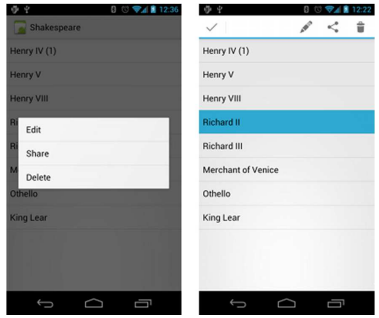
ANDROID APPS DEVELOPMENT FOR MOBILE AND TABLET DEVICE (LEVEL I)

Lecture 4: Menu and Action Bar

Peter Lo

Menu

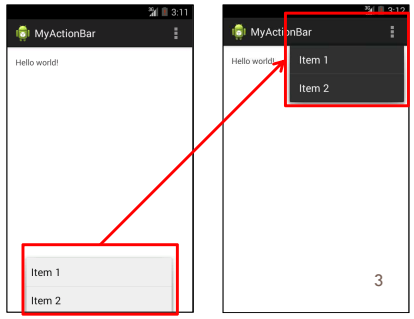
- Menus are a common user interface component in many types of applications.
- To provide a familiar and consistent user experience, you should use the Menu APIs to present user actions and other options in your activities.
  - Options Menu
  - Contextual Menus



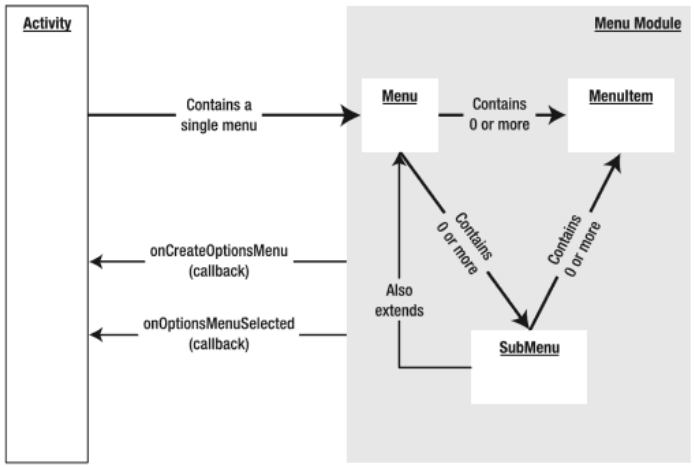
Option Menu

- Options Menu is the one that appears when you click the menu button on older Android devices, or via the action bar at the top of the screen in newer ones (> 3.0).
- The options menu should handle global application actions that make sense for the whole app.

Beginning with Android 3.0 (API level 11), Android-powered devices are no longer required to provide a dedicated Menu button. 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.

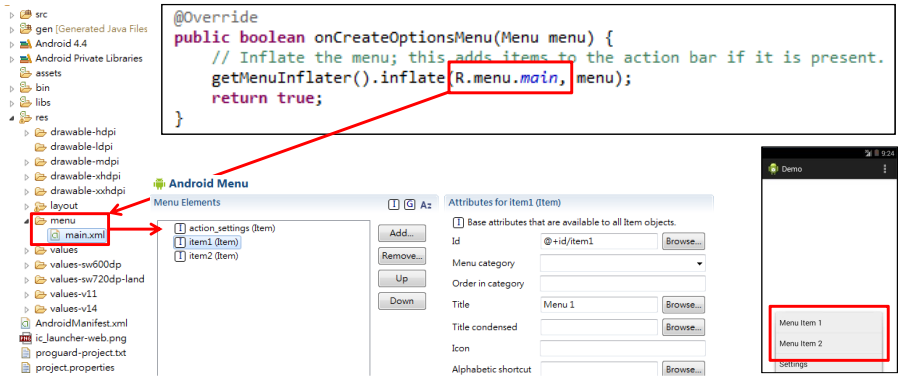


Option Menu Framework



Creating Option Menu

- To specify the options menu for an activity, override ***onCreateOptionsMenu()***. You can inflate your menu resource (defined in XML) into the Menu provided in the callback:



Handling Click Events for Option Menu

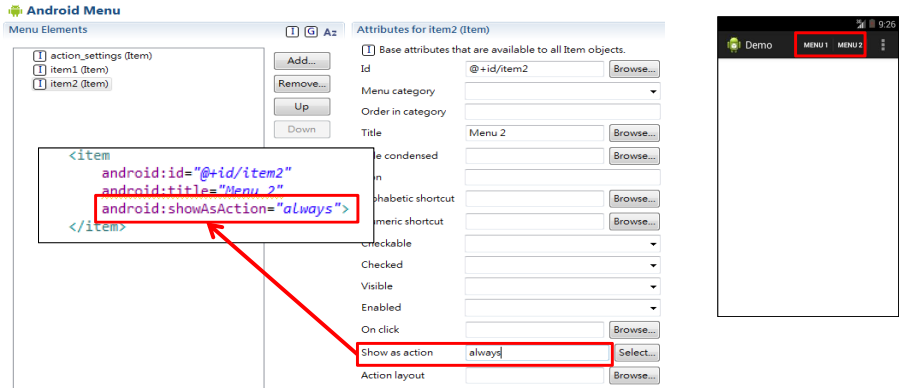
- When the user selects an item from the options menu, the system calls your activity's ***onOptionsItemSelected()*** method.
- This method passes the ***MenuItem*** selected. You can identify the item by calling ***getItemId()***, and match this ID against known menu items to perform the appropriate action

