MOBILE APPLICATION DEVELOPMENT (MAD): LAB 4

OBJECTIVES

- Understanding Android Action Bar Menus
- Understanding Context Menus
- Practice Activities

OBJECTIVE 1: Understanding Android Action Bar Menus

Defining a Menu in XML

For all menu types, Android provides a standard XML format to define menu items. Instead of building a menu in your activity's code, you should define a menu and all its items in an XML menu resource. You can then inflate the menu resource (load it as a Menu object) in your activity or fragment.

Using a menu resource is a good practice for a few reasons:

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

To define the 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 can hold one or more <item> and <group> elements.

`<item>`

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

`<group>`

An optional, invisible container for <item> elements. It allows you to categorize menu items so they share properties such as active state and visibility. For more information, see the section about Creating Menu Groups.

MOBILE APPLICATION DEVELOPMENT (MAD): LAB 4

```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"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
          android:icon="@drawable/ic_help"
          android:title="@string/help" />
</menu>
```

Figure 1 - Sample Menu Code

The `<item>` element supports several attributes you can use to define an item's appearance and behavior. The items in the above menu include the following attributes:

`android:id`
A resource ID that's unique to the item, which allows the application to recognize the item when the user selects it.

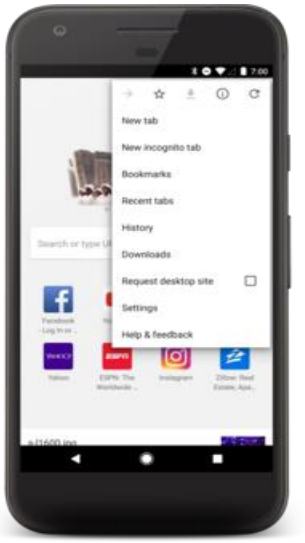
`android:icon`
A reference to a drawable to use as the item's icon.

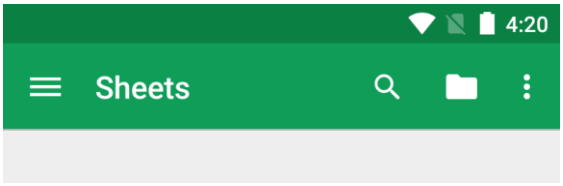
`android:title`
A reference to a string to use as the item's title.

`android:showAsAction`
Specifies when and how this item should appear as an action item in the app bar

To use the menu in your activity, you need to inflate the menu resource (convert the XML resource into a programmable object) using `MenuInflater.inflate()`. In the following sections, you'll see how to inflate a menu for each menu type.

- If you've developed your application for Android 2.3.x (API level 10) or lower, the contents of your options menu appear at the top of the screen when the user presses the Menu button, as shown in figure 1. When opened, the first visible portion is the icon menu, which holds up to six menu items. If your menu includes more than six items, Android places the sixth item and the rest into the overflow menu, which the user can open by selecting More.





- If you've developed your application for Android 3.0 (API level 11) and higher, items from the options menu are available in the app bar. By default, the system places all items in the action overflow, which the user can reveal with the action overflow icon on the right side of the app bar (or by pressing the device Menu button, if available). To enable quick access to important actions, you can promote a few items to appear in the app bar by adding android:showAsAction="ifRoom" to the corresponding <item> elements (see figure 2).

To specify the options menu for an activity, override onCreateOptionsMenu() (fragments provide their own `onCreateOptionsMenu()` callback). In this method, you can inflate your menu resource (defined in XML) into the Menu provided in the callback. For example:

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

MOBILE APPLICATION DEVELOPMENT (MAD): LAB 4

Handling click events

When the user selects an item from the options menu (including action items in the app bar), the system calls your activity's `onOptionsItemSelected()` method. This method passes the `MenuItem` selected. You can identify the item by calling `getItemId()`, which returns the unique ID for the menu item (defined by the `android:id` attribute in the menu resource or with an integer given to the add() method). You can match this ID against known menu items to perform the appropriate action. For example:

```
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)
    }
}
```

MOBILE APPLICATION DEVELOPMENT (MAD): LAB 4

OBJECTIVE 2: Understanding Context Menus

A contextual menu offers actions that affect a specific item or context frame in the UI. You can provide a context menu for any view, but they are most often used for items in a `ListView`, `GridView`, or other view collections in which the user can perform direct actions on each item.

There are two ways to provide contextual actions:

- In a floating context menu. A menu appears as a floating list of menu items (similar to a dialog) when the user performs a long-click (press and hold) on a view that declares support for a context menu. Users can perform a contextual action on one item at a time.
- In the contextual action mode. This mode is a system implementation of ActionMode that displays a contextual action bar at the top of the screen with action items that affect the selected item(s). When this mode is active, users can perform an action on multiple items at once (if your app allows it).

Creating a floating context menu

To provide a floating context menu:

1. Register the `View` to which the context menu should be associated by calling `registerForContextMenu`() and pass it the View.

   If your activity uses a `ListView` or `GridView` and you want each item to provide the same context menu, register all items for a context menu by passing the `ListView` or `GridView` to `registerForContextMenu`().

