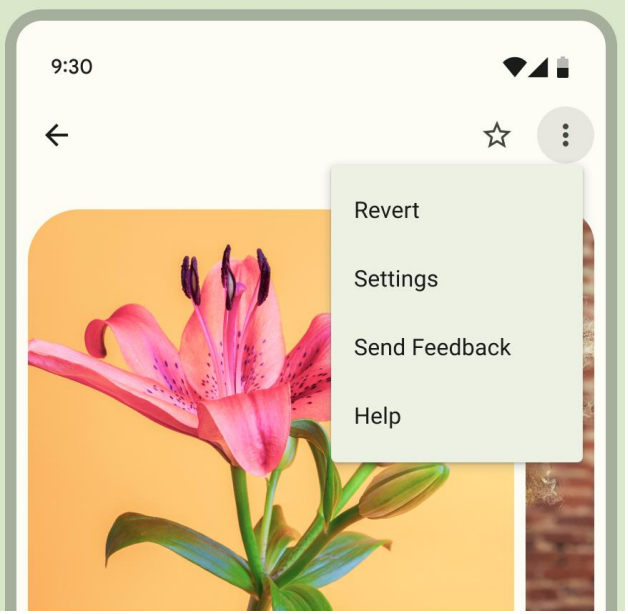
Menus:
Option menu, Context menu, Popup menu

Menu in Android application

- Menus are a common user interface component in many types of apps.
- Three fundamental types:
  - Options menu and app bar
  - Context menu and contextual action mode
  - Popup menu



Define a menu in XML

Android provides a standard XML format to define menu items. You can then inflate the menu resource in your activity or fragment.

Using a menu resource is good practice for the following reasons:

- It's easier to visualize the menu structure in XML.
- It separates the content for the menu from your app's behavioral code.
- It lets you create alternative menu configurations for different platform versions, screen sizes, and other configurations.

Define a menu in XML

To define a menu, create an XML file inside your project's **res/menu/** directory and build the menu with the following elements:

<menu>

Defines a Menu, which is a container for menu items. A **<menu>** element must be the root node for the file, and it can hold one or more **<item>** and **<group>** elements.

<item>

Creates a MenuItem, which represents a single item in a menu. This element can contain a nested **<menu>** element to create a submenu.

<group>

An optional, invisible container for **<item>** elements. It lets you categorize menu items so they share properties, such as active state and visibility. For more information, see the Create a menu group section.

Example menu XML

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
        android:icon="@drawable/ic_new_game"
        android:title="@string/new_game"
        app:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
        android:icon="@drawable/ic_help"
        android:title="@string/help" />
</menu>
```

Create an option menu

- The options menu includes actions and other options that are relevant to the current activity context, such as "Search," "Compose email," and "Settings.“
- Can be declared from Activity subclass or a Fragment subclass. If both activity and fragments declare items for the options menu, the items are combined in the UI.
- To specify the options menu for an activity or fragments, override **onCreateOptionsMenu()**. In this method, menu resource, defined in XML, can be inflated into the Menu provided in the callback.
- When a menu item is selected, the system calls activity's **onOptionsItemSelected()** method.

Option menu example

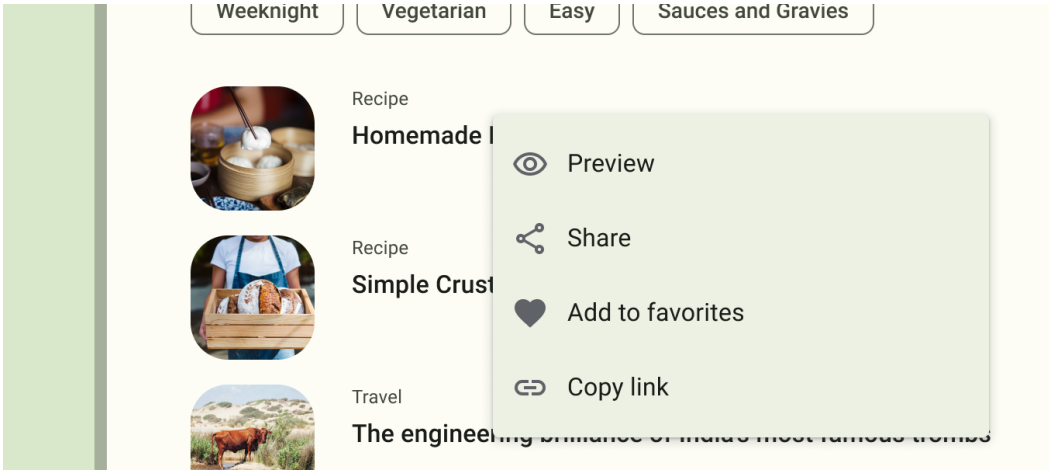
```
override fun onCreateOptionsMenu(menu: Menu): Boolean {
    val inflater: MenuInflater = menuInflater
    inflater.inflate(R.menu.game_menu, menu)
    return true
}

