Android API

Nathan Warner

Northern Illinois
University

Computer Science
Northern Illinois University
United States

Contents

XML Attributes

1.1 application (in android manifest)

- **android:allowTaskReparenting=["true" | "false"]**: If true, activities in this app can move to a different task that has the same task affinity when that task returns to the foreground.
- **android:allowBackup=["true" | "false"]**: Allows the app's data to be backed up and restored by the Android backup system (Google Drive or device transfer).
- **android:allowClearUserData=["true" | "false"]**: If true, the user can clear the app's data from system settings ("Clear storage"). If false, that option will be disabled.
- **android:allowNativeHeapPointerTagging=["true" | "false"]**: Enables detection of memory safety issues in native code by tagging heap pointers (Android 12+).
- **android:appCategory=["accessibility" | "audio" | "game" | "image" | "maps" | "news" | "productivity" | "social" | "video"]**: Indicates the main category of the app for system usage, Play Store recommendations, or ranking.
- **android:backupAgent="string"**: Specifies the custom BackupAgent class that handles how the app's data is backed up and restored.
- **android:backupInForeground=["true" | "false"]**: If true, allows backup to run even when the app is in the foreground. Normally backups only happen when in the background.
- **android:banner="drawable resource"**: A wide image displayed for the app on Android TV or other large-screen launcher interfaces.
- **android:dataExtractionRules="string resource"]**: Specifies an XML file that defines what data can be extracted for backup or device-to-device transfer (Android 12+).
- **android:debuggable=["true" | "false"]**: If true, the app can be debugged using tools like Android Studio. Should be false in release builds for security.
- **android:description="string resource"**: A user-readable description of the application, usually displayed in app info or launcher details.
- **android:enabled=["true" | "false"]**: Determines whether the application is enabled. If false, all components (activities, services) are disabled and cannot run.
- **android:enableOnBackInvokedCallback=["true" | "false"]**: Enables support for the modern back navigation system (OnBackInvokedCallback) for the entire application.
- **android:extractNativeLibs=["true" | "false"]**: If true, native .so libraries are extracted from the APK onto the filesystem. If false, they are loaded directly from the APK (improves startup and reduces size).
- **android:fullBackupContent="string"**: Refers to an XML file that specifies which files/directories to include or exclude from full app backup.
- **android:fullBackupOnly=["true" | "false"]**: If true, disables key-value backup and allows only full-data backup for this app.
- **android:gwpAsanMode=["always" | "never"]**: Enables or disables GWP-ASan, a lightweight memory error detection tool, for native code.

- **android:hasCode=["true" | "false"]**: Indicates whether the app includes compiled code (classes.dex). If false, the app contains only resources and no executable code.
- **android:hasFragileUserData=["true" | "false"]**: Marks the app's data as sensitive or prone to corruption, preventing unsafe backup or migration scenarios.
- **android:hardwareAccelerated=["true" | "false"]**: Enables GPU-accelerated rendering for the entire application unless overridden at the Activity level.
- **android:icon="drawable resource"**: Specifies the default application icon shown in the launcher and system UI.
- **android:isGame=["true" | "false"]**: Indicates whether the application is a game. Helps devices and Play Store organize the app in the "Games" category.
- **android:isMonitoringTool=["parental_control" | "enterprise_management" | "other"]**: Declares that the app monitors device usage or behavior (e.g., parental controls or enterprise apps).
- **android:killAfterRestore=["true" | "false"]**: If true, the system kills the app process after a restore operation, forcing a clean restart to apply restored data.
- **android:largeHeap=["true" | "false"]**: Requests a larger memory heap size for memory-intensive applications. Use carefully, as it increases RAM usage.
- **android:label="string resource"**: The human-readable name of the app shown on the home screen, in settings, and recent apps.
- **android:logo="drawable resource"**: An alternative icon (usually wider or stylized) used in the ActionBar or TaskSwitcher instead of the default app icon.
- **android:manageSpaceActivity="string"**: Specifies an Activity to launch when the user taps "Manage space" in app settings (used to clear cache or manage data).
- **android:name="string"**: Specifies a custom `Application` class name that extends `android.app.Application`. This class runs before any activity or service.
- **android:networkSecurityConfig="xml resource"**: Refers to an XML file that defines custom network security policies like certificate pinning, cleartext traffic rules, and trusted CAs.
- **android:permission="string"**: Specifies a permission that other applications must have in order to interact with this app's components (activities, services, receivers) by default.
- **android:persistent=["true" | "false"]**: If true, keeps the app running persistently in memory. Typically used only for system or core apps; ignored for normal third-party apps.
- **android:process="string"**: Specifies the name of the process in which the application components should run. A name starting with ":" indicates a private process.
- **android:restoreAnyVersion=["true" | "false"]**: Allows the app to restore backup data even if the backup was created using a newer version of the app.
- **android:requestLegacyExternalStorage=["true" | "false"]**: For Android 10 (API 29), this allows the app to temporarily use legacy storage access instead of scoped storage.
- **android:requiredAccountType="string"**: Specifies the type of account (e.g., Google, Exchange) required for the application to function, if any.

- **android:resizeableActivity=["true" | "false"]**: Declares whether activities in the application support multi-window mode (split-screen, freeform windows).
- **android:restrictedAccountType="string"**: Specifies an account type that the app uses when running in restricted profiles (e.g., parental control profile on tablets).
- **android:supportsRtl=["true" | "false"]**: Enables support for right-to-left (RTL) layouts when the system language is an RTL language (such as Arabic or Hebrew).
- **android:taskAffinity="string"**: Defines the default task affinity for all activities in the application. Activities with the same task affinity are grouped in the same task.

1.2 activity (in android manifest)

- **android:allowEmbedded=["true" | "false"]**: If true, this activity can be embedded inside another app's UI (mainly for automotive, desktop, or multi-window environments).
- **android:allowTaskReparenting=["true" | "false"]**: Allows the activity to move from the task it started in to another task with the same taskAffinity when that task comes to the foreground.
- **android:alwaysRetainTaskState=["true" | "false"]**: Prevents the system from clearing the activity's task state when the user re-launches the task after leaving it.
- **android:autoRemoveFromRecents=["true" | "false"]**: Automatically removes the task containing this activity from the "Recent Apps" screen when the user leaves it.
- **android:banner="drawable resource"**: Specifies a wide banner image for the activity, mainly shown on Android TV home screens.
- **android:canDisplayOnRemoteDevices=["true" | "false"]**: Allows the activity to be displayed remotely on other devices or screens (like casting or automotive displays).
- **android:clearTaskOnLaunch=["true" | "false"]**: Clears all other activities above this one in the task stack whenever the user launches the app again from the home screen.
- **android:colorMode=["hdr" | "wideColorGamut"]**: Specifies the color rendering mode this activity uses—HDR for high dynamic range or wideColorGamut for wider color spaces.
- **android:configChanges=["colorMode", "density", "fontScale", "fontWeightAdjustment", "grammaticalGender", "keyboard", "keyboardHidden", "layoutDirection", "locale", "mcc", "mnc", "navigation", "orientation", "screenLayout", "screenSize", "smallestScreenSize", "touchscreen", "uiMode"]**: Tells Android which configuration changes this activity will handle manually instead of being recreated (e.g., rotation, locale, font size).
- **android:directBootAware=["true" | "false"]**: If true, the activity can run before the user unlocks the device after reboot (Direct Boot mode) and access device-protected storage.
- **android:documentLaunchMode=["intoExisting" | "always" | "none" | "never"]**: Controls how this activity is launched when working with document-centric tasks (like recent documents in productivity apps). Determines whether a new instance is created or an existing one is reused.
- **android:enabled=["true" | "false"]**: Specifies whether this activity can be instantiated and used. If false, the activity is disabled and cannot be launched.
- **android:enableOnBackInvokedCallback=["true" | "false"]**: Enables the modern back navigation API (‘OnBackInvokedCallback‘) instead of the deprecated ‘onBackPressed()‘ method.
- **android:excludeFromRecents=["true" | "false"]**: If true, this activity's task will not appear in the recent apps overview screen.

- **android:exported=["true" | "false"]**: Defines whether this activity can be launched by components of other apps. Required to be explicitly set if the activity uses an intent filter (Android 12+).
- **android:finishOnTaskLaunch=["true" | "false"]**: If true, the activity will be destroyed whenever the user leaves the task and then later returns to it.
- **android:hardwareAccelerated=["true" | "false"]**: Enables or disables GPU hardware acceleration for rendering the activity's user interface.
- **android:icon="drawable resource"**: Specifies an icon for the activity; used when different from the application's default icon.
- **android:immersive=["true" | "false"]**: If true, requests immersive full-screen mode (hiding status and navigation bars) for distraction-free experiences like games or media.
- **android:label="string resource"**: Sets the display name of the activity shown in the title bar or recent apps list. Typically references a string resource.
- **android:launchMode=["standard" | "singleTop" | "singleTask" | "singleInstance" | "singleInstancePerTask"]**: Determines how a new instance of the activity is launched and added to the task stack. Controls whether multiple instances can exist or if an existing one is reused.
- **android:lockTaskMode=["normal" | "never" | "if_whitelisted" | "always"]**: Specifies whether the activity can enter Lock Task (Kiosk) Mode, which pins the app and restricts user navigation.
- **android:maxRecents="integer"**: Limits the number of recent tasks that can exist for this activity when launched in document/task mode.
- **android:maxAspectRatio="float"**: Specifies the maximum screen aspect ratio the activity supports. Prevents UI stretching on tall or wide screens.
- **android:minAspectRatio="float"**: Specifies the minimum aspect ratio supported by the activity to maintain UI usability.
- **android:multiprocess=["true" | "false"]**: (Deprecated) If true, allows the activity to run in multiple processes simultaneously. Rarely used now.
- **android:name="string"**: The fully qualified name of the Activity class (e.g., `.MainActivity` or `com.example.MyActivity`).
- **android:noHistory=["true" | "false"]**: If true, the activity is finished and removed from the back stack as soon as the user navigates away from it.
- **android:parentActivityName="string"**: Declares the logical parent of this activity for "Up" navigation in the app. Should match another activity's class name.
- **android:persistableMode=["persistRootOnly" | "persistAcrossReboots" | "persistNever"]**: Specifies whether the activity's state is preserved across reboots using `onSaveInstanceState(PersistableBundle)`.
- **android:permission="string"**: Specifies a permission that other apps must have in order to start this activity.
- **android:process="string"**: Specifies the name of the process in which this activity should run. If it starts with ":", the activity runs in a private process for the app.

- **android:relinquishTaskIdentity=["true" | "false"]**: If true, removes the task's association with the app (label and icon) when the activity finishes, making the task appear "generic" in Recents.
- **android:requireContentUriPermissionFromCaller=["none" | "read" | "readAndWrite" | "readOrWrite" | "write"]**: Specifies that callers launching this activity must have certain URI permissions (read/write) on the content being shared.
- **android:resizeableActivity=["true" | "false"]**: Determines whether the activity supports multi-window mode, freeform windowing, or resizing at runtime.
- **android:screenOrientation=["unspecified" | "behind" | "landscape" | "portrait" | "reverseLandscape" | "reversePortrait" | "sensorLandscape" | "sensorPortrait" | "userLandscape" | "userPortrait" | "sensor" | "fullSensor" | "nosensor" | "user" | "fullUser" | "locked"]**: Controls the orientation in which the activity is displayed or how it reacts to orientation changes.
- **android:showForAllUsers=["true" | "false"]**: If true, this activity will be visible and usable by all users on a multi-user Android device (like tablets, TVs).
- **android:stateNotNeeded=["true" | "false"]**: If true, the system will not save the activity's UI state and may destroy it without warning when it moves to the background.
- **android:supportsPictureInPicture=["true" | "false"]**: Enables Picture-in-Picture (PiP) mode, allowing this activity to shrink into a small floating window (e.g., video players).
- **android:taskAffinity="string"**: Defines which task this activity prefers to be associated with. Activities with the same affinity can be grouped in the same task.
- **android:theme="resource or theme"**: Specifies the UI theme or style applied to this activity (e.g., colors, action bar, dark mode). Overrides the application's default theme if set.
- **android:uiOptions=["none" | "splitActionBarWhenNarrow"]**: Provides additional UI behavior hints. Commonly used to force the action bar to split into top and bottom bars on narrow screens.
- **android:windowSoftInputMode=["stateUnspecified", "stateUnchanged", "stateHidden", "stateAlwaysHidden", "stateVisible", "stateAlwaysVisible", "adjustUnspecified", "adjustResize", "adjustPan"]**: Controls how the activity behaves when the soft keyboard appears — whether it resizes the layout, pans contents, or auto-opens/closes the keyboard.

1.3 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.

1.4 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)

1.5 RelativeLayout.LayoutParams

- **android:layout_above**: Positions the bottom edge of this view above the given anchor view ID.
- **android:layout_alignBaseline**: Positions the baseline of this view on the baseline of the given anchor view ID.
- **android:layout_alignBottom**: Makes the bottom edge of this view match the bottom edge of the given anchor view ID.
- **android:layout_alignEnd**: Makes the end edge of this view match the end edge of the given anchor view ID.
- **android:layout_alignLeft**: Makes the left edge of this view match the left edge of the given anchor view ID.
- **android:layout_alignParentBottom**: If true, makes the bottom edge of this view match the bottom edge of the parent.
- **android:layout_alignParentEnd**: If true, makes the end edge of this view match the end edge of the parent.
- **android:layout_alignParentLeft**: If true, makes the left edge of this view match the left edge of the parent.
- **android:layout_alignParentRight**: If true, makes the right edge of this view match the right edge of the parent.
- **android:layout_alignParentStart**: If true, makes the start edge of this view match the start edge of the parent.
- **android:layout_alignParentTop**: If true, makes the top edge of this view match the top edge of the parent.
- **android:layout_alignRight**: Makes the right edge of this view match the right edge of the given anchor view ID.
- **android:layout_alignStart**: Makes the start edge of this view match the start edge of the given anchor view ID.
- **android:layout_alignTop**: Makes the top edge of this view match the top edge of the given anchor view ID.
- **android:layout_alignWithParentIfMissing**: If set to true, the parent will be used as the anchor when the anchor cannot be be found for layout_toLeftOf, layout_toRightOf, etc.
- **android:layout_below**: Positions the top edge of this view below the given anchor view ID.
- **android:layout_centerHorizontal**: If true, centers this child horizontally within its parent.
- **android:layout_centerInParent**: If true, centers this child horizontally and vertically within its parent.
- **android:layout_centerVertical**: If true, centers this child vertically within its parent.
- **android:layout_toEndOf**: Positions the start edge of this view to the end of the given anchor view ID.

- **android:layout_toLeftOf**: Positions the right edge of this view to the left of the given anchor view ID.
- **android:layout_toRightOf**: Positions the left edge of this view to the right of the given anchor view ID.
- **android:layout_toStartOf**: Positions the end edge of this view to the start of the given anchor view ID.

1.6 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.

1.7 TableLayout

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

1.8 TableRow

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

1.9 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.

1.10 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.

1.11 AutoCompleteTextView

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

1.12 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

1.13 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.

1.14 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.

1.15 AdapterView

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

1.16 Adapter

1.17 ArrayAdapter

1.18 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.

1.19 ImageButton

- Only inherited from View and ImageView

1.20 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.

1.21 CheckBox

- Only inherited from CompoundButton, TextView, and View

1.22 RadioGroup

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

1.23 RadioButton

- Only inherited from CompoundButton, TextView, and View.

1.24 AbsSpinner

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

1.25 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.

1.26 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.

1.27 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.

1.28 SeekBar

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

1.29 GradientDrawable

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

1.30 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.

1.31 set (AnimationSet)

- Only inherited from Animation

1.32 alpha (AlphaAnimation)

- Only inherited from Animation

1.33 rotate (RotateAnimation)

- Only inherited from Animation

1.34 scale (ScaleAnimation)

- Only inherited from Animation

1.35 translate (TranslateAnimation)

- Only inherited from Animation

1.36 ScrollView

- Only those inherited from View, ViewGroup, and FrameLayout

1.37 HorizontalScrollView

- Only those inherited from View, ViewGroup, and FrameLayout

Java API

2.1 Activity

- **Hierarchy**

  java.lang.Object → android.content.Context → android.content.ContextWrapper → android.view.ContextThemeWrapper → android.app.Activity

- **Include**

```
android.app.Activity
```

- **Constructors**

```
Activity()
```

- **Public methods**
  - **void addContentView(View view, ViewGroup.LayoutParams params)**: Add an additional content view to the activity.
  - **void clearOverrideActivityTransition(int overrideType)**: Clears the animations which are set from overrideActivityTransition(int, int, int).
  - **void closeContextMenu()**: Programmatically closes the most recently opened context menu, if showing.
  - **void closeOptionsMenu()**: Progammatically closes the options menu.
  - **PendingIntent createPendingResult(int requestCode, Intent data, int flags)**: Create a new PendingIntent object which you can hand to others for them to use to send result data back to your onActivityResult(int, int, Intent) callback.
  - **final void dismissDialog(int id)**: This method was deprecated in API level 15. Use the new DialogFragment class with FragmentManager instead; this is also available on older platforms through the Android compatibility package.
  - **final void dismissKeyboardShortcutsHelper()**: Dismiss the Keyboard Shortcuts screen.
  - **boolean dispatchGenericMotionEvent(MotionEvent ev)**: Called to process generic motion events.
  - **boolean dispatchKeyEvent(KeyEvent event)**: Called to process key events.
  - **boolean dispatchKeyShortcutEvent(KeyEvent event)**: Called to process a key shortcut event.
  - **boolean dispatchPopulateAccessibilityEvent(AccessibilityEvent event)**: Called to process population of AccessibilityEvents.
  - **boolean dispatchTouchEvent(MotionEvent ev)**: Called to process touch screen events.
  - **boolean dispatchTrackballEvent(MotionEvent ev)**: Called to process trackball events.
  - **void dump(String prefix, FileDescriptor fd, PrintWriter writer, String[] args)**: Print the Activity's state into the given stream.

- **boolean enterPictureInPictureMode(PictureInPictureParams params)**: Puts the activity in picture-in-picture mode if possible in the current system state.
- **void enterPictureInPictureMode()**: Puts the activity in picture-in-picture mode if possible in the current system state.
- **<T extends View> T findViewById(int id)**: Finds a view that was identified by the android:id XML attribute that was processed in onCreate(Bundle).
- **void finish()**: Call this when your activity is done and should be closed.
- **void finishActivity(int requestCode)**: Force finish another activity that you had previously started with startActivityForResult(Intent, int).
- **void finishActivityFromChild(Activity child, int requestCode)**: This method was deprecated in API level 30. Use finishActivity(int) instead.
- **void finishAffinity()**: Finish this activity as well as all activities immediately below it in the current task that have the same affinity.
- **void finishAfterTransition()**: Reverses the Activity Scene entry Transition and triggers the calling Activity to reverse its exit Transition.
- **void finishAndRemoveTask()**: Call this when your activity is done and should be closed and the task should be completely removed as a part of finishing the root activity of the task.
- **void finishFromChild(Activity child)**: This method was deprecated in API level 30. Use finish() instead.
- **ActionBar getActionBar()**: Retrieve a reference to this activity's ActionBar.
- **final Application getApplication()**: Return the application that owns this activity.
- **ComponentCaller getCaller()**: Returns the ComponentCaller instance of the app that started this activity.
- **ComponentName getCallingActivity()**: Return the name of the activity that invoked this activity.
- **String getCallingPackage()**: Return the name of the package that invoked this activity.
- **int getChangingConfigurations()**: If this activity is being destroyed because it can not handle a configuration parameter being changed (and thus its onConfigurationChanged(android.content.res.Configuration) method is not being called), then you can use this method to discover the set of changes that have occurred while in the process of being destroyed.
- **ComponentName getComponentName()**: Returns the complete component name of this activity.
- **Scene getContentScene()**: Retrieve the Scene representing this window's current content.
- **TransitionManager getContentTransitionManager()**: Retrieve the TransitionManager responsible for default transitions in this window.
- **ComponentCaller getCurrentCaller()**: Returns the ComponentCaller instance of the app that re-launched this activity with a new intent via onNewIntent(Intent) or onActivityResult(int, int, Intent).
- **View getCurrentFocus()**: Calls Window.getCurrentFocus() on the Window of this Activity to return the currently focused view.
- **FragmentManager getFragmentManager()**: This method was deprecated in API level 28. Use FragmentActivity.getSupportFragmentManager()

- **ComponentCaller getInitialCaller()**: Returns the ComponentCaller instance of the app that initially launched this activity.
- **Intent getIntent()**: Returns the intent that started this activity.
- **Object getLastNonConfigurationInstance()**: Retrieve the non-configuration instance data that was previously returned by onRetainNonConfigurationInstance().
- **String getLaunchedFromPackage()**: Returns the package name of the app that initially launched this activity.
- **int getLaunchedFromUid()**: Returns the uid of the app that initially launched this activity.
- **LayoutInflater getLayoutInflater()**: Convenience for calling Window.getLayoutInflater().
- **LoaderManager getLoaderManager()**: This method was deprecated in API level 28. Use FragmentActivity.getSupportLoaderManager()
- **String getLocalClassName()**: Returns class name for this activity with the package prefix removed.
- **int getMaxNumPictureInPictureActions()**: Return the number of actions that will be displayed in the picture-in-picture UI when the user interacts with the activity currently in picture-in-picture mode.
- **final MediaController getMediaController()**: Gets the controller which should be receiving media key and volume events while this activity is in the foreground.
- **MenuInflater getMenuInflater()**: Returns a MenuInflater with this context.
- **OnBackInvokedDispatcher getOnBackInvokedDispatcher()**: Returns the OnBackInvokedDispatcher instance associated with the window that this activity is attached to.
- **final Activity getParent()**: This method was deprecated in API level 35. ActivityGroup is deprecated.
- **Intent getParentActivityIntent()**: Obtain an Intent that will launch an explicit target activity specified by this activity's logical parent.
- **SharedPreferences getPreferences(int mode)**: Retrieve a SharedPreferences object for accessing preferences that are private to this activity.
- **Uri getReferrer()**: Return information about who launched this activity.
- **int getRequestedOrientation()**: Returns the current requested orientation of the activity, which is either the orientation requested in the app manifest or the last orientation given to setRequestedOrientation(int).
- **final SearchEvent getSearchEvent()**: During the onSearchRequested() callbacks, this function will return the SearchEvent that triggered the callback, if it exists.
- **final SplashScreen getSplashScreen()**: Get the interface that activity use to talk to the splash screen.
- **Object getSystemService(String name)**: Return the handle to a system-level service by name.
- **int getTaskId()**: Return the identifier of the task this activity is in.
- **final CharSequence getTitle()**:
- **final int getTitleColor()**:
- **VoiceInteractor getVoiceInteractor()**: Retrieve the active VoiceInteractor that the user is going through to interact with this activity.

- **final int getVolumeControlStream()**: Gets the suggested audio stream whose volume should be changed by the hardware volume controls.
- **Window getWindow()**: Retrieve the current Window for the activity.
- **WindowManager getWindowManager()**: Retrieve the window manager for showing custom windows.
- **boolean hasWindowFocus()**: Returns true if this activity's main window currently has window focus.
- **void invalidateOptionsMenu()**: Declare that the options menu has changed, so should be recreated.
- **boolean isActivityTransitionRunning()**: Returns whether there are any activity transitions currently running on this activity.
- **boolean isChangingConfigurations()**: Check to see whether this activity is in the process of being destroyed in order to be recreated with a new configuration.
- **final boolean isChild()**: This method was deprecated in API level 35. ActivityGroup is deprecated.
- **boolean isDestroyed()**: Returns true if the final onDestroy() call has been made on the Activity, so this instance is now dead.
- **boolean isFinishing()**: Check to see whether this activity is in the process of finishing, either because you called finish() on it or someone else has requested that it finished.
- **boolean isImmersive()**: Bit indicating that this activity is "immersive" and should not be interrupted by notifications if possible.
- **boolean isInMultiWindowMode()**: Returns true if the activity is currently in multi-window mode.
- **boolean isInPictureInPictureMode()**: Returns true if the activity is currently in picture-in-picture mode.
- **boolean isLaunchedFromBubble()**: Indicates whether this activity is launched from a bubble.
- **boolean isLocalVoiceInteractionSupported()**: Queries whether the currently enabled voice interaction service supports returning a voice interactor for use by the activity.
- **boolean isTaskRoot()**: Return whether this activity is the root of a task.
- **boolean isVoiceInteraction()**: Check whether this activity is running as part of a voice interaction with the user.
- **boolean isVoiceInteractionRoot()**: Like isVoiceInteraction(), but only returns true if this is also the root of a voice interaction.
- **final Cursor managedQuery(Uri uri, String[] projection, String selection, String[] selectionArgs, String sortOrder)**: This method was deprecated in API level 15. Use CursorLoader instead.
- **boolean moveTaskToBack(boolean nonRoot)**: Move the task containing this activity to the back of the activity stack.
- **boolean navigateUpTo(Intent upIntent)**: Navigate from this activity to the activity specified by upIntent, finishing this activity in the process.
- **boolean navigateUpToFromChild(Activity child, Intent upIntent)**: This method was deprecated in API level 30. Use navigateUpTo(android.content.Intent) instead.
- **void onActionModeFinished(ActionMode mode)**: Notifies the activity that an action mode has finished.

