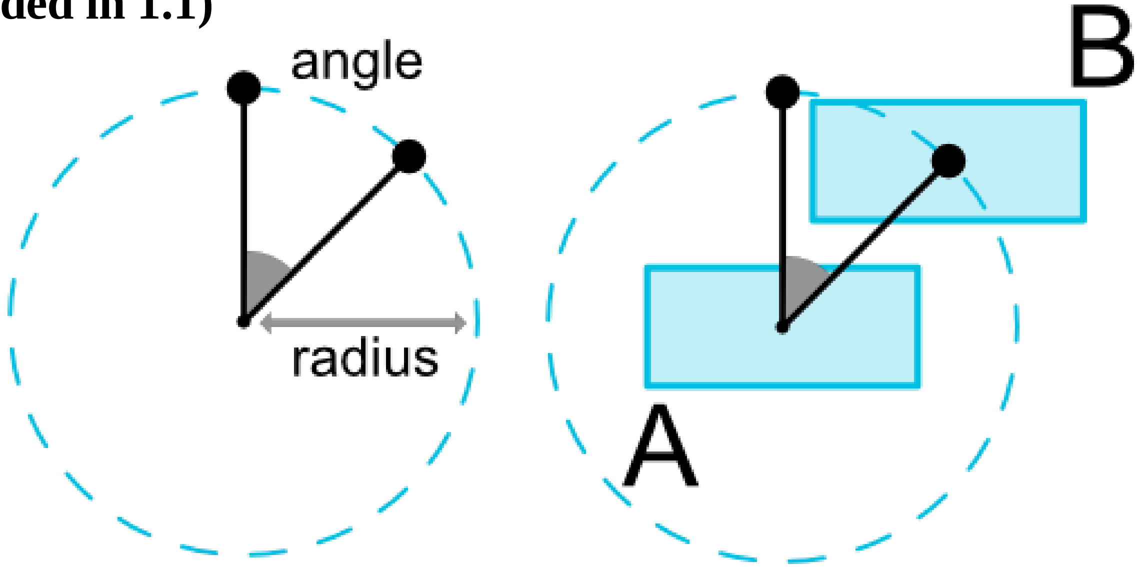
Circular positioning (Added in 1.1)

You can constrain a widget center relative to another widget center, at an angle and a distance.
This allows you to position a widget on a circle. The following attributes can be used:

- layout_constraintCircle : references another widget id.
- layout_constraintCircleRadius : the distance to the other widget center.
- layout_constraintCircleAngle : which angle the widget should be at (in degrees, from 0 to 360).

(Continued) Constraint Layout

Circular positioning (Added in 1.1)



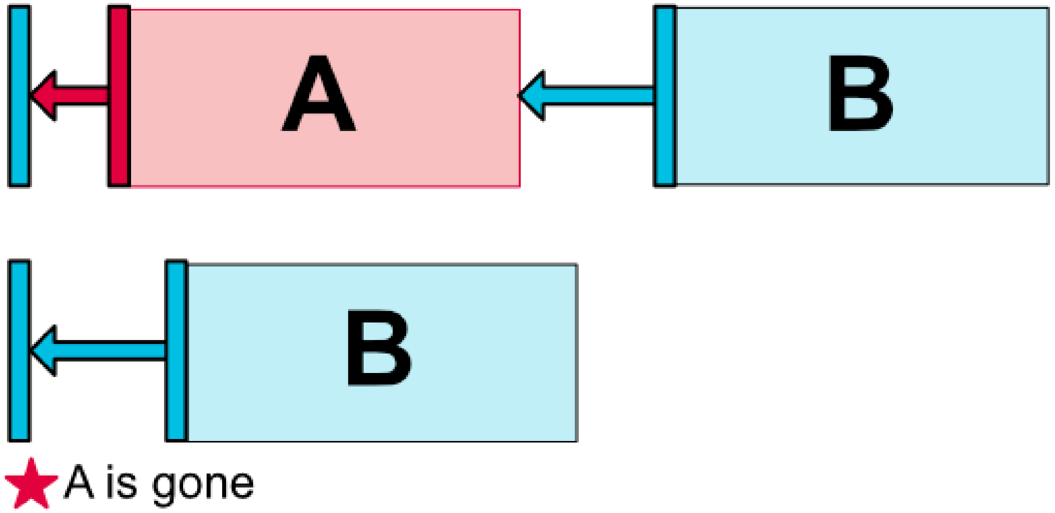
```
<Button android:id="@+id/buttonA" ... />
  <Button android:id="@+id/buttonB" ...
      app:layout_constraintCircle="@+id/buttonA"
      app:layout_constraintCircleRadius="100dp"
      app:layout_constraintCircleAngle="45" />
```

(Continued) Constraint Layout

Visibility behavior

ConstraintLayout has a specific handling of widgets being marked as **View.GONE**.
GONE widgets, as usual, are not going to be displayed and are not part of the layout itself (i.e. their actual dimensions will not be changed if marked as GONE).
But in terms of the layout computations, GONE widgets are still part of it, with an important distinction:

- For the layout pass, their dimension will be considered as zero (basically, they will be resolved to a point).
- If they have constraints to other widgets they will still be respected, but any margins will be as if equals to zero.



UI Layouts and UI Elements

(Continued) Constraint Layout

Dimensions constraints

- Minimum dimensions on ConstraintLayout
- Widgets dimension constraints
- WRAP_CONTENT : enforcing constraints (*Added in 1.1*)
- MATCH_CONSTRAINT dimensions (*Added in 1.1*)
- Min and Max
- Percent dimension
- Ratio

(Continued) Constraint Layout

Dimensions constraints
Minimum dimensions on ConstraintLayout

You can define minimum and maximum sizes for the ConstraintLayout itself:

- android:minWidth set the minimum width for the layout
- android:minHeight set the minimum height for the layout
- android:maxWidth set the maximum width for the layout
- android:maxHeight set the maximum height for the layout

Those minimum and maximum dimensions will be used by ConstraintLayout when its dimensions are set to WRAP_CONTENT.

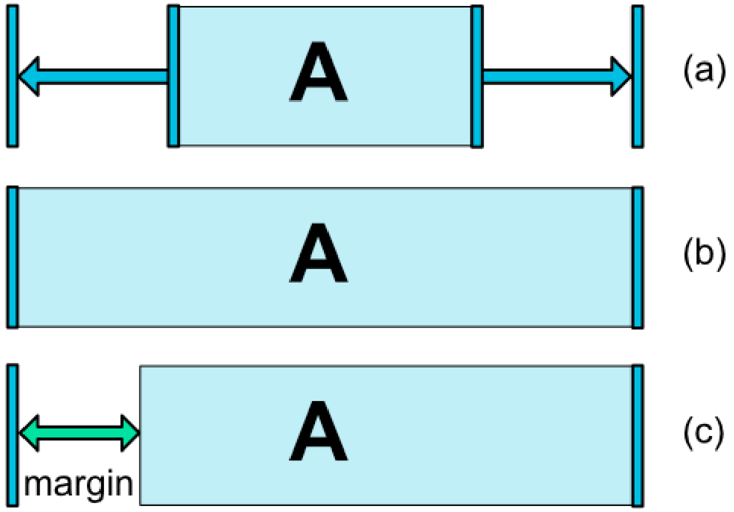
(Continued) Constraint Layout

Widgets dimension constraints

The dimension of the widgets can be specified by setting the android:layout_width and android:layout_height attributes in 3 different ways:

- Using a specific dimension (either a literal value such as 123dp or a Dimension reference)
- Using WRAP_CONTENT, which will ask the widget to compute its own size
- Using 0dp, which is the equivalent of "MATCH_CONSTRAINT"

(Continued) Constraint Layout



Widgets dimension constraints

The dimension of the widgets can be specified by setting the android:layout_width and android:layout_height attributes in 3 different ways:

Using a specific dimension (either a literal value such as 123dp or a Dimension reference)
Using WRAP_CONTENT, which will ask the widget to compute its own size
Using 0dp, which is the equivalent of "MATCH_CONSTRAINT"

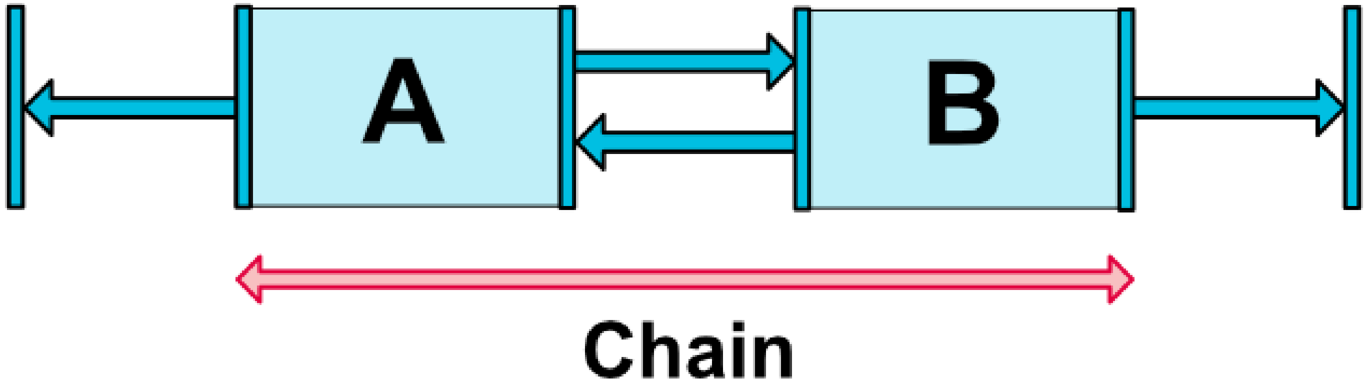
(Continued) Constraint Layout

Chains

Chains provide group-like behaviour in a single axis (horizontally or vertically). The other axis can be constrained independently.

Creating a chain

A set of widgets are considered a chain if they are linked together via a bi-directional connection (the below figure shows a minimal chain, with two widgets).



(Continued) Constraint Layout

Chain heads

Chains are controlled by attributes set on the first element of the chain (the "head" of the chain):
The head is the left-most widget for horizontal chains, and the top-most widget for vertical chains.

Margins in chains

If margins are specified on connections, they will be taken in account. In the case of spread chains, margins will be deducted from the allocated space.

