Nathan Warner

Computer Science
Northern Illinois University
United States

Contents

Android Development with Java and XML

1.1   The Basics

- **Layout file for apps "look"**: Note that the XML layout file for our app's "look" is named activity_main.xml

- **Creating a new project**: Click on the Empty Views Activity template to highlight it and click Next

    1. Enter the name of the project
    2. Enter the package name as edu.niu.zid.projectname
    3. Set language to Java
    4. Choose API 22 ("Lollipop"; Android 5.1) for the minimum SDK
    5. Click finish

- **Directory structure**: please be sure that you have no additional layers than what you see here.

Notice at the highest level in the hierarchy of our project folders that there are two main "pieces" to the Android project:

1. app
2. Gradle Scripts

Within the app, there are really three folders we are going to be concerned with for now:

1. manifests
2. java
3. res

- **Manifests**: The directory named manifests holds one xml file - so far. It is the AndroidManifest.xml file.

- **Java (directory):** The directory named java holds one Java source files - so far. It is the MainActivity.java file

- **Res (directory)**: The directory named res holds resource files, such as utility xml files for defining strings, themes, menus, layouts, dimensions, images, sounds, etc

- **MainActivity.java**: This is the logic/controller file. It's the Java class that defines what your app does when it runs. It extends AppCompatActivity (or another activity class), and inside it you set up event handling, lifecycle methods (like onCreate), and the code that reacts to user interactions. For example, if a button is clicked, the code for what happens lives here.

The simplest MainActivity.java file is as follows

```
0   package YOURPACKAGENAME
1
2   import androidx.appcompat.app.AppCompatActivity;
3
4   import android.os.Bundle;
5
6   public class MainActivity extends AppCompatActivity {
7
8       @Override
9       protected void onCreate(Bundle savedInstanceState) {
10          super.onCreate(savedInstanceState);
11          setContentView(R.layout.activity_main);
12      }
13  }
```

In this code, the most important import statement is the import for class android.appcompat.app.AppCompatActivity

MainActivity.java class extends, or inherits from, superclass AppCompatActivity class found in Android API package androidx.appcompat.app.AppCompatActivity

AppCompatActivity itself is a subclass itself of Activity.

An Activity provides the screen with which users can interact, in other words, the User Interface, or UI.

The MainActivity class is meant to be used as the Controller for every app we develop

Method onCreate() of an Android app is called automatically upon launch just like the main() method is called automatically by the Java Virtual Machine, or JVM, upon launch of a Java applications.

Inside method onCreate(), we create the initial View for the app, controlled by this Activity.

In method onCreate(), the superclass' onCreate() method is first called.

Then the View, or opening screen, for this Activity is set by calling method setContentView().

Method setContentView() is inherited from the AppCompatActivity , or Activity class.

Its API is

```
o    void setContentView(int layoutResID)
```

layoutResID is a resource found in the subfolder named layout in the folder named res.

This ID defaults to: *R.layout.activity_main.*

In MainActivity.java, we refer to and access activity_main.xml using the syntax

```
o    R.layout.activity_main
```

activity_main is a static constant of the static final class layout defined inside the R.java class or file.

The R.java file was automatically created when we began our project

R stands for resources and "represents" the res directory.

- **activity_main.xml**: This is the UI/layout file. It defines the visual structure of your screen using XML: buttons, text fields, images, layouts, etc. You don't put behavior here—only the arrangement and styling of the interface elements.

  activity_main is an xml file, a resource located in the layout subdirectory of the res directory

  It was also automatically created when we began our project using the template we chose.

  We can think of XML as a markup language similar to HTML, but with user-defined tags

- **Intro to design editor**: With the activity_main.xml file focused, click on this small button near the upper right

Note that this opens the Design editor in which we can drag and drop widgets into our app. More on this to come!

- **Intro to XML syntax**: In XML, an non-empty element uses the general syntax shown here

  - `<tagName attribute1="value1">Element Content</tagName>`

  For example:

  - `<color name="red">#FFFF0000</color>`

  which can be interpreted as there is an element color that has the name red and corresponds to the RGB value #FFFF0000.

  If an element has no content yet or will never have any content, we can use this syntax

  - `<tagName attribute1="value1" attribute2="value2".../>`

- **XML Naming Rules**: XML elements must follow these naming rules:

  1. Names can contain letters, numbers, and other characters.
  2. Names cannot start with a number or punctuation character.
  3. Names cannot start with the letters xml (or XML, or Xml, etc).
  4. Names cannot contain spaces.

- **XML Comments**: Documentation in XML is done with

  - `<!-- comment text -->`

- **Simple activity_main.xml Example, TextView and ConstraintLayout**: Let's consider a simple XML example

10

```
0   <?xml version="1.0" encoding="utf-8"?>
1   <androidx.constraintlayout.widget.ConstraintLayout xmlns:and⌋
    ↪  roid="http://schemas.android.com/apk/res/android"
2       xmlns:app="http://schemas.android.com/apk/res-auto"
3       xmlns:tools="http://schemas.android.com/tools"
4       android:layout_width="match_parent"
5       android:layout_height="match_parent"
6       tools:context=".MainActivity">
7
8       <TextView
9           android:layout_width="wrap_content"
10          android:layout_height="wrap_content"
11          android:text="@string/app_name"
12          app:layout_constraintBottom_toBottomOf="parent"
13          app:layout_constraintEnd_toEndOf="parent"
14          app:layout_constraintStart_toStartOf="parent"
15          app:layout_constraintTop_toTopOf="parent" />
16
17  </androidx.constraintlayout.widget.ConstraintLayout>
```

There are two elements in our current activity_main.xml:

1. ConstraintLayout
2. TextView

The TextView element is nested inside the ConstraintLayout element

A ConstraintLayout allows us to position and size elements in a flexible way.

One way is relative positioning, or where an element is placed on the screen, or view, relative to another.

The ConstraintLayout element has several attributes.

The android:layout_width and android:layout_height attributes define the size of the ConstraintLayout.

Their value match_parent means that it will be as big as its parent element, which is the screen of the Android device in this case.

The TextView element is empty, or, in other words, has no content, and has, by default, seven attributes.

The first three are layout_width, layout_height and text.

The android:text attribute specifies the content, or the words, of the TextView to be shown on the screen of the Android device.

By default, the four attributes that follow layout_width, layout_height and text are for positioning of the TextView within the ConstraintLayout that will be discussed later

11

A TextView element is an instance of the TextView class, which puts a label on the screen.

The android:layout_width and android:layout_height attributes define the size of the TextView.

The value wrap_content means that it will be as small as possible so that their contents (the text or words) fit inside it.

In other words, the element "wraps" around its content.

By default, the TextView appears in the middle of the screen

The four app:layout attributes specify the relative position of the TextView element's four sides

In this example, they are relative to the "parent", or the ConstraintLayout in which the Textview is nested

- **strings.xml**: is the resource file for text values in your app. Stores text like titles, labels, button names, or messages as key–value pairs. keep all text centralized for easy updates, consistency, and support for localization

You reference them in code (getString(R.string.welcome_message)) or XML (@string/welcome_message).

One string is defined in the strings.xml file by default

```
0    <resources>
1        <string name="app_name">Hello, Android!</string>
2    <resources>
```

The syntax for defining a string is

```
0    <string name="stringName">valueOfString</string>
```

- **themes.xml**: defines the overall visual style of your Android app.

Central place to configure app-wide appearance, like colors, fonts, backgrounds, status bar style, and widget looks.

A theme is a collection of style attributes applied globally.

- **themes.xml example**:

```
0    <resources xmlns:tools="http://schemas.android.com/tools">
1        <!-- Base application theme. -->
2
3        <style name="Base.Theme.MyApplication"
     ↪  parent="Theme.Material3.DayNight.NoActionBar">
4            <!-- Customize your light theme here. -->
5            <!-- <item name="colorPrimary">@color/my_light_prim⌋
     ↪  ary</item> -->
6        </style>
7
8        <style name="Theme.MyApplication"
     ↪  parent="Base.Theme.MyApplication" />
9    </resources>
```

It is used to define styles that the app uses.

We can modify a theme by adding an item element using this syntax

```
0    <item name="styleAttribute">valueOfItem</item>
```

The name of the theme attribute that specifies the text size inside a TextView is android:textSize.

Let us change the default text size to 40 by adding the following code to the themes.xml file just above the line </style>

```
0    <item name="android:textSize">40sp</item>
```

- **Removing and adding action bar (themes.xml)**: Also, this line removes the commonly shown Action Bar:

```
0    <style name="Base.Theme.HelloAndroid"
     ↪  parent="Theme.Material3.DayNight.NoActionBar">
```

Note: The Empty Views Activity template does not provide a default Action Bar

If we change the line to

```
0    <style name="Base.Theme.HelloAndroid"
     ↪  parent="Theme.AppCompat.Light.DarkActionBar">
```

And make two other changes, we will see the Action Bar with the name of our app displayed

- **colors.xml**: The colors.xml file in the res > values directory is another automatically generated XML file.

It is used to define colors we want to use in our apps

The syntax for defining a color is

```
0    <color name="colorName">valueOfColor</color>
```

The color value is defined using hexadecimal, or base 16, notation

#FFrrggbb (uses RGB color system) where:

- rr = amount of red in color
- gg = amount of green in color
- bb = amount of blue in color

- **colors.xml example**: If we create the color in colors.xml

```
0    <color name="colorPrimary"> #FF3F51B5 </color>
```

Then, in themes.xml, we can add or uncomment the line

```
0    <item name="colorPrimary">@color/colorPrimary</item>
```

inside of the action bar tag

```
0    <style name="Base.Theme.Test3"
  ↪    parent="Theme.AppCompat.Light.DarkActionBar">
1        <item name="colorPrimary">@color/colorPrimary</item>
2    </style>
```

Now we have a blue action bar.

- **AndroidManifest.xml**: The AndroidManifest.xml file is located in the manifests directory.

  It specifies the resources that the app uses, such as activities, the file system, the Internet, and hardware resources.

  Before a user downloads an app on Google Play, the user is notified about these details

- **Changing the launcher icon**: When your app is installed on a device, its icon and name appear with all other installed apps in the launcher

  Users can use the icon to launch our app on their device.

  We should always supply a launcher icon for our app

A launcher icon for a mobile device should be 48 x 48 dp.

Because various devices can have different screen densities, we can supply several launcher icons, one for each density

If we provide these different versions of our icon, we need to follow the 2/3/4/6/8 scaling ratios between the various densities from medium (2) to xxx-high (8)

If we supply only one icon - as we will do here - Android Studio will use that icon and expand its density as necessary.

If we plan to publish our app, we should provide a 512 x 512 launcher icon for display in Google's app store.

Right mouse click on mipmap in the project structure along the right side and choose New and then Image Asset.

To set the launch icon for the app to hi.png, we assign the String @mipmap/hi to the android:icon attribute and the String @mipmap/hi_round to the android:roundIcon attribute of the application element in the AndroidManifest.xml file.

The @mipmap/hi expression defines the resource in the mipmap directory (of the res directory) whose name is hi (note that we do not include the extension) and the same for hi_round

```
o   android:icon="@mipmap/hi"
↪   android:roundIcon="@mipmap/hi_round"
```

- **Orientation**: Sometimes, we want the app to run in only one orientation

  In other words, we do not want the app to rotate when the user rotates his or her device.

  Inside the activity element (in the manifest), we can add the android:screenOrientation attribute and specify either portrait or landscape as its value

  For example, if we want our app to run in vertical orientation only, we add

```
o   android:screenOrientation="portrait"
```

  Note that we can control the behavior of the app on a per activity basis

- **Gradle build system**: Android Application Package, or APK, is the file format for distributing applications that run on the Android operating system.

  The file extension is .apk

To create an apk file, the project is compiled and its various parts are packaged into the apk file.

The apk file can be found in the project directory:

$$projectName/app/build/outputs/apk$$

These .apk files are built using the gradle build system, which is integrated in the Android Studio environment.

When we start creating an app, the gradle build scripts are automatically created

They can be modified to build apps that require custom building

- **Debugging**: Just like in a regular Java program, we can send output to the console in addition to displaying data on the screen.

For this, we can use one of the static methods of the Log class.

The Log class is part of the android.util package

Selected methods of the Log class include d, e, i, v, w.

They all have the same parameter list and return type; for example, the API of d is

```
o   public static void d(String tag, String message)
```

- **Log.d(String tag, String msg):** Debug: Used for debugging messages. These are usually filtered out in release builds.
- **Log.e(String tag, String msg):** Error: Used to report error conditions. This is the highest-severity logging method.
- **Log.i(String tag, String msg):** Info: Used for general information messages that highlight the progress of the application.
- **Log.v(String tag, String msg):** Verbose: Used for the most detailed log messages, often too much for normal use, but helpful during deep debugging.
- **Log.w(String tag, String msg):** Warning: Used to report potential issues or unexpected states that aren't necessarily errors.

tag identifies the source of the message and can be associated with a "filter" (described in a few slides).

message is the String to be output.

To run the app in debug mode, we click on the debug icon on the toolbar.

The app runs and stops at the first breakpoint.

The Debug tab will open in the panel at the bottom of the screen.

Here we will see some debugging information and tools.

Under Frames, we can see where in the code we are currently executing.

To resume the app, we click on the green Resume icon at the top left of the panel.

As we resume, stop at breakpoints, and resume the app a few times, the values of the various variables in our app are displayed under Variables.

Under Variables, we can check the values of the various variables

If the app is running on a device, we can still log output statements in Logcat

This is much faster than starting the emulator.

- **Logcat**: Output from logging statements show up in the Logcat

To open the Logcat, click on the Logcat tab at the bottom of the IDE.

We can filter the output listed in the Logcat by clicking in the dropdown in the upper right of the Logcat

Suppose we create a new filter named *f_mainactivity*, with a tag MainActivity.

Now that a filter has been created along with its tag, we can output messages to Logcat.

Add the following line to the onCreate method of MainActivity.java but after the call to super.onCreate

```
0   Log.w("MainActivity", "Inside onCreate!");
```

The output we will see in Logcat is: Inside onCreate!

Or another way to do it

```
0   ...
1   public static String MA = "MainActivity";
2   ...
3   setContentView(R.layout.activity_main);
4   Log.w(MA, "View resource: " +
5   R.layout.activity_main);
6   ...
```

17

1.2 Reference

1.2.1 Includes

- androidx.appcompat.app.AppCompatActivity;
- android.os.Bundle;
- android.util.Log;
- android.view.View;
- android.view.Gravity;
- android.graphics.Color;
- android.widget.EditText;
- android.widget.TextView;
- java.text.NumberFormat;
- android.text.TextWatcher:
- android.text.Editable
- java.lang.CharSequence
- android.graphics.Point;
- android.widget.Button;
- android.widget.GridLayout;
- androidx.constraintlayout.widget.ConstraintLayout;
- androidx.constraintlayout.widget.ConstraintSet;
- androidx.constraintlayout.widget.Guideline
- import androidx.constraintlayout.widget.Barrier;
- android.view.ViewGroup;
- android.content.Context;
- android.content.DialogInterface;
- androidx.appcompat.app.AlertDialog;
- android.graphics.Typeface;
- android.view.Display;
- android.widget.RelativeLayout

1.2.2 Important information

- **AVD (Android virtual device)**: It's basically an emulated device configuration that runs inside the Android Emulator. An AVD defines things like:
    - **Device model**: (screen size, resolution, density, RAM, etc.)
    - **System image**: (Android version, API level, Google Play support, etc.)
    - **Hardware features**: (camera, GPS, sensors, etc.)

  So when you launch an emulator in Android Studio, you're really starting up an AVD that behaves like a specific Android phone or tablet.

- **Screen Density**: screen density means how many pixels are packed into a physical area of the screen, usually measured as dots per inch (dpi).

  A screen with high density has many pixels in a small space, making things look sharper but also smaller if not scaled.

  A screen with low density has fewer pixels in the same area, so things look larger but less sharp.

  Android groups devices into "density buckets" so developers don't have to manually calculate for every possible screen:

    - **ldpi (low)**: ∼120 dpi
    - **mdpi (medium)**: ∼160 dpi (baseline)
    - **hdpi (high)**: ∼240 dpi
    - **xhdpi (extra-high)**: ∼320 dpi
    - **xxhdpi (extra extra high)**: ∼480 dpi
    - **xxxhdpi (extra extra extra high)**: ∼640 dpi

  Android uses this formula under the hood:

$$px = \ dp \times \frac{dpi}{160}$$

  Where

    - dp = your value in density-independent pixels (e.g., 20)
    - dpi = the device's actual screen density in dots-per-inch
    - 160 = the baseline density (mdpi)

  Even though the number of pixels changes, the physical size (in inches or mm) stays about the same. That's because on a denser screen, pixels are smaller, so Android uses more of them to maintain the same real-world size.

  So your 20dp button will look like the same size button whether it's on a cheap low-res phone or a modern super high-res one.

- **Layout Params**: Generally, a layout class like ConstraintLayout or RelativeLayout contains a static inner class that contains XML attributes that allow us to arrange the components within the layout. This static inner class is often named **LayoutParams**

Every child view inside a ViewGroup needs layout parameters (LayoutParams).

These tell the parent how big the child should be and how it should be positioned.

Different ViewGroups define their own rules:

- **LinearLayout.LayoutParams**: weight, gravity
- **FrameLayout.LayoutParams**: gravity
- **ConstraintLayout.LayoutParams**: constraints like topToBottom, leftToLeft, etc.

- **Positioning components inside ConstraintLayout**: We will use the attributes of ConstraintLayout.LayoutParams to position the XML elements on the screen

- **Ids**: The android:id attribute allows us to give an id to an XML element

The syntax for assigning an id to an XML element is

```
o   android:id = "@+id/idValue"
```

- **Default colors (defined in @android:color)**:
  - **@android:color/black**: #FF000000
  - **@android:color/darker_gray**: #FF444444
  - **@android:color/dim_gray**: #FF696969
  - **@android:color/gray**: #FF888888
  - **@android:color/light_gray**: #FFCCCCCC
  - **@android:color/white**: #FFFFFFFF
  - **@android:color/red**: #FFFF0000
  - **@android:color/green**: #FF00FF00
  - **@android:color/blue**: #FF0000FF
  - **@android:color/yellow**: #FFFFFF00
  - **@android:color/cyan**: #FF00FFFF
  - **@android:color/magenta**: #FFFF00FF
  - **@android:color/transparent**: #00000000 (fully transparent)

- **ConstraintLayout Enable Autoconnection to Parent**: When you enable it, every new view you drag into the ConstraintLayout will automatically get constraints to its closest edges of the parent (top, bottom, start, end).

- **Styles vs Themes**: A style relates to a UI component or a View. A theme relates to an activity; it can even relate to the whole app.

We can also "theme" an app with a style by editing the AndroidManifest.xml file, changing the android:theme attribute of its application element using this syntax:

```
o   android:theme="@style/nameOfStyle"
```

- **findViewById**: findViewById is a fundamental method in Android development used to obtain a reference to a View object defined in your XML layout files. This method allows you to interact with UI elements programmatically, such as setting text, handling clicks, or changing visibility.

  For example,

```
o   EditText billEditText =
↪   (EditText)findViewById(R.id.amount_bill);
```

  findViewById returns the View that is part of the layout xml file that was inflated in method onCreate() of the Activity class.

  If we expect to retrieve a TextView and we want to assign the View retrieved to a TextView, we need to cast the View returned by method findViewById() to a TextView:

- **View Event handling with XML (only onClick)**: To set up a click event on a View, we use the android:onClick attribute of the View and assign to it a method using the form:

```
o   android:onClick="methodName"
```

  The event will be handled inside that method, which must have the following API

```
o   public void methodName(View v)
```

  v is the View where the event happened.

  When the user clicks on the button, we execute inside calculate, and its View parameter is the Button; if we have the following statement inside calculate

```
o   Log.w("MainActivity", "v = " + v);
```

  Inside LogCat, we have something like:

```
o   v=android.widget.Button@425a2e60
```

  The value above identifies the Button

- **IME**: It's the software keyboard or other input system (like voice typing, handwriting, predictive text) that allows users to enter text into your app.

Every EditText communicates with the IME through the Input Method Framework (IMF) in Android.

When you tap into an EditText, the IME pops up (usually the on-screen keyboard).

- **Get size of screen**:

```
0   import android.graphics.Point;
1
2   Point size = new Point();
3   getWindowManager().getDefaultDisplay().getSize(size);
```

- **Create grid layout in java (controller)**:

```
0   GridLayout gridLayout = new GridLayout(this);
1   gridLayout.setRowCount(int m);
2   gridLayout.setColumnCount(int n);
```

- **Adding views to GridLayout**:

```
0   gridLayout.addView(view, w, h);
```

- **setContentView**: the method setContentView(...) is used inside an Activity to specify which layout file (XML) should be used as the UI for that screen.

  Calling `setContentView(R.layout.some_layout)` tells the activity:

  "Use the layout resource some_layout.xml from the *res/layout* folder as the UI for this screen."

  If you're using programmatic UI (creating Views in Java instead of XML), you can pass a View object directly,

- **ViewGroup**: In Android, everything you see on screen is either a View or a ViewGroup.

  A View is a basic UI element — e.g. Button, TextView, EditText, ImageView.

  A ViewGroup is a container that holds other Views (and even other ViewGroups).

  So, a ViewGroup is essentially a layout manager. It defines how child views are positioned, sized, and arranged inside it.

  - **LinearLayout**: places children in a row or column.
  - **RelativeLayout (older, mostly replaced by ConstraintLayout)**: places children relative to each other.
  - **ConstraintLayout**: a flexible, modern layout system.
  - **FrameLayout**: stacks children on top of each other.
  - **RecyclerView**: a powerful scrolling container for lists/grids.

  Note: A ViewGroup is a subclass of View.

- **Giving id to view in java**: We can use

```
o  View.generateViewId()
```

as an argument to `view.setId()`. The method `generateViewId()` is a static method in the View class. It creates a unique integer ID at runtime that's guaranteed not to collide with other view IDs.

- **The finish() method**: finish() is a method of the Activity class. It tells Android:

  "I'm done with this Activity, please close it and remove it from the back stack."

So if you call finish() inside an Activity, that activity will end and control will go back to the previous one (or exit the app if it was the only activity).

- **Converting px to dp in java**: In Android, px (pixels) and dp (density-independent pixels) are not the same thing. You almost always want to work in dp because it scales properly across different screen densities.

```
o  float dp = px /
   ↪   context.getResources().getDisplayMetrics().density; // px
   ↪   to dp
1
2  float px = dp *
   ↪   context.getResources().getDisplayMetrics().density; // dp
   ↪   to px
```

when a Java function in Android Studio takes an int representing a dimension (like width, height, margin, radius, stroke width, etc.), it is almost always in raw pixels (px) — not dp.

- **xmlns:android="http://schemas.android.com/apk/res/android"**: One of the most important pieces of XML in Android layouts.

It tells the XML parser: "Whenever you see an attribute that starts with android:, it belongs to the Android system's XML namespace, located at this URI."

You must include this in:

  – Any layout XML (e.g. activity_main.xml)
  – Any drawable XML, menu XML, or style XML that uses android: attributes

Basically, any file where you use attributes like android:layout_width, android:padding, android:text, etc.

you only need to declare that line once, and it always goes on the root element of your XML layout (like <ConstraintLayout>, <RelativeLayout>, <LinearLayout>, etc.).

- **The dp() function (java)**: In Android, UI dimensions shouldn't be defined in plain pixels because screens have different pixel densities (dpi). Instead, we must use dp (density-independent pixels), which scale consistently on all screens.

XML converts dp to px for us, but in java we must do it ourselves. So, we write the following function

```
0   private int dp(int value) {
1       float density =
    ↪   getResources().getDisplayMetrics().density;
2       return (int) (value * density);
3   }
```

We pass in a **dp** values, and it gets converted to px.

- **Animation directory**: In order to create and use animations in our apps, we need to create a **Android Resource Directory** in the **res** directory with resource type **anim**. Then, we can define animations in XML (one animation per XML file).

- **Action bar**

```
0   <resources>
1     <!-- Base application theme. -->
2     <style name="Theme.MusicStore"
    ↪   parent="Theme.AppCompat.Light.DarkActionBar">
3       <!-- Customize your theme here. -->
4       <item name="android:textSize">15sp</item>
5       <item name="colorPrimary">@color/colorPrimary</item>
6     </style>
7
8   </resources>
```

- **No action bar**

```
0   <resources xmlns:tools="http://schemas.android.com/tools">
1       <!-- Base application theme. -->
2       <style name="Base.Theme.MusicStore"
    ↪   parent="Theme.Material3.DayNight.NoActionBar">
3           <!-- Customize your light theme here. -->
4           <!-- <item name="colorPrimary">@color/my_light_prima⌋
    ↪   ry</item> -->
5       </style>
6
7       <style name="Theme.MusicStore"
    ↪   parent="Base.Theme.MusicStore" />
8   </resources>
```

- **Menus**: Menus in Android Studio (and Android apps generally) are UI components that let users interact with common actions — similar to menus in desktop apps, but adapted for mobile.

  A menu is a set of action items your app displays in:

  - The top App Bar (toolbar)
  - A vertical overflow menu (three-dot icon)
  - A contextual action bar
  - Popup menus inside a view

Menus are defined in `res/menu`

- **Builtin icons**: These are the most frequently used system drawables (they all live under android.R.drawable):

  - **@android:drawable/ic_menu_preferences**: Settings or preferences
  - **@android:drawable/ic_menu_edit**: Edit / modify item
  - **@android:drawable/ic_menu_attach**: Attach file
  - **@android:drawable/ic_menu_send**: Send / share
  - **@android:drawable/ic_menu_delete**: Delete item
  - **@android:drawable/ic_menu_add**: Add item
  - **@android:drawable/ic_menu_edit**: Edit item
  - **@android:drawable/ic_input_add**: Add new item
  - **@android:drawable/ic_delete**: Remove / clear
  - **@android:drawable/ic_menu_search**: Search action
  - **@android:drawable/ic_menu_save**: Save action
  - **@android:drawable/ic_menu_agenda**: Calendar / agenda
  - **@android:drawable/ic_menu_compass**: Navigation / location
  - **@android:drawable/ic_menu_share**: Share data
  - **@android:drawable/ic_menu_info_details**: Info dialog
  - **@android:drawable/ic_menu_close_clear_cancel**: Cancel / close
  - **@android:drawable/ic_menu_myplaces**: Home / location
  - **@android:drawable/ic_menu_camera**: Camera
  - **@android:drawable/ic_menu_gallery**: Gallery or media
  - **@android:drawable/ic_menu_call**: Call / contact
  - **@android:drawable/ic_menu_manage**: Manage list
  - **@android:drawable/ic_dialog_alert**: Alert or warning dialog
  - **@android:drawable/ic_dialog_info**: Info dialog icon

1.2.3 Units

- **dp**: stands for density-independent pixels, or "dips" for short.

  The most common unit for layout dimensions (width, height, margins, padding). Scales with screen density so UI looks consistent across devices.

- **sp**: stands for scalable pixels. Maybe we can call them "sips"?

  Primarily for text size. Like dp, but also respects the user's font size settings (accessibility).

- **px (pixels)**: Actual screen pixels. Avoid using directly because it doesn't scale across different densities.

- **pt (points):** 1/72 of an inch. Rarely used, but supported.

- **in (inches)**: Physical size in inches (based on the screen's dpi).

- **mm (millimeter)**: Physical size in millimeters.

1.2.4   Files

- **AndroidManifest.xml**: The app's blueprint. Declares package name, permissions, minimum SDK, app components (activities, services, etc.), and the launcher activity.

- **MainActivity.java**: The main Java class that runs when the app starts. Controls app logic and connects the UI (XML layouts) with backend code.

- **activity_main.xml**: The layout file for MainActivity. Defines the UI elements (buttons, text, etc.) using XML.

- **colors.xml**: Central place for defining reusable color values. Used for themes, buttons, backgrounds, etc

- **strings.xml**: Stores all text strings (app name, labels, messages). Helps with reusability and localization (multi-language support).

- **dimens.xml**: Defines dimensions like margins, padding, font sizes (e.g., 16dp). Ensures consistent spacing and scaling.

- **themes.xml**: Holds styles and themes (colors, fonts, appearances) applied app-wide or to individual components.

1.2.5 XML tags

- **XML Declaration**: Android's resource compiler can usually parse XML without it. But it is strongly recommended to include it for consistency and to avoid encoding issues

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

- **Resources**: The root element of an XML resource file in the res/values/ directory (like strings.xml, colors.xml, styles.xml, etc.).

```
0   <resources>
1       ...
2   </resources>
```

- **String**:

```
0   <string name=""> ... </string>
```

- **Color**:

```
0   <color name=""> hex </color>
```

- **Dimen**:

```
0   <dimen name=""> ... </dimen>
```

- **Manifest**

```
0   <manifest>
1       ...
2   </manifest>
```

- **Style**: used to define styles and themes for your Android app

```
0   <style name="" parent="">
1       <item name=""> ... </item>
2   </style>
```

1.2.6 Components

- **ConstraintLayout**: In Android Studio, ConstraintLayout is a powerful and flexible layout manager used to design UIs. It's often the default choice in modern Android projects.

  It is a **ViewGroup** (container) that positions and sizes its child views based on constraints you define.

    - **Constraints:** Each view needs at least one horizontal and one vertical constraint (e.g., align left to parent, center in parent, align top to another view).
    - **No deep nesting:** Unlike LinearLayout or RelativeLayout, you can achieve complex designs without nesting multiple layouts, which improves performance.
    - **Flexible positioning:** You can center, chain, bias (percent-based positioning), or even set aspect ratios.

```
0   <androidx.constraintlayout.widget.ConstraintLayout
1   xmlns:android="http://schemas.android.com/apk/res/android"
2   xmlns:app="http://schemas.android.com/apk/res-auto"
3   xmlns:tools="http://schemas.android.com/tools"
4   ...
5   tools:context=".MainActivity">
6       ...
7   </androidx.constraintlayout.widget.ConstraintLayout>
```

  Note: xmlns stands for **XML Namespace**.

  `tools:context=".MainActivity"` tells the Layout Editor which Activity will load this layout. It's a design-time hint for Android Studio (not used at runtime).

- **TextView**: is a basic Android UI widget used to display text to the user. It's read-only by default (unlike EditText, which allows input). It can show plain text, styled text, or even links.

```
0   <TextView ...>
1       ...
2   </TextView
```

- **EditText**: Subclass of TextView that allows the user to enter and edit text. It's the standard widget for text input in Android (like forms, search boxes, chat inputs)

```
0   <EditText ...>
1       ...
2   </EditText>
```

1.2.7 Attributes

- **ConstraintLayout**:

  - **android:layout_width**: Defines the width of the view inside its parent.
  - **android:layout_height**: Same as above, but for height.
    * **wrap_content**: Size just big enough for its content.
    * **match_parent**: Fill the entire parent width.
    * **Specific size**: Like 20dp
  - **android:paddingBottom**: space inside the view, between its boundary and its content (like text or an image).
  - **android:paddingLeft**: space inside the view, between its boundary and its content (like text or an image).
  - **android:paddingRight**: space inside the view, between its boundary and its content (like text or an image).
  - **android:paddingTop**: space inside the view, between its boundary and its content (like text or an image).

  The following layout params are placed as attributes in the components nested inside ConstraintLayouts

  - **app:layout_constraintStart_toStartOf="targetId"**: Aligns the start edge (left in LTR layouts, right in RTL) of this view to the start edge of the targetId.
  - **app:layout_constraintTop_toTopOf="targetId"**: Aligns the top edge of this view to the top edge of the targetId.
  - **app:layout_constraintBottom_toBottomOf="targetId"**: Aligns the bottom edge of this view to the bottom edge of the targetId.
  - **app:layout_constraintLeft_toRightOf="targetId"**: Places the left edge of this view aligned to the right edge of the targetId.
  - **app:layout_constraintRight_toRightOf="targetId"**: Aligns the right edge of this view to the right edge of the targetId.
  - **app:layout_constraintLeft_toLeftOf="targetId"**: Aligns the left edge of this view to the left edge of the targetId.
  - **app:layout_constraintHorizontal_bias (no units, value 0-1.0)**:
  - **app:layout_constraintVertical_bias (no units, value 0-1.0)**:

  Note: Instead of *targetId*, we can specify *parent*

  Bias only works if you constrain both sides (e.g. start and end, or top and bottom). If there's only one constraint, the bias has no effect.

- **RelativeLayout**

  - **android:layout_alignParentTop="true"**: Stick to top edge
  - **android:layout_alignParentBottom="true"**: Stick to bottom edge
  - **android:layout_alignParentStart="true"**: Stick to left (or start) edge
  - **android:layout_alignParentEnd="true"**: Stick to right (or end) edge
  - **android:layout_centerInParent="true"**: Center both vertically and horizontally
  - **android:layout_centerHorizontal="true"**: Center horizontally only

- **android:layout_centerVertical="true"**: Center vertically only
- **android:layout_above="@id/viewId"**: Place above another view
- **android:layout_below="@id/viewId"**: Place below another view
- **android:layout_toStartOf="@id/viewId"**: Place to the left of another view
- **android:layout_toEndOf="@id/viewId"**: Place to the right of another view
- **android:layout_alignStart="@id/viewId"**: Align left edges
- **android:layout_alignEnd="@id/viewId"**: Align right edges
- **android:layout_alignTop="@id/viewId"**: Align top edges
- **android:layout_alignBottom="@id/viewId"**: Align bottom edges
- **android:layout_marginStart / android:layout_marginEnd**: Logical left-/right margins (RTL aware)
- **android:layout_marginLeft / android:layout_marginRight**: Physical left-/right margins (legacy)
- **android:layout_marginTop / android:layout_marginBottom**: Vertical margins
- **android:padding*** Padding inside the view (applies to content, not position)
- **android:layout_alignBaseline="@id/viewId"**: Align text baselines of two views
- **android:layout_alignWithParentIfMissing="true"**: If referenced ID is missing, align with parent instead (rarely used)

- **LinearLayout**:

  - **android:baselineAligned**: When set to false, prevents the layout from aligning its children's baselines.
  - **android:baselineAlignedChildIndex**: When a linear layout is part of another layout that is baseline aligned, it can specify which of its children to baseline align to (that is, which child TextView).
  - **android:divider**: Drawable to use as a vertical divider between buttons.
  - **android:gravity**: Specifies how an object should position its content, on both the X and Y axes, within its own bounds.
  - **android:measureWithLargestChild**: When set to true, all children with a weight will be considered having the minimum size of the largest child.
  - **android:orientation**: Should the layout be a column or a row? Use "horizontal" for a row, "vertical" for a column.
  - **android:weightSum**: Defines the maximum weight sum.

- **TableLayout**

  - **android:shrinkColumns**: The zero-based index of the columns to shrink.
  - **android:stretchColumns**: The zero-based index of the columns to stretch.

- **TableRow**: Doesn't have its own XML attributes, inherits from `LinearLayout`, `ViewGroup`, and `View`

- **FrameLayout**

  - **android:foregroundGravity**: Defines the gravity to apply to the foreground drawable.

– **android:measureAllChildren**: Determines whether to measure all children or just those in the VISIBLE or INVISIBLE state when measuring.

- **TextView**:

  – **android:text**: the actual text string (not just visual, it's the content).
  – **android:hint**: placeholder shown when empty.
  – **android:ellipsize**: controls truncation (end, marquee, etc.).
  – **android:scrollHorizontally**: enables horizontal scrolling.
  – **android:marqueeRepeatLimit**: how many times marquee scroll repeats.
  – **android:inputType**: defines the type of expected text (password, email, number, etc.).
  – **android:digits**: restricts input to specific characters.
  – **android:editable**: (deprecated, use EditText).
  – **android:ems**: sets width in units of "M" characters.
  – **android:freezesText**: whether text is preserved on screen rotation.
  – **android:phoneNumber**: (deprecated, use inputType="phone").
  – **android:selectAllOnFocus**: selects all text when focused.
  – **android:linksClickable**: whether links are clickable.
  – **android:autoLink**: auto-detect links (web, email, phone).
  – **android:focusable**: can this view take focus?
  – **android:focusableInTouchMode**: can it take focus during touch mode?
  – **android:longClickable**: whether it supports long-press actions.
  – **android:cursorVisible**: whether the text cursor is shown.
  – **android:inputMethod**: IME options (soft keyboard).
  – **android:imeOptions**: extra options for keyboard (e.g., actionDone).
  – **android:imeActionId**: action ID for IME.
  – **android:imeActionLabel**: label for IME action key.
  – **android:contentDescription**: for accessibility services (screen readers).
  – **android:autoLink**: auto-detect links.
  – **android:linksClickable**: enable link clicks.

- **EditText**:

  – **android:hint**: A gray placeholder text shown when the field is empty.
  – **android:inputType**: Controls what kind of text can be entered and how the keyboard looks:
    * **text**: normal text
    * **textPassword**: hidden input (••••)
    * **number**: numeric keyboard
    * **numberDecimal**: real numbers
    * **phone**: phone keypad
    * **textEmailAddress**: email-optimized keyboard
    * **android:ems**: Sets the default width in terms of characters.
  – **android:maxLines / android:lines**: Control number of visible lines.
  – **android:gravity**: Aligns the text inside the box.

- **android:drawableLeft / drawableRight**: Add icons inside the field.
- **android:textColor**: Sets the color of the text
- **android:textColorHint**: Sets the color of the hint text

- **Button**:
  - **android:clickable**: whether the button responds to clicks.
  - **android:longClickable**: whether the button responds to long presses.
  - **android:focusable**: can the button take focus.
  - **android:focusableInTouchMode**: focusable via touch navigation.
  - **android:soundEffectsEnabled**: enable/disable click sound.
  - **android:hapticFeedbackEnabled**: enable/disable vibration feedback.
  - **android:contentDescription**: spoken description for screen readers.
  - **android:importantForAccessibility**: whether this button should be exposed to accessibility services.
  - **android:labelFor**: associates this button as a label for another view.
  - **android:enabled**: whether the button can be interacted with.
  - **android:nextFocusUp / nextFocusDown / nextFocusLeft / nextFocusRight**: custom focus navigation.
  - **android:checkable (for ToggleButton/MaterialButton)**: whether it can act like a checkbox.
  - **android:checked (for toggleable buttons)**: initial checked state.
  - **android:duplicateParentState**: inherits enabled/pressed/selected state from parent.
  - **android:visibility**: visible, invisible, or gone.
  - **android:keepScreenOn**: keep the screen on while this button is visible.

- **ListView**
  - **android:divider**: Drawable or color to draw between list items.
  - **android:dividerHeight**: Height of the divider.
  - **android:entries**: Reference to an array resource that will populate the ListView.
  - **android:footerDividersEnabled**: When set to false, the ListView will not draw the divider before each footer view.
  - **android:headerDividersEnabled**: When set to false, the ListView will not draw the divider after each header view.

- **ImageView**
  - **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**: 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:maxHeight**: An optional argument to supply a maximum height for this view.

- **android:maxWidth**: An optional argument to supply a maximum width for this view.
  - **android:scaleType**: Controls how the image should be resized or moved to match the size of this ImageView.
  - **android:src**: Sets a drawable as the content of this ImageView.
  - **android:tint**: The tinting color for the image.
  - **android:tintMode**: Blending mode used to apply the image tint.

- **CompoundButton**:
  - **android:button**: Drawable used for the button graphic (for example, checkbox and radio button).
  - **android:buttonTint**: Tint to apply to the button graphic.
  - **android:buttonTintMode**: Blending mode used to apply the button graphic tint.

- **CheckBox**: Only inherited from CompoundButton, TextView, and View

- **RadioGroup**:
  - **android:checkedButton**: The id of the child radio button that should be checked by default within this radio group.

- **RadioButton**: Only inherited from CompoundButton, TextView, and View.

- **AbsSpinner**:
  - **android:entries**: Reference to an array resource that will populate the Spinner.

- **Spinner**
  - **android:dropDownHorizontalOffset**: Amount of pixels by which the drop down should be offset horizontally.
  - **android:dropDownSelector**: List selector to use for spinnerMode="dropdown" display.
  - **android:dropDownVerticalOffset**: Amount of pixels by which the drop down should be offset vertically.
  - **android:dropDownWidth**: Width of the dropdown in spinnerMode="dropdown".
  - **android:gravity**: Gravity setting for positioning the currently selected item.
  - **android:popupBackground**: Background drawable to use for the dropdown in spinnerMode="dropdown".
  - **android:prompt**: The prompt to display when the spinner's dialog is shown.
  - **android:spinnerMode**: Display mode for spinner options.

- **ProgressBar**
  - **android:animationResolution**: Timeout between frames of animation in milliseconds.
  - **android:indeterminate**: Allows to enable the indeterminate mode.
  - **android:indeterminateBehavior**: Defines how the indeterminate mode should behave when the progress reaches max.
  - **android:indeterminateDrawable**: Drawable used for the indeterminate mode.

- **android:indeterminateDuration**: Duration of the indeterminate animation.
- **android:indeterminateOnly**: Restricts to ONLY indeterminate mode (state-keeping progress mode will not work).
- **android:indeterminateTint**: Tint to apply to the indeterminate progress indicator.
- **android:indeterminateTintMode**: Blending mode used to apply the indeterminate progress indicator tint.
- **android:interpolator**: Sets the acceleration curve for the indeterminate animation.
- **android:max**: Defines the maximum value.
- **android:maxHeight**: An optional argument to supply a maximum height for this view.
- **android:maxWidth**: An optional argument to supply a maximum width for this view.
- **android:min**: Defines the minimum value.
- **android:minHeight**:
- **android:minWidth**:
- **android:mirrorForRtl**: Defines if the associated drawables need to be mirrored when in RTL mode.
- **android:progress**: Defines the default progress value, between 0 and max.
- **android:progressBackgroundTint**: Tint to apply to the progress indicator background.
- **android:progressBackgroundTintMode**: Blending mode used to apply the progress indicator background tint.
- **android:progressDrawable**: Drawable used for the progress mode.
- **android:progressTint**: Tint to apply to the progress indicator.
- **android:progressTintMode**: Blending mode used to apply the progress indicator tint.
- **android:secondaryProgress**: Defines the secondary progress value, between 0 and max.
- **android:secondaryProgressTint**: Tint to apply to the secondary progress indicator.
- **android:secondaryProgressTintMode**: Blending mode used to apply the secondary progress indicator tint.

- **AbsSeekBar**

  - **android:thumbTint**: Tint to apply to the thumb drawable.
  - **android:thumbTintMode**: Blending mode used to apply the thumb tint.
  - **android:tickMarkTint**: Tint to apply to the tick mark drawable.
  - **android:tickMarkTintMode**: Blending mode used to apply the tick mark tint.

- **SeekBar**

  - **android:thumb**: Draws the thumb on a seekbar.

- **GradientDrawable**

  - **android:angle**: Angle of the gradient, used only with linear gradient.

35

- **android:bottom**: Amount of bottom padding inside the gradient shape.
- **android:centerColor**: Optional center color.
- **android:centerX**: X-position of the center point of the gradient within the shape as a fraction of the width.
- **android:centerY**: Y-position of the center point of the gradient within the shape as a fraction of the height.
- **android:color**: Solid color for the gradient shape.
- **android:color**: Color of the gradient shape's stroke.
- **android:dashGap**: Gap between dashes in the stroke.
- **android:dashWidth**: Length of a dash in the stroke.
- **android:endColor**: End color of the gradient.
- **android:gradientRadius**: Radius of the gradient, used only with radial gradient.
- **android:height**: Height of the gradient shape.
- **android:innerRadius**: Inner radius of the ring.
- **android:innerRadiusRatio**: Inner radius of the ring expressed as a ratio of the ring's width.
- **android:left**: Amount of left padding inside the gradient shape.
- **android:right**: Amount of right padding inside the gradient shape.
- **android:shape**: Indicates what shape to fill with a gradient.
- **android:startColor**: Start color of the gradient.
- **android:thickness**: Thickness of the ring.
- **android:thicknessRatio**: Thickness of the ring expressed as a ratio of the ring's width.
- **android:top**: Amount of top padding inside the gradient shape.
- **android:type**: Type of gradient.
- **android:useLevel**: Whether the drawable level value (see Drawable.getLevel()) is used to scale the gradient.
- **android:useLevel**: Whether the drawable level value (see Drawable.getLevel()) is used to scale the shape.
- **android:visible**: Indicates whether the drawable should intially be visible.
- **android:width**: Width of the gradient shape.
- **android:width**: Width of the gradient shape's stroke.
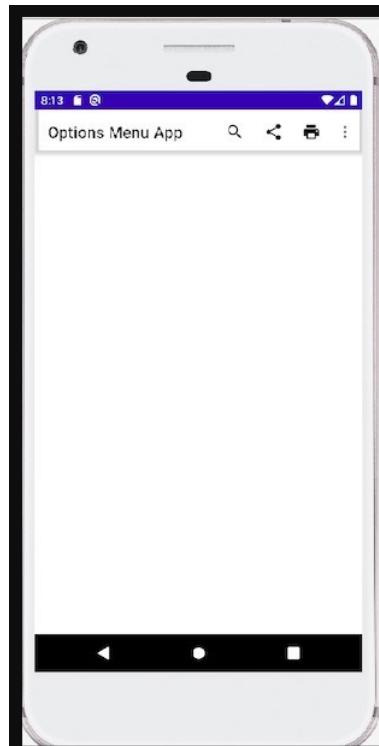
- **Animation**:
  - **android:backdropColor**: Special option for window animations: whether the window's background should be used as a background to the animation.
  - **android:detachWallpaper**: Special option for window animations: if this window is on top of a wallpaper, don't animate the wallpaper with it.
  - **android:duration**: Amount of time (in milliseconds) for the animation to run.
  - **android:fillAfter**: When set to true, the animation transformation is applied after the animation is over.
  - **android:fillBefore**: When set to true or when fillEnabled is not set to true, the animation transformation is applied before the animation has started.

- **android:fillEnabled**: When set to true, the value of fillBefore is taken into account.
- **android:interpolator**: Defines the interpolator used to smooth the animation movement in time.
- **android:repeatCount**: Defines how many times the animation should repeat.
- **android:repeatMode**: Defines the animation behavior when it reaches the end and the repeat count is greater than 0 or infinite.
- **android:showBackdrop**: Special option for window animations: whether to show a background behind the animating windows.
- **android:startOffset**: Delay in milliseconds before the animation runs, once start time is reached.
- **android:zAdjustment**: Allows for an adjustment of the Z ordering of the content being animated for the duration of the animation.

- **set (AnimationSet)**: Only inherited from Animation

- **alpha (AlphaAnimation)**: Only inherited from Animation

- **rotate (RotateAnimation)**: Only inherited from Animation

- **scale (ScaleAnimation)**: Only inherited from Animation

- **translate (TranslateAnimation)**: Only inherited from Animation

- **ScrollView**: Only those inherited from View, ViewGroup, and FrameLayout

- **HorizontalScrollView**: Only those inherited from View, ViewGroup, and Frame-Layout

1.2.8   Menus

- **Menu Resource directory**: Menus are defined in `res/menu`

- **Menu types**

    – **Options menu**: Defined in the action bar / app bar, or in the overflow menu if there's no room.

    – **Context menu**: Appears when the user long-presses an item. Appears next to the pressed item OR as a floating context bar

    – **Popup menu**: A tiny menu that pops up anchored to a button, Triggered by tapping a specific UI element (not long-press)

- **Options menu**

We need to override the `onCreateOptionsMenu` method

```
0   @Override
1   public boolean onCreateOptionsMenu(Menu menu) {
2       getMenuInflater().inflate(R.menu.menu_main, menu);
3       return true;
4   }
```

The menu parameter is an empty Menu object created by Android. It represents the Action Bar options menu that will be shown to the user.

Your job in this method is to add items into this menu, usually done by inflating XML into it

Returning true means yes, we successfully created and want to show this menu.

To handle clicks, we override

```
0   @Override
1   public boolean onOptionsItemSelected(MenuItem item) {
2       if (item.getItemId() == R.id.action_settings) {
3           // action here
4           return true;
5       }
6       return super.onOptionsItemSelected(item);
7   }
```

Calling the super version lets the default Android behavior run when your code doesn't handle the menu click. If it's not your menu item, let the Android system decide what to do.

- **Context Menu**: We first need to register the view

```
0   registerForContextMenu(myView);
```

Then, override

```
0   @Override
1   public void onCreateContextMenu(ContextMenu menu, View v,
2   ContextMenu.ContextMenuInfo menuInfo) {
3       getMenuInflater().inflate(R.menu.context_menu, menu);
4   }
```

To handle clicks, override

```
0   @Override
1   public boolean onContextItemSelected(MenuItem item) {
2       if (item.getItemId() == R.id.action_delete) {
3           // do delete
4           return true;
5       }
6       return super.onContextItemSelected(item);
7   }
```

- **Popup menu**: First, create in code where needed

```
0   PopupMenu popup = new PopupMenu(this, myButton);
1   popup.getMenuInflater().inflate(R.menu.popup_menu,
    ↪  popup.getMenu());
2   popup.show();
```

Handle clicks using a listener

```
0   popup.setOnMenuItemClickListener(item -> {
1       if (item.getItemId() == R.id.action_share) {
2           // do share
3           return true;
4       }
5       return false;
6   });
```

- **Menu attributes**

  - **android:id="Resource ID"**:. A unique resource ID. To create a new resource ID for this item, use the form: "@+id/name". The plus symbol indicates that this is created as a new ID.

  - **android:title="String resource"**:. The menu title as a string resource or raw string.

  - **android:titleCondensed="String resource"**:. A condensed title as a string resource or a raw string. This title is used for situations in which the normal title is too long.

  - **android:icon="Drawable resource"**:. An image to be used as the menu item icon.

  - **android:onClick="Method name"**:. The method to call when this menu item is clicked. The method must be declared in the activity as public. It accepts a MenuItem as its only parameter, which indicates the item clicked. This method takes precedence over the standard callback to onOptionsItemSelected(). See the example at the end of this page.

  - **android:showAsAction="Keyword."**: When and how this item appears as an action item in the app bar. A menu item can appear as an action item only when the activity includes an app bar. Valid values:

    * **ifRoom**: Only place this item in the app bar if there is room for it. If there isn't room for all the items marked "ifRoom", the items with the lowest orderInCategory values are displayed as actions, and the remaining items are displayed in the overflow menu.

    * **withText**: Also include the title text (defined by android:title) with the action item. You can include this value along with one of the others as a flag set by separating them with a pipe |.

    * **never**: Never place this item in the app bar. Instead, list the item in the app bar's overflow menu.

    * **always**: Always place this item in the app bar. Avoid using this unless it's critical that the item always appear in the action bar. Setting multiple items to always appear as action items can result in them overlapping with other UI in the app bar.

* **collapseActionView**: The action view associated with this action item (as declared by android:actionLayout or android:actionViewClass) is collapsible. Introduced in API level 14.
- **android:actionLayout="Layout resource"**:. A layout to use as the action view.
- **android:actionViewClass="Class name"**:. A fully-qualified class name for the View to use as the action view. For example, "android.widget.SearchView" to use SearchView as an action view.
- **android:actionProviderClass="Class name"**:. A fully qualified class name for the ActionProvider to use in place of the action item. For example, "android.widget.ShareActionProvider" to use ShareActionProvider.
- **android:alphabeticShortcut="Char."**: A character for the alphabetic shortcut key.
- **android:numericShortcut="Integer."**: A number for the numeric shortcut key.
- **android:alphabeticModifiers="Keyword."**: A modifier for the menu item's alphabetic shortcut. The default value corresponds to the Control key. Valid values:
  * **META**: Corresponds to the Meta meta key.
  * **CTRL**: Corresponds to the Control meta key.
  * **ALT**: Corresponds to the Alt meta key.
  * **SHIFT**: Corresponds to the Shift meta key.
  * **SYM**: Corresponds to the Sym meta key.
  * **FUNCTION**: Corresponds to the Function meta key.