- **void onActionModeStarted(ActionMode mode)**: Notifies the Activity that an action mode has been started.
- **void onActivityReenter(int resultCode, Intent data)**: Called when an activity you launched with an activity transition exposes this Activity through a returning activity transition, giving you the resultCode and any additional data from it.
- **void onActivityResult(int requestCode, int resultCode, Intent data, ComponentCaller caller)**: Same as onActivityResult(int, int, android.content.Intent), but with an extra parameter for the ComponentCaller instance associated with the app that sent the result.
- **void onAttachFragment(Fragment fragment)**: This method was deprecated in API level 28. Use FragmentActivity.onAttachFragment(androidx.fragment.app.Fragment)
- **void onAttachedToWindow()**: Called when the main window associated with the activity has been attached to the window manager.
- **void onBackPressed()**: This method was deprecated in API level 33. Use OnBackInvokedCallback or androidx.activity.OnBackPressedCallback to handle back navigation instead.

  Starting from Android 13 (API level 33), back event handling is moving to an ahead-of-time model and Activity.onBackPressed() and KeyEvent.KEYCODE_BACK should not be used to handle back events (back gesture or back button click). Instead, an OnBackInvokedCallback should be registered using Activity.getOnBackInvokedDispatcher() .registerOnBackInvokedCallback(priority, callback).
- **void onConfigurationChanged(Configuration newConfig)**: Called by the system when the device configuration changes while your activity is running.
- **void onContentChanged()**: This hook is called whenever the content view of the screen changes (due to a call to Window.setContentView or Window.addContentView).
- **boolean onContextItemSelected(MenuItem item)**: This hook is called whenever an item in a context menu is selected.
- **void onContextMenuClosed(Menu menu)**: This hook is called whenever the context menu is being closed (either by the user canceling the menu with the back/menu button, or when an item is selected).
- **void onCreate(Bundle savedInstanceState, PersistableBundle persistentState)**: Same as onCreate(android.os.Bundle) but called for those activities created with the attribute R.attr.persistableMode set to persistAcrossReboots.
- **void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo menuInfo)**: Called when a context menu for the view is about to be shown.
- **CharSequence onCreateDescription()**: Generate a new description for this activity.
- **void onCreateNavigateUpTaskStack(TaskStackBuilder builder)**: Define the synthetic task stack that will be generated during Up navigation from a different task.
- **boolean onCreateOptionsMenu(Menu menu)**: Initialize the contents of the Activity's standard options menu.
- **boolean onCreatePanelMenu(int featureId, Menu menu)**: Default implementation of Window.Callback.onCreatePanelMenu(int, Menu) for activities.

- **View onCreatePanelView(int featureId)**: Default implementation of Window.Callback.onCreatePanelView(int) for activities.
- **boolean onCreateThumbnail(Bitmap outBitmap, Canvas canvas)**: This method was deprecated in API level 28. Method doesn't do anything and will be removed in the future.
- **View onCreateView(View parent, String name, Context context, AttributeSet attrs)**: Standard implementation of LayoutInflater.Factory2.onCreateView(View, String, Context, AttributeSet) used when inflating with the LayoutInflater returned by Context.getSystemService(Class).
- **View onCreateView(String name, Context context, AttributeSet attrs)**: Standard implementation of LayoutInflater.Factory.onCreateView(String, Context, AttributeSet) used when inflating with the LayoutInflater returned by Context.getSystemService(Class).
- **void onDetachedFromWindow()**: Called when the main window associated with the activity has been detached from the window manager.
- **void onEnterAnimationComplete()**: Activities cannot draw during the period that their windows are animating in.
- **boolean onGenericMotionEvent(MotionEvent event)**: Called when a generic motion event was not handled by any of the views inside of the activity.
- **void onGetDirectActions(CancellationSignal cancellationSignal, Consumer<List<DirectAction» callback)**: Returns the list of direct actions supported by the app.
- **boolean onKeyDown(int keyCode, KeyEvent event)**: Called when a key was pressed down and not handled by any of the views inside of the activity.
- **boolean onKeyLongPress(int keyCode, KeyEvent event)**: Default implementation of KeyEvent.Callback.onKeyLongPress(): always returns false (doesn't handle the event).
- **boolean onKeyMultiple(int keyCode, int repeatCount, KeyEvent event)**: Default implementation of KeyEvent.Callback.onKeyMultiple(): always returns false (doesn't handle the event).
- **boolean onKeyShortcut(int keyCode, KeyEvent event)**: Called when a key shortcut event is not handled by any of the views in the Activity.
- **boolean onKeyUp(int keyCode, KeyEvent event)**: Called when a key was released and not handled by any of the views inside of the activity.
- **void onLocalVoiceInteractionStarted()**: Callback to indicate that startLocalVoiceInteraction(android.os.Bundle) has resulted in a voice interaction session being started.
- **void onLocalVoiceInteractionStopped()**: Callback to indicate that the local voice interaction has stopped either because it was requested through a call to stopLocalVoiceInteraction() or because it was canceled by the user.
- **void onLowMemory()**: This is called when the overall system is running low on memory, and actively running processes should trim their memory usage.
- **boolean onMenuItemSelected(int featureId, MenuItem item)**: Default implementation of Window.Callback.onMenuItemSelected(int, MenuItem) for activities.
- **boolean onMenuOpened(int featureId, Menu menu)**: Called when a panel's menu is opened by the user.
- **void onMultiWindowModeChanged(boolean isInMultiWindowMode)**: This method was deprecated in API level 26. Use onMultiWindowModeChanged(boolean, android.content.res.Configuration) instead.

- **void onMultiWindowModeChanged(boolean isInMultiWindowMode, Configuration newConfig)**: Called by the system when the activity changes from fullscreen mode to multi-window mode and visa-versa.
- **boolean onNavigateUp()**: This method is called whenever the user chooses to navigate Up within your application's activity hierarchy from the action bar.
- **boolean onNavigateUpFromChild(Activity child)**: This method was deprecated in API level 30. Use onNavigateUp() instead.
- **void onNewIntent(Intent intent, ComponentCaller caller)**: Same as onNewIntent(android.content.Intent), but with an extra parameter for the ComponentCaller instance associated with the app that sent the intent.
- **boolean onOptionsItemSelected(MenuItem item)**: This hook is called whenever an item in your options menu is selected.
- **void onOptionsMenuClosed(Menu menu)**: This hook is called whenever the options menu is being closed (either by the user canceling the menu with the back/menu button, or when an item is selected).
- **void onPanelClosed(int featureId, Menu menu)**: Default implementation of Window.Callback.onPanelClosed(int, Menu) for activities.
- **void onPerformDirectAction(String actionId, Bundle arguments, CancellationSignal cancellationSignal, Consumer<Bundle> resultListener)**: This is called to perform an action previously defined by the app.
- **void onPictureInPictureModeChanged(boolean isInPictureInPictureMode, Configuration newConfig)**: Called by the system when the activity changes to and from picture-in-picture mode.
- **void onPictureInPictureModeChanged(boolean isInPictureInPictureMode)**: This method was deprecated in API level 26. Use onPictureInPictureModeChanged(boolean, android.content.res.Configuration) instead.
- **boolean onPictureInPictureRequested()**: This method is called by the system in various cases where picture in picture mode should be entered if supported.
- **void onPictureInPictureUiStateChanged(PictureInPictureUiState pipState)**: Called by the system when the activity is in PiP and has state changes.
- **void onPostCreate(Bundle savedInstanceState, PersistableBundle persistentState)**: This is the same as onPostCreate(android.os.Bundle) but is called for activities created with the attribute R.attr.persistableMode set to persistAcrossReboots.
- **void onPrepareNavigateUpTaskStack(TaskStackBuilder builder)**: Prepare the synthetic task stack that will be generated during Up navigation from a different task.
- **boolean onPrepareOptionsMenu(Menu menu)**: Prepare the Screen's standard options menu to be displayed.
- **boolean onPreparePanel(int featureId, View view, Menu menu)**: Default implementation of Window.Callback.onPreparePanel(int, View, Menu) for activities.
- **void onProvideAssistContent(AssistContent outContent)**: This is called when the user is requesting an assist, to provide references to content related to the current activity.
- **void onProvideAssistData(Bundle data)**: This is called when the user is requesting an assist, to build a full Intent.ACTION_ASSIST Intent with all of the context of the current application.

- **void onProvideKeyboardShortcuts(List<KeyboardShortcutGroup> data, Menu menu, int deviceId)**: Called when Keyboard Shortcuts are requested for the current window.
- **Uri onProvideReferrer()**: Override to generate the desired referrer for the content currently being shown by the app.
- **void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults)**: Callback for the result from requesting permissions.
- **void onRequestPermissionsResult(int requestCode, String[] permissions, int[] grantResults, int deviceId)**: Callback for the result from requesting permissions.
- **void onRestoreInstanceState(Bundle savedInstanceState, PersistableBundle persistentState)**: This is the same as onRestoreInstanceState(android.os.Bundle) but is called for activities created with the attribute R.attr.persistableMode set to persistAcrossReboots.
- **Object onRetainNonConfigurationInstance()**: Called by the system, as part of destroying an activity due to a configuration change, when it is known that a new instance will immediately be created for the new configuration.
- **void onSaveInstanceState(Bundle outState, PersistableBundle outPersistentState)**: This is the same as onSaveInstanceState(Bundle) but is called for activities created with the attribute R.attr.persistableMode set to persistAcrossReboots.
- **boolean onSearchRequested(SearchEvent searchEvent)**: This hook is called when the user signals the desire to start a search.
- **boolean onSearchRequested()**: Called when the user signals the desire to start a search.
- **void onStateNotSaved()**: This method was deprecated in API level 29. starting with Build.VERSION_CODES.P onSaveInstanceState is called after onStop(), so this hint isn't accurate anymore: you should consider your state not saved in between onStart and onStop callbacks inclusively.
- **void onTopResumedActivityChanged(boolean isTopResumedActivity)**: Called when activity gets or loses the top resumed position in the system.
- **boolean onTouchEvent(MotionEvent event)**: Called when a touch screen event was not handled by any of the views inside of the activity.
- **boolean onTrackballEvent(MotionEvent event)**: Called when the trackball was moved and not handled by any of the views inside of the activity.
- **void onTrimMemory(int level)**: Called when the operating system has determined that it is a good time for a process to trim unneeded memory from its process.
- **void onUserInteraction()**: Called whenever a key, touch, or trackball event is dispatched to the activity.
- **void onVisibleBehindCanceled()**: This method was deprecated in API level 26. This method's functionality is no longer supported as of Build.VERSION_CODES.O and will be removed in a future release.
- **void onWindowAttributesChanged(WindowManager.LayoutParams params)**: This is called whenever the current window attributes change.
- **void onWindowFocusChanged(boolean hasFocus)**: Called when the current Window of the activity gains or loses focus.
- **ActionMode onWindowStartingActionMode(ActionMode.Callback callback, int type)**: Called when an action mode is being started for this window.

- **ActionMode onWindowStartingActionMode(ActionMode.Callback callback)**: Give the Activity a chance to control the UI for an action mode requested by the system.
- **void openContextMenu(View view)**: Programmatically opens the context menu for a particular view.
- **void openOptionsMenu()**: Programmatically opens the options menu.
- **void overrideActivityTransition(int overrideType, int enterAnim, int exitAnim, int backgroundColor)**: Customizes the animation and background color for activity transitions.
- **void overrideActivityTransition(int overrideType, int enterAnim, int exitAnim)**: Customizes the animation for activity transitions.
- **void overridePendingTransition(int enterAnim, int exitAnim)**: This method was deprecated in API level 34. Use overrideActivityTransition(int, int, int) instead.
- **void overridePendingTransition(int enterAnim, int exitAnim, int backgroundColor)**: This method was deprecated in API level 34. Use overrideActivityTransition(int, int, int, int) instead.
- **void postponeEnterTransition()**: Postpone the entering activity transition when Activity was started with ActivityOptions.makeSceneTransitionAnimation(Activity, android.util.Pair[]).
- **void recreate()**: Cause this Activity to be recreated with a new instance.
- **void registerActivityLifecycleCallbacks(Application.ActivityLifecycleCallbacks callback)**: Register an Application.ActivityLifecycleCallbacks instance that receives lifecycle callbacks for only this Activity.
- **void registerComponentCallbacks(ComponentCallbacks callback)**: Add a new ComponentCallbacks to the base application of the Context, which will be called at the same times as the ComponentCallbacks methods of activities and other components are called.
- **void registerForContextMenu(View view)**: Registers a context menu to be shown for the given view (multiple views can show the context menu).
- **void registerScreenCaptureCallback(Executor executor, Activity.ScreenCaptureCallback callback)**: Registers a screen capture callback for this activity.
- **boolean releaseInstance()**: Ask that the local app instance of this activity be released to free up its memory.
- **final void removeDialog(int id)**: This method was deprecated in API level 15. Use the new DialogFragment class with FragmentManager instead; this is also available on older platforms through the Android compatibility package.
- **void reportFullyDrawn()**: Report to the system that your app is now fully drawn, for diagnostic and optimization purposes.
- **DragAndDropPermissions requestDragAndDropPermissions(DragEvent event)**: Create DragAndDropPermissions object bound to this activity and controlling the access permissions for content URIs associated with the DragEvent.
- **void requestFullscreenMode(int request, OutcomeReceiver<Void, Throwable> approvalCallback)**: Request to put the activity into fullscreen.
- **final void requestOpenInBrowserEducation()**: Requests to show the 201cOpen in browser201d education.
- **final void requestPermissions(String[] permissions, int requestCode, int deviceId)**: Requests permissions to be granted to this application.

- **final void requestPermissions(String[] permissions, int requestCode)**: Requests permissions to be granted to this application.
- **final void requestShowKeyboardShortcuts()**: Request the Keyboard Shortcuts screen to show up.
- **boolean requestVisibleBehind(boolean visible)**: This method was deprecated in API level 26. This method's functionality is no longer supported as of Build.VERSION_CODES.O and will be removed in a future release.
- **final boolean requestWindowFeature(int featureId)**: Enable extended window features.
- **final <T extends View> T requireViewById(int id)**: Finds a view that was identified by the android:id XML attribute that was processed in onCreate(Bundle), or throws an IllegalArgumentException if the ID is invalid, or there is no matching view in the hierarchy.
- **final void runOnUiThread(Runnable action)**: Runs the specified action on the UI thread.
- **void setActionBar(Toolbar toolbar)**: Set a Toolbar to act as the ActionBar for this Activity window.
- **void setAllowCrossUidActivitySwitchFromBelow(boolean allowed)**: Specifies whether the activities below this one in the task can also start other activities or finish the task.
- **void setContentTransitionManager(TransitionManager tm)**: Set the TransitionManager to use for default transitions in this window.
- **void setContentView(View view, ViewGroup.LayoutParams params)**: Set the activity content to an explicit view.
- **void setContentView(View view)**: Set the activity content to an explicit view.
- **void setContentView(int layoutResID)**: Set the activity content from a layout resource.
- **final void setDefaultKeyMode(int mode)**: Select the default key handling for this activity.
- **void setEnterSharedElementCallback(SharedElementCallback callback)**: When ActivityOptions.makeSceneTransitionAnimation(Activity, android.view.View, String) was used to start an Activity, callback will be called to handle shared elements on the launched Activity.
- **void setExitSharedElementCallback(SharedElementCallback callback)**: When ActivityOptions.makeSceneTransitionAnimation(Activity, android.view.View, String) was used to start an Activity, callback will be called to handle shared elements on the launching Activity.
- **final void setFeatureDrawable(int featureId, Drawable drawable)**: Convenience for calling Window.setFeatureDrawable(int, Drawable).
- **final void setFeatureDrawableAlpha(int featureId, int alpha)**: Convenience for calling Window.setFeatureDrawableAlpha(int, int).
- **final void setFeatureDrawableResource(int featureId, int resId)**: Convenience for calling Window.setFeatureDrawableResource(int, int).
- **final void setFeatureDrawableUri(int featureId, Uri uri)**: Convenience for calling Window.setFeatureDrawableUri(int, Uri).
- **void setFinishOnTouchOutside(boolean finish)**: Sets whether this activity is finished when touched outside its window's bounds.

- **void setImmersive(boolean i)**: Adjust the current immersive mode setting.
- **void setInheritShowWhenLocked(boolean inheritShowWhenLocked)**: Specifies whether this Activity should be shown on top of the lock screen whenever the lockscreen is up and this activity has another activity behind it with the showWhenLock attribute set.
- **void setIntent(Intent newIntent)**: Changes the intent returned by getIntent().
- **void setIntent(Intent newIntent, ComponentCaller newCaller)**: Changes the intent returned by getIntent(), and ComponentCaller returned by getCaller().
- **void setLocusContext(LocusId locusId, Bundle bundle)**: Sets the LocusId for this activity.
- **final void setMediaController(MediaController controller)**: Sets a MediaController to send media keys and volume changes to.
- **void setPictureInPictureParams(PictureInPictureParams params)**: Updates the properties of the picture-in-picture activity, or sets it to be used later when enterPictureInPictureMode() is called.
- **final void setProgress(int progress)**: This method was deprecated in API level 24. No longer supported starting in API 21.
- **final void setProgressBarIndeterminate(boolean indeterminate)**: This method was deprecated in API level 24. No longer supported starting in API 21.
- **final void setProgressBarIndeterminateVisibility(boolean visible)**: This method was deprecated in API level 24. No longer supported starting in API 21.
- **final void setProgressBarVisibility(boolean visible)**: This method was deprecated in API level 24. No longer supported starting in API 21.
- **void setRecentsScreenshotEnabled(boolean enabled)**: If set to false, this indicates to the system that it should never take a screenshot of the activity to be used as a representation in recents screen.
- **void setRequestedOrientation(int requestedOrientation)**: Change the desired orientation of this activity.
- **final void setResult(int resultCode, Intent data)**: Call this to set the result that your activity will return to its caller.
- **final void setResult(int resultCode)**: Call this to set the result that your activity will return to its caller.
- **final void setSecondaryProgress(int secondaryProgress)**: This method was deprecated in API level 24. No longer supported starting in API 21.
- **void setShouldDockBigOverlays(boolean shouldDockBigOverlays)**: Specifies a preference to dock big overlays like the expanded picture-in-picture on TV (see PictureInPictureParams.Builder.setExpandedAspectRatio).
- **void setShowWhenLocked(boolean showWhenLocked)**: Specifies whether an Activity should be shown on top of the lock screen whenever the lockscreen is up and the activity is resumed.
- **void setTaskDescription(ActivityManager.TaskDescription taskDescription)**: Sets information describing the task with this activity for presentation inside the Recents System UI.
- **void setTheme(int resid)**: Set the base theme for this context.
- **void setTitle(CharSequence title)**: Change the title associated with this activity.
- **void setTitle(int titleId)**: Change the title associated with this activity.

- **void setTitleColor(int textColor)**: This method was deprecated in API level 21. Use action bar styles instead.
- **boolean setTranslucent(boolean translucent)**: Convert an activity, which particularly with R.attr.windowIsTranslucent or R.attr.windowIsFloating attribute, to a fullscreen opaque activity, or convert it from opaque back to translucent.
- **void setTurnScreenOn(boolean turnScreenOn)**: Specifies whether the screen should be turned on when the Activity is resumed.
- **void setVisible(boolean visible)**: Control whether this activity's main window is visible.
- **final void setVolumeControlStream(int streamType)**: Suggests an audio stream whose volume should be changed by the hardware volume controls.
- **void setVrModeEnabled(boolean enabled, ComponentName requestedComponent)**: Enable or disable virtual reality (VR) mode for this Activity.
- **boolean shouldDockBigOverlays()**: Returns whether big overlays should be docked next to the activity as set by setShouldDockBigOverlays(boolean).
- **boolean shouldShowRequestPermissionRationale(String permission)**: Gets whether you should show UI with rationale before requesting a permission.
- **boolean shouldShowRequestPermissionRationale(String permission, int deviceId)**: Gets whether you should show UI with rationale before requesting a permission.
- **boolean shouldUpRecreateTask(Intent targetIntent)**: Returns true if the app should recreate the task when navigating 'up' from this activity by using targetIntent.
- **boolean showAssist(Bundle args)**: Ask to have the current assistant shown to the user.
- **final boolean showDialog(int id, Bundle args)**: This method was deprecated in API level 15. Use the new DialogFragment class with FragmentManager instead; this is also available on older platforms through the Android compatibility package.
- **final void showDialog(int id)**: This method was deprecated in API level 15. Use the new DialogFragment class with FragmentManager instead; this is also available on older platforms through the Android compatibility package.
- **void showLockTaskEscapeMessage()**: Shows the user the system defined message for telling the user how to exit lock task mode.
- **ActionMode startActionMode(ActionMode.Callback callback, int type)**: Start an action mode of the given type.
- **ActionMode startActionMode(ActionMode.Callback callback)**: Start an action mode of the default type ActionMode.TYPE_PRIMARY.
- **void startActivities(Intent[] intents, Bundle options)**: Launch a new activity.
- **void startActivities(Intent[] intents)**: Same as startActivities(android.content.Intent[], android.os.Bundle) with no options specified.
- **void startActivity(Intent intent)**: Same as startActivity(android.content.Intent, android.os.Bundle) with no options specified.
- **void startActivity(Intent intent, Bundle options)**: Launch a new activity.
- **void startActivityForResult(Intent intent, int requestCode)**: Same as calling startActivityForResult(android.content.Intent, int, android.os.Bundle) with no options.

- **void startActivityForResult(Intent intent, int requestCode, Bundle options)**: Launch an activity for which you would like a result when it finished.
- **void startActivityFromChild(Activity child, Intent intent, int requestCode)**: This method was deprecated in API level 30. Use androidx.fragment.app.FragmentActivity#startActivityFromFragment(androidx.fragment.app.Fragment,Intent,int)
- **void startActivityFromChild(Activity child, Intent intent, int requestCode, Bundle options)**: This method was deprecated in API level 30. Use androidx.fragment.app.FragmentActivity#startActivityFromFragment(androidx.fragment.app.Fragment,Intent,int,Bundle)
- **void startActivityFromFragment(Fragment fragment, Intent intent, int requestCode, Bundle options)**: This method was deprecated in API level 28. Use androidx.fragment.app.FragmentActivity#startActivityFromFragment(androidx.fragment.app.Fragment,Intent,int,Bundle)
- **void startActivityFromFragment(Fragment fragment, Intent intent, int requestCode)**: This method was deprecated in API level 28. Use androidx.fragment.app.FragmentActivity#startActivityFromFragment(androidx.fragment.app.Fragment,Intent,int)
- **boolean startActivityIfNeeded(Intent intent, int requestCode, Bundle options)**: A special variation to launch an activity only if a new activity instance is needed to handle the given Intent.
- **boolean startActivityIfNeeded(Intent intent, int requestCode)**: Same as calling startActivityIfNeeded(android.content.Intent, int, android.os.Bundle) with no options.
- **void startIntentSender(IntentSender intent, Intent fillInIntent, int flagsMask, int flagsValues, int extraFlags)**: Same as calling startIntentSender(android.content.IntentSender, android.content.Intent, int, int, int, android.os.Bundle) with no options.
- **void startIntentSender(IntentSender intent, Intent fillInIntent, int flagsMask, int flagsValues, int extraFlags, Bundle options)**: Like startActivity(android.content.Intent, android.os.Bundle), but taking a IntentSender to start; see startIntentSenderForResult(android.content.IntentSender, int, android.content.Intent, int, int, int, android.os.Bundle) for more information.
- **void startIntentSenderForResult(IntentSender intent, int requestCode, Intent fillInIntent, int flagsMask, int flagsValues, int extraFlags)**: Same as calling startIntentSenderForResult(android.content.IntentSender, int, android.content.Intent, int, int, int, android.os.Bundle) with no options.
- **void startIntentSenderForResult(IntentSender intent, int requestCode, Intent fillInIntent, int flagsMask, int flagsValues, int extraFlags, Bundle options)**: Like startActivityForResult(android.content.Intent, int), but allowing you to use a IntentSender to describe the activity to be started.
- **void startIntentSenderFromChild(Activity child, IntentSender intent, int requestCode, Intent fillInIntent, int flagsMask, int flagsValues, int extraFlags, Bundle options)**: This method was deprecated in API level 30. Use startIntentSenderForResult(android.content.IntentSender, int, android.content.Intent, int, int, int, android.os.Bundle) instead.
- **void startIntentSenderFromChild(Activity child, IntentSender intent, int requestCode, Intent fillInIntent, int flagsMask, int flagsValues, int extraFlags)**: This method was deprecated in API level 30. Use startIntentSenderForResult(android.content.IntentSender, int, android.content.Intent, int, int, int) instead.