2. Implement the onCreateContextMenu() method in your Activity or Fragment.

   When the registered view receives a long-click event, the system calls your onCreateContextMenu() method. This is where you define the menu items, usually by inflating a menu resource. For 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)
}
```

3. Implement onContextItemSelected().
   When the user selects a menu item, the system calls this method so you can perform the appropriate action. For example

```
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)
    }
}
```

MOBILE APPLICATION DEVELOPMENT (MAD): LAB 4

Using 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 currently selected item(s). While this mode is enabled, the user can select multiple items (if you allow it), deselect items, and continue to navigate within the activity (as much as you're willing to allow). The action mode is disabled and the contextual action bar disappears when the user deselects all items, presses the BACK button, or selects the Done action on the left side of the bar

> **Note:** The contextual action bar is not necessarily associated with the app bar. They operate independently, even though the contextual action bar visually overtakes the app bar position.

For views that provide contextual actions, you should usually invoke the contextual action mode upon one of two events (or both):

1. The user performs a long-click on the view.
2. The user selects a checkbox or similar UI component within the view.

How your application invokes the contextual action mode and defines the behavior for each action depends on your design. There are basically two designs:

1. For contextual actions on individual, arbitrary views.
2. For batch contextual actions on groups of items in a ListView or GridView (allowing the user to select multiple items and perform an action on them all).

Enabling the contextual action mode for individual views

If you want to invoke the contextual action mode only when the user selects specific views, you should:

1. Implement the ActionMode.Callback interface. In its callback methods, you can specify the actions for the contextual action bar, respond to click events on action items, and handle other lifecycle events for the action mode.
2. Call startActionMode() when you want to show the bar (such as when the user long-clicks the view).

MOBILE APPLICATION DEVELOPMENT (MAD): LAB 4

Implementing **ActionMode.Callback**

```
private val actionModeCallback = object : ActionMode.Callback {
    // Called when the action mode is created; startActionMode() was called
    override fun onCreateActionMode(mode: ActionMode, menu: Menu): Boolean {
        // Inflate a menu resource providing context menu items
        val inflater: MenuInflater = mode.menuInflater
        inflater.inflate(R.menu.context_menu, menu)
        return true
    }

    // Called each time the action mode is shown. Always called after onCreateActionMode, but
    // may be called multiple times if the mode is invalidated.
    override fun onPrepareActionMode(mode: ActionMode, menu: Menu): Boolean {
        return false // Return false if nothing is done
    }

    // Called when the user selects a contextual menu item
    override fun onActionItemClicked(mode: ActionMode, item: MenuItem): Boolean {
        return when (item.itemId) {
            R.id.menu_share -> {
                shareCurrentItem()
                mode.finish() // Action picked, so close the CAB
                true
            }
            else -> false
        }
    }

    // Called when the user exits the action mode
    override fun onDestroyActionMode(mode: ActionMode) {
        actionMode = null
    }
}
```

Call startActionMode() to enable the contextual action mode when appropriate, such as in response to a long-click on a View:

```
someView.setOnLongClickListener { view ->
    // Called when the user long-clicks on someView
    when (actionMode) {
        null -> {
            // Start the CAB using the ActionMode.Callback defined above
            actionMode = activity?.startActionMode(actionModeCallback)
            view.isSelected = true
            true
        }
        else -> false
    }
}
```

MOBILE APPLICATION DEVELOPMENT (MAD): LAB 4

Objective 3: Practice Activities

Activity 1 : Upgrade Activity 1 of Lab 4

Add an Action Menu bar to log out the user, and move the user to main activity. And also remove all activities from memory.

MOBILE APPLICATION DEVELOPMENT (MAD): LAB 4

Activity 2 : Create floating context menu and Contextual Action Bar

- Task 1 : Create an app with a single picture in center, on holding which the menu on left should be created.
- Task 2 : Create a Contextual Action Bar shown on right side of the picture.

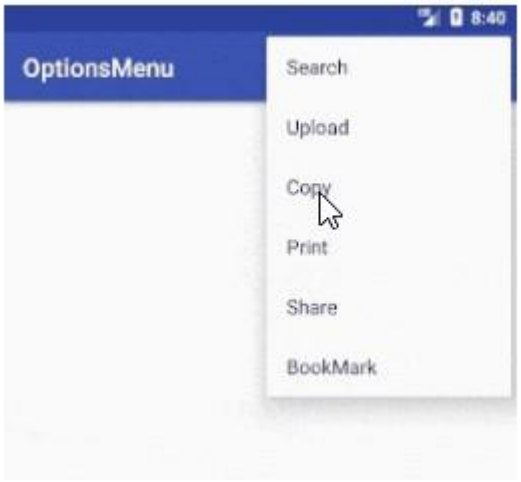
MOBILE APPLICATION DEVELOPMENT (MAD): LAB 4

Activity 3: Create the menu bar shown in below picture

Add following options in the menu

- **Music On / Off button in Menu (To turn on and turn off Music, which will be played from internet)**
- **Speak Button (To speak a sentence written in textbox)**

Add both as "showAsAction = Always", use suitable icons.



MOBILE APPLICATION DEVELOPMENT (MAD): LAB 4

Activity 4: Create context menu shown below