CODE NOTES Fab2081Student App

All quotes in this document from developer.android.com
No linked pages are required reading in this document

MATERIAL DESIGN

Gradle Scripts/build.gradle (Module App).

Note the dependency: compile 'com.android.support:design:...'.

This design library includes many Views (UI components) such as the FAB (Floating Action Button) that implement the look/feel and behaviour of a UI according to Material Design principles. Some other MD related components such as CardView and RecyclerView require their own libraries to be added.

The appcompat-v7 library also supplies MD support with backward compatibility

FAB (FLOATING ACTION BUTTON)

Android Studio's **Basic Activity** Template includes one.

FAB should be used to perform a single, obvious. most common action.

There are many guidelines about its contents and location if you want to conform to Material Design principles. See here for details if you are interested.

One Material Design principle is that UI components should appear in different layers i.e. at different depth. FABs appear to float above a background level.

SNACKBAR

Android Studio's **Basic Activity** Template includes one.

A panel that appears at the bottom of the screen containing a message and an optional action button. The panel can be made to disappear after some time or require the user to swipe it away.

Contrast to Toast: Toasts always disappear by themselves and cannot include an action button i.e. no user response is possible.

Material Design has a lot to say about the layout and behaviour of standard UI components. The FAB, SnackBar and AppBar are 3 such UI components. Also note the behaviour of the FAB when the SnackBar appears.

LAYOUTS

- activity_main
  - Displayed by launch Activity (MainActivity)
  - **includes** another layout (content_main)
    - Nesting layouts in this way simplifies individual layouts allowing, in this case, separation of app "décor" such as CoordinatorLayout, AppBarLayout and FAB from actual app content, in this case a ListView
  - CordinatorLayout coordinates interaction between its child views including some default interactions and some customised actions as expressed by certain child view properties
    - e.g. FAB moves up to make way for a SnackBar's appearance (a default interaction)
- content__main
  - Contains a ListView View container (see below for details)

LAYOUT ELEMENT ATTRIBUTES – EXOTICA

- Attributes beginning with “tools:” address the Layout editor and will have no effect on the actual running app.
- Attribute values beginning with “?” refers to that attribute in the theme currently in effect.

CUSTOMISING FAB LOOKS

If customising, set its layout attributes appropriately (remember there are Material Design guidelines).

Use Android Studio →Tools → Android → Theme Editor to customise colours. By default, the FAB background colour (backgroundTint) is taken from the theme’s “colorAccent” unless overridden.

Use its srcCompat property to customise the button’s image.

RESPONDING TO A FAB BEING CLICKED

See MainActivity’s onCreate.

- Get a reference to the component
- Assign an on click listener object (type required is View.OnClickListener interface type)
- Code an event handler in the listener object’s class (onClick required by View.OnClickListener interface)

An anonymous listener object of the correct type has been instantiated and coded in place using standard ultra-compact syntax. This is a standard Java code pattern.

MAKING A SNACKBAR

Like Toasts Snackbars are made and displayed in code (no Layout XML). See MainActivity’s onCreate method.

Because some Snackbar methods are both invoked on a Snackbar and return the Snack bar they can be chained for compact syntax.

The first parameter of the Snackbar.make(…) method is a starting point for the SnackBar to walk up the UI’s View tree looking for a CoordinatorLayout to be its parent. This will allow the CoordinatorLayout to lift the FAB up out of the way when the SnackBar is shown for instance.

The Snackbar.make(…) returns a SnackBar object. The Snackbar’s setAction() method is invoked on this object to specify both the text on its action button and the listener object that contains the event handler for when the action button is clicked.

The Snackbar’s setAction() method returns the Snackbar object it was invoked on. This returned Snackbar object is then finally displayed using the Snackbar’s show() method.

continued …

LISTVIEW

Lists are an important part of many Android apps. Often clicking a list item will lead to some action concerning whatever is associated with the clicked list item (not the case in this app though).

ListView components can be created and added to a layout in code or XML. There is a palette item in the layout editor for instance.

The ListView component implements a View that shows items in a vertically scrolling list. The items come from the ListAdapter associated with this View.

The arrangement is as follows:

data (ArrayList of String in this case) ←→ **adapter** ←→ **display component** (ListView in this case)

Setting up a ListView (see MainActivity's onCreate):

- get a reference to the ListView
- instantiate an adapter (specifying its data source, an ArrayList of String called listItems in this case)
- plug the adapter into the ListView using the ListView's setAdapter(…) method.

A ListView's setAdapter(…) method takes a ListAdapter as its only parameter. This is an Interface type that several classes implement including ArrayAdapter<T> (for array data sources) and CursorAdapter (for database sources).

What is ArrayAdapter<T>. This is a Java Generic Class Type. When coding (i.e. pre-compilation), the T must be replaced with a specific reference type. Generics Types remove the need for super class references in many common coding situations (not polymorphism though) that are the source of run-time crashes. Generic Types allow for more thorough compile error detection rather than run time crashes.

ADDING AND REMOVING LISTVIEW ITEMS

Use the ListView's add(itemToAdd) and remove(indexToRemove) methods.

In this case the item to remove is the last added i.e. the one at index array size – 1.

Use the adapter's notifyDataSetChange() method to ensure the adapter's connected display component is refreshed to allow for the add/remove.

continued …

OPTIONS MENU

App Bar is Material Design’s name for the Action Bar. It’s a “dedicated piece of real estate at the top of each [Activity’s] screen [below the status bar] that is generally persistent throughout the app.”

The Options menu is opened by clicking the 3 vertical dots at the extreme right hand end of the App Bar. It can contain action buttons. If there are too many action buttons or there is not enough room in the App Bar these actions buttons are pushed into the Overflow (aka Options) menu as menu items.

It’s possible to insist actions stay in the Options menu or allow them to display in the App Bar if there is room (see showAsAction attribute of menu items).

Making an Options Menu

It’s possible to build an Activity’s options menu in code but for maintainability they are best specified as an XML layout resource in the res/menu project directory. Just as for other layout XML files there is a dedicated Design view that allows a menu to be built quickly by dragging/dropping and setting properties.

The basic XML vocabulary is simple but it is possible to add features such as groups of menu items that, for instance, could act like radio buttons. It’s also possible to create checkable items, items with icons and/or text and dividers between items etc.

INFLATING

- Override the Activity method onCreateOptionsMenu(menu)
- Use the passed in menu reference (an Activity can only have at most 1 Options menu so there is no ambiguity here) to inflate the Options menu according to the specified menu resource (res/menu/menu_fab_example.xml in this case)
- return true to display the menu
    - You will often see: return super.onCreateOptionsMenu(menu) which is probably more correct as the super gets to do any work it does and decide whether or not to display the menu
    - Just returning true seems to work just fine though

RESPONDING TO MENU ITEM SELECTION IN THE OPTIONS MENU

- Override the Activity method onOptionsItemSelected(MenuItem menu)
- Extract the id of the menu item passed in (which is the menu item that was clicked)
- Code a conditional control structure (if/if else/else or switch) to deal with different cases of the id (i.e. different menu item selections)
- In each case return true to indicate you have consumed the event
    - This prevents the default action which involves calling the item's associated Runnable or sending a message to its Handler (these have to be set up but it’s rare to see this default approach used)
- Note we defer to the super’s actions and its return value if we do not know what to do