- **void startLocalVoiceInteraction(Bundle privateOptions)**: Starts a local voice interaction session.
- **void startLockTask()**: Request to put this activity in a mode where the user is locked to a restricted set of applications.
- **void startManagingCursor(Cursor c)**: This method was deprecated in API level 15. Use the new CursorLoader class with LoaderManager instead; this is also available on older platforms through the Android compatibility package.
- **boolean startNextMatchingActivity(Intent intent, Bundle options)**: Special version of starting an activity, for use when you are replacing other activity components.
- **boolean startNextMatchingActivity(Intent intent)**: Same as calling startNextMatchingActivity(android.content.Intent, android.os.Bundle) with no options.
- **void startPostponedEnterTransition()**: Begin postponed transitions after postponeEnterTransition() was called.
- **void startSearch(String initialQuery, boolean selectInitialQuery, Bundle appSearchData, boolean globalSearch)**: This hook is called to launch the search UI.
- **void stopLocalVoiceInteraction()**: Request to terminate the current voice interaction that was previously started using startLocalVoiceInteraction(android.os.Bundle).
- **void stopLockTask()**: Stop the current task from being locked.
- **void stopManagingCursor(Cursor c)**: This method was deprecated in API level 15. Use the new CursorLoader class with LoaderManager instead; this is also available on older platforms through the Android compatibility package.
- **void takeKeyEvents(boolean get)**: Request that key events come to this activity.
- **void triggerSearch(String query, Bundle appSearchData)**: Similar to startSearch(String, boolean, Bundle, boolean), but actually fires off the search query after invoking the search dialog.
- **void unregisterActivityLifecycleCallbacks(Application.ActivityLifecycleCallbacks callback)**: Unregister an Application.ActivityLifecycleCallbacks previously registered with registerActivityLifecycleCallbacks(ActivityLifecycleCallbacks).
- **void unregisterComponentCallbacks(ComponentCallbacks callback)**: Remove a ComponentCallbacks object that was previously registered with registerComponentCallbacks(android.content.ComponentCallbacks).
- **void unregisterForContextMenu(View view)**: Prevents a context menu to be shown for the given view.
- **void unregisterScreenCaptureCallback(Activity.ScreenCaptureCallback callback)**: Unregisters a screen capture callback for this surface.

- **Protected methods**
  - **void attachBaseContext(Context newBase)**: Set the base context for this ContextWrapper.
  - **void onActivityResult(int requestCode, int resultCode, Intent data)**: Called when an activity you launched exits, giving you the requestCode you started it with, the resultCode it returned, and any additional data from it.

- **void onApplyThemeResource(Resources.Theme theme, int resid, boolean first)**: Called by setTheme(Theme) and getTheme() to apply a theme resource to the current Theme object.
- **void onChildTitleChanged(Activity childActivity, CharSequence title)**:
- **void onCreate(Bundle savedInstanceState)**: Called when the activity is starting.
- **Dialog onCreateDialog(int id)**: This method was deprecated in API level 15. Old no-arguments version of onCreateDialog(int, android.os.Bundle).
- **Dialog onCreateDialog(int id, Bundle args)**: This method was deprecated in API level 15. Use the new DialogFragment class with FragmentManager instead; this is also available on older platforms through the Android compatibility package.
- **void onDestroy()**: Perform any final cleanup before an activity is destroyed.
- **void onNewIntent(Intent intent)**: This is called for activities that set launchMode to "singleTop" in their package, or if a client used the Intent.FLAG_ACTIVITY_SINGLE_TOP flag when calling startActivity(Intent).
- **void onPause()**: Called as part of the activity lifecycle when the user no longer actively interacts with the activity, but it is still visible on screen.
- **void onPostCreate(Bundle savedInstanceState)**: Called when activity start-up is complete (after onStart() and onRestoreInstanceState(Bundle) have been called).
- **void onPostResume()**: Called when activity resume is complete (after onResume() has been called).
- **void onPrepareDialog(int id, Dialog dialog, Bundle args)**: This method was deprecated in API level 15. Use the new DialogFragment class with FragmentManager instead; this is also available on older platforms through the Android compatibility package.
- **void onPrepareDialog(int id, Dialog dialog)**: This method was deprecated in API level 15. Old no-arguments version of onPrepareDialog(int, android.app.Dialog, android.os.Bundle).
- **void onRestart()**: Called after onStop() when the current activity is being re-displayed to the user (the user has navigated back to it).
- **void onRestoreInstanceState(Bundle savedInstanceState)**: This method is called after onStart() when the activity is being re-initialized from a previously saved state, given here in savedInstanceState.
- **void onResume()**: Called after onRestoreInstanceState(Bundle), onRestart(), or onPause().
- **void onSaveInstanceState(Bundle outState)**: Called to retrieve per-instance state from an activity before being killed so that the state can be restored in onCreate(Bundle) or onRestoreInstanceState(Bundle) (the Bundle populated by this method will be passed to both).
- **void onStart()**: Called after onCreate(Bundle) — or after onRestart() when the activity had been stopped, but is now again being displayed to the user.
- **void onStop()**: Called when you are no longer visible to the user.
- **void onTitleChanged(CharSequence title, int color)**:
- **void onUserLeaveHint()**: Called as part of the activity lifecycle when an activity is about to go into the background as the result of user choice.

- **Fields**

- **protected static final int[] FOCUSED_STATE_SET**:

- **Constants**
  - **int DEFAULT_KEYS_DIALER**: Use with setDefaultKeyMode(int) to launch the dialer during default key handling.
  - **int DEFAULT_KEYS_DISABLE**: Use with setDefaultKeyMode(int) to turn off default handling of keys.
  - **int DEFAULT_KEYS_SEARCH_GLOBAL**: Use with setDefaultKeyMode(int) to specify that unhandled keystrokes will start a global search (typically web search, but some platforms may define alternate methods for global search)

    See android.app.SearchManager for more details.
  - **int DEFAULT_KEYS_SEARCH_LOCAL**: Use with setDefaultKeyMode(int) to specify that unhandled keystrokes will start an application-defined search.
  - **int DEFAULT_KEYS_SHORTCUT**: Use with setDefaultKeyMode(int) to execute a menu shortcut in default key handling.
  - **int FULLSCREEN_MODE_REQUEST_ENTER**: Request type of requestFullscreenMode(int, android.os.OutcomeReceiver), to request enter fullscreen mode from multi-window mode.
  - **int FULLSCREEN_MODE_REQUEST_EXIT**: Request type of requestFullscreenMode(int, android.os.OutcomeReceiver), to request exiting the requested fullscreen mode and restore to the previous multi-window mode.
  - **int OVERRIDE_TRANSITION_CLOSE**: Request type of overrideActivityTransition(int, int, int) or overrideActivityTransition(int, int, int, int), to override the closing transition.
  - **int OVERRIDE_TRANSITION_OPEN**: Request type of overrideActivityTransition(int, int, int) or overrideActivityTransition(int, int, int, int), to override the opening transition.
  - **int RESULT_CANCELED**: Standard activity result: operation canceled.
  - **int RESULT_FIRST_USER**: Start of user-defined activity results.
  - **int RESULT_OK**: Standard activity result: operation succeeded.

2.2 View

- **Hierarchy**

  java.lang.Object → android.view.View

- **Include**

```
android.view.View
```

- **Constructors**

```
View(Context context)
View(Context context, AttributeSet attrs)
View(Context context, AttributeSet attrs, int defStyleAttr)
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.
  - **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.
- **final int getMeasuredHeight()**: Like getMeasuredHeightAndState(), but only returns the raw height component (that is the result is masked by MEASURED$_{S}IZE_{M}ASK$).
- **final int getMeasuredHeightAndState()**: Return the full height measurement information for this view as computed by the most recent call to measure(int, int).
- **final int getMeasuredState()**: Return only the state bits of getMeasuredWidthAndState() and getMeasuredHeightAndState(), combined into one integer.
- **final int getMeasuredWidth()**: Like getMeasuredWidthAndState(), but only returns the raw width component (that is the result is masked by MEASURED$_{S}IZE_{M}ASK$).
- **final int getMeasuredWidthAndState()**: Return the full width measurement information for this view as computed by the most recent call to measure(int, int).
- **int getMinimumHeight()**: Returns the minimum height of the view.
- **int getMinimumWidth()**: Returns the minimum width of the view.
- **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.
- **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.
- **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_FOCUS**: 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_CURSOR**: 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_PREFERENCE**: 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.

2.3 ViewGroup

- **Hierarchy**

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

- **Include**

```
android.view.ViewGroup
```

- **Constructors**

```
ViewGroup(Context context)
ViewGroup(Context context, AttributeSet attrs)
ViewGroup(Context context, AttributeSet attrs, int
    ↪ defStyleAttr)
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 previously-FocusedRect)**: 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.

2.4 ViewGroup.LayoutParams

- **Hierarchy**

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

- **Include**

```
android.view.ViewGroup.LayoutParams
```

- **Constructors**

```
LayoutParams(Context c, AttributeSet attrs)
LayoutParams(ViewGroup.LayoutParams source)
LayoutParams(int width, int height)
```

- **Public methods**
  - **void resolveLayoutDirection(int layoutDirection)**: Resolve layout parameters depending on the layout direction.
- **Protected methods**
  - **void setBaseAttributes(TypedArray a, int widthAttr, int heightAttr)**: Extracts the layout parameters from the supplied attributes.
- **Fields**
  - **public int height**: Information about how tall the view wants to be.
  - **public LayoutAnimationController.AnimationParameters layoutAnimationParameters**: Used to animate layouts.
  - **public int width**: Information about how wide the view wants to be.
- **Constants**
  - **int FILL_PARENT**: Special value for the height or width requested by a View.
  - **int MATCH_PARENT**: Special value for the height or width requested by a View.
  - **int WRAP_CONTENT**: Special value for the height or width requested by a View.

2.5 ViewGroup.MarginLayoutParams

- **Hierarchy**

  java.lang.Object → android.view.ViewGroup.LayoutParams → android.view.ViewGroup.MarginLayoutParams

- **Include**

```
android.view.ViewGroup.MarginLayoutParams
```

- **Constructors**

```
MarginLayoutParams(Context c, AttributeSet attrs)
MarginLayoutParams(ViewGroup.LayoutParams source)
MarginLayoutParams(ViewGroup.MarginLayoutParams source)
MarginLayoutParams(int width, int height)
```

- **Public methods**
  - **int getLayoutDirection()**: Retuns the layout direction.
  - **int getMarginEnd()**: Returns the end margin in pixels.
  - **int getMarginStart()**: Returns the start margin in pixels.
  - **boolean isMarginRelative()**: Check if margins are relative.
  - **void resolveLayoutDirection(int layoutDirection)**: This will be called by View.requestLayout().
  - **void setLayoutDirection(int layoutDirection)**: Set the layout direction
  - **void setMarginEnd(int end)**: Sets the relative end margin.
  - **void setMarginStart(int start)**: Sets the relative start margin.
  - **void setMargins(int left, int top, int right, int bottom)**: Sets the margins, in pixels.
- **Fields**
  - **public int bottomMargin**: The bottom margin in pixels of the child.
  - **public int leftMargin**: The left margin in pixels of the child.
  - **public int rightMargin**: The right margin in pixels of the child.
  - **public int topMargin**: The top margin in pixels of the child.

2.6 Color

- **Hierarchy**

java.lang.Object → android.graphics.Color

- **Include**

```
android.graphics.Color
```

- **Constructors**

```
Color()
```

- **Public methods**
  - **static int HSVToColor(float[] hsv)**: Convert HSV components to an ARGB color.
  - **static int HSVToColor(int alpha, float[] hsv)**: Convert HSV components to an ARGB color.
  - **static void RGBToHSV(int red, int green, int blue, float[] hsv)**: Convert RGB components to HSV.
  - **static int alpha(int color)**: Return the alpha component of a color int.
  - **float alpha()**: Returns the value of the alpha component in the range .
  - **static float alpha(long color)**: Returns the alpha component encoded in the specified color long.
  - **static int argb(int alpha, int red, int green, int blue)**: Return a color-int from alpha, red, green, blue components.
  - **static int argb(float alpha, float red, float green, float blue)**: Return a color-int from alpha, red, green, blue float components in the range .
  - **static int blue(int color)**: Return the blue component of a color int.
  - **float blue()**: Returns the value of the blue component in the range defined by this color's color space (see ColorSpace.getMinValue(int) and ColorSpace.getMaxValue(int)).
  - **static float blue(long color)**: Returns the blue component encoded in the specified color long.
  - **static ColorSpace colorSpace(long color)**: Returns the color space encoded in the specified color long.
  - **static void colorToHSV(int color, float[] hsv)**: Convert the ARGB color to its HSV components.
  - **static long convert(long color, ColorSpace.Connector connector)**: Converts the specified color long from a color space to another using the specified color space connector.
  - **Color convert(ColorSpace colorSpace)**: Converts this color from its color space to the specified color space.
  - **static long convert(int color, ColorSpace colorSpace)**: Converts the specified ARGB color int from the sRGB color space into the specified destination color space.

- **static long convert(float r, float g, float b, float a, ColorSpace source, ColorSpace destination)**: Converts the specified 3 component color from the source color space to the destination color space.
- **static long convert(float r, float g, float b, float a, ColorSpace.Connector connector)**: Converts the specified 3 component color from a color space to another using the specified color space connector.
- **static long convert(long color, ColorSpace colorSpace)**: Converts the specified color long from its color space into the specified destination color space.
- **boolean equals(Object o)**: Indicates whether some other object is "equal to" this one.
- **ColorSpace getColorSpace()**: Returns this color's color space.
- **float getComponent(int component)**: Returns the value of the specified component in the range defined by this color's color space (see ColorSpace.getMinValue(int) and ColorSpace.getMaxValue(int)).
- **int getComponentCount()**: Returns the number of components that form a color value according to this color space's color model, plus one extra component for alpha.
- **float[] getComponents()**: Returns this color's components as a new array.
- **float[] getComponents(float[] components)**: Copies this color's components in the supplied array.
- **ColorSpace.Model getModel()**: Returns the color model of this color.
- **static float green(long color)**: Returns the green component encoded in the specified color long.
- **float green()**: Returns the value of the green component in the range defined by this color's color space (see ColorSpace.getMinValue(int) and ColorSpace.getMaxValue(int)).
- **static int green(int color)**: Return the green component of a color int.
- **int hashCode()**: Returns a hash code value for the object.
- **static boolean isInColorSpace(long color, ColorSpace colorSpace)**: Indicates whether the specified color is in the specified color space.
- **boolean isSrgb()**: Indicates whether this color is in the sRGB color space.
- **static boolean isSrgb(long color)**: Indicates whether the specified color is in the sRGB color space.
- **static boolean isWideGamut(long color)**: Indicates whether the specified color is in a wide-gamut color space.
- **boolean isWideGamut()**: Indicates whether this color color is in a wide-gamut color space.
- **static float luminance(long color)**: Returns the relative luminance of a color.
- **static float luminance(int color)**: Returns the relative luminance of a color.
- **float luminance()**: Returns the relative luminance of this color.
- **static long pack(int color)**: Converts the specified ARGB color int to an RGBA color long in the sRGB color space.
- **static long pack(float red, float green, float blue, float alpha)**: Packs the sRGB color defined by the specified red, green, blue and alpha component values into an RGBA color long in the sRGB color space.

- **static long pack(float red, float green, float blue, float alpha, ColorSpace colorSpace)**: Packs the 3 component color defined by the specified red, green, blue and alpha component values into a color long in the specified color space.
- **static long pack(float red, float green, float blue)**: Packs the sRGB color defined by the specified red, green and blue component values into an RGBA color long in the sRGB color space.
- **long pack()**: Packs this color into a color long.
- **static int parseColor(String colorString)**: Parse the color string, and return the corresponding color-int.
- **float red()**: Returns the value of the red component in the range defined by this color's color space (see ColorSpace.getMinValue(int) and ColorSpace.getMaxValue(int)).
- **static float red(long color)**: Returns the red component encoded in the specified color long.
- **static int red(int color)**: Return the red component of a color int.
- **static int rgb(float red, float green, float blue)**: Return a color-int from red, green, blue float components in the range .
- **static int rgb(int red, int green, int blue)**: Return a color-int from red, green, blue components.
- **int toArgb()**: Converts this color to an ARGB color int.
- **static int toArgb(long color)**: Converts the specified color long to an ARGB color int.
- **String toString()**: Returns a string representation of the object.
- **static Color valueOf(float r, float g, float b)**: Creates a new opaque Color in the sRGB color space with the specified red, green and blue component values.
- **static Color valueOf(float r, float g, float b, float a)**: Creates a new Color in the sRGB color space with the specified red, green, blue and alpha component values.
- **static Color valueOf(int color)**: Creates a new Color instance from an ARGB color int.
- **static Color valueOf(float[] components, ColorSpace colorSpace)**: Creates a new Color in the specified color space with the specified component values.
- **static Color valueOf(long color)**: Creates a new Color instance from a color long.
- **static Color valueOf(float r, float g, float b, float a, ColorSpace colorSpace)**: Creates a new Color in the specified color space with the specified red, green, blue and alpha component values.

- **Constants**
  - **int BLACK**:
  - **int BLUE**:
  - **int CYAN**:
  - **int DKGRAY**:
  - **int GRAY**:
  - **int GREEN**:
  - **int LTGRAY**:

- **int MAGENTA**:
- **int RED**:
- **int TRANSPARENT**:
- **int WHITE**:
- **int YELLOW**:

2.7 Context

- **Hierarchy**

  java.lang.Object → android.content.Context

- **Include**

```
android.content.Context
```

- **Constructors**

```
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.

2.8 Configuration

- **Getting a Configuration object**: First, we get a Resources reference with getResources() from the activity class, then we can call getConfiguration() on that reference

```
Configuration config = getResources().getConfiguration();
```

- **Hierarchy**

java.lang.Object → android.content.res.Configuration

- **Include**