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    switch (item.getItemId()) {
        case R.id.item1:
            YourAction1();
            return true;
        case R.id.item2:
            YourAction1();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }
}
```

When you successfully handle a menu item, return true. If you don't handle the menu item, you should call the superclass implementation of ***onOptionsItemSelected()***, the default implementation returns FALSE.

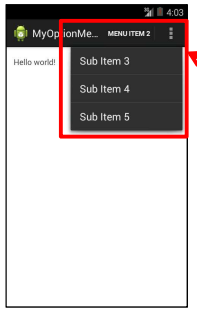
Convert Menu Item into Action Bar

- In order to convert the menu item into action bar, set ***ShowAsAction = always*** in the menu.



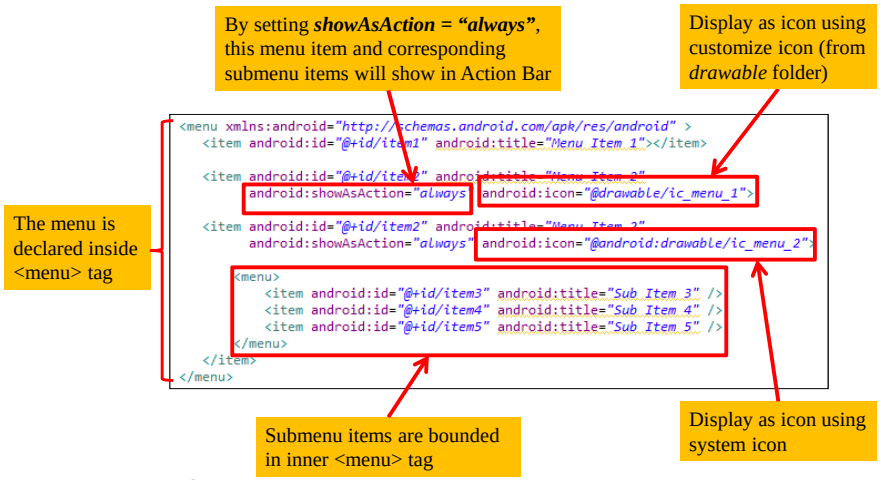
Submenu

- This is a floating list of menu items that is revealed by an item in the Options Menu (in Action Bar) or a Context Menu.
- A Submenu item cannot support nested Submenus.



Beginning with Android 3.0 (API level 11), Submenu only available in Action Bar item

Simple XML for Menu



System icon can be found from \sdk\platforms\android-18\data\res\drawable-hdpi

Defining Menu in XML

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

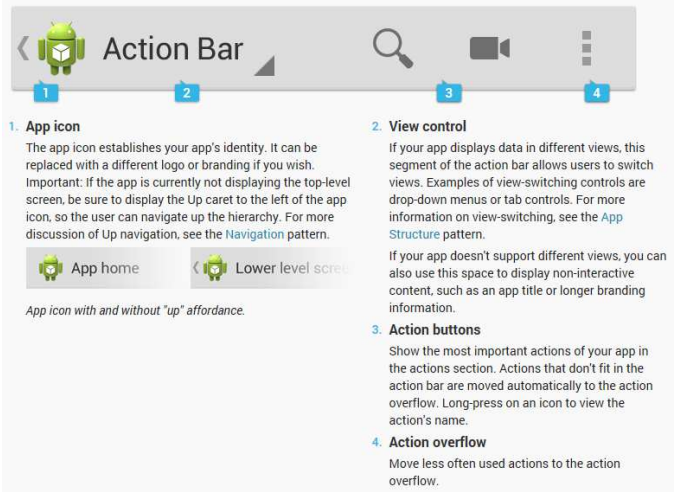
Tag	Description
<menu>	• Defines a Menu, which is a container for menu items. • A <menu> element must be the root node for the file and can hold one or more <item> and <group> elements.
<item>	• Creates a MenuItem, which represents a single item in a menu. • This element may contain a nested <menu> element 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.

Action Bar

- Located at the top of the activity.
- Can display the activity title, icon, actions which can be triggered, additional views and other interactive items.
- Provides several features that make your app immediately familiar to users by offering consistency between other Android apps:
  1. A dedicated space for giving your app an identity and indicating the user's location in the app.
  2. Access to important actions in a predictable way (e.g. Search).
  3. Support for navigation and view switching (with tabs or drop-down lists).

General Organization

- The action bar is split into four different functional areas



Layout Considerations for Split Action Bars

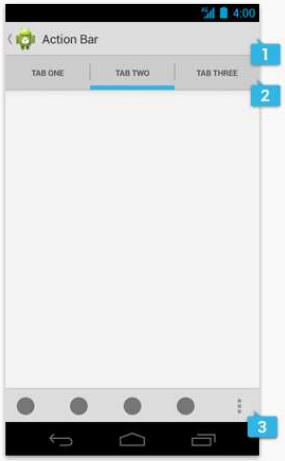
When splitting up content across multiple action bars, you generally have three possible locations for action bar content:

1. **Main action bar**
2. **Top bar**
3. **Bottom bar**

If the user can navigate up the hierarchy from a given screen, the main action bar contains the up caret, at a minimum.

To allow the user to quickly switch between the views your app provides, use tabs or a spinner in the top bar.

To display actions and, if necessary, the action overflow, use the bottom bar.



Action Buttons

Action buttons on the action bar surface your app's most important activities. Think about which buttons will get used most often, and order them accordingly. Depending on available screen real estate, the system shows your most important actions as action buttons and moves the rest to the action overflow. The action bar should show only those actions that are available to the user. If an action is unavailable in the current context, hide it. Do not show it as disabled.

A sampling of action buttons used throughout the Gmail application.

For guidance on prioritizing actions, use the FIT scheme.

F — Frequent

- Will people use this action at least 7 out of 10 times they visit the screen?
- Will they typically use it several times in a row?
- Would taking an extra step every time truly be burdensome?

I — Important

- Do you want everyone to discover this action because it's especially cool or a selling point?
- Is it something that needs to be effortless in the rare cases it's needed?

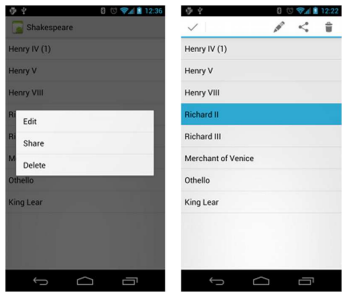
T — Typical

- Is it typically presented as a first-class action in similar apps?
- Given the context, would people be surprised if it were buried in the action overflow?

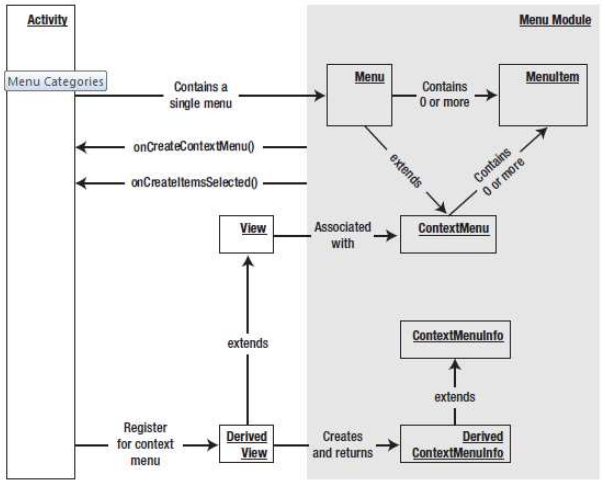
 More icon can be found from \sdk\platforms\android-18\data\res\drawable-hdpi

Contextual Menus

- Contextual Menus appear when you long-click on an element.
- Contextual menus should handle element-specific actions. They're particularly useful in GridView or ListView layouts, where you are showing the user a list of elements.



Contextual Menu Framework



Creating Context Menu

- To specify the options menu for an activity, override ***onCreateContextMenu()***. You can inflate your menu resource (defined in XML) into the Menu provided in the callback:

```
this.registerForContextMenu(item);
```

By calling ***registerForContextMenu***() and passing it a View you assign it a context menu. When this View receives a long-press, it displays a context menu.

```
public void onCreateContextMenu(ContextMenu contextMenu,
                                View view, ContextMenuInfo menuInfo) {
    // Inflate the context menu
    super.onCreateContextMenu(contextMenu, view, menuInfo);
    getMenuInflater().inflate(R.menu.contextmenu, contextMenu);
}
```

By overriding the activity's context menu create callback method, ***onCreateContextMenu()***

Handling Click Events for Context Menu

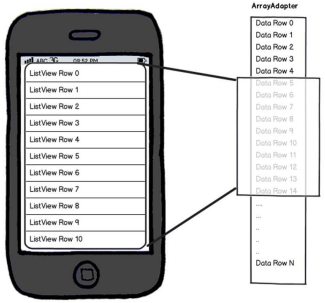
- When the user selects an item from the context menu, the system calls your activity's ***onContextItemSelected ()*** method.
- This method passes the ***MenuItem*** selected. You can identify the item by calling ***getItemId()***, and match this ID against known menu items to perform the appropriate action

```
public boolean onContextItemSelected(MenuItem item) {
    switch(item.getItemId()) {
            case R.id.item1:
                // Context Menu Item 1 Action
                return true;

            case R.id.item2:
                // Context Menu Item 2 Action
                return true;