override fun onOptionsItemSelected(item: MenuItem): Boolean {
    // Handle item selection.
    return when (item.itemId) {
        R.id.new_game -> {
            newGame()
            true
        }
        R.id.help -> {
            showHelp()
            true
        }
        else -> super.onOptionsItemSelected(item)
    }
}
```

Create a contextual menu

A contextual menu offers actions that affect a specific item or context frame in the UI.

- Floating context menu: A menu appears as a floating list of menu items, similar to a dialog, when the user performs a touch & hold on a view that declares support for a context menu.
- Contextual action mode: This mode is a system implementation of ActionMode that displays a contextual action bar.



Create a floating context menu

- Register the View the context menu is associated with by calling **registerForContextMenu()** and passing it the View.
- Implement the **onCreateContextMenu()** method in your Activity or Fragment.
- Implement **onContextItemSelected()**. When the user selects a menu item, the system calls this method to perform the appropriate action.

Floating context menu example

```
override fun onCreateContextMenu(menu: ContextMenu, v: View,
                                 menuInfo: ContextMenu.ContextMenuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo)
    val inflater: MenuInflater = menuInflater
    inflater.inflate(R.menu.context_menu, menu)
}

override fun onContextItemSelected(item: MenuItem): Boolean {
    val info = item.menuInfo as AdapterView.AdapterContextMenuInfo
    return when (item.itemId) {
        R.id.edit -> {
            editNote(info.id)
            true
        }
        R.id.delete -> {
            deleteNote(info.id)
            true
        }
        else -> super.onContextItemSelected(item)
    }
}
```

Use the contextual action mode

- The contextual action mode is a system implementation of ActionMode that focuses user interaction toward performing contextual actions.
- When a user enables this mode by selecting an item, a contextual action bar appears at the top of the screen to present actions the user can perform on the selected items.
- While this mode is enabled, the user can select multiple items, if your app supports that, and can deselect items and continue to navigate within the activity.
- The action mode is disabled and the contextual action bar disappears when the user deselects all items, taps the Back button, or taps the Done action on the left side of the bar.

Use the contextual action mode

- For views that provide contextual actions, you usually invoke the contextual action mode when one or both of these two events occurs:
  - The user performs a touch & hold on the view.
  - The user selects a checkbox or similar UI component within the view.
- How your app invokes the contextual action mode and defines the behavior for each action depends on your design. There are two designs:
  - For contextual actions on individual, arbitrary views.
  - For batch contextual actions on groups of items in a RecyclerView, letting the user select multiple items and perform an action on them all.

Contextual action mode example

- Implement the **ActionMode.Callback** interface, specify the actions for the contextual action bar, respond to click events on action items, and handle other lifecycle events for the action mode.