(Continued) Constraint Layout

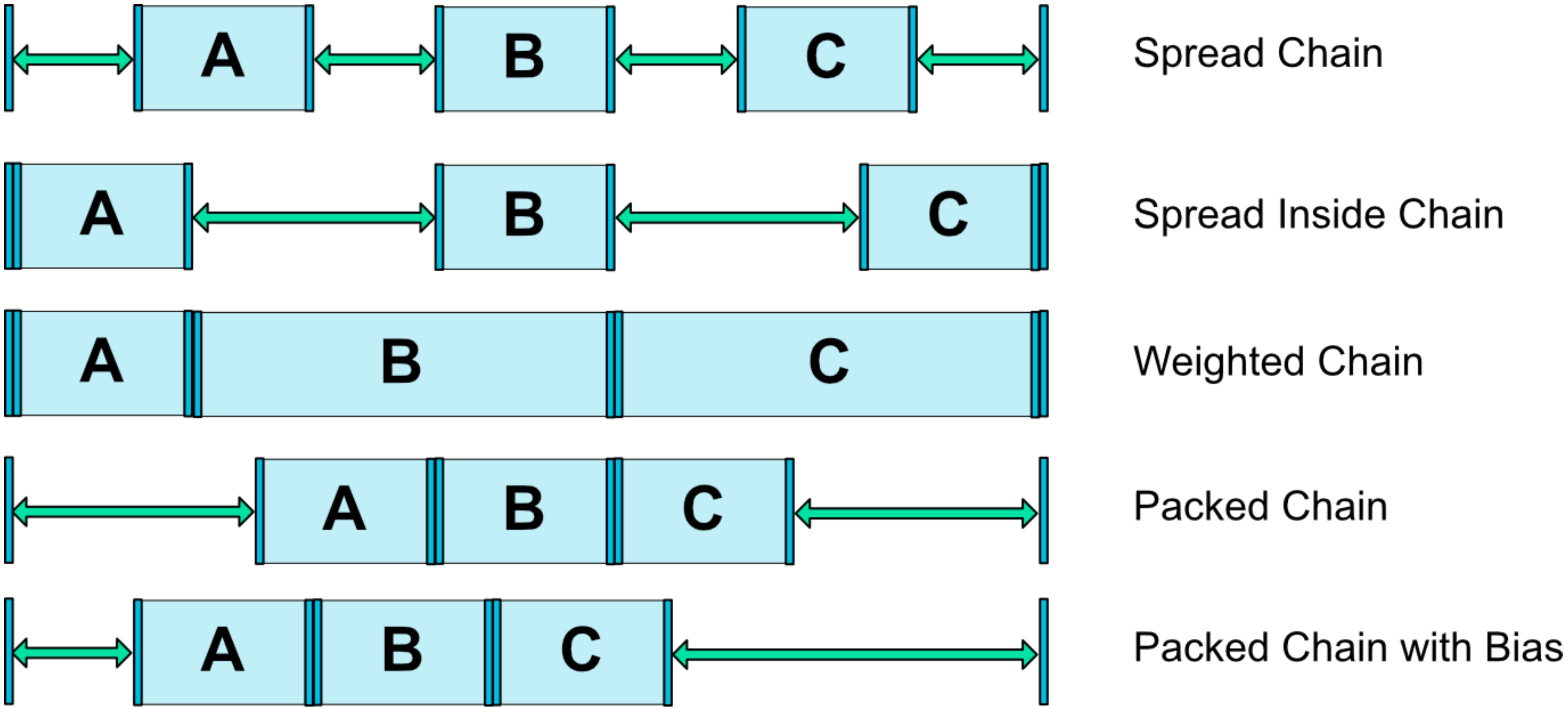
Chain Style

When setting the attribute layout_constraintHorizontal_chainStyle or layout_constraintVertical_chainStyle on the first element of a chain, the behaviour of the chain will change according to the specified style (default is CHAIN_SPREAD).

- CHAIN_SPREAD -- the elements will be spread out (default style).
- Weighted chain -- in CHAIN_SPREAD mode, if some widgets are set to MATCH_CONSTRAINT, they will split the available space.
- CHAIN_SPREAD_INSIDE -- similar, but the endpoints of the chain will not be spread out.
- CHAIN_PACKED -- the elements of the chain will be packed together. The horizontal or vertical bias attribute of the child will then affect the positioning of the packed elements.

(Continued) Constraint Layout

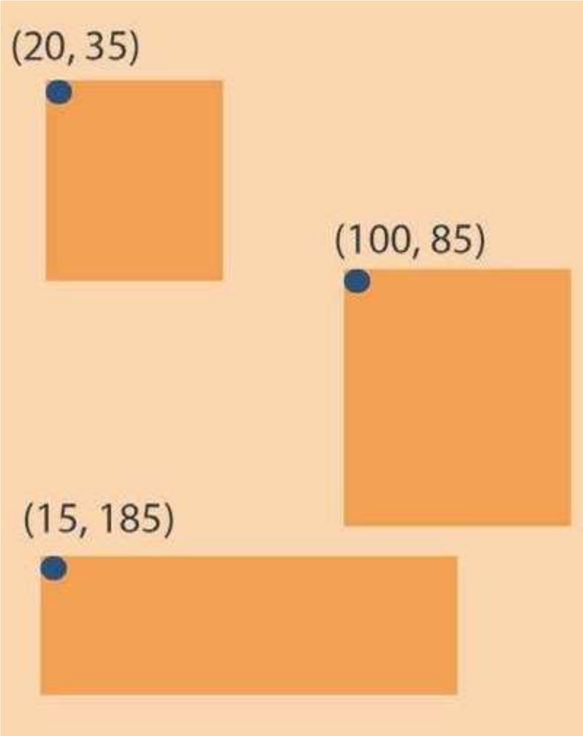
Chain Style



UI Layouts and UI Elements

Absolute Layout (Deprecated)

- This layout lets you specify the exact location of the children views.
- An Absolute Layout lets you specify exact locations (x/y coordinates) of its children.
- Absolute layouts are less flexible and harder to maintain than other types of layouts without absolute positioning.
- In the below image the numbers (20,35) denotes the X-axis value (20) and Y-axis value (35).



UI Layouts and UI Elements

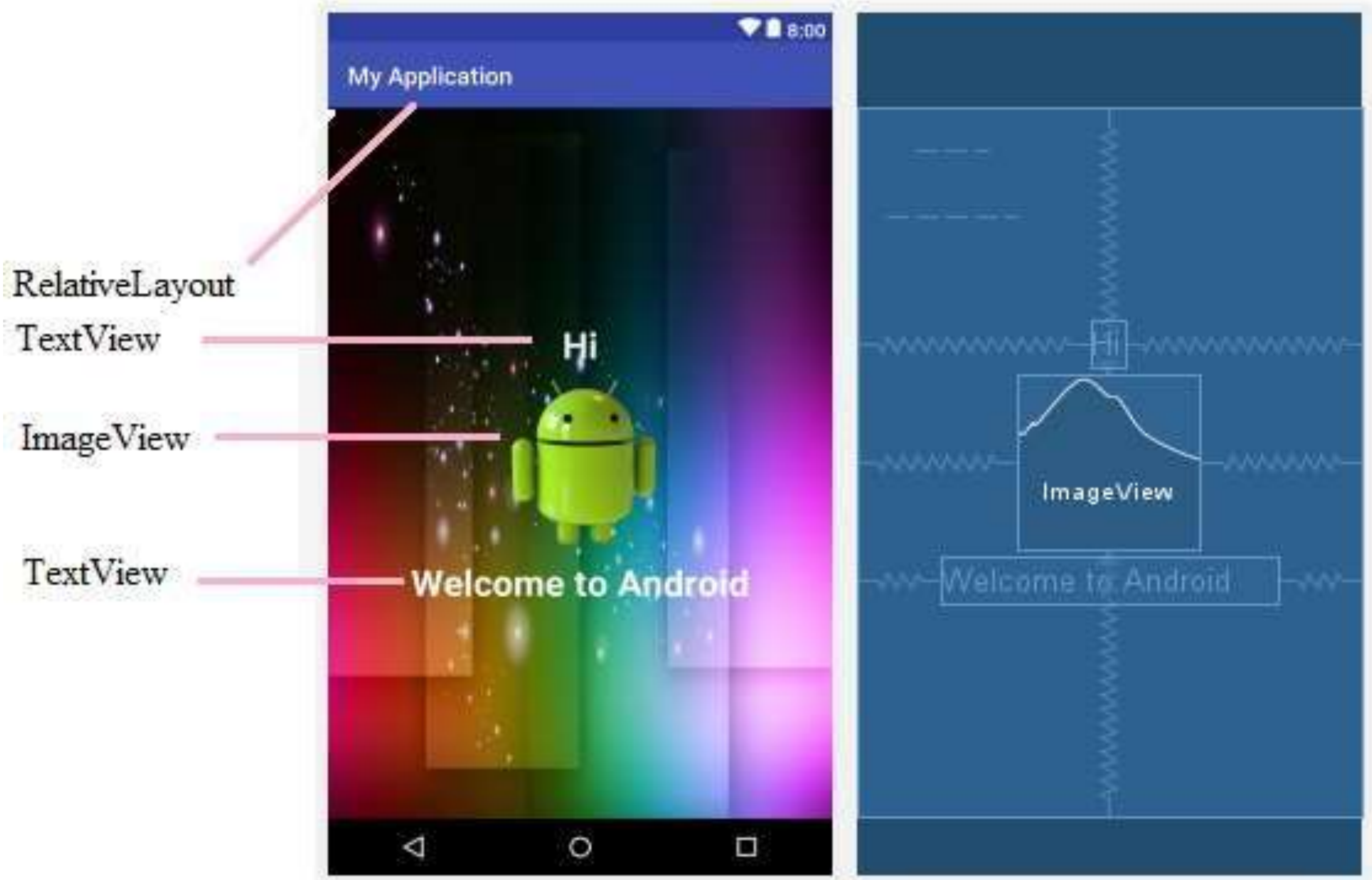
Code to define AbsoluteLayout in Your XML

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
    >
    <Button
        android:layout_width="188dp"
        android:layout_height="wrap_content"
        android:text="simplecode"
        android:layout_x="126dp"
        android:layout_y="361dp" /> <!-- X & Y position declaration-->

    <Button
        android:layout_width="113dp"
        android:layout_height="wrap_content"
        android:text=“Button"
        android:layout_x="12dp"
        android:layout_y="361dp" />  <!-- X & Y position declaration-->

</AbsoluteLayout>
```

UI Layouts and UI Elements

Frame Layout

- This layout serves as a placeholder on the screen so that you can use it to display a single view.



UI Layouts and UI Elements

Code to define FrameLayout in Your XML

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


    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_alignParentBottom="true"
        android:layout_alignParentEnd="true"
        android:layout_alignParentRight="true"
        android:background="@drawable/b2">
    </FrameLayout>
```

UI Layouts and UI Elements

(Continued) Code to define FrameLayout in Your XML

```
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_above="@+id/imageView2"
        android:layout_centerHorizontal="true"
        android:shadowColor="White"
        android:text="Hi"
        android:textStyle="bold"
        android:textColor="@color/white"
        android:textSize="30sp"
        android:visibility="visible"
        tools:layout_margin="5dp" />

<ImageView
    android:id="@+id/imageView2"
    android:layout_width="150dp"
    android:layout_height="150dp"
    android:layout_centerInParent="true"
    android:layout_gravity="center"
    app:srcCompat="@drawable/logo" />
```

UI Layouts and UI Elements

(Continued) Code to define FrameLayout in Your XML

```
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/imageView2"
        android:layout_centerHorizontal="true"
        android:shadowColor="White"
        android:textStyle="bold"
        android:text="Welcome to Android"
        android:textColor="@color/white"
        android:textSize="30sp"
        android:visibility="visible"
        tools:layout_margin="5dp" />

</RelativeLayout>
```

UI Layouts and UI Elements

Grid Layout

- The grid is composed of a set of infinitely thin lines that separate the viewing area into cells.
- Throughout the API, grid lines are referenced by grid indices.
- A grid with N columns has N + 1 grid indices that run from 0 through N inclusive.
- Regardless of how GridLayout is configured, grid index 0 is fixed to the leading edge of the container and grid index N is fixed to its trailing edge (after padding is taken into account).