  You can use the setAlphabeticShortcut() method to set the attribute values programmatically. For more information about the alphabeticModifier attribute

- **android:numericModifiers="Keyword"**:. A modifier for the menu item's numeric shortcut. The default value corresponds to the Control key. Valid values:
  * **META**: Corresponds to the Meta meta key.
  * **CTRL**: Corresponds to the Control meta key.
  * **ALT**: Corresponds to the Alt meta key.
  * **SHIFT**: Corresponds to the Shift meta key.
  * **SYM**: Corresponds to the Sym meta key.
  * **FUNCTION**: Corresponds to the Function meta key.

  You can use the setNumericShortcut() method to set the attribute values programmatically. For more information about the numericModifier attribute

- **android:checkable="Boolean"**:. True if the item is checkable.
- **android:checked="Boolean"**:. True if the item is checked by default.
- **android:visible="Boolean"**:. True if the item is visible by default.
- **android:enabled="Boolean"**:. True if the item is enabled by default.
- **android:menuCategory="Keyword"**:. Value corresponding to the Menu CATEGORY_* constants, which define the item's priority. Valid values:
  * **container**: For items that are part of a container.
  * **system**: For items that are provided by the system.
  * **secondary**: For items that are user-supplied secondary (infrequently used) options.

- ∗ **alternative**: For items that are alternative actions on the data that is currently displayed.
  - **android:orderInCategory="Integer":**. The order of importance of the item within a group.

- **<group>**: A menu group, to create a collection of items that share traits, such as whether they are visible, enabled, or selectable. Contains one or more <item> elements. Must be a child of a <menu> element.

- **<group> attributes**:

  - **android:id="Resource ID"**: A unique resource ID. To create a new resource ID for this item, use the form: "@+id/name". The plus symbol indicates that this is created as a new ID.

  - **android:checkableBehavior="Keyword":**. The type of selectable behavior for the group. Valid values:
    - ∗ **none**: Not selectable.
    - ∗ **all**: All items can be selected (use checkboxes).
    - ∗ **single**: Only one item can be selected (use radio buttons).

  - **android:visible="Boolean":**. True if the group is visible.

  - **android:enabled="Boolean":**. True if the group is enabled.

  - **android:menuCategory="Keyword":**. Value corresponding to the Menu CATEGORY_* constants, which define the group's priority. Valid values:
    - ∗ **container**: For groups that are part of a container.
    - ∗ **system**: For groups that are provided by the system.
    - ∗ **secondary**: For groups that are user-supplied secondary (infrequently used) options.
    - ∗ **alternative**: For groups that are alternative actions on the data that is currently displayed.

  - **android:orderInCategory**: Integer. The default order of the items within the category.

1.2.9 SQLite

- **Extending SQLiteOpenHelper**: We can create a `DatebaseManager` class file, in which we extend `SQLiteOpenHelper`. That's Android's built-in helper for:

    - **Creating tables**: onCreate(SQLiteDatabase db)
    - **Upgrading schema versions**: onUpgrade(SQLiteDatabase db, int oldV, int newV)
    - **Opening / closing the DB safely**: handled by the parent class

    Android automatically calls onCreate() the first time your app accesses the database.

- **Define name and version**: For example,

```
0    private static final String DATABASE_NAME = "recordDB";
1    private static final int DATABASE_VERSION = 1;
```

    Stored in your app's private folder:

```
1    /data/data/<your-package>/databases/recordDB
```

    Version controls schema changes; bumping it triggers onUpgrade().

- **Table creation**: For example,

```
0    public void onCreate(SQLiteDatabase db) {
1        String sqlCreate = "create table record(" +
2            "id integer primary key autoincrement, " +
3            "name text, price real)";
4        db.execSQL(sqlCreate);
5    }
```

    Called automatically the first time the database is created. Builds one table named record with:

    - **id**: primary key (auto-increment)
    - **name**: text column
    - **price**: real (floating-point number)

- **Upgrading the schema**: For example,

```
0    public void onUpgrade(SQLiteDatabase db, int oldVersion, int
  ↪    newVersion) {
1        db.execSQL("drop table if exists record");
2        onCreate(db);
3    }
```

When you increment DATABASE_VERSION, Android calls this. It drops the old table and recreates it. (In production you'd use ALTER TABLE instead of dropping data.)

- **Inserting data**: For example,

```
0  public void insert(Record record) {
1      SQLiteDatabase db = this.getWritableDatabase();
2      String sqlInsert = "insert into record values(null, '" +
3                          record.getName() + "', '" +
       ↪  record.getPrice() + "')";
4      db.execSQL(sqlInsert);
5      db.close();
6  }
```

Opens a writable connection, Executes a raw SQL INSERT statement, Uses null for the auto-incrementing ID, Closes the database

Note: In real apps, you'd use ContentValues to avoid SQL injection.

- **Deleting data**: For example,

```
0  public void deleteById(int id) {
1      SQLiteDatabase db = this.getWritableDatabase();
2      db.execSQL("delete from record where id = " + id);
3      db.close();
4  }
```

Removes one row where the ID matches.

- **Updating data**

```
0  public void updateById(int id, String name, double price) {
1      SQLiteDatabase db = this.getWritableDatabase();
2      db.execSQL("update record set name = '" + name + "',
       ↪  price = '" + price + "' where id = " + id);
3      db.close();
4  }
```

Changes both name and price columns for the selected record.

- **Querying one record**

44

```
0   public Record selectById(int id) {
1       String sqlQuery = "select * from record where id = " +
        ↪  id;
2       SQLiteDatabase db = this.getWritableDatabase();
3       Cursor cursor = db.rawQuery(sqlQuery, null);
4
5       Record record = null;
6       if (cursor.moveToFirst())
7           record = new Record(
8               Integer.parseInt(cursor.getString(0)),
9               cursor.getString(1),
10              cursor.getDouble(2));
11      return record;
12  }
```

- Executes a SELECT query
- Returns a Cursor (a pointer over result rows)
- Reads columns using indices (0=id, 1=name, 2=price)
- Builds and returns a Record object

A Cursor in Android is a data access object that points to the result set (rows and columns) returned from a database query. It works like an iterator that lets you move across each row of the query result and read each column's value.

- sqlQuery = a plain SQL command to get 1 row (SELECT * FROM record WHERE id = ?).
- db.rawQuery(sqlQuery, null) executes the SQL and returns a Cursor.
- The cursor now contains the query result (in memory, inside SQLite).

Important methods

- **moveToFirst()** Moves the cursor to the first row of the result set. Returns false if there are no rows.
- **moveToNext()** Moves the cursor forward one row. Returns false if there are no more rows.
- **moveToPrevious()** Moves the cursor back one row.
- **moveToLast()** Moves the cursor to the last row.
- **move(int offset)** Moves the cursor relative to its current position.
- **moveToPosition(int position)** Moves to an exact row index (0-based). Returns false if out of range.
- **isFirst() / isLast()** Returns true if the cursor is at the first/last row.
- **isBeforeFirst() / isAfterLast()** Returns true if the cursor hasn't started or has finished iterating.
- **getPosition()** Returns the current row index.
- **getInt(int columnIndex)**: Reads an int value.
- **getLong(int columnIndex)**: Reads a long.
- **getDouble(int columnIndex)**: Reads a double.
- **getFloat(int columnIndex)**: Reads a float.
- **getString(int columnIndex)**: Reads a String.

- **getBlob(int columnIndex)**: Reads a byte[] (for images or binary data).
- **isNull(int columnIndex)**: Returns true if the column's value is NULL.
- **getColumnCount()**: Returns number of columns in the result.
- **getColumnName(int index)**: Returns the name of the column at a given index.
- **getColumnNames()**: Returns a String[] of all column names.
- **getColumnIndex(String columnName)**: Returns the column's index (or -1 if not found).
- **getColumnIndexOrThrow(String columnName)**: Same as above, but throws IllegalArgumentException if not found.
- **getCount()**: Returns how many rows are in the result set.
- **isClosed()**: Returns true if the cursor is already closed.
- **getPosition()**: Returns current row index (0 = first).
- **close()**: Frees database and native resources. Always call when done!

- **Query all records**

```
0   public ArrayList<Record> selectAll() {
1       String sqlQuery = "select * from record";
2       SQLiteDatabase db = this.getWritableDatabase();
3       Cursor cursor = db.rawQuery(sqlQuery, null);
4
5       ArrayList<Record> records = new ArrayList<>();
6       while (cursor.moveToNext()) {
7           Record current = new Record(
8               Integer.parseInt(cursor.getString(0)),
9               cursor.getString(1),
10              cursor.getDouble(2));
11          records.add(current);
12      }
13      db.close();
14      return records;
15  }
```

- **Impotant SQLiteDatabase methods**

  - **getReadableDatabase()**: Opens the database in read-only mode.
  - **getWritableDatabase()**: Opens the database in read/write mode (creates it if needed).
  - **close()**: Closes the database connection.
  - **isOpen()**: Returns true if the DB is currently open.
  - **getPath()**: Returns the absolute file path of the database.
  - **execSQL(String sql)**: Executes a single SQL statement (no results returned). For example: db.execSQL("DELETE FROM record WHERE id=5");
  - **insert(String table, String nullColumnHack, ContentValues values)**: Inserts a new row. Returns the new row ID.
  - **update(String table, ContentValues values, String whereClause, String[] whereArgs)**: Updates existing rows matching a condition. Returns number of rows affected.

46

- **delete(String table, String whereClause, String[] whereArgs)**: Deletes rows matching the condition. Returns number of rows deleted.
- **beginTransaction()**: Starts a new transaction.
- **setTransactionSuccessful()**: Marks the transaction to commit.
- **endTransaction()**: Ends the transaction (commits if marked successful, rolls back otherwise).
- **inTransaction()**: Returns true if currently inside a transaction.
- **execSQL(String sql)**: Run any SQL (CREATE, DROP, etc.).
- **getVersion()**: / setVersion(int version) Read or change DB version number.
- **getMaximumSize()**: / setMaximumSize(long numBytes) Check or set max DB file size.
- **needUpgrade(int newVersion)**: Returns true if the database version is lower than the given version.
- **isDatabaseIntegrityOk()**: Verifies database file integrity.
- **isReadOnly()**: Returns true if database is opened read-only.
- **isWriteAheadLoggingEnabled()**: Returns true if WAL mode is active.

1.2.10 Styles

- **How to style**: We can use many attributes to specify how a UI component will look. For example, we can specify the background color of a component.

  We can specify its text size and color and whether the text is centered or not. We can also specify the component's size and the padding inside it

  We use styles and themes to organize our project better

  Themes and styles enable us to separate the look and feel of a View from its content. This is similar to the concept of CSS (Cascading Style Sheets) in web design.

  We can define more styles in the file named themes.xml in the res > values directory.

- **Define a style** The syntax for defining a style is

```
0   <style name="nameOfStyle"
1       [parent="styleThisStyleInheritsFrom"]>
2       <item name="attributeName" parent="styleThisStyleInherit⌋
    ↪    sFrom">attributeValue</item>
3       ...
4   </style>
```

  The (optional) parent attribute allows us to create a hierarchy of styles, i.e., styles can inherit from other styles.

- **Example style**: For example, this style specifies width, height, text size, and padding

```
0   <style name="TextStyle"
1       parent="@android:style/TextAppearance">
2       <item name="android:layout_width">wrap_content</item>
3       <item name="android:layout_height">wrap_content</item>
4       <item name="android:textSize">32sp</item>
5       <item name="android:padding">10dp</item>
6   </style>
```

- **Apply a style**: We give the style attribute to a component. For example,

```
0   <button style="@style/styleName" ...> ... </button>
```

- **TextView Styles**:
  - **android:layout_width**
  - **android:layout_height**
  - **android:layout_margin**
  - **android:layout_gravity**
  - **android:ellipsize**: how text is cut off (none, start, middle, end, marquee)

- **android:singleLine**: (deprecated, use maxLines="1")
- **android:textSize**: font size (e.g. 16sp)
- **android:textColor**: text color (e.g. @color/black)
- **android:textColorHint**: color of the hint text
- **android:textColorHighlight**: color of text selection highlight
- **android:textColorLink**: color of hyperlinks
- **android:textStyle**: normal, bold, italic
- **android:fontFamily**: font family (e.g. sans-serif, serif, or custom font from res/font/)
- **android:typeface**: older way of setting (normal, sans, serif, monospace)
- **android:lineSpacingExtra**: add extra space between lines
- **android:lineSpacingMultiplier**: scale spacing between lines
- **android:letterSpacing**: adjust space between characters
- **android:gravity**: how text is positioned inside its box (top, bottom, left, right, center)
- **android:textAlignment**: alignment relative to parent (gravity, center, viewStart, etc.)
- **android:includeFontPadding**: extra font padding (default true)
- **android:scrollHorizontally**: allow horizontal scroll if needed
- **android:ems**: width in "M" units
- **android:shadowColor**: color of text shadow
- **android:shadowDx, android:shadowDy**: shadow offset
- **android:shadowRadius**: shadow blur radius

- **EditText**: Since EditText is a subclass of TextView, it inherits all of TextView's styling attributes

  - **android:textCursorDrawable**: lets you set a custom drawable for the blinking cursor.
  - **android:textSelectHandle**: base selection handle drawable.
  - **android:textSelectHandleLeft**: left handle for text selection.
  - **android:textSelectHandleRight**: right handle for text selection.
  - **android:colorControlActivated**: (theme attr, but affects EditText focus underline in Material/AppCompat).
  - **android:colorControlNormal**: normal underline/tint when unfocused.
  - **android:colorControlHighlight**: ripple/highlight color when focused.

- **Button**: Button is another subclass of TextView, so it inherits all of TextView's styling attributes

  - **android:text**: label text.
  - **android:textSize**: text size (14sp, 18sp).
  - **android:textColor**: text color.
  - **android:textStyle**: normal, bold, italic.
  - **android:fontFamily**: custom font (@font/roboto_bold).
  - **android:letterSpacing**: adjust spacing between characters.

- **android:lineSpacingExtra / android:lineSpacingMultiplier**: line spacing.
- **android:textAllCaps**: force all caps (default true on Material buttons).
- **android:ellipsize**: how text is cut off if too long.
- **android:background**: drawable for button background (e.g. custom shape).
- **android:foreground**: optional overlay (e.g. ripple).
- **android:insetLeft, android:insetRight, android:insetTop, android:inset-Bottom**: padding inside button background (mostly for legacy Button).
- **android:padding, android:paddingStart, android:paddingEnd**: space inside button.
- **android:layout_width, android:layout_height**: sizing.
- **android:minWidth, android:minHeight**: minimum size (Material buttons have built-in minimums).
- **android:backgroundTint**: tint for the button background.
- **android:backgroundTintMode**: blend mode for tint.
- **android:drawableTint**: tint icons/drawables inside button.
- **android:drawableTintMode**: blending for icon tint.
- **android:drawableStart / android:drawableLeft**: icon at start.
- **android:drawableEnd / android:drawableRight**: icon at end.
- **android:drawableTop, android:drawableBottom**: icons above/below text.
- **android:drawablePadding**: space between icon and text
- **android:elevation**: z-depth shadow (Material design).
- **android:translationZ**: raised elevation when pressed.
- **android:shadowColor, android:shadowDx, android:shadowDy, android:shadowRadius**: text shadow.
- **app:cornerRadius**: rounded corners.
- **app:strokeColor**: outline color.
- **app:strokeWidth**: outline width.
- **app:icon**: set an icon.
- **app:iconPadding**: space between icon and text.
- **app:iconGravity**: where the icon appears (start, end, textStart, textEnd).
- **app:iconTint**: tint icon color.

1.2.11 Events

- **View (XML)**

  – **android:onClick**: name of a method in your Activity that gets called when the button is clicked

1.2.12 Java event listeners

- **Views:**

  - **void setOnClickListener(OnClickListener l)**: Registers a listener to be invoked when the view is clicked (short tap).

  - **void setOnLongClickListener(OnLongClickListener l)**: Registers a listener to be invoked when the view is long-pressed.

  - **void setOnFocusChangeListener(OnFocusChangeListener l)**: Sets a listener that is called whenever the view gains or loses input focus.

  - **void setOnTouchListener(OnTouchListener l)**: Sets a listener to receive touch events (e.g., finger down, move, lift) before they are processed by `onTouchEvent()`.

  - **void setOnKeyListener(OnKeyListener l)**: Sets a listener to receive key events (e.g., hardware button presses) before they are passed to `onKeyDown()` or `onKeyUp()`.

  - **boolean onKeyDown(int keyCode, KeyEvent event)**: Called when a hardware key is pressed down while the view has focus.

  - **boolean onKeyUp(int keyCode, KeyEvent event)**: Called when a hardware key is released while the view has focus.

  - **boolean onTouchEvent(MotionEvent event)**: Handles touch screen interaction with the view (default implementation supports clicks, scrolls, etc.).

  - **boolean onGenericMotionEvent(MotionEvent event)**: Handles non-touch input events such as mouse, joystick, or stylus actions.

  - **boolean onKeyPreIme(int keyCode, KeyEvent event)**: Called when a key event occurs before it reaches the input method (useful for intercepting Back button presses while an IME is visible).

  - **boolean onTrackballEvent(MotionEvent event)**: Handles trackball events (legacy input for older devices).

- **TextView**

  - **void addTextChangedListener(TextWatcher watcher)**: Registers a TextWatcher to receive callbacks when the text changes (before, during, or after editing).

  - **void removeTextChangedListener(TextWatcher watcher)**: Unregisters a previously added TextWatcher so it no longer receives callbacks.

  - **void setOnEditorActionListener(TextView.OnEditorActionListener l)**: Sets a listener to handle editor actions from the soft keyboard (e.g., pressing "Done", "Next", or "Search").

  - **void setOnClickListener(View.OnClickListener l)**: Assigns a listener to handle click events when the view is tapped.

  - **void setOnLongClickListener(View.OnLongClickListener l)**: Assigns a listener to handle long-click (press-and-hold) events on the view.

  - **void setOnFocusChangeListener(View.OnFocusChangeListener l)**: Sets a listener that is triggered when the view gains or loses input focus.

- **EditText**:

  - **void addTextChangedListener(TextWatcher watcher)**: Registers a listener for text changes.

  - **void removeTextChangedListener(TextWatcher watcher)**: Unregisters a text change listener.

– **void setOnEditorActionListener(TextView.OnEditorActionListener l)**:
    Sets a listener for handling IME action events (e.g., pressing Enter, Done, Search).

- **Button:** Subclass of TextView, which is a subclass of View, so all the ones listed above

1.2.13 Java event listeners (2)

- **TextWatcher**: The `TextWatcher` interface, from the android.text package, provides three methods to handle key events.

  You can attach a TextWatcher to any view that implements Editable text content — meaning subclasses of TextView that allow editing.

```java
0   import android.text.TextWatcher;
1   import android.text.Editable;
2   import java.lang.CharSequence;
3
4   .
5   .
6   .
7
8
9   private class TextChangeHandler implements TextWatcher {
10      public void beforeTextChanged(CharSequence s, int start,
    ↪   int count, int after )
11
12      public void onTextChanged(CharSequence s, int start, int
    ↪   before, int after)
13
14      public void afterTextChanged(Editable e)
15  }
```

Where (`beforeTextChanged`)

  - **CharSequence s**: The text before the change.
  - **int start**: The position (index) in the text where the change will begin.
  - **int count**: How many characters are about to be replaced (i.e., how many will be removed).
  - **int after**: How many characters will replace the old ones (i.e., how many will be added).

(onTextChanged)

  - **CharSequence s**: The text after the change (current state).
  - **int start**: The position in the text where the change happened.
  - **int before**: Number of characters that were replaced (removed).
  - **int count**: Number of new characters added.

(afterTextChanged)

  - **Editable:** Represents the text content of the EditText after a change has occurred.

  Instantiate an instance of the class and attach it to a view with `.addTextChangedListener()`

- **onClick listener**: Create a private inner class that implements View.OnClickListener, and overrides the function onClick with signature

```
0   public void onClick(View v)
```

Create an instance of the inner class and use the function `setOnClickListener()` to
add it to a view.

Or, use an anonymous inner class

```
0   myView.setOnClickListener(new View.OnClickListener() {
1       @Override
2       public void onClick(View v) {
3           ...
4       }
5   });
```

1.2.14   Code examples

- **Minimum code for controller**:

```
0   package com.example.test3;
1
2   import androidx.appcompat.app.AppCompatActivity;
3
4   import android.os.Bundle;
5
6   public class MainActivity extends AppCompatActivity {
7
8       @Override
9       protected void onCreate(Bundle savedInstanceState) {
10          super.onCreate(savedInstanceState);
11          setContentView(R.layout.activity_main);
12      }
13  }
```

- **Base theme:**

```
0   <style name="Base.Theme.TipCalculator"
    ↪   parent="Theme.Material3.DayNight.NoActionBar">
1       <!-- Customize your light theme here. -->
2       <!-- <item
    ↪   name="colorPrimary">@color/my_light_primary</item>
    ↪   -->
3   </style>
```

1.3 Java reference

1.3.1 Activity

- **Lifecycle methods**

  - **onCreate(Bundle)**: Initialize activity (UI layout, variables, listeners). Most important one.
  - **onStart()**: Activity is becoming visible.
  - **onResume()**: Activity is now in the foreground and interactive.
  - **onPause()**: Partially visible — pause resources, stop animations. called when Android starts or resumes another
  - **onStop()**: No longer visible — release resources.
  - **onRestart()**: Called before restarting after being stopped.
  - **onDestroy()**: Final cleanup before activity is removed from memory.

- **Transitions**

  - **overridePendingTransition(int enterAnim, int exitAnim)**: Overrides the default transition right after calling startActivity() or finish()
  - **finishAfterTransition()**: Finishes the Activity after shared-element transitions complete
  - **postponeEnterTransition()**: Waits before running transition (useful while loading images)
  - **startPostponedEnterTransition()**: Resumes the delayed enter transition
  - **setEnterSharedElementCallback(SharedElementCallback)**: Customize shared element mapping during enter transition
  - **setExitSharedElementCallback(SharedElementCallback)**: Same as above, but for exit transitions

- **Transition constants**

  - **TRANSITION_NONE**: No animation
  - **TRANSITION_OPEN**: When an Activity is opened
  - **TRANSITION_CLOSE**: When an Activity is closed

- **UI and layout management**

  - **setContentView(int layoutResId)**: Sets the XML layout for the Activity.
  - **findViewById(int id)**: Get references to UI elements.

- **Navigation and intents**

  - **startActivity(Intent)**: Launch another Activity.
  - **startActivityForResult(Intent, int)**: Launch an Activity and get returned data. (Deprecated in favor of ActivityResult API, but still widely used.)
  - **finish()**: Close the current Activity.
  - **getIntent()**: Retrieve the Intent that started the Activity.

- **State Preservation**: When configuration changes happen (like screen rotation):

  - **onSaveInstanceState(Bundle)**: Save UI state before destruction.
  - **onRestoreInstanceState(Bundle)**: Restore saved state after recreation.

- **Menu / UI controls**

  - **onCreateOptionsMenu(Menu)**: Initialize menu options.
  - **onOptionsItemSelected(MenuItem)**: Handle menu selections.

- **Context-provided Methods**

  - **getApplicationContext()**: Access app-level context.
  - **getSystemService(String)**: Access system services (e.g., LOCATION_SERVICE).
  - **getSharedPreferences(name, mode)**: App storage for small data.

- **Dialogs, toasts, and interaction**

  - **runOnUiThread(Runnable)**: Update UI from a background thread.
  - **requestPermissions(String[], int)**: Ask for runtime permissions.
  - **onRequestPermissionsResult(...)**: Handle permission results.

1.3.2   android.content.Context

- **What is it**: Context is an interface to global information about your application environment. It gives you access to:

  - App resources (colors, strings, layouts, drawables, etc.)
  - System services (e.g. LayoutInflater, PowerManager, NotificationManager, etc.)
  - Permissions
  - Starting new Activities, Services, etc.

When you create a View in code:

```
0   Button btn = new Button(this);
1   GridLayout grid = new GridLayout(this);
```

That this is a Context. In an Activity, this works because Activity is a subclass of Context.

The View needs a Context to:

  - Know which theme/style to apply
  - Access resources (e.g., strings, colors, dimensions)
  - Hook into the Android system for rendering

Without a Context, a View has no "connection" to the running Android app environment.

1.3.3 Constraint layout in java (and ConstraintSet)

- **Needed includes**:

```
0   import androidx.constraintlayout.widget.ConstraintLayout;
1   import androidx.constraintlayout.widget.ConstraintSet;
2   import android.view.ViewGroup;
```

- **Create ConstraintLayout**:

```
0   ConstraintLayout layout = new ConstraintLayout(this);
```
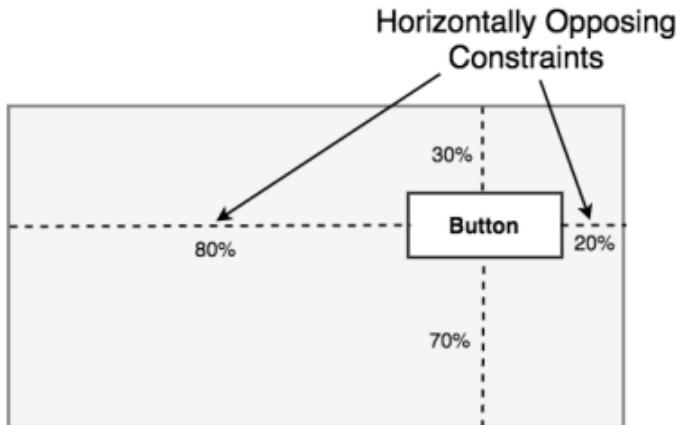
  The argument *this* is the **context**, which in this case is the activity.

- **Constraints**: Constraints are essentially sets of rules that dictate the way in which a widget is aligned and distanced in relation to other widgets. The sides of the containing ConstraintLayout and special elements are called **guidelines**.

  Constraints also dictate how the user interface layout of an activity will respond to changes in device orientation, or when displayed on devices of differing screen sizes. In order to be adequately configured, a widget must have sufficient constraint connections such that it's position can be resolved by the ConstraintLayout layout engine in both the horizontal and vertical planes.

- **Margins**: A margin is a form of constraint that specifies a fixed distance.

- **Opposing Constraints**: Two constraints operating along the same axis on a single widget are referred to as opposing constraints. In other words, a widget with constraints on both its left and right-hand sides is considered to have horizontally opposing constraints.

  The key point to understand here is that once opposing constraints are implemented on a particular axis, the positioning of the widget is now based on percentages of the overall dimensions of the ConstraintLayout rather than coordinate based.
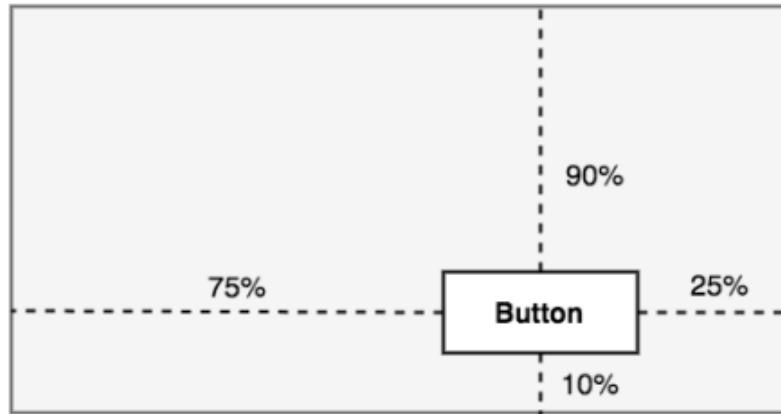
Instead of being fixed at 20dp from the top of the layout, for example, the widget is now positioned at a point 30% from the top of the layout, regardless of the physical dimensions of the container, or parent layout.

In different orientations and when running on larger or smaller screens, the Button will always be in the same location relative to the dimensions of the parent layout.

- **Constraint Bias**: By default, opposing constraints are equal, resulting in the corresponding widget being centered along the axis of opposition.

To allow for the adjustment of widget position in the case of opposing constraints, the ConstraintLayout implements a feature known as constraint bias. Constraint bias allows the positioning of a widget along the axis of opposition to be biased by a specified percentage in favor of one constraint.

Widget Offset using Constraint Bias
Figure 2b-4

Figure 2b-4, for example, shows the previous constraint layout with a 75% horizontal bias and 10% vertical bias:

- **Chains**: ConstraintLayout chains provide a way for the layout behavior of two or more widgets to be defined as a group.

Chains can be declared in either the vertical or horizontal axis and configured to define how the widgets in the chain are spaced and sized. Widgets are chained when connected together by bi-directional constraints.
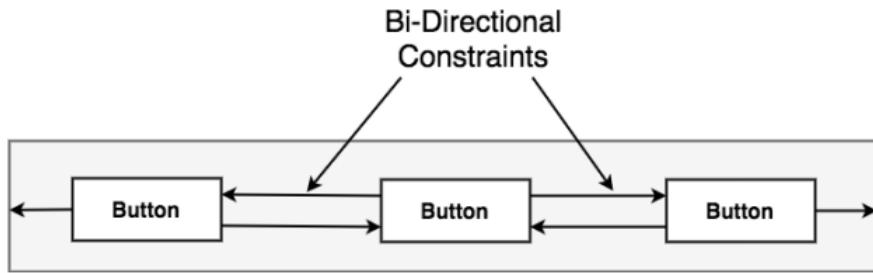
Figure 2b-5

Figure 2b-5, for example, illustrates three widgets chained in this way

The first element in the chain is the chain head which translates to the top widget in a vertical chain or, in the case of a horizontal chain, the left-most widget.

The layout behavior of the entire chain is primarily configured by setting attributes on the chain head widget.

- **Chain Styles**: The layout behavior of a ConstraintLayout chain is dictated by the chain style setting applied to the chain head widget; these are as described next.

  – **Spread Chain**: The widgets contained within the chain are distributed evenly across the available space which is the default behavior for chains.

Figure 2b-6

  – **Spread Inside Chain**: The widgets contained within the chain are spread evenly between the chain head and the last widget in the chain. The head and last widgets are not included in the distribution of spacing

Figure 2b-7

  – **Weighted Chain**: Allows the space taken up by each widget in the chain to be defined via weighting properties.
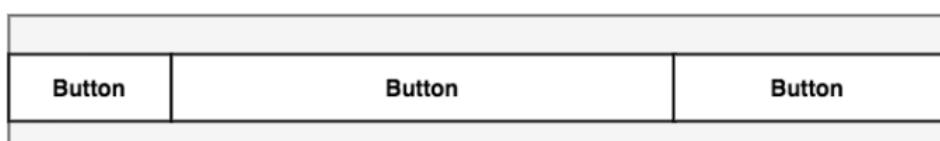
Figure 2b-8

– **Packed Chain**: The widgets that make up the chain are packed together without any spacing. A bias may be applied to control the horizontal or vertical positioning of the chain in relation to the parent container.
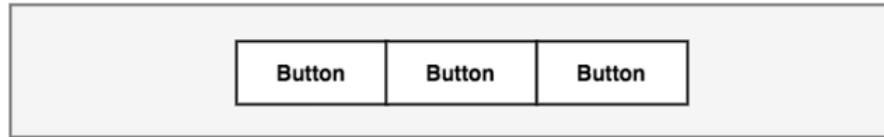
Figure 2b-9

- **Creating chains with java code**: You do it by manually setting constraints directly on each view's ConstraintLayout.LayoutParams, using the leftToRight, rightToLeft, topToBottom, etc. fields.

Suppose we have three buttons in a ConstraintLayout,

```java
0   // --- Button 1 constraints ---
1   ConstraintLayout.LayoutParams lp1 = new
↪       ConstraintLayout.LayoutParams(
2           ViewGroup.LayoutParams.WRAP_CONTENT,
3           ViewGroup.LayoutParams.WRAP_CONTENT
4   );
5   lp1.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
6   lp1.rightToLeft = btn2.getId();     // Chain with Button 2
7   lp1.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
8   lp1.bottomToBottom = ConstraintLayout.LayoutParams.PARENT_ID;
9   lp1.horizontalChainStyle =
↪       ConstraintLayout.LayoutParams.CHAIN_SPREAD;
10  btn1.setLayoutParams(lp1);
11
12  // --- Button 2 constraints ---
13  ConstraintLayout.LayoutParams lp2 = new
↪       ConstraintLayout.LayoutParams(
14          ViewGroup.LayoutParams.WRAP_CONTENT,
15          ViewGroup.LayoutParams.WRAP_CONTENT
16  );
17  lp2.leftToRight = btn1.getId();
18  lp2.rightToLeft = btn3.getId();
19  lp2.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
20  lp2.bottomToBottom = ConstraintLayout.LayoutParams.PARENT_ID;
21  btn2.setLayoutParams(lp2);
22
23  // --- Button 3 constraints ---
24  ConstraintLayout.LayoutParams lp3 = new
↪       ConstraintLayout.LayoutParams(
25          ViewGroup.LayoutParams.WRAP_CONTENT,
26          ViewGroup.LayoutParams.WRAP_CONTENT
27  );
28  lp3.leftToRight = btn2.getId();
29  lp3.rightToRight = ConstraintLayout.LayoutParams.PARENT_ID;
30  lp3.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
31  lp3.bottomToBottom = ConstraintLayout.LayoutParams.PARENT_ID;
32  btn3.setLayoutParams(lp3);
```

- **Chain styles in java**:

```java
0   ConstraintLayout.LayoutParams.CHAIN_SPREAD
1   ConstraintLayout.LayoutParams.CHAIN_SPREAD_INSIDE
2   ConstraintLayout.LayoutParams.CHAIN_PACKED
```

Notice that in the above example, we give `ConstraintLayout.LayoutParams.CHAIN_SPREAD` to the head button in the chain.

- **Weighted chain in java**: To make a weighted chain, you must:

  1. Use `MATCH_CONSTRAINT` (0dp) for the dimension you want to weight (width in a horizontal chain, height in a vertical one).

  2. Assign a weight to each view using layoutParams.horizontalWeight or layoutParams.verticalWeight.

3. Use either `CHAIN_SPREAD` or `CHAIN_SPREAD_INSIDE` style.

For example, consider again those three buttons

```
// ---- Button 1 ----
ConstraintLayout.LayoutParams lp1 = new
↪    ConstraintLayout.LayoutParams(
        0,  // MATCH_CONSTRAINT for weighted width
        ViewGroup.LayoutParams.WRAP_CONTENT
);
lp1.leftToLeft = ConstraintLayout.LayoutParams.PARENT_ID;
lp1.rightToLeft = btn2.getId();
lp1.horizontalWeight = 1f; // weight = 1
lp1.horizontalChainStyle =
↪    ConstraintLayout.LayoutParams.CHAIN_SPREAD;
btn1.setLayoutParams(lp1);

// ---- Button 2 ----
ConstraintLayout.LayoutParams lp2 = new
↪    ConstraintLayout.LayoutParams(
        0,
        ViewGroup.LayoutParams.WRAP_CONTENT
);
lp2.leftToRight = btn1.getId();
lp2.rightToLeft = btn3.getId();
lp2.horizontalWeight = 2f; // weight = 2
btn2.setLayoutParams(lp2);

// ---- Button 3 ----
ConstraintLayout.LayoutParams lp3 = new
↪    ConstraintLayout.LayoutParams(
        0,
        ViewGroup.LayoutParams.WRAP_CONTENT
);
lp3.leftToRight = btn2.getId();
lp3.rightToRight = ConstraintLayout.LayoutParams.PARENT_ID;
lp3.horizontalWeight = 1f; // weight = 1
btn3.setLayoutParams(lp3);
```

- **MATCH_CONSTRAINT**: Let's break down what MATCH_CONSTRAINT means, how it differs from WRAP_CONTENT and MATCH_PARENT, and when you use it.

  MATCH_CONSTRAINT is a special size mode in ConstraintLayout that tells a view:

  "Size yourself dynamically based on your constraints, rather than fixed content or parent size."

  In Java, it's specified by setting the dimension (width or height) to 0 in the layout params:

```
o  ConstraintLayout.LayoutParams params = new
→  ConstraintLayout.LayoutParams(0, WRAP_CONTENT);
```

View size is flexible and determined by the constraints and optionally by weights or ratios

- **Baseline Alignment**: So far, we have only referred to constraints that dictate alignment relative to the sides of a widget (typically referred to as side constraints).

A common requirement, however, is for a widget to be aligned relative to the content that it displays rather than the boundaries of the widget itself. To address this need, ConstraintLayout provides baseline alignment support.
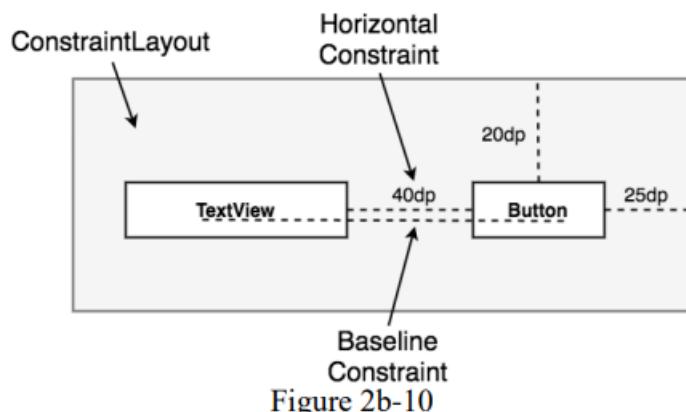
Every view that displays text (e.g. TextView, EditText, Button) has a **text baseline** - the imaginary horizontal line upon which the text "sits."

Baseline alignment means you're aligning two or more views based on that text baseline, instead of their tops or bottoms.

Suppose we require a TextView widget to be positioned 40dp to the left of the Button.

In this case, the TextView needs to be baseline aligned with the Button view.

This means that the text within the Button needs to be vertically aligned with the text within the TextView.

Figure 2b-10

The TextView is now aligned vertically along the baseline of the Button and positioned 40dp horizontally from the Button object's left-hand edge.

In Java, you connect one view's baseline to another view's baseline using:

```
o  params.baselineToBaseline = otherView.getId();
```

66

You can combine this with other constraints like leftToLeft, topToTop, etc.

- **Working with Guidelines**: Guidelines are special elements available within the ConstraintLayout that provide an additional target to which constraints may be connected.

  Multiple guidelines may be added to a ConstraintLayout instance which may, in turn, be configured in horizontal or vertical orientations.

  Once added, constraint connections may be established from widgets in the layout to the guidelines. This is particularly useful when multiple widgets need to be aligned along an axis.
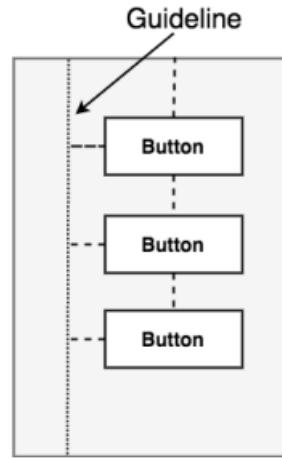
Figure 2b-11

- **Guidelines in java**: First we create a `Guideline` object, then configure `ConstraintLayout.LayoutParams` for it.

```java
0    import androidx.constraintlayout.widget.Guideline
1
2    // 1) Make the guideline
3    Guideline vGuide = new Guideline(this);
4    vGuide.setId(View.generateViewId());
5
6    ConstraintLayout.LayoutParams vgLP = new
     ↪   ConstraintLayout.LayoutParams(
7            ConstraintLayout.LayoutParams.WRAP_CONTENT,
8            ConstraintLayout.LayoutParams.WRAP_CONTENT
9    );
10   vgLP.orientation = ConstraintLayout.LayoutParams.VERTICAL;
11   // Choose ONE of these three ways to position it:
12   vgLP.guidePercent = 0.30f;    // 30% from the left (0..1)
13   // vgLP.guideBegin  = 120;    // 120px from the left
14   // vgLP.guideEnd    = 16;     // 16px from the right
15
16   vGuide.setLayoutParams(vgLP);
17   layout.addView(vGuide);
18
19   // 2) Constrain a view to the guideline
20   TextView tv = new TextView(this);
21   tv.setId(View.generateViewId());
22   tv.setText("Hello");
23   ConstraintLayout.LayoutParams tvLP = new
     ↪   ConstraintLayout.LayoutParams(
24   0,   // MATCH_CONSTRAINT width so it can expand between
     ↪   guideline and parent
25   ConstraintLayout.LayoutParams.WRAP_CONTENT
26   );
27   tvLP.leftToRight = vGuide.getId();                        //
     ↪   to the right of guideline
28   tvLP.rightToRight = ConstraintLayout.LayoutParams.PARENT_ID;
29   tvLP.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
30
31   tv.setLayoutParams(tvLP);
32   layout.addView(tv);
33
34   setContentView(layout);
```

Note: `ConstraintLayout.LayoutParams.orientation` is a property that only applies to Guidelines, not to regular Views.

`ConstraintLayout.LayoutParams.orientation` tells the ConstraintLayout whether a Guideline is:

– **Vertical**: divides the layout left ↔ right
– **Horizontal**: divides the layout top ↔ bottom

• **Working with Barriers**: Rather like guidelines, barriers are virtual views that can be used to constrain views within a layout

As with guidelines, a barrier can be vertical or horizontal and one or more views may be constrained to it (to avoid confusion, these will be referred to as constrained views).

Unlike guidelines where the guideline remains at a fixed position within the layout, however, the position of a barrier is defined by a set of so called reference views.

Barriers were introduced to address an issue that occurs with some frequency involving overlapping views.

Consider the following example
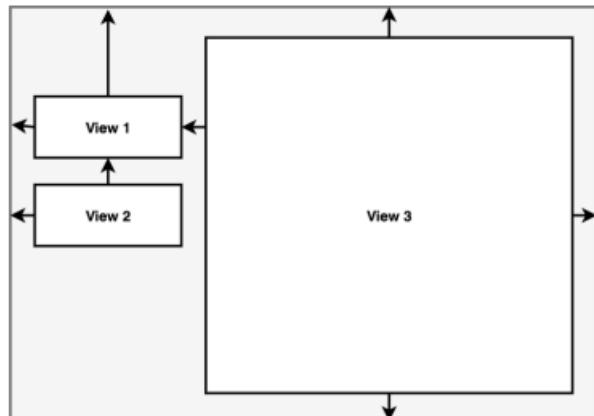
Figure 2b-12

The key points to note about the above layout is that the width of View 3 is set to match constraint mode, and the left-hand edge of the view is connected to the right hand edge of View 1.

As currently implemented, an increase in width of View 1 will have the desired effect of reducing the width of View 3:
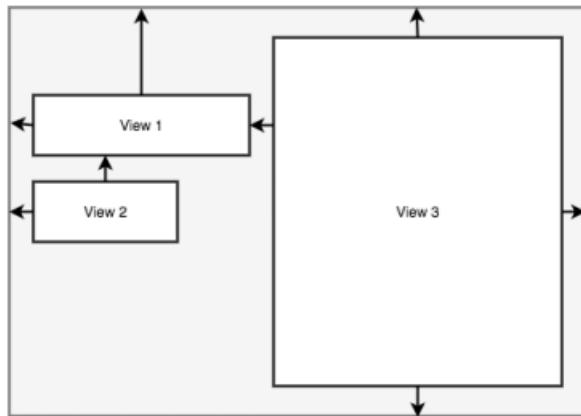
Figure 2b-13

A problem arises, however, if View 2 increases in width instead of View 1:
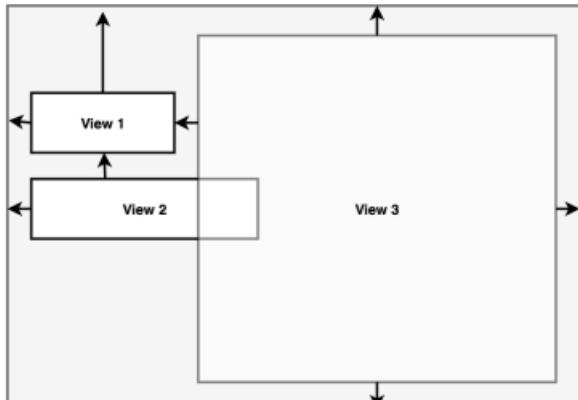
Figure 2b-14

Clearly because View 3 is only constrained by View 1, it does not resize to accommodate the increase in width of View 2 causing the views to overlap.

A solution to this problem is to add a vertical barrier and assign Views 1 and 2 as the barrier's reference views so that they control the barrier position.

The left-hand edge of View 3 will then be constrained in relation to the barrier, making it a constrained view.

Now when either View 1 or View 2 increase in width, the barrier will move to accommodate the widest of the two views, causing the width of View 3 change in relation to the new barrier position:
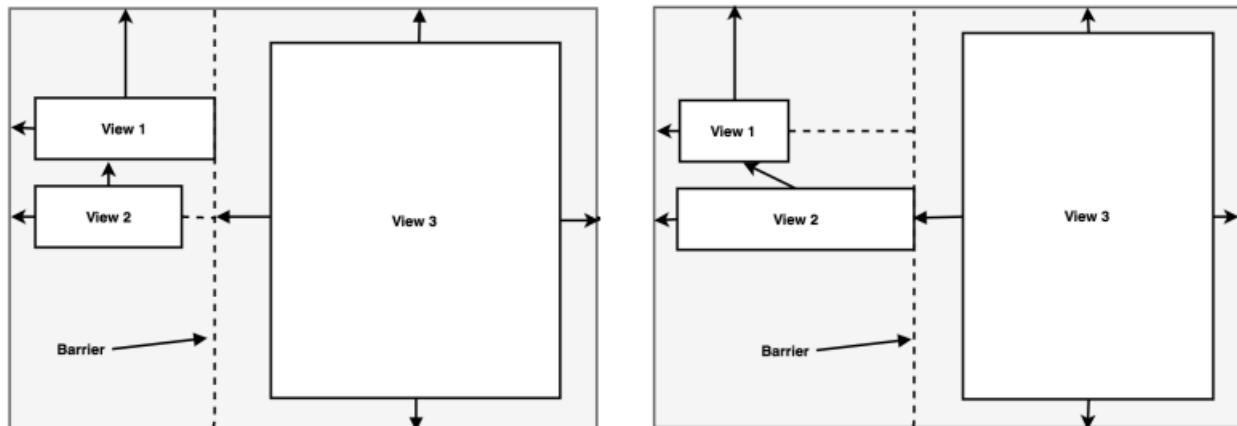
Figure 2b-15

When working with barriers there is no limit to the number of reference views and constrained views that can be associated with a single barrier.

- **Barriers in java**: A Barrier is a virtual helper object in ConstraintLayout that positions itself dynamically based on the position of other views. It's like a movable guideline that automatically adjusts to the furthest edge of a group of views.

Assume we have two TextViews

```
0    import androidx.constraintlayout.widget.Barrier;
1
2    // --- Create a barrier ---
3    Barrier barrier = new Barrier(this);
4    barrier.setId(View.generateViewId());
5    barrier.setType(Barrier.END); // or LEFT, RIGHT, START, TOP,
     ↪   BOTTOM
6    barrier.setReferencedIds(new int[]{label1.getId(),
     ↪   label2.getId()});
7    layout.addView(barrier);
8
9    // Create a button that stays to the right of both text views
10   Button btn = new Button(this);
11   btn.setId(View.generateViewId());
12   btn.setText("Aligned with longest");
13   ConstraintLayout.LayoutParams lp3 = new
     ↪   ConstraintLayout.LayoutParams(
14           ConstraintLayout.LayoutParams.WRAP_CONTENT,
15           ConstraintLayout.LayoutParams.WRAP_CONTENT
16   );
17   lp3.startToEnd = barrier.getId(); // use barrier directly
18   lp3.topToTop = t1.getId();        // align top with first
     ↪   label
19   lp3.leftMargin = 16;
20   btn.setLayoutParams(lp3);
21
22   layout.addView(btn);
23
24   ...
```

- **Barrier types**: Barrier types you can use:
    - Barrier.START
    - Barrier.END
    - Barrier.LEFT
    - Barrier.RIGHT
    - Barrier.TOP
    - Barrier.BOTTOM

Which are explained in the following table

Constant	Tracks the...	Moves toward...	Typical Use
Barrier.START	Start edges (left in LTR, right in RTL)	Start side	Keep something to the left of the leftmost view
Barrier.END End	edges (right in LTR, left in RTL)	End side	Keep something to the right of the rightmost view
Barrier.LEFT	Left edges (absolute)	Left side	Same as START but ignores layout direction
Barrier.RIGHT	Right edges (absolute)	Right side	Same as END but ignores layout direction
Barrier.TOP	Top edges	Upward	Keep something below the topmost/tallest view
Barrier.BOTTOM	Bottom edges	Downward	Keep something below the lowest/bottommost view

- **ConstraintLayout.LayoutParams**: Has constructor

```
0   ConstraintLayout.LayoutParams(int width, int height)
```

Lets you define width and height (MATCH_PARENT, WRAP_CONTENT, or px).

For example,

```
0   ConstraintLayout.LayoutParams lp =
1       new ConstraintLayout.LayoutParams(
2       ViewGroup.LayoutParams.WRAP_CONTENT,
3       ViewGroup.LayoutParams.WRAP_CONTENT
4   );
```

Then, we can add constraints, Each field takes an ID of another view (or PARENT_ID):

- **leftToLeft**: Align the left edge of this view to the left edge of another view (or parent).
- **leftToRight**: Align the left edge of this view to the right edge of another view.
- **rightToLeft**: Align the right edge of this view to the left edge of another view.
- **rightToRight**: Align the right edge of this view to the right edge of another view.
- **topToTop**: Align the top edge of this view to the top edge of another view.

72

- **topToBottom**: Align the top edge of this view to the bottom edge of another view.
- **bottomToTop**: Align the bottom edge of this view to the top edge of another view.
- **bottomToBottom**: Align the bottom edge of this view to the bottom edge of another view.
- **startToStart**: Align the start edge of this view to the start edge of another view.
- **startToEnd**: Align the start edge of this view to the end edge of another view.
- **endToStart**: Align the end edge of this view to the start edge of another view.
- **endToEnd**: Align the end edge of this view to the end edge of another view.
- **horizontalBias(float b)**: $b \times 100$ percent from left
- **verticalBias(float b)**: $b \times 100$ percent from top

Note:

- left/right = physical edges (always left/right of the screen).
- start/end = logical edges (switch meaning in RTL layouts).

For example,

```
0   lp.topToTop = ConstraintLayout.LayoutParams.PARENT_ID;
```

We also have chains, for arranging multiple views in a line with flexible spacing:

- **Retrieve and update LayoutParams**:

```
0   ConstraintLayout.LayoutParams lp =
    ↪   (ConstraintLayout.LayoutParams) view.getLayoutParams();
1   ...
2   view.setLayoutParams(lp);
```

1.3.4 Grid layout in java

GridLayout is a type of ViewGroup that arranges its children in a grid of rows and columns.

Each child view is placed into a "cell" defined by its row and column. You can make a child span multiple rows or columns. It's similar to a table layout but more flexible (alignments, spans, etc.).

- **Needed Includes**:

```
0  import android.content.Context;
1  import android.view.ViewGroup;
2  import android.widget.GridLayout;
```

- **Create GridLayout**: You can make a GridLayout in code just like any other layout:

```
0  GridLayout grid = new GridLayout(this);   // "this" =
   ↪  Context, usually Activity
1  grid.setRowCount(3);                      // number of rows
2  grid.setColumnCount(3);                   // number of
   ↪  columns
```

- **Setting LayoutParams for GridLayout**: We use ViewGroup.LayoutParams to set the layout params for the entire grid container.

```
0  grid.setLayoutParams(
1      new ViewGroup.LayoutParams(
2          ViewGroup.LayoutParams.MATCH_PARENT,
3          ViewGroup.LayoutParams.MATCH_PARENT
4      )
5  );
```

- **GridLayout.LayoutParams**: You need GridLayout.LayoutParams for child views placed inside a GridLayout. GridLayout.LayoutParams is the special layout parameter class that children of a GridLayout must use.

  It tells the GridLayout parent:

  – Which row and column this child belongs to
  – How many cells to span
  – How to align inside those cells
  – How much margin space it should have

  For example,

```
0   Button btn = new Button(this);
1   btn.setText("Hi");
2
3   GridLayout.LayoutParams btnLp = new GridLayout.LayoutParams(
4           GridLayout.spec(0),  // row 0
5           GridLayout.spec(1)   // column 1
6   );
7   btnLp.width = GridLayout.LayoutParams.WRAP_CONTENT;
8   btnLp.height = GridLayout.LayoutParams.WRAP_CONTENT;
9
10  btn.setLayoutParams(btnLp);
11  grid.addView(btn);
```

`GridLayout.spec(int index)` creates a Spec object. A Spec describes a position in either rows or columns. Here, GridLayout.spec(0) means row 0 (the first row), and GridLayout.spec(1) means the first column.

