Mobile Programming

[week 05]

[Apps Menu]

Hilmy A. T.

hilmi.tawakal@gmail.com

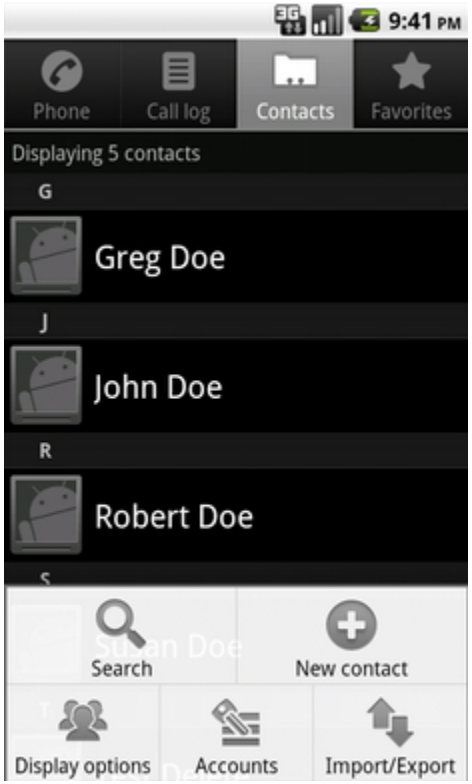
Menus

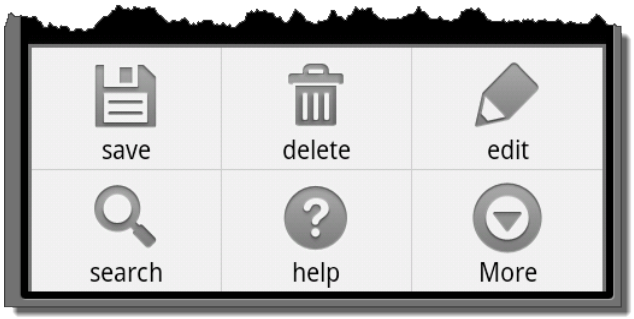
- Menus are a common user interface component in many types of applications.
- To provide a familiar and consistent user experience, you should use the Menu APIs to present user actions and other options in your activities.
- Beginning with Android 3.0 (API level 11), Android-powered devices are no longer required to provide a dedicated Menu button.

Menus

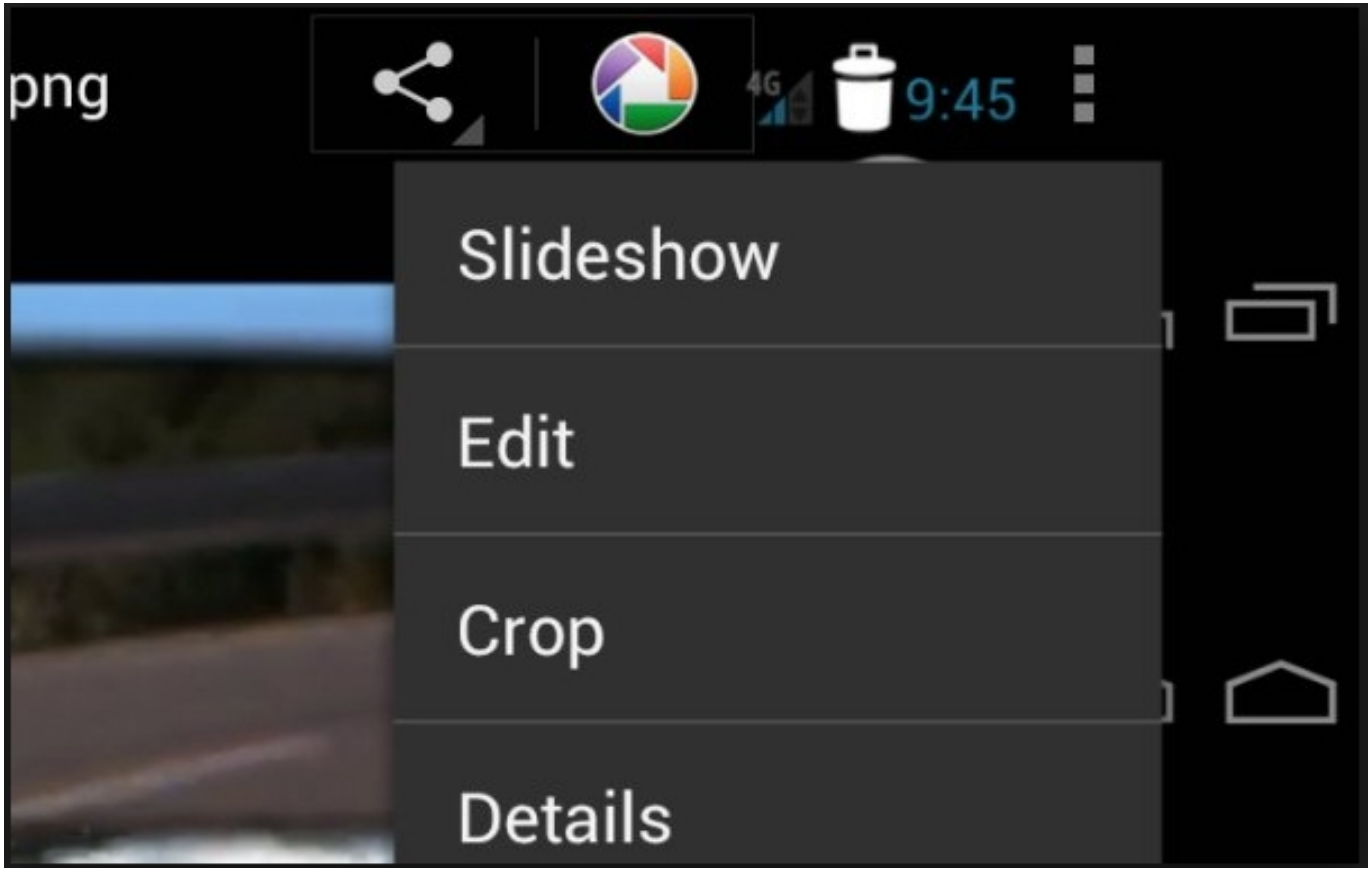
- With this change, Android apps should migrate away from a dependence on the traditional 6-item menu panel and instead provide an action bar to present common user actions.
- Although the design and user experience for some menu items have changed, the semantics to define a set of actions and options is still based on the Menu APIs.