(Continued) Grid Layout

Row and Column Specs

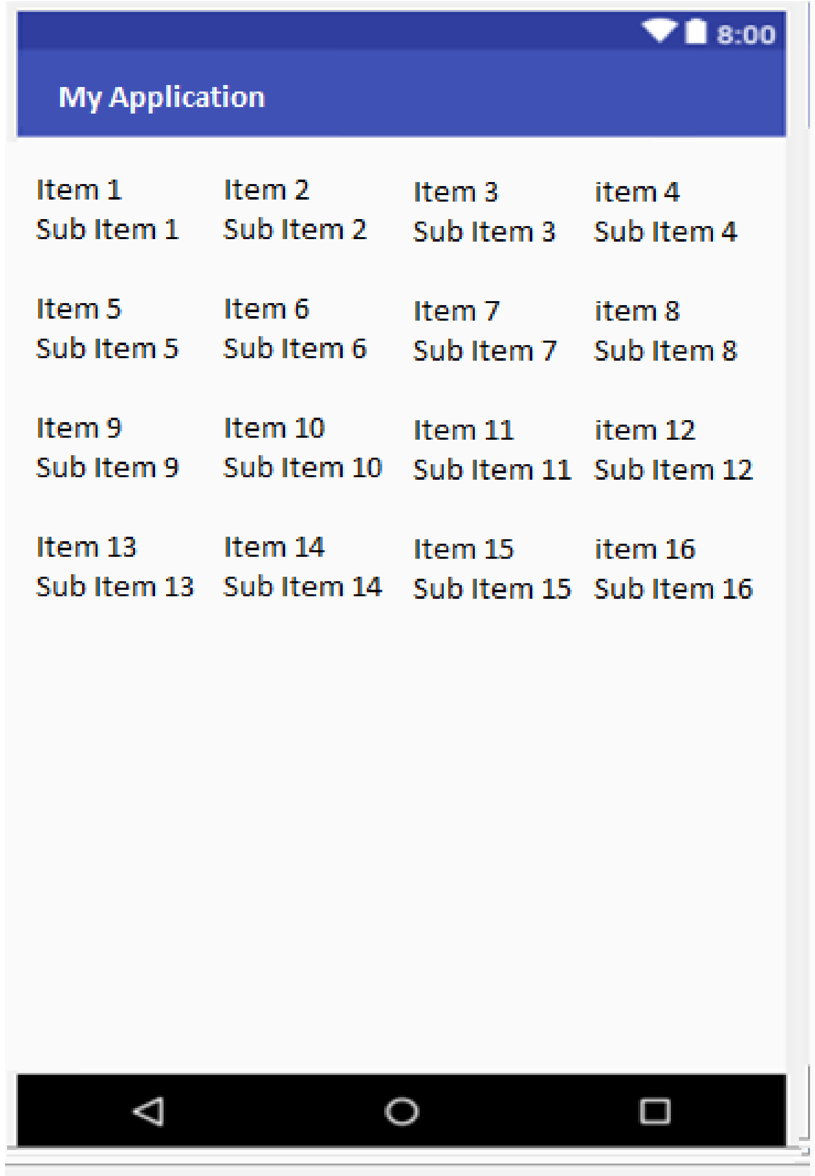
Children occupy one or more contiguous cells, as defined by their **rowSpec** and **columnSpec** layout parameters. Each spec defines the set of rows or columns that are to be occupied; and how children should be aligned within the resulting group of cells. Although cells do not normally overlap in a GridLayout, GridLayout does not prevent children being defined to occupy the same cell or group of cells. In this case however, there is no guarantee that children will not themselves overlap after the layout operation completes.

Default Cell Assignment

If a child does not specify the row and column indices of the cell it wishes to occupy, **GridLayout** assigns cell locations automatically using its: orientation, **rowCount** and **columnCount** properties.

UI Layouts and UI Elements

(Continued) Grid Layout



8:00

My Application

Item 1 Sub Item 1	Item 2 Sub Item 2	Item 3 Sub Item 3	item 4 Sub Item 4
Item 5 Sub Item 5	Item 6 Sub Item 6	Item 7 Sub Item 7	item 8 Sub Item 8
Item 9 Sub Item 9	Item 10 Sub Item 10	Item 11 Sub Item 11	item 12 Sub Item 12
Item 13 Sub Item 13	Item 14 Sub Item 14	Item 15 Sub Item 15	item 16 Sub Item 16

UI Layouts and UI Elements

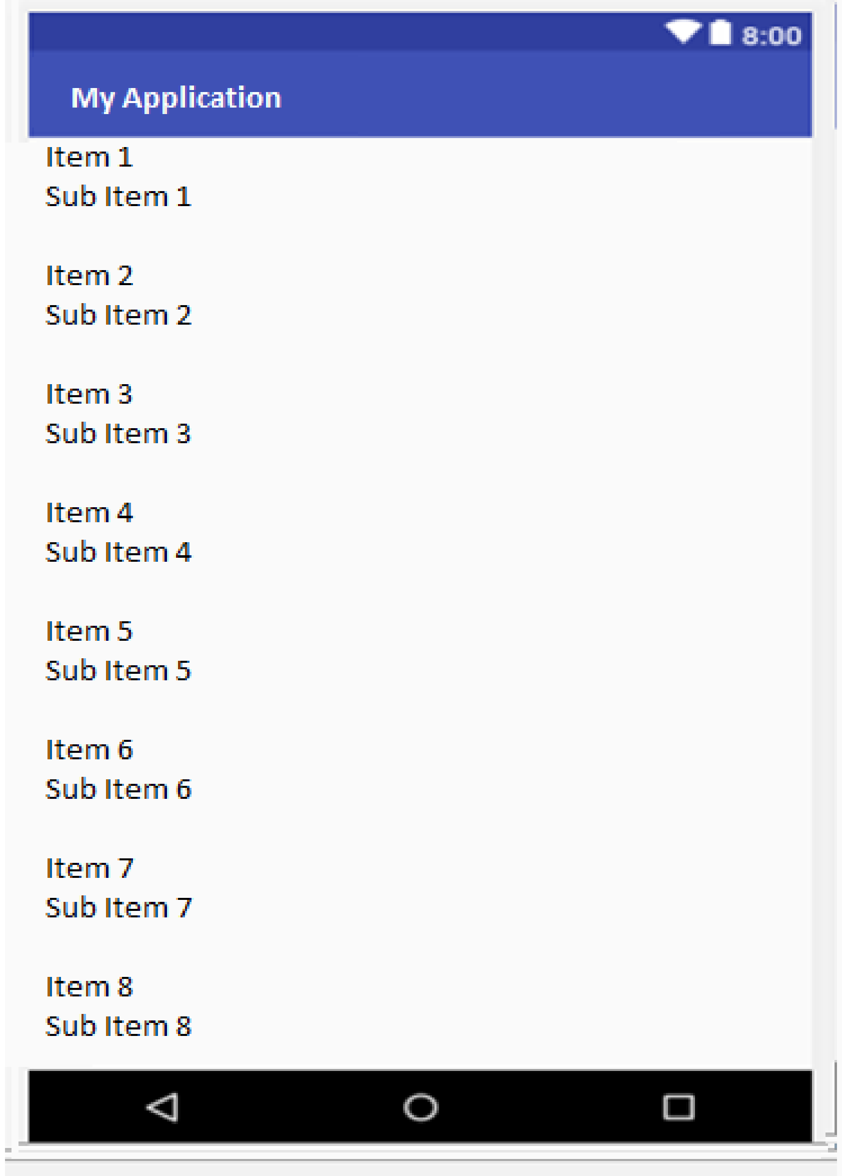
Code to define GridLayout in Your XML

```xml
<?xml version="1.0" encoding="utf-8"?>
<GridView
xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/grid_view"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:numColumns="auto_fit"
    android:columnWidth="90dp"
    android:horizontalSpacing="10dp"
    android:verticalSpacing="10dp"
    android:gravity="center"
    android:stretchMode="columnWidth" >
</GridView>
```

UI Layouts and UI Elements

ListView

- This ViewGroup displays a list of items that are scrollable.



My Application

Item 1
Sub Item 1

Item 2
Sub Item 2

Item 3
Sub Item 3

Item 4
Sub Item 4

Item 5
Sub Item 5

Item 6
Sub Item 6

Item 7
Sub Item 7

Item 8
Sub Item 8

UI Layouts and UI Elements

Code to define ListView in Your XML

```
<?xml version="1.0" encoding="utf-8"?>
<ListView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/list_item"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:numColumns="auto_fit"
    android:columnWidth="90dp"
    android:horizontalSpacing="10dp"
    android:verticalSpacing="10dp"
    android:gravity="center"
    android:stretchMode="columnWidth" >
</ListView>
```

UI Layouts and UI Elements

View

Views are the basic building blocks used to design the components needed for a screen, of an application.

Examples of View:

- **Button**: Used for click event by the user.
- **TextField**: Used to fetch answers from the user.
- **RadioButton**: Used to fetch yes or no type answers from the user. Only one choice can be selected.
- **Toggle Button**: It is like fan switches. Allows user to turn on or off.
- **CheckBox**: Used to provide the user multiple options to select their choice on own.
- **Spinner**: Used to provide user a range of options to select in drop down format.
- **ImageSwitcher**: Used to auto scroll or provide animation while displaying images.