            default:
                return super.onContextItemSelected(item);
    }
}
```

When you successfully handle a menu item, return true. If you don't handle the menu item, you should call the superclass implementation of ***onContextItemSelected()***, the default implementation returns FALSE.

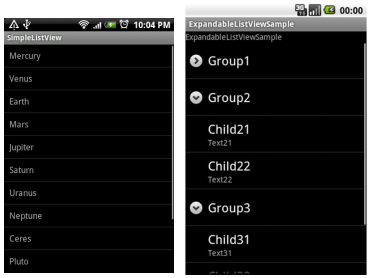
List

- The display of elements in a lists is a very common pattern in mobile applications.
- The user sees a list of items and can scroll through them.
- Typically the user interacts with the list via the action bar.
- Individual list item can be selected, this selection can update the action bar or can trigger a detailed screen for the selection.



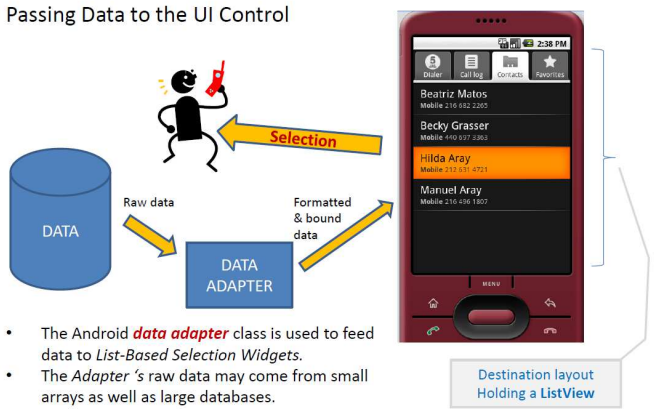
Using Lists in Android

- Android provides the ListView or ExpandableListView classes for displaying a scrollable list of items.
  - **ListView** – used everywhere from short lists of menu options to long lists of contacts or internet favorites.
  - **ExpandableListView** – supports also a grouping of items. The items in the list can be of any type.
- An adapter is used for managing the items in the list (the data model or data source).



ListView

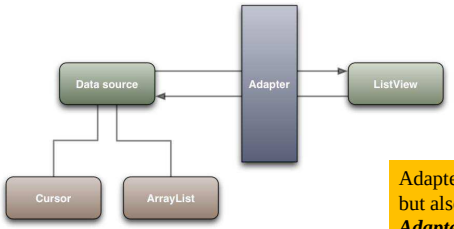
- ListView provides a simple way to present a scrolling list of rows with a built-in style or customized format extensively.
- It requires an adapter to feed it with data contained in row views.



- The Android ***data adapter*** class is used to feed data to *List-Based Selection Widgets.*
- The *Adapter 's* raw data may come from small arrays as well as large databases.

Adapters

- An adapters manages the data model and adapts it to the individual rows in the list view.
- The adapter would inflate the layout for each row in its ***getView()*** method and assign the data to the individual views in the row.
- The adapter is assigned to the ***ListView*** via the ***setAdapter*** method on the ***ListView*** object.



Adapter are not only used by ***ListView*** but also by other views which extends ***AdapterView***, as for example Spinner, GridView, Gallery and StackView.

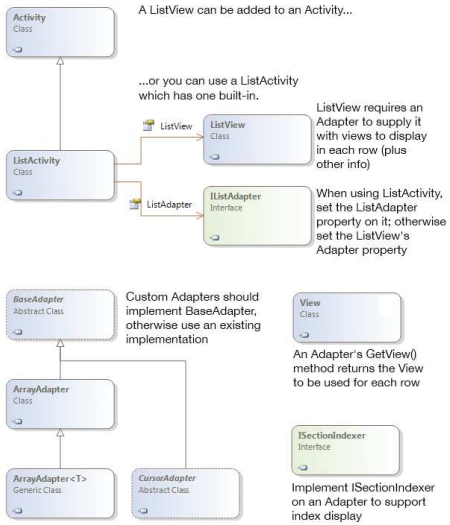
ArrayAdapter

- ***ArrayAdapter*** is a concrete BaseAdapter that is backed by an array of arbitrary objects.
- By default this class expects that the provided resource id references a single TextView.
- An ***ArrayAdapter*** can be used to wrap the contents of a Java array or ***java.util.List*** and supply data rows to the UI.