```
android.content.res.Configuration
```

- **Constructors**

```
Configuration()
Configuration(Configuration o)
```

- **Public methods**
  - **int compareTo(Configuration that)**:
  - **int describeContents()**: Parcelable methods
  - **int diff(Configuration delta)**: Return a bit mask of the differences between this Configuration object and the given one.
  - **boolean equals(Configuration that)**:
  - **boolean equals(Object that)**: Indicates whether some other object is "equal to" this one.
  - **static Configuration generateDelta(Configuration base, Configuration change)**: Generate a delta Configuration between base and change.
  - **int getGrammaticalGender()**: Returns the user preference for the grammatical gender.
  - **int getLayoutDirection()**: Return the layout direction.
  - **LocaleList getLocales()**: Get the locale list.
  - **int hashCode()**: Returns a hash code value for the object.
  - **boolean isLayoutSizeAtLeast(int size)**: Check if the Configuration's current screenLayout is at least the given size.
  - **boolean isNightModeActive()**: Retuns whether the configuration is in night mode
  - **boolean isScreenHdr()**: Return whether the screen has a high dynamic range.
  - **boolean isScreenRound()**: Return whether the screen has a round shape.
  - **boolean isScreenWideColorGamut()**: Return whether the screen has a wide color gamut and wide color gamut rendering is supported by this device.
  - **static boolean needNewResources(int configChanges, int interestingChanges)**: Determines if a new resource needs to be loaded from the bit set of configuration changes returned by updateFrom(android.content.res.Configuration).

- **void readFromParcel(Parcel source)**:
- **void setLayoutDirection(Locale loc)**: Set the layout direction from a Locale.
- **void setLocale(Locale loc)**: Set the locale list to a list of just one locale.
- **void setLocales(LocaleList locales)**: Set the locale list.
- **void setTo(Configuration o)**: Sets the fields in this object to those in the given Configuration.
- **void setToDefaults()**: Set this object to the system defaults.
- **String toString()**: Returns a string representation of the object.
- **int updateFrom(Configuration delta)**: Copies the fields from delta into this Configuration object, keeping track of which ones have changed.
- **void writeToParcel(Parcel dest, int flags)**: Flatten this object in to a Parcel.

- **Fields**
  - **public static final Creator<Configuration> CREATOR**:
  - **public int colorMode**: Bit mask of color capabilities of the screen.
  - **public int densityDpi**: The target screen density being rendered to, corresponding to density resource qualifier.
  - **public float fontScale**: Current user preference for the scaling factor for fonts, relative to the base density scaling.
  - **public int fontWeightAdjustment**: Adjustment in text font weight.
  - **public int hardKeyboardHidden**: A flag indicating whether the hard keyboard has been hidden.
  - **public int keyboard**: The kind of keyboard attached to the device.
  - **public int keyboardHidden**: A flag indicating whether any keyboard is available.
  - **public Locale locale**: This field was deprecated in API level 24. Do not set or read this directly. Use getLocales() and setLocales(android.os.LocaleList). If only the primary locale is needed, getLocales().get(0) is now the preferred accessor.
  - **public int mcc**: IMSI MCC (Mobile Country Code), corresponding to mcc resource qualifier.
  - **public int mnc**: IMSI MNC (Mobile Network Code), corresponding to mnc resource qualifier.
  - **public int navigation**: The kind of navigation method available on the device.
  - **public int navigationHidden**: A flag indicating whether any 5-way or DPAD navigation available.
  - **public int orientation**: Overall orientation of the screen.
  - **public int screenHeightDp**: The height of the available screen space in dp units.
  - **public int screenLayout**: Bit mask of overall layout of the screen.
  - **public int screenWidthDp**: The width of the available screen space in dp units.
  - **public int smallestScreenWidthDp**: The smallest screen size an application will see in normal operation.
  - **public int touchscreen**: The kind of touch screen attached to the device.
  - **public int uiMode**: Bit mask of the ui mode.

- **Constants**
  - **int COLOR_MODE_HDR_MASK**: Constant for colorMode: bits that encode the dynamic range of the screen.
  - **int COLOR_MODE_HDR_NO**: Constant for colorMode: a COLOR_MODE_HDR_MASK value indicating that the screen is not HDR (low/standard dynamic range).
  - **int COLOR_MODE_HDR_SHIFT**: Constant for colorMode: bits shift to get the screen dynamic range.
  - **int COLOR_MODE_HDR_UNDEFINED**: Constant for colorMode: a COLOR_MODE_HDR_MASK value indicating that it is unknown whether or not the screen is HDR.
  - **int COLOR_MODE_HDR_YES**: Constant for colorMode: a COLOR_MODE_HDR_MASK value indicating that the screen is HDR (dynamic range).
  - **int COLOR_MODE_UNDEFINED**: Constant for colorMode: a value indicating that the color mode is undefined
  - **int COLOR_MODE_WIDE_COLOR_GAMUT_MASK**: Constant for colorMode: bits that encode whether the screen is wide gamut.
  - **int COLOR_MODE_WIDE_COLOR_GAMUT_NO**: Constant for colorMode: a COLOR_MODE_WIDE_COLOR_GAMUT_MASK value indicating that the screen is not wide gamut.
  - **int COLOR_MODE_WIDE_COLOR_GAMUT_UNDEFINED**: Constant for colorMode: a COLOR_MODE_WIDE_COLOR_GAMUT_MASK value indicating that it is unknown whether or not the screen is wide gamut.
  - **int COLOR_MODE_WIDE_COLOR_GAMUT_YES**: Constant for colorMode: a COLOR_MODE_WIDE_COLOR_GAMUT_MASK value indicating that the screen is wide gamut.
  - **int DENSITY_DPI_UNDEFINED**: Default value for densityDpi indicating that no width has been specified.
  - **int FONT_WEIGHT_ADJUSTMENT_UNDEFINED**: An undefined fontWeightAdjustment.
  - **int GRAMMATICAL_GENDER_FEMININE**: Constant for grammatical gender: to indicate the terms of address the user preferred in an application is feminine.
  - **int GRAMMATICAL_GENDER_MASCULINE**: Constant for grammatical gender: to indicate the terms of address the user preferred in an application is masculine.
  - **int GRAMMATICAL_GENDER_NEUTRAL**: Constant for grammatical gender: to indicate the terms of address the user preferred in an application is neuter.
  - **int GRAMMATICAL_GENDER_NOT_SPECIFIED**: Constant for grammatical gender: to indicate the user has not specified the terms of address for the application.
  - **int HARDKEYBOARDHIDDEN_NO**: Constant for hardKeyboardHidden, value corresponding to the physical keyboard being exposed.
  - **int HARDKEYBOARDHIDDEN_UNDEFINED**: Constant for hardKeyboardHidden: a value indicating that no value has been set.
  - **int HARDKEYBOARDHIDDEN_YES**: Constant for hardKeyboardHidden, value corresponding to the physical keyboard being hidden.

- **int KEYBOARDHIDDEN_NO**: Constant for keyboardHidden, value corresponding to the keysexposed resource qualifier.
- **int KEYBOARDHIDDEN_UNDEFINED**: Constant for keyboardHidden: a value indicating that no value has been set.
- **int KEYBOARDHIDDEN_YES**: Constant for keyboardHidden, value corresponding to the keyshidden resource qualifier.
- **int KEYBOARD_12KEY**: Constant for keyboard, value corresponding to the 12key resource qualifier.
- **int KEYBOARD_NOKEYS**: Constant for keyboard, value corresponding to the nokeys resource qualifier.
- **int KEYBOARD_QWERTY**: Constant for keyboard, value corresponding to the qwerty resource qualifier.
- **int KEYBOARD_UNDEFINED**: Constant for keyboard: a value indicating that no value has been set.
- **int MNC_ZERO**: Constant used to to represent MNC (Mobile Network Code) zero.
- **int NAVIGATIONHIDDEN_NO**: Constant for navigationHidden, value corresponding to the navexposed resource qualifier.
- **int NAVIGATIONHIDDEN_UNDEFINED**: Constant for navigationHidden: a value indicating that no value has been set.
- **int NAVIGATIONHIDDEN_YES**: Constant for navigationHidden, value corresponding to the navhidden resource qualifier.
- **int NAVIGATION_DPAD**: Constant for navigation, value corresponding to the dpad resource qualifier.
- **int NAVIGATION_NONAV**: Constant for navigation, value corresponding to the nonav resource qualifier.
- **int NAVIGATION_TRACKBALL**: Constant for navigation, value corresponding to the trackball resource qualifier.
- **int NAVIGATION_UNDEFINED**: Constant for navigation: a value indicating that no value has been set.
- **int NAVIGATION_WHEEL**: Constant for navigation, value corresponding to the wheel resource qualifier.
- **int ORIENTATION_LANDSCAPE**: Constant for orientation, value corresponding to the land resource qualifier.
- **int ORIENTATION_PORTRAIT**: Constant for orientation, value corresponding to the port resource qualifier.
- **int ORIENTATION_SQUARE**: This constant was deprecated in API level 16. Not currently supported or used.
- **int ORIENTATION_UNDEFINED**: Constant for orientation: a value indicating that no value has been set.
- **int SCREENLAYOUT_LAYOUTDIR_LTR**: Constant for screenLayout: a SCREENLAYOUT_LAYOUTDIR_MASK value indicating that a layout dir has been set to LTR.
- **int SCREENLAYOUT_LAYOUTDIR_MASK**: Constant for screenLayout: bits that encode the layout direction.
- **int SCREENLAYOUT_LAYOUTDIR_RTL**: Constant for screenLayout: a SCREENLAYOUT_LAYOUTDIR_MASK value indicating that a layout dir has been set to RTL.

- **int SCREENLAYOUT_LAYOUTDIR_SHIFT**: Constant for screenLayout: bits shift to get the layout direction.
- **int SCREENLAYOUT_LAYOUTDIR_UNDEFINED**: Constant for screenLayout: a SCREENLAYOUT_LAYOUTDIR_MASK value indicating that no layout dir has been set.
- **int SCREENLAYOUT_LONG_MASK**: Constant for screenLayout: bits that encode the aspect ratio.
- **int SCREENLAYOUT_LONG_NO**: Constant for screenLayout: a SCREENLAYOUT_LONG_MASK value that corresponds to the notlong resource qualifier.
- **int SCREENLAYOUT_LONG_UNDEFINED**: Constant for screenLayout: a SCREENLAYOUT_LONG_MASK value indicating that no size has been set.
- **int SCREENLAYOUT_LONG_YES**: Constant for screenLayout: a SCREENLAYOUT_LONG_MASK value that corresponds to the long resource qualifier.
- **int SCREENLAYOUT_ROUND_MASK**: Constant for screenLayout: bits that encode roundness of the screen.
- **int SCREENLAYOUT_ROUND_NO**: Constant for screenLayout: a SCREENLAYOUT_ROUND_MASK value indicating that the screen does not have a rounded shape.
- **int SCREENLAYOUT_ROUND_UNDEFINED**: Constant for screenLayout: a SCREENLAYOUT_ROUND_MASK value indicating that it is unknown whether or not the screen has a round shape.
- **int SCREENLAYOUT_ROUND_YES**: Constant for screenLayout: a SCREENLAYOUT_ROUND_MASK value indicating that the screen has a rounded shape.
- **int SCREENLAYOUT_SIZE_LARGE**: Constant for screenLayout: a SCREENLAYOUT_SIZE_MASK value indicating the screen is at least approximately 480x640 dp units, corresponds to the large resource qualifier.
- **int SCREENLAYOUT_SIZE_MASK**: Constant for screenLayout: bits that encode the size.
- **int SCREENLAYOUT_SIZE_NORMAL**: Constant for screenLayout: a SCREENLAYOUT_SIZE_MASK value indicating the screen is at least approximately 320x470 dp units, corresponds to the normal resource qualifier.
- **int SCREENLAYOUT_SIZE_SMALL**: Constant for screenLayout: a SCREENLAYOUT_SIZE_MASK value indicating the screen is at least approximately 320x426 dp units, corresponds to the small resource qualifier.
- **int SCREENLAYOUT_SIZE_UNDEFINED**: Constant for screenLayout: a SCREENLAYOUT_SIZE_MASK value indicating that no size has been set.
- **int SCREENLAYOUT_SIZE_XLARGE**: Constant for screenLayout: a SCREENLAYOUT_SIZE_MASK value indicating the screen is at least approximately 720x960 dp units, corresponds to the xlarge resource qualifier.
- **int SCREENLAYOUT_UNDEFINED**: Constant for screenLayout: a value indicating that screenLayout is undefined
- **int SCREEN_HEIGHT_DP_UNDEFINED**: Default value for screenHeightDp indicating that no width has been specified.
- **int SCREEN_WIDTH_DP_UNDEFINED**: Default value for screenWidthDp indicating that no width has been specified.
- **int SMALLEST_SCREEN_WIDTH_DP_UNDEFINED**: Default value for smallestScreenWidthDp indicating that no width has been specified.

- **int TOUCHSCREEN_FINGER**: Constant for touchscreen, value corresponding to the finger resource qualifier.
- **int TOUCHSCREEN_NOTOUCH**: Constant for touchscreen, value corresponding to the notouch resource qualifier.
- **int TOUCHSCREEN_STYLUS**: This constant was deprecated in API level 16. Not currently supported or used.
- **int TOUCHSCREEN_UNDEFINED**: Constant for touchscreen: a value indicating that no value has been set.
- **int UI_MODE_NIGHT_MASK**: Constant for uiMode: bits that encode the night mode.
- **int UI_MODE_NIGHT_NO**: Constant for uiMode: a UI_MODE_NIGHT_MASK value that corresponds to the notnight resource qualifier.
- **int UI_MODE_NIGHT_UNDEFINED**: Constant for uiMode: a UI_MODE_NIGHT_MASK value indicating that no mode type has been set.
- **int UI_MODE_NIGHT_YES**: Constant for uiMode: a UI_MODE_NIGHT_MASK value that corresponds to the night resource qualifier.
- **int UI_MODE_TYPE_APPLIANCE**: Constant for uiMode: a UI_MODE_TYPE_MASK value that corresponds to the appliance resource qualifier.
- **int UI_MODE_TYPE_CAR**: Constant for uiMode: a UI_MODE_TYPE_MASK value that corresponds to the car resource qualifier.
- **int UI_MODE_TYPE_DESK**: Constant for uiMode: a UI_MODE_TYPE_MASK value that corresponds to the desk resource qualifier.
- **int UI_MODE_TYPE_MASK**: Constant for uiMode: bits that encode the mode type.
- **int UI_MODE_TYPE_NORMAL**: Constant for uiMode: a UI_MODE_TYPE_MASK value that corresponds to no UI mode resource qualifier specified.
- **int UI_MODE_TYPE_TELEVISION**: Constant for uiMode: a UI_MODE_TYPE_MASK value that corresponds to the television resource qualifier.
- **int UI_MODE_TYPE_UNDEFINED**: Constant for uiMode: a UI_MODE_TYPE_MASK value indicating that no mode type has been set.
- **int UI_MODE_TYPE_VR_HEADSET**: Constant for uiMode: a UI_MODE_TYPE_MASK value that corresponds to the vrheadset resource qualifier.
- **int UI_MODE_TYPE_WATCH**: Constant for uiMode: a UI_MODE_TYPE_MASK value that corresponds to the watch resource qualifier.

2.9 Resources

- **Getting a resources object**: Inside the Activity class, we can call method getResources() from the Context class (from which class Activity inherits) to get a Resources reference for the application's package.

```
Resources resources = getResources();
```

- **Hierarchy**

java.lang.Object → android.content.res.Resources

- **Include**

```
android.content.res.Resources
```

- **Constructors**

```
Resources(AssetManager assets, DisplayMetrics metrics,
    Configuration config)
```

- **Public methods**
  - **void addLoaders(ResourcesLoader... loaders)**: Adds a loader to the list of loaders.
  - **final void finishPreloading()**: Called by zygote when it is done preloading resources, to change back to normal Resources operation.
  - **final void flushLayoutCache()**: Call this to remove all cached loaded layout resources from the Resources object.
  - **XmlResourceParser getAnimation(int id)**: Return an XmlResourceParser through which you can read an animation description for the given resource ID.
  - **final AssetManager getAssets()**: Retrieve underlying AssetManager storage for these resources.
  - **static int getAttributeSetSourceResId(AttributeSet set)**: Returns the resource ID of the resource that was used to create this AttributeSet.
  - **boolean getBoolean(int id)**: Return a boolean associated with a particular resource ID.
  - **int getColor(int id)**: This method was deprecated in API level 23. Use getColor(int, android.content.res.Resources.Theme) instead.
  - **int getColor(int id, Resources.Theme theme)**: Returns a themed color integer associated with a particular resource ID.
  - **ColorStateList getColorStateList(int id, Resources.Theme theme)**: Returns a themed color state list associated with a particular resource ID.
  - **ColorStateList getColorStateList(int id)**: This method was deprecated in API level 23. Use getColorStateList(int, android.content.res.Resources.Theme) instead.
  - **Configuration getConfiguration()**: Return the current configuration that is in effect for this resource object.

- **float getDimension(int id)**: Retrieve a dimensional for a particular resource ID.
- **int getDimensionPixelOffset(int id)**: Retrieve a dimensional for a particular resource ID for use as an offset in raw pixels.
- **int getDimensionPixelSize(int id)**: Retrieve a dimensional for a particular resource ID for use as a size in raw pixels.
- **DisplayMetrics getDisplayMetrics()**: Returns the current display metrics that are in effect for this resource object.
- **Drawable getDrawable(int id, Resources.Theme theme)**: Return a drawable object associated with a particular resource ID and styled for the specified theme.
- **Drawable getDrawable(int id)**: This method was deprecated in API level 22. Use getDrawable(int, android.content.res.Resources.Theme) instead.
- **Drawable getDrawableForDensity(int id, int density)**: This method was deprecated in API level 22. Use getDrawableForDensity(int, int, android.content.res.Resources.Theme) instead.
- **Drawable getDrawableForDensity(int id, int density, Resources.Theme theme)**: Return a drawable object associated with a particular resource ID for the given screen density in DPI and styled for the specified theme.
- **float getFloat(int id)**: Retrieve a floating-point value for a particular resource ID.
- **Typeface getFont(int id)**: Return the Typeface value associated with a particular resource ID.
- **float getFraction(int id, int base, int pbase)**: Retrieve a fractional unit for a particular resource ID.
- **int getIdentifier(String name, String defType, String defPackage)**: Return a resource identifier for the given resource name.
- **int[] getIntArray(int id)**: Return the int array associated with a particular resource ID.
- **int getInteger(int id)**: Return an integer associated with a particular resource ID.
- **XmlResourceParser getLayout(int id)**: Return an XmlResourceParser through which you can read a view layout description for the given resource ID.
- **Movie getMovie(int id)**: This method was deprecated in API level 29. Prefer AnimatedImageDrawable.
- **String getQuantityString(int id, int quantity, Object... formatArgs)**: Formats the string necessary for grammatically correct pluralization of the given resource ID for the given quantity, using the given arguments.
- **String getQuantityString(int id, int quantity)**: Returns the string necessary for grammatically correct pluralization of the given resource ID for the given quantity.
- **CharSequence getQuantityText(int id, int quantity)**: Returns the character sequence necessary for grammatically correct pluralization of the given resource ID for the given quantity.
- **String getResourceEntryName(int resid)**: Return the entry name for a given resource identifier.
- **String getResourceName(int resid)**: Return the full name for a given resource identifier.

- **String getResourcePackageName(int resid)**: Return the package name for a given resource identifier.
- **String getResourceTypeName(int resid)**: Return the type name for a given resource identifier.
- **String getString(int id)**: Return the string value associated with a particular resource ID. It will be stripped of any styled text information.
- **String getString(int id, Object... formatArgs)**: Return the string value associated with a particular resource ID, substituting the format arguments as defined in Formatter and String.format(String, Object). It will be stripped of any styled text information.
- **String[] getStringArray(int id)**: Return the string array associated with a particular resource ID.
- **static Resources getSystem()**: Return a global shared Resources object that provides access to only system resources (no application resources), is not configured for the current screen (can not use dimension units, does not change based on orientation, etc), and is not affected by Runtime Resource Overlay.
- **CharSequence getText(int id, CharSequence def)**: Return the string value associated with a particular resource ID.
- **CharSequence getText(int id)**: Return the string value associated with a particular resource ID. The returned object will be a String if this is a plain string; it will be some other type of CharSequence if it is styled.
- **CharSequence[] getTextArray(int id)**: Return the styled text array associated with a particular resource ID.
- **void getValue(String name, TypedValue outValue, boolean resolveRefs)**: Return the raw data associated with a particular resource ID.
- **void getValue(int id, TypedValue outValue, boolean resolveRefs)**: Return the raw data associated with a particular resource ID.
- **void getValueForDensity(int id, int density, TypedValue outValue, boolean resolveRefs)**: Get the raw value associated with a resource with associated density.
- **XmlResourceParser getXml(int id)**: Return an XmlResourceParser through which you can read a generic XML resource for the given resource ID.
- **final Resources.Theme newTheme()**: Generate a new Theme object for this set of Resources.
- **TypedArray obtainAttributes(AttributeSet set, int[] attrs)**: Retrieve a set of basic attribute values from an AttributeSet, not performing styling of them using a theme and/or style resources.
- **TypedArray obtainTypedArray(int id)**: Return an array of heterogeneous values.
- **InputStream openRawResource(int id, TypedValue value)**: Open a data stream for reading a raw resource.
- **InputStream openRawResource(int id)**: Open a data stream for reading a raw resource.
- **AssetFileDescriptor openRawResourceFd(int id)**: Open a file descriptor for reading a raw resource.
- **void parseBundleExtra(String tagName, AttributeSet attrs, Bundle outBundle)**: Parse a name/value pair out of an XML tag holding that data.

  - **void parseBundleExtras(XmlResourceParser parser, Bundle outBundle)**: Parse a series of <extra> tags from an XML file.
  - **static void registerResourcePaths(String uniqueId, ApplicationInfo appInfo)**: Register the resources paths of a package (e.g. a shared library).
  - **void removeLoaders(ResourcesLoader... loaders)**: Removes loaders from the list of loaders.
  - **void updateConfiguration(Configuration config, DisplayMetrics metrics)**: This method was deprecated in API level 25. See Context.createConfigurationContext(Configuration).

- **Constants**
  - **int ID_NULL**: The null resource ID.

2.10 Resources.Theme

- **Hierarchy**

  java.lang.Object → android.content.res.Resources.Theme

- **Include**

```
android.content.res.Resources.Theme
```

- **Public methods**
  - **void applyStyle(int resId, boolean force)**: Place new attribute values into the theme.
  - **void dump(int priority, String tag, String prefix)**: Print contents of this theme out to the log.
  - **boolean equals(Object o)**: Indicates whether some other object is "equal to" this one.
  - **int[] getAttributeResolutionStack(int defStyleAttr, int defStyleRes, int explicitStyleRes)**: Returns the ordered list of resource ID that are considered when resolving attribute values when making an equivalent call to obtainStyledAttributes(android.util.AttributeSet, int[], int, int) .
  - **int getChangingConfigurations()**: Returns a bit mask of configuration changes that will impact this theme (and thus require completely reloading it).
  - **Drawable getDrawable(int id)**: Return a drawable object associated with a particular resource ID and styled for the Theme.
  - **int getExplicitStyle(AttributeSet set)**: Returns the resource ID for the style specified using style="..." in the AttributeSet's backing XML element or Resources.ID_NULL otherwise if not specified or otherwise not applicable.
  - **Resources getResources()**: Returns the resources to which this theme belongs.
  - **int hashCode()**: Returns a hash code value for the object.
  - **TypedArray obtainStyledAttributes(AttributeSet set, int[] attrs, int defStyleAttr, int defStyleRes)**: Return a TypedArray holding the attribute values in set that are listed in attrs.
  - **TypedArray obtainStyledAttributes(int[] attrs)**: Return a TypedArray holding the values defined by Theme which are listed in attrs.
  - **TypedArray obtainStyledAttributes(int resId, int[] attrs)**: Return a TypedArray holding the values defined by the style resource resid which are listed in attrs.
  - **void rebase()**: Rebases the theme against the parent Resource object's current configuration by re-applying the styles passed to applyStyle(int, boolean).
  - **boolean resolveAttribute(int resid, TypedValue outValue, boolean resolveRefs)**: Retrieve the value of an attribute in the Theme.
  - **void setTo(Resources.Theme other)**: Set this theme to hold the same contents as the theme other.
  - **String toString()**: Returns a string representation of the object.

2.11 TypedValue

- **Hierarchy**

java.lang.Object → android.util.TypedValue

- **Include**

```
android.util.TypedValue
```

- **Constructors**

```
TypedValue()
```

- **Public methods**
  - **static float applyDimension(int unit, float value, DisplayMetrics metrics)**: Converts an unpacked complex data value holding a dimension to its final floating point pixel value.
  - **static final String coerceToString(int type, int data)**: Perform type conversion as per coerceToString() on an explicitly supplied type and data.
  - **final CharSequence coerceToString()**: Regardless of the actual type of the value, try to convert it to a string value.
  - **static float complexToDimension(int data, DisplayMetrics metrics)**: Converts a complex data value holding a dimension to its final floating point value.
  - **static int complexToDimensionPixelOffset(int data, DisplayMetrics metrics)**: Converts a complex data value holding a dimension to its final value as an integer pixel offset.
  - **static int complexToDimensionPixelSize(int data, DisplayMetrics metrics)**: Converts a complex data value holding a dimension to its final value as an integer pixel size.
  - **static float complexToFloat(int complex)**: Retrieve the base value from a complex data integer.
  - **static float complexToFraction(int data, float base, float pbase)**: Converts a complex data value holding a fraction to its final floating point value.
  - **static float convertDimensionToPixels(int unitToConvertFrom, float value, DisplayMetrics metrics)**: Converts a dimension value to raw pixels, e.g. DP to PX.
  - **static float convertPixelsToDimension(int unitToConvertTo, float pixelValue, DisplayMetrics metrics)**: Converts a pixel value to the given dimension, e.g. PX to DP.
  - **static float deriveDimension(int unitToConvertTo, float pixelValue, DisplayMetrics metrics)**: Converts a pixel value to the given dimension, e.g. PX to DP.
  - **int getComplexUnit()**: Return the complex unit type for this value.
  - **float getDimension(DisplayMetrics metrics)**: Return the data for this value as a dimension.

- **final float getFloat()**: Return the data for this value as a float.
- **float getFraction(float base, float pbase)**: Return the data for this value as a fraction.
- **boolean isColorType()**: Determine if a value is a color.
- **void setTo(TypedValue other)**:
- **String toString()**: Returns a string representation of the object.

- **Constants**
  - **int COMPLEX_MANTISSA_MASK**: Complex data: mask to extract mantissa information (after shifting by COMPLEX_MANTISSA_SHIFT).
  - **int COMPLEX_MANTISSA_SHIFT**: Complex data: bit location of mantissa information.
  - **int COMPLEX_RADIX_0p23**: Complex data: the mantissa magnitude is 0 bits – i.e, 0x0.nnnnnn
  - **int COMPLEX_RADIX_16p7**: Complex data: the mantissa magnitude is 16 bits – i.e, 0xnnnn.nn
  - **int COMPLEX_RADIX_23p0**: Complex data: the mantissa is an integral number – i.e., 0xnnnnnn.0
  - **int COMPLEX_RADIX_8p15**: Complex data: the mantissa magnitude is 8 bits – i.e, 0xnn.nnnn
  - **int COMPLEX_RADIX_MASK**: Complex data: mask to extract radix information (after shifting by COMPLEX_RADIX_SHIFT).
  - **int COMPLEX_RADIX_SHIFT**: Complex data: where the radix information is, telling where the decimal place appears in the mantissa.
  - **int COMPLEX_UNIT_DIP**: TYPE_DIMENSION complex unit: Value is Device Independent Pixels.
  - **int COMPLEX_UNIT_FRACTION**: TYPE_FRACTION complex unit: A basic fraction of the overall size.
  - **int COMPLEX_UNIT_FRACTION_PARENT**: TYPE_FRACTION complex unit: A fraction of the parent size.
  - **int COMPLEX_UNIT_IN**: TYPE_DIMENSION complex unit: Value is in inches.
  - **int COMPLEX_UNIT_MASK**: Complex data: mask to extract unit information (after shifting by COMPLEX_UNIT_SHIFT).
  - **int COMPLEX_UNIT_MM**: TYPE_DIMENSION complex unit: Value is in millimeters.
  - **int COMPLEX_UNIT_PT**: TYPE_DIMENSION complex unit: Value is in points.
  - **int COMPLEX_UNIT_PX**: TYPE_DIMENSION complex unit: Value is raw pixels.
  - **int COMPLEX_UNIT_SHIFT**: Complex data: bit location of unit information.
  - **int COMPLEX_UNIT_SP**: TYPE_DIMENSION complex unit: Value is a scaled pixel.
  - **int DATA_NULL_EMPTY**: TYPE_NULL data indicating the value was explicitly set to null.

- **int DATA_NULL_UNDEFINED**: TYPE_NULL data indicating the value was not specified.
- **int DENSITY_DEFAULT**: If density is equal to this value, then the density should be treated as the system's default density value: DisplayMetrics.DENSITY_DEFAULT.
- **int DENSITY_NONE**: If density is equal to this value, then there is no density associated with the resource and it should not be scaled.
- **int TYPE_ATTRIBUTE**: The data field holds an attribute resource identifier (referencing an attribute in the current theme style, not a resource entry).
- **int TYPE_DIMENSION**: The data field holds a complex number encoding a dimension value.
- **int TYPE_FIRST_COLOR_INT**: Identifies the start of integer values that were specified as color constants (starting with ").
- **int TYPE_FIRST_INT**: Identifies the start of plain integer values.
- **int TYPE_FLOAT**: The data field holds an IEEE 754 floating point number.
- **int TYPE_FRACTION**: The data field holds a complex number encoding a fraction of a container.
- **int TYPE_INT_BOOLEAN**: data holds 0 to represent false, or a value different from 0 to represent true.
- **int TYPE_INT_COLOR_ARGB4**: The data field holds a color that was originally specified as argb.
- **int TYPE_INT_COLOR_ARGB8**: The data field holds a color that was originally specified as aarrggbb.
- **int TYPE_INT_COLOR_RGB4**: The data field holds a color that was originally specified as rgb.
- **int TYPE_INT_COLOR_RGB8**: The data field holds a color that was originally specified as rrggbb.
- **int TYPE_INT_DEC**: The data field holds a number that was originally specified in decimal.
- **int TYPE_INT_HEX**: The data field holds a number that was originally specified in hexadecimal (0xn).
- **int TYPE_LAST_COLOR_INT**: Identifies the end of integer values that were specified as color constants.
- **int TYPE_LAST_INT**: Identifies the end of plain integer values.
- **int TYPE_NULL**: The value contains no data.
- **int TYPE_REFERENCE**: The data field holds a resource identifier.
- **int TYPE_STRING**: The string field holds string data.

- **Fields**
  - **public int assetCookie**: Additional information about where the value came from; only set for strings.
  - **public int changingConfigurations**: If the value came from a resource, these are the configurations for which its contents can change.
  - **public int data**: Basic data in the value, interpreted according to type
  - **public int density**: If the Value came from a resource, this holds the corresponding pixel density.

- **public int resourceId**: If Value came from a resource, this holds the corresponding resource id.
- **public int sourceResourceId**: If the Value came from a style resource or a layout resource (set in an XML layout), this holds the corresponding style or layout resource id against which the attribute was resolved.
- **public CharSequence string**: If the value holds a string, this is it.
- **public int type**: The type held by this value, as defined by the constants here.

2.12 DisplayMetrics

- **Getting a DisplayMetrics object**: We get an instance using the getDisplayMetrics() method from the Resource class