“Old” Option Menu

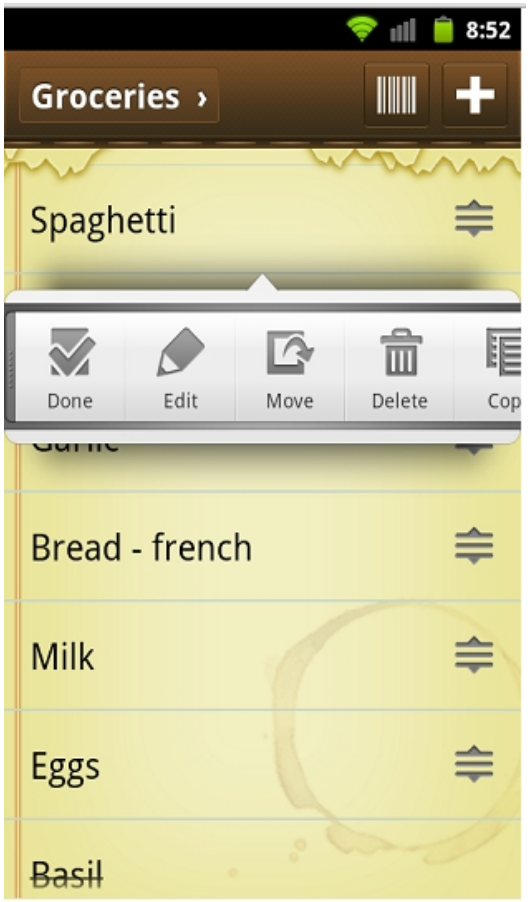




“New” Option Menu

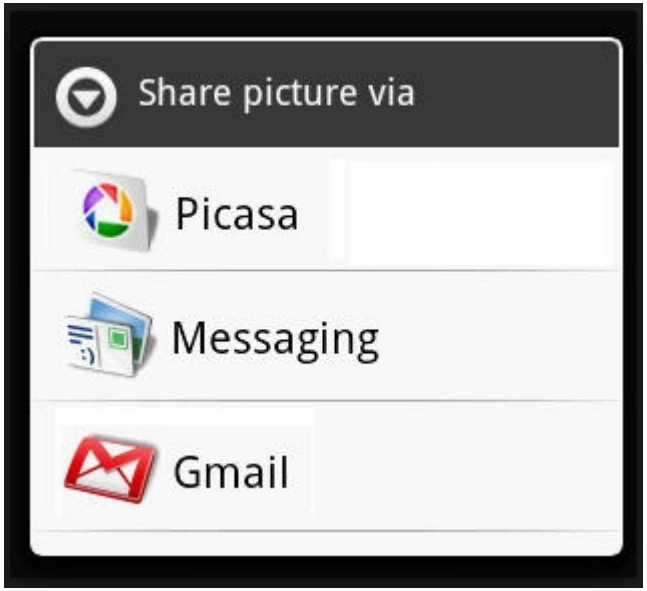


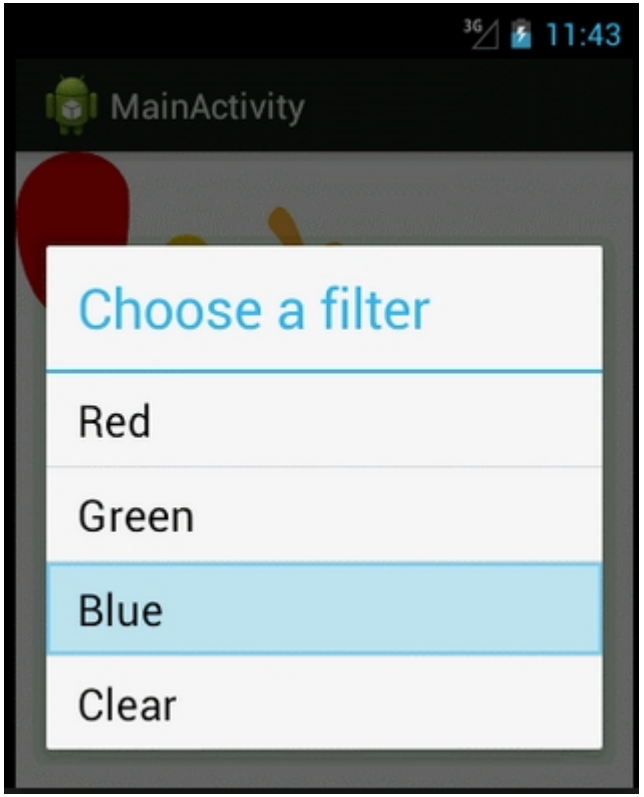
Popup Menu


8:52
Groceries ›
Spaghetti
Done
Edit
Move
Delete
Bread - french
Milk
Eggs
Basil

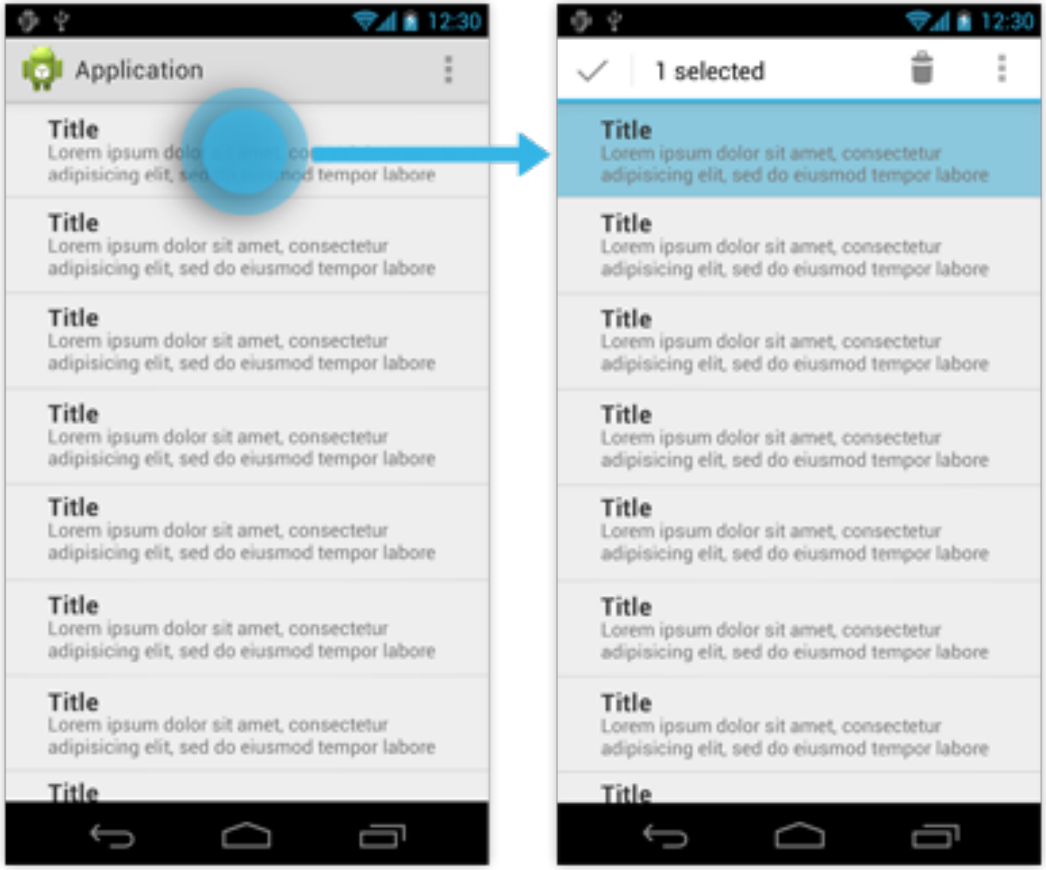

17:02
Tutorial Context Rail & Popup Menu
www.anddev.it
Button 1
Button 2
Button 3
Block
Delete
Direct Message
Edit List
Favorite
Location Pin

Floating Contextual Menu





Contextual Action Mode



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.

Defining a Menu in XML

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.

Defining a Menu in XML

<menu> : Defines a menu (must be root)

<item> : Create MenuItem

<group> : An optional, invisible container

```
<?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>
```

Defining a Menu in XML

The <item> element attributes:

android:id

A resource ID that's unique to the item, which allows the application can 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 action bar.

Defining a Menu in XML

To add submenu:

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/file"
          android:title="@string/file" >
        <!-- "file" submenu -->
        <menu>
            <item android:id="@+id/create_new"
                  android:title="@string/create_new" />
            <item android:id="@+id/open"
                  android:title="@string/open" />
        </menu>
    </item>
</menu>
```

Creating an Options Menu

- The options menu is where you should include actions and other options that are relevant to the current activity context, such as "Search," "Compose email," and "Settings."
- Where the items in your options menu appear on the screen depends on the version for which you've developed your application

Creating an Options Menu

- 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 bottom of the screen when the user presses the Menu button
- 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.

Creating an Options Menu