```
private val actionModeCallback = object : ActionMode.Callback {
    override fun onCreateActionMode(mode: ActionMode, menu: Menu): Boolean {
        val inflater: MenuInflater = mode.menuInflater
        inflater.inflate(R.menu.test_menu, menu)
        return true
    }
    override fun onPrepareActionMode(mode: ActionMode, menu: Menu): Boolean {
        return false // Return false if nothing is done
    }
    override fun onActionItemClicked(mode: ActionMode, item: MenuItem): Boolean {
        return when (item.itemId) {
            R.id.action_share -> {
                mode.finish() // Action picked, so close the CAB.
                true
            }
            R.id.action_delete -> {
                mode.finish() // Action picked, so close the CAB.
                true
            }
            else -> false
        }
    }
override fun onDestroyActionMode(mode: ActionMode) {
        actionMode = null
    }
}
```

Contextual action mode example

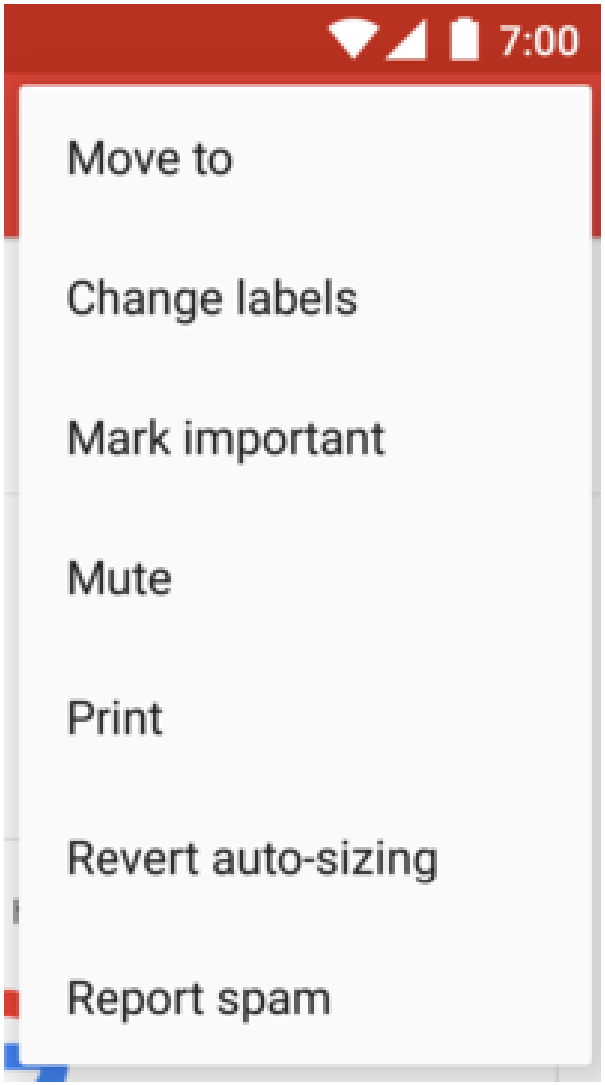
- Call **startActionMode()** to show the bar, such as when the user performs a touch & hold on the view.

```
findViewById<ImageView>(R.id.image_view).setOnLongClickListener {
    when (actionMode) {
        null -> {
            actionMode = this.startSupportActionMode(actionModeCallback)
            true
        }
        else -> false
    }
}
```

Create a popup menu

A PopupMenu is a modal menu anchored to a View. It appears below the anchor view if there is room, or above the view otherwise.

- Providing an overflow-style menu for actions that relate to specific content.
- Providing a second part of a command sentence, such as a button marked Add that produces a popup menu with different Add options.
- Providing a menu similar to a Spinner that doesn't retain a persistent selection.



Create a popup menu

If you define your menu in XML, here's how you can show the popup menu:

- Instantiate a **PopupMenu** with its constructor, which takes the current app **Context** and the **View** to which the menu is anchored.
- Use **MenuInflater** to inflate your menu resource into the **Menu** object returned by **PopupMenu.getMenu()**.
- Call **PopupMenu.show()**.
- Implement the **PopupMenu.OnMenuItemClickListener** interface and register it with PopupMenu by calling **setOnMenuItemclickListener()**. When the user selects an item, the system calls the **onMenuItemClick()** callback.

Popup menu example

```
fun showMenu(v: View) {
    PopupMenu(this, v).apply {
        setOnMenuItemClickListener(this@SecondActivity)
        inflate(R.menu.test_menu)
        show()
    }
}

override fun onMenuItemClick(item: MenuItem): Boolean {
    return when (item.itemId) {
        R.id.action_share -> {
            share(item)
            true
        }
        R.id.action_delete -> {
            delete(item)
            true
        }
        else -> false
    }
}
```