```
DisplayMetrics metrics = getResources().getDisplayMetrics();
```

- **Hierarchy**

java.lang.Object → android.util.DisplayMetrics

- **Include**

```
android.util.DisplayMetrics
```

- **Constructors**

```
DisplayMetrics()
```

- **Public methods**
  - **boolean equals(DisplayMetrics other)**: Returns true if these display metrics equal the other display metrics.
  - **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 setTo(DisplayMetrics o)**:
  - **void setToDefaults()**:
  - **String toString()**: Returns a string representation of the object.
- **Fields**
  - **public static final int DENSITY_DEVICE_STABLE**: The device's stable density.
  - **public float density**: The logical density of the display.
  - **public int densityDpi**: The screen density expressed as dots-per-inch.
  - **public int heightPixels**: The absolute height of the available display size in pixels.
  - **public float scaledDensity**: This field was deprecated in API level 34. this scalar factor is no longer accurate due to adaptive non-linear font scaling. Please use TypedValue.applyDimension(int, float, DisplayMetrics) or TypedValue.deriveDimension(int, float, DisplayMetrics) to convert between SP font sizes and pixels.
  - **public int widthPixels**: The absolute width of the available display size in pixels.
  - **public float xdpi**: The exact physical pixels per inch of the screen in the X dimension.

- **public float ydpi**: The exact physical pixels per inch of the screen in the Y dimension.

- **Constants**
  - **int DENSITY_140**: Intermediate density for screens that sit between DENSITY_LOW (120dpi) and DENSITY_MEDIUM (160dpi).
  - **int DENSITY_180**: Intermediate density for screens that sit between DENSITY_MEDIUM (160dpi) and DENSITY_HIGH (240dpi).
  - **int DENSITY_200**: Intermediate density for screens that sit between DENSITY_MEDIUM (160dpi) and DENSITY_HIGH (240dpi).
  - **int DENSITY_220**: Intermediate density for screens that sit between DENSITY_MEDIUM (160dpi) and DENSITY_HIGH (240dpi).
  - **int DENSITY_260**: Intermediate density for screens that sit between DENSITY_HIGH (240dpi) and DENSITY_XHIGH (320dpi).
  - **int DENSITY_280**: Intermediate density for screens that sit between DENSITY_HIGH (240dpi) and DENSITY_XHIGH (320dpi).
  - **int DENSITY_300**: Intermediate density for screens that sit between DENSITY_HIGH (240dpi) and DENSITY_XHIGH (320dpi).
  - **int DENSITY_340**: Intermediate density for screens that sit somewhere between DENSITY_XHIGH (320 dpi) and DENSITY_XXHIGH (480 dpi).
  - **int DENSITY_360**: Intermediate density for screens that sit somewhere between DENSITY_XHIGH (320 dpi) and DENSITY_XXHIGH (480 dpi).
  - **int DENSITY_390**: Intermediate density for screens that sit somewhere between DENSITY_XHIGH (320 dpi) and DENSITY_XXHIGH (480 dpi).
  - **int DENSITY_400**: Intermediate density for screens that sit somewhere between DENSITY_XHIGH (320 dpi) and DENSITY_XXHIGH (480 dpi).
  - **int DENSITY_420**: Intermediate density for screens that sit somewhere between DENSITY_XHIGH (320 dpi) and DENSITY_XXHIGH (480 dpi).
  - **int DENSITY_440**: Intermediate density for screens that sit somewhere between DENSITY_XHIGH (320 dpi) and DENSITY_XXHIGH (480 dpi).
  - **int DENSITY_450**: Intermediate density for screens that sit somewhere between DENSITY_XHIGH (320 dpi) and DENSITY_XXHIGH (480 dpi).
  - **int DENSITY_520**: Intermediate density for screens that sit somewhere between DENSITY_XXHIGH (480 dpi) and DENSITY_XXXHIGH (640 dpi).
  - **int DENSITY_560**: Intermediate density for screens that sit somewhere between DENSITY_XXHIGH (480 dpi) and DENSITY_XXXHIGH (640 dpi).
  - **int DENSITY_600**: Intermediate density for screens that sit somewhere between DENSITY_XXHIGH (480 dpi) and DENSITY_XXXHIGH (640 dpi).
  - **int DENSITY_DEFAULT**: The reference density used throughout the system.
  - **int DENSITY_HIGH**: Standard quantized DPI for high-density screens.
  - **int DENSITY_LOW**: Standard quantized DPI for low-density screens.
  - **int DENSITY_MEDIUM**: Standard quantized DPI for medium-density screens.
  - **int DENSITY_TV**: This is a secondary density, added for some common screen configurations.
  - **int DENSITY_XHIGH**: Standard quantized DPI for extra-high-density screens.

- **int DENSITY_XXHIGH**: Standard quantized DPI for extra-extra-high-density screens.
- **int DENSITY_XXXHIGH**: Standard quantized DPI for extra-extra-extra-high-density screens.

2.13 Log

- **Hierarchy**

  java.lang.Object → android.util.Log

- **Include**

```
android.util.Log
```

- **Public methods**

  - **static int d(String tag, String msg, Throwable tr)**: Send a DEBUG log message and log the exception.
  - **static int d(String tag, String msg)**: Send a DEBUG log message.
  - **static int e(String tag, String msg)**: Send an ERROR log message.
  - **static int e(String tag, String msg, Throwable tr)**: Send a ERROR log message and log the exception.
  - **static String getStackTraceString(Throwable tr)**: Handy function to get a loggable stack trace from a Throwable

    If any of the throwables in the cause chain is an UnknownHostException, this returns an empty string.
  - **static int i(String tag, String msg, Throwable tr)**: Send a INFO log message and log the exception.
  - **static int i(String tag, String msg)**: Send an INFO log message.
  - **static boolean isLoggable(String tag, int level)**: Checks to see whether or not a log for the specified tag is loggable at the specified level.
  - **static int println(int priority, String tag, String msg)**: Low-level logging call.
  - **static int v(String tag, String msg)**: Send a VERBOSE log message.
  - **static int v(String tag, String msg, Throwable tr)**: Send a VERBOSE log message and log the exception.
  - **static int w(String tag, Throwable tr)**: Send a WARN log message and log the exception.
  - **static int w(String tag, String msg, Throwable tr)**: Send a WARN log message and log the exception.
  - **static int w(String tag, String msg)**: Send a WARN log message.
  - **static int wtf(String tag, String msg)**: What a Terrible Failure: Report a condition that should never happen.
  - **static int wtf(String tag, Throwable tr)**: What a Terrible Failure: Report an exception that should never happen.
  - **static int wtf(String tag, String msg, Throwable tr)**: What a Terrible Failure: Report an exception that should never happen.

- **Constants**

  - **int ASSERT**: Priority constant for the println method.
  - **int DEBUG**: Priority constant for the println method; use Log.d.

– **int ERROR**: Priority constant for the println method; use Log.e.
– **int INFO**: Priority constant for the println method; use Log.i.
– **int VERBOSE**: Priority constant for the println method; use Log.v.
– **int WARN**: Priority constant for the println method; use Log.w.

2.14 WindowManager (Interface)

- **Getting WindowManagerObject**: We can get a WindowManager reference through the .getWindowManager() method in the activity class

```
WindowManager wm = getWindowManager();
```

- **Signature**

```
public interface WindowManager implements ViewManager
```

- **Include**

```
android.view.WindowManager
```

- **Default methods**
  - **default void addCrossWindowBlurEnabledListener(Consumer<Boolean> listener)**: Adds a listener, which will be called when cross-window blurs are enabled/disabled at runtime.
  - **default void addCrossWindowBlurEnabledListener(Executor executor, Consumer<Boolean> listener)**: Adds a listener, which will be called when cross-window blurs are enabled/disabled at runtime.
  - **default void addProposedRotationListener(Executor executor, IntConsumer listener)**: Adds a listener to start monitoring the proposed rotation of the current associated context.
  - **default int addScreenRecordingCallback(Executor executor, Consumer<Integer> callback)**: Adds a screen recording callback.
  - **default WindowMetrics getCurrentWindowMetrics()**: Returns the WindowMetrics according to the current system state.
  - **abstract Display getDefaultDisplay()**: This method was deprecated in API level 30. Use Context.getDisplay() instead.
  - **default WindowMetrics getMaximumWindowMetrics()**: Returns the largest WindowMetrics an app may expect in the current system state.
  - **default boolean isCrossWindowBlurEnabled()**: Returns whether cross-window blur is currently enabled.
  - **default InputTransferToken registerBatchedSurfaceControlInputReceiver(InputTransferToken hostInputTransferToken, SurfaceControl surfaceControl, Choreographer choreographer, SurfaceControlInputReceiver receiver)**: Registers a SurfaceControlInputReceiver for a SurfaceControl that will receive batched input event.
  - **default void registerTrustedPresentationListener(IBinder window, TrustedPresentationThresholds thresholds, Executor executor, Consumer<Boolean> listener)**: Sets a callback to receive feedback about the presentation of a Window.

- **default InputTransferToken registerUnbatchedSurfaceControlInputReceiver(InputTransferToken hostInputTransferToken, SurfaceControl surfaceControl, Looper looper, SurfaceControlInputReceiver receiver)**: Registers a SurfaceControlInputReceiver for a SurfaceControl that will receive every input event.
- **default void removeCrossWindowBlurEnabledListener(Consumer<Boolean> listener)**: Removes a listener, previously added with addCrossWindowBlurEnabledListener(Executor, Consumer)
- **default void removeProposedRotationListener(IntConsumer listener)**: Removes a listener, previously added with addProposedRotationListener(Executor, IntConsumer).
- **default void removeScreenRecordingCallback(Consumer<Integer> callback)**: Removes a screen recording callback.
- **abstract void removeViewImmediate(View view)**: Special variation of ViewManager.removeView(View) that immediately invokes the given view hierarchy's View.onDetachedFromWindow() methods before returning.
- **default boolean transferTouchGesture(InputTransferToken transferFromToken, InputTransferToken transferToToken)**: Transfer the currently in progress touch gesture from the transferFromToken to the transferToToken.
- **default void unregisterSurfaceControlInputReceiver(SurfaceControl surfaceControl)**: Unregisters and cleans up the registered SurfaceControlInputReceiver for the specified token.
- **default void unregisterTrustedPresentationListener(Consumer<Boolean> listener)**: Removes a presentation listener associated with a window.

- **Constants**
  - **int COMPAT_SMALL_COVER_SCREEN_OPT_IN**: Value applicable for the PROPERTY_COMPAT_ALLOW_SMALL_COVER_SCREEN property to provide a signal to the system that an application or its specific activities explicitly opt into being displayed on small cover screens on flippable style foldable devices that measure at least 1.5 inches up to 2.2 inches for the shorter dimension and at least 2.4 inches up to 3.4 inches for the longer dimension

    Value is COMPAT_SMALL_COVER_SCREEN_OPT_IN
  - **String PROPERTY_ACTIVITY_EMBEDDING_ALLOW_SYSTEM_OVERRIDE**: Application-level PackageManager.Property tag that specifies whether OEMs are permitted to provide activity embedding split-rule configurations on behalf of the app.
  - **String PROPERTY_ACTIVITY_EMBEDDING_SPLITS_ENABLED**: Application level PackageManager.Property that an app can specify to inform the system that the app is activity embedding split feature enabled.
  - **String PROPERTY_CAMERA_COMPAT_ALLOW_FORCE_ROTATION**: Application level PackageManager.Property for an app to inform the system that the app should be excluded from the camera compatibility force rotation treatment.
  - **String PROPERTY_CAMERA_COMPAT_ALLOW_REFRESH**: Application level PackageManager.Property for an app to inform the system that the app should be excluded from the activity "refresh" after the camera compatibility force rotation treatment.

- **String PROPERTY_CAMERA_COMPAT_ALLOW_SIMULATE_REQUESTED_ORIENTATION**: Application-level [PackageManager][android.content.pm.PackageManager.Property] tag that (when set to false) informs the system the app has opted out of the camera compatibility treatment for fixed-orientation apps, which simulates running on a portrait device, in the orientation requested by the app.
- **String PROPERTY_CAMERA_COMPAT_ENABLE_REFRESH_VIA_PAUSE**: Application level PackageManager.Property for an app to inform the system that the activity should be or shouldn't be "refreshed" after the camera compatibility force rotation treatment using "paused -> resumed" cycle rather than "stopped -> resumed".
- **String PROPERTY_COMPAT_ALLOW_DISPLAY_ORIENTATION_OVERRIDE**: Application level PackageManager.Property for an app to inform the system that the app should be opted-out from the compatibility override that fixes display orientation to landscape natural orientation when an activity is fullscreen.
- **String PROPERTY_COMPAT_ALLOW_IGNORING_ORIENTATION_REQUEST_WHEN_LOOP_DETECTED**: Application level PackageManager.Property for an app to inform the system that the app can be opted-out from the compatibility treatment that avoids Activity#setRequestedOrientation() loops.
- **String PROPERTY_COMPAT_ALLOW_MIN_ASPECT_RATIO_OVERRIDE**: Application level PackageManager.Property for an app to inform the system that the app should be opted-out from the compatibility override that changes the min aspect ratio.
- **String PROPERTY_COMPAT_ALLOW_ORIENTATION_OVERRIDE**: Application level PackageManager.Property for an app to inform the system that the app should be excluded from the compatibility override for orientation set by the device manufacturer.
- **String PROPERTY_COMPAT_ALLOW_RESIZEABLE_ACTIVITY_OVERRIDES**: Application level PackageManager.Property for an app to inform the system that the app should be opted-out from the compatibility overrides that change the resizability of the app.
- **String PROPERTY_COMPAT_ALLOW_SANDBOXING_VIEW_BOUNDS_APIS**: Application level PackageManager.Property for an app to inform the system that it needs to be opted-out from the compatibility treatment that sandboxes the View API.
- **String PROPERTY_COMPAT_ALLOW_SMALL_COVER_SCREEN**: Application or Activity level PackageManager.Property to provide any preferences for showing all or specific Activities on small cover displays of foldable style devices.
- **String PROPERTY_COMPAT_ALLOW_USER_ASPECT_RATIO_FULLSCREEN_OVERRIDE**: Application level PackageManager.Property tag that (when set to false) informs the system the app has opted out of the full-screen option of the user aspect ratio compatibility override settings.
- **String PROPERTY_COMPAT_ALLOW_USER_ASPECT_RATIO_OVERRIDE**: Application level PackageManager. Property tag that (when set to false) informs the system the app has opted out of the user-facing aspect ratio compatibility override.
- **String PROPERTY_COMPAT_ENABLE_FAKE_FOCUS**: Application level PackageManager.Property for an app to inform the system that the application can be opted-in or opted-out from the compatibility treatment that enables sending a fake focus event for unfocused resumed split-screen activities.

- **String PROPERTY_COMPAT_IGNORE_REQUESTED_ORIENTATION**: Application level PackageManager.Property for an app to inform the system that the app can be opted-in or opted-out from the compatibility treatment that avoids Activity#setRequestedOrientation() loops.
- **String PROPERTY_SUPPORTS_MULTI_INSTANCE_SYSTEM_UI**: Activity or Application level PackageManager.Property for an app to declare that System UI should be shown for this app/component to allow it to be launched as multiple instances.
- **int SCREEN_RECORDING_STATE_NOT_VISIBLE**: Indicates the app that registered the callback is not visible in screen recording.
- **int SCREEN_RECORDING_STATE_VISIBLE**: Indicates the app that registered the callback is visible in screen recording.

2.15 Display

- **Getting Display object**: We can get a Display object reference through the WindowManager object.

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

- **Hierarchy**

java.lang.Object → android.view.Display

- **Include**

```
android.view.Display
```

- **Public methods**
  - **long getAppVsyncOffsetNanos()**: Gets the app VSYNC offset, in nanoseconds.
  - **void getCurrentSizeRange(Point outSmallestSize, Point outLargestSize)**: Return the range of display sizes an application can expect to encounter under normal operation, as long as there is no physical change in screen size.
  - **DisplayCutout getCutout()**: Returns the DisplayCutout, or null if there is none.
  - **DeviceProductInfo getDeviceProductInfo()**: Returns the product-specific information about the display or the directly connected device on the display chain.
  - **int getDisplayId()**: Gets the display id.
  - **int getFlags()**: Returns a combination of flags that describe the capabilities of the display.
  - **Display.HdrCapabilities getHdrCapabilities()**: Returns the current display mode's HDR capabilities.
  - **float getHdrSdrRatio()**:
  - **int getHeight()**: This method was deprecated in API level 15. Use WindowMetrics.getBounds() instead.
  - **float getHighestHdrSdrRatio()**:
  - **void getMetrics(DisplayMetrics outMetrics)**: This method was deprecated in API level 30. Use WindowMetrics.getBounds() to get the dimensions of the application window. Use WindowMetrics.getDensity() to get the density of the application window.
  - **Display.Mode getMode()**: Returns the active mode of the display.
  - **String getName()**: Gets the name of the display.
  - **int getOrientation()**: This method was deprecated in API level 15. use getRotation()
  - **OverlayProperties getOverlaySupport()**: Returns the OverlayProperties of the display.
  - **int getPixelFormat()**: This method was deprecated in API level 17. This method is no longer supported. The result is always PixelFormat.RGBA_8888.

- **ColorSpace getPreferredWideGamutColorSpace()**: Returns the preferred wide color space of the Display.
- **long getPresentationDeadlineNanos()**: This is how far in advance a buffer must be queued for presentation at a given time.
- **void getRealMetrics(DisplayMetrics outMetrics)**: This method was deprecated in API level 31. Use WindowManager.getCurrentWindowMetrics() to identify the current size of the activity window. UI-related work, such as choosing UI layouts, should rely upon WindowMetrics.getBounds(). Use Configuration.densityDpi to get the current density.
- **void getRealSize(Point outSize)**: This method was deprecated in API level 31. Use WindowManager.getCurrentWindowMetrics() to identify the current size of the activity window. UI-related work, such as choosing UI layouts, should rely upon WindowMetrics.getBounds().
- **void getRectSize(Rect outSize)**: This method was deprecated in API level 30. Use WindowMetrics.getBounds() to get the dimensions of the application window.
- **float getRefreshRate()**: Gets the refresh rate of this display in frames per second.
- **int getRotation()**: Returns the rotation of the screen from its "natural" orientation.
- **RoundedCorner getRoundedCorner(int position)**: Returns the RoundedCorner of the given position if there is one.
- **DisplayShape getShape()**: Returns the DisplayShape which is based on display coordinates.
- **void getSize(Point outSize)**: This method was deprecated in API level 30. Use WindowMetrics instead. Obtain a WindowMetrics instance by calling WindowManager.getCurrentWindowMetrics(), then call WindowMetrics.getBounds() to get the dimensions of the application window.
- **int getState()**: Gets the state of the display, such as whether it is on or off.
- **float getSuggestedFrameRate(int category)**: Gets the display-defined frame rate given a descriptive frame rate category.
- **Mode[] getSupportedModes()**: Gets the supported modes of this display, might include synthetic modes
- **float[] getSupportedRefreshRates()**: Get the supported refresh rates of this display in frames per second.
- **int getWidth()**: This method was deprecated in API level 15. Use WindowMetrics.getBounds instead.
- **boolean hasArrSupport()**: Returns whether display supports adaptive refresh rate or not.
- **boolean isHdr()**: Returns whether this display supports any HDR type.
- **boolean isHdrSdrRatioAvailable()**:
- **boolean isInternal()**: Check if this is a built-in display, i.e.
- **boolean isMinimalPostProcessingSupported()**: Returns true if the connected display can be switched into a mode with minimal post processing.
- **boolean isValid()**: Returns true if this display is still valid, false if the display has been removed.
- **boolean isWideColorGamut()**: Returns whether this display can be used to display wide color gamut content.

- **void registerHdrSdrRatioChangedListener(Executor executor, Consumer<Display> listener)**: Registers a listener that will be invoked whenever the display's hdr/sdr ratio has changed.
- **String toString()**: Returns a string representation of the object.
- **void unregisterHdrSdrRatioChangedListener(Consumer<Display> listener)**:

- **Constants**
  - **int DEFAULT_DISPLAY**: The default Display id, which is the id of the primary display assuming there is one.
  - **int FLAG_PRESENTATION**: Display flag: Indicates that the display is a presentation display.
  - **int FLAG_PRIVATE**: Display flag: Indicates that the display is private.
  - **int FLAG_ROUND**: Display flag: Indicates that the display has a round shape.
  - **int FLAG_SECURE**: Display flag: Indicates that the display has a secure video output and supports compositing secure surfaces.
  - **int FLAG_SUPPORTS_PROTECTED_BUFFERS**: Display flag: Indicates that the display supports compositing content that is stored in protected graphics buffers.
  - **int FRAME_RATE_CATEGORY_HIGH**: High category determines the framework's recommended high frame rate.
  - **int FRAME_RATE_CATEGORY_NORMAL**: Normal category determines the framework's recommended normal frame rate.
  - **int INVALID_DISPLAY**: Invalid display id.
  - **int STATE_DOZE**: Display state: The display is dozing in a low power state; it is still on but is optimized for showing system-provided content while the device is non-interactive.
  - **int STATE_DOZE_SUSPEND**: Display state: The display is dozing in a suspended low power state; it is still on but the CPU is not updating it.
  - **int STATE_OFF**: Display state: The display is off.
  - **int STATE_ON**: Display state: The display is on.
  - **int STATE_ON_SUSPEND**: Display state: The display is in a suspended full power state; it is still on but the CPU is not updating it.
  - **int STATE_UNKNOWN**: Display state: The display state is unknown.
  - **int STATE_VR**: Display state: The display is on and optimized for VR mode.

2.16 ConstraintLayout

- **Hierarchy**:

  java.lang.Object → android.view.View → android.view.ViewGroup → androidx.constraintlayout.widget.ConstraintLayout

- **Include**:

```
androidx.constraintlayout.widget.ConstraintLayout
```

- **Constructors**:

```
ConstraintLayout(@NonNull Context context)
ConstraintLayout(@NonNull Context context, @Nullable AttributeSet attrs)
ConstraintLayout( @NonNull Context context, @Nullable AttributeSet attrs, int defStyleAttr)

@TargetApi(value = Build.VERSION_CODES.LOLLIPOP)
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.

  - **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**:

2.17 ConstraintLayout.LayoutParams

- **Hierarchy**:

  java.lang.Object → android.view.View → android.view.ViewGroup → androidx.constraintlayout.widget.ConstraintLayout

- **Include**:

```
androidx.constraintlayout.widget.ConstraintLayout
```

- **Constructors**:

```
LayoutParams(ViewGroup.LayoutParams params)
LayoutParams(Context c, AttributeSet attrs)
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.

- **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.

2.18 RelativeLayout

- **Hierarchy**:

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

- **Include**:

```
android.widget.RelativeLayout
```

- **Constructors**:

```
RelativeLayout(Context context)
RelativeLayout(Context context, AttributeSet attrs)
RelativeLayout(Context context, AttributeSet attrs, int
    ↪ defStyleAttr)
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.LayoutParams 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.