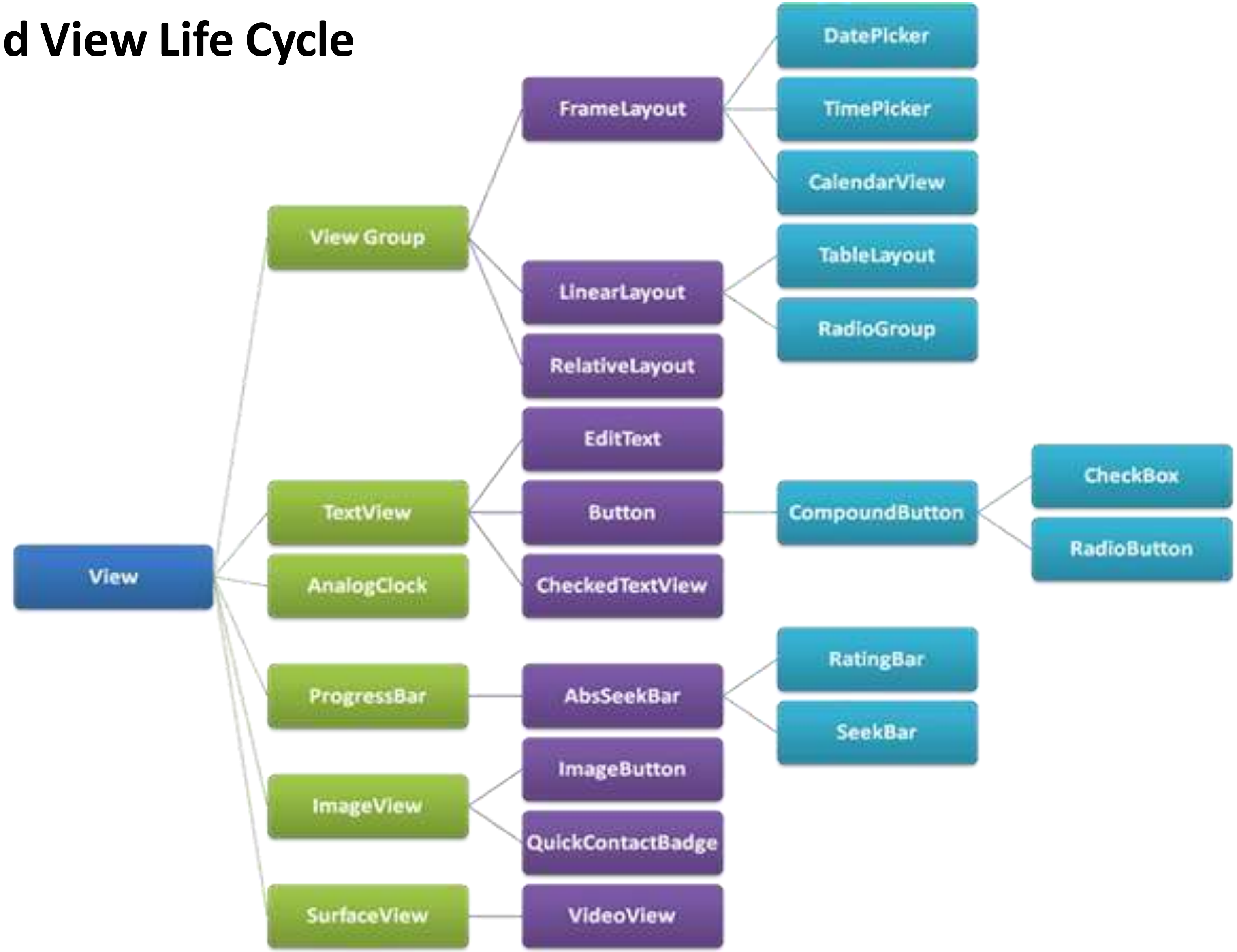
(Continued) View

- The basic unit for the user interface is a View / View object.
- The View object is created from the View class.
- It occupies a rectangular area on the screen.
- It is on this screen that there are the drawing and event handling.
- The View class is the base class for the widgets that are used to create interactive User Interface components like the buttons, labels, text fields.

A view can also be comprised of multiple other views (otherwise known as a composite view). Such views are subclassed from the Android **ViewGroup** class (**android.view.ViewGroup**) which is itself a subclass of View. An example of such a view is the **RadioGroup**, which is intended to contain multiple **RadioButton** objects such that only one can be in the “on” position at any one time. In terms of structure, composite views consist of a single parent view (derived from the ViewGroup class and otherwise known as a container view or root element) that is capable of containing other views (known as child views).

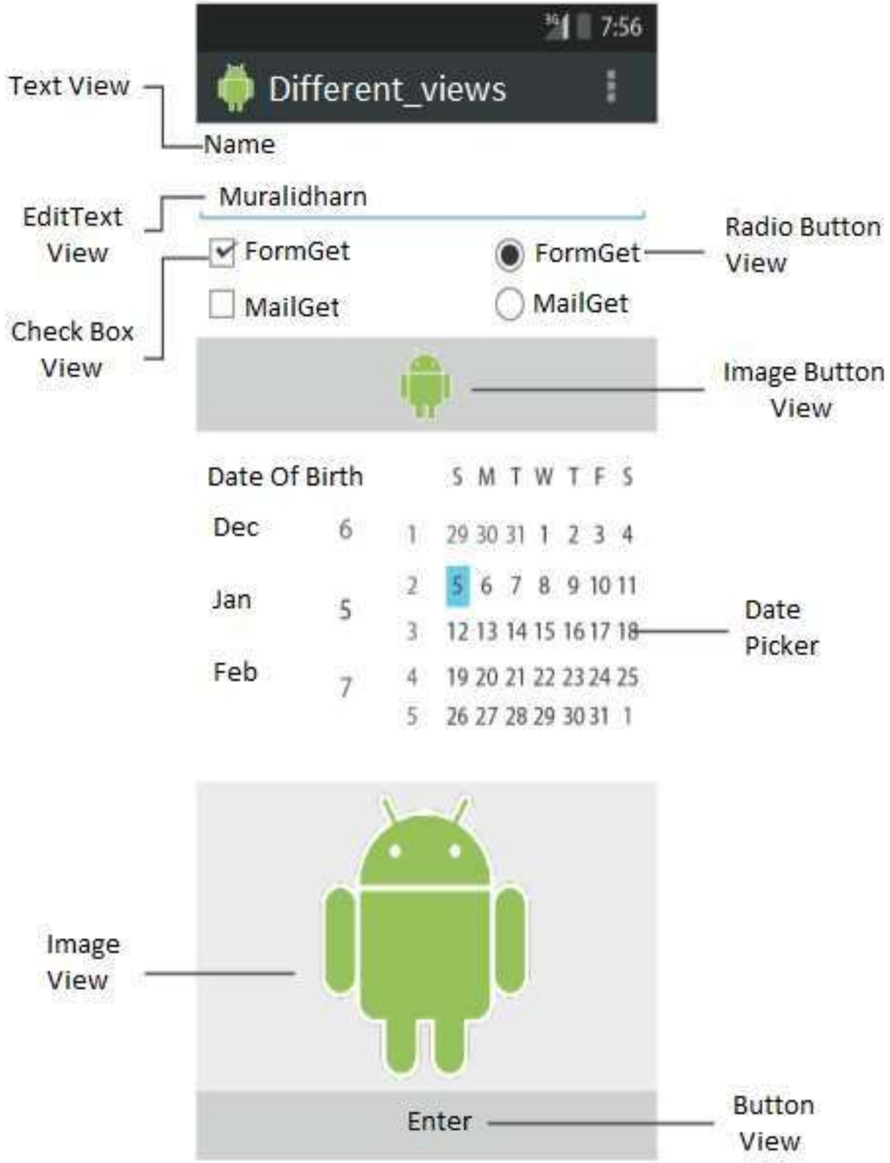
UI Layouts and UI Elements

Android View Life Cycle



UI Layouts and UI Elements

Different Views



7:56

Different_views

Name

Muralidharn

☑ FormGet ● FormGet

□ MailGet ○ MailGet

Date Of Birth

		S	M	T	W	T	F	S
Dec 6	1	29	30	31	1	2	3	4
Jan 5	2	5	6	7	8	9	10	11
	3	12	13	14	15	16	17	18
Feb 7	4	19	20	21	22	23	24	25
	5	26	27	28	29	30	31	1

Enter

UI Layouts and UI Elements

TextView

A **TextView** displays text to the user and optionally allows them to edit it. A TextView is a complete text editor, however the basic class is configured to not allow editing..

TextView declararation in XML file:

```
<TextView
    android:id="@+id/textView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/imageView2"
    android:layout_centerHorizontal="true"
    android:shadowColor="White"
    android:textStyle="bold"
    android:text="Welcome to Android"
    android:textColor="@color/white"
    android:textSize="30sp" />
```

UI Layouts and UI Elements

EditText / Plain Text

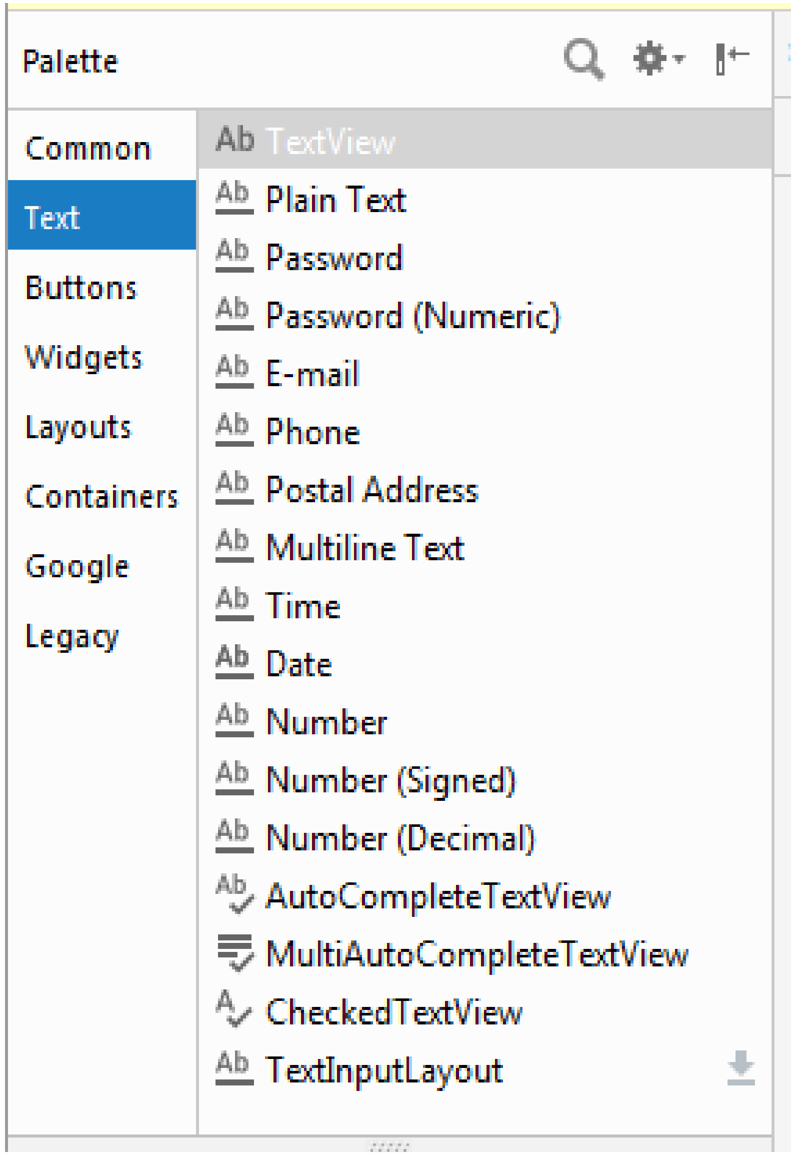
A **TextView** displays text to the user and optionally allows them to edit it. A TextView is a complete text editor, however the basic class is configured to not allow editing..

