Module - IV

Android Menus

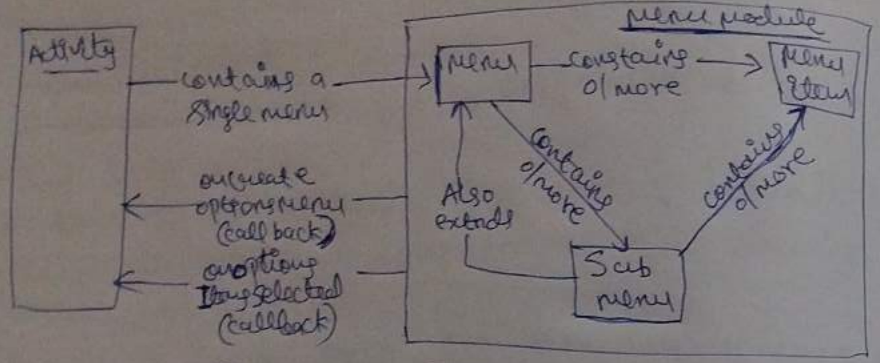


fig : Menu Str.

* menus are a common user interface component in many types of appli~
* The key cls in (An) menu sprt is android. view.Menu
* menu items are reprsnted by android.view. menuItem.
* submenus are reprsnted by android. view.SubMenu
* A menu obj contains a set of menu items.
* Attributes → Name → String title
  → menu item ID → An intgr
  → grp ID → ~~[illegible]~~ you can grp menu items togthr by assigning each 1 a grp ID.
  → Sort ordr → ordr of this menu when it is displayed in the menu.

* using a menu resource for a few reasons-
  * easier to visualize the menu Str in XML
  * Allows to create alternative menu configurations for diffrnt versions, screen size, etc.
* To define the menu, create an XML file inside res/menu/directory with follo- elemnts
* `<menu>` : define a menu, which is a containr for menu items. It must be the root node for the file & can hold 1/more `<item>` & `<group>` elemnt.
* `<item>` : creates a MenuItem, which reprsnts a single item in a menu.
* `<group>` : optional, invisible containr for `<item>` elemnts.

* ~~[illegible]~~ eg →

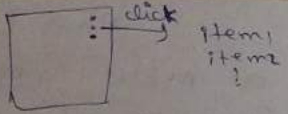
```
<menu>
    <item
        an:id ="@+id/file"
        an:title = "file" />.

    <item
        an:id = "@+id/new
        an:title = "create new" />
        :
</menu>
```

an:icon → A refrnce to a drawble to use as the item's icon.

an:showAsAction → specifies when & how this item should appear as an action item in app bar.

* 3 types —

1) <u>option menu</u> : →



* primary collection of menu items for an activity.
* It's where you should place actions that hue a global impact on the app like search, settings, compose email, etc.
* to make a option menu, we need to override oncreateoptionsMenu() methode—

```
@override
    public boolean oncreateoptionsMenu(Menu
                                      menu){
        //create an inflater obj
        MenuInflater inflater = getMenInflater()
        inflater.inflater(R.menu.menu file, menu);
        return true;
    }
```

used to inflate (convert our xml file into Java obj) the menu-file.xml file.

inflater () used to inflate the menu-file.xml file

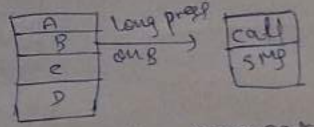
* <u>handling click evnt</u> (user ഒരു menu-ൽ clicks ചെയ്താൽ)
· when the user selects an item from option menu, the system calls activity's <u>onoptionsItemselected () method</u>.
· This method passes the menuItem selected

· The item can be identified by calling <u>getItemid () method</u> (return unique id for the menu item)

override when the () is built-in

```
@override
    public boolean Onoptionsitemselected
                    (MenuItem item) {
        //handle item selection
        switch (item.getItemId ()) {
          case R.id.i1 :
                //perform any action
                return true;
          case R.id.a :
                // ~
                return tue
          default :
                return false ;
        }
    }
```