- **GridLayout.spec overloads**:

```
0   GridLayout.spec(int index) // Single index, default span=1,
    ↪  default alignment = UNDEFINED
1   GridLayout.spec(int index, int size) // Index + span
2   GridLayout.spec(int index, Alignment align) // Index +
    ↪  alignment
3   GridLayout.spec(int index, int size, Alignment align) //
    ↪  Index + span + alignment
```

You can pass these alignment constants:

- **GridLayout.START**: align to start (left or top).
- **GridLayout.END**: align to end (right or bottom).
- **GridLayout.CENTER**: center inside the cell.
- **GridLayout.FILL**: expand to fill the cell.

(BASELINE exists for aligning text baselines in rows.)

- **Rule of thumb**: Use ViewGroup.LayoutParams when sizing a container relative to its parent.

  Use the container's own LayoutParams subclass (GridLayout.LayoutParams, LinearLayout.LayoutParams, etc.) when adding children inside that container.

- **Using MarginLayoutParams on GridLayout.setLayoutParams**: This affects the GridLayout as a whole, not its children.

  For example,

```
0   ViewGroup.MarginLayoutParams lp =
1   new ViewGroup.MarginLayoutParams(
2       ViewGroup.LayoutParams.MATCH_PARENT,
3       ViewGroup.LayoutParams.WRAP_CONTENT
4   );
5
6   lp.setMargins(20, 40, 20, 40); // left, top, right, bottom in
    ↪   px
7
8   grid.setLayoutParams(lp);
```

1.3.5 Buttons in java

- **Subclass**: Buttons are a subclass of TextView, which is a subclass of View

- **Creating buttons**: There are basically 3 parts:

  1. Construct the Button (needs a Context)
  2. Customize it (text, size, colors, etc.)
  3. Add it to a parent layout (like GridLayout, LinearLayout, etc.) with proper LayoutParams.

  To create the button,

  ```
  o   Button myButton = new Button(context);
  ```

- **Button methods**:
  - setTextSize(int size)
  - setOnClickListener(listener)
  - setEnabled(boolean status)
  - setText(string text)
  - setLayoutParams(lp)

1.3.6 TextView and EditText

- **Creating TextView**: TextView is a subclass of View

  ```
  o   TextView textView = new TextView(context);
  ```

- **TextView methods**:
  - setWidth(int w)
  - setHeight(int h)
  - setGravity(Gravity g)
  - setBackgroundColor(Color c)
  - setTextSize(int size)
  - setText(string text)
  - setBackgroundColor(Color c)
  - setLayoutParams(lp)
  - setTypeFace(Typeface font, int style)

- **Creating EditText**: EditText is a subclass of TextView, which is a subclass of View

  ```
  o   EditText editText = new EditText(context);
  ```

- **EditText methods**:
  - setWidth(int w)
  - setHeight(int h)
  - setHint(String)
  - setGravity(Gravity g)
  - setBackgroundColor(Color c)
  - setTextSize(int size)
  - setText(string text)
  - setBackgroundColor(Color c)
  - setLayoutParams(lp)

1.3.7 android.graphics.Color

- **Color constants**: This class defines color constants and helper methods for working with colors.

```
0   Color.RED
1   Color.BLUE
2   Color.BLACK
3   Color.WHITE
```

These are just integer values representing ARGB colors.

- **Create colors**:
  - ARGB values (alpha, red, green, blue):

```
0   int myColor = Color.argb(255, 100, 200, 150);
```

Here, 255 = fully opaque.
  - RGB values (no alpha, alpha = 255):

```
0   int myColor = Color.rgb(100, 200, 150);
```

  - Parse from string:

```
0   int myColor = Color.parseColor("#FF5722");  // hex code
```

1.3.8 android.view.Gravity

- **Intro**: This class defines constants used to position or align content inside a View.

  It doesn't move the view itself — it controls how the content inside a view (like text in a TextView) or a child inside a parent layout is aligned.

    – Gravity.LEFT / Gravity.RIGHT
    – Gravity.TOP / Gravity.BOTTOM
    – Gravity.CENTER (both horizontally + vertically)
    – Gravity.CENTER_HORIZONTAL
    – Gravity.CENTER_VERTICAL
    – Gravity.FILL (stretch to fill)

  Use bitwise OR (|) to combine:

```
o   textView.setGravity(Gravity.CENTER | Gravity.BOTTOM);
```

1.3.9 DiaglogInterface and AlertDialog

- **AlertDialog**: A subclass of Dialog. Used to show a modal pop-up window on top of the activity — typically for alerts, confirmations, or choices It can have:

  - A title
  - A message
  - Optional icon
  - Up to 3 buttons (Positive, Negative, Neutral)
  - Custom layouts (if you want more than just text)

- **Create an alert**:

```
0  AlertDialog.Builder alert = new AlertDialog.Builder(this);
1  ...
2  AlertDialog dialog = alert.create();
3  dialog.show();
```

- **AlertDialog.builder methods**

  - alert.setTitle(string title);
  - alert.setMessage(string message);
  - alert.setPositiveButton(string buttonText, DialogInterface.onClickListener);
  - alert.setNegativeButton(string buttonText, DialogInterface.onClickListener);
  - alert.show();

- **DialogInterface**: This is just an interface. Many dialog-related classes (including AlertDialog) implement it.

```
0  public interface DialogInterface {
1      void cancel();
2      void dismiss();
3
4      interface OnCancelListener {
5          void onCancel(DialogInterface dialog);
6      }
7
8      interface OnDismissListener {
9          void onDismiss(DialogInterface dialog);
10     }
11
12     interface OnClickListener {
13         void onClick(DialogInterface dialog, int which);
14     }
15
16     // ... and others like OnKeyListener,
    ↪   OnMultiChoiceClickListener
17 }
```

It gives you common methods to control the dialog:

- **dismiss()**: close the dialog
- **cancel()**: cancel the dialog (triggers onCancel() callback)

It's also used in listeners for button clicks.

- **DialogInterface.OnClickListener**: We can use this interface to show the alert when something is clicked.

```
0   private class MyDialog implements
    ↪   DialogInterface.OnClickListener
1   {
2
3       public void onClick(DialogInterface dialog, int which)
4       {
5
6       }
7   }
```

The which parameter in listeners

- DialogInterface.BUTTON_POSITIVE (-1)
- DialogInterface.BUTTON_NEGATIVE (-2)
- DialogInterface.BUTTON_NEUTRAL (-3)

So you know which button was pressed.

The dialog parameter is a reference to the dialog that triggered the click. Its type is the interface DialogInterface, but in practice it will usually be an instance of a concrete class like AlertDialog. You can use this reference to control the dialog inside the callback:

- **dialog.dismiss()**: close the dialog immediately.
- **dialog.cancel()**: cancel the dialog (triggers OnCancelListener if one is set).

```
0   builder.setPositiveButton("OK", new
    ↪   DialogInterface.OnClickListener() {
1       @Override
2       public void onClick(DialogInterface dialog, int id) {
3           // dialog is the AlertDialog, typed as
            ↪   DialogInterface
4           dialog.dismiss();   // closes it
5       }
6   });
```

1.3.10 GradientDrawable

- **Includes**:

  ```
  o  import android.graphics.drawable.GradientDrawable;
  ```

- **GradientDrawable**: A GradientDrawable is a drawable object (something you can use as a background or graphic) that can display:

  - Solid colors, or Gradients (color transitions),

  and can have:

  - Rounded corners,
  - Borders (strokes),
  - Different shapes (rectangle, oval, line, ring).

  Essentially, it's Android's built-in shape painter

- **Creation**:

  ```
  o  GradientDrawable shape = new GradientDrawable();
  ```

- **Methods**:

Method	Description	Example Usage
`setShape(int shape)`	Sets the type: `RECTANGLE`, `OVAL`, `LINE`, `RING`	`setShape(GradientDrawable.OVAL)`
`setCornerRadius(float radius)`	Rounds the corners (only works for rectangles)	`setCornerRadius(30f)`
`setCornerRadii(float[] radii)`	Gives each corner a different roundness	`setCornerRadii(new float[]20,20,0,0,20,20,0,0)`
`setColor(int color)`	Fills with a solid color	`setColor(Color.BLUE)`
`setStroke(int width, int color)`	Adds a border	`setStroke(3, Color.WHITE)`
`setGradientType(int type)`	Chooses gradient: `LINEAR`, `RADIAL`, `SWEEP`	`setGradientType(GradientDrawable.LINEAR_GRA`
`setColors(int[] colors)`	Defines colors for gradient transitions	`setColors(new int[]Color.RED, Color.YELLOW)`

- **Using GradientDrawable**: We can then call .setBackground() on a view, passing in our GradientDrawable.

- **Giving a button rounded edges**:

  ```
  o  Button b = new Button();
  1
  2  GradientDrawable button_shape = new GradientDrawable();
  3  button_shape.setShape(GradientDrawable.RECTANGLE);
  4  button_shape.setCornerRadius(35f);
  5  button_shape.setColor(PURPLE);
  6
  7  b.setBackground(button_shape);
  ```

1.3.11 android.graphics.Typeface

- **Typeface class**: the Typeface class in Android is the foundation for all text styling related to fonts and weight (bold, italic, etc.). It represents the font face used to render text on screen — in a TextView, Canvas, or anywhere text is drawn.

  A `Typeface` is an object that describes the style and family of a font. It defines how text looks — e.g. whether it's serif or sans-serif, bold or italic, or even a custom font you loaded.

- **Font family constants**:

Constant	Font family	Appearance
Typeface.DEFAULT	Default system font (usually Roboto on newer Androids)	Plain
Typeface.SANS_SERIF	Sans-serif	Clean, modern
Typeface.SERIF	Serif	Classic (like Times New Roman)
Typeface.MONOSPACE	Monospace	Fixed-width (like Courier New)

- **Weight constants**:

Constant	Style
Typeface.NORMAL	0
Typeface.BOLD	1
Typeface.ITALIC	2
Typeface.BOLD_ITALIC	3

- **Using with setTypeFace**

```
o   label.setTypeface(Typeface.SERIF, Typeface.BOLD);
```

- **Null as a parameter**: We can set the font family but not weight

```
o   label.setTypeface(null, Typeface.BOLD);
```

  We can set the family but not weight by using the overload that only accepts a family

```
o   label.setTypeface(Typeface.SERIF);
```

- **Loading custom font from res/font**: If you have `res/font/roboto_bold.ttf`, you can load it like:

```
0   Typeface roboto = ResourcesCompat.getFont(this,
↪    R.font.roboto_bold);
1   textView.setTypeface(roboto);
```

or programmatically from assets:

```
0   Typeface tf = Typeface.createFromAsset(getAssets(),
↪    "fonts/CustomFont.ttf");
1   textView.setTypeface(tf);
```

- **Methods**:

Method	Description
`create(Typeface family, int style)`	Returns a new Typeface based on an existing family and style.
`createFromAsset(AssetManager mgr, String path):`	Loads a Typeface from an asset file (assets/fonts/...).
`createFromFile(File path):`	Loads from a file on disk.
`defaultFromStyle(int style):`	Returns the default Typeface for a style (e.g. Typeface.defaultFromStyle(Typeface.BOLD)).
`equals(Object obj):`	Compares two typefaces.
`hashCode():`	Hash for comparison.

1.3.12 Relative layout

- **Include**

```
0   android.widget.RelativeLayout
```

- **Relative layout**: RelativeLayout is a ViewGroup that lets you position child views relative to each other or to the parent container.

  Each child view can be placed relative to the parent (top, bottom, left, right, center, etc.) or relative to another view (above, below, to the left/right of another widget).

- **Create a RelativeLayout**:

```
0   RelativeLayout layout = new RelativeLayout(this);
1   layout.setLayoutParams(new ViewGroup.LayoutParams(
2       ViewGroup.LayoutParams.MATCH_PARENT,
3       ViewGroup.LayoutParams.MATCH_PARENT
4   ));
```

- **Create RelativeLayout.LayoutParams**

```
0   RelativeLayout.LayoutParams rlp = new
    ↪  RelativeLayout.LayoutParams(new ViewGroup.LayoutParams(
1       RelativeLayout.LayoutParams.WRAP_CONTENT,
2       RelativeLayout.LayoutParams.WRAP_CONTENT
3   ));
```

- **.addRule()**: Adds a layout rule to be interpreted by the RelativeLayout. There are two versions

```
0   void addRule(int verb, int subject)
1   void addRule(int verb)
```

The first version applies a standalone rule - one that does not reference another view.

```
0   params.addRule(RelativeLayout.CENTER_IN_PARENT);
```

The rule means: "center this view both horizontally and vertically inside the parent"

The second version defines a relationship between this view and another view (by ID).

1.3.13 Linear layout

- **Creating linear layout with layout params**:

```
0  LinearLayout root = new LinearLayout(this);
1  root.setOrientation(LinearLayout.VERTICAL);
2  root.setPadding(dp(16), dp(16), dp(16), dp(16));
3  root.setLayoutParams(new LinearLayout.LayoutParams(
4          LinearLayout.LayoutParams.MATCH_PARENT,
5          LinearLayout.LayoutParams.MATCH_PARENT
6  ));
7  root.setGravity(Gravity.START);
```

1.3.14 Table layout and Table row

- **TableLayout**: TableLayout arranges children into rows and columns. But unlike HTML tables, it only positions elements, it does not:

  - Draw borders
  - Automatically size columns evenly
  - Support row/column spanning without special params

  Each row must be a TableRow, and each TableRow contains children like TextView, EditText, Button, etc.

- **Creating TableLayout**

```
0   TableLayout table = new TableLayout(this);
1   table.setLayoutParams(new TableLayout.LayoutParams(
2           TableLayout.LayoutParams.MATCH_PARENT,
3           TableLayout.LayoutParams.MATCH_PARENT
4   ));
5   table.setStretchAllColumns(true); // Makes columns expand
    ↪   evenly
```

- **Creating rows**:

```
0   // Create TableRows and Views (cells)
1   for (int i = 0; i < 3; i++) {
2       TableRow row = new TableRow(this);
3       row.setLayoutParams(new TableLayout.LayoutParams(
4               TableLayout.LayoutParams.MATCH_PARENT,
5               TableLayout.LayoutParams.WRAP_CONTENT
6       ));
7
8       for (int j = 0; j < 3; j++) {
9           TextView cell = new TextView(this);
10          cell.setText("R" + i + " C" + j);
11          cell.setPadding(dp(8), dp(8), dp(8), dp(8));
12          cell.setGravity(Gravity.CENTER);
13
14          row.addView(cell);
15      }
16
17      // Add row to table
18      table.addView(row);
19  }
```

1.3.15   Frame layout

1.3.16   List view

1.3.17 Image view

1.3.18 Compound Button

1.3.19 Check box

1.3.20 RadioGroup and Radio Buttons

1.3.21 Adapter view

1.3.22   Abs spinner

1.3.23 Spinner

1.3.24 Progress bar

1.3.25 Abs seek bar

1.3.26 Seek bar

1.3.27 AttributeSet

1.3.28 Constraint set

1.3.29 defStyleAttr

1.3.30 defStyleRes

1.3.31   android.content.Intent

- **Intent**: An Intent is an Android messaging object used to:
  - Start a new Activity
  - Start a Service
  - Deliver a broadcast to other apps or system components
  - Request an action from another app (e.g., open browser, camera)

  An Intent is how Android apps request actions, move between screens, and share data.

- **Explicit intent**: You specify the exact component (Activity) you want to start.

  ```
  0   Intent intent = new Intent(MainActivity.this,
      ↪   SecondActivity.class);
  1   startActivity(intent);
  ```

  Start SecondActivity from MainActivity:

- **Implicit intent**: You ask the system to find an app that can handle the requested action, used to interact with other apps.

  ```
  0   Uri url = Uri.parse("https://www.google.com");
  1   Intent intent = new Intent(Intent.ACTION_VIEW, url);
  2   startActivity(intent);
  ```

  Here, Android opens whatever browser the user chooses.

- **Passing data with an intent**: To send data to another Activity:

  ```
  0   Intent intent = new Intent(MainActivity.this,
      ↪   SecondActivity.class);
  1   intent.putExtra("username", "Nathan");
  2   intent.putExtra("age", 21);
  3   startActivity(intent);
  ```

  Retrieve it in SecondActivity:

  ```
  0   String name = getIntent().getStringExtra("username");
  1   int age = getIntent().getIntExtra("age", 0);
  ```

- **Updating AndroidManifest**: When we add an activity to an app, we need to add a corresponding activity element to the AndroidManifest.xml file

  ```
  0   <activity
  1       android:name=".classname"
  2       android:screenOrientation="portrait">
  3   </activity>
  ```

1.3.32 android.view.Display

- **Display class**: The android.view.Display class represents a physical screen or display device that your app's UI can be shown on. It provides detailed information about the screen your app is running on — such as its size, refresh rate, orientation, and pixel density.

  It is useful when adapting layouts, scaling graphics, or handling multi-screen setups

- **Getting a display object**: You usually don't create Display yourself. Instead, you retrieve it from a system service like `WindowManager`

```
0   WindowManager wm = (WindowManager)
    ↪   getSystemService(Context.WINDOW_SERVICE);
1   Display display = wm.getDefaultDisplay();
```

  Or, with

```
0   Display display = getWindowManager().getDefaultDisplay();
```

  or, in newer Android versions (API 30+):

```
0   Display display = getDisplay();   // available from any
    ↪   Activity or View
```

  If you're in a non-Activity class (like a helper or controller class), you can get it through a Context:

```
0   WindowManager wm = (WindowManager)
    ↪   context.getSystemService(Context.WINDOW_SERVICE);
1   Display display = wm.getDefaultDisplay();
```

- **Getting display size**: To get the size of the usable screen in pixels:

```
0   Display display = getWindowManager().getDefaultDisplay();
1
2   Point size = new Point();
3   display.getSize(size);
4
5   int width = size.x;
6   int height = size.y;
```

- **Getting Real Screen Size**: To include everything (status bar, navigation bar):

```
0   Point realSize = new Point();
1   display.getRealSize(realSize);
```

- **Getting Refresh Rate**: Returns how many times per second the screen updates:

```
0   float refreshRate = display.getRefreshRate();
```

- **Getting Display Rotation**: Tells you how the screen is currently rotated relative to its "natural" orientation:

```
0   int rotation = display.getRotation();
```

where the possible values are

- **Surface.ROTATION_0**: natural orientation
- **Surface.ROTATION_90**: rotated right
- **Surface.ROTATION_180**: upside down
- **Surface.ROTATION_270**: rotated left

- **Getting Display Metrics**: To obtain screen density and scaling info:

```
0   DisplayMetrics metrics = new DisplayMetrics();
1   display.getMetrics(metrics);
2
3   int densityDpi = metrics.densityDpi;
4   float density = metrics.density;  // Scale factor for dp → px
5   float scaledDensity = metrics.scaledDensity;  // Scale for sp
    ↪  → px
```

- **In Multi-Display or External Display Scenarios**: Starting from Android 4.2+, a device can have multiple displays (like casting to a TV or projector). You can access all of them with:

```
0   DisplayManager dm = (DisplayManager)
    ↪  getSystemService(Context.DISPLAY_SERVICE);
1   Display[] displays = dm.getDisplays();
```

- **Methods**:
  - **getSize(Point)**: Gets the app-usable screen size (in pixels).
  - **getRealSize(Point)**: Gets the full physical display size.
  - **getRotation()**: Returns the screen rotation (0, 90, 180, 270).
  - **getRefreshRate()**: Returns display refresh rate in Hz.
  - **getMetrics(DisplayMetrics)**: Returns logical density and scaling info.
  - **getName()**: Returns display name (useful in multi-display setups).

1.3.33 android.view.KeyEvent

- **KeyEvent**: `android.view.KeyEvent` represents a hardware key press or release event — like when the user presses or releases a key on the device's keyboard, a game controller, or a button such as Volume Up, Back, or Enter.

  KeyEvent objects are delivered to your app whenever a key action happens.

  They describe:

  - Which key was pressed (keyCode)
  - Whether it was a press or release
  - The time, source, modifiers (Shift, Ctrl, etc.)
  - The Unicode character it represents (if any)

- **Lifecycle**

Stage	Event Type	Constant
Key pressed down	ACTION_DOWN	KeyEvent.AC-TION_DOWN
Key released	ACTION_UP	KeyEvent.ACTION_UP
Key held (repeats)	multiple AC-TION_DOWN events	with getRepeatCount() > 0

- **Getting a KeyEvent object**: In Android, you don't manually create KeyEvent objects in most cases. Instead, the Android framework automatically provides them to your app when a user presses or releases a hardware or software key.

  When a key is pressed, the system calls your Activity, View, or Dialog methods and passes a KeyEvent object as a parameter.

  - In an Activity:

```
0   @Override
1   public boolean onKeyDown(int keyCode, KeyEvent event) {
2       Log.d("KeyEvent", "Pressed key: " +
        ↪   event.getKeyCode());
3       return super.onKeyDown(keyCode, event);
4   }
5
6   @Override
7   public boolean onKeyUp(int keyCode, KeyEvent event) {
8       Log.d("KeyEvent", "Released: " +
        ↪   event.getKeyCode());
9       return true;
10  }
```

  Here, the system creates and passes the KeyEvent object automatically.

  - In a View:

108

```
0    @Override
1    public boolean onKeyDown(int keyCode, KeyEvent event) {
2        // Handle key press inside your custom view
3        return true;
4    }
```

Or use a listener:

```
0    view.setOnKeyListener(new View.OnKeyListener() {
1        @Override
2        public boolean onKey(View v, int keyCode, KeyEvent
    ↪ event) {
3            if (event.getAction() == KeyEvent.ACTION_DOWN) {
4                Log.d("KeyEvent", "Key pressed: " +
                ↪ event.getKeyCode());
5                return true;
6            }
7            return false;
8        }
9    });
```

Note: onKeyDown() and onKeyUp() return a boolean because the return value tells the Android framework whether your code has consumed (handled) the key event or no

When you return true, it means "I've handled this key event — don't send it anywhere else."

Returning false means: "I didn't handle this — let the system or another component handle it." Then Android passes the event along:

– From the current View up to its parent

– From the Activity to the Window

– Or eventually to the system (for default behavior)

super.onKeyDown() calls the default handler in the base Activity class, which performs standard Android behaviors (like handling BACK or MENU keys).

- **Getting the action**

```
0    MotionEvent event.getAction()
```

- **Action constants**:

  – **ACTION_DOWN**: Key was pressed down

  – **ACTION_UP**: Key was released

  – **ACTION_MULTIPLE**: Multiple repeated key events (e.g., long press)

For example,

```
0    if (event.getAction() == KeyEvent.ACTION_DOWN) { ... }
```

- **Getting the key code**:

```
0    int event.getKeyCode();
```

- **Key Code Constants**: These tell you which key was pressed.

  There are hundreds of these - a few common groups:

```
0    KEYCODE_A, KEYCODE_B, ..., KEYCODE_Z
1    KEYCODE_0, KEYCODE_1, ..., KEYCODE_9
```

```
0    KEYCODE_ENTER
1    KEYCODE_DEL             // Backspace
2    KEYCODE_TAB
3    KEYCODE_ESCAPE
4    KEYCODE_SPACE
5    KEYCODE_BACK
6    KEYCODE_MENU
7    KEYCODE_HOME
```

```
0    KEYCODE_VOLUME_UP
1    KEYCODE_VOLUME_DOWN
2    KEYCODE_MUTE
3    KEYCODE_MEDIA_PLAY_PAUSE
4    KEYCODE_MEDIA_NEXT
5    KEYCODE_MEDIA_PREVIOUS
```

```
0    KEYCODE_DPAD_UP
1    KEYCODE_DPAD_DOWN
2    KEYCODE_DPAD_LEFT
3    KEYCODE_DPAD_RIGHT
4    KEYCODE_BUTTON_A
5    KEYCODE_BUTTON_B
```

```
0    KEYCODE_POWER
1    KEYCODE_SLEEP
2    KEYCODE_WAKEUP
```

- **Checking if shift was pressed**:

```
o    boolean event.isShiftPressed()
```

- **Getting meta state**

```
o    event.getMetaState()
```

- **Meta / modifier key flags**: Used for Shift, Alt, Ctrl, etc. These can be combined using bitwise OR (|).

  - **META_SHIFT_ON**: Shift key active
  - **META_ALT_ON**: Alt key active
  - **META_CTRL_ON**: Control key active
  - **META_META_ON**: Meta/Command key active
  - **META_SYM_ON**: Symbol modifier active
  - **META_CAPS_LOCK_ON**: Caps lock active
  - **META_NUM_LOCK_ON**: Num lock active
  - **META_SCROLL_LOCK_ON**: Scroll lock active

- **KeyEvent Methods**

  - **getAction()**: int Down, Up, or Multiple
  - **getKeyCode()**: int Which key was pressed
  - **getMetaState()**: int Modifier flags
  - **getRepeatCount()**: int How many times repeated
  - **getEventTime()**: long Time when event occurred
  - **getDownTime()**: long Time when key was first pressed
  - **getDeviceId()**: int ID of the input device (keyboard/gamepad)
  - **getScanCode()**: int Raw hardware scan code
  - **getUnicodeChar()**: int Unicode value (e.g., 'A' $\rightarrow$ 65)
  - **getFlags()**: int Internal system flags
  - **getSource()**: int Input source (keyboard, gamepad, etc.)
  - **isShiftPressed()**: boolean True if Shift active
  - **isCtrlPressed()**: boolean True if Ctrl active
  - **isAltPressed()**: boolean True if Alt active

- **Other useful constants**

  - **Action Constants**: ACTION_DOWN, ACTION_UP
  - **Key Codes**: KEYCODE_A, KEYCODE_ENTER, KEYCODE_BACK
  - **Meta Flags**: META_SHIFT_ON, META_CTRL_ON
  - **Event Data Members**: getAction(), getKeyCode(), getDownTime(), getRepeatCount(), etc.
  - **Flags**: FLAG_LONG_PRESS, FLAG_SOFT_KEYBOARD, etc.

1.3.34 Animations

- **Create AnimationSet**:

```
0   AnimationSet animation = new AnimationSet(boolean
    ↪   shareInterpolator)
```

An Interpolator controls the animation speed pattern over time:

- **true**: (default) All animations inside AnimationSet use the same interpolator (the one set on the AnimationSet itself)
- **false**: Each animation can define its own interpolator

```
0   AnimationSet set = new AnimationSet(true); // true = share
    ↪   interpolator
1
2   // Fade-in animation
3   AlphaAnimation alpha = new AlphaAnimation(0f, 1f);
4   alpha.setDuration(1000);
5
6   // Move-up animation
7   TranslateAnimation move = new TranslateAnimation(
8   0, 0,
9   50f, 0f
10  );
11  move.setDuration(1000);
12
13  // Add animations to set
14  set.addAnimation(alpha);
15  set.addAnimation(move);
16
17  // Start the animation
18  view.startAnimation(set);
```

1.3.35 SharedPreferences, SharedPreferences.Editor, and PreferencesManager

- **Get a SharedPreferences object**

```
0  SharedPreferences prefs = getSharedPreferences("MyPrefs",
   ↪  MODE_PRIVATE);
```

  - **"MyPrefs"**: the filename to store in
  - **MODE_PRIVATE**: → only your app can access it

- **Write / save data**:

```
0  SharedPreferences.Editor editor = prefs.edit();
1  editor.putString("username", "Nate");
2  editor.putBoolean("isDarkMode", true);
3  editor.apply(); // async (recommended)
```

  - **.apply()**: saves in the background
  - **.commit()**: saves immediately but blocks the thread — only use if you must know the result instantly

- **Read data**

```
0  String username = prefs.getString("username", "Guest");
1  boolean darkMode = prefs.getBoolean("isDarkMode", false);
```

  The second argument is the default value if the key does not exist

- **Remove data**

```
0  prefs.edit().remove("username").apply();
```

- **Wipe everything**

```
0  prefs.edit().clear().apply();
```

- **Where is the data stored**: SharedPreferences saves to:

```
1  /data/data/<your package name>/shared_prefs/MyPrefs.xml
```

1.3.36 Menu and MenuItem

- **Which menu type?**: We use the Menu and MenuItem class to create an **Options Menu**.

- **Creating an options menu**: We override `onCreateOptionsMenu()`. Android automatically calls this when the Activity starts. Here, you'll use the Menu object (passed as a parameter) to add items manually.

```
0   @Override
1   public boolean onCreateOptionsMenu(Menu menu) {
2       // groupId, itemId, order, title
3       menu.add(0, 101, 0, "Settings");
4       menu.add(0, 102, 1, "Help");
5       menu.add(0, 103, 2, "Exit");
6
7       // You can also configure each item after adding it
8       MenuItem settingsItem = menu.findItem(101);
9       settingsItem.setIcon(android.R.drawable.ic_menu_preferen↲
    ↪   ces);
10      settingsItem.setShowAsAction(MenuItem.SHOW_AS_ACTION_IF_↲
    ↪   ROOM);
11
12      return true;  // tells Android to display this menu
13  }
```

menu is the Menu interface object, menu.add(...) returns a MenuItem object. You can customize each MenuItem afterward.

- **Handle clicks**: When the user taps a menu item, Android passes a MenuItem object representing what was clicked.

```
0   @Override
1   public boolean onOptionsItemSelected(MenuItem item) {
2       switch (item.getItemId()) {
3           case 101:
4           Toast.makeText(this, "Settings clicked",
    ↪   Toast.LENGTH_SHORT).show();
5           return true;
6           case 102:
7           Toast.makeText(this, "Help clicked",
    ↪   Toast.LENGTH_SHORT).show();
8           return true;
9           case 103:
10          finish(); // close app
11          return true;
12          default:
13          return super.onOptionsItemSelected(item);
14      }
15  }
```

1.3.37 SubMenu

- **What is a SubMenu**: A SubMenu is basically a menu nested inside another menu item. SubMenu is an interface that extends Menu, you don't instantiate it directly — instead, you get it from a Menu object using addSubMenu()

- **Example, creating SubMenu**:

```java
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Create a regular menu item
    menu.add(0, 1, 0, "Settings");

    // Create a submenu under "File"
    SubMenu fileSubMenu = menu.addSubMenu("File");

    // Add items to that submenu
    fileSubMenu.add(0, 2, 0, "New");
    fileSubMenu.add(0, 3, 1, "Open");
    fileSubMenu.add(0, 4, 2, "Save");

    // Another normal menu item
    menu.add(0, 5, 3, "Help");

    return true;
}
```

- **Handle clicks**: You handle clicks the same way as normal menu items:

```
0   @Override
1   public boolean onOptionsItemSelected(MenuItem item) {
2       switch (item.getItemId()) {
3           case 1:
4               Toast.makeText(this, "Settings clicked",
                ↪   Toast.LENGTH_SHORT).show();
5               return true;
6           case 2:
7               Toast.makeText(this, "New clicked",
                ↪   Toast.LENGTH_SHORT).show();
8               return true;
9           case 3:
10              Toast.makeText(this, "Open clicked",
                ↪   Toast.LENGTH_SHORT).show();
11              return true;
12          case 4:
13              Toast.makeText(this, "Save clicked",
                ↪   Toast.LENGTH_SHORT).show();
14              return true;
15          case 5:
16              Toast.makeText(this, "Help clicked",
                ↪   Toast.LENGTH_SHORT).show();
17              return true;
18      }
19      return super.onOptionsItemSelected(item);
20  }
```

- **SubMenus in XML**: Example,

```
0   <menu xmlns:android="http://schemas.android.com/apk/res/andr⌋
    ↪   oid">
1       <item android:title="File">
2           <menu>
3               <item android:id="@+id/action_new"
                ↪   android:title="New" />
4               <item android:id="@+id/action_open"
                ↪   android:title="Open" />
5               <item android:id="@+id/action_save"
                ↪   android:title="Save" />
6           </menu>
7       </item>
8       <item android:id="@+id/action_help" android:title="Help"
        ↪   />
9   </menu>
```

Same structure; you just nest a <menu> inside an <item>.

1.3.38 ContextMenu

- **What is it**: Context menus appear when you long-press a View — like text, a button, or a list item. They're used for actions specific to that item, not the whole Activity.

- **Register a view for a Context Menu**: We use `registerForContextMenu(view v)`

```
0  @Override
1  protected void onCreate(Bundle savedInstanceState) {
2      super.onCreate(savedInstanceState);
3      setContentView(R.layout.activity_main);
4
5      TextView myTextView = findViewById(R.id.myTextView);
6
7      // Tell Android this view should show a context menu on
   ↪   long-press
8      registerForContextMenu(myTextView);
9  }
```

- **Override onCreateContextMenu**: This method is called automatically when the user long-presses the registered view. Here you use the Menu interface to build the menu in code.

```
0  @Override
1  public void onCreateContextMenu(ContextMenu menu, View v,
2                                  ContextMenu.ContextMenuInfo
   ↪   menuInfo) {
3      super.onCreateContextMenu(menu, v, menuInfo);
4
5      // You can set a header or title
6      menu.setHeaderTitle("Choose an action");
7
8      // Add items programmatically
9      menu.add(0, 101, 0, "Edit");
10     menu.add(0, 102, 1, "Share");
11     menu.add(0, 103, 2, "Delete");
12 }
```

  – **ContextMenu Menu**: This is the menu object that you fill with items.
  – **View v**: The view that was long pressed
  – **ContextMenu.ContextMenuInfo menuInfo**: Provides extra context information about the view that was pressed. It's often null unless the view supports structured data — e.g., a ListView or RecyclerView.

     For lists, it contains which row was long-pressed:

You call super to allow the parent class (AppCompatActivity) to perform its own setup logic before or after you modify the menu.

- **Handle clicks**:

```java
@Override
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case 101:
            Toast.makeText(this, "Edit clicked",
              ↪  Toast.LENGTH_SHORT).show();
            return true;
        case 102:
            Toast.makeText(this, "Share clicked",
              ↪  Toast.LENGTH_SHORT).show();
            return true;
        case 103:
            Toast.makeText(this, "Delete clicked",
              ↪  Toast.LENGTH_SHORT).show();
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

1.3.39   PopupMenu

- **What is it**: A PopupMenu is a small floating menu that appears anchored to a specific View — for example, when you tap a button with "⋮" or "More options." It's a temporary dropdown menu used for quick actions — not tied to the Action Bar or long-press events.

- **Example**:

```java
Button button = findViewById(R.id.myButton);

button.setOnClickListener(v -> {
    // Create the popup menu, anchored to the button
    PopupMenu popup = new PopupMenu(MainActivity.this, v);

    // Get the Menu object inside the popup
    Menu menu = popup.getMenu();

    // Add items manually
    menu.add(0, 101, 0, "Edit");
    menu.add(0, 102, 1, "Share");
    menu.add(0, 103, 2, "Delete");

    // Handle clicks on the popup items
    popup.setOnMenuItemClickListener(item -> {
        switch (item.getItemId()) {
            case 101:
                Toast.makeText(MainActivity.this, "Edit
                ↪  clicked", Toast.LENGTH_SHORT).show();
                return true;
            case 102:
                Toast.makeText(MainActivity.this, "Share
                ↪  clicked", Toast.LENGTH_SHORT).show();
                return true;
            case 103:
                Toast.makeText(MainActivity.this, "Delete
                ↪  clicked", Toast.LENGTH_SHORT).show();
                return true;
            default:
                return false;
        }
    });

    // Finally, show the popup
    popup.show();
});
```

1.3.40 Toast

- **What is it**: A Toast is a small popup message that briefly appears at the bottom (or top/center) of the screen. It automatically disappears after a short time.

- **Basic syntax**:

```
0  Toast.makeText(context, "Hello, world!",
↪    Toast.LENGTH_SHORT).show();
```

  - **context**: Usually this, or getApplicationContext() — tells Android which app is showing the Toast
  - **Toast.LENGTH_SHORT**: How long it shows (LENGTH_SHORT $\approx$ 2s, LENGTH_LONG $\approx$3.5s)
  - **.show()**: Displays the Toast

- **Duration options**

  - **Toast.LENGTH_SHORT**: $\sim$2 seconds
  - **Toast.LENGTH_LONG**: $\sim$3.5 seconds

- **Changing position**: You can move them using setGravity():

```
0  Toast toast = Toast.makeText(this, "Top toast!",
↪    Toast.LENGTH_SHORT);
1  toast.setGravity(Gravity.TOP | Gravity.CENTER_HORIZONTAL, 0,
↪    200);
2  toast.show();
```

The last two numbers (xOffset, yOffset) adjust the offset in pixels

- **Custom Layout Toast**: You can even use a custom XML layout instead of plain text:

```
0  LayoutInflater inflater = getLayoutInflater();
1  View layout = inflater.inflate(R.layout.custom_toast,
↪    findViewById(R.id.toastRoot));
2
3  Toast toast = new Toast(getApplicationContext());
4  toast.setDuration(Toast.LENGTH_LONG);
5  toast.setView(layout);
6  toast.show();
```

And res/layout/custom_toast.xml might look like:

```
0   <LinearLayout xmlns:android="http://schemas.android.com/apk/↵
    ↪  res/android"
1       android:id="@+id/toastRoot"
2       android:background="#AA000000"
3       android:padding="10dp"
4       android:orientation="horizontal">
5       <ImageView
        ↪  android:src="@android:drawable/ic_dialog_info" />
6       <TextView
7           android:text="Custom Toast"
8           android:textColor="#fff"
9           android:paddingStart="10dp"/>
10  </LinearLayout>
```

1.3.41 LayoutInflator

- **What is it**: LayoutInflater is a class that converts XML layout files (.xml) into actual View objects in memory — that your app can display or interact with.

Your XML layout is like a blueprint for a house. LayoutInflater is the builder that reads that blueprint and constructs the actual house (View hierarchy) in Java.

Suppose you have a layout file: `res/layout/my_custom_view.xml`

```
0   <LinearLayout xmlns:android="http://schemas.android.com/apk/↵
    ↪   res/android"
1       android:orientation="horizontal"
2       android:padding="10dp">
3       <ImageView android:src="@android:drawable/ic_menu_info_d↵
        ↪   etails"/>
4       <TextView android:text="Hello LayoutInflater!"
        ↪   android:paddingStart="8dp"/>
5   </LinearLayout>
```

You can load it into memory like this:

```
0   LayoutInflater inflater = getLayoutInflater();   // or
    ↪   LayoutInflater.from(context)
1   View view = inflater.inflate(R.layout.my_custom_view, null);
```

Now view is a fully constructed View object tree — a LinearLayout containing an ImageView and TextView.

You can then:

- Add it dynamically to another layout
- Use it in a custom Toast
- Return it from an adapter (for example, in a ListView)

- **inflate()**

```
0   View inflate(int resource, ViewGroup root, boolean
    ↪   attachToRoot)
```

Where

- **resource**: The XML layout resource to inflate (e.g., R.layout.my_view)
- **root**: Optional parent layout to attach to
- **attachToRoot**: Whether to attach the inflated layout to root immediately

1.3.42 ScrollView

- **What is it**: A ScrollView is a layout container that provides vertical scrolling for its single child view.

  If your content doesn't fit on the screen (too tall), wrapping it inside a ScrollView lets the user scroll up and down to see the rest.

- **Key Rules**:

  - **Only one direct child**: A ScrollView can host only one child view. If you need multiple items, put them inside a container like LinearLayout or ConstraintLayout inside the ScrollView.
  - **Vertical only**: ScrollView scrolls vertically. For horizontal scrolling, use HorizontalScrollView.
  - **The child must be taller than the screen**: Otherwise, it won't scroll because everything fits on screen already.

- **Example**:

```
0   ScrollView scrollView = new ScrollView(this);
1   LinearLayout layout = new LinearLayout(this);
2   layout.setOrientation(LinearLayout.VERTICAL);
3
4   // Add some child views
5   for (int i = 1; i <= 20; i++) {
6       TextView tv = new TextView(this);
7       tv.setText("Item " + i);
8       layout.addView(tv);
9   }
10
11  // Add layout inside scrollview
12  scrollView.addView(layout);
13
14  // Set as activity content
15  setContentView(scrollView);
```

- **Scrolling programmatically**

```
0   scrollView.fullScroll(View.FOCUS_DOWN); // scroll to bottom
1   scrollView.fullScroll(View.FOCUS_UP);   // scroll to top
2   scrollView.scrollTo(0, 500);            // scroll to specific
    ↪  Y position
3   scrollView.smoothScrollBy(0, 100);      // scroll smoothly
```

- **Common XML attributes**

  - **android:fillViewport="true"**: Forces the child view to expand to fill the screen height (even if content is short).
  - **android:scrollbars="none"**: Hide scrollbars.
  - **android:fadeScrollbars="false"**: Keep scrollbars always visible.
  - **android:overScrollMode="never"**: Disable the "stretch" overscroll glow effect.

1.4 Java Documentation

1.4.1 View

- **Hierarchy**

$$\text{java.lang.Object} \rightarrow \text{android.view.View}$$

- **Include**