2.19 RelativeLayout.LayoutParams

- **Hierarchy**:

  java.lang.Object → android.view.ViewGroup.LayoutParams →
  android.view.ViewGroup.MarginLayoutParams →
  android.widget.RelativeLayout.LayoutParams

- **Include**

```
android.widget.RelativeLayout.LayoutParams
```

- **Constructors**:

```
LayoutParams(Context c, AttributeSet attrs)
LayoutParams(ViewGroup.LayoutParams source)
LayoutParams(ViewGroup.MarginLayoutParams source)
LayoutParams(RelativeLayout.LayoutParams source)
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.

2.20 LinearLayout

- **Hierarchy**:

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

- **Include**

```
android.widget.LinearLayout
```

- **Constructors**:

```
LinearLayout(Context context)
LinearLayout(Context context, AttributeSet attrs)
LinearLayout(Context context, AttributeSet attrs, int
    ↪ defStyleAttr)
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.

- **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.

2.21 LinearLayout.LayoutParams

- **Hierarchy**:

  java.lang.Object → android.view.ViewGroup.LayoutParams →
  android.view.ViewGroup.MarginLayoutParams →
  android.widget.LinearLayout.LayoutParams

- **Include**

```
android.widget.LinearLayout.LayoutParams
```

- **Constructors**:

```
LayoutParams(Context c, AttributeSet attrs)
LayoutParams(ViewGroup.LayoutParams p)
LayoutParams(ViewGroup.MarginLayoutParams source)
LayoutParams(LinearLayout.LayoutParams source)
LayoutParams(int width, int height)
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.

2.22 GridLayout

- **Hierarchy**:

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

- **Include**:

```
android.widget.GridLayout
```

- **Constructors**

```
GridLayout(Context context)
GridLayout(Context context, AttributeSet attrs)
GridLayout(Context context, AttributeSet attrs, int
    ↪ defStyleAttr)
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 positioning 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 ordered 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.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 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.

2.23 GridLayout.LayoutParams

- **Hierarchy**:

  java.lang.Object → android.view.ViewGroup.LayoutParams →
  android.view.ViewGroup.MarginLayoutParams →
  android.widget.GridLayout.LayoutParams

- **Include**:

```
android.widget.GridLayout.LayoutParams
```

- **Constructors**:

```
LayoutParams()
LayoutParams(Context context, AttributeSet attrs)
LayoutParams(ViewGroup.LayoutParams params)
LayoutParams(ViewGroup.MarginLayoutParams params)
LayoutParams(GridLayout.LayoutParams source)
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.

2.24 GridLayout.Spec

- **Hierarchy**

  java.lang.Object → android.widget.GridLayout.Spec

- **Include**

```
android.widget.GridLayout.Spec
```

- **Public methods**
  - **boolean equals(Object that)**: Returns true if the class, alignment and span properties of this Spec and the supplied parameter are pairwise equal, false otherwise.
  - **int hashCode()**: Returns a hash code value for the object.

2.25 TableLayout

- **Hierarchy**:

  java.lang.Object → android.view.View → android.view.ViewGroup → android.widget.LinearLayout → android.widget.TableLayout

- **Include**

```
android.widget.TableLayout
```

- **Constructors**

```
TableLayout(Context context)
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.OnHierarchyChangeListener 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.LayoutParams 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)**: Measures the view and its content to determine the measured width and height.

2.26 TableLayout.LayoutParams

- **Hierarchy**

  java.lang.Object → android.view.ViewGroup.LayoutParams →
  android.view.ViewGroup.MarginLayoutParams →
  android.widget.LinearLayout.LayoutParams →
  android.widget.TableLayout.LayoutParams

- **Include**

```
android.widget.TableLayout.LayoutParams
```

- **Constructors**

```
LayoutParams()
LayoutParams(Context c, AttributeSet attrs)
LayoutParams(ViewGroup.LayoutParams p)
LayoutParams(ViewGroup.MarginLayoutParams source)
LayoutParams(int w, int h)
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.

2.27 TableRow

- **Hierarchy**

  java.lang.Object → android.view.View → android.view.ViewGroup → android.widget.LinearLayout → android.widget.TableRow

- **Include**

```
android.widget.TableRow
```

- **Constructors**

```
TableRow(Context context)
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.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, a height of ViewGroup.LayoutParams.WRAP_CONTENT and no spanning.
  - **LinearLayout.LayoutParams generateLayoutParams(ViewGroup.LayoutParams 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)**: Measure the view and its content to determine the measured width and the measured height.

2.28 FrameLayout

- **Hierarchy**

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

- **Include**

```
android.widget.FrameLayout
```

- **Constructors**

```
FrameLayout(Context context)
FrameLayout(Context context, AttributeSet attrs)
FrameLayout(Context context, AttributeSet attrs, int
↪ defStyleAttr, int defStyleRes)
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_PARE`
  - **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.

2.29 FrameLayout.LayoutParams

- **Hierarchy**

  java.lang.Object → android.view.ViewGroup.LayoutParams →
  android.view.ViewGroup.MarginLayoutParams →
  android.widget.FrameLayout.LayoutParams

- **Include**

```
android.widget.FrameLayout.LayoutParams
```

- **Constructors**

```
LayoutParams(Context c, AttributeSet attrs)
LayoutParams(ViewGroup.LayoutParams source)
LayoutParams(ViewGroup.MarginLayoutParams source)
LayoutParams(FrameLayout.LayoutParams source)
LayoutParams(int width, int height)
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.

2.30 ListView

- **Hierarchy**:

  java.lang.Object → android.view.View → android.view.ViewGroup → android.widget.AdapterView<android.widget.ListAdapter> → android.widget.AbsListView → android.widget.ListView

- **Include**

```
android.widget.ListView
```

- **Constructors**

```
ListView(Context context)
ListView(Context context, AttributeSet attrs)
ListView(Context context, AttributeSet attrs, int
    ↪ defStyleAttr)
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.

- **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.

2.31 AdapterView (Abstract)

- **Hierarchy**

  java.lang.Object → android.view.View → android.view.ViewGroup → android.widget.AdapterView<T extends android.widget.Adapter>

- **Include**

```
android.widget.AdapterView<T extends android.widget.Adapter>
```

- **Constructors**

```
AdapterView(Context context)
AdapterView(Context context, AttributeSet attrs)
AdapterView(Context context, AttributeSet attrs, int
↪ defStyleAttr)
AdapterView(Context context, AttributeSet attrs, int
↪ defStyleAttr, int defStyleRes)
```

- **Public methods**
  - **void addView(View child, int index)**: This method is not supported and throws an UnsupportedOperationException when called.
  - **void addView(View child)**: This method is not supported and throws an UnsupportedOperationException when called.
  - **void addView(View child, ViewGroup.LayoutParams params)**: This method is not supported and throws an UnsupportedOperationException when called.
  - **void addView(View child, int index, ViewGroup.LayoutParams params)**: This method is not supported and throws an UnsupportedOperationException when called.
  - **CharSequence getAccessibilityClassName()**: Return the class name of this object to be used for accessibility purposes.
  - **abstract T getAdapter()**: Returns the adapter currently associated with this widget.
  - **int getCount()**:
  - **View getEmptyView()**: When the current adapter is empty, the AdapterView can display a special view called the empty view.
  - **int getFirstVisiblePosition()**: Returns the position within the adapter's data set for the first item displayed on screen.
  - **Object getItemAtPosition(int position)**: Gets the data associated with the specified position in the list.
  - **long getItemIdAtPosition(int position)**:
  - **int getLastVisiblePosition()**: Returns the position within the adapter's data set for the last item displayed on screen.
  - **final AdapterView.OnItemClickListener getOnItemClickListener()**:
  - **final AdapterView.OnItemLongClickListener getOnItemLongClickListener()**:

  - **final AdapterView.OnItemSelectedListener getOnItemSelectedListener()**:
  - **int getPositionForView(View view)**: Returns the position within the adapter's data set for the view, where view is a an adapter item or a descendant of an adapter item.
  - **Object getSelectedItem()**:
  - **long getSelectedItemId()**:
  - **int getSelectedItemPosition()**: Return the position of the currently selected item within the adapter's data set
  - **abstract View getSelectedView()**:
  - **void onProvideAutofillStructure(ViewStructure structure, int flags)**: Populates a ViewStructure to fullfil an autofill request. It also sets the autofill options in the structure; when overridden, it should set it as well, either explicitly by calling ViewStructure.setAutofillOptions(CharSequence[]) or implicitly by calling super.onProvideAutofillStructure(structure, flags).
  - **boolean performItemClick(View view, int position, long id)**: Call the OnItemClickListener, if it is defined.
  - **void removeAllViews()**: This method is not supported and throws an UnsupportedOperationException when called.
  - **void removeView(View child)**: This method is not supported and throws an UnsupportedOperationException when called.
  - **void removeViewAt(int index)**: This method is not supported and throws an UnsupportedOperationException when called.
  - **abstract void setAdapter(T adapter)**: Sets the adapter that provides the data and the views to represent the data in this widget.
  - **void setEmptyView(View emptyView)**: Sets the view to show if the adapter is empty
  - **void setFocusable(int focusable)**: Sets whether this view can receive focus.
  - **void setFocusableInTouchMode(boolean focusable)**: Set whether this view can receive focus while in touch mode.
  - **void setOnClickListener(View.OnClickListener l)**: Register a callback to be invoked when this view is clicked.
  - **void setOnItemClickListener(AdapterView.OnItemClickListener listener)**: Register a callback to be invoked when an item in this AdapterView has been clicked.
  - **void setOnItemLongClickListener(AdapterView.OnItemLongClickListener listener)**: Register a callback to be invoked when an item in this AdapterView has been clicked and held
  - **void setOnItemSelectedListener(AdapterView.OnItemSelectedListener listener)**: Register a callback to be invoked when an item in this AdapterView has been selected.
  - **abstract void setSelection(int position)**: Sets the currently selected item.

- **Protected methods**
  - **boolean canAnimate()**: Indicates whether the view group has the ability to animate its children after the first layout.
  - **void dispatchRestoreInstanceState(SparseArray<Parcelable> container)**: Override to prevent thawing of any views created by the adapter.

  - **void dispatchSaveInstanceState(SparseArray<Parcelable> container)**: Override to prevent freezing of any views created by the adapter.
  - **void onDetachedFromWindow()**: This is called when the view is detached from a window.
  - **void onLayout(boolean changed, int left, int top, int right, int bottom)**: Called from layout when this view should assign a size and position to each of its children.

- **Constants**
  - **int INVALID_POSITION**: Represents an invalid position.
  - **long INVALID_ROW_ID**: Represents an empty or invalid row id
  - **int ITEM_VIEW_TYPE_HEADER_OR_FOOTER**: The item view type returned by Adapter.getItemViewType(int) when the item is a header or footer.
  - **int ITEM_VIEW_TYPE_IGNORE**: The item view type returned by Adapter.getItemViewType(int) when the adapter does not want the item's view recycled.

2.32 Adapter (Interface)

- **Signature**

```
public interface Adapter
```

- **Include**

```
android.widget.Adapter
```

- **Default methods**
  - **default CharSequence[] getAutofillOptions()**: Gets a string representation of the adapter data that can help AutofillService autofill the view backed by the adapter.
- **Abstract methods**
  - **abstract int getCount()**: How many items are in the data set represented by this Adapter.
  - **abstract Object getItem(int position)**: Get the data item associated with the specified position in the data set.
  - **abstract long getItemId(int position)**: Get the row id associated with the specified position in the list.
  - **abstract int getItemViewType(int position)**: Get the type of View that will be created by getView(int, View, ViewGroup) for the specified item.
  - **abstract View getView(int position, View convertView, ViewGroup parent)**: Get a View that displays the data at the specified position in the data set.
  - **abstract int getViewTypeCount()**: Returns the number of types of Views that will be created by getView(int, View, ViewGroup).
  - **abstract boolean hasStableIds()**: Indicates whether the item ids are stable across changes to the underlying data.
  - **abstract boolean isEmpty()**
  - **abstract void registerDataSetObserver(DataSetObserver observer)**: Register an observer that is called when changes happen to the data used by this adapter.
  - **abstract void unregisterDataSetObserver(DataSetObserver observer)**: Unregister an observer that has previously been registered with this adapter via registerDataSetObserver(DataSetObserver).
- **Constants**
  - **int IGNORE_ITEM_VIEW_TYPE**: An item view type that causes the AdapterView to ignore the item view.
  - **int NO_SELECTION**:

2.33 BaseAdapter (Abstract)

- **Hierarchy**

  java.lang.Object → android.widget.BaseAdapter

- **Include**

```
android.widget.BaseAdapter
```

- **Constructors**

```
BaseAdapter()
```

- **Public methods**
  - **boolean areAllItemsEnabled()**: Indicates whether all the items in this adapter are enabled.
  - **CharSequence[] getAutofillOptions()**: Gets a string representation of the adapter data that can help AutofillService autofill the view backed by the adapter.
  - **View getDropDownView(int position, View convertView, ViewGroup parent)**: Gets a View that displays in the drop down popup the data at the specified position in the data set.
  - **int getItemViewType(int position)**: Get the type of View that will be created by getView(int, View, ViewGroup) for the specified item.
  - **int getViewTypeCount()**: Returns the number of types of Views that will be created by getView(int, View, ViewGroup).
  - **boolean hasStableIds()**: Indicates whether the item ids are stable across changes to the underlying data.
  - **boolean isEmpty()**:
  - **boolean isEnabled(int position)**: Returns true if the item at the specified position is not a separator.
  - **void notifyDataSetChanged()**: Notifies the attached observers that the underlying data has been changed and any View reflecting the data set should refresh itself.
  - **void notifyDataSetInvalidated()**: Notifies the attached observers that the underlying data is no longer valid or available.
  - **void registerDataSetObserver(DataSetObserver observer)**: Register an observer that is called when changes happen to the data used by this adapter.
  - **void setAutofillOptions(CharSequence... options)**: Sets the value returned by getAutofillOptions()
  - **void unregisterDataSetObserver(DataSetObserver observer)**: Unregister an observer that has previously been registered with this adapter via registerDataSetObserver(DataSetObserver).

2.34 ArrayAdapter

- **Hierarchy**

java.lang.Object → android.widget.BaseAdapter →
android.widget.ArrayAdapter<T>

- **Include**

```
android.widget.ArrayAdapter<T>
```

- **Constructors**

```
ArrayAdapter(Context context, int resource)
ArrayAdapter(Context context, int resource, int
↪ textViewResourceId)
ArrayAdapter(Context context, int resource, int
↪ textViewResourceId, List<T> objects)
ArrayAdapter(Context context, int resource, int
↪ textViewResourceId, T[] objects)
ArrayAdapter(Context context, int resource, List<T> objects)
ArrayAdapter(Context context, int resource, T[] objects)
```

- **Public methods**
  - **void add(T object)**: Adds the specified object at the end of the array.
  - **void addAll(Collection<? extends T> collection)**: Adds the specified Collection at the end of the array.
  - **void addAll(T... items)**: Adds the specified items at the end of the array.
  - **void clear()**: Remove all elements from the list.
  - **static ArrayAdapter<CharSequence> createFromResource(Context context, int textArrayResId, int textViewResId)**: Creates a new ArrayAdapter from external resources.
  - **CharSequence[] getAutofillOptions()**: Gets a string representation of the adapter data that can help AutofillService autofill the view backed by the adapter.
  - **Context getContext()**: Returns the context associated with this array adapter.
  - **int getCount()**: How many items are in the data set represented by this Adapter.
  - **View getDropDownView(int position, View convertView, ViewGroup parent)**: Gets a View that displays in the drop down popup the data at the specified position in the data set.
  - **Resources.Theme getDropDownViewTheme()**: Returns the value previously set by a call to setDropDownViewTheme(android.content.res.Resources.Theme).
  - **Filter getFilter()**: Returns a filter that can be used to constrain data with a filtering pattern.
  - **T getItem(int position)**: Get the data item associated with the specified position in the data set.
  - **long getItemId(int position)**: Get the row id associated with the specified position in the list.

- **int getPosition(T item)**: Returns the position of the specified item in the array.
- **View getView(int position, View convertView, ViewGroup parent)**: Get a View that displays the data at the specified position in the data set.
- **void insert(T object, int index)**: Inserts the specified object at the specified index in the array.
- **void notifyDataSetChanged()**: Notifies the attached observers that the underlying data has been changed and any View reflecting the data set should refresh itself.
- **void remove(T object)**: Removes the specified object from the array.
- **void setDropDownViewResource(int resource)**: Sets the layout resource to create the drop down views.
- **void setDropDownViewTheme(Resources.Theme theme)**: Sets the Resources.Theme against which drop-down views are inflated.
- **void setNotifyOnChange(boolean notifyOnChange)**: Control whether methods that change the list (add(T), addAll(java.util.Collection), addAll(java.lang.Object[]), insert(T, int), remove(T), clear(), sort(java.util.Comparator)) automatically call notifyDataSetChanged().
- **void sort(Comparator<? super T> comparator)**: Sorts the content of this adapter using the specified comparator.

2.35 TextView

- **Hierarchy**

  java.lang.Object → android.view.View → android.widget.TextView

- **Include**

```
android.widget.TextView
```

- **Constructors**

```
TextView(Context context)
TextView(Context context, AttributeSet attrs)
TextView(Context context, AttributeSet attrs, int
    ↪ defStyleAttr)
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 outText)**: 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.

- **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.

- **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 autosize 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.

2.36 AutoCompleteTextView

- **Hierarchy**

  java.lang.Object → android.view.View → android.widget.TextView → android.widget.EditText → android.widget.AutoCompleteTextView

- **Include**

```
android.widget.AutoCompleteTextView
```

- **Constructors**

```
AutoCompleteTextView(Context context)
AutoCompleteTextView(Context context, AttributeSet attrs)
AutoCompleteTextView(Context context, AttributeSet attrs,
    ↪ int defStyleAttr)
AutoCompleteTextView(Context context, AttributeSet attrs,
    ↪ int defStyleAttr, int defStyleRes)
AutoCompleteTextView(Context context, AttributeSet attrs,
    ↪ int defStyleAttr, int defStyleRes, Resources.Theme
    ↪ popupTheme)
```

- **Public methods**
  - **void clearListSelection()**: Clear the list selection.
  - **void dismissDropDown()**: Closes the drop down if present on screen.
  - **boolean enoughToFilter()**: Returns true if the amount of text in the field meets or exceeds the getThreshold() requirement.
  - **CharSequence getAccessibilityClassName()**: Return the class name of this object to be used for accessibility purposes.
  - **ListAdapter getAdapter()**: Returns a filterable list adapter used for auto completion.
  - **CharSequence getCompletionHint()**: Gets the optional hint text displayed at the bottom of the the matching list.
  - **int getDropDownAnchor()**: Returns the id for the view that the auto-complete drop down list is anchored to.
  - **Drawable getDropDownBackground()**: Gets the background of the auto-complete drop-down list.
  - **int getDropDownHeight()**: Returns the current height for the auto-complete drop down list.
  - **int getDropDownHorizontalOffset()**: Gets the horizontal offset used for the auto-complete drop-down list.
  - **int getDropDownVerticalOffset()**: Gets the vertical offset used for the auto-complete drop-down list.
  - **int getDropDownWidth()**: Returns the current width for the auto-complete drop down list.
  - **int getInputMethodMode()**: Returns the input method mode used by the auto complete dropdown.

- **AdapterView.OnItemClickListener getItemClickListener()**: This method was deprecated in API level 15. Use getOnItemClickListener() intead
- **AdapterView.OnItemSelectedListener getItemSelectedListener()**: This method was deprecated in API level 15. Use getOnItemSelectedListener() intead
- **int getListSelection()**: Get the position of the dropdown view selection, if there is one.
- **AdapterView.OnItemClickListener getOnItemClickListener()**: Returns the listener that is notified whenever the user clicks an item in the drop down list.
- **AdapterView.OnItemSelectedListener getOnItemSelectedListener()**: Returns the listener that is notified whenever the user selects an item in the drop down list.
- **int getThreshold()**: Returns the number of characters the user must type before the drop down list is shown.
- **AutoCompleteTextView.Validator getValidator()**: Returns the Validator set with setValidator(Validator), or null if it was not set.
- **boolean isPerformingCompletion()**: Identifies whether the view is currently performing a text completion, so subclasses can decide whether to respond to text changed events.
- **boolean isPopupShowing()**: Indicates whether the popup menu is showing.
- **void onCommitCompletion(CompletionInfo completion)**: Called by the framework in response to a text completion from the current input method, provided by it calling InputConnection.commitCompletion().
- **void onFilterComplete(int count)**: Notifies the end of a filtering operation.
- **boolean onKeyDown(int keyCode, KeyEvent event)**: Default implementation of KeyEvent.Callback.onKeyDown(): perform press of the view when KeyEvent.KEYCODE_DPAD_CENTER or KeyEvent.KEYCODE_ENTER is released, if the view is enabled and clickable.
- **boolean onKeyPreIme(int keyCode, KeyEvent event)**: Handle a key event before it is processed by any input method associated with the view hierarchy.
- **boolean onKeyUp(int keyCode, KeyEvent event)**: Default implementation of KeyEvent.Callback.onKeyUp(): perform clicking of the view when KeyEvent.KEYCODE_DPAD_CENTER, KeyEvent.KEYCODE_ENTER or KeyEvent.KEYCODE_SPACE is released.
- **void onWindowFocusChanged(boolean hasWindowFocus)**: Called when the window containing this view gains or loses focus.
- **void performCompletion()**: Performs the text completion by converting the selected item from the drop down list into a string, replacing the text box's content with this string and finally dismissing the drop down menu.
- **void performValidation()**: If a validator was set on this view and the current string is not valid, ask the validator to fix it.
- **final void refreshAutoCompleteResults()**: Refreshes the auto complete results.
- **<T extends ListAdapter Filterable> void setAdapter(T adapter)**: Changes the list of data used for auto completion.
- **void setCompletionHint(CharSequence hint)**: Sets the optional hint text that is displayed at the bottom of the the matching list.

- **void setDropDownAnchor(int id)**: Sets the view to which the auto-complete drop down list should anchor.
- **void setDropDownBackgroundDrawable(Drawable d)**: Sets the background of the auto-complete drop-down list.
- **void setDropDownBackgroundResource(int id)**: Sets the background of the auto-complete drop-down list.
- **void setDropDownHeight(int height)**: Sets the current height for the auto-complete drop down list.
- **void setDropDownHorizontalOffset(int offset)**: Sets the horizontal offset used for the auto-complete drop-down list.
- **void setDropDownVerticalOffset(int offset)**: Sets the vertical offset used for the auto-complete drop-down list.
- **void setDropDownWidth(int width)**: Sets the current width for the auto-complete drop down list.
- **void setInputMethodMode(int mode)**: Use this method to specify when the IME should be displayed.
- **void setListSelection(int position)**: Set the position of the dropdown view selection.
- **void setOnClickListener(View.OnClickListener listener)**: Register a callback to be invoked when this view is clicked.
- **void setOnDismissListener(AutoCompleteTextView.OnDismissListener dismissListener)**: Set a listener that will be invoked whenever the AutoCompleteTextView's list of completions is dismissed.
- **void setOnItemClickListener(AdapterView.OnItemClickListener l)**: Sets the listener that will be notified when the user clicks an item in the drop down list.
- **void setOnItemSelectedListener(AdapterView.OnItemSelectedListener l)**: Sets the listener that will be notified when the user selects an item in the drop down list.
- **void setText(CharSequence text, boolean filter)**: Like TextView.setText(java.lang.CharSequence), except that it can disable filtering.
- **void setThreshold(int threshold)**: Specifies the minimum number of characters the user has to type in the edit box before the drop down list is shown.
- **void setValidator(AutoCompleteTextView.Validator validator)**: Sets the validator used to perform text validation.
- **void showDropDown()**: Displays the drop down on screen.

- **Protected methods**
  - **CharSequence convertSelectionToString(Object selectedItem)**: Converts the selected item from the drop down list into a sequence of character that can be used in the edit box.
  - **Filter getFilter()**: Returns the Filter obtained from Filterable.getFilter, or null if setAdapter(T) was not called with a Filterable.
  - **void onAttachedToWindow()**: This is called when the view is attached to a window.
  - **void onDetachedFromWindow()**: This is called when the view is detached from a window.