2) <u>Context Menu</u>: →



* It is a <u>floating menu</u> that appears when the user perform a long click on an element
* provides actions that affect the selected item.
* eg →

```
override
    public void onCreateContextMenu(
                ContextMenu menu, View v,
                ContextMenuInfo menuInfo) {
        super.onCreateContextMenu(
                menu, v, menuInfo);
```

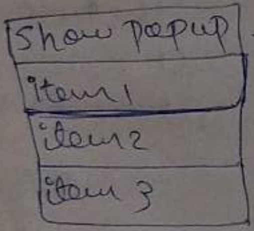
to get default behaviour of the method ←

```
// inflate
menuInflater inflater = getMenuInflater();
inflater.inflates(R.menu.menu_file, menu);
```

when user selects a menu item, the system calls ~~[illegible]~~ OnContextItemSelected()

3) popup menu:

* displays list of items in a vertical list that's anchored ([illegible]) to the view that invoked the menu.
* useful for providing an overflow-style menu for actions that relate to specific content.



→ by clicking show pop up, it displays item 1, 2, 3 –

* eg →

```
<Button
    an:id = ~ button"
    an:l_w ~ , an:l_h ~, an:txt: "btn"
    an:onClick = "popp" />.
```

Java →

```
~~pu~~ public void pop (view v) {
    popupmenu popup = new
            PopupMenu(this, v);
    menuInflater inflates = ~
    inflates.inflate(R.menu.menu_file, popup.getMenu());
    popup.show();
}
```

(onMenuItemClick()
↳ handling click event)

working with menu grps:

* collection of menu items that share certain traits ([illegible])
* show/hide all items with setGroupVisible()
* Enable/disable all items with setGroupEnabled()
* specify whether all items are checkable with setGroupCheckable().

* eg →

```
<menu
    <item
        an:id ~
        an:icon = "menu_save"
        an:title = " menu"   />

    <group an:id ~
     <item an:id ~ ~
          an:title ....

      <item an:id ~
          an:title ~

    </group>

</menu>
```

⇒ Responding to menu items =

A menu item can be associated with an intent by using the menuitem's method setIntent(intent).

I Add action btns :

To add actions to the action bar, create a new XML file in yr project's res/menu/directory. Add <item> element for each item you want to include in the action bar.

II Icon Menu :

menu icons are graphical elemnts placed in options menu.
can be set in XMl file using —

```
<item
    an:id ~
    an:icon = "@android:drawable/plusicon"
    an:title = "Add Contacts" />.
```

II Sub menu:

can be added to an item in any menu by adding <menu> element as the child of an <item>.

eg →
```
<menu>
    <item
        an:id ~
        an:title ~
        <menu>
            <item
                an:id ~
                an:title ~  />
            <item
                an:id ~
                an:title ~ />
        </menu>
    </item>
</menu>
```

III Dynamic menu :

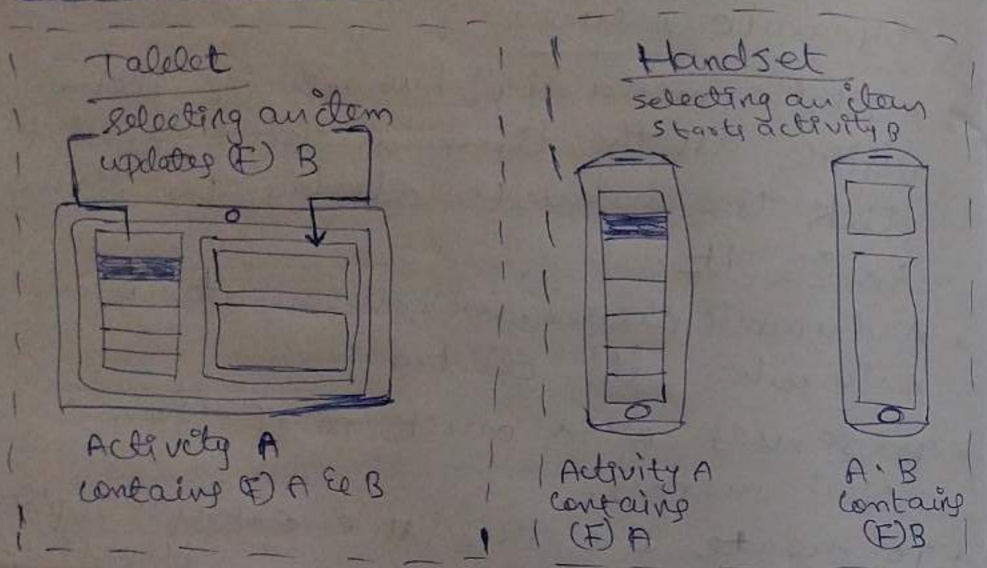
* A static menu will hve all its items visible all the time without need to click the menu to display the items inside it.
* A dynamic menu, will display its items only whn its clicked.
* preserves area much more than static menus.
* To create D. menus, use onprepareOptionsMenu().
used to prepare the standard option menu.