```
0   android.view.View
```

- **Constructors**

```
0   View(Context context)
1   View(Context context, AttributeSet attrs)
2   View(Context context, AttributeSet attrs, int defStyleAttr)
3   View(Context context, AttributeSet attrs, int defStyleAttr,
    ↪   int defStyleRes)
```

- **Public methods (Only most important)**

  - **void setId(int id)**: Sets the unique identifier for the view.
  - **void setVisibility(int visibility)**: Sets whether the view is visible, invisible, or gone.
  - **int getVisibility()**: Returns the current visibility state.
  - **void setEnabled(boolean enabled)**: Enables or disables user interaction.
  - **boolean isEnabled()**: Returns whether the view is currently enabled.
  - **void setFocusable(boolean focusable)**: Controls whether the view can gain focus.
  - **void requestFocus()**: Requests focus for this view.
  - **boolean hasFocus()**: Returns true if this view currently has focus.
  - **void invalidate()**: Redraws the view on screen.
  - **void requestLayout()**: Requests a new layout pass for this view.
  - **void layout(int l, int t, int r, int b)**: Assigns size and position to the view.
  - **void setOnClickListener(View.OnClickListener l)**: Sets a callback to handle click events.
  - **void setOnLongClickListener(View.OnLongClickListener l)**: Sets a listener for long press events.
  - **boolean performClick()**: Programmatically triggers the click listener.
  - **boolean onTouchEvent(MotionEvent event)**: Handles touch interactions.
  - **boolean onKeyDown(int keyCode, KeyEvent event)**: Called when a hardware key is pressed.
  - **void onDraw(Canvas canvas)**: Called to render the view's visual content.
  - **void measure(int widthMeasureSpec, int heightMeasureSpec)**: Determines the measured size of the view.
  - **int getWidth() / int getHeight()**: Return the current dimensions of the view.

124

- **int getLeft() / getTop() / getRight() / getBottom()**: Return the view's position relative to its parent.
- **ViewGroup.LayoutParams getLayoutParams()**: Returns layout parameters assigned to this view.
- **void setLayoutParams(ViewGroup.LayoutParams params)**: Updates the layout parameters.
- **ViewPropertyAnimator animate()**: Starts an animation for view properties (translation, alpha, rotation, etc.).
- **void setAlpha(float alpha)**: Sets the transparency (0.0 = fully transparent, 1.0 = opaque).
- **void setTranslationX(float translationX)**: Moves the view horizontally relative to its position.
- **void setTranslationY(float translationY)**: Moves the view vertically relative to its position.
- **void setRotation(float rotation)**: Rotates the view around its pivot point.
- **void setScaleX(float scaleX) / setScaleY(float scaleY)**: Scales the view's size in X or Y direction.
- **void setClickable(boolean clickable)**: Enables or disables clickability.
- **void setLongClickable(boolean longClickable)**: Enables long-click behavior.
- **boolean isClickable()**: Returns whether the view handles clicks.
- **boolean isLongClickable()**: Returns whether the view handles long clicks.
- **void setPressed(boolean pressed)**: Sets the pressed state for visual feedback.
- **boolean isPressed()**: Returns whether the view is currently pressed.
- **void setContentDescription(CharSequence contentDescription)**: Sets a description for accessibility tools.
- **CharSequence getContentDescription()**: Returns the view's accessibility description.
- **void announceForAccessibility(CharSequence text)**: Announces a message for accessibility services.

- **Protected methods**
  - **boolean awakenScrollBars(int startDelay, boolean invalidate)**: Trigger the scrollbars to draw.
  - **boolean awakenScrollBars(int startDelay)**: Trigger the scrollbars to draw.
  - **boolean awakenScrollBars()**: Trigger the scrollbars to draw.
  - **int computeHorizontalScrollExtent()**: Compute the horizontal extent of the scrollbar thumb within its range.
  - **int computeHorizontalScrollOffset()**: Compute the horizontal offset of the scrollbar thumb within its range.
  - **int computeHorizontalScrollRange()**: Compute the horizontal scrollable range.
  - **int computeVerticalScrollExtent()**: Compute the vertical extent of the scrollbar thumb within its range.
  - **int computeVerticalScrollOffset()**: Compute the vertical offset of the scrollbar thumb within its range.

- **int computeVerticalScrollRange()**: Compute the vertical scrollable range.
- **void dispatchDraw(Canvas canvas)**: Called by `draw()` to draw child views.
- **boolean dispatchGenericFocusedEvent(MotionEvent event)**: Dispatch a generic motion event to the currently focused view.
- **boolean dispatchGenericPointerEvent(MotionEvent event)**: Dispatch a generic motion event to the view under the first pointer.
- **boolean dispatchHoverEvent(MotionEvent event)**: Dispatch a hover event.
- **void dispatchRestoreInstanceState(SparseArray<Parcelable> container)**: Restores the state for this view and its children.
- **void dispatchSaveInstanceState(SparseArray<Parcelable> container)**: Saves the state for this view and its children.
- **void dispatchSetActivated(boolean activated)**: Dispatches activation to all of this view's children.
- **void dispatchSetPressed(boolean pressed)**: Dispatches pressed state to all of this view's children.
- **void dispatchSetSelected(boolean selected)**: Dispatches selected state to all of this view's children.
- **void dispatchVisibilityChanged(View changedView, int visibility)**: Propagates a visibility change down the hierarchy.
- **void drawableStateChanged()**: Called whenever the view's state changes in a way that affects its drawables.
- **boolean fitSystemWindows(Rect insets)**: (Deprecated) Apply window insets to adjust for system decorations.
- **float getBottomFadingEdgeStrength()**: Returns the intensity of the bottom faded edge.
- **int getBottomPaddingOffset()**: Amount by which to extend the bottom fading region.
- **float getLeftFadingEdgeStrength()**: Returns the intensity of the left faded edge.
- **int getLeftPaddingOffset()**: Amount by which to extend the left fading region.
- **float getRightFadingEdgeStrength()**: Returns the intensity of the right faded edge.
- **int getRightPaddingOffset()**: Amount by which to extend the right fading region.
- **float getTopFadingEdgeStrength()**: Returns the intensity of the top faded edge.
- **int getTopPaddingOffset()**: Amount by which to extend the top fading region.
- **boolean isPaddingOffsetRequired()**: Returns true if this view draws inside its padding and requires offset support.
- **int getSuggestedMinimumHeight()**: Returns the suggested minimum height for this view.
- **int getSuggestedMinimumWidth()**: Returns the suggested minimum width for this view.
- **boolean overScrollBy(int deltaX, int deltaY, int scrollX, int scrollY, int scrollRangeX, int scrollRangeY, int maxOverScrollX, int maxOverScrollY, boolean isTouchEvent)**: Scrolls the view with standard over-scroll behavior.

126

- **void onOverScrolled(int scrollX, int scrollY, boolean clampedX, boolean clampedY)**: Responds to an over-scroll operation.
- **void onScrollChanged(int l, int t, int oldl, int oldt)**: Called when the view scrolls its own content.
- **void onAttachedToWindow()**: Called when the view is attached to a window.
- **void onDetachedFromWindow()**: Called when the view is detached from a window.
- **void onConfigurationChanged(Configuration newConfig)**: Called when the app configuration changes (e.g., orientation).
- **void onDisplayHint(int hint)**: Receives a hint about whether the view is displayed or not.
- **int getWindowAttachCount()**: Returns how many times this view has been attached to a window.
- **void onLayout(boolean changed, int left, int top, int right, int bottom)**: Called when assigning size and position to child views.
- **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: Measures the view's width and height.
- **final void setMeasuredDimension(int measuredWidth, int measuredHeight)**: Must be called in `onMeasure()` to store measured dimensions.
- **void onSizeChanged(int w, int h, int oldw, int oldh)**: Called when the size of this view changes during layout.
- **void onDraw(Canvas canvas)**: Implement this to perform custom drawing.
- **final void onDrawScrollBars(Canvas canvas)**: Draws horizontal and vertical scrollbars.
- **void onAnimationStart()**: Called when an animation starts.
- **void onAnimationEnd()**: Called when an animation ends.
- **boolean onSetAlpha(int alpha)**: Called when a transform involving alpha occurs.
- **void onCreateContextMenu(ContextMenu menu)**: Implement if the view contributes items to a context menu.
- **ContextMenu.ContextMenuInfo getContextMenuInfo()**: Returns extra info associated with the context menu.
- **int[] onCreateDrawableState(int extraSpace)**: Generates the new drawable state array for this view.
- **boolean verifyDrawable(Drawable who)**: Override if the view displays custom drawables; return true for those drawables.
- **static int[] mergeDrawableStates(int[] baseState, int[] additionalState)**: Merges additional drawable states into the base state.
- **void onFocusChanged(boolean gainFocus, int direction, Rect previouslyFocusedRect)**: Called when the focus state of this view changes.
- **void onVisibilityChanged(View changedView, int visibility)**: Called when the visibility of this view or its ancestor changes.
- **void onWindowVisibilityChanged(int visibility)**: Called when the containing window's visibility changes (GONE, INVISIBLE, or VISIBLE).
- **Parcelable onSaveInstanceState()**: Generates a representation of this view's internal state for later restoration.

127

- **void onRestoreInstanceState(Parcelable state)**: Restores the view's internal state from a saved instance.
- **void dispatchRestoreInstanceState(SparseArray<Parcelable> container)**: Restores hierarchy state for this view and its children.
- **void dispatchSaveInstanceState(SparseArray<Parcelable> container)**: Saves hierarchy state for this view and its children.

- **Fields**
  - **public static final Property<View, Float> ALPHA**: A Property wrapper around the alpha functionality handled by the `View.setAlpha(float)` and `View.getAlpha()` methods.
  - **public static final Property<View, Float> ROTATION**: A Property wrapper around the rotation functionality handled by `View.setRotation(float)` and `View.getRotation()`.
  - **public static final Property<View, Float> ROTATION__X**: A Property wrapper around the rotationX functionality handled by `View.setRotationX(float)` and `View.getRotationX()`.
  - **public static final Property<View, Float> ROTATION__Y**: A Property wrapper around the rotationY functionality handled by `View.setRotationY(float)` and `View.getRotationY()`.
  - **public static final Property<View, Float> SCALE__X**: A Property wrapper around the scaleX functionality handled by `View.setScaleX(float)` and `View.getScaleX()`.
  - **public static final Property<View, Float> SCALE__Y**: A Property wrapper around the scaleY functionality handled by `View.setScaleY(float)` and `View.getScaleY()`.
  - **public static final Property<View, Float> TRANSLATION__X**: A Property wrapper around the translationX functionality handled by `View.setTranslationX(float)` and `View.getTranslationX()`.
  - **public static final Property<View, Float> TRANSLATION__Y**: A Property wrapper around the translationY functionality handled by `View.setTranslationY(float)` and `View.getTranslationY()`.
  - **public static final Property<View, Float> TRANSLATION__Z**: A Property wrapper around the translationZ functionality handled by `View.setTranslationZ(float)` and `View.getTranslationZ()`.
  - **public static final Property<View, Float> X**: A Property wrapper around the x-position handled by `View.setX(float)` and `View.getX()`.
  - **public static final Property<View, Float> Y**: A Property wrapper around the y-position handled by `View.setY(float)` and `View.getY()`.
  - **public static final Property<View, Float> Z**: A Property wrapper around the z-position handled by `View.setZ(float)` and `View.getZ()`.
  - **protected static final int[] EMPTY__STATE__SET**: Indicates the view has no states set.
  - **protected static final int[] ENABLED__STATE__SET**: Indicates the view is enabled.
  - **protected static final int[] ENABLED__FOCUSED__STATE__SET**: Indicates the view is enabled and has focus.
  - **protected static final int[] ENABLED__SELECTED__STATE__SET**: Indicates the view is enabled and selected.

- **protected static final int[] ENABLED_FOCUSED_SELECTED_STATE_SET**: Indicates the view is enabled, focused, and selected.
- **protected static final int[] ENABLED_WINDOW_FOCUSED_STATE_SET**: Indicates the view is enabled and its window has focus.
- **protected static final int[] ENABLED_FOCUSED_WINDOW_FOCUSED_STATE_SET**: Indicates the view is enabled, focused, and its window has focus.
- **protected static final int[] ENABLED_SELECTED_WINDOW_FOCUSED_STATE_SET**: Indicates the view is enabled, selected, and its window has focus.
- **protected static final int[] ENABLED_FOCUSED_SELECTED_WINDOW_FOCUSED_STATE_SET**: Indicates the view is enabled, focused, selected, and its window has focus.
- **protected static final int[] FOCUSED_STATE_SET**: Indicates the view is focused.
- **protected static final int[] SELECTED_STATE_SET**: Indicates the view is selected.
- **protected static final int[] WINDOW_FOCUSED_STATE_SET**: Indicates the view's window has focus.
- **protected static final int[] PRESSED_STATE_SET**: Indicates the view is pressed.
- **protected static final int[] PRESSED_ENABLED_STATE_SET**: Indicates the view is pressed and enabled.
- **protected static final int[] PRESSED_ENABLED_FOCUSED_STATE_SET**: Indicates the view is pressed, enabled, and focused.
- **protected static final int[] PRESSED_ENABLED_SELECTED_STATE_SET**: Indicates the view is pressed, enabled, and selected.
- **protected static final int[] PRESSED_ENABLED_FOCUSED_SELECTED_STATE_SET**: Indicates the view is pressed, enabled, focused, and selected.
- **protected static final int[] PRESSED_FOCUSED_STATE_SET**: Indicates the view is pressed and focused.
- **protected static final int[] PRESSED_SELECTED_STATE_SET**: Indicates the view is pressed and selected.
- **protected static final int[] FOCUSED_SELECTED_STATE_SET**: Indicates the view is focused and selected.
- **protected static final int[] PRESSED_ENABLED_FOCUSED_SELECTED_WINDOW_FOCUSED_STATE_SET**: Indicates the view is pressed, enabled, focused, selected, and its window has the focus.
- **protected static final int[] PRESSED_ENABLED_WINDOW_FOCUSED_STATE_SET**: Indicates the view is pressed, enabled, and its window has focus.
- **protected static final int[] PRESSED_FOCUSED_WINDOW_FOCUSED_STATE_SET**: Indicates the view is pressed, focused, and its window has focus.
- **protected static final int[] PRESSED_SELECTED_WINDOW_FOCUSED_STATE_SET**: Indicates the view is pressed, selected, and its window has focus.

- **protected static final int[] SELECTED_WINDOW_FOCUSED_STATE_SET**: Indicates the view is selected and its window has focus.
  - **protected static final int[] FOCUSED_WINDOW_FOCUSED_STATE_SET**: Indicates the view is focused and its window has focus.
  - **protected static final int[] PRESSED_FOCUSED_SELECTED_STATE_SET**: Indicates the view is pressed, focused, and selected.
  - **protected static final int[] PRESSED_FOCUSED_SELECTED_WINDOW_FOCUSED_STATE_SET**: Indicates the view is pressed, focused, selected, and its window has focus.

- **Constants**
  - **int ACCESSIBILITY_DATA_SENSITIVE_AUTO**: Automatically determine whether only accessibility tools may interact with this view.
  - **int ACCESSIBILITY_DATA_SENSITIVE_NO**: Allow interactions from all AccessibilityServices.
  - **int ACCESSIBILITY_DATA_SENSITIVE_YES**: Only allow interactions from AccessibilityServices marked as tools.
  - **int ACCESSIBILITY_LIVE_REGION_ASSERTIVE**: Announce changes immediately.
  - **int ACCESSIBILITY_LIVE_REGION_NONE**: Do not automatically announce changes.
  - **int ACCESSIBILITY_LIVE_REGION_POLITE**: Announce changes politely.
  - **int AUTOFILL_FLAG_INCLUDE_NOT_IMPORTANT_VIEWS**: Include not-important-for-autofill views in `ViewStructure`.
  - **String AUTOFILL_HINT_CREDIT_CARD_EXPIRATION_DATE**: Hint for credit card expiration date.
  - **String AUTOFILL_HINT_CREDIT_CARD_EXPIRATION_DAY**: Hint for credit card expiration day.
  - **String AUTOFILL_HINT_CREDIT_CARD_EXPIRATION_MONTH**: Hint for credit card expiration month.
  - **String AUTOFILL_HINT_CREDIT_CARD_EXPIRATION_YEAR**: Hint for credit card expiration year.
  - **String AUTOFILL_HINT_CREDIT_CARD_NUMBER**: Hint for credit card number.
  - **String AUTOFILL_HINT_CREDIT_CARD_SECURITY_CODE**: Hint for credit card CVC/CVV.
  - **String AUTOFILL_HINT_EMAIL_ADDRESS**: Hint for email address.
  - **String AUTOFILL_HINT_NAME**: Hint for real name.
  - **String AUTOFILL_HINT_PASSWORD**: Hint for password.
  - **String AUTOFILL_HINT_PHONE**: Hint for phone number.
  - **String AUTOFILL_HINT_POSTAL_ADDRESS**: Hint for postal address.
  - **String AUTOFILL_HINT_POSTAL_CODE**: Hint for postal/ZIP code.
  - **String AUTOFILL_HINT_USERNAME**: Hint for username.
  - **int AUTOFILL_TYPE_DATE**: Field is a date (millis since epoch).
  - **int AUTOFILL_TYPE_LIST**: Field is a selection list (int index).

- **int AUTOFILL_TYPE_NONE**: Not autofillable.
- **int AUTOFILL_TYPE_TEXT**: Field is text.
- **int AUTOFILL_TYPE_TOGGLE**: Field is boolean/toggle.
- **int CONTENT_SENSITIVITY_AUTO**: Framework determines content sensitivity.
- **int CONTENT_SENSITIVITY_NOT_SENSITIVE**: Content not sensitive.
- **int CONTENT_SENSITIVITY_SENSITIVE**: Content is sensitive.
- **int DRAG_FLAG_ACCESSIBILITY_ACTION**: Drag initiated via accessibility action.
- **int DRAG_FLAG_GLOBAL**: Drag can cross window boundaries.
- **int DRAG_FLAG_GLOBAL_PERSISTABLE_URI_PERMISSION**: Persist granted URI permissions across reboots.
- **int DRAG_FLAG_GLOBAL_PREFIX_URI_PERMISSION**: URI permission applies to prefix matches.
- **int DRAG_FLAG_GLOBAL_SAME_APPLICATION**: Drag can cross windows within same app.
- **int DRAG_FLAG_GLOBAL_URI_READ**: Recipient may request read access to URIs.
- **int DRAG_FLAG_GLOBAL_URI_WRITE**: Recipient may request write access to URIs.
- **int DRAG_FLAG_HIDE_CALLING_TASK_ON_DRAG_START**: Hide caller task during drag.
- **int DRAG_FLAG_OPAQUE**: Drag shadow is opaque.
- **int DRAG_FLAG_START_INTENT_SENDER_ON_UNHANDLED_DRAG**: Delegate unhandled drag to system to start.
- **int DRAWING_CACHE_QUALITY_AUTO**: *Deprecated.* Auto drawing cache quality.
- **int DRAWING_CACHE_QUALITY_HIGH**: *Deprecated.* High drawing cache quality.
- **int DRAWING_CACHE_QUALITY_LOW**: *Deprecated.* Low drawing cache quality.
- **int FIND_VIEWS_WITH_CONTENT_DESCRIPTION**: Find by content description.
- **int FIND_VIEWS_WITH_TEXT**: Find by text.
- **int FOCUSABLE**: View wants keystrokes.
- **int FOCUSABLES_ALL**: Add all focusables, regardless of touch mode.
- **int FOCUSABLES_TOUCH_MODE**: Add only focusables in touch mode.
- **int FOCUSABLE_AUTO**: Determine focusability automatically.
- **int FOCUS_BACKWARD**: Focus search backward.
- **int FOCUS_DOWN**: Focus search down.
- **int FOCUS_FORWARD**: Focus search forward.
- **int FOCUS_LEFT**: Focus search left.
- **int FOCUS_RIGHT**: Focus search right.
- **int FOCUS_UP**: Focus search up.

- **int GONE**: View is hidden and takes no space.
- **int HAPTIC_FEEDBACK_ENABLED**: Enable haptic feedback.
- **int IMPORTANT_FOR_ACCESSIBILITY_AUTO**: Determine importance for accessibility automatically.
- **int IMPORTANT_FOR_ACCESSIBILITY_NO**: Not important for accessibility.
- **int IMPORTANT_FOR_ACCESSIBILITY_NO_HIDE_DESCENDANTS**: Neither view nor descendants are important.
- **int IMPORTANT_FOR_ACCESSIBILITY_YES**: Important for accessibility.
- **int IMPORTANT_FOR_AUTOFILL_AUTO**: Determine importance for autofill automatically.
- **int IMPORTANT_FOR_AUTOFILL_NO**: Not important for autofill; traverse children.
- **int IMPORTANT_FOR_AUTOFILL_NO_EXCLUDE_DESCENDANTS**: Not important; do not traverse children.
- **int IMPORTANT_FOR_AUTOFILL_YES**: Important; traverse children.
- **int IMPORTANT_FOR_AUTOFILL_YES_EXCLUDE_DESCENDANTS**: Important; do not traverse children.
- **int IMPORTANT_FOR_CONTENT_CAPTURE_AUTO**: Determine importance for content capture automatically.
- **int IMPORTANT_FOR_CONTENT_CAPTURE_NO**: Not important; traverse children.
- **int IMPORTANT_FOR_CONTENT_CAPTURE_NO_EXCLUDE_DESCENDANTS**: Not important; exclude children.
- **int IMPORTANT_FOR_CONTENT_CAPTURE_YES**: Important; traverse children.
- **int IMPORTANT_FOR_CONTENT_CAPTURE_YES_EXCLUDE_DESCENDANTS**: Important; exclude children.
- **int INVISIBLE**: View is invisible but takes space.
- **int KEEP_SCREEN_ON**: Keep screen on while visible.
- **int LAYER_TYPE_HARDWARE**: Hardware layer.
- **int LAYER_TYPE_NONE**: No layer.
- **int LAYER_TYPE_SOFTWARE**: Software layer.
- **int LAYOUT_DIRECTION_INHERIT**: Inherit layout direction from parent.
- **int LAYOUT_DIRECTION_LOCALE**: Layout direction from locale script.
- **int LAYOUT_DIRECTION_LTR**: Left-to-right layout.
- **int LAYOUT_DIRECTION_RTL**: Right-to-left layout.
- **int MEASURED_HEIGHT_STATE_SHIFT**: Bit shift to height state.
- **int MEASURED_SIZE_MASK**: Mask for measured size bits.
- **int MEASURED_STATE_MASK**: Mask for measured state bits.
- **int MEASURED_STATE_TOO_SMALL**: Measured size is smaller than desired.
- **int NOT_FOCUSABLE**: View does not want keystrokes.

- **int NO_ID**: Marks a view with no ID.
- **int OVER_SCROLL_ALWAYS**: Always allow overscroll.
- **int OVER_SCROLL_IF_CONTENT_SCROLLS**: Allow overscroll only if content can scroll.
- **int OVER_SCROLL_NEVER**: Never allow overscroll.
- **int RECTANGLE_ON_SCREEN_REQUEST_SOURCE_INPUT_FO-CUS**: Request due to input focus.
- **int RECTANGLE_ON_SCREEN_REQUEST_SOURCE_SCROLL_ONLY**: Request only to scroll, not tied to cursor/focus.
- **int RECTANGLE_ON_SCREEN_REQUEST_SOURCE_TEXT_CUR-SOR**: Request due to text cursor.
- **int RECTANGLE_ON_SCREEN_REQUEST_SOURCE_UNDEFINED**: Request via legacy APIs (undefined source).
- **float REQUESTED_FRAME_RATE_CATEGORY_DEFAULT**: Preferred frame rate: default.
- **float REQUESTED_FRAME_RATE_CATEGORY_HIGH**: Preferred frame rate: high.
- **float REQUESTED_FRAME_RATE_CATEGORY_LOW**: Preferred frame rate: low.
- **float REQUESTED_FRAME_RATE_CATEGORY_NORMAL**: Preferred frame rate: normal.
- **float REQUESTED_FRAME_RATE_CATEGORY_NO_PREFER-ENCE**: No frame rate preference.
- **int SCREEN_STATE_OFF**: Screen is off.
- **int SCREEN_STATE_ON**: Screen is on.
- **int SCROLLBARS_INSIDE_INSET**: Scrollbars inside padded area; increases padding.
- **int SCROLLBARS_INSIDE_OVERLAY**: Scrollbars inside content; no padding increase.
- **int SCROLLBARS_OUTSIDE_INSET**: Scrollbars at edge; increases padding.
- **int SCROLLBARS_OUTSIDE_OVERLAY**: Scrollbars at edge; no padding increase.
- **int SCROLLBAR_POSITION_DEFAULT**: Scrollbar at system default position.
- **int SCROLLBAR_POSITION_LEFT**: Scrollbar on left edge.
- **int SCROLLBAR_POSITION_RIGHT**: Scrollbar on right edge.
- **int SCROLL_AXIS_HORIZONTAL**: Horizontal scroll axis.
- **int SCROLL_AXIS_NONE**: No scroll axis.
- **int SCROLL_AXIS_VERTICAL**: Vertical scroll axis.
- **int SCROLL_CAPTURE_HINT_AUTO**: Consider for scroll capture if scrollable.
- **int SCROLL_CAPTURE_HINT_EXCLUDE**: Exclude this view from scroll capture.
- **int SCROLL_CAPTURE_HINT_EXCLUDE_DESCENDANTS**: Exclude descendants from scroll capture.

- **int SCROLL_CAPTURE_HINT_INCLUDE**: Include this view for scroll capture.
- **int SCROLL_INDICATOR_BOTTOM**: Scroll indicator on bottom edge.
- **int SCROLL_INDICATOR_END**: Scroll indicator on end edge.
- **int SCROLL_INDICATOR_LEFT**: Scroll indicator on left edge.
- **int SCROLL_INDICATOR_RIGHT**: Scroll indicator on right edge.
- **int SCROLL_INDICATOR_START**: Scroll indicator on start edge.
- **int SCROLL_INDICATOR_TOP**: Scroll indicator on top edge.
- **int SOUND_EFFECTS_ENABLED**: Enable click/touch sound effects.
- **int STATUS_BAR_HIDDEN**: *Deprecated.* Use low profile instead.
- **int STATUS_BAR_VISIBLE**: *Deprecated.* Use `SYSTEM_UI_FLAG_VISIBLE`.
- **int SYSTEM_UI_FLAG_FULLSCREEN**: *Deprecated.* Use `WindowInsetsController.hide(Type`
- **int SYSTEM_UI_FLAG_HIDE_NAVIGATION**: *Deprecated.* Use `WindowInsetsController.hi`
- **int SYSTEM_UI_FLAG_IMMERSIVE**: *Deprecated.* Use default behavior.
- **int SYSTEM_UI_FLAG_IMMERSIVE_STICKY**: *Deprecated.* Use show-transient-bars-by-swipe behavior.
- **int SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN**: *Deprecated.* See modern insets APIs.
- **int SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION**: *Deprecated.* See modern insets APIs.
- **int SYSTEM_UI_FLAG_LAYOUT_STABLE**: *Deprecated.* Use `WindowInsets.getInsetsIgnori`
- **int SYSTEM_UI_FLAG_LIGHT_NAVIGATION_BAR**: *Deprecated.* Use appearance flags.
- **int SYSTEM_UI_FLAG_LIGHT_STATUS_BAR**: *Deprecated.* Use appearance flags.
- **int SYSTEM_UI_FLAG_LOW_PROFILE**: *Deprecated.* Hide system bars instead.
- **int SYSTEM_UI_FLAG_VISIBLE**: *Deprecated.* Use `WindowInsetsController`.
- **int SYSTEM_UI_LAYOUT_FLAGS**: *Deprecated.* System UI layout flags deprecated.
- **int TEXT_ALIGNMENT_CENTER**: Center paragraph alignment.
- **int TEXT_ALIGNMENT_GRAVITY**: Default for root view (gravity).
- **int TEXT_ALIGNMENT_INHERIT**: Inherit text alignment.
- **int TEXT_ALIGNMENT_TEXT_END**: Align to paragraph end.
- **int TEXT_ALIGNMENT_TEXT_START**: Align to paragraph start.
- **int TEXT_ALIGNMENT_VIEW_END**: Align to view end (RTL-aware).
- **int TEXT_ALIGNMENT_VIEW_START**: Align to view start (RTL-aware).
- **int TEXT_DIRECTION_ANY_RTL**: Any-RTL algorithm.
- **int TEXT_DIRECTION_FIRST_STRONG**: First-strong algorithm.
- **int TEXT_DIRECTION_FIRST_STRONG_LTR**: First-strong, force LTR.
- **int TEXT_DIRECTION_FIRST_STRONG_RTL**: First-strong, force RTL.

- **int TEXT_DIRECTION_INHERIT**: Inherit text direction.
- **int TEXT_DIRECTION_LOCALE**: From system locale.
- **int TEXT_DIRECTION_LTR**: Force LTR.
- **int TEXT_DIRECTION_RTL**: Force RTL.
- **String VIEW_LOG_TAG**: Logging tag for this class.
- **int VISIBLE**: View is visible.

1.4.2 ViewGroup

- **Hierarchy**

  java.lang.Object → android.view.View → android.view.ViewGroup

- **Include**

  ```
  0   android.view.ViewGroup
  ```

- **Constructors**

  ```
  0   ViewGroup(Context context)
  1   ViewGroup(Context context, AttributeSet attrs)
  2   ViewGroup(Context context, AttributeSet attrs, int
      ↪   defStyleAttr)
  3   ViewGroup(Context context, AttributeSet attrs, int
      ↪   defStyleAttr, int defStyleRes)
  ```

- **Public methods**

  - **void addView(View child)**: Adds a child view to this ViewGroup.
  - **void addView(View child, int index)**: Inserts a child view at a specific position.
  - **void addView(View child, ViewGroup.LayoutParams params)**: Adds a child with explicit layout params.
  - **void addView(View child, int index, ViewGroup.LayoutParams params)**: Inserts a child at a position with params.
  - **void addView(View child, int width, int height)**: Adds a child using default params plus width/height.
  - **void removeView(View view)**: Removes the specified child view.
  - **void removeViewAt(int index)**: Removes the child at the given index.
  - **void removeViews(int start, int count)**: Removes a range of children.
  - **void removeAllViews()** : Removes all child views.
  - **int getChildCount()**: Returns the number of children in this ViewGroup.
  - **View getChildAt(int index)**: Returns the child at the specified index.
  - **int indexOfChild(View child)**: Returns this child's index within the group.
  - **void bringChildToFront(View child)**: Moves a child to the top of the Z-order.
  - **boolean dispatchTouchEvent(MotionEvent ev)**: Dispatches touch events down the hierarchy.
  - **boolean onInterceptTouchEvent(MotionEvent ev)**: Intercepts touch events before children (gesture handling).
  - **void requestDisallowInterceptTouchEvent(boolean disallow)**: Child requests parent not to intercept touch.
  - **final void layout(int l, int t, int r, int b)**: Assigns size/position to this view and descendants.

– **static int getChildMeasureSpec(int spec, int padding, int childDimension)**: Computes a child's MeasureSpec.

– **ViewGroup.LayoutParams generateLayoutParams(AttributeSet attrs)**: Creates layout params from XML.

– **void setClipToPadding(boolean clipToPadding)**: Controls clipping of children within padding area.

– **void setClipChildren(boolean clipChildren)**: Controls whether children are clipped to this ViewGroup's bounds.

– **int getDescendantFocusability()** : Gets how focus is handled among descendants.

– **void setDescendantFocusability(int focusability)**: Sets focus behavior for descendants.

– **View getFocusedChild()** : Returns the currently focused child, if any.

– **View focusSearch(View focused, int direction)**: Finds the next focusable view in a direction.

– **void requestChildFocus(View child, View focused)**: Notifies parent that a child wants focus.

– **void setOnHierarchyChangeListener(ViewGroup.OnHierarchyChangeListener l)**: Listens for child add/remove events.

– **void setLayoutTransition(LayoutTransition transition)**: Animates child appearance/disappearance/changes.

– **void suppressLayout(boolean suppress)**: Temporarily defers layout passes for batched changes.

– **void updateViewLayout(View view, ViewGroup.LayoutParams params)**: Updates layout params for an existing child.

- **Protected methods**

  – **boolean addViewInLayout(View child, int index, ViewGroup.LayoutParams params, boolean preventRequestLayout)**: Adds a view during layout, optionally preventing a layout request.

  – **boolean addViewInLayout(View child, int index, ViewGroup.LayoutParams params)**: Adds a view during layout.

  – **void attachLayoutAnimationParameters(View child, ViewGroup.LayoutParams params, int index, int count)**: For subclasses to set layout animation parameters on the child.

  – **void attachViewToParent(View child, int index, ViewGroup.LayoutParams params)**: Attaches a view to this ViewGroup.

  – **boolean canAnimate()**: Returns whether this ViewGroup can animate its children after first layout.

  – **boolean checkLayoutParams(ViewGroup.LayoutParams p)**: Checks if the given LayoutParams are valid for this ViewGroup.

  – **void cleanupLayoutState(View child)**: Prevents the specified child from being laid out during the next layout pass.

  – **void debug(int depth)**: Outputs debug information about this view hierarchy.

  – **void detachAllViewsFromParent()**: Detaches all views from the parent.

  – **void detachViewFromParent(int index)**: Detaches the child at the given index from its parent.

- **void detachViewFromParent(View child)**: Detaches the specified child from its parent.
- **void detachViewsFromParent(int start, int count)**: Detaches a range of children from their parent.
- **void dispatchDraw(Canvas canvas)**: Called by `draw` to draw the child views.
- **void dispatchFreezeSelfOnly(SparseArray<Parcelable> container)**: Saves state only for this view (not its children).
- **boolean dispatchGenericFocusedEvent(MotionEvent event)**: Dispatches a generic motion event to the currently focused view.
- **boolean dispatchGenericPointerEvent(MotionEvent event)**: Dispatches a generic motion event to the view under the first pointer.
- **boolean dispatchHoverEvent(MotionEvent event)**: Dispatches a hover event.
- **void dispatchRestoreInstanceState(SparseArray<Parcelable> container)**: Restores state for this view and its children.
- **void dispatchSaveInstanceState(SparseArray<Parcelable> container)**: Saves state for this view and its children.
- **void dispatchSetPressed(boolean pressed)**: Propagates the pressed state to all children.
- **void dispatchThawSelfOnly(SparseArray<Parcelable> container)**: Restores state only for this view (not its children).
- **void dispatchVisibilityChanged(View changedView, int visibility)**: Dispatches visibility changes down the hierarchy.
- **boolean drawChild(Canvas canvas, View child, long drawingTime)**: Draws a single child of this ViewGroup.
- **void drawableStateChanged()**: Called when the view's state changes in a way that affects shown drawables.
- **ViewGroup.LayoutParams generateDefaultLayoutParams()**: Returns a set of default layout parameters.
- **ViewGroup.LayoutParams generateLayoutParams(ViewGroup.LayoutParams p)**: Returns a safe set of layout params based on the supplied params.
- **int getChildDrawingOrder(int childCount, int drawingPosition)**: Maps drawing-order position to container position.
- **boolean getChildStaticTransformation(View child, Transformation t)**: Sets `t` to the child's static transform if present; returns true if set.
- **boolean isChildrenDrawingOrderEnabled()**: Returns whether children are drawn in the order from `getChildDrawingOrder`.
- **boolean isChildrenDrawnWithCacheEnabled()**: Deprecated (API 23). Child caching forced by parents is ignored; use `View.setLayerType`.
- **void measureChild(View child, int parentWidthMeasureSpec, int parentHeightMeasureSpec)**: Measures a child with this ViewGroup's specs and padding.
- **void measureChildWithMargins(View child, int parentWidthMeasureSpec, int widthUsed, int parentHeightMeasureSpec, int heightUsed)**: Measures a child accounting for margins and used space.
- **void measureChildren(int widthMeasureSpec, int heightMeasureSpec)**: Measures all children with the given specs and padding.

- **void onAttachedToWindow()** : Called when the view is attached to a window.
- **int[] onCreateDrawableState(int extraSpace)**: Generates the Drawable state for this view.
- **void onDetachedFromWindow()** : Called when the view is detached from a window.
- **abstract void onLayout(boolean changed, int l, int t, int r, int b)**: Assigns size and position to each child (subclasses must implement).
- **boolean onRequestFocusInDescendants(int direction, Rect previouslyFocusedRect)**: Requests focus on a suitable descendant.
- **void removeDetachedView(View child, boolean animate)**: Finishes removing a detached view, optionally with animation.
- **void setChildrenDrawingCacheEnabled(boolean enabled)**: Deprecated (API 28). View drawing cache largely obsolete with hardware acceleration; prefer `View.setLayerType` or `PixelCopy` for screenshots.
- **void setChildrenDrawingOrderEnabled(boolean enabled)**: Controls whether children are drawn using custom drawing order.
- **void setChildrenDrawnWithCacheEnabled(boolean enabled)**: Deprecated (API 23). Forcing child render caching is ignored; use `View.setLayerType`.
- **void setStaticTransformationsEnabled(boolean enabled)**: Enables static child transformations (invokes `getChildStaticTransformation` during draw).

- **Constants**

  - **int CLIP_TO_PADDING_MASK**: Clips to padding when both `FLAG_CLIP_TO_PADDING` and `FLAG_PADDING_NOT_NULL` are set.
  - **int FOCUS_AFTER_DESCENDANTS**: The view receives focus only if none of its descendants request it.
  - **int FOCUS_BEFORE_DESCENDANTS**: The view receives focus before any of its descendants.
  - **int FOCUS_BLOCK_DESCENDANTS**: Prevents any descendants from receiving focus, even if they are focusable.
  - **int LAYOUT_MODE_CLIP_BOUNDS**: Layout mode constant that aligns layout to the view's clip bounds.
  - **int LAYOUT_MODE_OPTICAL_BOUNDS**: Layout mode constant that aligns layout to the view's optical bounds.
  - **int PERSISTENT_ALL_CACHES**: *Deprecated in API 28.* Formerly kept all drawing caches (animation, scrolling, etc.). Superseded by hardware acceleration and `View.setLayerType(int, Paint)`.
  - **int PERSISTENT_ANIMATION_CACHE**: *Deprecated in API 28.* Formerly kept only animation caches. Hardware acceleration now handles such effects efficiently.
  - **int PERSISTENT_NO_CACHE**: *Deprecated in API 28.* Disabled view drawing caches. Replaced by hardware rendering mechanisms.
  - **int PERSISTENT_SCROLLING_CACHE**: *Deprecated in API 28.* Formerly maintained caches for scrolling operations; hardware acceleration now replaces this feature.

1.4.3 Context

- **Hierarchy**

$$\text{java.lang.Object} \rightarrow \text{android.content.Context}$$

- **Include**

  o `android.content.Context`

- **Constructors**

  o `Context()`

- **Public methods**

  - **Context getApplicationContext()**: Returns the global application context.
  - **Resources getResources()**: Provides access to the app's resources (layouts, strings, drawables, etc.).
  - **PackageManager getPackageManager()**: Returns a PackageManager for querying installed apps and permissions.
  - **ContentResolver getContentResolver()**: Gives access to content providers (e.g., Contacts, MediaStore).
  - **SharedPreferences getSharedPreferences(String name, int mode)**: Access or create a preferences file for storing key–value pairs.
  - **File getFilesDir()**: Returns the app's private file storage directory.
  - **File getCacheDir()**: Returns the app's private cache directory.
  - **Drawable getDrawable(int id)**: Retrieves a drawable resource styled for the current theme.
  - **int getColor(int id)**: Returns a color resource styled for the current theme.
  - **String getString(int resId)**: Returns a localized string from resources.
  - **String getString(int resId, Object... formatArgs)**: Returns a formatted localized string.
  - **void startActivity(Intent intent)**: Launches a new activity.
  - **ComponentName startService(Intent service)**: Starts a service.
  - **boolean stopService(Intent service)**: Stops a running service.
  - **boolean bindService(Intent service, ServiceConnection conn, int flags)**: Connects to a service for interaction.
  - **void unbindService(ServiceConnection conn)**: Disconnects from a bound service.
  - **void sendBroadcast(Intent intent)**: Sends a broadcast to all interested receivers.
  - **Intent registerReceiver(BroadcastReceiver receiver, IntentFilter filter)**: Registers a broadcast receiver.
  - **void unregisterReceiver(BroadcastReceiver receiver)**: Unregisters a previously registered receiver.

– **Object getSystemService(String name)**: Returns a handle to a system-level service (e.g., `LAYOUT_INFLATER_SERVICE`).

– **<T> T getSystemService(Class<T> serviceClass)**: Type-safe version of `getSystemService`.

– **Resources.Theme getTheme()**: Returns the current theme for styling and inflation.

– **TypedArray obtainStyledAttributes(int[] attrs)**: Retrieves styled attributes in the current theme.

– **FileInputStream openFileInput(String name)**: Opens a private file for reading.

– **FileOutputStream openFileOutput(String name, int mode)**: Opens a private file for writing.

- **Constants**

  – **String ACTIVITY_SERVICE**: For `ActivityManager` (process/app state) via `getSystemService`.

  – **String WINDOW_SERVICE**: For `WindowManager` (windows, display metrics).

  – **String DISPLAY_SERVICE**: For `DisplayManager` (displays, modes).

  – **String LAYOUT_INFLATER_SERVICE**: For `LayoutInflater` (inflate XML layouts).

  – **String POWER_SERVICE**: For `PowerManager` (wake locks, power state).

  – **String UI_MODE_SERVICE**: For `UiModeManager` (night mode, car/TV mode).

  – **String CONNECTIVITY_SERVICE**: For `ConnectivityManager` (network state, requests).

  – **String WIFI_SERVICE**: For `WifiManager` (Wi-Fi control).

  – **String WIFI_P2P_SERVICE**: For `WifiP2pManager` (Wi-Fi Direct).

  – **String TETHERING_SERVICE**: For `TetheringManager` (tethering APIs).

  – **String USB_SERVICE**: For `UsbManager` (USB host/device).

  – **String NFC_SERVICE**: For `NfcManager` (NFC features).

  – **String VPN_MANAGEMENT_SERVICE**: For `VpnManager` (built-in VPN profiles).

  – **String LOCATION_SERVICE**: For `LocationManager` (location providers).

  – **String SENSOR_SERVICE**: For `SensorManager` (accelerometer, gyro, etc.).

  – **String BLUETOOTH_SERVICE**: For `BluetoothManager` (Bluetooth stack).

  – **String CAMERA_SERVICE**: For `CameraManager` (camera devices).

  – **String AUDIO_SERVICE**: For `AudioManager` (volume, routing).

  – **String NOTIFICATION_SERVICE**: For `NotificationManager` (post/cancel notifications).

  – **String MEDIA_SESSION_SERVICE**: For `MediaSessionManager` (media controls).

  – **String MEDIA_PROJECTION_SERVICE**: For `MediaProjectionManager` (screen capture).

  – **String DOWNLOAD_SERVICE**: For `DownloadManager` (HTTP downloads).

- **String INPUT_METHOD_SERVICE**: For `InputMethodManager` (soft keyboard).
- **String CLIPBOARD_SERVICE**: For `ClipboardManager` (global clipboard).
- **String KEYGUARD_SERVICE**: For `KeyguardManager` (lock screen).
- **String BIOMETRIC_SERVICE**: For `BiometricManager` (biometric auth).
- **String STORAGE_SERVICE**: For `StorageManager` (volumes, storage ops).
- **String USER_SERVICE**: For `UserManager` (multi-user info).
- **String JOB_SCHEDULER_SERVICE**: For `JobScheduler` (deferrable background work).
- **String APP_OPS_SERVICE**: For `AppOpsManager` (app operation checks).
- **String VIBRATOR_MANAGER_SERVICE**: For `VibratorManager` (multi-vibrator control).
- **int MODE_PRIVATE**: Default file mode for `openFileOutput`; file private to the app.
- **int MODE_APPEND**: Append mode for `openFileOutput`.
- **int MODE_ENABLE_WRITE_AHEAD_LOGGING**: DB flag to enable WAL by default.
- **int MODE_NO_LOCALIZED_COLLATORS**: DB flag to omit localized collators.
- **int RECEIVER_EXPORTED**: `registerReceiver` flag — receiver accepts broadcasts from other apps.
- **int RECEIVER_NOT_EXPORTED**: `registerReceiver` flag — receiver is app-internal only.
- **int RECEIVER_VISIBLE_TO_INSTANT_APPS**: `registerReceiver` flag — visible to Instant Apps.
- **int BIND_AUTO_CREATE**: Auto-create service while bound.
- **int BIND_NOT_FOREGROUND**: Do not raise target service to foreground priority.
- **int BIND_IMPORTANT**: Treat service as important to the client.
- **int BIND_DEBUG_UNBIND**: Include debugging help for unbind mismatches.
- **int BIND_WAIVE_PRIORITY**: Do not affect service process priority.

1.4.4 ConstraintLayout

- **Hierarchy**:

  java.lang.Object → android.view.View → android.view.ViewGroup →
  androidx.constraintlayout.widget.ConstraintLayout

- **Include**:

```
0    androidx.constraintlayout.widget.ConstraintLayout
```

- **Constructors**:

```
0    ConstraintLayout(@NonNull Context context)
1    ConstraintLayout(@NonNull Context context, @Nullable
 ↪    AttributeSet attrs)
2    ConstraintLayout( @NonNull Context context, @Nullable
 ↪    AttributeSet attrs, int defStyleAttr)
3
4    @TargetApi(value = Build.VERSION_CODES.LOLLIPOP)
5    ConstraintLayout( @NonNull Context context, @Nullable
 ↪    AttributeSet attrs, int defStyleAttr, int defStyleRes)
```

- **Public methods**:

  - **void addValueModifier(ConstraintLayout.ValueModifier modifier)**: Adds
    a `ValueModifier` to the `ConstraintLayout`.
  - **void fillMetrics(Metrics metrics)**: Populates the provided `Metrics` object
    with performance and measurement data.
  - **void forceLayout()**: Forces a layout pass, marking the layout as needing to be
    re-measured and re-laid out.
  - **ConstraintLayout.LayoutParams generateLayoutParams(AttributeSet
    attrs)**: Returns a new set of layout parameters based on the supplied attributes
    set.
  - **Object getDesignInformation(int type, Object value)**: Retrieves design-
    time information associated with the layout.
  - **int getMaxHeight()**: Returns the maximum height of this view.
  - **int getMaxWidth()**: Returns the maximum width of this view.
  - **int getMinHeight()**: Returns the minimum height of this view.
  - **int getMinWidth()**: Returns the minimum width of this view.
  - **int getOptimizationLevel()**: Returns the current optimization level for the
    layout resolution.
  - **String getSceneString()**: Returns a JSON5 string useful for debugging the
    constraints currently applied.
  - **static SharedValues getSharedValues()**: Returns the `SharedValues` instance,
    creating it if it does not already exist.
  - **View getViewById(int id)**: Returns the `View` corresponding to the given ID.
  - **final ConstraintWidget getViewWidget(View view)**: Returns the internal
    `ConstraintWidget` associated with a given view.

- **void loadLayoutDescription(int layoutDescription)**: Loads a layout description file from the application's resources.
- **void onViewAdded(View view)**: Called when a child view is added to the layout.
- **void onViewRemoved(View view)**: Called when a child view is removed from the layout.
- **void requestLayout()**: Requests a re-layout of this view hierarchy.
- **void setConstraintSet(ConstraintSet set)**: Sets a `ConstraintSet` object to manage constraints.
- **void setDesignInformation(int type, Object value1, Object value2)**: Stores design-time information associated with the layout.
- **void setId(int id)**: Sets the ID for this view.
- **void setMaxHeight(int value)**: Sets the maximum height for this view.
- **void setMaxWidth(int value)**: Sets the maximum width for this view.
- **void setMinHeight(int value)**: Sets the minimum height for this view.
- **void setMinWidth(int value)**: Sets the minimum width for this view.
- **void setOnConstraintsChanged(ConstraintsChangedListener constraintsChangedListener)**: Registers a listener to be notified when constraints change.
- **void setOptimizationLevel(int level)**: Sets the optimization level for layout resolution.
- **void setState(int id, int screenWidth, int screenHeight)**: Sets the state of the `ConstraintLayout`, causing it to load a specific `ConstraintSet`.
- **boolean shouldDelayChildPressedState()**: Returns true if the pressed state should be delayed for children or descendants of this `ViewGroup`.

- **Protected methods**:

  - **void applyConstraintsFromLayoutParams(boolean isInEditMode, View child, ConstraintWidget widget, ConstraintLayout.LayoutParams layoutParams, SparseArray<ConstraintWidget> idToWidget)**: Applies constraints from the given layout parameters to the specified `ConstraintWidget`.
  - **boolean checkLayoutParams(ViewGroup.LayoutParams p)**: Determines whether the supplied layout parameters are valid for this layout.
  - **void dispatchDraw(Canvas canvas)**: Called to draw the layout's children onto the provided `Canvas`.
  - **boolean dynamicUpdateConstraints(int widthMeasureSpec, int heightMeasureSpec)**: Can be overridden to change how `ValueModifier`s are used during dynamic updates of constraints.
  - **ConstraintLayout.LayoutParams generateDefaultLayoutParams()**: Returns a set of default layout parameters for this `ConstraintLayout`.
  - **ViewGroup.LayoutParams generateLayoutParams(ViewGroup.LayoutParams p)**: Returns a safe set of layout parameters based on the supplied parameters.
  - **boolean isRtl()**: Returns `true` if the layout direction is right-to-left (RTL).
  - **void onLayout(boolean changed, int left, int top, int right, int bottom)**: Called during layout to assign a size and position to each child.
  - **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: Measures the layout and its children to determine width and height.

144

- **void parseLayoutDescription(int id)**: Called to handle layout descriptions; subclasses may override this method.

- **void resolveMeasuredDimension(int widthMeasureSpec, int heightMeasureSpec, int measuredWidth, int measuredHeight, boolean isWidthMeasuredTooSmall, boolean isHeightMeasuredTooSmall)**: Handles setting the measured dimensions for the layout.

- **void resolveSystem(ConstraintWidgetContainer layout, int optimizationLevel, int widthMeasureSpec, int heightMeasureSpec)**: Handles the measuring and constraint resolution of the layout.

- **void setSelfDimensionBehaviour(ConstraintWidgetContainer layout, int widthMode, int widthSize, int heightMode, int heightSize)**: Configures the layout's own dimension behavior during constraint resolution.

- **Constants**:

    - **static final int DESIGN_INFO_ID = 0**:
    - **static final String VERSION = "ConstraintLayout-2.2.0-alpha04"**:

- **Protected fields**:

    - **ConstraintLayoutStates mConstraintLayoutSpec**:
    - **boolean mDirtyHierarchy**:
    - **ConstraintWidgetContainer mLayoutWidget**:

1.4.5 ConstraintLayout.LayoutParams

- **Hierarchy**:

  java.lang.Object → android.view.View → android.view.ViewGroup →
  androidx.constraintlayout.widget.ConstraintLayout

- **Include**:

  ```
  0   androidx.constraintlayout.widget.ConstraintLayout
  ```

- **Constructors**:

  ```
  0   LayoutParams(ViewGroup.LayoutParams params)
  1   LayoutParams(Context c, AttributeSet attrs)
  2   LayoutParams(int width, int height)
  ```

- **Public methods**:

  - **String getConstraintTag()**: Returns a tag that can be used to identify a view as being part of a constraint group.
  - **ConstraintWidget getConstraintWidget()**: Returns the underlying `ConstraintWidget` object associated with this layout parameter or view.
  - **void reset()**: Resets the associated `ConstraintWidget` to its default state.
  - **void resolveLayoutDirection(int layoutDirection)**: Resolves layout direction–dependent constraints such as start/end alignment.
  - **void setWidgetDebugName(String text)**: Sets a debug name for the `ConstraintWidget`, useful for diagnostics or logging.
  - **void validate()**: Validates the layout and ensures that all parameters and constraints are consistent.

- **Public fields**:

  - **int baselineMargin**: The baseline margin.
  - **int baselineToBaseline**: Constrains the baseline of a child to the baseline of a target child (contains the target child ID).
  - **int baselineToBottom**: Constrains the baseline of a child to the bottom of a target child (contains the target child ID).
  - **int baselineToTop**: Constrains the baseline of a child to the top of a target child (contains the target child ID).
  - **int bottomToBottom**: Constrains the bottom side of a child to the bottom side of a target child (contains the target child ID).
  - **int bottomToTop**: Constrains the bottom side of a child to the top side of a target child (contains the target child ID).
  - **float circleAngle**: The angle used for a circular constraint.
  - **int circleConstraint**: Constrains the center of a child to the center of a target child (contains the target child ID).
  - **int circleRadius**: The radius used for a circular constraint.

146

– **boolean constrainedHeight**: Specifies if the vertical dimension is constrained when both top and bottom constraints are set and the dimension is not fixed.

– **boolean constrainedWidth**: Specifies if the horizontal dimension is constrained when both left and right constraints are set and the dimension is not fixed.

– **String constraintTag**: Defines a category of view to be used by helpers and `MotionLayout`.

– **String dimensionRatio**: The ratio information defining the aspect ratio of the view.

– **int editorAbsoluteX**: The design-time X coordinate (left position) of the child.

– **int editorAbsoluteY**: The design-time Y coordinate (top position) of the child.

– **int endToEnd**: Constrains the end side of a child to the end side of a target child (contains the target child ID).

– **int endToStart**: Constrains the end side of a child to the start side of a target child (contains the target child ID).

– **int goneBaselineMargin**: The baseline margin to use when the target is gone.

– **int goneBottomMargin**: The bottom margin to use when the target is gone.

– **int goneEndMargin**: The end margin to use when the target is gone.

– **int goneLeftMargin**: The left margin to use when the target is gone.

– **int goneRightMargin**: The right margin to use when the target is gone.

– **int goneStartMargin**: The start margin to use when the target is gone.

– **int goneTopMargin**: The top margin to use when the target is gone.

– **int guideBegin**: The distance of a guideline from the top or left edge of its parent.

– **int guideEnd**: The distance of a guideline from the bottom or right edge of its parent.

– **float guidePercent**: The ratio of the distance to the parent's sides.

– **boolean guidelineUseRtl**: Determines whether guideline position respects RTL layout direction.

– **boolean helped**: Indicates whether the view was modified by a helper.

– **float horizontalBias**: The ratio between two connections when left and right (or start and end) sides are constrained.

– **int horizontalChainStyle**: Defines how elements of a horizontal chain are positioned.

– **float horizontalWeight**: The child's weight used to distribute available horizontal space in a chain when using `MATCH_CONSTRAINT`.

– **int leftToLeft**: Constrains the left side of a child to the left side of a target child (contains the target child ID).

– **int leftToRight**: Constrains the left side of a child to the right side of a target child (contains the target child ID).

– **int matchConstraintDefaultHeight**: Defines how the widget's vertical dimension is handled when set to `MATCH_CONSTRAINT`.

– **int matchConstraintDefaultWidth**: Defines how the widget's horizontal dimension is handled when set to `MATCH_CONSTRAINT`.

– **int matchConstraintMaxHeight**: Specifies a maximum height for the widget.

– **int matchConstraintMaxWidth**: Specifies a maximum width for the widget.

- **int matchConstraintMinHeight**: Specifies a minimum height for the widget.
- **int matchConstraintMinWidth**: Specifies a minimum width for the widget.
- **float matchConstraintPercentHeight**: Specifies a percentage value when using the match-constraint percent mode for height.
- **float matchConstraintPercentWidth**: Specifies a percentage value when using the match-constraint percent mode for width.
- **int orientation**: The orientation of the layout (horizontal or vertical).
- **int rightToLeft**: Constrains the right side of a child to the left side of a target child (contains the target child ID).
- **int rightToRight**: Constrains the right side of a child to the right side of a target child (contains the target child ID).
- **int startToEnd**: Constrains the start side of a child to the end side of a target child (contains the target child ID).
- **int startToStart**: Constrains the start side of a child to the start side of a target child (contains the target child ID).
- **int topToBottom**: Constrains the top side of a child to the bottom side of a target child (contains the target child ID).
- **int topToTop**: Constrains the top side of a child to the top side of a target child (contains the target child ID).
- **float verticalBias**: The ratio between two connections when the top and bottom sides are constrained.
- **int verticalChainStyle**: Defines how elements of a vertical chain are positioned.
- **float verticalWeight**: The child's weight used to distribute available vertical space in a chain when using `MATCH_CONSTRAINT`.
- **int wrapBehaviorInParent**: Specifies how this view is considered during the parent's wrap computation:
  * `WRAP_BEHAVIOR_INCLUDED`: Included in both directions (default).
  * `WRAP_BEHAVIOR_HORIZONTAL_ONLY`: Included only horizontally.
  * `WRAP_BEHAVIOR_VERTICAL_ONLY`: Included only vertically.
  * `WRAP_BEHAVIOR_SKIPPED`: Excluded from wrap computation.

- **Constants**:
  - **static final int BASELINE = 5**: The baseline of the text in a view.
  - **static final int BOTTOM = 4**: The bottom side of a view.
  - **static final int CHAIN_PACKED = 2**: Chain packed style.
  - **static final int CHAIN_SPREAD = 0**: Chain spread style.
  - **static final int CHAIN_SPREAD_INSIDE = 1**: Chain spread inside style.
  - **static final int CIRCLE = 8**: Circle reference from a view.
  - **static final int END = 7**: The right side of a view in left-to-right languages.
  - **static final int GONE_UNSET = -2147483648**: Defines an ID that is not set (default unset state).
  - **static final int HORIZONTAL = 0**: The horizontal orientation.
  - **static final int LEFT = 1**: The left side of a view.
  - **static final int MATCH_CONSTRAINT = 0**: Dimension will be controlled by constraints.

- **static final int MATCH_CONSTRAINT_PERCENT = 2**: Sets `matchConstraintDefault*` percent mode to be based on a percent of another dimension (usually the parent). Used for `matchConstraintDefaultWidth` and `matchConstraintDefaultHeight`.

- **static final int MATCH_CONSTRAINT_SPREAD = 0**: Sets `matchConstraintDefault*` to spread as much as possible within its constraints.

- **static final int MATCH_CONSTRAINT_WRAP = 1**: Sets `matchConstraintDefault*` to wrap content size.

- **static final int PARENT_ID = 0**: References the ID of the parent layout.

- **static final int RIGHT = 2**: The right side of a view.

- **static final int START = 6**: The left side of a view in left-to-right languages.

- **static final int TOP = 3**: The top side of a view.

- **static final int UNSET = -1**: Defines an ID that is not set.

- **static final int VERTICAL = 1**: The vertical orientation.

- **static final int WRAP_BEHAVIOR_HORIZONTAL_ONLY = 1**: Specifies that wrapping occurs only horizontally.

- **static final int WRAP_BEHAVIOR_INCLUDED = 0**: Specifies that wrapping includes both horizontal and vertical directions (default).

- **static final int WRAP_BEHAVIOR_SKIPPED = 3**: Specifies that the widget is excluded from wrap computation.

- **static final int WRAP_BEHAVIOR_VERTICAL_ONLY = 2**: Specifies that wrapping occurs only vertically.

1.4.6 RelativeLayout

- **Hierarchy**:

  java.lang.Object → android.view.View → android.view.ViewGroup →
  android.widget.RelativeLayout

- **Include**:

```
0   android.widget.RelativeLayout
```

- **Constructors**:

```
0   RelativeLayout(Context context)
1   RelativeLayout(Context context, AttributeSet attrs)
2   RelativeLayout(Context context, AttributeSet attrs, int
    ↪  defStyleAttr)
3   RelativeLayout(Context context, AttributeSet attrs, int
    ↪  defStyleAttr, int defStyleRes)
```

- **Public methods**:

  - **RelativeLayout.LayoutParams generateLayoutParams(AttributeSet attrs)**: Returns a new set of layout parameters based on the supplied attributes set.
  - **CharSequence getAccessibilityClassName()**: Return the class name of this object to be used for accessibility purposes.
  - **int getBaseline()**: Return the offset of the widget's text baseline from the widget's top boundary.
  - **int getGravity()**: Describes how the child views are positioned.
  - **int getIgnoreGravity()**: Get the ID of the `View` to be ignored by gravity.
  - **void requestLayout()**: Call this when something has changed that invalidates the layout of this view.
  - **void setGravity(int gravity)**: Describes how the child views are positioned.
  - **void setHorizontalGravity(int horizontalGravity)**: Sets the horizontal gravity of the layout.
  - **void setIgnoreGravity(int viewId)**: Defines which `View` is ignored when gravity is applied.
  - **void setVerticalGravity(int verticalGravity)**: Sets the vertical gravity of the layout.
  - **boolean shouldDelayChildPressedState()**: Returns true if the pressed state should be delayed for children or descendants of this `ViewGroup`.

- **Protected methods**:

  - **boolean checkLayoutParams(ViewGroup.LayoutParams p)**: Determines whether the supplied layout parameters are valid for this layout.
  - **ViewGroup.LayoutParams generateDefaultLayoutParams()**: Returns a set of layout parameters with a width of `ViewGroup.LayoutParams.WRAP_CONTENT`, a height of `ViewGroup.LayoutParams.WRAP_CONTENT`, and no spanning.

- **ViewGroup.LayoutParams generateLayoutParams(ViewGroup.Layout-Params lp)**: Returns a safe set of layout parameters based on the supplied layout parameters.
- **void onLayout(boolean changed, int l, int t, int r, int b)**: Called from layout when this view should assign a size and position to each of its children.
- **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: Measures the view and its content to determine the measured width and height.

- **Constants**:
  - **int ABOVE**: Rule that aligns a child's bottom edge with another child's top edge.
  - **int ALIGN_BASELINE**: Rule that aligns a child's baseline with another child's baseline.
  - **int ALIGN_BOTTOM**: Rule that aligns a child's bottom edge with another child's bottom edge.
  - **int ALIGN_END**: Rule that aligns a child's end edge with another child's end edge.
  - **int ALIGN_LEFT**: Rule that aligns a child's left edge with another child's left edge.
  - **int ALIGN_PARENT_BOTTOM**: Rule that aligns the child's bottom edge with its `RelativeLayout` parent's bottom edge.
  - **int ALIGN_PARENT_END**: Rule that aligns the child's end edge with its `RelativeLayout` parent's end edge.
  - **int ALIGN_PARENT_LEFT**: Rule that aligns the child's left edge with its `RelativeLayout` parent's left edge.
  - **int ALIGN_PARENT_RIGHT**: Rule that aligns the child's right edge with its `RelativeLayout` parent's right edge.
  - **int ALIGN_PARENT_START**: Rule that aligns the child's start edge with its `RelativeLayout` parent's start edge.
  - **int ALIGN_PARENT_TOP**: Rule that aligns the child's top edge with its `RelativeLayout` parent's top edge.
  - **int ALIGN_RIGHT**: Rule that aligns a child's right edge with another child's right edge.
  - **int ALIGN_START**: Rule that aligns a child's start edge with another child's start edge.
  - **int ALIGN_TOP**: Rule that aligns a child's top edge with another child's top edge.
  - **int BELOW**: Rule that aligns a child's top edge with another child's bottom edge.
  - **int CENTER_HORIZONTAL**: Rule that centers the child horizontally with respect to the bounds of its `RelativeLayout` parent.
  - **int CENTER_IN_PARENT**: Rule that centers the child with respect to the bounds of its `RelativeLayout` parent.
  - **int CENTER_VERTICAL**: Rule that centers the child vertically with respect to the bounds of its `RelativeLayout` parent.
  - **int END_OF**: Rule that aligns a child's start edge with another child's end edge.

- **int LEFT_OF**: Rule that aligns a child's right edge with another child's left edge.
- **int RIGHT_OF**: Rule that aligns a child's left edge with another child's right edge.
- **int START_OF**: Rule that aligns a child's end edge with another child's start edge.
- **int TRUE**: Constant used for layout rules that take a boolean value.

1.4.7 RelativeLayout.LayoutParams

- **Hierarchy**:

  java.lang.Object → android.view.ViewGroup.LayoutParams →
  android.view.ViewGroup.MarginLayoutParams →
  android.widget.RelativeLayout.LayoutParams

- **Include**