– **void onDisplayHint(int hint)**: Gives this view a hint about whether is displayed or not.

– **void onFocusChanged(boolean focused, int direction, Rect previouslyFocusedRect)**: Called by the view system when the focus state of this view changes.

– **void performFiltering(CharSequence text, int keyCode)**: Starts filtering the content of the drop down list.

– **void replaceText(CharSequence text)**: Performs the text completion by replacing the current text by the selected item.

– **boolean setFrame(int l, int t, int r, int b)**:

2.37 EditText

- **Hierarchy**

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

- **Include**

```
android.widget.EditText
```

- **Constructors**

```
EditText(Context context)
EditText(Context context, AttributeSet attrs)
EditText(Context context, AttributeSet attrs, int
    ↪ defStyleAttr)
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.

2.38 InputType (Interface)

- **Signature**

```
public interface InputType
```

- **Include**

```
android.text.InputType
```

- **Constants**
  - **int TYPE_CLASS_DATETIME**: Class for dates and times.
  - **int TYPE_CLASS_NUMBER**: Class for numeric text.
  - **int TYPE_CLASS_PHONE**: Class for a phone number.
  - **int TYPE_CLASS_TEXT**: Class for normal text.
  - **int TYPE_DATETIME_VARIATION_DATE**: Default variation of TYPE_CLASS_DATETIME: allows entering only a date.
  - **int TYPE_DATETIME_VARIATION_NORMAL**: Default variation of TYPE_CLASS_DATETIME: allows entering both a date and time.
  - **int TYPE_DATETIME_VARIATION_TIME**: Default variation of TYPE_CLASS_DATETIME: allows entering only a time.
  - **int TYPE_MASK_CLASS**: Mask of bits that determine the overall class of text being given.
  - **int TYPE_MASK_FLAGS**: Mask of bits that provide addition bit flags of options.
  - **int TYPE_MASK_VARIATION**: Mask of bits that determine the variation of the base content class.
  - **int TYPE_NULL**: Special content type for when no explicit type has been specified.
  - **int TYPE_NUMBER_FLAG_DECIMAL**: Flag of TYPE_CLASS_NUMBER: the number is decimal, allowing a decimal point to provide fractional values.
  - **int TYPE_NUMBER_FLAG_SIGNED**: Flag of TYPE_CLASS_NUMBER: the number is signed, allowing a positive or negative sign at the start.
  - **int TYPE_NUMBER_VARIATION_NORMAL**: Default variation of TYPE_CLASS_NUMBER: plain normal numeric text.
  - **int TYPE_NUMBER_VARIATION_PASSWORD**: Variation of TYPE_CLASS_NUMBER: entering a numeric password.
  - **int TYPE_TEXT_FLAG_AUTO_COMPLETE**: Flag for TYPE_CLASS_TEXT: the text editor (which means the application) is performing auto-completion of the text being entered based on its own semantics, which it will present to the user as they type.
  - **int TYPE_TEXT_FLAG_AUTO_CORRECT**: Flag for TYPE_CLASS_TEXT: the user is entering free-form text that should have auto-correction applied to it.
  - **int TYPE_TEXT_FLAG_CAP_CHARACTERS**: Flag for TYPE_CLASS_TEXT: capitalize all characters.

- **int TYPE_TEXT_FLAG_CAP_SENTENCES**: Flag for TYPE_CLASS_TEXT: capitalize the first character of each sentence.
- **int TYPE_TEXT_FLAG_CAP_WORDS**: Flag for TYPE_CLASS_TEXT: capitalize the first character of every word.
- **int TYPE_TEXT_FLAG_ENABLE_TEXT_CONVERSION_SUGGESTIONS**: Flag for TYPE_CLASS_TEXT: Let the IME know the text conversion suggestions are required by the application.
- **int TYPE_TEXT_FLAG_IME_MULTI_LINE**: Flag for TYPE_CLASS_TEXT: the regular text view associated with this should not be multi-line, but when a fullscreen input method is providing text it should use multiple lines if it can.
- **int TYPE_TEXT_FLAG_MULTI_LINE**: Flag for TYPE_CLASS_TEXT: multiple lines of text can be entered into the field.
- **int TYPE_TEXT_FLAG_NO_SUGGESTIONS**: Flag for TYPE_CLASS_TEXT: the input method does not need to display any dictionary-based candidates.
- **int TYPE_TEXT_VARIATION_EMAIL_ADDRESS**: Variation of TYPE_CLASS_TEXT: entering an e-mail address.
- **int TYPE_TEXT_VARIATION_EMAIL_SUBJECT**: Variation of TYPE_CLASS_TEXT: entering the subject line of an e-mail.
- **int TYPE_TEXT_VARIATION_FILTER**: Variation of TYPE_CLASS_TEXT: entering text to filter contents of a list etc.
- **int TYPE_TEXT_VARIATION_LONG_MESSAGE**: Variation of TYPE_CLASS_TEXT: entering the content of a long, possibly formal message such as the body of an e-mail.
- **int TYPE_TEXT_VARIATION_NORMAL**: Default variation of TYPE_CLASS_TEXT: plain old normal text.
- **int TYPE_TEXT_VARIATION_PASSWORD**: Variation of TYPE_CLASS_TEXT: entering a password.
- **int TYPE_TEXT_VARIATION_PERSON_NAME**: Variation of TYPE_CLASS_TEXT: entering the name of a person.
- **int TYPE_TEXT_VARIATION_PHONETIC**: Variation of TYPE_CLASS_TEXT: entering text for phonetic pronunciation, such as a phonetic name field in contacts.
- **int TYPE_TEXT_VARIATION_POSTAL_ADDRESS**: Variation of TYPE_CLASS_TEXT: entering a postal mailing address.
- **int TYPE_TEXT_VARIATION_SHORT_MESSAGE**: Variation of TYPE_CLASS_TEXT: entering a short, possibly informal message such as an instant message or a text message.
- **int TYPE_TEXT_VARIATION_URI**: Variation of TYPE_CLASS_TEXT: entering a URI.
- **int TYPE_TEXT_VARIATION_VISIBLE_PASSWORD**: Variation of TYPE_CLASS_TEXT: entering a password, which should be visible to the user.
- **int TYPE_TEXT_VARIATION_WEB_EDIT_TEXT**: Variation of TYPE_CLASS_TEXT: entering text inside of a web form.
- **int TYPE_TEXT_VARIATION_WEB_EMAIL_ADDRESS**: Variation of TYPE_CLASS_TEXT: entering e-mail address inside of a web form.
- **int TYPE_TEXT_VARIATION_WEB_PASSWORD**: Variation of TYPE_CLASS_TEXT: entering password inside of a web form.

2.39 Button

- **Hierarchy**

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

- **Include**

```
android.widget.Button
```

- **Constructors**

```
Button(Context context)
Button(Context context, AttributeSet attrs)
Button(Context context, AttributeSet attrs, int defStyleAttr)
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.

2.40 ImageView

- **Hierarchy**

  java.lang.Object → android.view.View → android.widget.ImageView

- **Include**

```
android.widget.ImageView
```

- **Constructors**

```
ImageView(Context context)
ImageView(Context context, AttributeSet attrs)
ImageView(Context context, AttributeSet attrs, int
↪ defStyleAttr)
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.

2.41 ImageView.ScaleType (enum)

- **Enum values**
  - **ImageView.ScaleType CENTER**: Center the image in the view, but perform no scaling.
  - **ImageView.ScaleType CENTER_CROP**: Scale the image uniformly (maintain the image's aspect ratio) so that both dimensions (width and height) of the image will be equal to or larger than the corresponding dimension of the view (minus padding).
  - **ImageView.ScaleType CENTER_INSIDE**: Scale the image uniformly (maintain the image's aspect ratio) so that both dimensions (width and height) of the image will be equal to or less than the corresponding dimension of the view (minus padding).
  - **ImageView.ScaleType FIT_CENTER**: Scale the image using Matrix.ScaleToFit.CENTER.
  - **ImageView.ScaleType FIT_END**: Scale the image using Matrix.ScaleToFit.END.
  - **ImageView.ScaleType FIT_START**: Scale the image using Matrix.ScaleToFit.START.
  - **ImageView.ScaleType FIT_XY**: Scale the image using Matrix.ScaleToFit.FILL.
  - **ImageView.ScaleType MATRIX**: Scale using the image matrix when drawing.
- **Public methods**
  - **static ImageView.ScaleType valueOf(String name)**:
  - **static final ScaleType[] values()**:

2.42 ImageButton

- **Hierarchy**

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

- **Include**

```
android.widget.ImageButton
```

- **Constructors**

```
ImageButton(Context context)
ImageButton(Context context, AttributeSet attrs)
ImageButton(Context context, AttributeSet attrs, int
↪ defStyleAttr)
ImageButton(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.
- **Protected methods**
  - **boolean onSetAlpha(int alpha)**: Invoked if there is a Transform that involves alpha.

2.43 CompoundButton

- **Hierarchy**

  java.lang.Object → android.view.View → android.widget.TextView → android.widget.Button → android.widget.CompoundButton

- **Include**

```
android.widget.CompoundButton
```

- **Constructors**

```
CompoundButton(Context context)
CompoundButton(Context context, AttributeSet attrs)
CompoundButton(Context context, AttributeSet attrs, int
↪ defStyleAttr)
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, including 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.

2.44 CheckBox

- **Hierarchy**

  java.lang.Object → android.view.View → android.widget.TextView → android.widget.Button → android.widget.CompoundButton → android.widget.CheckBox

- **Include**

```
android.widget.CheckBox
```

- **Constructors**

```
CheckBox(Context context)
CheckBox(Context context, AttributeSet attrs)
CheckBox(Context context, AttributeSet attrs, int
    ↪ defStyleAttr)
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.

2.45 RadioGroup

- **Hierarchy**

  java.lang.Object → android.view.View → android.view.ViewGroup → android.widget.LinearLayout → android.widget.RadioGroup

- **Include**

```
android.widget.RadioGroup
```

- **Constructors**

```
RadioGroup(Context context)
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.

2.46 RadioGroup.LayoutParams

- **Hierarchy**:

  java.lang.Object → android.view.ViewGroup.LayoutParams →
  android.view.ViewGroup.MarginLayoutParams →
  android.widget.LinearLayout.LayoutParams →
  android.widget.RadioGroup.LayoutParams

- **Include**

```
android.widget.RadioGroup.LayoutParams
```

- **constructors**

```
LayoutParams(Context c, AttributeSet attrs)
LayoutParams(ViewGroup.LayoutParams p)
LayoutParams(ViewGroup.MarginLayoutParams source)
LayoutParams(int w, int h)
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.

2.47 RadioButton

- **Hierarchy**

  java.lang.Object → android.view.View → android.widget.TextView → android.widget.Button → android.widget.CompoundButton → android.widget.RadioButton

- **Include**

```
android.widget.RadioButton
```

- **Constructors**

```
Public constructors
RadioButton(Context context)
RadioButton(Context context, AttributeSet attrs)
RadioButton(Context context, AttributeSet attrs, int
↪ defStyleAttr)
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.

2.48 AbsSpinner

- **Hierarchy**

java.lang.Object → android.view.View → android.view.ViewGroup → android.widget.AdapterView<android.widget.SpinnerAdapter> → android.widget.AbsSpinner

- **Include**

```
android.widget.AbsSpinner
```

- **Constructors**

```
AbsSpinner(Context context)
AbsSpinner(Context context, AttributeSet attrs)
AbsSpinner(Context context, AttributeSet attrs, int
↪ defStyleAttr)
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.

  - **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.

2.49 Spinner

- **Hierarchy**

java.lang.Object → android.view.View → android.view.ViewGroup → android.widget.AdapterView<android.widget.SpinnerAdapter> → android.widget.AbsSpinner → android.widget.Spinner

- **Include**

```
android.widget.Spinner
```

- **Constructors**

```
Spinner(Context context)
Spinner(Context context, AttributeSet attrs)
Spinner(Context context, AttributeSet attrs, int
    ↪ defStyleAttr)
Spinner(Context context, AttributeSet attrs, int
    ↪ defStyleAttr, int mode)
Spinner(Context context, AttributeSet attrs, int
    ↪ defStyleAttr, int defStyleRes, int mode)
Spinner(Context context, AttributeSet attrs, int
    ↪ defStyleAttr, int defStyleRes, int mode, Resources.Theme
    ↪ popupTheme)
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.

- **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.

2.50 Progessbar

- **Hierarchy**

  java.lang.Object → android.view.View → android.widget.ProgressBar

- **Include**

```
android.widget.ProgressBar
```

- **Constructors**

```
ProgressBar(Context context)
ProgressBar(Context context, AttributeSet attrs)
ProgressBar(Context context, AttributeSet attrs, int
    ↪ defStyleAttr)
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.

- **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.

2.51 AbsSeekBar

- **Hierarchy**

java.lang.Object → android.view.View → android.widget.ProgressBar → android.widget.AbsSeekBar

- **Include**

```
android.widget.AbsSeekBar
```

- **Constructors**

```
AbsSeekBar(Context context)
AbsSeekBar(Context context, AttributeSet attrs)
AbsSeekBar(Context context, AttributeSet attrs, int
↪ defStyleAttr)
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 beyond 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.

2.52 SeekBar

- **Hierarchy**

  java.lang.Object → android.view.View → android.widget.ProgressBar → android.widget.AbsSeekBar → android.widget.SeekBar

- **Include**

```
android.widget.SeekBar
```

- **Constructors**

```
SeekBar(Context context)
SeekBar(Context context, AttributeSet attrs)
SeekBar(Context context, AttributeSet attrs, int
    ↪ defStyleAttr)
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.

2.53 Drawable

- **Hierarchy**

  java.lang.Object → android.graphics.drawable.Drawable

- **Include**

```
android.graphics.drawable.Drawable
```

- **Constructors**

```
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.

2.54 GradientDrawable

- **Hierarchy**

  java.lang.Object → android.graphics.drawable.Drawable → android.graphics.drawable.GradientDrawable

- **Include**

```
android.graphics.drawable.GradientDrawable
```

- **Constructors**

```
GradientDrawable()
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.

- **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.

2.55 Intent

- **Hierarchy**

java.lang.Object → android.content.Intent

- **Include**

```
android.content.Intent
```

- **Constructors**

```
Intent()
Intent(Context packageContext, Class<?> cls)
Intent(Intent o)
Intent(String action)
Intent(String action, Uri uri)
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)

2.56 Animation

- **Hierarchy**

  java.lang.Object → android.view.animation.Animation

- **Include**

```
android.view.animation.Animation
```

- **Constructors**

```
Animation()
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.

2.57 AnimationSet

- **Hierarchy**

  java.lang.Object → android.view.animation.Animation →
  android.view.animation.AnimationSet

- **Include**

```
android.view.animation.AnimationSet
```

- **Constructors**

```
AnimationSet(Context context, AttributeSet attrs)
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.

  - **willChangeTransformationMatrix()**: Indicates whether this animation affects the transformation matrix.

- **Protected methods**
  - **AnimationSet clone()**: Creates and returns a copy of this object.

2.58 AlphaAnimation

- **Hierarchy**

  java.lang.Object → android.view.animation.Animation →
  android.view.animation.AlphaAnimation

- **Include**

```
android.view.animation.AlphaAnimation
```

- **Constructors**

```
AlphaAnimation(Context context, AttributeSet attrs)
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

2.59 RotateAnimation

- **Hierarchy**

  java.lang.Object → android.view.animation.Animation → android.view.animation.RotateAnimation

- **Include**

```
android.view.animation.RotateAnimation
```

- **Constructors**

```
RotateAnimation(Context context, AttributeSet attrs)
RotateAnimation(float fromDegrees, float toDegrees)
RotateAnimation(float fromDegrees, float toDegrees, float pivotX, float pivotY)
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.

2.60 ScaleAnimation

- **Hierarchy**

  java.lang.Object → android.view.animation.Animation → android.view.animation.ScaleAnimation

- **Include**

```
android.view.animation.ScaleAnimation
```

- **Constructors**

```
ScaleAnimation(Context context, AttributeSet attrs)
ScaleAnimation(float fromX, float toX, float fromY, float toY)
ScaleAnimation(float fromX, float toX, float fromY, float toY, float pivotX, float pivotY)
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.

2.61 TranslateAnimation

- **Hierarchy**

  java.lang.Object → android.view.animation.Animation → android.view.animation.TranslateAnimation

- **Include**

```
android.view.animation.TranslateAnimation
```

- **Constructors**

```
TranslateAnimation(Context context, AttributeSet attrs)
TranslateAnimation(float fromXDelta, float toXDelta, float fromYDelta, float toYDelta)
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.

2.62 PreferenceManager

- **Include**:

```
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.

2.63 SharedPreferences (interface)

- **Include**

```
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.

2.64 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 SharedPreferences 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.

2.65 Menu (Interface)

- **Include**

```
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

2.66 MenuItem (Interface)

– **Include**

```
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.OnMenuItemClickListener 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.

2.67 ContextMenu (interface)

– **Hierarchy**

android.view.Menu → android.view.ContextMenu

– **Include**

```
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.

2.68 PopupMenu

- **Hierarchy**

  java.lang.Object → android.widget.PopupMenu

- **Include**

```
android.widget.PopupMenu
```

- **Constructors**

```
PopupMenu(Context context, View anchor)
PopupMenu(Context context, View anchor, int gravity)
PopupMenu(Context context, View anchor, int gravity,
    ↪  int popupStyleAttr, int popupStyleRes)
```

- **Public methods**
  - **void dismiss()**: Dismiss the menu popup.
  - **View.OnTouchListener getDragToOpenListener()**: Returns an OnTouchListener 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.OnMenuItemClickListener 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.

2.69 SubMenu (interface)

- **Signature**

```
public interface SubMenu implements Menu
```

- **Include**

```
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.

2.70 MenuInflator

– **Hierarchy**

java.lang.Object → android.view.MenuInflater

– **Include**

```
android.view.MenuInflater
```

– **Constructors**

```
MenuInflater(Context context)
```

– **Public methods**

∗ **void inflate(int menuRes, Menu menu)**: Inflate a menu hierarchy from the specified XML resource.

2.71 Toast

- **Hierarchy**

  java.lang.Object → android.widget.Toast

- **Include**

```
android.widget.Toast
```

- **Constructors**

```
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.

2.72 LayoutInflator (Abstract class)

– **Hierarchy**

java.lang.Object → android.view.LayoutInflater

– **Include**

```
android.view.LayoutInflater
```

– **Constructors (Protected)**

```
LayoutInflater(Context context)
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.

2.73 SQLiteDatabase

– **Hierarchy**

java.lang.Object → android.database.sqlite.SQLiteClosable → android.database.sqlite.SQLiteDatabase

– **Include**

```
0  android.database.sqlite.SQLiteDatabase
```

– **Public methods**

* **void beginTransaction()**: Begins a transaction in EXCLUSIVE mode.
* **void beginTransactionNonExclusive()**: Begins a transaction in IMMEDIATE 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 pathnames 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 ROLLBACK 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.

2.74 Cursor (Interface)

- **Signature** :

```
public interface Cursor implements Closeable
```

- **Include**

```
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).

2.75 ScrollView

– **Hierarchy**

java.lang.Object → android.view.View → android.view.ViewGroup → android.widget.FrameLayout → android.widget.ScrollView

– **Include**

```
android.widget.ScrollView
```

– **Constructors**

```
ScrollView(Context context)
ScrollView(Context context, AttributeSet attrs)
ScrollView(Context context, AttributeSet attrs, int
    ↪ defStyleAttr)
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).

* **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.

2.76 HorizontalScrollView

– **Hierarchy**

java.lang.Object → android.view.View → android.view.ViewGroup → android.widget.FrameLayout → android.widget.HorizontalScrollView

– **Include**

```
android.widget.HorizontalScrollView
```

– **Constructors**

```
HorizontalScrollView(Context context)
HorizontalScrollView(Context context, AttributeSet
    ↪ attrs)
HorizontalScrollView(Context context, AttributeSet
    ↪ attrs, int defStyleAttr)
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.

2.77 InputEvent (Abstract)

- **Hierarchy**

  java.lang.Object → android.view.InputEvent

- **Include**

```
android.view.InputEvent
```

- **Public methods**
  - **int describeContents()**: Describe the kinds of special objects contained in this Parcelable instance's marshaled representation.
  - **final InputDevice getDevice()**: Gets the device that this event came from.
  - **abstract int getDeviceId()**: Gets the id for the device that this event came from.
  - **abstract long getEventTime()**: Retrieve the time this event occurred, in the SystemClock.uptimeMillis() time base.
  - **abstract int getSource()**: Gets the source of the event.
  - **boolean isFromSource(int source)**: Determines whether the event is from the given source.
- **Fields**
  - **public static final Creator<InputEvent> CREATOR**:

2.78 MotionEvent (Extends InputEvent)

- **Hierarchy**

  java.lang.Object → android.view.InputEvent → android.view.MotionEvent

- **Include**

```
0  android.view.MotionEvent
```

- **Public methods**
  - **static String actionToString(int action)**: Returns a string that represents the symbolic name of the specified unmasked action such as "ACTION_DOWN", "ACTION_POINTER_DOWN(3)" or an equivalent numeric constant such as "35" if unknown.
  - **void addBatch(long eventTime, PointerCoords[] pointerCoords, int metaState)**: Add a new movement to the batch of movements in this event.
  - **void addBatch(long eventTime, float x, float y, float pressure, float size, int metaState)**: Add a new movement to the batch of movements in this event.
  - **static int axisFromString(String symbolicName)**: Gets an axis by its symbolic name such as "AXIS_X" or an equivalent numeric constant such as "42".
  - **static String axisToString(int axis)**: Returns a string that represents the symbolic name of the specified axis such as "AXIS_X" or an equivalent numeric constant such as "42" if unknown.
  - **int findPointerIndex(int pointerId)**: Given a pointer identifier, find the index of its data in the event.
  - **int getAction()**: Return the kind of action being performed.
  - **int getActionButton()**: Gets which button has been modified during a press or release action.
  - **int getActionIndex()**: For ACTION_POINTER_DOWN or ACTION_POINTER_UP as returned by getActionMasked(), this returns the associated pointer index.
  - **int getActionMasked()**: Return the masked action being performed, without pointer index information.
  - **float getAxisValue(int axis, int pointerIndex)**: Returns the value of the requested axis for the given pointer index (use getPointerId(int) to find the pointer identifier for this index).
  - **float getAxisValue(int axis)**: getAxisValue(int) for the first pointer index (may be an arbitrary pointer identifier).
  - **int getButtonState()**: Gets the state of all buttons that are pressed such as a mouse or stylus button.
  - **int getClassification()**: Returns the classification for the current gesture.
  - **int getDeviceId()**: Gets the id for the device that this event came from.
  - **long getDownTime()**: Returns the time (in ms) when the user originally pressed down to start a stream of position events.
  - **int getEdgeFlags()**: Returns a bitfield indicating which edges, if any, were touched by this MotionEvent.
  - **long getEventTime()**: Retrieve the time this event occurred, in the SystemClock.uptimeMillis() time base.
  - **long getEventTimeNanos()**: Retrieve the time this event occurred, in the SystemClock.uptimeMillis() time base but with nanosecond precision.