EditText declararation in XML file:

```
<EditText
    android:id="@+id/editText"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:inputType="textPersonName"
    android:hint="Name"
    android:textSize="20sp"
    android:textColor="@color/colorPrimaryDark"/>
```

UI Layouts and UI Elements

Text/EditText Options



UI Layouts and UI Elements

TextView/ EditText Attributes

Attribute	Description
android:id	This is the ID which uniquely identifies the control.
android:capitalize	If set, specifies that this TextView has a textual input method and should automatically capitalize what the user types. • Don't automatically capitalize anything - 0 • Capitalize the first word of each sentence - 1 • Capitalize the first letter of every word - 2 • Capitalize every character – 3
android:cursorVisible	Makes the cursor visible (the default) or invisible. Default is false.
android:editable	If set to true, specifies that this TextView has an input method.
android:fontFamily	Font family (named by string) for the text.
android:gravity	Specifies how to align the text by the view's x- and/or y-axis when the text is smaller than the view.
android:hint	Hint text to display when the text is empty.
android:inputType	The type of data being placed in a text field. Phone, Date, Time, Number, Password etc.
android:maxHeight	Makes the TextView be at most this many pixels tall.

UI Layouts and UI Elements

(Continued) TextView Attributes

Attribute	Description
android:minHeight	Makes the TextView be at least this many pixels tall.
android:minWidth	Makes the TextView be at least this many pixels wide.
android:password	Whether the characters of the field are displayed as password dots instead of themselves. Possible value either "true" or "false".
android:phoneNumber	If set, specifies that this TextView has a phone number input method. Possible value either "true" or "false".
android:text	Text to display.
android:textAllCaps	Present the text in ALL CAPS. Possible value either "true" or "false".
android:textColor	Text color. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
android:textColorHint	Color of the hint text. May be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
android:textIsSelectable	Indicates that the content of a non-editable text can be selected. Possible value either "true" or "false".
android:textSize	Size of the text. Recommended dimension type for text is "sp" for scaled-pixels (example: 15sp).
android:textStyle	Style (bold, italic, bolditalic) for the text. You can use or more of the following values separated by '|'. normal – 0, bold – 1, italic – 2
android:typeface	Typeface (normal, sans, serif, monospace) for the text. You can use or more of the following values separated by '|'. normal – 0, sans – 1, serif – 2, monospace - 3

UI Layouts and UI Elements

Button

A button control in android is just like any other in which when the button is clicked or pressed, leads to a corresponding action. To define a button view <Button> element is used.

Button declaration in XML file:

```
<Button
    android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Button" />
```

UI Layouts and UI Elements

Button declaration in Code

```
public class MainActivity extends Activity
{
    private Button button;
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main); //Layout where button decalred xml
        addListenerOnButton();
    }
    public void addListenerOnButton(){
        button=(Button)findViewById(R.id.button); //Assigning the button variable
        buttonSum.setOnClickListener(new OnClickListener(){ //button click event
            @Override
            public void onClick(View view) {
        // code to do when click is happened
            }
        });
    }
}
```

UI Layouts and UI Elements

Button Attributes

Below is a list of important attributes that are related to the button. Additionally, there are other attributes too and also you can change the attributes at run time. The following table

Attribute	Description
android:autoText	This attribute corrects some common spelling errors
android:drawableBottom	This attribute is used to draw the button below the text
android:drawableRight	This attribute is used to draw the button to the right of the text
android:editable	This attribute specifies that this TextView has an input method
android:text	Text to display
android:background	Set the background
android:contentDescription	Defines the text that briefly describes content of the view
android:id	Identifier names for this view
android:onClick	This specifies the name of the method to be invoked when the view is clicked
android:visibility	Controls the initial visibility of the view

Events & Listeners

Events are nothing but an action taking place in the application. They are useful to collect data about a user's interaction with interactive components of Applications. They help the application to identify the interaction level with the user. Event Handling is nothing but handling the event taken place by user interaction.

For example, In your application you have a registration form to fill and submit. You request the user to fill all the required details. Once the details have been entered, User will click on submit button. The act of clicking that button is called as event. On clicking the submit button what should happen is taken care by Event handlers.

UI Layouts and UI Elements

(Continued) Events & Listeners

Most used event handler methods in Android are

- onClick() – This handler is used when any click action happens. Ex: Submit button click
- onTouch() – This handler is used when any touch action happens. Ex: In gallery folder you touch a pic to view.

(Continued) Events & Listeners

Example:

This example shows you how to declare event handler for button click. Here once button is clicked Textview is made to display the text "Button clicked".

```
button.setOnClickListener( new OnClickListener()
{
    public void onClick(View v) { //event handler declaration
        TextView myTextView = (TextView) findViewById(R.id.myTextView);
        myTextView.setText("Button clicked");
    }
}
);
```

(Continued) Events & Listeners

Input Events

- WithAndroid, there is more than one way to intercept the events based on the user interaction with the application.
- For example, button presses, screen touches etc.
- There happens to be an Android framework that maintains the event queue as a first-in-first-out (FIFO) basis.
- These events can be captured in the program using handlers and appropriate actions can be taken.

(Continued) Events & Listeners

There are three concepts related to Android Event Management:

1. **Event Listeners -** An interface in the View class that has just one callback method. The listener methods will be called by the Android framework when the registered View of the listener has been triggered by the user interaction.

2. **Event Listeners Registration** - This is the process by which the Event Handler gets registered with the event listener that will let the handler be called whenever an event is fired by the event listener.

3. **Event Handlers -** The event listener calls the Event handlers when an event happens and we have registered an event listener for the event.

(Continued) Events & Listeners

There are three ways to register the event listener for an event:

1. By using an anonymous inner class.

2. The Activity class implements the Listener Interface.

3. By making use of the Layout activity_main.xml to specify the handler.

UI Layouts and UI Elements

(Continued) Events & Listeners

Some of the most commonly used event listeners are listed below:

Event Handler	Event Listener & Description
onClick()	OnClickListener() OnClickListener() is called when the user tries to click, focus or touch on any device like an image, button, text, etc. You will use the onClick() event handler to handle such event.
onLongClick()	OnLongClickListener() This OnLongClickListener() method is called when the user clicks or touches or focuses upon any device like button, text, image, etc. for more than one second. Then you can use the onLongClick() event handler to handle such event

(Continued) Events & Listeners

Event Handler	Event Listener & Description
onFocusChange()	OnFocusChangeListener() This method is called when the device misses its focus, i.e; the user moves away from the view item. Then you will use the onFocusChange() event handler to handle such event.
onKey()	OnKey() This method is called when the user is attentive on the item and press or release a hardware key on the device. Then you can use onKey() event handler to handle such event.
onTouch()	OnTouchListener() This method is called when the user presses or releases the key, or if there is any movement gesture on the screen. Then you can use the onTouch() event handler to handle such event.
onMenuItemClick()	OnMenuItemClickListener() This method is called when the user chooses a menu item. Then you can use the onMenuItemClick() event handler to handle such event.
onCreateContextMenu()	onCreateContextMenuItemListener() This method is called when the context menu is being built (as the result of a sustained "long click)

Image Button

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 setImageResource(int) method.

To remove the standard button background image, define your own background image or set the background color to be transparent.

```
<ImageButton
    android:id="@+id/imageButton"
    android:layout_width="100dp"
    android:layout_height="50dp"
    app:srcCompat="@drawable/b4" />
```

(Continued) Image Button

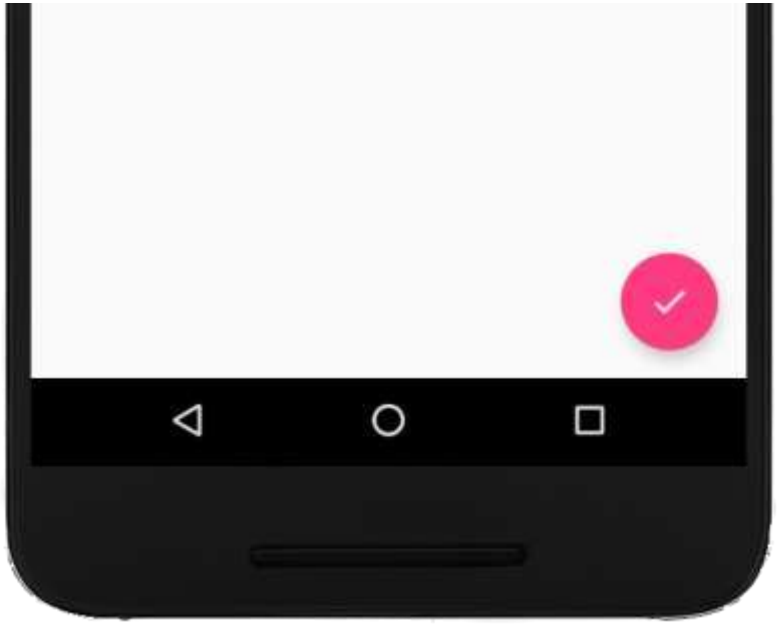
ImageButton Attributes

Attribute	Description
android:adjustViewBounds	Set this to true if you want the ImageView to adjust its bounds to preserve the aspect ratio of its drawable.
android:baseline	This is the offset of the baseline within this view.
android:baselineAlignBottom	If true, the image view will be baseline aligned with based on its bottom edge.
android:cropToPadding	If true, the image will be cropped to fit within its padding.
android:src	This sets a drawable as the content of this ImageView.

Floating Button

A floating action button (FAB) is a circular button that triggers the primary action in your app's UI.

```
<android.support.design.widget.FloatingActionButton
    android:id="@+id/fab"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_gravity="end|bottom"
    android:src="@drawable/ic_my_icon"
    android:layout_margin="16dp" />
```



UI Layouts and UI Elements

Respond to Floating button taps

```
FloatingActionButton fab = findViewById(R.id.fab);
fab.setOnClickListener(new View.OnClickListener()
{
    @Override
    public void onClick(View view) {
        Snackbar.make(view, "Here's a Snackbar",
            ackbar.LENGTH_LONG)
                .setAction("Action", null).show();
    }
});
```

AutoCompleteTextView

- An editable text view that shows completion suggestions automatically while the user is typing. The list of suggestions is displayed in a drop down menu from which the user can choose an item to replace the content of the edit box with.
- The drop down can be dismissed at any time by pressing the back key or, if no item is selected in the drop down, by pressing the enter/dpad center key.
- The list of suggestions is obtained from a data adapter and appears only after a given number of characters defined by the threshold.