  ```
  0   android.widget.RelativeLayout.LayoutParams
  ```

- **Constructors**:

  ```
  0   LayoutParams(Context c, AttributeSet attrs)
  1   LayoutParams(ViewGroup.LayoutParams source)
  2   LayoutParams(ViewGroup.MarginLayoutParams source)
  3   LayoutParams(RelativeLayout.LayoutParams source)
  4   LayoutParams(int w, int h)
  ```

- **Public methods**:
  - **void addRule(int verb, int subject)**: Adds a layout rule to be interpreted by the `RelativeLayout`, relative to another view.
  - **void addRule(int verb)**: Adds a layout rule to be interpreted by the `RelativeLayout`.
  - **String debug(String output)**: Returns a string representation of this set of layout parameters, typically used for debugging.
  - **int getRule(int verb)**: Returns the layout rule associated with a specific verb.
  - **int[] getRules()**: Retrieves a complete list of all supported rules, where each index represents a rule verb and each value represents the associated parameter (or `false` if not set).
  - **void removeRule(int verb)**: Removes a layout rule from interpretation by the `RelativeLayout`.
  - **void resolveLayoutDirection(int layoutDirection)**: Called by `View.requestLayout()` to resolve layout parameters that depend on layout direction (e.g., start/end alignment).

- **Fields**:
  - **public boolean alignWithParent**: When true, uses the parent as the anchor if the anchor doesn't exist or if the anchor's visibility is GONE.

1.4.8 LinearLayout

- **Hierarchy**:

  java.lang.Object → android.view.View → android.view.ViewGroup →
  android.widget.LinearLayout

- **Include**

  ```
  0   android.widget.LinearLayout
  ```

- **Constructors**:

  ```
  0   LinearLayout(Context context)
  1   LinearLayout(Context context, AttributeSet attrs)
  2   LinearLayout(Context context, AttributeSet attrs, int
  ↪   defStyleAttr)
  3   LinearLayout(Context context, AttributeSet attrs, int
  ↪   defStyleAttr, int defStyleRes)
  ```

- **Public Methods**:

  - **LinearLayout.LayoutParams generateLayoutParams(AttributeSet attrs)**: Returns a new set of layout parameters based on the supplied attributes set.
  - **CharSequence getAccessibilityClassName()**: Return the class name of this object to be used for accessibility purposes.
  - **int getBaseline()**: Return the offset of the widget's text baseline from the widget's top boundary.
  - **int getBaselineAlignedChildIndex()**: Returns the index used for baseline alignment.
  - **Drawable getDividerDrawable()**: Returns the drawable used as a divider between child views.
  - **int getDividerPadding()**: Get the padding size used to inset dividers in pixels.
  - **int getGravity()**: Returns the current gravity.
  - **int getOrientation()**: Returns the current orientation.
  - **int getShowDividers()**: Returns how dividers are displayed between items.
  - **float getWeightSum()**: Returns the desired weights sum.
  - **boolean isBaselineAligned()**: Indicates whether widgets contained within this layout are aligned on their baseline or not.
  - **boolean isMeasureWithLargestChildEnabled()**: When true, all children with a weight will be considered having the minimum size of the largest child.
  - **void onRtlPropertiesChanged(int layoutDirection)**: Called when any RTL property (layout direction or text direction or text alignment) has been changed.
  - **void setBaselineAligned(boolean baselineAligned)**: Defines whether widgets contained in this layout are baseline-aligned or not.
  - **void setBaselineAlignedChildIndex(int i)**: Sets which child is used for baseline alignment.

154

- **void setDividerDrawable(Drawable divider)**: Set a drawable to be used as a divider between items.
- **void setDividerPadding(int padding)**: Set padding displayed on both ends of dividers.
- **void setGravity(int gravity)**: Describes how the child views are positioned.
- **void setHorizontalGravity(int horizontalGravity)**: Sets the horizontal gravity of the layout.
- **void setMeasureWithLargestChildEnabled(boolean enabled)**: When set to true, all children with a weight will be considered having the minimum size of the largest child.
- **void setOrientation(int orientation)**: Should the layout be a column or a row.
- **void setShowDividers(int showDividers)**: Set how dividers should be shown between items in this layout.
- **void setVerticalGravity(int verticalGravity)**: Sets the vertical gravity of the layout.
- **void setWeightSum(float weightSum)**: Defines the desired weights sum.
- **boolean shouldDelayChildPressedState()**: Return true if the pressed state should be delayed for children or descendants of this ViewGroup.

- **Protected Methods**:

  - **boolean checkLayoutParams(ViewGroup.LayoutParams p)**: Determines whether the supplied layout parameters are valid for this layout.
  - **LinearLayout.LayoutParams generateDefaultLayoutParams()**: Returns a set of layout parameters with a width of `ViewGroup.LayoutParams.MATCH_PARENT` and a height of `ViewGroup.LayoutParams.WRAP_CONTENT` when the layout's orientation is vertical.
  - **LinearLayout.LayoutParams generateLayoutParams(ViewGroup.LayoutParams lp)**: Returns a safe set of layout parameters based on the supplied layout parameters.
  - **void onDraw(Canvas canvas)**: Implement this method to perform custom drawing operations on the layout.
  - **void onLayout(boolean changed, int l, int t, int r, int b)**: Called from layout when this view should assign a size and position to each of its children.
  - **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: Measures the view and its content to determine the measured width and height.

- **Constants**:

  - **int HORIZONTAL**: Constant indicating a horizontal orientation for the layout.
  - **int SHOW_DIVIDER_BEGINNING**: Show a divider at the beginning of the group.
  - **int SHOW_DIVIDER_END**: Show a divider at the end of the group.
  - **int SHOW_DIVIDER_MIDDLE**: Show dividers between each item in the group.
  - **int SHOW_DIVIDER_NONE**: Do not show any dividers.
  - **int VERTICAL**: Constant indicating a vertical orientation for the layout.

1.4.9 LinearLayout.LayoutParams

- **Hierarchy**:

  java.lang.Object → android.view.ViewGroup.LayoutParams →
  android.view.ViewGroup.MarginLayoutParams →
  android.widget.LinearLayout.LayoutParams

- **Include**

  ```
  0   android.widget.LinearLayout.LayoutParams
  ```

- **Constructors**:

  ```
  0   LayoutParams(Context c, AttributeSet attrs)
  1   LayoutParams(ViewGroup.LayoutParams p)
  2   LayoutParams(ViewGroup.MarginLayoutParams source)
  3   LayoutParams(LinearLayout.LayoutParams source)
  4   LayoutParams(int width, int height)
  5   LayoutParams(int width, int height, float weight)
  ```

- **Public methods**:

  - **String debug(String output)**:

- **Fields**:

  - **public int gravity**: Gravity for the view associated with these LayoutParams.

  - **public float weight**: Indicates how much of the extra space in the LinearLayout will be allocated to the view associated with these LayoutParams.

1.4.10 GridLayout

- **Hierarchy**:

  java.lang.Object → android.view.View → android.view.ViewGroup →
  android.widget.GridLayout

- **Include**:

```
0   android.widget.GridLayout
```

- **Constructors**

```
0   GridLayout(Context context)
1   GridLayout(Context context, AttributeSet attrs)
2   GridLayout(Context context, AttributeSet attrs, int
    ↪   defStyleAttr)
3   GridLayout(Context context, AttributeSet attrs, int
    ↪   defStyleAttr, int defStyleRes)
```

- **Public methods**

  - **GridLayout.LayoutParams generateLayoutParams(AttributeSet attrs)**:
    Returns a new set of layout parameters based on the supplied attribute set.
  - **CharSequence getAccessibilityClassName()**: Returns the class name of this
    object to be used for accessibility purposes.
  - **int getAlignmentMode()**: Returns the current alignment mode used for posi-
    tioning children within their grid cells.
  - **int getColumnCount()**: Returns the current number of columns.
  - **int getOrientation()**: Returns the current orientation (horizontal or vertical).
  - **int getRowCount()**: Returns the current number of rows.
  - **boolean getUseDefaultMargins()**: Returns whether this `GridLayout` will
    allocate default margins when none are defined in layout parameters.
  - **boolean isColumnOrderPreserved()**: Returns whether column boundaries
    are ordered by their grid indices.
  - **boolean isRowOrderPreserved()**: Returns whether row boundaries are or-
    dered by their grid indices.
  - **void onViewAdded(View child)**: Called when a new child view is added to
    this `ViewGroup`.
  - **void onViewRemoved(View child)**: Called when a child view is removed
    from this `ViewGroup`.
  - **void requestLayout()**: Call this when something has changed that invalidates
    the layout of this view.
  - **void setAlignmentMode(int alignmentMode)**: Sets the alignment mode
    used for alignments between children of this container.
  - **void setColumnCount(int columnCount)**: Sets the total number of columns.
    Used to generate default column indices when none are specified.

- **void setColumnOrderPreserved(boolean columnOrderPreserved)**: When true, forces `GridLayout` to place column boundaries so their grid indices appear in ascending order.

- **void setOrientation(int orientation)**: Sets the layout's orientation. This controls:
  * The direction in which default row/column indices are generated when not specified.
  * Whether the grid is treated as row-major or column-major.

- **void setRowCount(int rowCount)**: Sets the total number of rows. Used to generate default row indices when none are specified.

- **void setRowOrderPreserved(boolean rowOrderPreserved)**: When true, forces `GridLayout` to place row boundaries in ascending order of their grid indices.

- **void setUseDefaultMargins(boolean useDefaultMargins)**: When true, `GridLayout` allocates default margins based on child visual characteristics.

- **static GridLayout.Spec spec(int start, float weight)**: Equivalent to `spec(start, 1, weight)`.

- **static GridLayout.Spec spec(int start)**: Returns a `Spec` where:
  * `span = [start, start + 1]`
  * To leave the start index undefined, use `UNDEFINED`.

- **static GridLayout.Spec spec(int start, int size, GridLayout.Alignment alignment, float weight)**: Returns a `Spec` where:
  * `span = [start, start + size]`
  * `alignment = alignment`
  * `weight = weight`
  * To leave the start index undefined, use `UNDEFINED`.

- **static GridLayout.Spec spec(int start, GridLayout.Alignment alignment, float weight)**: Equivalent to `spec(start, 1, alignment, weight)`.

- **static GridLayout.Spec spec(int start, int size, GridLayout.Alignment alignment)**: Equivalent to `spec(start, size, alignment, 0f)`.

- **static GridLayout.Spec spec(int start, GridLayout.Alignment alignment)**: Returns a `Spec` where:
  * `span = [start, start + 1]`
  * `alignment = alignment`
  * To leave the start index undefined, use `UNDEFINED`.

- **static GridLayout.Spec spec(int start, int size, float weight)**: Equivalent to `spec(start, size, default_alignment, weight)`, where `default_alignment` is defined in `GridLayout.LayoutParams`.

- **static GridLayout.Spec spec(int start, int size)**: Returns a `Spec` where:
  * `span = [start, start + size]`
  * To leave the start index undefined, use `UNDEFINED`.

- **Protected methods**

  - **boolean checkLayoutParams(ViewGroup.LayoutParams p)**: Determines whether the supplied layout parameters are valid for this layout.

  - **GridLayout.LayoutParams generateDefaultLayoutParams()**: Returns a set of default layout parameters used by this `GridLayout`.

– **GridLayout.LayoutParams generateLayoutParams(ViewGroup.Layout-Params lp)**: Returns a safe set of layout parameters based on the supplied layout parameters.

– **void onLayout(boolean changed, int left, int top, int right, int bottom)**: Called during layout when this view should assign a size and position to each of its children.

– **void onMeasure(int widthSpec, int heightSpec)**: Measures the view and its content to determine the measured width and height.

- **Fields**

  – **public static final GridLayout.Alignment BASELINE**: Indicates that a view should be aligned with the baselines of the other views in its cell group.

  – **public static final GridLayout.Alignment BOTTOM**: Indicates that a view should be aligned with the bottom edges of the other views in its cell group.

  – **public static final GridLayout.Alignment CENTER**: Indicates that a view should be centered with the other views in its cell group.

  – **public static final GridLayout.Alignment END**: Indicates that a view should be aligned with the end edges of the other views in its cell group.

  – **public static final GridLayout.Alignment FILL**: Indicates that a view should expand to fill the boundaries of its cell group.

  – **public static final GridLayout.Alignment LEFT**: Indicates that a view should be aligned with the left edges of the other views in its cell group.

  – **public static final GridLayout.Alignment RIGHT**: Indicates that a view should be aligned with the right edges of the other views in its cell group.

  – **public static final GridLayout.Alignment START**: Indicates that a view should be aligned with the start edges of the other views in its cell group.

  – **public static final GridLayout.Alignment TOP**: Indicates that a view should be aligned with the top edges of the other views in its cell group.

- **Constants**

  – **int ALIGN_BOUNDS**: Constant representing an `alignmentMode`. Child bounds are aligned within their grid cells.

  – **int ALIGN_MARGINS**: Constant representing an `alignmentMode`. Child margins are aligned within their grid cells.

  – **int HORIZONTAL**: Constant representing a horizontal orientation for the layout.

  – **int UNDEFINED**: Constant used to indicate that a value is undefined.

  – **int VERTICAL**: Constant representing a vertical orientation for the layout.

1.4.11   GridLayout.LayoutParams

- **Hierarchy**:

  java.lang.Object → android.view.ViewGroup.LayoutParams →
  android.view.ViewGroup.MarginLayoutParams →
  android.widget.GridLayout.LayoutParams

- **Include**:

  ```
  0   android.widget.GridLayout.LayoutParams
  ```

- **Constructors**:

  ```
  0   LayoutParams()
  1   LayoutParams(Context context, AttributeSet attrs)
  2   LayoutParams(ViewGroup.LayoutParams params)
  3   LayoutParams(ViewGroup.MarginLayoutParams params)
  4   LayoutParams(GridLayout.LayoutParams source)
  5   LayoutParams(GridLayout.Spec rowSpec, GridLayout.Spec
      ↪   columnSpec)
  ```

  Note: Values not defined in the attribute set take the default values defined in LayoutParams.

- **Public methods**

  - **boolean equals(Object o)**: Indicates whether some other object is "equal to" this one.
  - **int hashCode()**: Returns a hash code value for the object.
  - **void setGravity(int gravity)**: Describes how the child views are positioned.

- **Protected methods**

  - **void setBaseAttributes(TypedArray attributes, int widthAttr, int heightAttr)**: Extracts the layout parameters from the supplied attributes.

- **Fields**

  - **public GridLayout.Spec columnSpec**: The spec that defines the horizontal characteristics of the cell group described by these layout parameters.
  - **public GridLayout.Spec rowSpec**: The spec that defines the vertical characteristics of the cell group described by these layout parameters.

1.4.12 TableLayout

- **Hierarchy**:

  java.lang.Object → android.view.View → android.view.ViewGroup →
  android.widget.LinearLayout → android.widget.TableLayout

- **Include**

  > o   android.widget.TableLayout

- **Constructors**

  > o   TableLayout(Context context)
  > 1   TableLayout(Context context, AttributeSet attrs)

- **Public methods**

  - **void addView(View child, int index)**: Adds a child view at the specified index.
  - **void addView(View child, ViewGroup.LayoutParams params)**: Adds a child view with the specified layout parameters.
  - **void addView(View child)**: Adds a child view to the layout.
  - **void addView(View child, int index, ViewGroup.LayoutParams params)**: Adds a child view at the specified index with the given layout parameters.
  - **TableLayout.LayoutParams generateLayoutParams(AttributeSet attrs)**: Returns a new set of layout parameters based on the supplied attribute set.
  - **CharSequence getAccessibilityClassName()**: Returns the class name of this object to be used for accessibility purposes.
  - **boolean isColumnCollapsed(int columnIndex)**: Returns the collapsed state of the specified column.
  - **boolean isColumnShrinkable(int columnIndex)**: Returns whether the specified column is shrinkable.
  - **boolean isColumnStretchable(int columnIndex)**: Returns whether the specified column is stretchable.
  - **boolean isShrinkAllColumns()**: Indicates whether all columns in the table are shrinkable.
  - **boolean isStretchAllColumns()**: Indicates whether all columns in the table are stretchable.
  - **void requestLayout()**: Call this when something has changed that invalidates the layout of this view.
  - **void setColumnCollapsed(int columnIndex, boolean isCollapsed)**: Collapses or restores the specified column.
  - **void setColumnShrinkable(int columnIndex, boolean isShrinkable)**: Sets whether the specified column can shrink if necessary.
  - **void setColumnStretchable(int columnIndex, boolean isStretchable)**: Sets whether the specified column can stretch to fill available space.

– **void setOnHierarchyChangeListener(ViewGroup.OnHierarchyChange-
Listener listener)**: Registers a listener to be notified when a child view is added
to or removed from this layout.

– **void setShrinkAllColumns(boolean shrinkAllColumns)**: Convenience method
to mark all columns as shrinkable.

– **void setStretchAllColumns(boolean stretchAllColumns)**: Convenience
method to mark all columns as stretchable.

- **Protected methods**

  – **boolean checkLayoutParams(ViewGroup.LayoutParams p)**: Determines
  whether the supplied layout parameters are valid for this layout.

  – **LinearLayout.LayoutParams generateDefaultLayoutParams()**: Returns
  a set of default layout parameters with a width of `ViewGroup.LayoutParams.MATCH_PARENT`
  and a height of `ViewGroup.LayoutParams.WRAP_CONTENT`.

  – **LinearLayout.LayoutParams generateLayoutParams(ViewGroup.Lay-
  outParams p)**: Returns a safe set of layout parameters based on the supplied
  layout parameters.

  – **void onLayout(boolean changed, int l, int t, int r, int b)**: Called during
  layout when this view should assign a size and position to each of its children.

  – **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: Mea-
  sures the view and its content to determine the measured width and height.

1.4.13 TableLayout.LayoutParams

- **Hierarchy**

  java.lang.Object → android.view.ViewGroup.LayoutParams →
  android.view.ViewGroup.MarginLayoutParams →
  android.widget.LinearLayout.LayoutParams →
  android.widget.TableLayout.LayoutParams

- **Include**

  ```
  0   android.widget.TableLayout.LayoutParams
  ```

- **Constructors**

  ```
  0   LayoutParams()
  1   LayoutParams(Context c, AttributeSet attrs)
  2   LayoutParams(ViewGroup.LayoutParams p)
  3   LayoutParams(ViewGroup.MarginLayoutParams source)
  4   LayoutParams(int w, int h)
  5   LayoutParams(int w, int h, float initWeight)
  ```

- **Protected methods**

  - **void setBaseAttributes(TypedArray a, int widthAttr, int heightAttr)**:
    Fixes the row's width to ViewGroup.LayoutParams.MATCH_PARENT; the row's
    height is fixed to ViewGroup.LayoutParams.WRAP_CONTENT if no layout
    height is specified.

1.4.14   TableRow

- **Hierarchy**

  java.lang.Object → android.view.View → android.view.ViewGroup →
  android.widget.LinearLayout → android.widget.TableRow

- **Include**

  0   android.widget.TableRow

- **Constructors**

  0   TableRow(Context context)
  1   TableRow(Context context, AttributeSet attrs)

- **Public methods**

  – **TableRow.LayoutParams generateLayoutParams(AttributeSet attrs)**:
  Returns a new set of layout parameters based on the supplied attributes set.

  – **CharSequence getAccessibilityClassName()**: Return the class name of this
  object to be used for accessibility purposes.

  – **View getVirtualChildAt(int i)**:

  – **int getVirtualChildCount()**:

  – **void setOnHierarchyChangeListener(ViewGroup.OnHierarchyChange-
  Listener listener)**: Register a callback to be invoked when a child is added to
  or removed from this view.

- **Protected methods**

  – **boolean checkLayoutParams(ViewGroup.LayoutParams p)**:

  – **LinearLayout.LayoutParams generateDefaultLayoutParams()**: Returns
  a set of layout parameters with a width of ViewGroup.LayoutParams.MATCH_PAR-
  ENT, a height of ViewGroup.LayoutParams.WRAP_CONTENT and no span-
  ning.

  – **LinearLayout.LayoutParams generateLayoutParams(ViewGroup.Lay-
  outParams p)**: Returns a safe set of layout parameters based on the supplied
  layout params.

  – **void onLayout(boolean changed, int l, int t, int r, int b)**: Called from
  layout when this view should assign a size and position to each of its children.

  – **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: Mea-
  sure the view and its content to determine the measured width and the measured
  height.

1.4.15 FrameLayout

- **Hierarchy**

  java.lang.Object → android.view.View → android.view.ViewGroup →
  android.widget.FrameLayout

- **Include**

  ```
  0   android.widget.FrameLayout
  ```

- **Constructors**

  ```
  0   FrameLayout(Context context)
  1   FrameLayout(Context context, AttributeSet attrs)
  2   FrameLayout(Context context, AttributeSet attrs, int
  ↪    defStyleAttr, int defStyleRes)
  3   FrameLayout(Context context, AttributeSet attrs, int
  ↪    defStyleAttr)
  ```

- **Public methods**

  - **FrameLayout.LayoutParams generateLayoutParams(AttributeSet attrs)**: Returns a new set of layout parameters based on the supplied attribute set.
  - **CharSequence getAccessibilityClassName()**: Returns the class name of this object to be used for accessibility purposes.
  - **boolean getConsiderGoneChildrenWhenMeasuring()**: *(Deprecated in API level 15)* — Previously determined whether to include `GONE` children in measurement. Replaced by `getMeasureAllChildren()` for naming consistency.
  - **boolean getMeasureAllChildren()**: Determines whether all children, or only those in the `VISIBLE` or `INVISIBLE` state, are considered during measurement.
  - **void setForegroundGravity(int foregroundGravity)**: Describes how the foreground drawable is positioned within the layout.
  - **void setMeasureAllChildren(boolean measureAll)**: Sets whether all children, or only visible ones, should be considered when measuring the layout.
  - **boolean shouldDelayChildPressedState()**: Returns true if the pressed state should be delayed for children or descendants of this `ViewGroup`.

- **Protected methods**

  - **boolean checkLayoutParams(ViewGroup.LayoutParams p)**: Determines whether the supplied layout parameters are valid for this layout.
  - **FrameLayout.LayoutParams generateDefaultLayoutParams()**: Returns a set of default layout parameters with both width and height set to `ViewGroup.LayoutParams.MATCH_PARENT`
  - **ViewGroup.LayoutParams generateLayoutParams(ViewGroup.LayoutParams lp)**: Returns a safe set of layout parameters based on the supplied layout parameters.
  - **void onLayout(boolean changed, int left, int top, int right, int bottom)**: Called during layout when this view should assign a size and position to each of its children.
  - **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: Measures the view and its content to determine the measured width and height.

1.4.16 FrameLayout.LayoutParams

- **Hierarchy**

  java.lang.Object → android.view.ViewGroup.LayoutParams →
  android.view.ViewGroup.MarginLayoutParams →
  android.widget.FrameLayout.LayoutParams

- **Include**

  ```
  0  android.widget.FrameLayout.LayoutParams
  ```

- **Constructors**

  ```
  0  LayoutParams(Context c, AttributeSet attrs)
  1  LayoutParams(ViewGroup.LayoutParams source)
  2  LayoutParams(ViewGroup.MarginLayoutParams source)
  3  LayoutParams(FrameLayout.LayoutParams source)
  4  LayoutParams(int width, int height)
  5  LayoutParams(int width, int height, int gravity)
  ```

- **Fields**

  – **public int gravity**: The gravity to apply with the View to which these layout
    parameters are associated.

- **Constants**

  – **int UNSPECIFIED_GRAVITY**: Value for gravity indicating that a gravity
    has not been explicitly specified.

1.4.17 ListView

- **Hierarchy**:

    java.lang.Object → android.view.View → android.view.ViewGroup →
    android.widget.AdapterView<android.widget.ListAdapter> →
    android.widget.AbsListView → android.widget.ListView

- **Include**

    ```
    0   android.widget.ListView
    ```

- **Constructors**

    ```
    0   ListView(Context context)
    1   ListView(Context context, AttributeSet attrs)
    2   ListView(Context context, AttributeSet attrs, int
    ↪     defStyleAttr)
    3   ListView(Context context, AttributeSet attrs, int
    ↪     defStyleAttr, int defStyleRes)
    ```

- **Public methods**

    - **void addFooterView(View v)**: Adds a fixed view to appear at the bottom of the list.
    - **void addFooterView(View v, Object data, boolean isSelectable)**: Adds a fixed view to appear at the bottom of the list with optional data and selectable state.
    - **void addHeaderView(View v, Object data, boolean isSelectable)**: Adds a fixed view to appear at the top of the list with optional data and selectable state.
    - **void addHeaderView(View v)**: Adds a fixed view to appear at the top of the list.
    - **boolean areFooterDividersEnabled()**: Returns whether footer dividers are currently enabled.
    - **boolean areHeaderDividersEnabled()**: Returns whether header dividers are currently enabled.
    - **boolean dispatchKeyEvent(KeyEvent event)**: Dispatches a key event to the next view on the focus path.
    - **CharSequence getAccessibilityClassName()**: Returns the accessibility class name. A `TYPE_VIEW_SCROLLED` event should be sent whenever a scroll happens, even if position and child count remain unchanged.
    - **ListAdapter getAdapter()**: Returns the adapter currently in use by this `ListView`.
    - **long[] getCheckItemIds()**: *(Deprecated in API 15)* — Use `AbsListView.getCheckedItemIds()` instead. Returns the IDs of the currently checked items.
    - **Drawable getDivider()**: Returns the drawable that is drawn between each list item.
    - **int getDividerHeight()**: Returns the height of the divider between list items.

167

- **int getFooterViewsCount()**: Returns the number of fixed footer views currently added.
- **int getHeaderViewsCount()**: Returns the number of fixed header views currently added.
- **boolean getItemsCanFocus()**: Returns whether list items can contain focusable elements.
- **int getMaxScrollAmount()**: Returns the maximum amount the list can scroll in response to arrow events.
- **Drawable getOverscrollFooter()**: Returns the drawable drawn below all list content when overscrolling.
- **Drawable getOverscrollHeader()**: Returns the drawable drawn above all list content when overscrolling.
- **boolean isOpaque()**: Indicates whether this view is opaque.
- **void onInitializeAccessibilityNodeInfoForItem(View view, int position, AccessibilityNodeInfo info)**: Initializes accessibility node info for a specific list item.
- **boolean onKeyDown(int keyCode, KeyEvent event)**: Default implementation handles key presses such as `KEYCODE_DPAD_CENTER` or `KEYCODE_ENTER` to trigger selection.
- **boolean onKeyMultiple(int keyCode, int repeatCount, KeyEvent event)**: Default implementation always returns false (does not handle multiple key events).
- **boolean onKeyUp(int keyCode, KeyEvent event)**: Default implementation handles key releases such as `KEYCODE_DPAD_CENTER`, `KEYCODE_ENTER`, or `KEYCODE_SPACE` to perform clicks.
- **boolean removeFooterView(View v)**: Removes a previously added footer view.
- **boolean removeHeaderView(View v)**: Removes a previously added header view.
- **boolean requestChildRectangleOnScreen(View child, Rect rect, boolean immediate)**: Called when a child requests that a specific rectangle within it be visible on the screen.
- **void setAdapter(ListAdapter adapter)**: Sets the adapter that provides the data and views for this `ListView`.
- **void setCacheColorHint(int color)**: When set to a nonzero value, indicates that the list is drawn on top of a solid, opaque background of that color.
- **void setDivider(Drawable divider)**: Sets the drawable that will be drawn between list items.
- **void setDividerHeight(int height)**: Sets the height of the divider drawn between list items.
- **void setFooterDividersEnabled(boolean footerDividersEnabled)**: Enables or disables drawing of dividers for footer views.
- **void setHeaderDividersEnabled(boolean headerDividersEnabled)**: Enables or disables drawing of dividers for header views.
- **void setItemsCanFocus(boolean itemsCanFocus)**: Indicates whether views created by the adapter can contain focusable items.
- **void setOverscrollFooter(Drawable footer)**: Sets the drawable to be drawn below all list content during overscroll.

- **void setOverscrollHeader(Drawable header)**: Sets the drawable to be drawn above all list content during overscroll.

- **void setRemoteViewsAdapter(Intent intent)**: Sets up this `ListView` to use a remote views adapter connected via a `RemoteViewsService`.

- **void setSelection(int position)**: Sets the currently selected item in the list.

- **void setSelectionAfterHeaderView()**: Sets the selection to the first list item following the header views.

- **void smoothScrollByOffset(int offset)**: Smoothly scrolls the list by the specified adapter position offset.

- **void smoothScrollToPosition(int position)**: Smoothly scrolls to the specified adapter position.

- **Protected methods**

  - **boolean canAnimate()**: Indicates whether the view group can animate its children after the first layout pass.

  - **void dispatchDraw(Canvas canvas)**: Called by the system's `draw()` method to render all child views within this layout.

  - **boolean drawChild(Canvas canvas, View child, long drawingTime)**: Draws a single child view of this `ViewGroup` onto the provided `Canvas`.

  - **void layoutChildren()**: Abstract method that subclasses must override to define how their child views are positioned and sized.

  - **void onDetachedFromWindow()**: Called when the view is detached from its window, typically used to clean up resources or listeners.

  - **void onFinishInflate()**: Called after a view and all its children have been inflated from XML to perform any final initialization.

  - **void onFocusChanged(boolean gainFocus, int direction, Rect previouslyFocusedRect)**: Invoked when the view's focus state changes, providing the direction and previously focused rectangle.

  - **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: Measures the view and its children to determine the overall measured width and height.

  - **void onSizeChanged(int w, int h, int oldw, int oldh)**: Called during layout when the view's size changes, allowing for recalculation of layout-dependent properties.

1.4.18 TextView

- **Hierarchy**

  java.lang.Object → android.view.View → android.widget.TextView

- **Include**

  ```
  0   android.widget.TextView
  ```

- **Constructors**

  ```
  0   TextView(Context context)
  1   TextView(Context context, AttributeSet attrs)
  2   TextView(Context context, AttributeSet attrs, int
      ↪   defStyleAttr)
  3   TextView(Context context, AttributeSet attrs, int
      ↪   defStyleAttr, int defStyleRes)
  ```

- **Public methods**

  - **void addExtraDataToAccessibilityNodeInfo(AccessibilityNodeInfo info, String extraDataKey, Bundle arguments)**: Adds extra data to an `AccessibilityNodeInfo` based on an explicit request for the additional data.

  - **void addTextChangedListener(TextWatcher watcher)**: Adds a `TextWatcher` whose methods are called whenever this `TextView`'s text changes.

  - **final void append(CharSequence text)**: Appends text to the display buffer, upgrading to `BufferType.EDITABLE` if needed.

  - **void append(CharSequence text, int start, int end)**: Appends a slice of text to the display buffer, upgrading to `BufferType.EDITABLE` if needed.

  - **void autofill(AutofillValue value)**: Automatically fills this view's content with the provided value.

  - **void beginBatchEdit()**: Begins a batch edit session.

  - **boolean bringPointIntoView(int offset)**: Moves the character offset into view if needed.

  - **boolean bringPointIntoView(int offset, boolean requestRectWithoutFocus)**: Moves the insertion position at the given offset into the visible area.

  - **void cancelLongPress()**: Cancels a pending long press.

  - **void clearComposingText()**: Uses `BaseInputConnection.removeComposingSpans()` to clear IME composing state.

  - **void computeScroll()**: Requests the child to update `mScrollX` and `mScrollY` if necessary.

  - **void debug(int depth)**: Outputs debug information with the given depth.

  - **boolean didTouchFocusSelect()**: During a touch gesture, returns true iff initial touch moved focus to this view and changed selection.

  - **void drawableHotspotChanged(float x, float y)**: Propagates view hotspot changes to drawables/children.

  - **void endBatchEdit()**: Ends a batch edit session.

- **boolean extractText(ExtractedTextRequest request, ExtractedText out-Text)**: Extracts a portion of editable content into `outText`.
- **void findViewsWithText(ArrayList<View> outViews, CharSequence searched, int flags)**: Finds views containing the given text.
- **CharSequence getAccessibilityClassName()**: Returns the accessibility class name.
- **final int getAutoLinkMask()**: Gets the autolink mask.
- **int getAutoSizeMaxTextSize()**: Returns max auto-size text size.
- **int getAutoSizeMinTextSize()**: Returns min auto-size text size.
- **int getAutoSizeStepGranularity()**: Returns auto-size step granularity.
- **int[] getAutoSizeTextAvailableSizes()**: Returns available auto-size text sizes.
- **int getAutoSizeTextType()**: Returns the auto-size text type.
- **String[] getAutofillHints()**: Gets autofill hints for `AutofillService`.
- **int getAutofillType()**: Describes the autofill type for this view.
- **AutofillValue getAutofillValue()**: Returns current text as an `AutofillValue`.
- **int getBaseline()**: Returns the text baseline offset from the top.
- **int getBreakStrategy()**: Gets the paragraph line-break strategy.
- **int getCompoundDrawablePadding()**: Returns padding between compound drawables and text.
- **BlendMode getCompoundDrawableTintBlendMode()**: Returns the tint blend mode for compound drawables.
- **ColorStateList getCompoundDrawableTintList()**: Returns the tint list for compound drawables.
- **PorterDuff.Mode getCompoundDrawableTintMode()**: Returns the Porter-Duff tint mode for compound drawables.
- **Drawable[] getCompoundDrawables()**: Returns left, top, right, bottom drawables.
- **Drawable[] getCompoundDrawablesRelative()**: Returns start, top, end, bottom drawables.
- **int getCompoundPaddingBottom()**: Returns bottom padding plus drawable space.
- **int getCompoundPaddingEnd()**: Returns end padding plus drawable space.
- **int getCompoundPaddingLeft()**: Returns left padding plus drawable space.
- **int getCompoundPaddingRight()**: Returns right padding plus drawable space.
- **int getCompoundPaddingStart()**: Returns start padding plus drawable space.
- **int getCompoundPaddingTop()**: Returns top padding plus drawable space.
- **final int getCurrentHintTextColor()**: Returns current hint text color.
- **final int getCurrentTextColor()**: Returns current text color.
- **ActionMode.Callback getCustomInsertionActionModeCallback()**: Gets the custom insertion `ActionMode` callback.
- **ActionMode.Callback getCustomSelectionActionModeCallback()**: Gets the custom selection `ActionMode` callback.
- **Editable getEditableText()**: Returns the text as an `Editable`.
- **TextUtils.TruncateAt getEllipsize()**: Returns where long text is ellipsized.

- **CharSequence getError()**: Returns the current error message or `null`.
- **int getExtendedPaddingBottom()**: Returns extended bottom padding.
- **int getExtendedPaddingTop()**: Returns extended top padding.
- **InputFilter[] getFilters()**: Returns the list of input filters.
- **int getFirstBaselineToTopHeight()**: Distance from first baseline to top.
- **void getFocusedRect(Rect r)**: Fills `r` with the focus search rectangle.
- **int getFocusedSearchResultHighlightColor()**: Gets focused search result highlight color.
- **int getFocusedSearchResultIndex()**: Gets focused search result index.
- **String getFontFeatureSettings()**: Returns font feature settings.
- **String getFontVariationSettings()**: Returns font variation settings.
- **boolean getFreezesText()**: Whether full text is saved in icicles.
- **int getGravity()**: Returns horizontal/vertical text gravity.
- **int getHighlightColor()**: Returns selection highlight color.
- **Highlights getHighlights()**: Returns highlights.
- **CharSequence getHint()**: Returns the hint text.
- **final ColorStateList getHintTextColors()**: Returns hint text colors.
- **int getHyphenationFrequency()**: Gets automatic hyphenation frequency.
- **int getImeActionId()**: Gets IME action ID set via `setImeActionLabel`.
- **CharSequence getImeActionLabel()**: Gets IME action label.
- **LocaleList getImeHintLocales()**: Gets IME hint locales.
- **int getImeOptions()**: Gets IME options.
- **boolean getIncludeFontPadding()**: Whether extra ascent/descent padding is included.
- **Bundle getInputExtras(boolean create)**: Retrieves/creates input extras bundle.
- **int getInputType()**: Gets the editable content type.
- **int getJustificationMode()**: Gets text justification mode.
- **final KeyListener getKeyListener()**: Gets the current `KeyListener`.
- **int getLastBaselineToBottomHeight()**: Distance from last baseline to bottom.
- **final Layout getLayout()**: Gets the current text `Layout`.
- **float getLetterSpacing()**: Gets letter spacing (em).
- **int getLineBounds(int line, Rect bounds)**: Returns baseline for a line; optionally fills `bounds`.
- **int getLineBreakStyle()**: Gets line-break style.
- **int getLineBreakWordStyle()**: Gets line-break word style.
- **int getLineCount()**: Returns line count or 0 if layout not built.
- **int getLineHeight()**: Returns line height in pixels.
- **float getLineSpacingExtra()**: Returns extra line spacing.
- **float getLineSpacingMultiplier()**: Returns line spacing multiplier.
- **final ColorStateList getLinkTextColors()**: Returns link text colors.

172

- **final boolean getLinksClickable()**: Whether `LinkMovementMethod` is auto-set for links.
- **int getMarqueeRepeatLimit()**: Returns marquee repeat count.
- **int getMaxEms()**: Returns max width in ems, or -1.
- **int getMaxHeight()**: Returns max height in px, or -1.
- **int getMaxLines()**: Returns max lines, or -1.
- **int getMaxWidth()**: Returns max width in px, or -1.
- **int getMinEms()**: Returns min width in ems, or -1.
- **int getMinHeight()**: Returns min height in px, or -1.
- **int getMinLines()**: Returns min lines, or -1.
- **int getMinWidth()**: Returns min width in px, or -1.
- **Paint.FontMetrics getMinimumFontMetrics()**: Returns minimum font metrics used for line spacing.
- **final MovementMethod getMovementMethod()**: Gets the `MovementMethod`.
- **int getOffsetForPosition(float x, float y)**: Returns closest character offset to the given position.
- **TextPaint getPaint()**: Gets the `TextPaint`.
- **int getPaintFlags()**: Gets the current paint flags.
- **String getPrivateImeOptions()**: Gets private IME options.
- **int getSearchResultHighlightColor()**: Gets search result highlight color.
- **int[] getSearchResultHighlights()**: Gets current search result ranges.
- **int getSelectionEnd()**: Convenience for `Selection.getSelectionEnd`.
- **int getSelectionStart()**: Convenience for `Selection.getSelectionStart`.
- **int getShadowColor()**: Gets shadow color.
- **float getShadowDx()**: Gets shadow X offset.
- **float getShadowDy()**: Gets shadow Y offset.
- **float getShadowRadius()**: Gets shadow blur radius.
- **boolean getShiftDrawingOffsetForStartOverhang()**: True if shifting x-offset for start overhang.
- **final boolean getShowSoftInputOnFocus()**: Whether soft input shows on focus.
- **CharSequence getText()**: Returns the displayed text.
- **TextClassifier getTextClassifier()**: Returns the `TextClassifier`.
- **final ColorStateList getTextColors()**: Returns text colors by state.
- **Drawable getTextCursorDrawable()**: Returns the cursor drawable.
- **TextDirectionHeuristic getTextDirectionHeuristic()**: Returns resolved text direction heuristic.
- **Locale getTextLocale()**: Returns primary text `Locale`.
- **LocaleList getTextLocales()**: Returns text `LocaleList`.
- **PrecomputedText.Params getTextMetricsParams()**: Returns precomputed-text layout params.
- **float getTextScaleX()**: Returns horizontal text scale factor.
- **Drawable getTextSelectHandle()**: Returns selection handle drawable.

– **Drawable getTextSelectHandleLeft()**: Returns left selection handle drawable.

– **Drawable getTextSelectHandleRight()**: Returns right selection handle drawable.

– **float getTextSize()**: Returns text size (sp units).

– **int getTextSizeUnit()**: Returns the defined text size unit.

– **int getTotalPaddingBottom()**: Returns total bottom padding.

– **int getTotalPaddingEnd()**: Returns total end padding.

– **int getTotalPaddingLeft()**: Returns total left padding.

– **int getTotalPaddingRight()**: Returns total right padding.

– **int getTotalPaddingStart()**: Returns total start padding.

– **int getTotalPaddingTop()**: Returns total top padding.

– **final TransformationMethod getTransformationMethod()**: Returns the current text transformation method.

– **Typeface getTypeface()**: Returns the current `Typeface`.

– **URLSpan[] getUrls()**: Returns `URLSpan`s attached to the text.

– **boolean getUseBoundsForWidth()**: True if bounding box width is used for line breaking/drawing.

– **boolean hasOverlappingRendering()**: Whether the view has overlapping rendering.

– **boolean hasSelection()**: True iff there is a nonzero-length selection.

– **void invalidateDrawable(Drawable drawable)**: Invalidates the given drawable.

– **boolean isAllCaps()**: Whether ALL CAPS transformation is applied.

– **boolean isAutoHandwritingEnabled()**: Whether automatic handwriting initiation is allowed.

– **boolean isCursorVisible()**: Whether the cursor is visible.

– **boolean isElegantTextHeight()**: Gets elegant height metrics flag.

– **boolean isFallbackLineSpacing()**: Whether fallback font ascent/descent is respected.

– **final boolean isHorizontallyScrollable()**: Whether text may be wider than the view.

– **boolean isInputMethodTarget()**: Whether this view is the current input method target.

– **boolean isLocalePreferredLineHeightForMinimumUsed()**: True if locale-preferred line height is used for minimum line height.

– **boolean isSingleLine()**: Whether text is constrained to a single scrolling line.

– **boolean isSuggestionsEnabled()**: Whether suggestions are enabled.

– **boolean isTextSelectable()**: Returns `textIsSelectable` state.

– **void jumpDrawablesToCurrentState()**: Calls `jumpToCurrentState()` on associated drawables.

– **int length()**: Returns the text length in characters.

– **boolean moveCursorToVisibleOffset()**: Moves cursor to a visible offset if needed.

- **void onBeginBatchEdit()**: Called when a batch edit begins.
- **boolean onCheckIsTextEditor()**: Whether this view is a text editor.
- **void onCommitCompletion(CompletionInfo text)**: Called on IME completion.
- **void onCommitCorrection(CorrectionInfo info)**: Called on IME auto-correction.
- **InputConnection onCreateInputConnection(EditorInfo outAttrs)**: Creates an `InputConnection`.
- **void onCreateViewTranslationRequest(int[] supportedFormats, Consumer<ViewTranslationRequest> requestsCollector)**: Collects view translation requests.
- **boolean onDragEvent(DragEvent event)**: Handles drag events.
- **void onEditorAction(int actionCode)**: Called for `performEditorAction()`.
- **void onEndBatchEdit()**: Called when a batch edit ends.
- **boolean onGenericMotionEvent(MotionEvent event)**: Handles generic motion events.
- **boolean onKeyDown(int keyCode, KeyEvent event)**: Default handling for key down (e.g., DPAD_CENTER/ENTER).
- **boolean onKeyMultiple(int keyCode, int repeatCount, KeyEvent event)**: Default returns false for multiple key events.
- **boolean onKeyPreIme(int keyCode, KeyEvent event)**: Handles a key event before IME processes it.
- **boolean onKeyShortcut(int keyCode, KeyEvent event)**: Called when key shortcut isn't handled.
- **boolean onKeyUp(int keyCode, KeyEvent event)**: Default handling for key up (e.g., DPAD_CENTER/ENTER/SPACE).
- **boolean onPreDraw()**: Called when the view tree is about to be drawn.
- **boolean onPrivateIMECommand(String action, Bundle data)**: Handles private IME commands.
- **ContentInfo onReceiveContent(ContentInfo payload)**: Default content reception.
- **PointerIcon onResolvePointerIcon(MotionEvent event, int pointerIndex)**: Resolves pointer icon for the event.
- **void onRestoreInstanceState(Parcelable state)**: Restores internal state from `Parcelable`.
- **void onRtlPropertiesChanged(int layoutDirection)**: Called when RTL-related properties change.
- **Parcelable onSaveInstanceState()**: Saves internal state to a `Parcelable`.
- **void onScreenStateChanged(int screenState)**: Called when the screen state changes.
- **boolean onTextContextMenuItem(int id)**: Handles a text context menu selection.
- **boolean onTouchEvent(MotionEvent event)**: Handles pointer events.
- **boolean onTrackballEvent(MotionEvent event)**: Handles trackball events.
- **void onVisibilityAggregated(boolean isVisible)**: Called when user-visibility may change.

- **void onWindowFocusChanged(boolean hasWindowFocus)**: Called when the containing window's focus changes.

- **boolean performLongClick()**: Invokes `OnLongClickListener`, if defined.

- **void removeTextChangedListener(TextWatcher watcher)**: Removes a `TextWatcher`.

- **void sendAccessibilityEventUnchecked(AccessibilityEvent event)**: Sends an accessibility event without checking if accessibility is enabled.

- **void setAllCaps(boolean allCaps)**: Transforms input to ALL CAPS display.

- **final void setAutoLinkMask(int mask)**: Sets the autolink mask.

- **void setAutoSizeTextTypeUniformWithConfiguration(int autoSizeMinTextSize, int autoSizeMaxTextSize, int autoSizeStepGranularity, int unit)**: Configures uniform auto-size text.

- **void setAutoSizeTextTypeUniformWithPresetSizes(int[] presetSizes, int unit)**: Sets preset sizes for auto-size text.

- **void setAutoSizeTextTypeWithDefaults(int autoSizeTextType)**: Enables default auto-size configuration.

- **void setBreakStrategy(int breakStrategy)**: Sets paragraph break strategy.

- **void setCompoundDrawablePadding(int pad)**: Sets padding between drawables and text.

- **void setCompoundDrawableTintBlendMode(BlendMode blendMode)**: Sets blend mode for compound drawable tints.

- **void setCompoundDrawableTintList(ColorStateList tint)**: Applies tint to compound drawables.

- **void setCompoundDrawableTintMode(PorterDuff.Mode tintMode)**: Sets Porter-Duff blend mode for drawable tints.

- **void setCompoundDrawables(Drawable left, Drawable top, Drawable right, Drawable bottom)**: Sets left/top/right/bottom drawables.

- **void setCompoundDrawablesRelative(Drawable start, Drawable top, Drawable end, Drawable bottom)**: Sets start/top/end/bottom drawables.

- **void setCompoundDrawablesRelativeWithIntrinsicBounds(Drawable start, Drawable top, Drawable end, Drawable bottom)**: Sets start/top/end/bottom drawables with intrinsic bounds.

- **void setCompoundDrawablesRelativeWithIntrinsicBounds(int start, int top, int end, int bottom)**: Same as above with resource IDs.

- **void setCompoundDrawablesWithIntrinsicBounds(Drawable left, Drawable top, Drawable right, Drawable bottom)**: Sets left/top/right/bottom drawables with intrinsic bounds.

- **void setCompoundDrawablesWithIntrinsicBounds(int left, int top, int right, int bottom)**: Same as above with resource IDs.

- **void setCursorVisible(boolean visible)**: Shows/hides the cursor.

- **void setCustomInsertionActionModeCallback(ActionMode.Callback actionModeCallback)**: Sets custom insertion `ActionMode` callback.

- **void setCustomSelectionActionModeCallback(ActionMode.Callback actionModeCallback)**: Sets custom selection `ActionMode` callback.

- **final void setEditableFactory(Editable.Factory factory)**: Sets the `Editable.Factory`.

- **void setElegantTextHeight(boolean elegant)**: Sets elegant height metrics flag.

– **void setEllipsize(TextUtils.TruncateAt where)**: Enables ellipsizing of long words.

– **void setEms(int ems)**: Sets exact width in ems.

– **void setEnabled(boolean enabled)**: Enables/disables the view.

– **void setError(CharSequence error)**: Shows an error icon and message popup.

– **void setError(CharSequence error, Drawable icon)**: Shows a custom error icon and message popup.

– **void setExtractedText(ExtractedText text)**: Applies extracted text to the view.

– **void setFallbackLineSpacing(boolean enabled)**: Respects fallback font ascent/descent when enabled.

– **void setFilters(InputFilter[] filters)**: Sets input filters for editable content.

– **void setFirstBaselineToTopHeight(int firstBaselineToTopHeight)**: Adjusts top padding so first baseline is at the given distance from top.

– **void setFocusedSearchResultHighlightColor(int color)**: Sets focused search result highlight color.

– **void setFocusedSearchResultIndex(int index)**: Sets focused search result index.

– **void setFontFeatureSettings(String fontFeatureSettings)**: Sets font feature settings.

– **boolean setFontVariationSettings(String fontVariationSettings)**: Sets TrueType/OpenType variation settings.

– **void setFreezesText(boolean freezesText)**: Controls saving full text on instance state.

– **void setGravity(int gravity)**: Sets horizontal text alignment and vertical gravity.

– **void setHeight(int pixels)**: Sets exact height in pixels.

– **void setHighlightColor(int color)**: Sets selection highlight color.

– **void setHighlights(Highlights highlights)**: Sets highlights.

– **final void setHint(CharSequence hint)**: Sets hint text.

– **final void setHint(int resid)**: Sets hint text from a resource.

– **final void setHintTextColor(ColorStateList colors)**: Sets hint text colors.

– **final void setHintTextColor(int color)**: Sets hint text color for all states.

– **void setHorizontallyScrolling(boolean whether)**: Allows text to exceed view width.

– **void setHyphenationFrequency(int hyphenationFrequency)**: Sets hyphenation frequency.

– **void setImeActionLabel(CharSequence label, int actionId)**: Sets custom IME action label and ID.

– **void setImeHintLocales(LocaleList hintLocales)**: Sets IME hint locales.

– **void setImeOptions(int imeOptions)**: Sets editor type/IME options.

– **void setIncludeFontPadding(boolean includepad)**: Toggles extra ascent/descent font padding.

– **void setInputExtras(int xmlResId)**: Sets extra input data bundle from XML.

- **void setInputType(int type)**: Sets the input content type.
- **void setJustificationMode(int justificationMode)**: Sets text justification mode.
- **void setKeyListener(KeyListener input)**: Sets the `KeyListener`.
- **void setLastBaselineToBottomHeight(int lastBaselineToBottomHeight)**: Adjusts bottom padding so last baseline is at the given distance from bottom.
- **void setLetterSpacing(float letterSpacing)**: Sets letter spacing (em).
- **void setLineBreakStyle(int lineBreakStyle)**: Sets line-break style.
- **void setLineBreakWordStyle(int lineBreakWordStyle)**: Sets line-break word style.
- **void setLineHeight(int unit, float lineHeight)**: Sets explicit line height with unit.
- **void setLineHeight(int lineHeight)**: Sets explicit line height (px).
- **void setLineSpacing(float add, float mult)**: Sets line spacing extra and multiplier.
- **void setLines(int lines)**: Sets exact line count.
- **final void setLinkTextColor(ColorStateList colors)**: Sets link text colors.
- **final void setLinkTextColor(int color)**: Sets link text color.
- **final void setLinksClickable(boolean whether)**: Controls auto-setting of `LinkMovementMethod`.
- **void setLocalePreferredLineHeightForMinimumUsed(boolean flag)**: Uses locale-preferred line height for minimum line height.
- **void setMarqueeRepeatLimit(int marqueeLimit)**: Sets marquee repeat count.
- **void setMaxEms(int maxEms)**: Sets maximum width in ems.
- **void setMaxHeight(int maxPixels)**: Sets maximum height in px.
- **void setMaxLines(int maxLines)**: Sets maximum number of lines.
- **void setMaxWidth(int maxPixels)**: Sets maximum width in px.
- **void setMinEms(int minEms)**: Sets minimum width in ems.
- **void setMinHeight(int minPixels)**: Sets minimum height in px.
- **void setMinLines(int minLines)**: Sets minimum number of lines.
- **void setMinWidth(int minPixels)**: Sets minimum width in px.
- **void setMinimumFontMetrics(Paint.FontMetrics minimumFontMetrics)**: Sets minimum font metrics for line spacing.
- **final void setMovementMethod(MovementMethod movement)**: Sets the `MovementMethod`.
- **void setOnEditorActionListener(TextView.OnEditorActionListener l)**: Sets a listener for editor actions.
- **void setPadding(int left, int top, int right, int bottom)**: Sets absolute padding.
- **void setPaddingRelative(int start, int top, int end, int bottom)**: Sets relative padding.
- **void setPaintFlags(int flags)**: Sets paint flags and reflows text if changed.
- **void setPrivateImeOptions(String type)**: Sets private IME options string.

- **void setRawInputType(int type)**: Directly sets the content type integer.
- **void setScroller(Scroller s)**: Sets the `Scroller` for scrolling animation.
- **void setSearchResultHighlightColor(int color)**: Sets search result highlight color.
- **void setSearchResultHighlights(int... ranges)**: Sets search result ranges (flattened).
- **void setSelectAllOnFocus(boolean selectAllOnFocus)**: Selects all text when gaining focus.
- **void setSelected(boolean selected)**: Changes selection state of this view.
- **void setShadowLayer(float radius, float dx, float dy, int color)**: Applies a text shadow.
- **void setShiftDrawingOffsetForStartOverhang(boolean shiftDrawingOffsetForStartOverhang)**: Enables shifting x offset to show start overhang.
- **final void setShowSoftInputOnFocus(boolean show)**: Controls soft input visibility on focus.
- **void setSingleLine(boolean singleLine)**: Toggles single-line properties.
- **void setSingleLine()**: Sets single-line properties.
- **final void setSpannableFactory(Spannable.Factory factory)**: Sets the `Spannable.Factory`.
- **final void setText(int resid)**: Sets text from a string resource.
- **final void setText(CharSequence text)**: Sets the displayed text.
- **void setText(CharSequence text, TextView.BufferType type)**: Sets text and buffer type.
- **final void setText(int resid, TextView.BufferType type)**: Sets text from a resource with buffer type.
- **final void setText(char[] text, int start, int len)**: Displays a slice of a char array.
- **void setTextAppearance(Context context, int resId)**: *Deprecated API 23* — use `setTextAppearance(int)`.
- **void setTextAppearance(int resId)**: Sets the text appearance from a style resource.
- **void setTextClassifier(TextClassifier textClassifier)**: Sets the `TextClassifier`.
- **void setTextColor(int color)**: Sets text color for all states.
- **void setTextColor(ColorStateList colors)**: Sets text colors.
- **void setTextCursorDrawable(Drawable textCursorDrawable)**: Sets cursor drawable.
- **void setTextCursorDrawable(int textCursorDrawable)**: Sets cursor drawable by resource.
- **void setTextIsSelectable(boolean selectable)**: Toggles text selectability.
- **final void setTextKeepState(CharSequence text)**: Sets text while retaining cursor position.
- **final void setTextKeepState(CharSequence text, TextView.BufferType type)**: Sets text and buffer type while retaining cursor position.
- **void setTextLocale(Locale locale)**: Sets text `Locale` to a single-locale list.
- **void setTextLocales(LocaleList locales)**: Sets text `LocaleList`.

- **void setTextMetricsParams(PrecomputedText.Params params)**: Applies text layout parameters.
- **void setTextScaleX(float size)**: Sets horizontal text scale.
- **void setTextSelectHandle(int textSelectHandle)**: Sets selection handle drawable (resource).
- **void setTextSelectHandle(Drawable textSelectHandle)**: Sets selection handle drawable.
- **void setTextSelectHandleLeft(int textSelectHandleLeft)**: Sets left selection handle (resource).
- **void setTextSelectHandleLeft(Drawable textSelectHandleLeft)**: Sets left selection handle.
- **void setTextSelectHandleRight(Drawable textSelectHandleRight)**: Sets right selection handle.
- **void setTextSelectHandleRight(int textSelectHandleRight)**: Sets right selection handle (resource).
- **void setTextSize(int unit, float size)**: Sets text size with unit.
- **void setTextSize(float size)**: Sets text size in scaled pixels.
- **final void setTransformationMethod(TransformationMethod method)**: Sets text transformation method.
- **void setTypeface(Typeface tf)**: Sets the typeface.
- **void setTypeface(Typeface tf, int style)**: Sets typeface and style (enables fake bold/italic if needed).
- **void setUseBoundsForWidth(boolean useBoundsForWidth)**: Uses bounding-box width for line breaking/drawing.
- **void setWidth(int pixels)**: Sets exact width in pixels.
- **boolean showContextMenu()**: Shows the context menu.
- **boolean showContextMenu(float x, float y)**: Shows the context menu anchored at the given coordinates.

- **Protected methods**

  - **int computeHorizontalScrollRange()**: Computes the horizontal range represented by the horizontal scrollbar.
  - **int computeVerticalScrollExtent()**: Computes the vertical extent of the scrollbar thumb within the total vertical range.
  - **int computeVerticalScrollRange()**: Computes the vertical range represented by the vertical scrollbar.
  - **void drawableStateChanged()**: Called when the view state changes in a way that affects any displayed drawables.
  - **int getBottomPaddingOffset()**: Returns the amount by which to extend the bottom fading region.
  - **boolean getDefaultEditable()**: Indicates whether the view has a default `KeyListener` even if not requested via XML.
  - **MovementMethod getDefaultMovementMethod()**: Returns the default `MovementMethod` for this view.
  - **float getLeftFadingEdgeStrength()**: Returns the intensity of the left faded edge.

180

- **int getLeftPaddingOffset()**: Returns the amount to extend the left fading region.
- **float getRightFadingEdgeStrength()**: Returns the intensity of the right faded edge.
- **int getRightPaddingOffset()**: Returns the amount to extend the right fading region.
- **int getTopPaddingOffset()**: Returns the amount to extend the top fading region.
- **boolean isPaddingOffsetRequired()**: True if the view draws inside padding and supports fading edge padding offsets.
- **void onAttachedToWindow()**: Called when the view is attached to a window.
- **void onConfigurationChanged(Configuration newConfig)**: Called when the device/app configuration changes.
- **void onCreateContextMenu(ContextMenu menu)**: Allows the view to add items to its context menu.
- **int[] onCreateDrawableState(int extraSpace)**: Generates a new drawable state array for this view.
- **void onDraw(Canvas canvas)**: Performs custom drawing for this view.
- **void onFocusChanged(boolean focused, int direction, Rect previouslyFocusedRect)**: Called when the view's focus state changes.
- **void onLayout(boolean changed, int left, int top, int right, int bottom)**: Assigns size and position to child views.
- **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: Measures the view and its content to determine width and height.
- **void onScrollChanged(int horiz, int vert, int oldHoriz, int oldVert)**: Called when the view scrolls its own contents.
- **void onSelectionChanged(int selStart, int selEnd)**: Called when the text selection changes.
- **void onTextChanged(CharSequence text, int start, int lengthBefore, int lengthAfter)**: Called when the text content changes.
- **void onVisibilityChanged(View changedView, int visibility)**: Called when visibility of this view or an ancestor changes.
- **boolean setFrame(int l, int t, int r, int b)**: Sets the view's frame; returns true if it changed.
- **boolean verifyDrawable(Drawable who)**: Returns true for any drawable that this view is displaying.

- **Constants**

  - **int AUTO_SIZE_TEXT_TYPE_NONE**: The TextView does not auto-size text (default).
  - **int AUTO_SIZE_TEXT_TYPE_UNIFORM**: The TextView scales text size both horizontally and vertically to fit within the container.
  - **int FOCUSED_SEARCH_RESULT_INDEX_NONE**: A special index used for setFocusedSearchResultIndex(int) and getFocusedSearchResultIndex() inidicating there is no focused search result.

1.4.19 EditText

- **Hierarchy**

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

- **Include**