```
ArrayAdapter<String> aa = new ArrayAdapter<String>(
                    this,
                    android.R.layout.simple_list_item_1,
                    android.R.id.text1,
                    items);
```

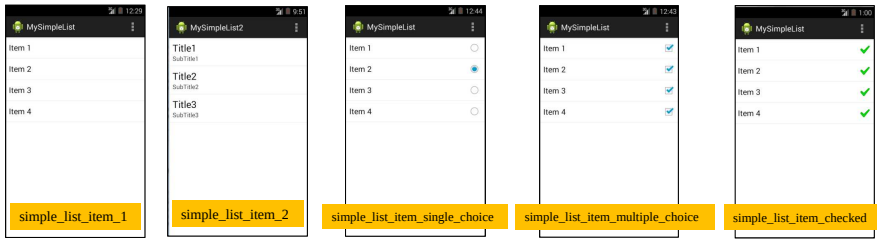

List Classes Overview

- **ListView** – User interface element that displays a scrollable collection of rows. It usually uses up the entire screen on mobile, or being part of a larger layout on tablet devices.
- **View** – ListView requires a View to be supplied for each row.
- **BaseAdapter** – Base class for Adapter implementations to bind a ListView to a data source.
- **ArrayAdapter** – Built-in Adapter class that binds an array of strings to a ListView for display.
- **CursorAdapter** – Use CursorAdapteror SimpleCursorAdapter to display data based on an SQLite query.



Common List Layout

- android.R.layout.simple_list_item_1
- android.R.layout.simple_list_item_2
- android.R.layout.simple_list_item_single_choice
- android.R.layout.simple_list_item_multiple_choice
- android.R.layout.simple_list_item_checked



Create a Simple ListView

- The list items are automatically inserted to the list using an ***Adapter*** that pulls content from a source such as an array or database query and converts each item result into a view that's placed into the list.

Get ListView object from XML layout.

Defined Array values to show in ListView

```
ListView1 = (ListView)findViewById(R.id.ListView1);
String ListItems[] = {"Item 0", "Item 1", "Item 2", "Item 3", "Item 4"};
ArrayAdapter<String> adapter = new ArrayAdapter<String> (this,
                                    android.R.layout.simple_list_item_1,
                                    ListItems);
ListView1.setAdapter(adapter);
```

Assign adapter to ListView

Define a new Adapter
- *context* - The current context.
- *resource* -The resource ID for a layout file (e.g. simple_list_item_1, simple_list_item_checked, etc)
- *objects* - The objects to represent in the ListView.

Handling Click Events for ListView

- Callback method to be invoked when an item in this ***AdapterView*** has been clicked.
- Implementers can call ***getItemAtPosition(position)*** if they need to access the data associated with the selected item.

Register a callback to be invoked when an item in the list view clicked

```
ListView1.setOnItemClickListener(new OnItemClickListener() {
    public void onItemClick(AdapterView<?> parent, View view, int position, long id) {
        String itemValue = (String) parent.getItemAtPosition(position);
        switch (position) {
            case 0:
                // Do something
            case 1:
                // Do something
            default:
                // Handle other case
        }
    }
});
```

Obtain the clicked item value

Response to user selection

Create List using ListActivity

- ListActivity makes it simple to create an activity whose main purpose is to show a list of elements. You can extend Activity and achieve the same thing but ListActivity makes it easier to achieve.

Extends ListActivity rather than Activity

```
public class MainActivity extends ListActivity
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        String ListItems[] = { "Item 1", "Item 2", "Item 3", "Item 4"};
        ArrayAdapter<String> adapter = new ArrayAdapter<String>
            (this,android.R.layout.simple_list_item_1, ListItems);
        setListAdapter(adapter);
    }
    protected void onListItemClick(ListView list, View view, int position, long id) {
        switch(position) {
            case 0:
                // Do something
            case 1:
                // Do something
            default:
                // Handle other case
        }
    }
}
```

Defined Array values

Define a new Adapter

Assign adapter to ListView

Response to user selection