⇒ Fragments in (An) : (F)

* Represnts a behaviour / a option of UI in a FragmntActivity.
* You can combine multiple fragmnts in a single activity to build a multi-pane UI & reuse a fragmnt in multiple activities.
* (F) is a modular section of an activity, which has its own lifecycle.
* A (F) must always be hosted in an

activity & the (F) lifecycle is directly affected by the host activitys lifecycle.

* Structure of (F) :

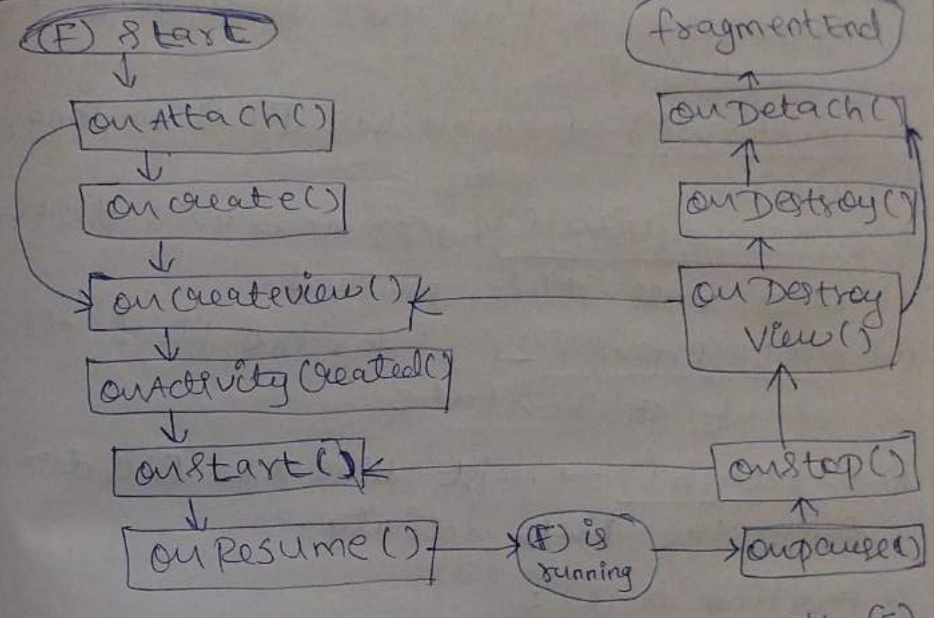


Tablet
selecting an item updates (F) B
Activity A containg (F) A & B
Handset
selecting an item starts activity B
Activity A containg (F) A
A·B containg (F) B

* use —

- Modularity : If a single activity is hving too many (?)-al components, its better to ÷ it into indepndnt fragmnts, hnce making the code more easier to maintain.
- Reusability : If we define any particular feature in a (F), then that feature more or less becomes a reusable component.
- Adaptability : If we break UI components of an app screen into (F), then it becomes easier to change their orientatn & placemnt based on screen size, etc.