  ```
  0   android.widget.EditText
  ```

- **Constructors**

  ```
  0   EditText(Context context)
  1   EditText(Context context, AttributeSet attrs)
  2   EditText(Context context, AttributeSet attrs, int
      ↪  defStyleAttr)
  3   EditText(Context context, AttributeSet attrs, int
      ↪  defStyleAttr, int defStyleRes)
  ```

- **Public methods**

  - **void extendSelection(int index)**: Convenience method for `Selection.extendSelection`.
  - **CharSequence getAccessibilityClassName()**: Returns the class name of this object to be used for accessibility purposes.
  - **boolean getFreezesText()**: Returns whether this `TextView` includes its entire text contents in frozen icicles.
  - **Editable getText()**: Returns the text that the `TextView` is displaying.
  - **boolean isStyleShortcutEnabled()**: Returns true if style shortcuts are enabled, otherwise false.
  - **boolean onKeyShortcut(int keyCode, KeyEvent event)**: Called on the focused view when a key shortcut event is not handled.
  - **boolean onTextContextMenuItem(int id)**: Called when a context menu option for the text view is selected.
  - **void selectAll()**: Convenience method for `Selection.selectAll`.
  - **void setEllipsize(TextUtils.TruncateAt ellipsis)**: Specifies how overflowing text should be ellipsized instead of wrapped.
  - **void setSelection(int index)**: Convenience method for `Selection.setSelection(Spannable, int)`.
  - **void setSelection(int start, int stop)**: Convenience method for `Selection.setSelection(Spannable, int, int)`.
  - **void setStyleShortcutsEnabled(boolean enabled)**: Enables style shortcuts such as `Ctrl+B` for bold.
  - **void setText(CharSequence text, TextView.BufferType type)**: Sets the text to be displayed and the `TextView.BufferType`.

- **Protected methods**

– **boolean getDefaultEditable()**: Subclasses override this to specify that they have a KeyListener by default even if not specifically called for in the XML options.

– **MovementMethod getDefaultMovementMethod()**: Subclasses override this to specify a default movement method.

1.4.20   Button

- **Hierarchy**

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

- **Include**

  ```
  0   android.widget.Button
  ```

- **Constructors**

  ```
  0   Button(Context context)
  1   Button(Context context, AttributeSet attrs)
  2   Button(Context context, AttributeSet attrs, int defStyleAttr)
  3   Button(Context context, AttributeSet attrs, int
      ↪  defStyleAttr, int defStyleRes)
  ```

- **Public methods**

  - **CharSequence getAccessibilityClassName()**: Return the class name of this
    object to be used for accessibility purposes.

  - **PointerIcon onResolvePointerIcon(MotionEvent event, int pointerIndex)**:
    Resolve the pointer icon that should be used for specified pointer in the motion
    event.

1.4.21   ImageView

- **Hierarchy**

  java.lang.Object → android.view.View → android.widget.ImageView

- **Include**

```
0   android.widget.ImageView
```

- **Constructors**

```
0   ImageView(Context context)
1   ImageView(Context context, AttributeSet attrs)
2   ImageView(Context context, AttributeSet attrs, int
 ↪   defStyleAttr)
3   ImageView(Context context, AttributeSet attrs, int
 ↪   defStyleAttr, int defStyleRes)
```

- **Public methods**

  - **void animateTransform(Matrix matrix)**: Applies a temporary transformation matrix to the view's drawable when it is drawn.
  - **final void clearColorFilter()**: Removes the image's `ColorFilter`.
  - **void drawableHotspotChanged(float x, float y)**: Called whenever the view hotspot changes and needs to be propagated to drawables or child views managed by the view.
  - **CharSequence getAccessibilityClassName()**: Returns the class name of this object to be used for accessibility purposes.
  - **boolean getAdjustViewBounds()**: Returns true if the ImageView is adjusting its bounds to preserve the aspect ratio of its drawable.
  - **int getBaseline()**: Returns the offset of the widget's text baseline from the widget's top boundary.
  - **boolean getBaselineAlignBottom()**: Checks whether this view's baseline is considered the bottom of the view.
  - **ColorFilter getColorFilter()**: Returns the active color filter for this ImageView.
  - **boolean getCropToPadding()**: Returns whether this ImageView crops to its padding.
  - **Drawable getDrawable()**: Gets the current drawable, or null if none has been assigned.
  - **int getImageAlpha()**: Returns the alpha value applied to the drawable of this ImageView.
  - **Matrix getImageMatrix()**: Returns the view's transformation matrix, if any.
  - **BlendMode getImageTintBlendMode()**: Gets the blending mode used to apply the tint to the image drawable.
  - **ColorStateList getImageTintList()**: Returns the current color tint list used for the image drawable.

- **PorterDuff.Mode getImageTintMode()**: Gets the blending mode used to apply the tint to the image drawable.
- **int getMaxHeight()**: Returns the maximum height of this view.
- **int getMaxWidth()**: Returns the maximum width of this view.
- **ImageView.ScaleType getScaleType()**: Returns the current scale type used to resize or move the image.
- **boolean hasOverlappingRendering()**: Returns whether this view has overlapping content.
- **void invalidateDrawable(Drawable dr)**: Invalidates the specified drawable.
- **boolean isOpaque()**: Indicates whether this view is opaque.
- **void jumpDrawablesToCurrentState()**: Calls `Drawable.jumpToCurrentState()` on all associated drawables.
- **int[] onCreateDrawableState(int extraSpace)**: Generates the new drawable state for this view.
- **void onRtlPropertiesChanged(int layoutDirection)**: Called when any RTL property (layout direction or text alignment) changes.
- **void onVisibilityAggregated(boolean isVisible)**: Called when the visibility of this view or one of its ancestors changes.
- **void setAdjustViewBounds(boolean adjustViewBounds)**: When true, adjusts the bounds to preserve the drawable's aspect ratio.
- **void setAlpha(int alpha)**: *Deprecated in API 16.* Use `setImageAlpha(int)` instead.
- **void setBaseline(int baseline)**: Sets the offset of the widget's text baseline from the widget's top boundary.
- **void setBaselineAlignBottom(boolean aligned)**: Sets whether the baseline is considered the bottom of the view.
- **final void setColorFilter(int color, PorterDuff.Mode mode)**: Sets a tint color and mode for the image.
- **void setColorFilter(ColorFilter cf)**: Applies a custom color filter to the image.
- **final void setColorFilter(int color)**: Sets a tint color for the image.
- **void setCropToPadding(boolean cropToPadding)**: Sets whether this ImageView will crop its content to padding.
- **void setImageAlpha(int alpha)**: Sets the alpha transparency applied to the image.
- **void setImageBitmap(Bitmap bm)**: Sets a bitmap as the content of the ImageView.
- **void setImageDrawable(Drawable drawable)**: Sets a drawable as the content of the ImageView.
- **void setImageIcon(Icon icon)**: Sets an icon as the content of the ImageView.
- **void setImageLevel(int level)**: Sets the level for a `LevelListDrawable`.
- **void setImageMatrix(Matrix matrix)**: Applies a transformation matrix to the drawable.
- **void setImageResource(int resId)**: Sets a drawable resource as the content of the ImageView.

– **void setImageState(int[] state, boolean merge)**: Sets the drawable state for a `StateListDrawable`.

– **void setImageTintBlendMode(BlendMode blendMode)**: Sets the blending mode for applying the tint.

– **void setImageTintList(ColorStateList tint)**: Applies a tint list to the image drawable.

– **void setImageTintMode(PorterDuff.Mode tintMode)**: Sets the blending mode for applying the tint.

– **void setImageURI(Uri uri)**: Sets the content of this ImageView to the image located at the specified URI.

– **void setMaxHeight(int maxHeight)**: Sets a maximum height for the view.

– **void setMaxWidth(int maxWidth)**: Sets a maximum width for the view.

– **void setScaleType(ImageView.ScaleType scaleType)**: Defines how the image should be resized or moved to fit the view bounds.

– **void setSelected(boolean selected)**: Changes the selection state of this view.

– **void setVisibility(int visibility)**: Sets the visibility state of this view.

- **Protected methods**

  – **void drawableStateChanged()**: Called whenever the state of the view changes in a way that affects the state of its drawables.

  – **void onAttachedToWindow()**: Called when the view is attached to a window.

  – **void onDetachedFromWindow()**: Called when the view is detached from a window.

  – **void onDraw(Canvas canvas)**: Implement this method to perform custom drawing for the view.

  – **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: Measures the view and its content to determine the measured width and height.

  – **boolean setFrame(int l, int t, int r, int b)**: Assigns size and position to the view within its parent layout.

  – **boolean verifyDrawable(Drawable dr)**: Returns true if the view is displaying the given drawable; subclasses should override when managing their own drawables.

1.4.22   CompoundButton

- **Hierarchy**

  java.lang.Object → android.view.View → android.widget.TextView →
  android.widget.Button → android.widget.CompoundButton

- **Include**

  ```
  0   android.widget.CompoundButton
  ```

- **Constructors**

  ```
  0   CompoundButton(Context context)
  1   CompoundButton(Context context, AttributeSet attrs)
  2   CompoundButton(Context context, AttributeSet attrs, int
      ↪   defStyleAttr)
  3   CompoundButton(Context context, AttributeSet attrs, int
      ↪   defStyleAttr, int defStyleRes)
  ```

- **Public methods**

  - **void autofill(AutofillValue value)**: Automatically fills the content of this view
    with the given autofill value.
  - **void drawableHotspotChanged(float x, float y)**: Called when the view
    hotspot changes and must be propagated to drawables or child views.
  - **CharSequence getAccessibilityClassName()**: Returns the class name of this
    object for accessibility purposes.
  - **int getAutofillType()**: Describes the autofill type of this view, allowing an
    `AutofillService` to create an appropriate `AutofillValue`.
  - **AutofillValue getAutofillValue()**: Returns the current text of this `TextView`
    for autofill purposes.
  - **Drawable getButtonDrawable()**: Returns the button drawable associated
    with this compound button.
  - **BlendMode getButtonTintBlendMode()**: Returns the blending mode used
    to apply the tint to the button drawable.
  - **ColorStateList getButtonTintList()**: Returns the tint list applied to the
    button drawable.
  - **PorterDuff.Mode getButtonTintMode()**: Returns the blending mode used
    to apply the tint to the button drawable.
  - **int getCompoundPaddingLeft()**: Returns the left padding of the view, in-
    cluding space for the left drawable if present.
  - **int getCompoundPaddingRight()**: Returns the right padding of the view,
    including space for the right drawable if present.
  - **boolean isChecked()**: Returns the checked state of this button.
  - **void jumpDrawablesToCurrentState()**: Calls `Drawable.jumpToCurrentState()`
    on all drawable objects associated with this view.
  - **void onRestoreInstanceState(Parcelable state)**: Restores the internal state
    of the view from a previously saved state.

- **Parcelable onSaveInstanceState()**: Saves the internal state of the view for later restoration.
- **boolean performClick()**: Calls this view's `OnClickListener`, if one is defined.
- **void setButtonDrawable(int resId)**: Sets a drawable as the compound button image using its resource identifier.
- **void setButtonDrawable(Drawable drawable)**: Sets a drawable as the compound button image.
- **void setButtonIcon(Icon icon)**: Sets the button of this compound button to the specified icon.
- **void setButtonTintBlendMode(BlendMode tintMode)**: Sets the blending mode used to apply the tint specified by `setButtonTintList()`.
- **void setButtonTintList(ColorStateList tint)**: Applies a color tint to the button drawable.
- **void setButtonTintMode(PorterDuff.Mode tintMode)**: Specifies the blending mode used to apply the tint specified by `setButtonTintList()`.
- **void setChecked(boolean checked)**: Changes the checked state of this button.
- **void setOnCheckedChangeListener(CompoundButton.OnCheckedChangeListener listener)**: Registers a callback to be invoked when the checked state changes.
- **void setStateDescription(CharSequence stateDescription)**: Called when the view or subclass sets the state description for accessibility.
- **void toggle()**: Toggles the checked state of the button to the opposite of its current state.

- **Protected methods**
  - **void drawableStateChanged()**: Called whenever the state of the view changes in a way that affects the state of its drawables.
  - **int[] onCreateDrawableState(int extraSpace)**: Generates and returns the new drawable state array for this view, allocating extra space if needed.
  - **void onDraw(Canvas canvas)**: Implement this method to perform custom drawing for the view.
  - **boolean verifyDrawable(Drawable who)**: Returns true if the specified drawable is being displayed by this view; subclasses should override when managing their own drawables.

1.4.23 CheckBox

- **Hierarchy**

  java.lang.Object → android.view.View → android.widget.TextView →
  android.widget.Button → android.widget.CompoundButton →
  android.widget.CheckBox

- **Include**

  ```
  0  android.widget.CheckBox
  ```

- **Constructors**

  ```
  0  CheckBox(Context context)
  1  CheckBox(Context context, AttributeSet attrs)
  2  CheckBox(Context context, AttributeSet attrs, int
     ↪  defStyleAttr)
  3  CheckBox(Context context, AttributeSet attrs, int
     ↪  defStyleAttr, int defStyleRes)
  ```

- **Public methods**

  – **CharSequence getAccessibilityClassName()**: Return the class name of this
    object to be used for accessibility purposes.

1.4.24 RadioGroup

- **Hierarchy**

  java.lang.Object → android.view.View → android.view.ViewGroup →
  android.widget.LinearLayout → android.widget.RadioGroup

- **Include**

  ```
  0   android.widget.RadioGroup
  ```

- **Constructors**

  ```
  0   RadioGroup(Context context)
  1   RadioGroup(Context context, AttributeSet attrs)
  ```

- **Public methods**

  - **void addView(View child, int index, ViewGroup.LayoutParams params)**: Adds a child view with the specified layout parameters.
  - **void autofill(AutofillValue value)**: Automatically fills the content of this view with the value.
  - **void check(int id)**: Sets the selection to the radio button whose identifier is passed in parameter.
  - **void clearCheck()**: Clears the selection.
  - **RadioGroup.LayoutParams generateLayoutParams(AttributeSet attrs)**: Returns a new set of layout parameters based on the supplied attributes set.
  - **CharSequence getAccessibilityClassName()**: Return the class name of this object to be used for accessibility purposes.
  - **int getAutofillType()**: Describes the autofill type of this view, so an AutofillService can create the proper AutofillValue when autofilling the view.
  - **AutofillValue getAutofillValue()**: Gets the View's current autofill value.
  - **int getCheckedRadioButtonId()**: Returns the identifier of the selected radio button in this group.
  - **void onInitializeAccessibilityNodeInfo(AccessibilityNodeInfo info)**: Initializes an AccessibilityNodeInfo with information about this view.
  - **void setOnCheckedChangeListener(RadioGroup.OnCheckedChangeListener listener)**: Register a callback to be invoked when the checked radio button changes in this group.
  - **void setOnHierarchyChangeListener(ViewGroup.OnHierarchyChangeListener listener)**: Register a callback to be invoked when a child is added to or removed from this view.

- **Protected methods**

  - **boolean checkLayoutParams(ViewGroup.LayoutParams p)**:
  - **LinearLayout.LayoutParams generateDefaultLayoutParams()**: Returns a set of layout parameters with a width of ViewGroup.LayoutParams.MATCH_PARENT and a height of ViewGroup.LayoutParams.WRAP_CONTENT when the layout's orientation is VERTICAL.
  - **void onFinishInflate()**: Finalize inflating a view from XML.

1.4.25 RadioGroup.LayoutParams

- **Hierarchy**:

  java.lang.Object → android.view.ViewGroup.LayoutParams →
  android.view.ViewGroup.MarginLayoutParams →
  android.widget.LinearLayout.LayoutParams →
  android.widget.RadioGroup.LayoutParams

- **Include**

  ```
  0   android.widget.RadioGroup.LayoutParams
  ```

- **constructors**

  ```
  0   LayoutParams(Context c, AttributeSet attrs)
  1   LayoutParams(ViewGroup.LayoutParams p)
  2   LayoutParams(ViewGroup.MarginLayoutParams source)
  3   LayoutParams(int w, int h)
  4   LayoutParams(int w, int h, float initWeight)
  ```

- **Protected methods**

  - **void setBaseAttributes(TypedArray a, int widthAttr, int heightAttr)**:
    Fixes the child's width to ViewGroup.LayoutParams.WRAP_CONTENT and
    the child's height to ViewGroup.LayoutParams.WRAP_CONTENT when not
    specified in the XML file.

1.4.26   RadioButton

- **Hierarchy**

  java.lang.Object → android.view.View → android.widget.TextView →
  android.widget.Button → android.widget.CompoundButton →
  android.widget.RadioButton

- **Include**

```
0   android.widget.RadioButton
```

- **Constructors**

```
0   Public constructors
1   RadioButton(Context context)
2   RadioButton(Context context, AttributeSet attrs)
3   RadioButton(Context context, AttributeSet attrs, int
    ↪   defStyleAttr)
4   RadioButton(Context context, AttributeSet attrs, int
    ↪   defStyleAttr, int defStyleRes)
```

- **Public methods**

  - **CharSequence getAccessibilityClassName()**: Return the class name of this object to be used for accessibility purposes.

  - **void onInitializeAccessibilityNodeInfo(AccessibilityNodeInfo info)**: Initializes an AccessibilityNodeInfo with information about this view.

  - **void toggle()**: Change the checked state of the view to the inverse of its current state

    If the radio button is already checked, this method will not toggle the radio button.

1.4.27 AbsSpinner

- **Hierarchy**

  java.lang.Object → android.view.View → android.view.ViewGroup →
  android.widget.AdapterView<android.widget.SpinnerAdapter> →
  android.widget.AbsSpinner

- **Include**

  ```
  0   android.widget.AbsSpinner
  ```

- **Constructors**

  ```
  0   AbsSpinner(Context context)
  1   AbsSpinner(Context context, AttributeSet attrs)
  2   AbsSpinner(Context context, AttributeSet attrs, int
      ↪  defStyleAttr)
  3   AbsSpinner(Context context, AttributeSet attrs, int
      ↪  defStyleAttr, int defStyleRes)
  ```

- **Public methods**

  - **void autofill(AutofillValue value)**: Automatically fills the content of this view with the value.
  - **CharSequence getAccessibilityClassName()**: Return the class name of this object to be used for accessibility purposes.
  - **SpinnerAdapter getAdapter()**: Returns the adapter currently associated with this widget.
  - **int getAutofillType()**: Describes the autofill type of this view, so an AutofillService can create the proper AutofillValue when autofilling the view.
  - **AutofillValue getAutofillValue()**: Gets the View's current autofill value.
  - **int getCount()**:
  - **View getSelectedView()**:
  - **void onRestoreInstanceState(Parcelable state)**: Hook allowing a view to re-apply a representation of its internal state that had previously been generated by onSaveInstanceState().
  - **Parcelable onSaveInstanceState()**: Hook allowing a view to generate a representation of its internal state that can later be used to create a new instance with that same state.
  - **int pointToPosition(int x, int y)**: Maps a point to a position in the list.
  - **void requestLayout()**: Override to prevent spamming ourselves with layout requests as we place views
  - **void setAdapter(SpinnerAdapter adapter)**: The Adapter is used to provide the data which backs this Spinner.
  - **void setSelection(int position, boolean animate)**: Jump directly to a specific item in the adapter data.

194

– **void setSelection(int position)**: Sets the currently selected item.

- **Protected methods**

  – **void dispatchRestoreInstanceState(SparseArray<Parcelable> container)**: Override to prevent thawing of any views created by the adapter.

  – **ViewGroup.LayoutParams generateDefaultLayoutParams()**: Returns a set of default layout parameters.

  – **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: Measure the view and its content to determine the measured width and the measured height.

1.4.28 Spinner

- **Hierarchy**

    java.lang.Object → android.view.View → android.view.ViewGroup →
    android.widget.AdapterView<android.widget.SpinnerAdapter> →
    android.widget.AbsSpinner → android.widget.Spinner

- **Include**

```
0   android.widget.Spinner
```

- **Constructors**

```
0   Spinner(Context context)
1   Spinner(Context context, AttributeSet attrs)
2   Spinner(Context context, AttributeSet attrs, int
    ↪   defStyleAttr)
3   Spinner(Context context, AttributeSet attrs, int
    ↪   defStyleAttr, int mode)
4   Spinner(Context context, AttributeSet attrs, int
    ↪   defStyleAttr, int defStyleRes, int mode)
5   Spinner(Context context, AttributeSet attrs, int
    ↪   defStyleAttr, int defStyleRes, int mode, Resources.Theme
    ↪   popupTheme)
6   Spinner(Context context, int mode)
```

- **Public methods**

    – **CharSequence getAccessibilityClassName()**: Returns the class name of this object to be used for accessibility purposes.

    – **int getBaseline()**: Returns the offset of the widget's text baseline from the widget's top boundary.

    – **int getDropDownHorizontalOffset()**: Returns the configured horizontal offset in pixels for the spinner's popup window of choices.

    – **int getDropDownVerticalOffset()**: Returns the configured vertical offset in pixels for the spinner's popup window of choices.

    – **int getDropDownWidth()**: Returns the configured width of the spinner's popup window of choices in pixels.

    – **int getGravity()**: Describes how the selected item view is positioned within the spinner.

    – **Drawable getPopupBackground()**: Returns the background drawable for the spinner's popup window of choices.

    – **Context getPopupContext()**: Returns the context used to inflate the spinner's popup window.

    – **CharSequence getPrompt()**: Returns the prompt text displayed when the spinner dialog is shown.

    – **void onClick(DialogInterface dialog, int which)**: Invoked when a button in the dialog is clicked.

196

- **PointerIcon onResolvePointerIcon(MotionEvent event, int pointerIndex)**: Resolves the pointer icon that should be used for the specified pointer in the motion event.
- **void onRestoreInstanceState(Parcelable state)**: Restores the internal state of the spinner from a previously saved state.
- **Parcelable onSaveInstanceState()**: Saves the spinner's current state for later restoration.
- **boolean onTouchEvent(MotionEvent event)**: Handles touch input events for the spinner.
- **boolean performClick()**: Calls this spinner's `OnClickListener`, if one is defined.
- **void setAdapter(SpinnerAdapter adapter)**: Sets the `SpinnerAdapter` that provides the data backing this spinner.
- **void setDropDownHorizontalOffset(int pixels)**: Sets a horizontal offset in pixels for the spinner's popup window of choices.
- **void setDropDownVerticalOffset(int pixels)**: Sets a vertical offset in pixels for the spinner's popup window of choices.
- **void setDropDownWidth(int pixels)**: Sets the width of the spinner's popup window of choices in pixels.
- **void setEnabled(boolean enabled)**: Sets the enabled state of this spinner.
- **void setGravity(int gravity)**: Defines how the selected item view is positioned within the spinner.
- **void setOnItemClickListener(AdapterView.OnItemClickListener l)**: No-op; spinners do not support item click events.
- **void setPopupBackgroundDrawable(Drawable background)**: Sets the background drawable for the spinner's popup window of choices.
- **void setPopupBackgroundResource(int resId)**: Sets the background resource for the spinner's popup window of choices.
- **void setPrompt(CharSequence prompt)**: Sets the prompt text to display when the dialog is shown.
- **void setPromptId(int promptId)**: Sets the prompt text to display when the dialog is shown using a resource ID.

- **Protected methods**

  - **void onDetachedFromWindow()**: This is called when the view is detached from a window.
  - **void onLayout(boolean changed, int l, int t, int r, int b)**: Called from layout when this view should assign a size and position to each of its children.
  - **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: Measure the view and its content to determine the measured width and the measured height.

- **Constants**

  - **int MODE_DIALOG**: Use a dialog window for selecting spinner options.
  - **int MODE_DROPDOWN**: Use a dropdown anchored to the Spinner for selecting spinner options.

1.4.29 Progessbar

- **Hierarchy**

  java.lang.Object → android.view.View → android.widget.ProgressBar

- **Include**

  ```
  0  android.widget.ProgressBar
  ```

- **Constructors**

  ```
  0  ProgressBar(Context context)
  1  ProgressBar(Context context, AttributeSet attrs)
  2  ProgressBar(Context context, AttributeSet attrs, int
     ↪  defStyleAttr)
  3  ProgressBar(Context context, AttributeSet attrs, int
     ↪  defStyleAttr, int defStyleRes)
  ```

- **Public methods**

  - **void drawableHotspotChanged(float x, float y)**: Called whenever the view hotspot changes and must be propagated to drawables or child views.
  - **CharSequence getAccessibilityClassName()**: Returns the class name of this object for accessibility purposes.
  - **Drawable getCurrentDrawable()**: Returns the drawable currently used to draw the progress bar.
  - **Drawable getIndeterminateDrawable()**: Returns the drawable used to draw the progress bar in indeterminate mode.
  - **BlendMode getIndeterminateTintBlendMode()**: Returns the blending mode used to apply the tint to the indeterminate drawable, if specified.
  - **ColorStateList getIndeterminateTintList()**: Returns the color tint list used for the indeterminate drawable.
  - **PorterDuff.Mode getIndeterminateTintMode()**: Returns the blending mode used to apply the tint to the indeterminate drawable.
  - **Interpolator getInterpolator()**: Gets the acceleration curve type for the indeterminate animation.
  - **int getMax()**: Returns the upper limit of this progress bar's range.
  - **int getMaxHeight()**: Returns the maximum height of the progress bar, in pixels.
  - **int getMaxWidth()**: Returns the maximum width of the progress bar, in pixels.
  - **int getMin()**: Returns the lower limit of this progress bar's range.
  - **int getMinHeight()**: Returns the minimum height of the progress bar, in pixels.
  - **int getMinWidth()**: Returns the minimum width of the progress bar, in pixels.
  - **int getProgress()**: Returns the current progress level of the progress bar.
  - **BlendMode getProgressBackgroundTintBlendMode()**: Returns the blending mode used to apply the tint to the progress background, if specified.

- **ColorStateList getProgressBackgroundTintList()**: Returns the tint list applied to the progress background, if specified.
- **PorterDuff.Mode getProgressBackgroundTintMode()**: Returns the blending mode used to apply the tint to the progress background.
- **Drawable getProgressDrawable()**: Returns the drawable used to draw the progress bar in progress mode.
- **BlendMode getProgressTintBlendMode()**: Returns the blending mode used to apply the tint to the progress drawable, if specified.
- **ColorStateList getProgressTintList()**: Returns the color tint list applied to the progress drawable.
- **PorterDuff.Mode getProgressTintMode()**: Returns the blending mode used to apply the tint to the progress drawable.
- **int getSecondaryProgress()**: Returns the current level of secondary progress.
- **BlendMode getSecondaryProgressTintBlendMode()**: Returns the blending mode used to apply the tint to the secondary progress drawable.
- **ColorStateList getSecondaryProgressTintList()**: Returns the color tint list applied to the secondary progress drawable.
- **PorterDuff.Mode getSecondaryProgressTintMode()**: Returns the blending mode used to apply the tint to the secondary progress drawable.
- **final void incrementProgressBy(int diff)**: Increases the primary progress by the specified amount.
- **final void incrementSecondaryProgressBy(int diff)**: Increases the secondary progress by the specified amount.
- **void invalidateDrawable(Drawable dr)**: Invalidates the specified drawable, forcing a redraw.
- **boolean isAnimating()**: Returns whether the progress bar is currently animating.
- **boolean isIndeterminate()**: Indicates whether this progress bar is in indeterminate mode.
- **void jumpDrawablesToCurrentState()**: Calls `Drawable.jumpToCurrentState()` on all associated drawables.
- **void onRestoreInstanceState(Parcelable state)**: Restores the internal state of the progress bar from a previously saved state.
- **Parcelable onSaveInstanceState()**: Saves the internal state of the progress bar for later restoration.
- **void onVisibilityAggregated(boolean isVisible)**: Called when the visibility of this view or its ancestors changes.
- **void postInvalidate()**: Schedules a redraw for the next event loop cycle.
- **void setIndeterminate(boolean indeterminate)**: Changes whether the progress bar is in indeterminate mode.
- **void setIndeterminateDrawable(Drawable d)**: Defines the drawable used to draw the progress bar in indeterminate mode.
- **void setIndeterminateDrawableTiled(Drawable d)**: Defines a tileable drawable used to draw the indeterminate progress bar.
- **void setIndeterminateTintBlendMode(BlendMode blendMode)**: Specifies the blending mode for applying the indeterminate tint.

199

- **void setIndeterminateTintList(ColorStateList tint)**: Applies a color tint to the indeterminate drawable.
- **void setIndeterminateTintMode(PorterDuff.Mode tintMode)**: Specifies the blending mode for applying the indeterminate tint.
- **void setInterpolator(Interpolator interpolator)**: Sets the acceleration curve for the indeterminate animation.
- **void setInterpolator(Context context, int resID)**: Sets the interpolator resource for the indeterminate animation.
- **void setMax(int max)**: Sets the upper range of the progress bar.
- **void setMaxHeight(int maxHeight)**: Sets the maximum height the progress bar can have.
- **void setMaxWidth(int maxWidth)**: Sets the maximum width the progress bar can have.
- **void setMin(int min)**: Sets the lower range of the progress bar.
- **void setMinHeight(int minHeight)**: Sets the minimum height the progress bar can have.
- **void setMinWidth(int minWidth)**: Sets the minimum width the progress bar can have.
- **void setProgress(int progress)**: Sets the current progress value.
- **void setProgress(int progress, boolean animate)**: Sets the current progress value, optionally animating the transition.
- **void setProgressBackgroundTintBlendMode(BlendMode blendMode)**: Specifies the blending mode for the progress background tint.
- **void setProgressBackgroundTintList(ColorStateList tint)**: Applies a tint to the progress background.
- **void setProgressBackgroundTintMode(PorterDuff.Mode tintMode)**: Specifies the blending mode for the progress background tint.
- **void setProgressDrawable(Drawable d)**: Defines the drawable used to draw the progress bar in progress mode.
- **void setProgressDrawableTiled(Drawable d)**: Defines a tileable drawable for the progress bar in progress mode.
- **void setProgressTintBlendMode(BlendMode blendMode)**: Specifies the blending mode for the progress indicator tint.
- **void setProgressTintList(ColorStateList tint)**: Applies a tint to the progress indicator.
- **void setProgressTintMode(PorterDuff.Mode tintMode)**: Specifies the blending mode for the progress indicator tint.
- **void setSecondaryProgress(int secondaryProgress)**: Sets the current secondary progress value.
- **void setSecondaryProgressTintBlendMode(BlendMode blendMode)**: Specifies the blending mode for the secondary progress tint.
- **void setSecondaryProgressTintList(ColorStateList tint)**: Applies a tint to the secondary progress indicator.
- **void setSecondaryProgressTintMode(PorterDuff.Mode tintMode)**: Specifies the blending mode for the secondary progress tint.
- **void setStateDescription(CharSequence stateDescription)**: Called when an instance or subclass sets the state description.

- **Protected methods**

  - **void drawableStateChanged()**: Called whenever the state of the view changes in a way that affects the state of its drawables.
  - **void onAttachedToWindow()**: Called when the view is attached to a window.
  - **void onDetachedFromWindow()**: Called when the view is detached from a window.
  - **void onDraw(Canvas canvas)**: Implement this method to perform custom drawing for the view.
  - **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: Measures the view and its content to determine the measured width and height.
  - **void onSizeChanged(int w, int h, int oldw, int oldh)**: Called during layout when the size of this view has changed.
  - **boolean verifyDrawable(Drawable who)**: Returns true if the specified drawable is being displayed by this view; subclasses should override this when managing their own drawables.

1.4.30 AbsSeekBar

- **Hierarchy**

  java.lang.Object → android.view.View → android.widget.ProgressBar →
  android.widget.AbsSeekBar

- **Include**

  ```
  0   android.widget.AbsSeekBar
  ```

- **Constructors**

  ```
  0   AbsSeekBar(Context context)
  1   AbsSeekBar(Context context, AttributeSet attrs)
  2   AbsSeekBar(Context context, AttributeSet attrs, int
      ↪  defStyleAttr)
  3   AbsSeekBar(Context context, AttributeSet attrs, int
      ↪  defStyleAttr, int defStyleRes)
  ```

- **Public methods**

  – **void drawableHotspotChanged(float x, float y)**: Called whenever the view
  hotspot changes and needs to be propagated to drawables or child views managed
  by the view.

  – **CharSequence getAccessibilityClassName()**: Returns the class name of this
  object to be used for accessibility purposes.

  – **int getKeyProgressIncrement()**: Returns the amount by which the progress
  changes when the user presses an arrow key.

  – **boolean getSplitTrack()**: Returns whether the track is split by the thumb.

  – **Drawable getThumb()**: Returns the drawable representing the scroll thumb
  — the component the user can drag to indicate progress.

  – **int getThumbOffset()**: Returns the amount by which the thumb extends be-
  yond the track.

  – **BlendMode getThumbTintBlendMode()**: Returns the blending mode used
  to apply the tint to the thumb drawable, if specified.

  – **ColorStateList getThumbTintList()**: Returns the tint color list applied to
  the thumb drawable, if specified.

  – **PorterDuff.Mode getThumbTintMode()**: Returns the blending mode used
  to apply the tint to the thumb drawable, if specified.

  – **Drawable getTickMark()**: Returns the drawable used as the tick mark for
  each progress position.

  – **BlendMode getTickMarkTintBlendMode()**: Returns the blending mode
  used to apply the tint to the tick mark drawable, if specified.

  – **ColorStateList getTickMarkTintList()**: Returns the tint color list applied
  to the tick mark drawable, if specified.

  – **PorterDuff.Mode getTickMarkTintMode()**: Returns the blending mode
  used to apply the tint to the tick mark drawable, if specified.

- **void jumpDrawablesToCurrentState()**: Immediately updates all drawables associated with this view to their current state.
- **boolean onKeyDown(int keyCode, KeyEvent event)**: Handles key press events such as DPAD center or enter when the view is enabled and clickable.
- **void onRtlPropertiesChanged(int layoutDirection)**: Called when any RTL (right-to-left) layout property or alignment has changed.
- **boolean onTouchEvent(MotionEvent event)**: Handles touch or pointer events for user interaction.
- **void setKeyProgressIncrement(int increment)**: Sets the amount by which progress changes when the user presses arrow keys.
- **void setMax(int max)**: Sets the maximum value of the progress range.
- **void setMin(int min)**: Sets the minimum value of the progress range.
- **void setSplitTrack(boolean splitTrack)**: Specifies whether the track should be visually split by the thumb.
- **void setSystemGestureExclusionRects(List<Rect> rects)**: Defines regions within the view where system gestures should not be intercepted.
- **void setThumb(Drawable thumb)**: Sets the drawable used as the thumb in the progress meter.
- **void setThumbOffset(int thumbOffset)**: Sets the offset allowing the thumb to extend beyond the track.
- **void setThumbTintBlendMode(BlendMode blendMode)**: Defines the blending mode used when applying tint to the thumb drawable.
- **void setThumbTintList(ColorStateList tint)**: Applies a tint color list to the thumb drawable.
- **void setThumbTintMode(PorterDuff.Mode tintMode)**: Specifies the blending mode used with the thumb tint.
- **void setTickMark(Drawable tickMark)**: Sets the drawable used as a tick mark at each progress position.
- **void setTickMarkTintBlendMode(BlendMode blendMode)**: Specifies the blending mode used to apply tint to the tick mark drawable.
- **void setTickMarkTintList(ColorStateList tint)**: Applies a tint color list to the tick mark drawable.
- **void setTickMarkTintMode(PorterDuff.Mode tintMode)**: Specifies the blending mode used with the tick mark tint.

- **Protected methods**

  - **void drawableStateChanged()**: Called whenever the state of the view changes in a way that affects the state of its drawables.
  - **void onDraw(Canvas canvas)**: Implement this method to perform custom drawing operations for the view.
  - **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: Measures the view and its content to determine the measured width and height.
  - **void onSizeChanged(int w, int h, int oldw, int oldh)**: Called during layout when the size of the view changes, providing both new and old dimensions.
  - **boolean verifyDrawable(Drawable who)**: Returns true if the specified drawable is managed and displayed by this view; subclasses should override when handling custom drawables.

1.4.31   SeekBar

- **Hierarchy**

  java.lang.Object → android.view.View → android.widget.ProgressBar →
  android.widget.AbsSeekBar → android.widget.SeekBar

- **Include**