UI Layouts and UI Elements

(Continued) AutoCompleteTextView

The following code snippet shows how to create a text view which suggests various countries names while the user is typing:

```
public class CountriesActivity extends Activity {
    protected void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.countries);

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_dropdown_item_1line, COUNTRIES);
        AutoCompleteTextView textView = (AutoCompleteTextView)
                findViewById(R.id.countries_list);
        textView.setAdapter(adapter);
    }

    private static final String[] COUNTRIES = new String[] {
            "Belgium", "France", "Italy", "Germany", "Spain"
    };
}
```

(Continued) AutoCompleteTextView

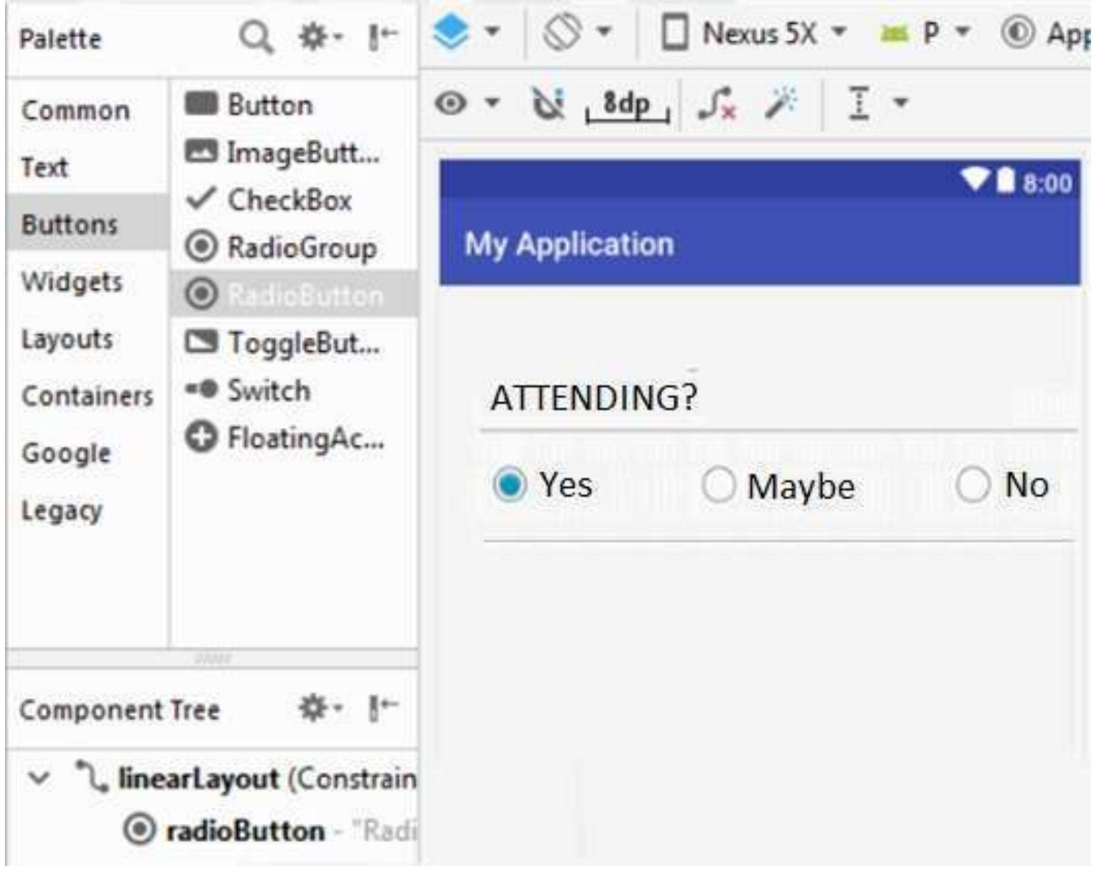
AutoCompletetextView Attributes

Attribute	Description
android:completionHint	Defines the hint displayed in the drop down menu.
android:completionHintView	Defines the hint view displayed in the drop down menu.
android:completionThreshold	Defines the number of characters that the user must type before completion suggestions are displayed in a drop down menu.
android:dropDownAnchor	View to anchor the auto-complete dropdown to.
android:dropDownHeight	Specifies the basic height of the dropdown.
android:dropDownHorizontalOffset	Amount of pixels by which the drop down should be offset horizontally.
android:dropDownSelector	Selector in a drop down list.
android:dropDownVerticalOffset	Amount of pixels by which the drop down should be offset vertically.
android:dropDownWidth	Specifies the basic width of the dropdown.
android:popupBackground	The background to use for the popup window.

RadioButton

- Radio button allows 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 is not necessary to show all options side-by-side, use a spinner instead.

```
<RadioButton
    android:id="@+id/radioButton"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:text=“Yes" />
```



RadioGroup

- RadioGroup class is used to create a multiple-exclusion scope for a set of radio buttons.
- Checking one radio button that belongs to a radio group unchecks any previously checked radio button within the same group.
- Initially, all of the radio buttons are unchecked.
- While it is not possible to uncheck a particular radio button, the radio group can be cleared to remove the checked state.
- The selection is identified by the unique id of the radio button as defined in the XML layout file.

(Continued) RadioGroup

RadioGroup Attributes

Attribute	Description
android:checkedButton	This is the id of child radio button that should be checked by default within this radio group.
android:background	This is a drawable to use as the background.
android:contentDescription	This defines text that briefly describes content of the view.
android:id	This supplies an identifier name for this view
android:onClick	This is the name of the method in this View's context to invoke when the view is clicked.
android:visibility	This controls the initial visibility of the view.

UI Layouts and UI Elements

RadioGroup Example

```
@Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      radioSexGroup=(RadioGroup)findViewById(R.id.radioGroup);
      btnDisplay=(Button)findViewById(R.id.button);
      btnDisplay.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            int selectedId=radioSexGroup.getCheckedRadioButtonId();
         radioSexButton=(RadioButton)findViewById(selectedId);
         Toast.makeText(MainActivity.this,radioSexButton.getText(),
              Toast.LENGTH_SHORT).show();
         }
      });
   }
}
```

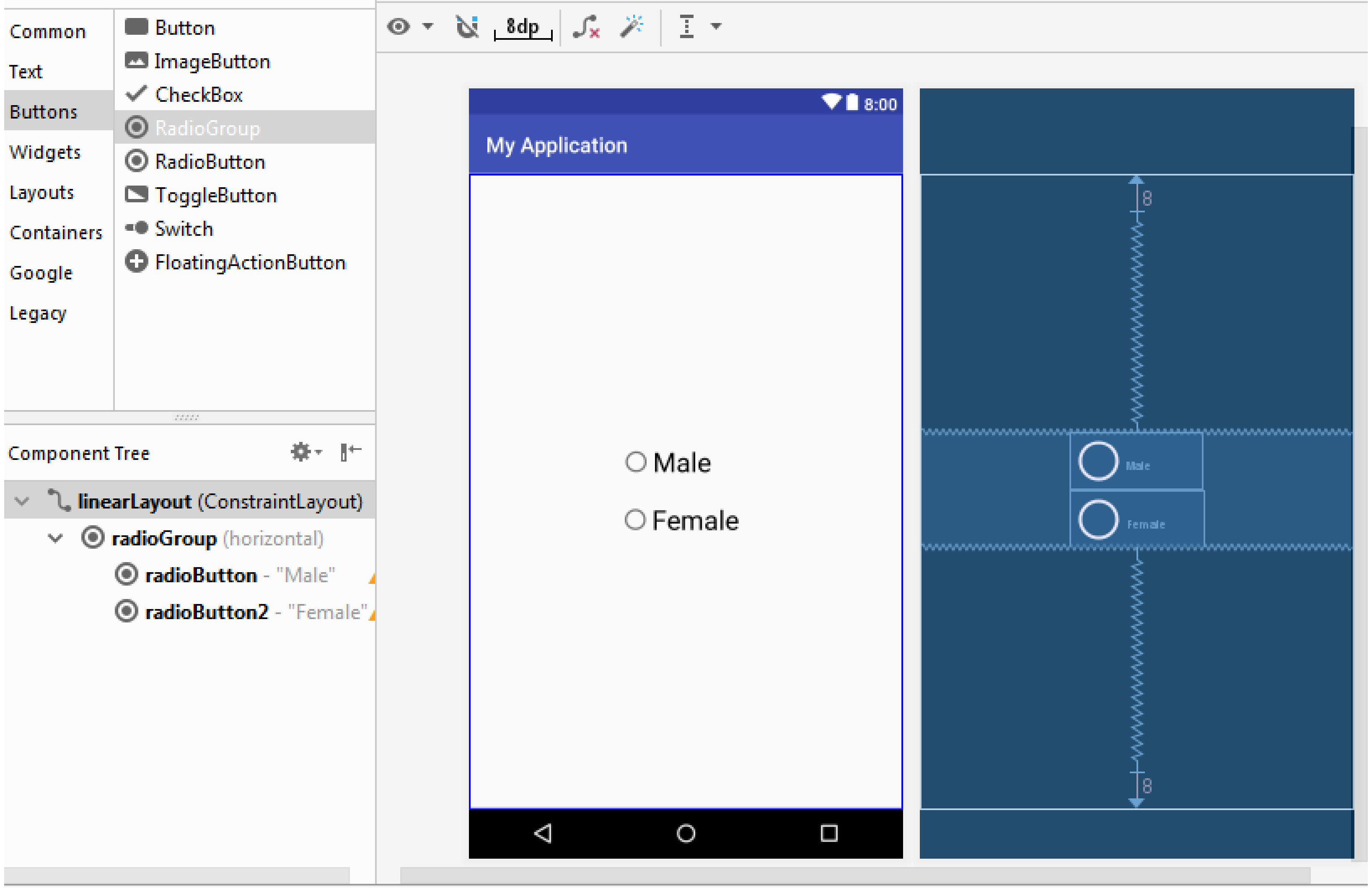
UI Layouts and UI Elements

RadioGroup Example

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/and
roid"
xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/linearLayout"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


    <RadioGroup
        android:id="@+id/radioGroup"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_marginBottom="8dp"
        android:layout_marginTop="8dp"
        android:weightSum="1"
     app:layout_constraintBottom_toBottomOf="parent"
   app:layout_constraintTop_toTopOf="parent">
```

```
        <RadioButton
            android:id="@+id/radioButton"
            android:layout_width="126dp"
            android:layout_height="55dp"
          android:layout_gravity="center_horizontal"
            android:checked="false"
            android:text="Male"
            android:textSize="25dp" />