- If you've developed your application for Android 3.0 (API level 11) and higher, items from the options menu are available in the action 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 action 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 action bar by adding `android:showAsAction="ifRoom"` to the corresponding `<item>` elements

Creating an Options Menu

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

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.game_menu, menu);
    return true;
}
```

Handling click events

- When the user selects an item from the options menu (including action items in the action 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.

Handling click events

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.new_game:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

Creating Contextual 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.

Creating Contextual Menus

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.
3. Implement onContextItemSelected().

Creating a floating context menu

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
    super.onCreateContextMenu(menu, v, menuInfo);
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.context_menu, menu);
}
```

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();
    switch (item.getItemId()) {
        case R.id.edit:
            editNote(info.id);
            return true;
        case R.id.delete:
            deleteNote(info.id);
            return true;
        default:
            return super.onContextItemSelected(item);
    }
}
```

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).

Using the contextual action mode

- 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.

Using the contextual action mode

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

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

Using the contextual action mode

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

- For contextual actions on individual, arbitrary views.
- 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

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).

Enabling batch contextual actions in a ListView or GridView

- Implement the AbsListView.MultiChoiceModeListener interface and set it for the view group with setMultiChoiceModeListener(). In the listener's callback methods, you can specify the actions for the contextual action bar, respond to click events on action items, and handle other callbacks inherited from the ActionMode.Callback interface.
- Call setChoiceMode() with the CHOICE_MODE_MULTIPLE_MODAL argument.

Creating 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. It's useful for:

- Providing an overflow-style menu for actions that relate to specific content
- Providing a second part of a command sentence
- Providing a drop-down similar to Spinner that does not retain a persistent selection.

Creating Popup Menu

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

1. Instantate a PopupMenu with its constructor, which takes the current application Context and the View to which the menu should be anchored.

2.Use MenuInflater to inflate your menu resource into the Menu object returned by PopupMenu.getMenu(). On API level 14 and above, you can use PopupMenu.inflate() instead.

3.Call PopupMenu.show().

Creating Popup Menu

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_overflow_holo_dark"
    android:contentDescription="@string/descr_overflow_button"
    android:onClick="showPopup" />
```

```
public void showPopup(View v) {
    PopupMenu popup = new PopupMenu(this, v);
    MenuInflater inflater = popup.getMenuInflater();
    inflater.inflate(R.menu.actions, popup.getMenu());
    popup.show();
}
```

Creating Popup Menu

```
public void showMenu(View v) {
    PopupMenu popup = new PopupMenu(this, v);

    // This activity implements OnMenuItemClickListener
    popup.setOnMenuItemClickListener(this);
    popup.inflate(R.menu.actions);
    popup.show();
}

@Override
public boolean onMenuItemClick(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.archive:
            archive(item);
            return true;
        case R.id.delete:
            delete(item);
            return true;
        default:
            return false;
    }
}
```

Question?