  ```
  0   android.widget.SeekBar
  ```

- **Constructors**

  ```
  0   SeekBar(Context context)
  1   SeekBar(Context context, AttributeSet attrs)
  2   SeekBar(Context context, AttributeSet attrs, int
      ↪   defStyleAttr)
  3   SeekBar(Context context, AttributeSet attrs, int
      ↪   defStyleAttr, int defStyleRes)
  ```

- **Public methods**

  - **CharSequence getAccessibilityClassName()**: Return the class name of this
    object to be used for accessibility purposes.

  - **void setOnSeekBarChangeListener(SeekBar.OnSeekBarChangeListener
    l)**: Sets a listener to receive notifications of changes to the SeekBar's progress
    level.

1.4.32 Drawable

- **Hierarchy**

  java.lang.Object → android.graphics.drawable.Drawable

- **Include**

  o   android.`graphics`.`drawable`.`Drawable`

- **Constructors**

  o   `Drawable()`

- **Public methods**

  – **void applyTheme(Resources.Theme t)**: Applies the specified theme to this `Drawable` and its children.

  – **boolean canApplyTheme()**: Returns true if this `Drawable` can apply a theme.

  – **void clearColorFilter()**: Removes any color filter currently applied to the drawable.

  – **final Rect copyBounds()**: Returns a copy of the drawable's bounds in a new `Rect` object.

  – **final void copyBounds(Rect bounds)**: Copies the drawable's bounds into the specified `Rect`.

  – **static Drawable createFromPath(String pathName)**: Creates a drawable from the specified file path name.

  – **static Drawable createFromResourceStream(Resources res, TypedValue value, InputStream is, String srcName, BitmapFactory.Options opts)**: *Deprecated in API 28.* Creates a drawable from an input stream using the specified options.

  – **static Drawable createFromResourceStream(Resources res, TypedValue value, InputStream is, String srcName)**: Creates a drawable from an input stream using the given resources and density information.

  – **static Drawable createFromStream(InputStream is, String srcName)**: Creates a drawable from the specified input stream.

  – **static Drawable createFromXml(Resources r, XmlPullParser parser)**: Creates a drawable from an XML document.

  – **static Drawable createFromXml(Resources r, XmlPullParser parser, Resources.Theme theme)**: Creates a drawable from an XML document using the specified theme.

  – **static Drawable createFromXmlInner(Resources r, XmlPullParser parser, AttributeSet attrs, Resources.Theme theme)**: Creates a drawable from within an XML document using an optional theme.

  – **static Drawable createFromXmlInner(Resources r, XmlPullParser parser, AttributeSet attrs)**: Creates a drawable from within an XML document.

  – **abstract void draw(Canvas canvas)**: Draws the drawable within its bounds, respecting alpha and color filters.

- **int getAlpha()**: Returns the current alpha value for the drawable.
- **final Rect getBounds()**: Returns the drawable's bounding rectangle.
- **Drawable.Callback getCallback()**: Returns the current callback attached to this drawable.
- **int getChangingConfigurations()**: Returns a mask of configuration parameters that may change and require the drawable to be re-created.
- **ColorFilter getColorFilter()**: Returns the current color filter, or null if none is set.
- **Drawable.ConstantState getConstantState()**: Returns the constant state of this drawable, allowing shared state across instances.
- **Drawable getCurrent()**: Returns the current drawable in use, if the drawable supports multiple states.
- **Rect getDirtyBounds()**: Returns the drawable's dirty bounds rectangle.
- **void getHotspotBounds(Rect outRect)**: Populates the provided `Rect` with the hotspot bounds.
- **int getIntrinsicHeight()**: Returns the drawable's intrinsic (default) height.
- **int getIntrinsicWidth()**: Returns the drawable's intrinsic (default) width.
- **int getLayoutDirection()**: Returns the resolved layout direction for this drawable.
- **final int getLevel()**: Returns the current drawable level.
- **int getMinimumHeight()**: Returns the minimum height suggested by this drawable.
- **int getMinimumWidth()**: Returns the minimum width suggested by this drawable.
- **abstract int getOpacity()**: *Deprecated in API 29.* Returns the drawable's opacity mode.
- **Insets getOpticalInsets()**: Returns the optical insets for alignment during layout.
- **void getOutline(Outline outline)**: Populates the given `Outline` with the drawable's shape for rendering effects such as shadows.
- **boolean getPadding(Rect padding)**: Fills the given `Rect` with content insets suggested by the drawable.
- **int[] getState()**: Returns the current state of the drawable as an array of state attributes.
- **Region getTransparentRegion()**: Returns a region representing areas of complete transparency.
- **boolean hasFocusStateSpecified()**: Returns true if the drawable explicitly specifies a focused state.
- **void inflate(Resources r, XmlPullParser parser, AttributeSet attrs, Resources.Theme theme)**: Inflates this drawable from XML, optionally styled by a theme.
- **void inflate(Resources r, XmlPullParser parser, AttributeSet attrs)**: Inflates this drawable from XML.
- **void invalidateSelf()**: Requests a redraw of the drawable using its current callback.
- **boolean isAutoMirrored()**: Returns whether the drawable automatically mirrors its image in RTL layouts.

– **boolean isFilterBitmap()**: Returns whether this drawable is filtering bitmaps when scaled or rotated.

– **boolean isProjected()**: Returns true if the drawable is projected.

– **boolean isStateful()**: Returns whether this drawable changes appearance based on its state.

– **final boolean isVisible()**: Returns true if the drawable is currently visible.

– **void jumpToCurrentState()**: Skips any active state transition animations and jumps directly to the current state.

– **Drawable mutate()**: Returns a mutable instance of this drawable that can be modified independently.

– **boolean onLayoutDirectionChanged(int layoutDirection)**: Called when the drawable's layout direction changes.

– **static int resolveOpacity(int op1, int op2)**: Resolves and returns an appropriate opacity value from two input opacities.

– **void scheduleSelf(Runnable what, long when)**: Schedules the drawable to execute the given runnable at a specified time.

– **abstract void setAlpha(int alpha)**: Sets the drawable's alpha value for transparency.

– **void setAutoMirrored(boolean mirrored)**: Sets whether this drawable automatically mirrors when layout direction is RTL.

– **void setBounds(int left, int top, int right, int bottom)**: Defines the bounding rectangle of the drawable.

– **void setBounds(Rect bounds)**: Sets the drawable's bounds using the provided rectangle.

– **final void setCallback(Drawable.Callback cb)**: Binds a callback to the drawable for invalidation and scheduling.

– **void setChangingConfigurations(int configs)**: Specifies which configuration changes require the drawable to be recreated.

– **void setColorFilter(int color, PorterDuff.Mode mode)**: *Deprecated in API 29.* Use `setColorFilter(ColorFilter)` instead.

– **abstract void setColorFilter(ColorFilter colorFilter)**: Sets an optional color filter to modify how the drawable's pixels are rendered.

– **void setDither(boolean dither)**: *Deprecated in API 23.* This property is ignored.

– **void setFilterBitmap(boolean filter)**: Enables or disables bilinear filtering for scaled or rotated bitmaps.

– **void setHotspot(float x, float y)**: Specifies the hotspot's location within the drawable.

– **void setHotspotBounds(int left, int top, int right, int bottom)**: Defines the bounds for the hotspot within the drawable.

– **final boolean setLayoutDirection(int layoutDirection)**: Sets the layout direction for the drawable (LTR or RTL).

– **final boolean setLevel(int level)**: Sets the drawable's current level, used by certain drawable types for animation or progress indication.

– **boolean setState(int[] stateSet)**: Sets the drawable's current state using the given array of state attributes.

– **void setTint(int tintColor)**: Applies a single color tint to the drawable.

- **void setTintBlendMode(BlendMode blendMode)**: Specifies the blending mode used to apply the tint color.

- **void setTintList(ColorStateList tint)**: Applies a tint color list to the drawable for different states.

- **void setTintMode(PorterDuff.Mode tintMode)**: Specifies the blending mode used to apply the tint list.

- **boolean setVisible(boolean visible, boolean restart)**: Sets the drawable's visibility, optionally restarting animations.

- **void unscheduleSelf(Runnable what)**: Cancels any scheduled runnables associated with this drawable.

- **Protected methods**

  - **void onBoundsChange(Rect bounds)**: Override this in your subclass to change appearance if you vary based on the bounds.

  - **boolean onLevelChange(int level)**: Override this in your subclass to change appearance if you vary based on level.

  - **boolean onStateChange(int[] state)**: Override this in your subclass to change appearance if you recognize the specified state.

1.4.33 GradientDrawable

- **Hierarchy**

> java.lang.Object → android.graphics.drawable.Drawable →
> android.graphics.drawable.GradientDrawable

- **Include**

> o   android.graphics.drawable.GradientDrawable

- **Constructors**

> o   GradientDrawable()
> 1   GradientDrawable(GradientDrawable.Orientation orientation,
> ↪   int[] colors)

- **Public methods**

  - **void applyTheme(Resources.Theme t)**: Applies the specified theme to this `Drawable` and its children.
  - **boolean canApplyTheme()**: Returns true if this `Drawable` can apply a theme.
  - **void draw(Canvas canvas)**: Draws the shape within its bounds, respecting alpha and color filter effects.
  - **int getAlpha()**: Returns the current alpha value of the drawable.
  - **int getChangingConfigurations()**: Returns a mask of configuration parameters that can change, requiring the drawable to be recreated.
  - **ColorStateList getColor()**: Returns the color state list used to fill the shape, or `null` if it uses a gradient or no fill color.
  - **ColorFilter getColorFilter()**: Returns the currently applied color filter, or `null` if none.
  - **int[] getColors()**: Returns the colors used for the gradient fill, or `null` if not applicable.
  - **Drawable.ConstantState getConstantState()**: Returns the constant state shared by this drawable.
  - **float[] getCornerRadii()**: Returns the corner radii for all four corners.
  - **float getCornerRadius()**: Returns the uniform corner radius set with `setCornerRadius(float)`.
  - **float getGradientCenterX()**: Returns the X position of the gradient center as a fraction of the width.
  - **float getGradientCenterY()**: Returns the Y position of the gradient center as a fraction of the height.
  - **float getGradientRadius()**: Returns the gradient's radius in pixels.
  - **int getGradientType()**: Returns the gradient type: `LINEAR_GRADIENT`, `RADIAL_GRADIENT`, or `SWEEP_GRADIENT`.
  - **int getInnerRadius()**: Returns the inner radius of the ring shape in pixels.
  - **float getInnerRadiusRatio()**: Returns the inner radius of the ring as a ratio of the ring's width.

209

- **int getIntrinsicHeight()**: Returns the drawable's intrinsic height.
- **int getIntrinsicWidth()**: Returns the drawable's intrinsic width.
- **int getOpacity()**: *Deprecated.* No longer used for optimization.
- **Insets getOpticalInsets()**: Returns the suggested layout insets for alignment operations.
- **GradientDrawable.Orientation getOrientation()**: Returns the orientation of the gradient.
- **void getOutline(Outline outline)**: Populates the given `Outline` with the drawable's shape outline.
- **boolean getPadding(Rect padding)**: Returns padding in the given `Rect`, as suggested by the drawable.
- **int getShape()**: Returns the shape type (`LINE`, `OVAL`, `RECTANGLE`, or `RING`).
- **int getThickness()**: Returns the ring's thickness in pixels.
- **float getThicknessRatio()**: Returns the ring's thickness as a ratio of its width.
- **boolean getUseLevel()**: Returns whether the drawable's level property is used to scale the gradient.
- **boolean hasFocusStateSpecified()**: Returns true if the drawable explicitly defines a focused state.
- **void inflate(Resources r, XmlPullParser parser, AttributeSet attrs, Resources.Theme theme)**: Inflates the drawable from XML, optionally using a theme.
- **boolean isStateful()**: Returns true if the drawable changes appearance based on state.
- **Drawable mutate()**: Returns a mutable instance of this drawable that can be modified independently.
- **void setAlpha(int alpha)**: Sets the transparency level of the drawable.
- **void setColor(ColorStateList colorStateList)**: Changes the drawable to use a solid color state list instead of a gradient.
- **void setColor(int argb)**: Changes the drawable to use a single solid color.
- **void setColorFilter(ColorFilter colorFilter)**: Applies a color filter to the drawable.
- **void setColors(int[] colors, float[] offsets)**: Defines multiple colors and their relative positions in the gradient.
- **void setColors(int[] colors)**: Sets multiple colors for the gradient fill.
- **void setCornerRadii(float[] radii)**: Sets custom radii for each corner of the shape.
- **void setCornerRadius(float radius)**: Sets a uniform radius for all corners.
- **void setDither(boolean dither)**: *Deprecated.* Ignored property.
- **void setGradientCenter(float x, float y)**: Sets the center of the gradient as a fraction of width and height.
- **void setGradientRadius(float gradientRadius)**: Sets the gradient's radius in pixels.
- **void setGradientType(int gradient)**: Defines the gradient type (`LINEAR`, `RADIAL`, or `SWEEP`).
- **void setInnerRadius(int innerRadius)**: Sets the inner radius of a ring shape.

- **void setInnerRadiusRatio(float innerRadiusRatio)**: Defines the ring's inner radius as a ratio of its width.
- **void setOrientation(GradientDrawable.Orientation orientation)**: Sets the direction of the gradient.
- **void setPadding(int left, int top, int right, int bottom)**: Defines the padding of the shape.
- **void setShape(int shape)**: Sets the shape type (`LINE`, `OVAL`, `RECTANGLE`, `RING`).
- **void setSize(int width, int height)**: Defines the overall size of the shape.
- **void setStroke(int width, ColorStateList colorStateList)**: Sets stroke width and color using a color state list.
- **void setStroke(int width, ColorStateList colorStateList, float dashWidth, float dashGap)**: Sets stroke width, color, and dash pattern.
- **void setStroke(int width, int color, float dashWidth, float dashGap)**: Sets stroke width, solid color, and dash pattern.
- **void setStroke(int width, int color)**: Sets stroke width and solid color.
- **void setThickness(int thickness)**: Sets the thickness of the ring shape.
- **void setThicknessRatio(float thicknessRatio)**: Sets the ring thickness as a ratio of its width.
- **void setTintBlendMode(BlendMode blendMode)**: Specifies how tint color should be blended with the drawable.
- **void setTintList(ColorStateList tint)**: Applies a tint color list for different drawable states.
- **void setUseLevel(boolean useLevel)**: Configures whether the drawable's level affects the gradient scaling.

- **Protected methods**

  - **void onBoundsChange(Rect r)**: Override this in your subclass to change appearance if you vary based on the bounds.
  - **boolean onLevelChange(int level)**: Override this in your subclass to change appearance if you vary based on level.
  - **boolean onStateChange(int[] stateSet)**: Override this in your subclass to change appearance if you recognize the specified state.

- **Constants**

  - **int ARC**: Shape is an arc.
  - **int LINE**: Shape is a line
  - **int LINEAR_GRADIENT**: Gradient is linear (default.)
  - **int OVAL**: Shape is an ellipse
  - **int RADIAL_GRADIENT**: Gradient is circular.
  - **int RECTANGLE**: Shape is a rectangle, possibly with rounded corners
  - **int RING**: Shape is a ring.
  - **int SWEEP_GRADIENT**: Gradient is a sweep.

1.4.34 Intent

- **Hierarchy**

  java.lang.Object → android.content.Intent

- **Include**

```
0   android.content.Intent
```

- **Constructors**

```
0   Intent()
1   Intent(Context packageContext, Class<?> cls)
2   Intent(Intent o)
3   Intent(String action)
4   Intent(String action, Uri uri)
5   Intent(String action, Uri uri, Context packageContext,
    ↪  Class<?> cls)
```

- **Public methods**

  - **setAction(String)**: Sets the action to perform (like ACTION_VIEW, ACTION_SEND).
  - **getAction()**: Returns the currently set action.
  - **setData(Uri)**: Sets a URI for the intent (like content to view/open).
  - **getData()**: Gets the URI associated with the intent.
  - **setType(String)**: Sets the MIME type of data (like "image/*").
  - **getType()**: Gets the MIME type.
  - **setDataAndType(Uri, String)**: Sets both URI and MIME type at once.
  - **addCategory(String)**: Adds a category (e.g. CATEGORY_LAUNCHER).
  - **removeCategory(String)**: Removes a category.
  - **getCategories()**: Returns all categories added.
  - **putExtra(String, T)**: (overloads) Stores data in the intent (String, int, ArrayList, etc.).
  - **get*Extra(...)**: (e.g., getStringExtra) Retrieves specific extra values.
  - **getExtras()**: Gets entire Bundle of extras.
  - **replaceExtras(Bundle)**: Replaces all extras.
  - **removeExtra(String)**: Removes a specific extra.
  - **setClass(Context, Class<?>)**: Directs the Intent to a specific Activity class (explicit intent).
  - **setClassName(String, String)**: Sets a target component by package/class name.
  - **setComponent(ComponentName)**: Specifies the component directly.
  - **getComponent()**: Returns the component if explicitly set.
  - **setPackage(String)**: Limits resolution to a specific package.

- **addFlags(int)**: Adds flags like FLAG_ACTIVITY_NEW_TASK, controlling launch behavior.
- **setFlags(int)**: Replaces all existing flags.
- **getFlags()**: Returns currently-set flags.
- **resolveActivity(PackageManager)**: Checks if an intent can be handled by installed apps.
- **toUri(int)**: Converts the intent to a URI string.
- **filterEquals(Intent)**: Compares actions/data/categories/type but ignores extras.
- **getScheme()**: Returns URI scheme (like "content", "http").
- **getClipData()**: / setClipData() Used for advanced data transfers (drag-drop, multiple selections).

- **Fields**

  - **public static final Creator<Intent> CREATOR**:

- **Constants**

  - **ACTION_MAIN**: Entry point Activity (launcher screen)
  - **ACTION_VIEW**: Display data to the user (webpage, map, file)
  - **ACTION_EDIT**: Edit existing data
  - **ACTION_PICK**: Pick an item from data (gallery, contacts)
  - **ACTION_GET_CONTENT**: Allow the user to select a type of data (e.g., file picker)
  - **ACTION_CHOOSER**: Show app-chooser dialog for an Intent
  - **ACTION_SEND**: Share text, images, files
  - **ACTION_SENDTO**: Send to a specific communication target (email, SMS)
  - **EXTRA_TEXT**: Extra: text to send
  - **EXTRA_SUBJECT**: Extra: email subject
  - **EXTRA_STREAM**: Extra: send images/files
  - **EXTRA_EMAIL**: Extra: receiver email addresses
  - **ACTION_DIAL**: Open dialer with number prefilled
  - **ACTION_CALL**: Directly place a call (requires permission)
  - **ACTION_SENDSMS / ACTION_VIEW**: with SMS URI Send SMS message
  - **EXTRA_PHONE_NUMBER**: Extra: phone number
  - **ACTION_IMAGE_CAPTURE**: Open camera app to take a picture
  - **ACTION_VIDEO_CAPTURE**: Record video
  - **EXTRA_OUTPUT**: Where to save captured image/video
  - **ACTION_SETTINGS**: Open device settings screen
  - **ACTION_WIRELESS_SETTINGS**: Wireless settings
  - **ACTION_APPLICATION_DETAILS_SETTINGS**: App details settings (like uninstall/permissions page)
  - **FLAG_ACTIVITY_NEW_TASK**: Start Activity in a new task (important for services)

- **FLAG_ACTIVITY_CLEAR_TOP**: Clear Activities on top of target
- **FLAG_ACTIVITY_SINGLE_TOP**: Reuse existing instance if already on top
- **ACTION_AIRPLANE_MODE_CHANGED**: Airplane mode switch
- **ACTION_BATTERY_LOW**: Battery low warning
- **ACTION_BOOT_COMPLETED**: Device boot finished (permission required)

1.4.35 Animation

- **Hierarchy**

  java.lang.Object → android.view.animation.Animation

- **Include**

  ```
  o   android.view.animation.Animation
  ```

- **Constructors**

  ```
  o   Animation()
  1   Animation(Context context, AttributeSet attrs)
  ```

- **Public methods**

  – **cancel()**: Cancel the animation.
  – **computeDurationHint()**: Compute a hint at how long the entire animation may last, in milliseconds.
  – **getBackdropColor()**: Returns the background color to show behind the animating windows.
  – **getBackgroundColor()**: *Deprecated in API level 30.* None of window animations are running with background color.
  – **getDetachWallpaper()**: *Deprecated in API level 29.* All window animations are running with detached wallpaper.
  – **getDuration()**: How long this animation should last.
  – **getFillAfter()**: If fillAfter is true, this animation will apply its transformation after the end time.
  – **getFillBefore()**: If fillBefore is true, this animation will apply its transformation before the start time.
  – **getInterpolator()**: Gets the acceleration curve type for this animation.
  – **getRepeatCount()**: Defines how many times the animation should repeat.
  – **getRepeatMode()**: Defines what this animation should do when it reaches the end.
  – **getShowBackdrop()**: If true, animation will show the backdrop behind window animations.
  – **getStartOffset()**: When this animation should start relative to StartTime.
  – **getStartTime()**: When this animation should start.
  – **getTransformation(long, Transformation, float)**: Gets the transformation at a specified point in time.
  – **getTransformation(long, Transformation)**: Gets the transformation at a specified point in time.
  – **getZAdjustment()**: Returns the Z ordering mode while running the animation.
  – **hasEnded()**: Indicates whether this animation has ended.
  – **hasStarted()**: Indicates whether this animation has started.

- **initialize(int, int, int, int)**: Initialize animation with view and parent dimensions.
- **isFillEnabled()**: If true, animation applies the value of fillBefore.
- **isInitialized()**: Whether the animation has been initialized.
- **reset()**: Reset the initialization state of this animation.
- **restrictDuration(long)**: Ensure the duration is not longer than durationMillis.
- **scaleCurrentDuration(float)**: Scale the duration by the given factor.
- **setAnimationListener(Animation.AnimationListener)**: Bind an animation listener to this animation.
- **setBackdropColor(int)**: Set the backdrop color behind animating windows.
- **setBackgroundColor(int)**: *Deprecated in API level 30.*
- **setDetachWallpaper(boolean)**: *Deprecated in API level 29.*
- **setDuration(long)**: Set how long this animation should last.
- **setFillAfter(boolean)**: If true, the transformation persists after animation finishes.
- **setFillBefore(boolean)**: If true, apply transformation before animation start.
- **setFillEnabled(boolean)**: If true, animation applies the value of fillBefore.
- **setInterpolator(Interpolator)**: Sets the acceleration curve for the animation.
- **setInterpolator(Context, int)**: Sets acceleration curve using resources.
- **setRepeatCount(int)**: Sets how many times animation repeats.
- **setRepeatMode(int)**: Defines behavior when animation reaches the end.
- **setShowBackdrop(boolean)**: Enable backdrop animation behind windows.
- **setStartOffset(long)**: When animation should start relative to start time.
- **setStartTime(long)**: When animation should start.
- **setZAdjustment(int)**: Set Z ordering mode while running the animation.
- **start()**: Start the animation the first time getTransformation is invoked.
- **startNow()**: Start the animation at the current system time.
- **willChangeBounds()**: Indicates if animation affects bounds of animated view.
- **willChangeTransformationMatrix()**: Indicates if animation affects transformation matrix.

- **Protected methods**

  - **applyTransformation(float interpolatedTime, Transformation t)**: Helper for getTransformation.
  - **clone()**: Creates and returns a copy of this object.
  - **ensureInterpolator()**: Guarantees that this animation has an interpolator.
  - **finalize()**: Called by the garbage collector when there are no more references to the object.
  - **getScaleFactor()**: Returns the scale factor set by the call to getTransformation.
  - **resolveSize(int type, float value, int size, int parentSize)**: Convert size description to an actual dimension.

- **Constants**

  - **applyTransformation(float interpolatedTime, Transformation t)**: Helper for getTransformation.

- **clone()**: Creates and returns a copy of this object.
- **ensureInterpolator()**: Guarantees that this animation has an interpolator.
- **finalize()**: Called by the garbage collector when there are no more references to the object.
- **getScaleFactor()**: Returns the scale factor set by the call to getTransformation.
- **resolveSize(int type, float value, int size, int parentSize)**: Convert size description to an actual dimension.

1.4.36 AnimationSet

- **Hierarchy**

  java.lang.Object → android.view.animation.Animation →
  android.view.animation.AnimationSet

- **Include**

  ```
  0   android.view.animation.AnimationSet
  ```

- **Constructors**

  ```
  0   AnimationSet(Context context, AttributeSet attrs)
  1   AnimationSet(boolean shareInterpolator)
  ```

- **Public methods**

  - **addAnimation(Animation a)**: Add a child animation to this animation set.
  - **computeDurationHint()**: Duration hint is the maximum duration hint of all child animations.
  - **getAnimations()**: Returns the list of child animations.
  - **getDuration()**: Duration is defined as the duration of the longest child animation.
  - **getStartTime()**: When this animation should start.
  - **getTransformation(long currentTime, Transformation t)**: Transformation is the concatenation of all component animations.
  - **initialize(int width, int height, int parentWidth, int parentHeight)**: Initialize with the dimensions of the object and its parent.
  - **reset()**: Reset the initialization state of this animation.
  - **restrictDuration(long durationMillis)**: Ensure duration does not exceed durationMillis.
  - **scaleCurrentDuration(float scale)**: Scale the duration by the specified factor.
  - **setDuration(long durationMillis)**: Sets the duration for every child animation.
  - **setFillAfter(boolean fillAfter)**: If true, transformation persists after animation ends.
  - **setFillBefore(boolean fillBefore)**: If true, apply transformation before animation starts.
  - **setRepeatMode(int repeatMode)**: Defines what happens when animation reaches the end.
  - **setStartOffset(long startOffset)**: When this animation should start relative to start time.
  - **setStartTime(long startTimeMillis)**: Sets start time for this animation and all child animations.
  - **willChangeBounds()**: Indicates whether this animation affects bounds of the animated view.

218

– **willChangeTransformationMatrix()**: Indicates whether this animation affects the transformation matrix.

- **Protected methods**

    – **AnimationSet clone()**: Creates and returns a copy of this object.

1.4.37 AlphaAnimation

- **Hierarchy**

  java.lang.Object → android.view.animation.Animation →
  android.view.animation.AlphaAnimation

- **Include**

  ```
  o   android.view.animation.AlphaAnimation
  ```

- **Constructors**

  ```
  o   AlphaAnimation(Context context, AttributeSet attrs)
  1   AlphaAnimation(float fromAlpha, float toAlpha)
  ```

- **Public methods**

  - **boolean willChangeBounds()**: Indicates whether or not this animation will affect the bounds of the animated view.
  - **boolean willChangeTransformationMatrix()**: Indicates whether or not this animation will affect the transformation matrix.

- **Protected methods**

  - **void applyTransformation(float interpolatedTime, Transformation t)**: Changes the alpha property of the supplied Transformation

1.4.38 RotateAnimation

- **Hierarchy**

  java.lang.Object → android.view.animation.Animation →
  android.view.animation.RotateAnimation

- **Include**

  ```
  0   android.view.animation.RotateAnimation
  ```

- **Constructors**

  ```
  0   RotateAnimation(Context context, AttributeSet attrs)
  1   RotateAnimation(float fromDegrees, float toDegrees)
  2   RotateAnimation(float fromDegrees, float toDegrees, float
  ↪    pivotX, float pivotY)
  3   RotateAnimation(float fromDegrees, float toDegrees, int
  ↪    pivotXType, float pivotXValue, int pivotYType, float
  ↪    pivotYValue)
  ```

- **Public methods**

  - **void initialize(int width, int height, int parentWidth, int parentHeight)**:
    Initialize this animation with the dimensions of the object being animated as well
    as the objects parents.

- **Protected methods**

  - **void applyTransformation(float interpolatedTime, Transformation t)**:
    Helper for getTransformation.

1.4.39 ScaleAnimation

- **Hierarchy**

  java.lang.Object → android.view.animation.Animation →
  android.view.animation.ScaleAnimation

- **Include**

  ```
  0  android.view.animation.ScaleAnimation
  ```

- **Constructors**

  ```
  0  ScaleAnimation(Context context, AttributeSet attrs)
  1  ScaleAnimation(float fromX, float toX, float fromY, float
  ↪  toY)
  2  ScaleAnimation(float fromX, float toX, float fromY, float
  ↪  toY, float pivotX, float pivotY)
  3  ScaleAnimation(float fromX, float toX, float fromY, float
  ↪  toY, int pivotXType, float pivotXValue, int pivotYType,
  ↪  float pivotYValue)
  ```

- **Public methods**

  - **void initialize(int width, int height, int parentWidth, int parentHeight)**:
    Initialize this animation with the dimensions of the object being animated as well
    as the objects parents.

- **Protected methods**

  - **void applyTransformation(float interpolatedTime, Transformation t)**:
    Helper for getTransformation.

1.4.40   TranslateAnimation

- **Hierarchy**

  java.lang.Object → android.view.animation.Animation →
  android.view.animation.TranslateAnimation

- **Include**

  ```
  0   android.view.animation.TranslateAnimation
  ```

- **Constructors**

  ```
  0   TranslateAnimation(Context context, AttributeSet attrs)
  1   TranslateAnimation(float fromXDelta, float toXDelta, float
  ↪    fromYDelta, float toYDelta)
  2   TranslateAnimation(int fromXType, float fromXValue, int
  ↪    toXType, float toXValue, int fromYType, float
  ↪    fromYValue, int toYType, float toYValue)
  ```

- **Public methods**

  – **void initialize(int width, int height, int parentWidth, int parentHeight)**:
    Initialize this animation with the dimensions of the object being animated as well
    as the objects parents.

- **Protected methods**

  – **void applyTransformation(float interpolatedTime, Transformation t)**:
    Helper for getTransformation.

1.4.41 PreferenceManager

- **Include**:

  > o `android.`preference`.PreferenceManager`

- **Nested interfaces**

  - **interface PreferenceManager.OnActivityDestroyListener**: This interface was deprecated in API level 29. Use the AndroidX Preference Library for consistent behavior across all devices. For more information on using the AndroidX Preference Library see Settings.
  - **interface PreferenceManager.OnActivityResultListener**: This interface was deprecated in API level 29. Use the AndroidX Preference Library for consistent behavior across all devices. For more information on using the AndroidX Preference Library see Settings.
  - **interface PreferenceManager.OnActivityStopListener**: This interface was deprecated in API level 29. Use the AndroidX Preference Library for consistent behavior across all devices. For more information on using the AndroidX Preference Library see Settings.

- **Public methods**

  - **PreferenceScreen createPreferenceScreen(Context context)**:
  - **Preference findPreference(CharSequence key)**: Finds a Preference based on its key.
  - **static SharedPreferences getDefaultSharedPreferences(Context context)**: Gets a SharedPreferences instance that points to the default file that is used by the preference framework in the given context.
  - **static String getDefaultSharedPreferencesName(Context context)**: Returns the name used for storing default shared preferences.
  - **PreferenceDataStore getPreferenceDataStore()**: Returns the PreferenceDataStore associated with this manager or null if the default SharedPreferences are used instead.
  - **SharedPreferences getSharedPreferences()**: Gets a SharedPreferences instance that preferences managed by this will use.
  - **int getSharedPreferencesMode()**: Returns the current mode of the SharedPreferences file that preferences managed by this will use.
  - **String getSharedPreferencesName()**: Returns the current name of the SharedPreferences file that preferences managed by this will use.
  - **boolean isStorageDefault()**: Indicates if the storage location used internally by this class is the default provided by the hosting Context.
  - **boolean isStorageDeviceProtected()**: Indicates if the storage location used internally by this class is backed by device-protected storage.
  - **static void setDefaultValues(Context context, String sharedPreferencesName, int sharedPreferencesMode, int resId, boolean readAgain)**: Similar to setDefaultValues(android.content.Context, int, boolean) but allows the client to provide the filename and mode of the shared preferences file.
  - **static void setDefaultValues(Context context, int resId, boolean readAgain)**: Sets the default values from an XML preference file by reading the values defined by each Preference item's android:defaultValue attribute.

- **void setPreferenceDataStore(PreferenceDataStore dataStore)**: Sets a PreferenceDataStore to be used by all Preferences associated with this manager that don't have a custom PreferenceDataStore assigned via Preference.setPreferenceDataStore(PreferenceDataStore).

- **void setSharedPreferencesMode(int sharedPreferencesMode)**: Sets the mode of the SharedPreferences file that preferences managed by this will use.

- **void setSharedPreferencesName(String sharedPreferencesName)**: Sets the name of the SharedPreferences file that preferences managed by this will use.

- **void setStorageDefault()**: Sets the storage location used internally by this class to be the default provided by the hosting Context.

- **void setStorageDeviceProtected()**: Explicitly set the storage location used internally by this class to be device-protected storage.

- **Constants**

  - **String KEY_HAS_SET_DEFAULT_VALUES**:

  - **String METADATA_KEY_PREFERENCES**: The Activity meta-data key for its XML preference hierarchy.

1.4.42 SharedPreferences (interface)

- **Include**

  ```
  o   android.content.SharedPreferences
  ```

- **Nested interfaces**

  - **interface SharedPreferences.Editor**: Interface used for modifying values in a SharedPreferences object.
  - **interface SharedPreferences.OnSharedPreferenceChangeListener**: Interface definition for a callback to be invoked when a shared preference is changed.

- **Abstract methods**

  - **abstract boolean contains(String key)**: Checks whether the preferences contains a preference.
  - **abstract SharedPreferences.Editor edit()**: Create a new Editor for these preferences, through which you can make modifications to the data in the preferences and atomically commit those changes back to the SharedPreferences object.
  - **abstract Map<String, ?> getAll()**: Retrieve all values from the preferences.
  - **abstract boolean getBoolean(String key, boolean defValue)**: Retrieve a boolean value from the preferences.
  - **abstract float getFloat(String key, float defValue)**: Retrieve a float value from the preferences.
  - **abstract int getInt(String key, int defValue)**: Retrieve an int value from the preferences.
  - **abstract long getLong(String key, long defValue)**: Retrieve a long value from the preferences.
  - **abstract String getString(String key, String defValue)**: Retrieve a String value from the preferences.
  - **abstract Set<String> getStringSet(String key, Set<String> defValues)**: Retrieve a set of String values from the preferences.
  - **abstract void registerOnSharedPreferenceChangeListener(SharedPreferences.OnSharedPreferenceChangeListener listener)**: Registers a callback to be invoked when a change happens to a preference.
  - **abstract void unregisterOnSharedPreferenceChangeListener(SharedPreferences.OnSharedPreferenceChangeListener listener)**: Unregisters a previous callback.

1.4.43 SharedPreferences.Editor (Interface)

- **Description**: Interface used for modifying values in a SharedPreferences object. All changes you make in an editor are batched, and not copied back to the original Shared-Preferences until you call commit() or apply()

- **Abstract methods**:

    - **abstract void apply()**: Commit your preferences changes back from this Editor to the SharedPreferences object it is editing.
    - **abstract SharedPreferences.Editor clear()**: Mark in the editor to remove all values from the preferences.
    - **abstract boolean commit()**: Commit your preferences changes back from this Editor to the SharedPreferences object it is editing.
    - **abstract SharedPreferences.Editor putBoolean(String key, boolean value)**: Set a boolean value in the preferences editor, to be written back once commit() or apply() are called.
    - **abstract SharedPreferences.Editor putFloat(String key, float value)**: Set a float value in the preferences editor, to be written back once commit() or apply() are called.
    - **abstract SharedPreferences.Editor putInt(String key, int value)**: Set an int value in the preferences editor, to be written back once commit() or apply() are called.
    - **abstract SharedPreferences.Editor putLong(String key, long value)**: Set a long value in the preferences editor, to be written back once commit() or apply() are called.
    - **abstract SharedPreferences.Editor putString(String key, String value)**: Set a String value in the preferences editor, to be written back once commit() or apply() are called.
    - **abstract SharedPreferences.Editor putStringSet(String key, Set<String> values)**: Set a set of String values in the preferences editor, to be written back once commit() or apply() is called.
    - **abstract SharedPreferences.Editor remove(String key)**: Mark in the editor that a preference value should be removed, which will be done in the actual preferences once commit() is called.

1.4.44 Menu (Interface)

- **Include**

  o   `android.view.Menu`

- **Public abstract methods**

  – **abstract MenuItem add(int groupId, int itemId, int order, CharSequence title)**: Add a new item to the menu.

  – **abstract MenuItem add(int titleRes)**: Add a new item to the menu.

  – **abstract MenuItem add(CharSequence title)**: Add a new item to the menu.

  – **abstract MenuItem add(int groupId, int itemId, int order, int titleRes)**: Variation on add(int, int, int, java.lang.CharSequence) that takes a string resource identifier instead of the string itself.

  – **abstract int addIntentOptions(int groupId, int itemId, int order, ComponentName caller, Intent[] specifics, Intent intent, int flags, MenuItem[] outSpecificItems)**: Add a group of menu items corresponding to actions that can be performed for a particular Intent.

  – **abstract SubMenu addSubMenu(CharSequence title)**: Add a new sub-menu to the menu.

  – **abstract SubMenu addSubMenu(int groupId, int itemId, int order, int titleRes)**: Variation on addSubMenu(int, int, int, java.lang.CharSequence) that takes a string resource identifier for the title instead of the string itself.

  – **abstract SubMenu addSubMenu(int groupId, int itemId, int order, CharSequence title)**: Add a new sub-menu to the menu.

  – **abstract SubMenu addSubMenu(int titleRes)**: Add a new sub-menu to the menu.

  – **abstract void clear()**: Remove all existing items from the menu, leaving it empty as if it had just been created.

  – **abstract void close()**: Closes the menu, if open.

  – **abstract MenuItem findItem(int id)**: Return the menu item with a particular identifier.

  – **abstract MenuItem getItem(int index)**: Gets the menu item at the given index.

  – **abstract boolean hasVisibleItems()**: Return whether the menu currently has item items that are visible.

  – **abstract boolean isShortcutKey(int keyCode, KeyEvent event)**: Is a keypress one of the defined shortcut keys for this window.

  – **abstract boolean performIdentifierAction(int id, int flags)**: Execute the menu item action associated with the given menu identifier.

  – **abstract boolean performShortcut(int keyCode, KeyEvent event, int flags)**: Execute the menu item action associated with the given shortcut character.

  – **abstract void removeGroup(int groupId)**: Remove all items in the given group.

  – **abstract void removeItem(int id)**: Remove the item with the given identifier.

- **abstract void setGroupCheckable(int group, boolean checkable, boolean exclusive)**: Control whether a particular group of items can show a check mark.
- **default void setGroupDividerEnabled(boolean groupDividerEnabled)**: Enable or disable the group dividers.
- **abstract void setGroupEnabled(int group, boolean enabled)**: Enable or disable all menu items that are in the given group.
- **abstract void setGroupVisible(int group, boolean visible)**: Show or hide all menu items that are in the given group.
- **abstract void setQwertyMode(boolean isQwerty)**: Control whether the menu should be running in qwerty mode (alphabetic shortcuts) or 12-key mode (numeric shortcuts).
- **abstract int size()**: Get the number of items in the menu.

- **Constants**

  - **int CATEGORY_ALTERNATIVE**: Category code for the order integer for items/groups that are alternative actions on the data that is currently displayed – or/add this with your base value.
  - **int CATEGORY_CONTAINER**: Category code for the order integer for items/groups that are part of a container – or/add this with your base value.
  - **int CATEGORY_SECONDARY**: Category code for the order integer for items/groups that are user-supplied secondary (infrequently used) options – or/add this with your base value.
  - **int CATEGORY_SYSTEM**: Category code for the order integer for items/groups that are provided by the system – or/add this with your base value.
  - **int FIRST**: First value for group and item identifier integers.
  - **int FLAG_ALWAYS_PERFORM_CLOSE**: Flag for performShortcut(int, android.view.KeyEvent, int): if set, always close the menu after executing the shortcut.
  - **int FLAG_APPEND_TO_GROUP**: Flag for addIntentOptions(int, int, int, ComponentName, Intent, Intent, int, MenuItem): if set, do not automatically remove any existing menu items in the same group.
  - **int FLAG_PERFORM_NO_CLOSE**: Flag for performShortcut(int, KeyEvent, int): if set, do not close the menu after executing the shortcut.
  - **int NONE**: Value to use for group and item identifier integers when you don't care about them.
  - **int SUPPORTED_MODIFIERS_MASK**: A mask of all supported modifiers for MenuItem's keyboard shortcuts

1.4.45 MenuItem (Interface)

– **Include**

o `android.view.MenuItem`

– **Public abstract methods**:
  * **abstract boolean collapseActionView()**: Collapse the action view associated with this menu item.
  * **abstract boolean expandActionView()**: Expand the action view associated with this menu item.
  * **abstract ActionProvider getActionProvider()**: Gets the ActionProvider.
  * **abstract View getActionView()**: Returns the currently set action view for this menu item.
  * **default int getAlphabeticModifiers()**: Return the modifier for this menu item's alphabetic shortcut.
  * **abstract char getAlphabeticShortcut()**: Return the char for this menu item's alphabetic shortcut.
  * **default CharSequence getContentDescription()**: Retrieve the content description associated with this menu item.
  * **abstract int getGroupId()**: Return the group identifier that this menu item is part of.
  * **abstract Drawable getIcon()**: Returns the icon for this item as a Drawable (getting it from resources if it hasn't been loaded before).
  * **default BlendMode getIconTintBlendMode()**: Returns the blending mode used to apply the tint to this item's icon, if specified.
  * **default ColorStateList getIconTintList()**: default PorterDuff.Mode getIconTintMode()
  * **abstract Intent getIntent()**: Return the Intent associated with this item.
  * **abstract int getItemId()**: Return the identifier for this menu item.
  * **abstract ContextMenu.ContextMenuInfo getMenuInfo()**: Gets the extra information linked to this menu item.
  * **default int getNumericModifiers()**: Return the modifiers for this menu item's numeric (12-key) shortcut.
  * **abstract char getNumericShortcut()**: Return the char for this menu item's numeric (12-key) shortcut.
  * **abstract int getOrder()**: Return the category and order within the category of this item.
  * **abstract SubMenu getSubMenu()**: Get the sub-menu to be invoked when this item is selected, if it has one.
  * **abstract CharSequence getTitle()**: Retrieve the current title of the item.
  * **abstract CharSequence getTitleCondensed()**: Retrieve the current condensed title of the item.
  * **default CharSequence getTooltipText()**: Retrieve the tooltip text associated with this menu item.
  * **abstract boolean hasSubMenu()**: Check whether this item has an associated sub-menu.
  * **abstract boolean isActionViewExpanded()**: Returns true if this menu item's action view has been expanded.

∗ **abstract boolean isCheckable()**: Return whether the item can currently display a check mark.

∗ **abstract boolean isChecked()**: Return whether the item is currently displaying a check mark.

∗ **abstract boolean isEnabled()**: Return the enabled state of the menu item.

∗ **abstract boolean isVisible()**: Return the visibility of the menu item.

∗ **abstract MenuItem setActionProvider(ActionProvider actionProvider)**: Sets the ActionProvider responsible for creating an action view if the item is placed on the action bar.

∗ **abstract MenuItem setActionView(int resId)**: Set an action view for this menu item.

∗ **abstract MenuItem setActionView(View view)**: Set an action view for this menu item.

∗ **abstract MenuItem setAlphabeticShortcut(char alphaChar)**: Change the alphabetic shortcut associated with this item.

∗ **default MenuItem setAlphabeticShortcut(char alphaChar, int alphaModifiers)**: Change the alphabetic shortcut associated with this item.

∗ **abstract MenuItem setCheckable(boolean checkable)**: Control whether this item can display a check mark.

∗ **abstract MenuItem setChecked(boolean checked)**: Control whether this item is shown with a check mark.

∗ **default MenuItem setContentDescription(CharSequence contentDescription)**: Change the content description associated with this menu item.

∗ **abstract MenuItem setEnabled(boolean enabled)**: Sets whether the menu item is enabled.

∗ **abstract MenuItem setIcon(Drawable icon)**: Change the icon associated with this item.

∗ **abstract MenuItem setIcon(int iconRes)**: Change the icon associated with this item.

∗ **default MenuItem setIconTintBlendMode(BlendMode blendMode)**: Specifies the blending mode used to apply the tint specified by setIconTintList(android.content.res.ColorStateList) to this item's icon.

∗ **default MenuItem setIconTintList(ColorStateList tint)**: Applies a tint to this item's icon.

∗ **default MenuItem setIconTintMode(PorterDuff.Mode tintMode)**: Specifies the blending mode used to apply the tint specified by setIconTintList(android.content.res.ColorStateList) to this item's icon.

∗ **abstract MenuItem setIntent(Intent intent)**: Change the Intent associated with this item.

∗ **default MenuItem setNumericShortcut(char numericChar, int numericModifiers)**: Change the numeric shortcut and modifiers associated with this item.

∗ **abstract MenuItem setNumericShortcut(char numericChar)**: Change the numeric shortcut associated with this item.

∗ **abstract MenuItem setOnActionExpandListener(MenuItem.OnActionExpandListener listener)**: Set an OnActionExpandListener on this menu item to be notified when the associated action view is expanded or collapsed.

* **abstract MenuItem setOnMenuItemClickListener(MenuItem.On-MenuItemClickListener menuItemClickListener)**: Set a custom listener for invocation of this menu item.
* **abstract MenuItem setShortcut(char numericChar, char alphaChar)**: Change both the numeric and alphabetic shortcut associated with this item.
* **default MenuItem setShortcut(char numericChar, char alphaChar, int numericModifiers, int alphaModifiers)**: Change both the numeric and alphabetic shortcut associated with this item.
* **abstract void setShowAsAction(int actionEnum)**: Sets how this item should display in the presence of an Action Bar.
* **abstract MenuItem setShowAsActionFlags(int actionEnum)**: Sets how this item should display in the presence of an Action Bar.
* **abstract MenuItem setTitle(CharSequence title)**: Change the title associated with this item.
* **abstract MenuItem setTitle(int title)**: Change the title associated with this item.
* **abstract MenuItem setTitleCondensed(CharSequence title)**: Change the condensed title associated with this item.
* **default MenuItem setTooltipText(CharSequence tooltipText)**: Change the tooltip text associated with this menu item.
* **abstract MenuItem setVisible(boolean visible)**: Sets the visibility of the menu item.

  – **Constants**

  * **int SHOW_AS_ACTION_ALWAYS**: Always show this item as a button in an Action Bar.
  * **int SHOW_AS_ACTION_COLLAPSE_ACTION_VIEW**: This item's action view collapses to a normal menu item.
  * **int SHOW_AS_ACTION_IF_ROOM**: Show this item as a button in an Action Bar if the system decides there is room for it.
  * **int SHOW_AS_ACTION_NEVER**: Never show this item as a button in an Action Bar.
  * **int SHOW_AS_ACTION_WITH_TEXT**: When this item is in the action bar, always show it with a text label even if it also has an icon specified.

1.4.46   ContextMenu (interface)

  – **Hierarchy**

android.view.Menu → android.view.ContextMenu

  – **Include**

> o   android.`view`.`ContextMenu`

  – **Public abstract methods**
      * **abstract void clearHeader()**: Clears the header of the context menu.
      * **abstract ContextMenu setHeaderIcon(int iconRes)**: Sets the context menu header's icon to the icon given in iconRes resource id.
      * **abstract ContextMenu setHeaderIcon(Drawable icon)**: Sets the context menu header's icon to the icon given in icon Drawable.
      * **abstract ContextMenu setHeaderTitle(int titleRes)**: Sets the context menu header's title to the title given in titleRes resource identifier.
      * **abstract ContextMenu setHeaderTitle(CharSequence title)**: Sets the context menu header's title to the title given in title.
      * **abstract ContextMenu setHeaderView(View view)**: Sets the header of the context menu to the View given in view.

1.4.47 PopupMenu

– **Hierarchy**

java.lang.Object → android.widget.PopupMenu

– **Include**