        <RadioButton
            android:id="@+id/radioButton2"
            android:layout_width="128dp"
            android:layout_height="55dp"
          android:layout_gravity="center_horizontal"
            android:checked="false"
            android:text="Female"
            android:textSize="25dp" />
    </RadioGroup>
</android.support.constraint.ConstraintLayout>
```

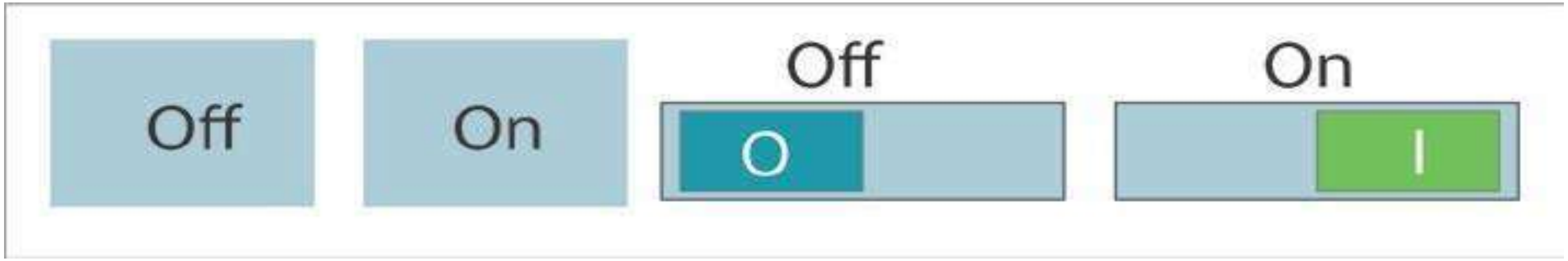
RadioGroup Example



Toggle Button

- The toggle button displays a checked or unchecked state of a button.
- This is similar to the on or off button that comes with a light indicator.
- Toggle button is defined in the xml layout using <ToggleButton> element. The following figure shows the design of Toggle Button in an application.

The basic toggle button can be added to the layout using ToggleButton object.
In case you want to change the button's state on your own then use the **CompoundButton.setChecked()** or **CompoundButton.toggle()** methods. To detect when the user activates the button or switch create the **CompoundButton.OnCheckedChangeListener** object and assign this to the button.



UI Layouts and UI Elements

(Continued) Toggle Button

```
<ToggleButton --------------------------------->Button 1 declaration for ON
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="On“ ------------>Text to be shown on button 1
android:id="@+id/toggleButton"
android:checked="true"/>

<ToggleButton --------------------------------------->Button 2 declaration for OFF
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Off"------------>Text to be shown on button 2
android:id="@+id/toggleButton"/>
```

UI Layouts and UI Elements

Code to declare Toggle Button

1. Declare a ToggleButton variable and assign it.
2. Set click listener for that toggle button
3. Write your code to perform the process when Button is ON and OFF.

```
ToggleButton toggle = (ToggleButton) findViewById(R.id.togglebutton);
toggle.setOnCheckedChangeListener(new CompoundButton.OnCheckedChangeListener()
{
 public void onCheckedChanged(CompoundButton buttonView, boolean isChecked) {
            if (isChecked)
                        { // The toggle is enabled
                        } else
                        {
                         // The toggle is disabled
                         }
            }
});
```

CheckBox

Check box is one of the viewgroup of Android which provides user to choose multiple choice or option from the provided set of choices or options.

Vertical display of options are mostly preferred while declaring checkboxes in the layout. CheckBox option is added to the XML layout using <CheckBox> element.

For multiple checkbox option, you have to create multiple CheckBox in your layout.
A set of checkbox options allows the user to select multiple items and each checkbox is managed separately.

Each checkbox should be declared with a click listener.

UI Layouts and UI Elements

(Continued) CheckBox

Check boxes are added to the layout using the Checkbox object. This can be done in the XML layout with the <Checkbox> element.

Syntax:

```
<CheckBox >..............................................</CheckBox>
```

Code declaration:

1. Define variables needed for as many Checkboxes.
2. Assign the Layout checkbox to the declared variable.
3. Set clicklistener for the checkboxes.
4. Write your code to perform once the checkbox is clicked.

UI Layouts and UI Elements

CheckBox Example code

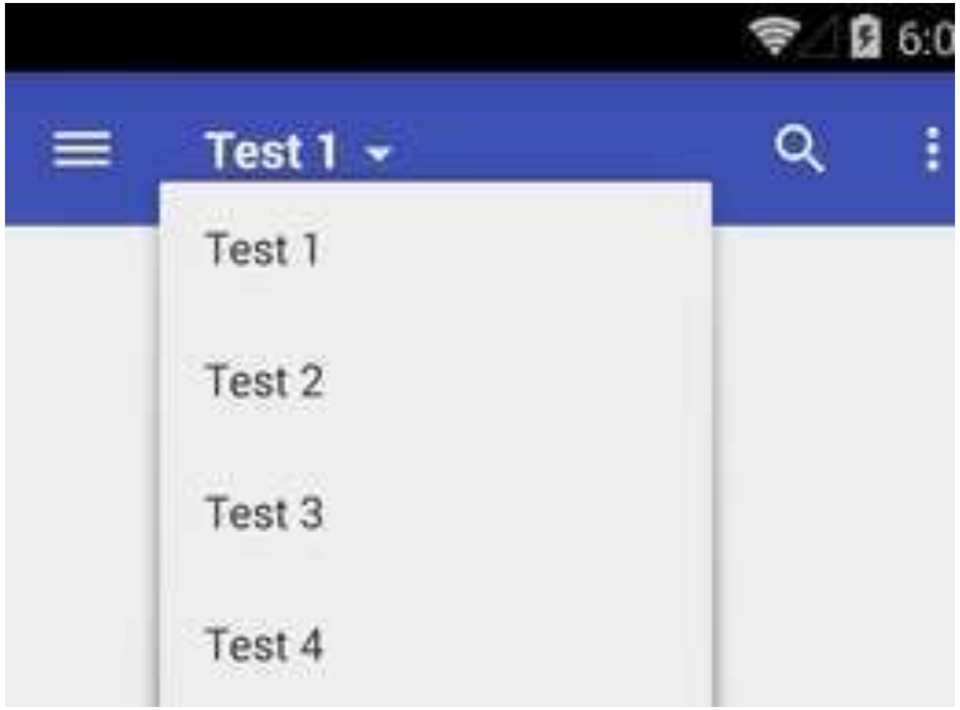
```
public class Checkbox extends Activity {
private CheckBox android, ios, windows; //Checkbox variable defined
private Button Click;

@Override
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main); //Multiple checkboxes are defined
        android = (CheckBox) findViewById(R.id.android); //Assign Layout checkbox to the declared variable
        ios = (CheckBox) findViewById(R.id. ios );
        windows = (CheckBox) findViewById(R.id. windows );
        Click= (Button) findViewById(R.id.Click);
        Click.setOnClickListener(new OnClickListener(){ //click event for the checkbox is defined
```

(Continued) CheckBox Example code

```
@Override
public void onClick(View v) { // step 4
            if(android.checked){   //this loop executed when first chexkbox is clicked
                        Toast.makeText(Checkbox.this, "android is checked"), Toast.LENGTH_LONG).show();
            }
            if(ios.checked){ //this loop executed when second checkbox is clicked
                        Toast.makeText(Checkbox.this, "ios is checked"), Toast.LENGTH_LONG).show();
            }
            if(windows.checked){    //this loop executed when third checkbox is clicked
                        Toast.makeText(Checkbox.this, "windows is checked"),
                        Toast.LENGTH_LONG).show();
            }
      }
  });
}
```

Spinner

Using spinners you can provide a quick way to select one value from a set. By default, the spinner shows the value that is currently selected. When the spinner is touched it displays a dropdown menu with all the available values and from that the user can select one. The following figure shows the dropdown menu of the spinner.



(Continued) Spinner

The spinner can be added to the layout using the Spinner object. This can be done in the XML layout with the <Spinner> element

Spinner declaration in XML layout:

```
<Spinner android:id="@+id/planets_spinner"
android:layout_width="fill_parent"
android:layout_height="wrap_content" />
```

UI Layouts and UI Elements

(Continued) Spinner

1.Declare the spinner items in your resource file:

```
<string-array name="planets_array">
<item>Test1</item>
<item>Test2</item>
<item>Test3</item>
<item>Test4</item>
</string-array>
```

2.Declaration in your class file:

```
Spinner spinner = (Spinner) findViewById(R.id.spinner);
// Create an ArrayAdapter using the string array and a default spinner layout
ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this, R.array.planets_array,
android.R.layout.simple_spinner_item); // Specify the layout to use when the list of choices appears
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
// set the adapter to the spinner
spinner.setAdapter(adapter);
```

ProgressBar

- ProgressBar is a user interface element that indicates the progress of an operation.
- ProgressBar supports two modes to represent progress:
  - **Determinate**
  - **Indeterminate**
- For a visual overview of the difference between determinate and indeterminate progress modes, see Progress and activity.
- Display ProgressBars 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.

UI Layouts and UI Elements

(Continued) ProgressBar

- **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" />
```

UI Layouts and UI Elements

(Continued) ProgressBar

- **Determinate Progress**

  To indicate determinate progress, you set the style of the progress bar to 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"/>
```

(Continued) ProgressBar

- 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

Toast

Toast is a quick message displayed in the screen for a short duration of time and fades out automatically.

Example: when we set an alarm, it says a message in a floating view 'Alarm set to ring after 2 hours' and so. We can specify the time duration while creating a toast message. A toast can be displayed from a service and activity.

(Continued) Toast

Placing your Toast message

Below is a sample piece of code that shows how to create and place a toast message in Andoird:

Placing your toast in a message

```
//display in short period of time
Toast.makeText(getApplicationContext(), "Toast Example", Toast.LENGTH_SHORT).show();
//display in long period of time
Toast.makeText(getApplicationContext(), "Toast Example", Toast.LENGTH_LONG).show();
//display for the time specified in millisecond
Toast.makeText(getApplicationContext(), "msgmsg", 2000).show();
```

Alert Dialogs

Dialog

Dialog is a small floating window that pop's up in front of the activity, it cannot exist independently it has to be embedded inside an activity or a fragment. We should use a dialog when we need to show a progress bar to show the status of progress or a short message that requires a confirmation from the user (such as an alert with 'ok' or 'cancel' buttons).

The Dialog class is the base class for dialogs, but you should avoid instantiating Dialog directly. Instead, use one of the following subclasses:

- **Alert dialog**
- **DatePickerDialog or TimePickerDialog**

(Continued) Alert Dialogs

- **Alert Dialog**

Alert dialog is a dialog that we commonly see in our application that shows a title, up to three buttons, a list of selectable items, or a custom layout. For example when we try to delete some messages from our message box we get a confirmation dialog with two buttons to cancel or delete this is an example of alert dialog.

Syntax:

```
AlertDialog.Builder alertDialogBuilder = new AlertDialog.Builder(this);
```

UI Layouts and UI Elements

(Continued) Alert Dialogs

Setting Positive and Negative Buttons to Alert Dialog

Setting positive (yes) or negative (no) button using the object of the AlertDialogBuilder class.

Syntax:

```
alertDialogBuilder.setPositiveButton(CharSequence text, DialogInterface.OnClickListener listener)
alertDialogBuilder.setNegativeButton(CharSequence text, DialogInterface.OnClickListener listener)
```

Custom Alert Dialog

The custom dialog uses DIALOG to create custom alert in android studio. Dialog displays a small window i.e., a popup which draws the user attention over the activity before they continue moving forward. The dialog appears over the current window and display the content defined in it.

AlertDialog Vs Custom AlertDialog:

The Alert Dialog and Custom Alert Dialog both prompts a small window to make decision. The AlertDialog makes use of the defined components or methods like setIcon, setTitle, setmessage etc., but with Custom AlertDialog we can have the dialog customised and can define the layout of dialog as required.

(Continued) Custom Alert Dialog

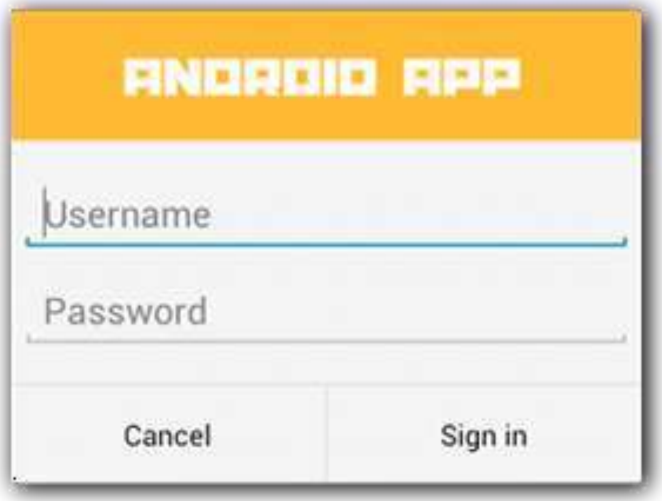
Create Custom Alert Dialog

- Create a layout.
- Add layout to an AlertDialog by calling setView() on your **AlertDialog.Builder** object.
- By default, the custom layout fills the dialog window, but you can still use **AlertDialog.Builder** methods to add buttons and a title.

UI Layouts and UI Elements

Custom Alert Dialog Example (Layout design)

```
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content">
  <ImageView
    android:src="@drawable/header_logo"
    android:layout_width="match_parent"
    android:layout_height="64dp"
    android:scaleType="center"
    android:background="#FFFFBB33"
    android:contentDescription="@string/app_name"/>
  <EditText
    android:id="@+id/username"
    android:inputType="textEmailAddress"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="16dp"
    android:layout_marginLeft="4dp"
    android:layout_marginRight="4dp"
    android:layout_marginBottom="4dp"
    android:hint="@string/username"/>
  <EditText
    android:id="@+id/password"
    android:inputType="textPassword"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="4dp"
    android:layout_marginLeft="4dp"
    android:layout_marginRight="4dp"
    android:layout_marginBottom="16dp"
    android:fontFamily="sans-serif"
    android:hint="@string/password"/>
</LinearLayout>
```



Create Custom Alert Dialog Example Control Section

```
@Override
public Dialog onCreateDialog(Bundle savedInstanceState) {
    AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
    // Get the layout inflater
    LayoutInflater inflater = getActivity().getLayoutInflater();
    // Inflate and set the layout for the dialog
    // Pass null as the parent view because its going in the dialog layout
    builder.setView(inflater.inflate(R.layout.dialog_signin, null))
    // Add action buttons
           .setPositiveButton(R.string.signin, new DialogInterface.OnClickListener() {
               @Override
               public void onClick(DialogInterface dialog, int id) {
                   // sign in the user ...
               }
           })
           .setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
               public void onClick(DialogInterface dialog, int id) {
                   LoginDialogFragment.this.getDialog().cancel();
               }
           });
    return builder.create();
}
```

Self Assessment Questions

1. The basic unit for the User Interface (UI) is _______________.

   a. View

   b. ViewGroup

   c. FrameLayout

   d. View subclass

Answer: **View.**

Self Assessment Questions

2. Which one of the given options is used to create View Object?

a. Viewclass

b. Viewgroup

c. Framelayout

d. View Subclass

Answer: **Viewclass**

Self Assessment Questions

3. Which one of the given options is used to create the visual structure for the userinterface?

a. View

b. Layout

c. FrameLayout

d. View subclass

Answer: **Layout**

Self Assessment Questions

4. The____________attribute specifies the position of the child view in the layout.

   a. android:layout_text

   b. android:layout_gravity

   c. android:layout_name

   d. android:layout_size

Answer: **android:layout_gravity**

Self Assessment Questions

5. Which one of the given options is used to specify the exact location of the childrenviews?

  a. Table layout
  b. Absolute layout
  c. Relative layout
  d. Linear layout

Answer: **Absolute layout**

Self Assessment Questions

6. Which one of the given options displays the items in a two-dimensional scrollable View?

   a. Grid View

   b. Table View

   c. Relative View

   d. Linear View

Answer: **Grid View**

Self Assessment Questions

7. Which one of the below Android UI Control displays the checked or unchecked state of the button?

a. Radio Button

b. Checkbox

c. Toggle button

d. Text field

Answer: **Toggle button**

Self Assessment Questions

8. ________________is a quick way to select one value from a set of values in Android user controls.

a. Toggle button

b. Radio Button

c. Spinner

d. Checkbox

Answer: **Spinner**

Self Assessment Questions

9. Which one of the given options is the identifier name for View?

a. android:gravity
b. android:id
c. android:name
d. android:text

Answer: **android:id**

Self Assessment Questions

10. Which one of the given options is used to maintain the event queues in a FIFO basis?

a. android:gravity

b. Android Activity

c. Android framework

d. android:text

Answer: **Android framework**

Self Assessment Questions

11. The events are captured in the program using________and then the appropriate actions can be taken.

a. handlers

b. receivers

c. viewgroups

d. intents

Answer: **handlers**

Self Assessment Questions

12. Which one of the given options is an interface in the View Class that has a callback method?

a. Event handlers

b. Event receivers

c. Event listeners

d. intents

Answer: **Event listeners**

Self Assessment Questions

13. Which one of the given options is a process in which the Event Handler gets registered with the event listener?

a. Event Listener Registration
b. Event receivers
c. Event Registration
d. intents

Answer: **Event Listener Registration**

Self Assessment Questions

14. In how many ways can we register the event listener for an event?

a. 1

b. 2

c. 3

d. 4

Answer: **3**

Self Assessment Questions

15. Which one of the below given activity class implements the interface?

.

a. Listener

b. receivers

c. Register

d. intents

Answer: **Listener**

Self Assessment Questions

16. Which one of the given options describes Dialog class in android?

i. ProgressDialog
ii. DatePickerDialog
iii. AlertDialog

a. Only i
b. Only i and ii
c. Only ii and iii
d. All i, ii and iii

Answer: **All i, ii and iii**

Self Assessment Questions

17. Which one of the given options are the subclasses of Layout?

a. Group

b. Layout

c. ViewGroup

d. None of these

Answer: **ViewGroup**

Self Assessment Questions

18. Which one of the given options is the main attribute of linear layout to arrange views?

a. Layout

b. Gravity

c. Orientation

d. Margin

Answer: Orientation

Self Assessment Questions

19. Constraint Layout is available as a support library that can used onAndroid systems starting with ___________.

    a. API level 13
    b. API level 9
    c. API level 18
    d. API level 8

Answer: **API level 9.**

Self Assessment Questions

20. CustomAlertDialog Adds layout to an AlertDialog by calling__________on your AlertDialog.Builder object.

   a. View()

   b. setView()

   c. SetData()

   d. SetContent()

Answer: **SetView()**

Assignment

General Instructions:

Please answer the below set of questions. These set of questions are meant for testing unit 2.

- *The answers should be clear, legible and well presented.*
- *Illustrate your answers with suitable examples wherever necessary.*
- *Please quote sources (if any) of data, images, facts etc.*

1. What is the difference between a View and ViewGroup?
2. List out the different types of layouts and explain each of them with suitable diagrams.
3. Explain in detail about each of the Android UI controls.
4. Explain TextView and different types of TextView.
5. Explain briefly about Events & Listeners.

Assignment

6. What is AutoCompleteTextView? Explain the advantages of AutoCompleteTextView.
7. What is Toast?
8. What is Alert Dialogs, and Custom Alert Dialog. Explain with examples.
9. Describe the process of Event Listener Registration.
10. Create one simple Application using all UI Elements.

UI Layouts and UI Elements

Summary

- ✓ Classifying Layout, Linear Layouts, Relative Layout, Table Layout, Constraint Layout.
- ✓ The basic unit of the user interface is the View object.
- ✓ The View object is created from the View class.
- ✓ Working on all UI Components like TextView, EditText, Button, Image Button, floating Button, AutoCompleteTextView, RadioButton, RadioGroup, ToogleButton, CheckBox, Spinner, ProgressBar. Toast, Alert Dialogs, Custom Alert Dialog.
- ✓ Working on Events & Listeners.

UI Layouts and UI Elements

Document Links

Topics	URL	Notes
Layout, Linear Layouts, Relative Layout, Table Layout, Constraint Layout, View, TextView, EditText, Button, Events & Listeners, Image Button, floating Button, AutoCompleteTextView, RadioButton, RadioGroup, ToogleButton, CheckBox, Spinner, ProgressBar. Toast, Alert Dialogs, Custom Alert Dialog.	https://www.tutorialspoint.com/android/index.htm https://developer.android.com/index.html	This link explains About all concepts of Layout, Linear Layouts, Relative Layout, Table Layout, Constraint Layout, View, TextView, EditText, Button, Events & Listeners, Image Button, floating Button, AutoCompleteTextView, RadioButton, RadioGroup, ToogleButton, CheckBox, Spinner, ProgressBar. Toast, Alert Dialogs, Custom Alert Dialog.

UI Layouts and UI Elements

Video Links

Topics	URL	Notes
Layout, Linear Layouts, Relative Layout, Table Layout, Constraint Layout, View, TextView, EditText, Button, Events & Listeners, Image Button, floating Button, AutoCompleteTextView, RadioButton, RadioGroup, ToogleButton, CheckBox, Spinner, ProgressBar. Toast, Alert Dialogs, Custom Alert Dialog.	https://www.tutorialspoint.com/android_online_training/index.asp	This link explains About all concepts of Layout, Linear Layouts, Relative Layout, Table Layout, Constraint Layout, View, TextView, EditText, Button, Events & Listeners, Image Button, floating Button, AutoCompleteTextView, RadioButton, RadioGroup, ToogleButton, CheckBox, Spinner, ProgressBar. Toast, Alert Dialogs, Custom Alert Dialog.

UI Layouts and UI Elements

E-Book Links

Topics	URL	Page Number
Layout, Linear Layouts, Relative Layout, Table Layout, Constraint Layout, View, TextView, EditText, Button, Events & Listeners, Image Button, floating Button, AutoCompleteTextView, RadioButton, RadioGroup, ToogleButton, CheckBox, Spinner, ProgressBar. Toast, Alert Dialogs, Custom Alert Dialog.	http://yuliana.lecturer.pens.ac.id/Android/Buku/professional_android_4_application_development.pdf	Page Number 95 to Page Number 106
	https://www.manning.com/books/android-in-action-third-edition	Chapter 3
	ftp://ftp.micronet-rostov.ru/linux-support/books/programming/Mobile-Apps/[Wiley.%20Wrox]%20-%20Beginning%20Android%20Application%20Development%20-%20[Wei-Meng%20Lee].pdf	Page Number 81 to Page Number 169