* (F) Lifecycle =



1) onAttach() → It is called once, whn the (F) is attached to the activity.
2) onCreate() → the systm calls this method whn a (F) is created.
3) onCreateview() → ~ whn the UI of the (F) has to be initialized.
4) onActivityCreated() → ~ whn the host activity is created. By this time we can access (F)'s view using findViewById().
5) onStart() → ~ whn the (F) becomes visible to the devices screen.
6) onResume() → (F) becomes active.

7) onPause() → ~ whn a (F) is no longer interactive & the user is about to leave the (F).

* 8) onStop() → ~ whn the (F) is no longer visible.

9) onDestroyView() → (F) view will destroy after call this method.

10) onDestroy() → ~ for the final clean up of (F)'s state

11) onDetach() → ~ jst bfre the (F) is detached from the host activity.

* Creating a (F):

• Define a (F) cls, which extends the cls (F)
* & overrides the necessary methods to
• create the (F).

• To provide a layout for a (F), implemnt the onCreateView() callback method.

* eg → public static cls Example(F) extends Fragment {

```
    @override
        public view onCreateView(LayoutInflater
            inflater, ViewGroup containr,
            Bundle savedInstanceState) {
        //Inflate the layout for this (F)
        return inflater.inflate(R.layout.
            example_fragment, container, false);
                    ↳ xml filename
    }
}
```

container → is the parnt viewGrp in which (F) layout is inserted.

inflate() - 3 arg → Ipres id of the layout
→ viewGrp of inflated layout
↳ boolean indicating whethr the inflated layout should be attached to the root during inflation (default → F)

* To add a (F) to any activity in main layout xml —

```
<fragment
    an:name = "com.android.fragments.
                ExampleFragment"

    an:id ~
    an: l_h ~ a: l_w ~ />
</fragment>
```

an:name → specifies (F) cls to instantiate in the layout.

~~[illegible]~~ (You can specify this <fragment> inside linearlayout by specifying ~~[illegible]~~ orientation, l_w, l_h ~)

* (F) Managr :

* manages (F) in (An), specifically it handles transactions b/w (F).

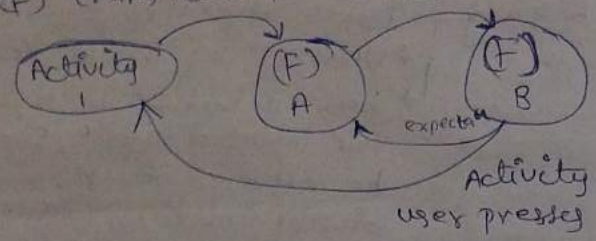
* whn there are several (F) & we want to manipulate several of thm at the same time undr 1 single opr, that kind of interaction can be sprted with the (F) transaction in (An).

* It includes add, remove, replace (F),

commit the transaction.

* (F) transactions & the (F) Back Stack.

*



* (F) do not added to the back stack by default.
* call addToBackStack() bfre calling commit on transaction.
* Saving (F) State:
* (F) hve a onSaveInstanceState() method which is called whn their state needs to be saved.
* It comes under the cls (F).SavedState.

* persistnt (F):

* whn an activity starts a (F) ie soon after the user rotates the screen, causing the activity to be destroyed & recreated.
* (An) does ~~this~~ this so that appli- can reload its resources based on the new configura-
* RetainInstance (pro) can be used to ask the systm not to destroy the (F).
* whn the activity is recreated the (F) is reattached.

* 3 ways a (F) & a activity can communicate-

a) Bundle - Activity can construct a (F) & set arg.

b) Methods - Activity can call methods on a (F) instnce.

c) Listener - (F) can fire listener evnts on an activity via an intrface.

* 82()s —

1) StartActivity () =

* Launch a new activity frm (F).
* startActivity (intent) used to start a new activity, which will be placed at the top of the activity stack.
* Smtimes, you want to get a rslt back frm an activity whn it ends.
* eg → you may start an activity that lets ~~you~~ the user pick a person in a list of contacts, whn it ends, it returns the person that was selected.
* whn an activity exits, it can setResult(int) to return data back to its parent.
* It must always supply a rslt code —

RESULT_CANCELED,
RESULT_OK, RESULT_FIRST_USER.