```
o   android.widget.PopupMenu
```

– **Constructors**

```
o   PopupMenu(Context context, View anchor)
1   PopupMenu(Context context, View anchor, int gravity)
2   PopupMenu(Context context, View anchor, int gravity,
    ↪   int popupStyleAttr, int popupStyleRes)
```

– **Public methods**
   * **void dismiss()**: Dismiss the menu popup.
   * **View.OnTouchListener getDragToOpenListener()**: Returns an On-TouchListener that can be added to the anchor view to implement drag-to-open behavior.
   * **int getGravity()**:
   * **Menu getMenu()**: Returns the Menu associated with this popup.
   * **MenuInflater getMenuInflater()**:
   * **void inflate(int menuRes)**: Inflate a menu resource into this PopupMenu.
   * **void setForceShowIcon(boolean forceShowIcon)**: Sets whether the popup menu's adapter is forced to show icons in the menu item views.
   * **void setGravity(int gravity)**: Sets the gravity used to align the popup window to its anchor view.
   * **void setOnDismissListener(PopupMenu.OnDismissListener listener)**: Sets a listener that will be notified when this menu is dismissed.
   * **void setOnMenuItemClickListener(PopupMenu.OnMenuItemClick-Listener listener)**: Sets a listener that will be notified when the user selects an item from the menu.
   * **void show()**: Show the menu popup anchored to the view specified during construction.

1.4.48 SubMenu (interface)

  – Signature

> o `public interface SubMenu implements Menu`

  – Include

> o `android.view.SubMenu`

  – Public abstract methods

  * **abstract void clearHeader()**: Clears the header of the submenu.
  * **abstract MenuItem getItem()**: Gets the MenuItem that represents this submenu in the parent menu.
  * **abstract SubMenu setHeaderIcon(int iconRes)**: Sets the submenu header's icon to the icon given in iconRes resource id.
  * **abstract SubMenu setHeaderIcon(Drawable icon)**: Sets the submenu header's icon to the icon given in icon Drawable.
  * **abstract SubMenu setHeaderTitle(int titleRes)**: Sets the submenu header's title to the title given in titleRes resource identifier.
  * **abstract SubMenu setHeaderTitle(CharSequence title)**: Sets the submenu header's title to the title given in title.
  * **abstract SubMenu setHeaderView(View view)**: Sets the header of the submenu to the View given in view.
  * **abstract SubMenu setIcon(Drawable icon)**: Change the icon associated with this submenu's item in its parent menu.
  * **abstract SubMenu setIcon(int iconRes)**: Change the icon associated with this submenu's item in its parent menu.

1.4.49 MenuInflator

– **Hierarchy**

java.lang.Object → android.view.MenuInflater

– **Include**

> o   android.view.MenuInflater

– **Constructors**

> o   MenuInflater(Context context)

– **Public methods**
  * **void inflate(int menuRes, Menu menu)**: Inflate a menu hierarchy from the specified XML resource.

1.4.50 Toast

– **Hierarchy**

java.lang.Object → android.widget.Toast

– **Include**

> o android.`widget`.`Toast`

– **Constructors**

> o `Toast(Context context)`

– **Public methods**
  * **void addCallback(Toast.Callback callback)**: Adds a callback to be notified when the toast is shown or hidden.
  * **void cancel()**: Close the view if it's showing, or don't show it if it isn't showing yet.
  * **int getDuration()**: Return the duration.
  * **int getGravity()**: Get the location at which the notification should appear on the screen.
  * **float getHorizontalMargin()**: Return the horizontal margin.
  * **float getVerticalMargin()**: Return the vertical margin.
  * **View getView()**: This method was deprecated in API level 30. Custom toast views are deprecated. Apps can create a standard text toast with the makeText(android.content.Context, java.lang.CharSequence, int) method, or use a Snackbar when in the foreground. Starting from Android Build.VERSION_CODES.R, apps targeting API level Build.VERSION_CODES.R or higher that are in the background will not have custom toast views displayed.
  * **int getXOffset()**: Return the X offset in pixels to apply to the gravity's location.
  * **int getYOffset()**: Return the Y offset in pixels to apply to the gravity's location.
  * **static Toast makeText(Context context, int resId, int duration)**: Make a standard toast that just contains text from a resource.
  * **static Toast makeText(Context context, CharSequence text, int duration)**: Make a standard toast that just contains text.
  * **void removeCallback(Toast.Callback callback)**: Removes a callback previously added with addCallback(android.widget.Toast.Callback).
  * **void setDuration(int duration)**: Set how long to show the view for.
  * **void setGravity(int gravity, int xOffset, int yOffset)**: Set the location at which the notification should appear on the screen.
  * **void setMargin(float horizontalMargin, float verticalMargin)**: Set the margins of the view.
  * **void setText(int resId)**: Update the text in a Toast that was previously created using one of the makeText() methods.
  * **void setText(CharSequence s)**: Update the text in a Toast that was previously created using one of the makeText() methods.

* **void setView(View view)**: This method was deprecated in API level 30. Custom toast views are deprecated. Apps can create a standard text toast with the makeText(android.content.Context, java.lang.CharSequence, int) method, or use a Snackbar when in the foreground. Starting from Android Build.VERSION_CODES.R, apps targeting API level Build.VERSION_CODES.R or higher that are in the background will not have custom toast views displayed.
* **void show()**: Show the view for the specified duration.

– **Constants**

* **int LENGTH_LONG**: Show the view or text notification for a long period of time.
* **int LENGTH_SHORT**: Show the view or text notification for a short period of time.

1.4.51 LayoutInflator (Abstract class)

– **Hierarchy**

java.lang.Object → android.view.LayoutInflater

– **Include**

> o android.view.LayoutInflater

– **Constructors (Protected)**

> o LayoutInflater(Context context)
> 1 LayoutInflater(LayoutInflater original, Context
> ↪ newContext)

– **Public methods**
  * **abstract LayoutInflater cloneInContext(Context newContext)**: Create a copy of the existing LayoutInflater object, with the copy pointing to a different Context than the original.
  * **final View createView(Context viewContext, String name, String prefix, AttributeSet attrs)**: Low-level function for instantiating a view by name.
  * **final View createView(String name, String prefix, AttributeSet attrs)**: Low-level function for instantiating a view by name.
  * **static LayoutInflater from(Context context)**: Obtains the LayoutInflater from the given context.
  * **Context getContext()**: Return the context we are running in, for access to resources, class loader, etc.
  * **final LayoutInflater.Factory getFactory()**: Return the current Factory (or null).
  * **final LayoutInflater.Factory2 getFactory2()**: Return the current Factory2.
  * **LayoutInflater.Filter getFilter()**:
  * **View inflate(int resource, ViewGroup root)**: Inflate a new view hierarchy from the specified xml resource.
  * **View inflate(XmlPullParser parser, ViewGroup root)**: Inflate a new view hierarchy from the specified xml node.
  * **View inflate(XmlPullParser parser, ViewGroup root, boolean attachToRoot)**: Inflate a new view hierarchy from the specified XML node.
  * **View inflate(int resource, ViewGroup root, boolean attachToRoot)**: Inflate a new view hierarchy from the specified xml resource.
  * **View onCreateView(Context viewContext, View parent, String name, AttributeSet attrs)**: Version of onCreateView(android.view.View, java.lang.String, android.util.AttributeSet) that also takes the inflation context.
  * **void setFactory(LayoutInflater.Factory factory)**: Attach a custom Factory interface for creating views while using this LayoutInflater.
  * **void setFactory2(LayoutInflater.Factory2 factory)**: Like setFactory(Factory), but allows you to set a Factory2 interface.
  * **void setFilter(LayoutInflater.Filter filter)**: Sets the Filter to by this LayoutInflater.

– **Protected methods**

  ∗ **View onCreateView(View parent, String name, AttributeSet attrs)**: Version of onCreateView(java.lang.String, android.util.AttributeSet) that also takes the future parent of the view being constructed.

  ∗ **View onCreateView(String name, AttributeSet attrs)**: This routine is responsible for creating the correct subclass of View given the xml element name.

1.4.52 SQLiteDatabase

– **Hierarchy**

> java.lang.Object → android.database.sqlite.SQLiteClosable →
> android.database.sqlite.SQLiteDatabase

– **Include**

> o   android.`database`.`sqlite`.`SQLiteDatabase`

– **Public methods**
  * **void beginTransaction()**: Begins a transaction in EXCLUSIVE mode.
  * **void beginTransactionNonExclusive()**: Begins a transaction in IMME-DIATE mode.
  * **void beginTransactionReadOnly()**: Begins a transaction in DEFERRED mode, with the android-specific constraint that the transaction is read-only.
  * **void beginTransactionWithListener(SQLiteTransactionListener transactionListener)**: Begins a transaction in EXCLUSIVE mode.
  * **void beginTransactionWithListenerNonExclusive(SQLiteTransactionListener transactionListener)**: Begins a transaction in IMMEDIATE mode.
  * **void beginTransactionWithListenerReadOnly(SQLiteTransactionListener transactionListener)**: Begins a transaction in read-only mode with a SQLiteTransactionListener listener.
  * **SQLiteStatement compileStatement(String sql)**: Compiles an SQL statement into a reusable pre-compiled statement object.
  * **static SQLiteDatabase create(SQLiteDatabase.CursorFactory factory)**: Create a memory backed SQLite database.
  * **static SQLiteDatabase createInMemory(SQLiteDatabase.OpenParams openParams)**: Create a memory backed SQLite database.
  * **SQLiteRawStatement createRawStatement(String sql)**: Return a SQLiteRawStatement connected to the database.
  * **int delete(String table, String whereClause, String[] whereArgs)**: Convenience method for deleting rows in the database.
  * **static boolean deleteDatabase(File file)**: Deletes a database including its journal file and other auxiliary files that may have been created by the database engine.
  * **void disableWriteAheadLogging()**: This method disables the features enabled by enableWriteAheadLogging().
  * **boolean enableWriteAheadLogging()**: Write-ahead logging enables parallel execution of queries from multiple threads on the same database, and reduces the likelihood of stalling on filesystem syncs.
  * **void endTransaction()**: End a transaction.
  * **void execPerConnectionSQL(String sql, Object[] bindArgs)**: Execute the given SQL statement on all connections to this database.
  * **void execSQL(String sql)**: Execute a single SQL statement that is NOT a SELECT or any other SQL statement that returns data.
  * **void execSQL(String sql, Object[] bindArgs)**: Execute a single SQL statement that is NOT a SELECT/INSERT/UPDATE/DELETE.
  * **static String findEditTable(String tables)**: Finds the name of the first table, which is editable.

- ∗ **List<Pair<String, String» getAttachedDbs()**: Returns list of full path-names of all attached databases including the main database by executing 'pragma database_list' on the database.
- ∗ **long getLastChangedRowCount()**: Return the number of database rows that were inserted, updated, or deleted by the most recent SQL statement within the current transaction.
- ∗ **long getLastInsertRowId()**: Return the "rowid" of the last row to be inserted on the current connection.
- ∗ **long getMaximumSize()**: Returns the maximum size the database may grow to.
- ∗ **long getPageSize()**: Returns the current database page size, in bytes.
- ∗ **String getPath()**: Gets the path to the database file.
- ∗ **Map<String, String> getSyncedTables()**: This method was deprecated in API level 15. This method no longer serves any useful purpose and has been deprecated.
- ∗ **long getTotalChangedRowCount()**: Return the total number of database rows that have been inserted, updated, or deleted on the current connection since it was created.
- ∗ **int getVersion()**: Gets the database version.
- ∗ **boolean inTransaction()**: Returns true if the current thread has a transaction pending.
- ∗ **long insert(String table, String nullColumnHack, ContentValues values)**: Convenience method for inserting a row into the database.
- ∗ **long insertOrThrow(String table, String nullColumnHack, ContentValues values)**: Convenience method for inserting a row into the database.
- ∗ **long insertWithOnConflict(String table, String nullColumnHack, ContentValues initialValues, int conflictAlgorithm)**: General method for inserting a row into the database.
- ∗ **boolean isDatabaseIntegrityOk()**: Runs 'pragma integrity_check' on the given database (and all the attached databases) and returns true if the given database (and all its attached databases) pass integrity_check, false otherwise.
- ∗ **boolean isDbLockedByCurrentThread()**: Returns true if the current thread is holding an active connection to the database.
- ∗ **boolean isDbLockedByOtherThreads()**: This method was deprecated in API level 16. Always returns false. Do not use this method.
- ∗ **boolean isOpen()**: Returns true if the database is currently open.
- ∗ **boolean isReadOnly()**: Returns true if the database is opened as read only.
- ∗ **boolean isWriteAheadLoggingEnabled()**: Returns true if write-ahead logging has been enabled for this database.
- ∗ **void markTableSyncable(String table, String deletedTable)**: This method was deprecated in API level 15. This method no longer serves any useful purpose and has been deprecated.
- ∗ **void markTableSyncable(String table, String foreignKey, String updateTable)**: This method was deprecated in API level 15. This method no longer serves any useful purpose and has been deprecated.
- ∗ **boolean needUpgrade(int newVersion)**: Returns true if the new version code is greater than the current database version.

* **static SQLiteDatabase openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)**: Open the database according to the flags OPEN_READWRITE OPEN_READONLY CREATE_IF_NECESSARY and/or NO_LOCALIZED_COLLATORS.
* **static SQLiteDatabase openDatabase(File path, SQLiteDatabase.OpenParams openParams)**: Open the database according to the specified parameters
* **static SQLiteDatabase openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler)**: Open the database according to the flags OPEN_READWRITE OPEN_READONLY CREATE_IF_NECESSARY and/or NO_LOCALIZED_COLLATORS.
* **static SQLiteDatabase openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)**: Equivalent to openDatabase(file.getPath(), factory, CREATE_IF_NECESSARY).
* **static SQLiteDatabase openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory, DatabaseErrorHandler errorHandler)**: Equivalent to openDatabase(path, factory, CREATE_IF_NECESSARY, errorHandler).
* **static SQLiteDatabase openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)**: Equivalent to openDatabase(path, factory, CREATE_IF_NECESSARY).
* **Cursor query(boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)**: Query the given URL, returning a Cursor over the result set.
* **Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)**: Query the given table, returning a Cursor over the result set.
* **Cursor query(boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit, CancellationSignal cancellationSignal)**: Query the given URL, returning a Cursor over the result set.
* **Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)**: Query the given table, returning a Cursor over the result set.
* **Cursor queryWithFactory(SQLiteDatabase.CursorFactory cursorFactory, boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit, CancellationSignal cancellationSignal)**: Query the given URL, returning a Cursor over the result set.
* **Cursor queryWithFactory(SQLiteDatabase.CursorFactory cursorFactory, boolean distinct, String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy, String limit)**: Query the given URL, returning a Cursor over the result set.
* **Cursor rawQuery(String sql, String[] selectionArgs, CancellationSignal cancellationSignal)**: Runs the provided SQL and returns a Cursor over the result set.
* **Cursor rawQuery(String sql, String[] selectionArgs)**: Runs the provided SQL and returns a Cursor over the result set.
* **Cursor rawQueryWithFactory(SQLiteDatabase.CursorFactory cursorFactory, String sql, String[] selectionArgs, String editTable, Can-**

cellationSignal cancellationSignal): Runs the provided SQL and returns a cursor over the result set.

* **Cursor rawQueryWithFactory(SQLiteDatabase.CursorFactory cursorFactory, String sql, String[] selectionArgs, String editTable)**: Runs the provided SQL and returns a cursor over the result set.
* **static int releaseMemory()**: Attempts to release memory that SQLite holds but does not require to operate properly.
* **long replace(String table, String nullColumnHack, ContentValues initialValues)**: Convenience method for replacing a row in the database.
* **long replaceOrThrow(String table, String nullColumnHack, ContentValues initialValues)**: Convenience method for replacing a row in the database.
* **void setCustomAggregateFunction(String functionName, BinaryOperator<String> aggregateFunction)**: Register a custom aggregate function that can be called from SQL expressions.
* **void setCustomScalarFunction(String functionName, UnaryOperator<String> scalarFunction)**: Register a custom scalar function that can be called from SQL expressions.
* **void setForeignKeyConstraintsEnabled(boolean enable)**: Sets whether foreign key constraints are enabled for the database.
* **void setLocale(Locale locale)**: Sets the locale for this database.
* **void setLockingEnabled(boolean lockingEnabled)**: This method was deprecated in API level 16. This method now does nothing. Do not use.
* **void setMaxSqlCacheSize(int cacheSize)**: Sets the maximum size of the prepared-statement cache for this database.
* **long setMaximumSize(long numBytes)**: Sets the maximum size the database will grow to.
* **void setPageSize(long numBytes)**: Sets the database page size.
* **void setTransactionSuccessful()**: Marks the current transaction as successful.
* **void setVersion(int version)**: Sets the database version.
* **String toString()**: Returns a string representation of the object.
* **int update(String table, ContentValues values, String whereClause, String[] whereArgs)**: Convenience method for updating rows in the database.
* **int updateWithOnConflict(String table, ContentValues values, String whereClause, String[] whereArgs, int conflictAlgorithm)**: Convenience method for updating rows in the database.
* **void validateSql(String sql, CancellationSignal cancellationSignal)**: Verifies that a SQL SELECT statement is valid by compiling it.
* **boolean yieldIfContended()**: This method was deprecated in API level 15. if the db is locked more than once (because of nested transactions) then the lock will not be yielded. Use yieldIfContendedSafely instead.
* **boolean yieldIfContendedSafely()**: Temporarily end the transaction to let other threads run.
* **boolean yieldIfContendedSafely(long sleepAfterYieldDelay)**: Temporarily end the transaction to let other threads run.

– **Protected methods**
* **void finalize()**: Called by the garbage collector on an object when garbage collection determines that there are no more references to the object.
* **void onAllReferencesReleased()**: Called when the last reference to the object was released by a call to releaseReference() or close().

– **Constants**
  * **int CONFLICT_ABORT**: When a constraint violation occurs,no ROLL-BACK is executed so changes from prior commands within the same transaction are preserved.
  * **int CONFLICT_FAIL**: When a constraint violation occurs, the command aborts with a return code SQLITE_CONSTRAINT.
  * **int CONFLICT_IGNORE**: When a constraint violation occurs, the one row that contains the constraint violation is not inserted or changed.
  * **int CONFLICT_NONE**: Use the following when no conflict action is specified.
  * **int CONFLICT_REPLACE**: When a UNIQUE constraint violation occurs, the pre-existing rows that are causing the constraint violation are removed prior to inserting or updating the current row.
  * **int CONFLICT_ROLLBACK**: When a constraint violation occurs, an immediate ROLLBACK occurs, thus ending the current transaction, and the command aborts with a return code of SQLITE_CONSTRAINT.
  * **int CREATE_IF_NECESSARY**: Open flag: Flag for openDatabase(File, OpenParams) to create the database file if it does not already exist.
  * **int ENABLE_WRITE_AHEAD_LOGGING**: Open flag: Flag for openDatabase(File, OpenParams) to open the database file with write-ahead logging enabled by default.
  * **String JOURNAL_MODE_DELETE**: The DELETE journaling mode is the normal behavior.
  * **String JOURNAL_MODE_MEMORY**: The MEMORY journaling mode stores the rollback journal in volatile RAM.
  * **String JOURNAL_MODE_OFF**: The OFF journaling mode disables the rollback journal completely.
  * **String JOURNAL_MODE_PERSIST**: The PERSIST journaling mode prevents the rollback journal from being deleted at the end of each transaction.
  * **String JOURNAL_MODE_TRUNCATE**: The TRUNCATE journaling mode commits transactions by truncating the rollback journal to zero-length instead of deleting it.
  * **String JOURNAL_MODE_WAL**: The WAL journaling mode uses a write-ahead log instead of a rollback journal to implement transactions.
  * **int MAX_SQL_CACHE_SIZE**: Absolute max value that can be set by setMaxSqlCacheSize(int).
  * **int NO_LOCALIZED_COLLATORS**: Open flag: Flag for openDatabase(File, OpenParams) to open the database without support for localized collators.
  * **int OPEN_READONLY**: Open flag: Flag for openDatabase(File, OpenParams) to open the database for reading only.
  * **int OPEN_READWRITE**: Open flag: Flag for openDatabase(File, OpenParams) to open the database for reading and writing. If the disk is full, this may fail even before you actually write anything.
  * **int SQLITE_MAX_LIKE_PATTERN_LENGTH**: Maximum Length Of A LIKE Or GLOB Pattern The pattern matching algorithm used in the default LIKE and GLOB implementation of SQLite can exhibit $O(N^2)$ performance (where N is the number of characters in the pattern) for certain pathological cases.
  * **String SYNC_MODE_EXTRA**: The EXTRA sync mode is like FULL sync mode with the addition that the directory containing a rollback journal

is synced after that journal is unlinked to commit a transaction in DELETE journal mode.

* **String SYNC_MODE_FULL**: In FULL sync mode the SQLite database engine will use the xSync method of the VFS to ensure that all content is safely written to the disk surface prior to continuing.
* **String SYNC_MODE_NORMAL**: The NORMAL sync mode, the SQLite database engine will still sync at the most critical moments, but less often than in FULL mode.
* **String SYNC_MODE_OFF**: In OFF sync mode SQLite continues without syncing as soon as it has handed data off to the operating system.

1.4.53 Cursor (Interface)

– **Signature** :

> o `public interface` `Cursor` `implements` `Closeable`

– **Include**

> o `android.``database.``Cursor`

– **Public abstract methods**
  * **abstract void close()**: Closes the Cursor, releasing all of its resources and making it completely invalid.
  * **abstract void copyStringToBuffer(int columnIndex, CharArrayBuffer buffer)**: Retrieves the requested column text and stores it in the buffer provided.
  * **abstract void deactivate()**: This method was deprecated in API level 16. Since requery() is deprecated, so too is this.
  * **abstract byte[] getBlob(int columnIndex)**: Returns the value of the requested column as a byte array.
  * **abstract int getColumnCount()**: Return total number of columns
  * **abstract int getColumnIndex(String columnName)**: Returns the zero-based index for the given column name, or -1 if the column doesn't exist.
  * **abstract int getColumnIndexOrThrow(String columnName)**: Returns the zero-based index for the given column name, or throws IllegalArgumentException if the column doesn't exist.
  * **abstract String getColumnName(int columnIndex)**: Returns the column name at the given zero-based column index.
  * **abstract String[] getColumnNames()**: Returns a string array holding the names of all of the columns in the result set in the order in which they were listed in the result.
  * **abstract int getCount()**: Returns the numbers of rows in the cursor.
  * **abstract double getDouble(int columnIndex)**: Returns the value of the requested column as a double.
  * **abstract Bundle getExtras()**: Returns a bundle of extra values.
  * **abstract float getFloat(int columnIndex)**: Returns the value of the requested column as a float.
  * **abstract int getInt(int columnIndex)**: Returns the value of the requested column as an int.
  * **abstract long getLong(int columnIndex)**: Returns the value of the requested column as a long.
  * **abstract Uri getNotificationUri()**: Return the URI at which notifications of changes in this Cursor's data will be delivered, as previously set by setNotificationUri(ContentResolver, Uri).
  * **default List<Uri> getNotificationUris()**: Return the URIs at which notifications of changes in this Cursor's data will be delivered, as previously set by setNotificationUris(ContentResolver, List).
  * **abstract int getPosition()**: Returns the current position of the cursor in the row set.

* **abstract short getShort(int columnIndex)**: Returns the value of the requested column as a short.
* **abstract String getString(int columnIndex)**: Returns the value of the requested column as a String.
* **abstract int getType(int columnIndex)**: Returns data type of the given column's value.
* **abstract boolean getWantsAllOnMoveCalls()**: onMove() will only be called across processes if this method returns true.
* **abstract boolean isAfterLast()**: Returns whether the cursor is pointing to the position after the last row.
* **abstract boolean isBeforeFirst()**: Returns whether the cursor is pointing to the position before the first row.
* **abstract boolean isClosed()**: return true if the cursor is closed
* **abstract boolean isFirst()**: Returns whether the cursor is pointing to the first row.
* **abstract boolean isLast()**: Returns whether the cursor is pointing to the last row.
* **abstract boolean isNull(int columnIndex)**: Returns true if the value in the indicated column is null.
* **abstract boolean move(int offset)**: Move the cursor by a relative amount, forward or backward, from the current position.
* **abstract boolean moveToFirst()**: Move the cursor to the first row.
* **abstract boolean moveToLast()**: Move the cursor to the last row.
* **abstract boolean moveToNext()**: Move the cursor to the next row.
* **abstract boolean moveToPosition(int position)**: Move the cursor to an absolute position.
* **abstract boolean moveToPrevious()**: Move the cursor to the previous row.
* **abstract void registerContentObserver(ContentObserver observer)**: Register an observer that is called when changes happen to the content backing this cursor.
* **abstract void registerDataSetObserver(DataSetObserver observer)**: Register an observer that is called when changes happen to the contents of the this cursors data set, for example, when the data set is changed via requery(), deactivate(), or close().
* **abstract boolean requery()**: This method was deprecated in API level 15. Don't use this. Just request a new cursor, so you can do this asynchronously and update your list view once the new cursor comes back.
* **abstract Bundle respond(Bundle extras)**: This is an out-of-band way for the user of a cursor to communicate with the cursor.
* **abstract void setExtras(Bundle extras)**: Sets a Bundle that will be returned by getExtras().
* **abstract void setNotificationUri(ContentResolver cr, Uri uri)**: Register to watch a content URI for changes.
* **default void setNotificationUris(ContentResolver cr, List<Uri> uris)**: Similar to setNotificationUri(android.content.ContentResolver, android.net.Uri), except this version allows to watch multiple content URIs for changes.
* **abstract void unregisterContentObserver(ContentObserver observer)**: Unregister an observer that has previously been registered with this cursor via registerContentObserver(ContentObserver).

* **abstract void unregisterDataSetObserver(DataSetObserver observer)**:
  Unregister an observer that has previously been registered with this cursor
  via registerContentObserver(ContentObserver).

1.4.54 ScrollView

– **Hierarchy**

> java.lang.Object → android.view.View → android.view.ViewGroup →
> android.widget.FrameLayout → android.widget.ScrollView

– **Include**

```
o  android.widget.ScrollView
```

– **Constructors**

```
o  ScrollView(Context context)
1  ScrollView(Context context, AttributeSet attrs)
2  ScrollView(Context context, AttributeSet attrs, int
   ↪  defStyleAttr)
3  ScrollView(Context context, AttributeSet attrs, int
   ↪  defStyleAttr, int defStyleRes)
```

– **Public methods**
  * **void addView(View child, int index)**: Adds a child view.
  * **void addView(View child, ViewGroup.LayoutParams params)**: Adds
    a child view with the specified layout parameters.
  * **void addView(View child, int index, ViewGroup.LayoutParams params)**:
    Adds a child view with the specified layout parameters.
  * **void addView(View child)**: Adds a child view.
  * **boolean arrowScroll(int direction)**: Handle scrolling in response to an
    up or down arrow click.
  * **void computeScroll()**: Called by a parent to request that a child update
    its values for mScrollX and mScrollY if necessary.
  * **boolean dispatchKeyEvent(KeyEvent event)**: Dispatch a key event to
    the next view on the focus path.
  * **void draw(Canvas canvas)**: Manually render this view (and all of its
    children) to the given Canvas.
  * **boolean executeKeyEvent(KeyEvent event)**: You can call this function
    yourself to have the scroll view perform scrolling from a key event, just as if
    the event had been dispatched to it by the view hierarchy.
  * **void fling(int velocityY)**: Fling the scroll view
  * **boolean fullScroll(int direction)**: Handles scrolling in response to a
    "home/end" shortcut press.
  * **CharSequence getAccessibilityClassName()**: Return the class name of
    this object to be used for accessibility purposes.
  * **int getBottomEdgeEffectColor()**: Returns the bottom edge effect color.
  * **int getMaxScrollAmount()**: int getTopEdgeEffectColor()
  * **boolean isFillViewport()**: Indicates whether this ScrollView's content is
    stretched to fill the viewport.
  * **boolean isSmoothScrollingEnabled()**:
  * **boolean onGenericMotionEvent(MotionEvent event)**: Implement this
    method to handle generic motion events.

* **boolean onInterceptTouchEvent(MotionEvent ev)**: Implement this method to intercept all touch screen motion events.
* **boolean onNestedFling(View target, float velocityX, float velocityY, boolean consumed)**: Request a fling from a nested scroll.
* **void onNestedScroll(View target, int dxConsumed, int dyConsumed, int dxUnconsumed, int dyUnconsumed)**: React to a nested scroll in progress.
* **void onNestedScrollAccepted(View child, View target, int axes)**: React to the successful claiming of a nested scroll operation.
* **boolean onStartNestedScroll(View child, View target, int nestedScrollAxes)**: React to a descendant view initiating a nestable scroll operation, claiming the nested scroll operation if appropriate.
* **void onStopNestedScroll(View target)**: React to a nested scroll operation ending.
* **boolean onTouchEvent(MotionEvent ev)**: Implement this method to handle pointer events.
* **boolean pageScroll(int direction)**: Handles scrolling in response to a "page up/down" shortcut press.
* **void requestChildFocus(View child, View focused)**: Called when a child of this parent wants focus
* **boolean requestChildRectangleOnScreen(View child, Rect rectangle, boolean immediate)**: Called when a child of this group wants a particular rectangle to be positioned onto the screen.
* **void requestDisallowInterceptTouchEvent(boolean disallowIntercept)**: Called when a child does not want this parent and its ancestors to intercept touch events with ViewGroup.onInterceptTouchEvent(MotionEvent).
* **void requestLayout()**: Call this when something has changed which has invalidated the layout of this view.
* **void scrollTo(int x, int y)**: Set the scrolled position of your view. This version also clamps the scrolling to the bounds of our child.
* **void scrollToDescendant(View child)**: Scrolls the view to the given child.
* **void setBottomEdgeEffectColor(int color)**: Sets the bottom edge effect color.
* **void setEdgeEffectColor(int color)**: Sets the edge effect color for both top and bottom edge effects.
* **void setFillViewport(boolean fillViewport)**: Indicates this ScrollView whether it should stretch its content height to fill the viewport or not.
* **void setSmoothScrollingEnabled(boolean smoothScrollingEnabled)**: Set whether arrow scrolling will animate its transition.
* **void setTopEdgeEffectColor(int color)**: Sets the top edge effect color.
* **boolean shouldDelayChildPressedState()**: Return true if the pressed state should be delayed for children or descendants of this ViewGroup.
* **final void smoothScrollBy(int dx, int dy)**: Like View.scrollBy, but scroll smoothly instead of immediately.
* **final void smoothScrollTo(int x, int y)**: Like scrollTo(int, int), but scroll smoothly instead of immediately.
- **Protected methods**
  * **int computeScrollDeltaToGetChildRectOnScreen(Rect rect)**: Compute the amount to scroll in the Y direction in order to get a rectangle completely on the screen (or, if taller than the screen, at least the first screen size chunk of it).

251

* **int computeVerticalScrollOffset()**: Compute the vertical offset of the vertical scrollbar's thumb within the horizontal range.
* **int computeVerticalScrollRange()**: The scroll range of a scroll view is the overall height of all of its children.
* **float getBottomFadingEdgeStrength()**: Returns the strength, or intensity, of the bottom faded edge.
* **float getTopFadingEdgeStrength()**: Returns the strength, or intensity, of the top faded edge.
* **void measureChild(View child, int parentWidthMeasureSpec, int parentHeightMeasureSpec)**: Ask one of the children of this view to measure itself, taking into account both the MeasureSpec requirements for this view and its padding.
* **void measureChildWithMargins(View child, int parentWidthMeasureSpec, int widthUsed, int parentHeightMeasureSpec, int heightUsed)**: Ask one of the children of this view to measure itself, taking into account both the MeasureSpec requirements for this view and its padding and margins.
* **void onDetachedFromWindow()**: This is called when the view is detached from a window.
* **void onLayout(boolean changed, int l, int t, int r, int b)**: Called from layout when this view should assign a size and position to each of its children.
* **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: Measure the view and its content to determine the measured width and the measured height.
* **void onOverScrolled(int scrollX, int scrollY, boolean clampedX, boolean clampedY)**: Called by overScrollBy(int, int, int, int, int, int, int, int, boolean) to respond to the results of an over-scroll operation.
* **boolean onRequestFocusInDescendants(int direction, Rect previouslyFocusedRect)**: When looking for focus in children of a scroll view, need to be a little more careful not to give focus to something that is scrolled off screen.
* **void onRestoreInstanceState(Parcelable state)**: Hook allowing a view to re-apply a representation of its internal state that had previously been generated by onSaveInstanceState().
* **Parcelable onSaveInstanceState()**: Hook allowing a view to generate a representation of its internal state that can later be used to create a new instance with that same state.
* **void onSizeChanged(int w, int h, int oldw, int oldh)**: This is called during layout when the size of this view has changed.

1.4.55 HorizontalScrollView

– **Hierarchy**

> java.lang.Object → android.view.View → android.view.ViewGroup →
> android.widget.FrameLayout → android.widget.HorizontalScrollView

– **Include**

```
o  android.widget.HorizontalScrollView
```

– **Constructors**

```
o  HorizontalScrollView(Context context)
1  HorizontalScrollView(Context context, AttributeSet
   ↪  attrs)
2  HorizontalScrollView(Context context, AttributeSet
   ↪  attrs, int defStyleAttr)
3  HorizontalScrollView(Context context, AttributeSet
   ↪  attrs, int defStyleAttr, int defStyleRes)
```

– **Public methods**
  * **void addView(View child, int index)**: Adds a child view.
  * **void addView(View child)**: Adds a child view.
  * **void addView(View child, ViewGroup.LayoutParams params)**: Adds a child view with the specified layout parameters.
  * **void addView(View child, int index, ViewGroup.LayoutParams params)**: Adds a child view with the specified layout parameters.
  * **boolean arrowScroll(int direction)**: Handle scrolling in response to a left or right arrow click.
  * **void computeScroll()**: Called by a parent to request that a child update its values for mScrollX and mScrollY if necessary.
  * **boolean dispatchKeyEvent(KeyEvent event)**: Dispatch a key event to the next view on the focus path.
  * **void draw(Canvas canvas)**: Manually render this view (and all of its children) to the given Canvas.
  * **boolean executeKeyEvent(KeyEvent event)**: You can call this function yourself to have the scroll view perform scrolling from a key event, just as if the event had been dispatched to it by the view hierarchy.
  * **void fling(int velocityX)**: Fling the scroll view
  * **boolean fullScroll(int direction)**: Handles scrolling in response to a "home/end" shortcut press.
  * **CharSequence getAccessibilityClassName()**: Return the class name of this object to be used for accessibility purposes.
  * **int getLeftEdgeEffectColor()**: Returns the left edge effect color.
  * **int getMaxScrollAmount()**:
  * **int getRightEdgeEffectColor()**: Returns the right edge effect color.
  * **boolean isFillViewport()**: Indicates whether this HorizontalScrollView's content is stretched to fill the viewport.
  * **boolean isSmoothScrollingEnabled()**:

* **boolean onGenericMotionEvent(MotionEvent event)**: Implement this method to handle generic motion events.
* **boolean onInterceptTouchEvent(MotionEvent ev)**: Implement this method to intercept all touch screen motion events.
* **boolean onTouchEvent(MotionEvent ev)**: Implement this method to handle pointer events.
* **boolean pageScroll(int direction)**: Handles scrolling in response to a "page up/down" shortcut press.
* **void requestChildFocus(View child, View focused)**: Called when a child of this parent wants focus
* **boolean requestChildRectangleOnScreen(View child, Rect rectangle, boolean immediate)**: Called when a child of this group wants a particular rectangle to be positioned onto the screen.
* **void requestDisallowInterceptTouchEvent(boolean disallowIntercept)**: Called when a child does not want this parent and its ancestors to intercept touch events with ViewGroup.onInterceptTouchEvent(MotionEvent).
* **void requestLayout()**: Call this when something has changed which has invalidated the layout of this view.
* **void scrollTo(int x, int y)**: Set the scrolled position of your view. This version also clamps the scrolling to the bounds of our child.
* **void setEdgeEffectColor(int color)**: Sets the edge effect color for both left and right edge effects.
* **void setFillViewport(boolean fillViewport)**: Indicates this HorizontalScrollView whether it should stretch its content width to fill the viewport or not.
* **void setLeftEdgeEffectColor(int color)**: Sets the left edge effect color.
* **void setRightEdgeEffectColor(int color)**: Sets the right edge effect color.
* **void setSmoothScrollingEnabled(boolean smoothScrollingEnabled)**: Set whether arrow scrolling will animate its transition.
* **boolean shouldDelayChildPressedState()**: Return true if the pressed state should be delayed for children or descendants of this ViewGroup.
* **final void smoothScrollBy(int dx, int dy)**: Like View.scrollBy, but scroll smoothly instead of immediately.
* **final void smoothScrollTo(int x, int y)**: Like scrollTo(int, int), but scroll smoothly instead of immediately.

– **Protected methods**

* **int computeHorizontalScrollOffset()**: Compute the horizontal offset of the horizontal scrollbar's thumb within the horizontal range.
* **int computeHorizontalScrollRange()**: The scroll range of a scroll view is the overall width of all of its children.
* **int computeScrollDeltaToGetChildRectOnScreen(Rect rect)**: Compute the amount to scroll in the X direction in order to get a rectangle completely on the screen (or, if taller than the screen, at least the first screen size chunk of it).
* **float getLeftFadingEdgeStrength()**: Returns the strength, or intensity, of the left faded edge.
* **float getRightFadingEdgeStrength()**: Returns the strength, or intensity, of the right faded edge.

- ∗ **void measureChild(View child, int parentWidthMeasureSpec, int parentHeightMeasureSpec)**: Ask one of the children of this view to measure itself, taking into account both the MeasureSpec requirements for this view and its padding.
- ∗ **void measureChildWithMargins(View child, int parentWidthMeasureSpec, int widthUsed, int parentHeightMeasureSpec, int heightUsed)**: Ask one of the children of this view to measure itself, taking into account both the MeasureSpec requirements for this view and its padding and margins.
- ∗ **void onLayout(boolean changed, int l, int t, int r, int b)**: Called from layout when this view should assign a size and position to each of its children.
- ∗ **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: Measure the view and its content to determine the measured width and the measured height.
- ∗ **void onOverScrolled(int scrollX, int scrollY, boolean clampedX, boolean clampedY)**: Called by overScrollBy(int, int, int, int, int, int, int, int, boolean) to respond to the results of an over-scroll operation.
- ∗ **boolean onRequestFocusInDescendants(int direction, Rect previouslyFocusedRect)**: When looking for focus in children of a scroll view, need to be a little more careful not to give focus to something that is scrolled off screen.
- ∗ **void onRestoreInstanceState(Parcelable state)**: Hook allowing a view to re-apply a representation of its internal state that had previously been generated by onSaveInstanceState().
- ∗ **Parcelable onSaveInstanceState()**: Hook allowing a view to generate a representation of its internal state that can later be used to create a new instance with that same state.
- ∗ **void onSizeChanged(int w, int h, int oldw, int oldh)**: This is called during layout when the size of this view has changed.

1.5 Styling widgets with java

1.5.1 View

- **void setBackgroundColor(int color)**: Fills the view's background with a solid color.
- **void setBackground(Drawable background)**: Sets a Drawable as the background.
- **Drawable getBackground()**: Returns the current background drawable.
- **void setForeground(Drawable foreground)**: Draws a Drawable on top of the view's content.
- **void setPadding(int left, int top, int right, int bottom)**: Sets the padding inside the view.
- **int getPaddingLeft() / getPaddingTop() / getPaddingRight() / getPaddingBottom()**: Returns padding values.
- **void setElevation(float elevation)**: Adds shadow depth to the view for visual layering.
- **void setClipToOutline(boolean clipToOutline)**: Clips the view's drawing to its outline (e.g., rounded corners).
- **void setOutlineProvider(ViewOutlineProvider provider)**: Defines the outline shape for shadows and clipping.
- **void setBackgroundTintList(ColorStateList tint)**: Applies tint coloring to the background.
- **void setBackgroundTintMode(PorterDuff.Mode mode)**: Defines how the background tint blends with the original color.
- **void setForegroundTintList(ColorStateList tint)**: Applies tint coloring to the foreground.
- **void setOutlineSpotShadowColor(int color)**: Sets the color of the view's spot shadow.
- **void setOutlineAmbientShadowColor(int color)**: Sets the color of the view's ambient shadow.
- **void setRotationX(float rotationX) / setRotationY(float rotationY)**: Rotates the view around the X or Y axis.
- **void setCameraDistance(float distance)**: Adjusts the 3D perspective depth for rotation effects.

1.5.2 TextView

– **Color, size, typeface**
  * **void setTextColor(int color)**: Sets the text color.
  * **void setTextColor(ColorStateList colors)**: Sets text colors for different states.
  * **void setHighlightColor(int color)**: Sets selection highlight color.
  * **void setLinkTextColor(int color)**: Sets link color.
  * **void setLinkTextColor(ColorStateList colors)**: Sets link colors for states.
  * **void setTextSize(float size)**: Sets text size in scaled pixels.
  * **void setTextSize(int unit, float size)**: Sets text size with unit.
  * **void setTextScaleX(float size)**: Horizontal text scale.
  * **void setTypeface(Typeface tf)**: Sets typeface.
  * **void setTypeface(Typeface tf, int style)**: Sets typeface and style.
  * **void setAllCaps(boolean allCaps)**: Transforms input to ALL CAPS display.
  * **void setTextAppearance(int resId)**: Applies a text appearance style.
  * **void setTextAppearance(Context context, int resId)**: *Deprecated* in API 23.

– **Auto-size text**
  * **void setAutoSizeTextTypeWithDefaults(int autoSizeTextType)**: Enables default auto-size.
  * **void setAutoSizeTextTypeUniformWithConfiguration(int min, int max, int step, int unit)**: Uniform auto-size config.
  * **void setAutoSizeTextTypeUniformWithPresetSizes(int[] presetSizes, int unit)**: Uniform auto-size with presets.

– **Typography, wrapping, justification**
  * **void setLetterSpacing(float letterSpacing)**: Sets letter spacing.
  * **void setLineSpacing(float add, float mult)**: Extra and multiplier.
  * **void setLineHeight(int lineHeight)**: Explicit line height (px).
  * **void setLineHeight(int unit, float lineHeight)**: Explicit line height with unit.
  * **void setEllipsize(TextUtils.TruncateAt where)**: Ellipsize strategy.
  * **void setBreakStrategy(int breakStrategy)**: Paragraph line-break strategy.
  * **void setLineBreakStyle(int lineBreakStyle)**: Line-break style.
  * **void setLineBreakWordStyle(int lineBreakWordStyle)**: Word-break style.
  * **void setHyphenationFrequency(int hyphenationFrequency)**: Hyphenation setting.
  * **void setJustificationMode(int justificationMode)**: Text justification.
  * **void setFallbackLineSpacing(boolean enabled)**: Respect fallback font metrics.
  * **void setIncludeFontPadding(boolean includepad)**: Include extra ascent/descent padding.
  * **void setFirstBaselineToTopHeight(int px)**: Align first baseline to top padding.
  * **void setLastBaselineToBottomHeight(int px)**: Align last baseline to bottom padding.

* **void setLocalePreferredLineHeightForMinimumUsed(boolean flag)**: Locale-preferred min line height.
– **Compound drawables and tints**
  * **void setCompoundDrawables(Drawable left, Drawable top, Drawable right, Drawable bottom)**: L/T/R/B drawables.
  * **void setCompoundDrawablesWithIntrinsicBounds(Drawable left, Drawable top, Drawable right, Drawable bottom)**: With intrinsic bounds.
  * **void setCompoundDrawablesWithIntrinsicBounds(int left, int top, int right, int bottom)**: By resource IDs.
  * **void setCompoundDrawablesRelative(Drawable start, Drawable top, Drawable end, Drawable bottom)**: Start/End variants.
  * **void setCompoundDrawablesRelativeWithIntrinsicBounds(Drawable start, Drawable top, Drawable end, Drawable bottom)**: With intrinsic bounds.
  * **void setCompoundDrawablesRelativeWithIntrinsicBounds(int start, int top, int end, int bottom)**: By resource IDs.
  * **void setCompoundDrawablePadding(int pad)**: Space between text and drawables.
  * **void setCompoundDrawableTintList(ColorStateList tint)**: Drawable tint list.
  * **void setCompoundDrawableTintMode(PorterDuff.Mode mode)**: PorterDuff tint mode.
  * **void setCompoundDrawableTintBlendMode(BlendMode blendMode)**: BlendMode tinting.
– **Hint & cursor/selection visuals**
  * **void setHint(CharSequence hint)**: Hint text.
  * **void setHint(int resid)**: Hint from resource.
  * **void setHintTextColor(int color)**: Hint color.
  * **void setHintTextColor(ColorStateList colors)**: Hint color states.
  * **void setTextCursorDrawable(Drawable d)**: Cursor drawable.
  * **void setTextCursorDrawable(int resId)**: Cursor drawable by resource.
  * **void setTextSelectHandle(int resId)**: Selection handle (resource).
  * **void setTextSelectHandle(Drawable d)**: Selection handle.
  * **void setTextSelectHandleLeft(int resId)**: Left handle (resource).
  * **void setTextSelectHandleLeft(Drawable d)**: Left handle.
  * **void setTextSelectHandleRight(int resId)**: Right handle (resource).
  * **void setTextSelectHandleRight(Drawable d)**: Right handle.
– **Shadows, transforms, and paint flags**
  * **void setShadowLayer(float radius, float dx, float dy, int color)**: Text shadow.
  * **final void setTransformationMethod(TransformationMethod method)**: Visual text transformation (e.g., password).
  * **void setPaintFlags(int flags)**: Underline/strike-through, etc.
  * **void setElegantTextHeight(boolean elegant)**: Use elegant height metrics.
  * **void setFontFeatureSettings(String settings)**: OpenType features.
  * **boolean setFontVariationSettings(String settings)**: Font variations (axes).

- **Alignment / layout-affecting (often part of style guides)**
  * **void setGravity(int gravity)**: Horizontal/vertical alignment within the view.
  * **void setEms(int ems)**: Exact width in ems (typographic sizing).
  * **void setLines(int lines)**: Exact number of lines (tight control of layout look).
- **Styling-related getters (useful to read current style)**
  * **int getCurrentTextColor()**, **ColorStateList getTextColors()**, **int getHighlightColor()**, **ColorStateList getHintTextColors()**, **ColorStateList getLinkTextColors()**
  * **float getTextSize()**, **Typeface getTypeface()**, **float getLetterSpacing()**, **int getLineHeight()**
  * **TextUtils.TruncateAt getEllipsize()**, **int getBreakStrategy()**, **int getHyphenationFrequency()**, **int getJustificationMode()**
  * **String getFontFeatureSettings()**, **String getFontVariationSettings()**
  * **Drawable[] getCompoundDrawables()**, **Drawable[] getCompoundDrawablesRelative()**, **int getCompoundDrawablePadding()**, **ColorStateList getCompoundDrawableTintList()**, **PorterDuff.Mode getCompoundDrawableTintMode()**, **BlendMode getCompoundDrawableTintBlendMode()**
- **Advanced styling hooks (from the *other* list)**
  * **void drawableStateChanged()**: React to state changes that affect drawables/tints.
  * **int[] onCreateDrawableState(int extraSpace)**: Customize drawable state for styling.
  * **void onDraw(Canvas canvas)**: Custom rendering of text/effects.

1.5.3 EditText (Use TextView methods)

– **boolean isStyleShortcutEnabled()**: Returns true if style shortcuts (e.g., `Ctrl+B` for bold) are enabled.

– **void setStyleShortcutsEnabled(boolean enabled)**: Enables or disables style shortcuts such as `Ctrl+B`, `Ctrl+I`, etc.

– **void setEllipsize(TextUtils.TruncateAt ellipsis)**: Specifies how overflowing text should be ellipsized (e.g., at the end or middle) instead of wrapped.

– **void setText(CharSequence text, TextView.BufferType type)**: Sets the text and determines how it's stored (e.g., as plain, styled, or editable text), affecting styling.

1.5.4 Button (Use TextView methods)

1.5.5 ListView

– **Drawable getDivider()**: Returns the drawable that is drawn between each list item.

– **int getDividerHeight()**: Returns the height of the divider between list items.

– **Drawable getOverscrollFooter()**: Returns the drawable drawn below all list content during overscroll.

– **Drawable getOverscrollHeader()**: Returns the drawable drawn above all list content during overscroll.

– **boolean areFooterDividersEnabled()**: Returns whether footer dividers are currently enabled.

– **boolean areHeaderDividersEnabled()**: Returns whether header dividers are currently enabled.

– **void setCacheColorHint(int color)**: Sets a hint color indicating the solid background behind the list, improving appearance on transparent backgrounds.

– **void setDivider(Drawable divider)**: Sets the drawable that will be drawn between list items.

– **void setDividerHeight(int height)**: Sets the height of the divider drawn between list items.

– **void setFooterDividersEnabled(boolean footerDividersEnabled)**: Enables or disables drawing dividers for footer views.

– **void setHeaderDividersEnabled(boolean headerDividersEnabled)**: Enables or disables drawing dividers for header views.

– **void setOverscrollFooter(Drawable footer)**: Sets the drawable to be drawn below all list content during overscroll.

– **void setOverscrollHeader(Drawable header)**: Sets the drawable to be drawn above all list content during overscroll.

– **void dispatchDraw(Canvas canvas)**: (Protected) Called to draw all child views — can be overridden for custom list-item rendering effects.

– **boolean drawChild(Canvas canvas, View child, long drawingTime)**: (Protected) Draws a single list item onto the canvas; used to customize per-item drawing style.

1.5.6 ImageView

- **void animateTransform(Matrix matrix)**: Applies a temporary transformation matrix for animation or visual effects.
- **final void clearColorFilter()**: Removes any color filter or tint applied to the image.
- **ColorFilter getColorFilter()**: Returns the active color filter used for tinting or blending.
- **int getImageAlpha()**: Returns the current alpha (transparency) level of the image.
- **Matrix getImageMatrix()**: Returns the current transformation matrix applied to the image.
- **BlendMode getImageTintBlendMode()**: Returns the blending mode used for applying tints.
- **ColorStateList getImageTintList()**: Returns the color tint list applied to the image drawable.
- **PorterDuff.Mode getImageTintMode()**: Returns the blending mode used for applying the tint.
- **ImageView.ScaleType getScaleType()**: Returns the scale type that defines how the image fits within the view bounds.
- **void setAlpha(int alpha)**: *Deprecated in API 16.* Sets the overall opacity of the view (use `setImageAlpha(int)` instead).
- **final void setColorFilter(int color, PorterDuff.Mode mode)**: Applies a tint color and blending mode to the image.
- **void setColorFilter(ColorFilter cf)**: Applies a custom color filter to modify image appearance.
- **final void setColorFilter(int color)**: Applies a tint color to the image using the default blending mode.
- **void setCropToPadding(boolean cropToPadding)**: Determines whether the image is cropped to the view's padding.
- **void setImageAlpha(int alpha)**: Sets the transparency level for the image drawable.
- **void setImageBitmap(Bitmap bm)**: Sets a bitmap as the content of the ImageView.
- **void setImageDrawable(Drawable drawable)**: Sets a drawable as the image content.
- **void setImageIcon(Icon icon)**: Sets an icon as the content.
- **void setImageMatrix(Matrix matrix)**: Applies a transformation matrix to the image drawable.
- **void setImageResource(int resId)**: Sets an image resource by its resource ID.
- **void setImageTintBlendMode(BlendMode blendMode)**: Sets how the tint blends with the image.
- **void setImageTintList(ColorStateList tint)**: Applies a color tint list to the image drawable.
- **void setImageTintMode(PorterDuff.Mode tintMode)**: Defines the blending mode for the tint.

– **void setScaleType(ImageView.ScaleType scaleType)**: Defines how the image should be scaled or cropped to fit within the view.

– **void onDraw(Canvas canvas)**: (Protected) Allows custom drawing of the image—useful for advanced visual effects.

1.5.7   CompoundButton

- **Drawable getButtonDrawable()**: Returns the drawable used as the button image (e.g., checkmark or radio indicator).

- **BlendMode getButtonTintBlendMode()**: Returns the blending mode used to apply the tint to the button drawable.

- **ColorStateList getButtonTintList()**: Returns the color tint list applied to the button drawable.

- **PorterDuff.Mode getButtonTintMode()**: Returns the blending mode used for tinting.

- **void setButtonDrawable(int resId)**: Sets a drawable resource as the button image.

- **void setButtonDrawable(Drawable drawable)**: Sets a drawable object as the button image.

- **void setButtonIcon(Icon icon)**: Sets an icon as the visual button image.

- **void setButtonTintBlendMode(BlendMode tintMode)**: Defines how the tint color blends with the drawable.

- **void setButtonTintList(ColorStateList tint)**: Applies a tint list (different colors for different states) to the button image.

- **void setButtonTintMode(PorterDuff.Mode tintMode)**: Specifies the tint blending mode.

- **void drawableStateChanged()**: (Protected) Called when the state of the view changes in a way that affects drawable appearance (e.g., pressed, focused, checked).

- **int[] onCreateDrawableState(int extraSpace)**: (Protected) Generates the drawable state array — affects how state-based drawables (like selectors) are drawn.

- **void onDraw(Canvas canvas)**: (Protected) Used for custom drawing — allows overriding default appearance.

- **boolean verifyDrawable(Drawable who)**: (Protected) Ensures the drawable being displayed belongs to this view; relevant for custom visual drawables.

1.5.8 CheckBox (Use Button and CompoundButton styles)

1.5.9 RadioButton (Use button and CompoundButton styles)

1.5.10 Spinner

- **int getDropDownHorizontalOffset()**: Returns the horizontal offset in pixels for positioning the dropdown popup.
- **int getDropDownVerticalOffset()**: Returns the vertical offset in pixels for positioning the dropdown popup.
- **int getDropDownWidth()**: Returns the configured width of the dropdown popup window.
- **int getGravity()**: Returns how the selected item view is positioned within the spinner (e.g., left, center, right).
- **Drawable getPopupBackground()**: Returns the background drawable used for the spinner's dropdown popup.
- **void setDropDownHorizontalOffset(int pixels)**: Sets the horizontal offset in pixels for the spinner's dropdown popup.
- **void setDropDownVerticalOffset(int pixels)**: Sets the vertical offset in pixels for the spinner's dropdown popup.
- **void setDropDownWidth(int pixels)**: Sets the width in pixels for the spinner's dropdown popup window.
- **void setGravity(int gravity)**: Defines how the selected item is positioned within the spinner (e.g., `Gravity.CENTER`).
- **void setPopupBackgroundDrawable(Drawable background)**: Sets a drawable as the background for the dropdown popup.
- **void setPopupBackgroundResource(int resId)**: Sets a background resource for the dropdown popup.
- **CharSequence getPrompt()**: Returns the prompt text displayed when the dialog version of the spinner is shown.
- **void setPrompt(CharSequence prompt)**: Sets custom prompt text for the spinner dialog.
- **void setPromptId(int promptId)**: Sets the prompt text by its string resource ID.
- **int getBaseline()**: Returns the text baseline offset (useful when aligning the spinner with other text views visually).
- **void onLayout(boolean changed, int l, int t, int r, int b)**: (Protected) Handles positioning of spinner items — can affect visual alignment.
- **void onMeasure(int widthMeasureSpec, int heightMeasureSpec)**: (Protected) Determines size and layout — affects how the spinner and dropdown appear.

Styles from AdapterView

- **void setSelection(int position)**: Visually highlights the selected item.
- **void setEmptyView(View emptyView)**: Specifies a view to display when the adapter is empty.

1.5.11 ProgessBar

- **void drawableHotspotChanged(float x, float y)**: Called when the view hotspot changes and must be propagated to drawables or child views.
- **Drawable getCurrentDrawable()**: Returns the drawable currently used to draw the progress bar.
- **Drawable getIndeterminateDrawable()**: Returns the drawable used to draw the progress bar in indeterminate mode.
- **BlendMode getIndeterminateTintBlendMode()**: Returns the blending mode used to apply the tint to the indeterminate drawable.
- **ColorStateList getIndeterminateTintList()**: Returns the color tint list applied to the indeterminate drawable.
- **PorterDuff.Mode getIndeterminateTintMode()**: Returns the blending mode used to apply the tint to the indeterminate drawable.
- **Interpolator getInterpolator()**: Gets the acceleration curve for the indeterminate animation.
- **BlendMode getProgressBackgroundTintBlendMode()**: Returns the blending mode used to apply the tint to the progress background.
- **ColorStateList getProgressBackgroundTintList()**: Returns the tint list applied to the progress background.
- **PorterDuff.Mode getProgressBackgroundTintMode()**: Returns the blending mode used to apply the tint to the progress background.
- **Drawable getProgressDrawable()**: Returns the drawable used to draw the progress bar in progress mode.
- **BlendMode getProgressTintBlendMode()**: Returns the blending mode used to apply the tint to the progress drawable.
- **ColorStateList getProgressTintList()**: Returns the color tint list applied to the progress drawable.
- **PorterDuff.Mode getProgressTintMode()**: Returns the blending mode used to apply the tint to the progress drawable.
- **BlendMode getSecondaryProgressTintBlendMode()**: Returns the blending mode used to apply the tint to the secondary progress drawable.
- **ColorStateList getSecondaryProgressTintList()**: Returns the color tint list applied to the secondary progress drawable.
- **PorterDuff.Mode getSecondaryProgressTintMode()**: Returns the blending mode used to apply the tint to the secondary progress drawable.
- **void invalidateDrawable(Drawable dr)**: Invalidates the specified drawable, forcing a redraw.
- **void jumpDrawablesToCurrentState()**: Jumps all drawables to their current state instantly, skipping animations.
- **void setIndeterminateDrawable(Drawable d)**: Defines the drawable used to draw the progress bar in indeterminate mode.
- **void setIndeterminateDrawableTiled(Drawable d)**: Defines a tileable drawable for the indeterminate mode.
- **void setIndeterminateTintBlendMode(BlendMode blendMode)**: Specifies the blending mode for applying the indeterminate tint.
- **void setIndeterminateTintList(ColorStateList tint)**: Applies a color tint to the indeterminate drawable.

– **void setIndeterminateTintMode(PorterDuff.Mode tintMode)**: Specifies the blending mode for the indeterminate tint.

– **void setInterpolator(Interpolator interpolator)**: Sets the acceleration curve for the indeterminate animation.

– **void setProgressBackgroundTintBlendMode(BlendMode blendMode)**: Specifies the blending mode for the progress background tint.

– **void setProgressBackgroundTintList(ColorStateList tint)**: Applies a tint to the progress background.

– **void setProgressBackgroundTintMode(PorterDuff.Mode tintMode)**: Specifies the blending mode for the progress background tint.

– **void setProgressDrawable(Drawable d)**: Defines the drawable used to draw the progress bar in progress mode.

– **void setProgressDrawableTiled(Drawable d)**: Defines a tileable drawable for the progress bar in progress mode.

– **void setProgressTintBlendMode(BlendMode blendMode)**: Specifies the blending mode for the progress indicator tint.

– **void setProgressTintList(ColorStateList tint)**: Applies a tint to the progress indicator.

– **void setProgressTintMode(PorterDuff.Mode tintMode)**: Specifies the blending mode for the progress indicator tint.

– **void setSecondaryProgressTintBlendMode(BlendMode blendMode)**: Specifies the blending mode for the secondary progress tint.

– **void setSecondaryProgressTintList(ColorStateList tint)**: Applies a tint to the secondary progress indicator.

– **void setSecondaryProgressTintMode(PorterDuff.Mode tintMode)**: Specifies the blending mode for the secondary progress tint.

– **void drawableStateChanged()**: Called whenever the state of the view changes in a way that affects drawable appearance.

– **void onDraw(Canvas canvas)**: Used to perform custom drawing for the progress bar.

– **boolean verifyDrawable(Drawable who)**: Returns true if the specified drawable is managed and displayed by this view.

1.5.12 AbsSeekBar

- **void drawableHotspotChanged(float x, float y)**: Called whenever the view hotspot changes and must be propagated to drawables or child views.
- **boolean getSplitTrack()**: Returns whether the track is visually split by the thumb.
- **Drawable getThumb()**: Returns the drawable representing the thumb — the draggable component indicating progress.
- **int getThumbOffset()**: Returns the amount by which the thumb extends beyond the track.
- **BlendMode getThumbTintBlendMode()**: Returns the blending mode used to apply the tint to the thumb drawable.
- **ColorStateList getThumbTintList()**: Returns the tint color list applied to the thumb drawable.
- **PorterDuff.Mode getThumbTintMode()**: Returns the blending mode used to apply the tint to the thumb drawable.
- **Drawable getTickMark()**: Returns the drawable used as the tick mark for each progress position.
- **BlendMode getTickMarkTintBlendMode()**: Returns the blending mode used to apply tint to the tick mark drawable.
- **ColorStateList getTickMarkTintList()**: Returns the tint color list applied to the tick mark drawable.
- **PorterDuff.Mode getTickMarkTintMode()**: Returns the blending mode used to apply tint to the tick mark drawable.
- **void jumpDrawablesToCurrentState()**: Immediately updates all drawables associated with this view to their current state.
- **void setSplitTrack(boolean splitTrack)**: Specifies whether the track should be visually split by the thumb.
- **void setThumb(Drawable thumb)**: Sets the drawable used as the thumb in the progress meter.
- **void setThumbOffset(int thumbOffset)**: Sets the offset allowing the thumb to extend beyond the track visually.
- **void setThumbTintBlendMode(BlendMode blendMode)**: Defines the blending mode used when applying tint to the thumb drawable.
- **void setThumbTintList(ColorStateList tint)**: Applies a tint color list to the thumb drawable.
- **void setThumbTintMode(PorterDuff.Mode tintMode)**: Specifies the blending mode used with the thumb tint.
- **void setTickMark(Drawable tickMark)**: Sets the drawable used as tick marks at each progress position.
- **void setTickMarkTintBlendMode(BlendMode blendMode)**: Specifies the blending mode used to apply tint to the tick mark drawable.
- **void setTickMarkTintList(ColorStateList tint)**: Applies a tint color list to the tick mark drawable.
- **void setTickMarkTintMode(PorterDuff.Mode tintMode)**: Specifies the blending mode used with the tick mark tint.
- **void drawableStateChanged()**: Called whenever the state of the view changes in a way that affects drawable appearance.

– **void onDraw(Canvas canvas)**: Implement this method to perform custom drawing operations for the view.

– **boolean verifyDrawable(Drawable who)**: Returns true if the specified drawable is being displayed by this view; subclasses override when managing custom drawables.

1.5.13 SeekBar (Use styles from ProgessBar and AbsSeekBar)

1.6 Used API

1.6.1 View

1.6.2 ViewGroup

1.6.3 Context

1.6.4 ConstraintLayout

1.6.5   ConstraintLayout.LayoutParams

1.6.6 RelativeLayout

1.6.7 RelativeLayout.LayoutParams

1.6.8 LinearLayout

1.6.9   LinearLayout.LayoutParams

1.6.10 GridLayout

1.6.11 GridLayout.LayoutParams

1.6.12 TableLayout

1.6.13 TableLayout.LayoutParams

1.6.14   FrameLayout

1.6.15 FrameLayout.LayoutParams

1.6.16   ListView

1.6.17 TextView

1.6.18 EditText

1.6.19   Button

1.6.20 ImageView

1.6.21 CompoundButton

1.6.22 CheckBox

1.6.23   RadioButton

1.6.24   Spinner

1.6.25 Progessbar

1.6.26 AbsSeekBar

1.6.27 SeekBar

1.6.28   Drawable

1.6.29 GradientDrawable