* **int getFlags()**: Gets the motion event flags.
* **float getHistoricalAxisValue(int axis, int pointerIndex, int pos)**: Returns the historical value of the requested axis, as per getAxisValue(int, int), occurred between this event and the previous event for the given pointer.
* **float getHistoricalAxisValue(int axis, int pos)**: getHistoricalAxisValue(int, int, int) for the first pointer index (may be an arbitrary pointer identifier).
* **long getHistoricalEventTime(int pos)**: Returns the time that a historical movement occurred between this event and the previous event, in the SystemClock.uptimeMillis() time base.
* **long getHistoricalEventTimeNanos(int pos)**: Returns the time that a historical movement occurred between this event and the previous event, in the SystemClock.uptimeMillis() time base but with nanosecond (instead of millisecond) precision.
* **float getHistoricalOrientation(int pointerIndex, int pos)**: Returns a historical orientation coordinate, as per getOrientation(int), that occurred between this event and the previous event for the given pointer.
* **float getHistoricalOrientation(int pos)**: getHistoricalOrientation(int, int) for the first pointer index (may be an arbitrary pointer identifier).
* **void getHistoricalPointerCoords(int pointerIndex, int pos, MotionEvent.PointerCoords outPointerCoords)**: Populates a PointerCoords object with historical pointer coordinate data, as per getPointerCoords(int, PointerCoords), that occurred between this event and the previous event for the given pointer.
* **float getHistoricalPressure(int pointerIndex, int pos)**: Returns a historical pressure coordinate, as per getPressure(int), that occurred between this event and the previous event for the given pointer.
* **float getHistoricalPressure(int pos)**: getHistoricalPressure(int, int) for the first pointer index (may be an arbitrary pointer identifier).
* **float getHistoricalSize(int pos)**: getHistoricalSize(int, int) for the first pointer index (may be an arbitrary pointer identifier).
* **float getHistoricalSize(int pointerIndex, int pos)**: Returns a historical size coordinate, as per getSize(int), that occurred between this event and the previous event for the given pointer.
* **float getHistoricalToolMajor(int pos)**: getHistoricalToolMajor(int, int) for the first pointer index (may be an arbitrary pointer identifier).
* **float getHistoricalToolMajor(int pointerIndex, int pos)**: Returns a historical tool major axis coordinate, as per getToolMajor(int), that occurred between this event and the previous event for the given pointer.
* **float getHistoricalToolMinor(int pos)**: getHistoricalToolMinor(int, int) for the first pointer index (may be an arbitrary pointer identifier).
* **float getHistoricalToolMinor(int pointerIndex, int pos)**: Returns a historical tool minor axis coordinate, as per getToolMinor(int), that occurred between this event and the previous event for the given pointer.
* **float getHistoricalTouchMajor(int pos)**: getHistoricalTouchMajor(int, int) for the first pointer index (may be an arbitrary pointer identifier).
* **float getHistoricalTouchMajor(int pointerIndex, int pos)**: Returns a historical touch major axis coordinate, as per getTouchMajor(int), that occurred between this event and the previous event for the given pointer.
* **float getHistoricalTouchMinor(int pointerIndex, int pos)**: Returns a historical touch minor axis coordinate, as per getTouchMinor(int), that occurred between this event and the previous event for the given pointer.

* **float getHistoricalTouchMinor(int pos)**: getHistoricalTouchMinor(int, int) for the first pointer index (may be an arbitrary pointer identifier).
* **float getHistoricalX(int pos)**: getHistoricalX(int, int) for the first pointer index (may be an arbitrary pointer identifier).
* **float getHistoricalX(int pointerIndex, int pos)**: Returns a historical X coordinate, as per getX(int), that occurred between this event and the previous event for the given pointer.
* **float getHistoricalY(int pos)**: getHistoricalY(int, int) for the first pointer index (may be an arbitrary pointer identifier).
* **float getHistoricalY(int pointerIndex, int pos)**: Returns a historical Y coordinate, as per getY(int), that occurred between this event and the previous event for the given pointer.
* **int getHistorySize()**: Returns the number of historical points in this event.
* **int getMetaState()**: Returns the state of any meta / modifier keys that were in effect when the event was generated.
* **float getOrientation()**: getOrientation(int) for the first pointer index (may be an arbitrary pointer identifier).
* **float getOrientation(int pointerIndex)**: Returns the value of AXIS_ORIENTATION for the given pointer index.
* **void getPointerCoords(int pointerIndex, MotionEvent.PointerCoords outPointerCoords)**: Populates a PointerCoords object with pointer coordinate data for the specified pointer index.
* **int getPointerCount()**: The number of pointers of data contained in this event.
* **int getPointerId(int pointerIndex)**: Return the pointer identifier associated with a particular pointer data index in this event.
* **void getPointerProperties(int pointerIndex, MotionEvent.PointerProperties outPointerProperties)**: Populates a PointerProperties object with pointer properties for the specified pointer index.
* **float getPressure(int pointerIndex)**: Returns the value of AXIS_PRESSURE for the given pointer index.
* **float getPressure()**: getPressure(int) for the first pointer index (may be an arbitrary pointer identifier).
* **float getRawX()**: Equivalent to getRawX(int) for pointer index 0 (regardless of the pointer identifier).
* **float getRawX(int pointerIndex)**: Returns the X coordinate of the pointer referenced by pointerIndex for this motion event.
* **float getRawY()**: Equivalent to getRawY(int) for pointer index 0 (regardless of the pointer identifier).
* **float getRawY(int pointerIndex)**: Returns the Y coordinate of the pointer referenced by pointerIndex for this motion event.
* **float getSize(int pointerIndex)**: Returns the value of AXIS_SIZE for the given pointer index.
* **float getSize()**: getSize(int) for the first pointer index (may be an arbitrary pointer identifier).
* **int getSource()**: Gets the source of the event.
* **float getToolMajor(int pointerIndex)**: Returns the value of AXIS_TOOL_MAJOR for the given pointer index.
* **float getToolMajor()**: getToolMajor(int) for the first pointer index (may be an arbitrary pointer identifier).

* **float getToolMinor()**: getToolMinor(int) for the first pointer index (may be an arbitrary pointer identifier).
* **float getToolMinor(int pointerIndex)**: Returns the value of AXIS_TOOL_MINOR for the given pointer index.
* **int getToolType(int pointerIndex)**: Gets the tool type of a pointer for the given pointer index.
* **float getTouchMajor()**: getTouchMajor(int) for the first pointer index (may be an arbitrary pointer identifier).
* **float getTouchMajor(int pointerIndex)**: Returns the value of AXIS_TOUCH_MAJOR for the given pointer index.
* **float getTouchMinor(int pointerIndex)**: Returns the value of AXIS_TOUCH_MINOR for the given pointer index.
* **float getTouchMinor()**: getTouchMinor(int) for the first pointer index (may be an arbitrary pointer identifier).
* **float getX(int pointerIndex)**: Returns the X coordinate of the pointer referenced by pointerIndex for this motion event.
* **float getX()**: Equivalent to getX(int) for pointer index 0 (regardless of the pointer identifier).
* **float getXPrecision()**: Return the precision of the X coordinates being reported.
* **float getY()**: Equivalent to getY(int) for pointer index 0 (regardless of the pointer identifier).
* **float getY(int pointerIndex)**: Returns the Y coordinate of the pointer referenced by pointerIndex for this motion event.
* **float getYPrecision()**: Return the precision of the Y coordinates being reported.
* **boolean isButtonPressed(int button)**: Checks if a mouse or stylus button (or combination of buttons) is pressed.
* **static MotionEvent obtain(long downTime, long eventTime, int action, float x, float y, int metaState)**: Create a new MotionEvent, filling in a subset of the basic motion values.
* **static MotionEvent obtain(long downTime, long eventTime, int action, int pointerCount, float x, float y, float pressure, float size, int metaState, float xPrecision, float yPrecision, int deviceId, int edgeFlags)**: This method was deprecated in API level 15. Use obtain(long, long, int, float, float, float, float, int, float, float, int, int) instead.
* **static MotionEvent obtain(long downTime, long eventTime, int action, int pointerCount, int[] pointerIds, PointerCoords[] pointerCoords, int metaState, float xPrecision, float yPrecision, int deviceId, int edgeFlags, int source, int flags)**: This method was deprecated in API level 15. Use obtain(long, long, int, int, android.view.MotionEvent.PointerProperties[], android.view.MotionEvent.PointerCoords[], int, int, float, float, int, int, int, int) instead.
* **static MotionEvent obtain(long downTime, long eventTime, int action, int pointerCount, PointerProperties[] pointerProperties, PointerCoords[] pointerCoords, int metaState, int buttonState, float xPrecision, float yPrecision, int deviceId, int edgeFlags, int source, int flags)**: Create a new MotionEvent, filling in all of the basic values that define the motion.
* **static MotionEvent obtain(MotionEvent other)**: Create a new MotionEvent, copying from an existing one.

* **static MotionEvent obtain(long downTime, long eventTime, int action, float x, float y, float pressure, float size, int metaState, float xPrecision, float yPrecision, int deviceId, int edgeFlags)**: Create a new MotionEvent, filling in all of the basic values that define the motion.
* **static MotionEvent obtain(long downTime, long eventTime, int action, int pointerCount, PointerProperties[] pointerProperties, PointerCoords[] pointerCoords, int metaState, int buttonState, float xPrecision, float yPrecision, int deviceId, int edgeFlags, int source, int displayId, int flags, int classification)**: Create a new MotionEvent, filling in all of the basic values that define the motion.
* **static MotionEvent obtainNoHistory(MotionEvent other)**: Create a new MotionEvent, copying from an existing one, but not including any historical point information.
* **void offsetLocation(float deltaX, float deltaY)**: Adjust this event's location.
* **void recycle()**: Recycle the MotionEvent, to be re-used by a later caller.
* **void setAction(int action)**: Sets this event's action.
* **void setEdgeFlags(int flags)**: Sets the bitfield indicating which edges, if any, were touched by this MotionEvent.
* **void setLocation(float x, float y)**: Set this event's location.
* **void setSource(int source)**:
* **String toString()**: Returns a string representation of the object.
* **void transform(Matrix matrix)**: Applies a transformation matrix to all of the points in the event.
* **void writeToParcel(Parcel out, int flags)**: Flatten this object in to a Parcel.

– **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.

– **Fields**

* **public static final Creator<MotionEvent> CREATOR**:

– **Constants**

* **int ACTION_BUTTON_PRESS**: Constant for getActionMasked(): A button has been pressed.
* **int ACTION_BUTTON_RELEASE**: Constant for getActionMasked(): A button has been released.
* **int ACTION_CANCEL**: Constant for getActionMasked(): The current gesture has been aborted.
* **int ACTION_DOWN**: Constant for getActionMasked(): A pressed gesture has started, the motion contains the initial starting location.
* **int ACTION_HOVER_ENTER**: Constant for getActionMasked(): The pointer is not down but has entered the boundaries of a window or view.
* **int ACTION_HOVER_EXIT**: Constant for getActionMasked(): The pointer is not down but has exited the boundaries of a window or view.
* **int ACTION_HOVER_MOVE**: Constant for getActionMasked(): A change happened but the pointer is not down (unlike ACTION_MOVE).
* **int ACTION_MASK**: Bit mask of the parts of the action code that are the action itself.
* **int ACTION_MOVE**: Constant for getActionMasked(): A change has happened during a press gesture (between ACTION_DOWN and ACTION_UP).

* **int ACTION_OUTSIDE**: Constant for getActionMasked(): A movement has happened outside of the normal bounds of the UI element.
* **int ACTION_POINTER_1_DOWN**: This constant was deprecated in API level 15. Use ACTION_POINTER_INDEX_MASK to retrieve the data index associated with ACTION_POINTER_DOWN.
* **int ACTION_POINTER_1_UP**: This constant was deprecated in API level 15. Use ACTION_POINTER_INDEX_MASK to retrieve the data index associated with ACTION_POINTER_UP.
* **int ACTION_POINTER_2_DOWN**: This constant was deprecated in API level 15. Use ACTION_POINTER_INDEX_MASK to retrieve the data index associated with ACTION_POINTER_DOWN.
* **int ACTION_POINTER_2_UP**: This constant was deprecated in API level 15. Use ACTION_POINTER_INDEX_MASK to retrieve the data index associated with ACTION_POINTER_UP.
* **int ACTION_POINTER_3_DOWN**: This constant was deprecated in API level 15. Use ACTION_POINTER_INDEX_MASK to retrieve the data index associated with ACTION_POINTER_DOWN.
* **int ACTION_POINTER_3_UP**: This constant was deprecated in API level 15. Use ACTION_POINTER_INDEX_MASK to retrieve the data index associated with ACTION_POINTER_UP.
* **int ACTION_POINTER_DOWN**: Constant for getActionMasked(): A non-primary pointer has gone down.
* **int ACTION_POINTER_ID_MASK**: This constant was deprecated in API level 15. Renamed to ACTION_POINTER_INDEX_MASK to match the actual data contained in these bits.
* **int ACTION_POINTER_ID_SHIFT**: This constant was deprecated in API level 15. Renamed to ACTION_POINTER_INDEX_SHIFT to match the actual data contained in these bits.
* **int ACTION_POINTER_INDEX_MASK**: Bits in the action code that represent a pointer index, used with ACTION_POINTER_DOWN and ACTION_POINTER_UP.
* **int ACTION_POINTER_INDEX_SHIFT**: Bit shift for the action bits holding the pointer index as defined by ACTION_POINTER_INDEX_MASK.
* **int ACTION_POINTER_UP**: Constant for getActionMasked(): A non-primary pointer has gone up.
* **int ACTION_SCROLL**: Constant for getActionMasked(): The motion event contains relative vertical and/or horizontal scroll offsets.
* **int ACTION_UP**: Constant for getActionMasked(): A pressed gesture has finished, the motion contains the final release location as well as any intermediate points since the last down or move event.
* **int AXIS_BRAKE**: Axis constant: Brake axis of a motion event.
* **int AXIS_DISTANCE**: Axis constant: Distance axis of a motion event.
* **int AXIS_GAS**: Axis constant: Gas axis of a motion event.
* **int AXIS_GENERIC_1**: Axis constant: Generic 1 axis of a motion event.
* **int AXIS_GENERIC_10**: Axis constant: Generic 10 axis of a motion event.
* **int AXIS_GENERIC_11**: Axis constant: Generic 11 axis of a motion event.
* **int AXIS_GENERIC_12**: Axis constant: Generic 12 axis of a motion event.

* **int AXIS_GENERIC_13**: Axis constant: Generic 13 axis of a motion event.
* **int AXIS_GENERIC_14**: Axis constant: Generic 14 axis of a motion event.
* **int AXIS_GENERIC_15**: Axis constant: Generic 15 axis of a motion event.
* **int AXIS_GENERIC_16**: Axis constant: Generic 16 axis of a motion event.
* **int AXIS_GENERIC_2**: Axis constant: Generic 2 axis of a motion event.
* **int AXIS_GENERIC_3**: Axis constant: Generic 3 axis of a motion event.
* **int AXIS_GENERIC_4**: Axis constant: Generic 4 axis of a motion event.
* **int AXIS_GENERIC_5**: Axis constant: Generic 5 axis of a motion event.
* **int AXIS_GENERIC_6**: Axis constant: Generic 6 axis of a motion event.
* **int AXIS_GENERIC_7**: Axis constant: Generic 7 axis of a motion event.
* **int AXIS_GENERIC_8**: Axis constant: Generic 8 axis of a motion event.
* **int AXIS_GENERIC_9**: Axis constant: Generic 9 axis of a motion event.
* **int AXIS_GESTURE_PINCH_SCALE_FACTOR**: Axis constant: pinch scale factor of a motion event.
* **int AXIS_GESTURE_SCROLL_X_DISTANCE**: Axis constant: X scroll distance axis of a motion event.
* **int AXIS_GESTURE_SCROLL_Y_DISTANCE**: Axis constant: Y scroll distance axis of a motion event.
* **int AXIS_GESTURE_X_OFFSET**: Axis constant: X gesture offset axis of a motion event.
* **int AXIS_GESTURE_Y_OFFSET**: Axis constant: Y gesture offset axis of a motion event.
* **int AXIS_HAT_X**: Axis constant: Hat X axis of a motion event.
* **int AXIS_HAT_Y**: Axis constant: Hat Y axis of a motion event.
* **int AXIS_HSCROLL**: Axis constant: Horizontal Scroll axis of a motion event.
* **int AXIS_LTRIGGER**: Axis constant: Left Trigger axis of a motion event.
* **int AXIS_ORIENTATION**: Axis constant: Orientation axis of a motion event.
* **int AXIS_PRESSURE**: Axis constant: Pressure axis of a motion event.
* **int AXIS_RELATIVE_X**: Axis constant: The movement of x position of a motion event.
* **int AXIS_RELATIVE_Y**: Axis constant: The movement of y position of a motion event.
* **int AXIS_RTRIGGER**: Axis constant: Right Trigger axis of a motion event.
* **int AXIS_RUDDER**: Axis constant: Rudder axis of a motion event.

* **int AXIS_RX**: Axis constant: X Rotation axis of a motion event.
* **int AXIS_RY**: Axis constant: Y Rotation axis of a motion event.
* **int AXIS_RZ**: Axis constant: Z Rotation axis of a motion event.
* **int AXIS_SCROLL**: Axis constant: Generic scroll axis of a motion event.
* **int AXIS_SIZE**: Axis constant: Size axis of a motion event.
* **int AXIS_THROTTLE**: Axis constant: Throttle axis of a motion event.
* **int AXIS_TILT**: Axis constant: Tilt axis of a motion event.
* **int AXIS_TOOL_MAJOR**: Axis constant: ToolMajor axis of a motion event.
* **int AXIS_TOOL_MINOR**: Axis constant: ToolMinor axis of a motion event.
* **int AXIS_TOUCH_MAJOR**: Axis constant: TouchMajor axis of a motion event.
* **int AXIS_TOUCH_MINOR**: Axis constant: TouchMinor axis of a motion event.
* **int AXIS_VSCROLL**: Axis constant: Vertical Scroll axis of a motion event.
* **int AXIS_WHEEL**: Axis constant: Wheel axis of a motion event.
* **int AXIS_X**: Axis constant: X axis of a motion event.
* **int AXIS_Y**: Axis constant: Y axis of a motion event.
* **int AXIS_Z**: Axis constant: Z axis of a motion event.
* **int BUTTON_BACK**: Button constant: Back button pressed (mouse back button).
* **int BUTTON_FORWARD**: Button constant: Forward button pressed (mouse forward button).
* **int BUTTON_PRIMARY**: Button constant: Primary button (left mouse button).
* **int BUTTON_SECONDARY**: Button constant: Secondary button (right mouse button).
* **int BUTTON_STYLUS_PRIMARY**: Button constant: Primary stylus button pressed.
* **int BUTTON_STYLUS_SECONDARY**: Button constant: Secondary stylus button pressed.
* **int BUTTON_TERTIARY**: Button constant: Tertiary button (middle mouse button).
* **int CLASSIFICATION_AMBIGUOUS_GESTURE**: Classification constant: Ambiguous gesture.
* **int CLASSIFICATION_DEEP_PRESS**: Classification constant: Deep press.
* **int CLASSIFICATION_NONE**: Classification constant: None.
* **int CLASSIFICATION_PINCH**: Classification constant: touchpad pinch.
* **int CLASSIFICATION_TWO_FINGER_SWIPE**: Classification constant: touchpad scroll.
* **int EDGE_BOTTOM**: Flag indicating the motion event intersected the bottom edge of the screen.
* **int EDGE_LEFT**: Flag indicating the motion event intersected the left edge of the screen.
* **int EDGE_RIGHT**: Flag indicating the motion event intersected the right edge of the screen.

* **int EDGE_TOP**: Flag indicating the motion event intersected the top edge of the screen.
* **int FLAG_CANCELED**: This flag is only set for events with ACTION_POINTER_UP and ACTION_CANCEL.
* **int FLAG_WINDOW_IS_OBSCURED**: This flag indicates that the window that received this motion event is partly or wholly obscured by another visible window above it and the event directly passed through the obscured area.
* **int FLAG_WINDOW_IS_PARTIALLY_OBSCURED**: This flag indicates that the window that received this motion event is partly or wholly obscured by another visible window above it and the event did not directly pass through the obscured area.
* **int INVALID_POINTER_ID**: An invalid pointer id.
* **int TOOL_TYPE_ERASER**: Tool type constant: The tool is an eraser or a stylus being used in an inverted posture.
* **int TOOL_TYPE_FINGER**: Tool type constant: The tool is a finger.
* **int TOOL_TYPE_MOUSE**: Tool type constant: The tool is a mouse.
* **int TOOL_TYPE_STYLUS**: Tool type constant: The tool is a stylus.
* **int TOOL_TYPE_UNKNOWN**: Tool type constant: Unknown tool type.

2.79 View.OnTouchListener (Interface)

– **Signature**

```
public static interface View.OnTouchListener
```

– **Include**

```
android.view.View.OnTouchListener
```

– **Abstract methods**

∗ **abstract boolean onTouch(View v, MotionEvent event)**: Called when a touch event is dispatched to a view.

2.80 GestureDetector

– **Hierarchy**

java.lang.Object → android.view.GestureDetector

– **Include**

```
android.view.GestureDetector
```

– **Constructors**

```
GestureDetector(Context context,
    ↪ GestureDetector.OnGestureListener listener)
GestureDetector(Context context,
    ↪ GestureDetector.OnGestureListener listener, Handler
    ↪ handler)
GestureDetector(Context context,
    ↪ GestureDetector.OnGestureListener listener, Handler
    ↪ handler, boolean unused)
GestureDetector(GestureDetector.OnGestureListener
    ↪ listener)
GestureDetector(GestureDetector.OnGestureListener
    ↪ listener, Handler handler)
```

– **Public methods**

* **boolean isLongpressEnabled()**:
* **boolean onGenericMotionEvent(MotionEvent ev)**: Analyzes the given generic motion event and if applicable triggers the appropriate callbacks on the OnGestureListener supplied.
* **boolean onTouchEvent(MotionEvent ev)**: Analyzes the given motion event and if applicable triggers the appropriate callbacks on the OnGestureListener supplied.
* **void setContextClickListener(GestureDetector.OnContextClickListener onContextClickListener)**: Sets the listener which will be called for context clicks.
* **void setIsLongpressEnabled(boolean isLongpressEnabled)**: Set whether longpress is enabled, if this is enabled when a user presses and holds down you get a longpress event and nothing further.
* **void setOnDoubleTapListener(GestureDetector.OnDoubleTapListener onDoubleTapListener)**: Sets the listener which will be called for double-tap and related gestures.

2.81 GestureDetector.OnGestureListener (Interface)

– **Signature**

```
public static interface
↪ GestureDetector.OnGestureListener
```

– **Include**

```
android.view.GestureDetector.OnGestureListener
```

– **Abstract methods**

* **abstract boolean onDown(MotionEvent e)**: Notified when a tap occurs with the down MotionEvent that triggered it.
* **abstract boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY)**: Notified of a fling event when it occurs with the initial on down MotionEvent and the matching up MotionEvent.
* **abstract void onLongPress(MotionEvent e)**: Notified when a long press occurs with the initial on down MotionEvent that trigged it.
* **abstract boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY)**: Notified when a scroll occurs with the initial on down MotionEvent and the current move MotionEvent.
* **abstract void onShowPress(MotionEvent e)**: The user has performed a down MotionEvent and not performed a move or up yet.
* **abstract boolean onSingleTapUp(MotionEvent e)**: Notified when a tap occurs with the up MotionEvent that triggered it.

2.82 GestureDetector.OnDoubleTapListener (Interface)

– **Signature**

```
public static interface
↪ GestureDetector.OnDoubleTapListener
```

– **Include**

```
GestureDetector.OnDoubleTapListener
```

– **Abstract methods**

* **abstract boolean onDoubleTap(MotionEvent e)**: Notified when a double-tap occurs.
* **abstract boolean onDoubleTapEvent(MotionEvent e)**: Notified when an event within a double-tap gesture occurs, including the down, move, and up events.
* **abstract boolean onSingleTapConfirmed(MotionEvent e)**: Notified when a single-tap occurs.

2.83 GestureDetector.SimpleOnGestureListener

– **Hierarchy**

```
0 java.lang.Object $\to$ android.view.GestureDetec⌋
  ↪ tor.SimpleOnGestureListener
```

– **Include**

```
0 android.view.GestureDetector.SimpleOnGestureListener
```

– **Constructors**

```
0 SimpleOnGestureListener()
```

– **Public methods**

* **boolean onContextClick(MotionEvent e)**: Notified when a context click occurs.
* **boolean onDoubleTap(MotionEvent e)**: Notified when a double-tap occurs.
* **boolean onDoubleTapEvent(MotionEvent e)**: Notified when an event within a double-tap gesture occurs, including the down, move, and up events.
* **boolean onDown(MotionEvent e)**: Notified when a tap occurs with the down MotionEvent that triggered it.
* **boolean onFling(MotionEvent e1, MotionEvent e2, float velocityX, float velocityY)**: Notified of a fling event when it occurs with the initial on down MotionEvent and the matching up MotionEvent.
* **void onLongPress(MotionEvent e)**: Notified when a long press occurs with the initial on down MotionEvent that trigged it.
* **boolean onScroll(MotionEvent e1, MotionEvent e2, float distanceX, float distanceY)**: Notified when a scroll occurs with the initial on down MotionEvent and the current move MotionEvent.
* **void onShowPress(MotionEvent e)**: The user has performed a down MotionEvent and not performed a move or up yet.
* **boolean onSingleTapConfirmed(MotionEvent e)**: Notified when a single-tap occurs.
* **boolean onSingleTapUp(MotionEvent e)**: Notified when a tap occurs with the up MotionEvent that triggered it.