2) SetTarget(F) ():

* optional target for this (F).

* StartActivityForResult () establishes a relationship b/w 2 activities.
* SetTarget(F) () defines the called relationship b/w 2 (F).

=> Using Dialogs in (An) =

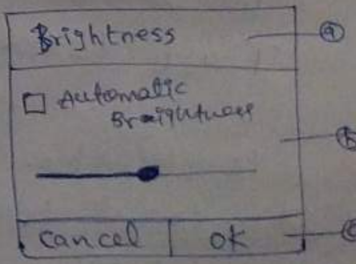
* A dialog is a smaller window that pops up in front of the crnt window to show an urgnt msg, to prompt a user for a piece of input / to show some sort of status like progress of a dwnload, do you want to exit / not, etc.
* Dialogs that are explicitly sprted in (An) include the alert, prompt, pick-list, multiple-choice, progress, etc
* Are Asynchronous in (An), which provides flexibility.
* A dialog does not fill the screen & is normally used for model evnts.

I Dialog (F):

* It is a utility cls which extends the (F) cls.
* It displays / shows a Dialog bt inside a (F).
* cls must extend Dialog(F) with at least onCreateDialog (here you can create the Alert Dialog using the AlertDialog.Builder cls) or onCreateView (here you can create a Dialog using a custom view defined).

* Building an Alert Dialog:
  3 regions of alert dialog—

1) Title → optional & should be used only when the content area is occupied by a detailed msg, a list / custom layout.
2) content area → display a msg, a list / custom layout.
3) Action btns → There should be no more than 3 action btns in a dialog [ok, cancel, neutral option (remaind me later)]
   3 diffrnt action btns —
a) +ve → use this to accept & continue with the action (ok action)
b) -ve → use this to cancel the action.
c) Neutral → use this whn user may not want to proceed with the action.



* 3 kinds of lists →
  * A traditional single-choice list
  * " " (radio btn)
  * A persistent multiple-choice list (Checkbox) ⟹

II working with Toast :

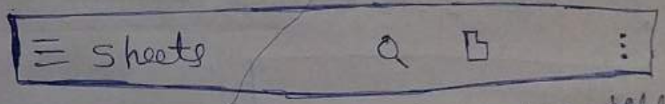
(msg snd ചെയ്താൽ 'sending message ....' എന്ന് കാണിക്കുന്നത്)

* A toast provides simple feedback about an opr in a small popup.
* It only fills the amount of space required for the msg & the crnt activity remains visible & interactive.
* Toast automatically disappear after a timeout
* takes 3 paramters → appli^n context
  → txt msg
  → duration of the Toast
* eg →
```
Context context = getApplicationContext();
CharSequence text = "Hello toast ";
int duration = Toast.LENGTH_SHORT;
Toast toast = Toast.maketext(context,
          text, duration);
toast.show ();
```
* A standard toast notifica^n appears near the bottom of the screen, centered horizontally.

This position can be changed with the setGravity (int, int, int) method.

Action Bar =

* It is an imp design elemnt, usually at the top of each screen in an app, that provides better user intraction & experience by sprting easy navigation through tabs / drop-dwn lists.

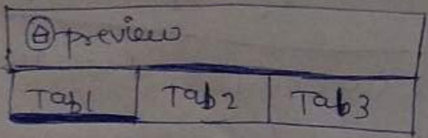


* Allows you to customize the title bar of an activity.
* Access to imp actions in a predictable way such as search.
* Sprt for naviga^n & view switching.
* Most recent features are added to the sprt library's version of Toolbar & they are available on any device.
*
```
@override
protected void onCreate(Bundle savedstate){
    super.onCreate (savedstate);
    set ContentView (R.layout.activity_my);
    Toolbar mybar= (Toolbar) findviewById
        (R.id.my_bar);
    setSupportActionBar(mybar);
}
```

This method sets the toolbar as the app bar for the activity.

I Tapped Navigation action bar activity:

* In tab navig^n, tabs appear across the top of a screen, providing navig^n to othr screens.
* Tabs are most appropriate for 4 / fewer sibling screens.
* The user can tap a tab to see a diffrnt screen, / swipe L/R to see diffrnt screen



* Tablayout used to implemnt horizontal tabs.
* popul^n of the tabs to display through Tablayout.Tab instances,
* To display tab, you need to add it to the layout via 1 of the addTab(Tab) methods-

```
Tablayout tab = (Tablayout) findViewById
                         (R.id.tab);
tab.addTab(tab.newTab().setText("Tab1"));
   "        "                  Tab2 "
   "        "                  Tab3 "
```

* Items can be added to Tablayout through the use of TabItem —

```
<com.google.android.material.tabs.TabLayout
      an:L_h = "w_c"
      an:L_w = "m_p" >
<com.google.android.material.tabs.
                              TabItem
          an:text = "Tab1" />
<com.google  — —   — —
          an:icon = "@drawable/icon_ic" />
</com.google.android.material.tabs.Tablayout>
```

Add a listner using addOnTabSelectedListener (OnTabSelectedListener) to be notified when any tab's selection state has been changed.

II Debug textview Layout :

* A txtview displays txt to the user & optionally allows them to edit it.
* A txtview is a complete txt editor.
* ~~Attributes~~ —

an:id, an:capitalize → 0 → don't automatically capi-
an:hint, an:gravity, an:fontFamily, → 1 → 1st word of each sentnce capi-
an:maxHeight, an:maxwidth, → 2 → 1st letter of every word capi-
an:password, an:phonenumber → 3 → 3 → Every char capi-
— —

⇒ List navigation & action bar activity:

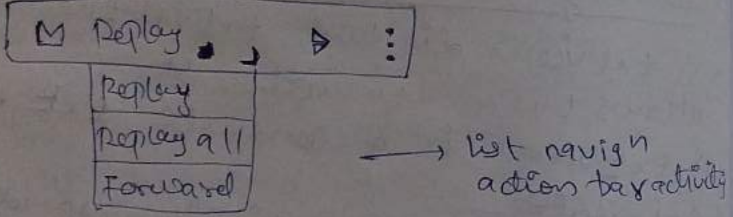
* To initialize the action bar with list navigan mode follo- things are needed-

1) A Spinner adapter:

* Extended adapter that is the bridge b/w a spinner & its data.
* It provides a quick way to select 1 value frm a set.

2) List listeners:

* whn the user selects an item in the list, the system calls onItemClick() & onItemClickListener.
* The content of onItemClick() method depends on the implemn of the app.



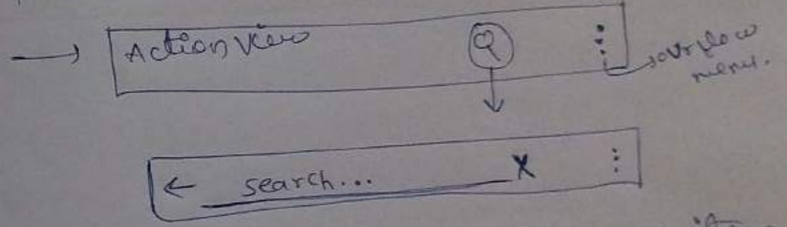
Standard navign action bar activity:

* Tabbed listeners are used to set up the tabbed action bar & list listeners are used for setting up the list navign actionbar
* For a standard action bar, there are no listeners othr than the men callbacks.
* To implemnt s.a.bar use foll~ code—
actionbar.setNavigationMode(actionbar.
NAVIGATION_MODE_STANDARD);

I Action bar & Search View:

To add a SearchView widget to the app bar, create a file named res/menu/options_menu.xml in our project & add follo~ code—

```
<menu
<item
        an:id —
        an:title = "search
        an:ShowAsAction = ifRoom/>

  </menu>
```



an: ShowAsAction → defines hw a menu item should be displayed on the action bar.
ifRoom(default) → displays the item as a btn only if there's enough space on the action bar.