Chapter 31 -- Menus

- Tool Bar -- top bar on the comapt app
  - Older versions of android had the "Action Bar"

- Overflow menu -- Right side of action bar
  - Allows control via menu rather than buttons
  - Again ... code or XML
  - MenuExample shows XML version

- XML
  - `<group android:checkableBehavior="single"...`
    - none, all, single
  - `<item .. :id .. :title />

- Code to create menu and use menu:
  - `onCreateOptionsMenu (Menu menu)`
    - `getMenuInflater(). inflate(R.menu.name, menu)`
  - `onOptionsItemSelected(MenuItem item)`
    - `getItemId()`
    - `isChecked(), setChecked()`
    - `findViewById to access a view`
    - `could call methods in the activity to do work`
Chapter 35 -- Floating action button, Snack bar ...

☐ (ignore Chapter 32-34 for now.)
☐ Toasts (not in chapter 35)
  ☐ Short message, no interaction

☐ Snackbar
  ☐ Short message, may have interaction, can be dismissed or timed out

☐ Floating Action Button -- from the "v7 support library"

☐ ListView
  ☐ A scrollable list of views
  ☐ Needs anListAdapter of some kind
    ☐ ArrayAdapter gets data from an array
    ☐ For your ListViews may need to create your own Adapter
  ☐ SQL database data is often the basis of an adapter

☐ Snackbar
  ☐ make(view,test,time_to_show)
  ☐ setAction(text, onClickListener)
  ☐ show()
List Controls -- a variety

- List controls include: ListView, GridView, Spinner, Gallery, RecyclerView, CardView
- But ... we need to do adapters first
- View hierarchy:
  - View -> ViewGroup -> AdapterView -> (List Controls)
- List controls manage a collection of views
  - not always identical
  - not always visible (includes scrolling)
- Adapter -- data management
  - go between -- view and data source
  - adapter manages data for the list control
  - may access a database (or more than one)
  - may use an array or java.util.List type structure
  - may talk across the network
    - may get data from the network and create a local array/List
- How does an adapter work?
  - View requests the element views that are visible
  - Can "reuse" views
- Let us look at a Adapter Interface to understand this
Adapter Interface

Source: .../Android/sdk/sources/android-27/android/widget/Adapter.java
URL:  https://developer.android.com/reference/android/widget/Adapter

Methods in the interface:
- int getCount() -- number of items in the data set to be displayed
  - includes headers and footers
- boolean isEmpty() -- any real data?
- long getItemId(int position) -- Not the View Id -- can be just position
  - mapping from "row position" in view to data element
- boolean hasStableIds()
  - do Ids for each row stay the same even if data changes
- int getItemViewType(int position)
- View getView(int position, View convertView, ViewGroup parent)
  - work together to allow different views in different rows
  - called when the view wants to display the row (or have it ready)
  - convertView => null, must create a new view
  - convertView => !null, may reuse or create a new one
- int getViewTypeCount()
  - How many different views will be returned by getView
Adapter Interface (page 2)

- Object getItem(int position)
  - returns the actual data object ... not used by the view often
- Other things ... read the reference/code (may talk about later)

Android provided adapters
- ArrayAdapter<T> -- easiest to use.
- CursorAdapter -- used with a "cursor"
  - Cursor: This interface provides random read-write access to the result set returned by a database query.
- SimpleAdapter
- ResourceCursorAdapter
- SimpleCursorAdapter
ArrayAdapter<T> example -- (old code)

```java
public class MainActivity extends ListActivity implements OnItemClickListener {
    @Override protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(this,
            R.array.str_list, android.R.layout.simple_list_item_1);
        setListAdapter(adapter);
        getListView().setOnItemClickListener(this);
    }

    public void onItemClick(AdapterView<?> arg0, View target, int position, long id) {
        // deal with click ...
    }
}
```

- ListActivity -- List as the full UI
- ListFragment -- List as the full UI but fragment form
- More control is you just add a ListView to your view and not use ListActivity or ListFragment
Building your own adapter

Base Adapter -- Base class for many adapters
  □ ArrayAdapter<T>, CursorAdapter, SimpleAdapter
  □ Could sub-class the ones above

Have three things
  □ View -- List, Grid, Spinner, Gallery, ...
  □ Data, in some form
  □ Adapter -- standard ones, special ones, subClasses of standard ones

One more method for Adapter
  □ What if the data changes in the data set?
  □ Call the adapter’s notifyDataSetChanged()
    □ AdapterView should already have registered with the adapter

CardDemo -- RecyclerView (Chapter 38) (Part of Android Jetpack)
  □ RecyclerView -- Somewhat replaces ListView ... makes using adapter easier
    □ URL: https://developer.android.com/guide/topics/ui/layout/recyclerview
  □ Code in support.v7
  □ Clearly what google wants you to use now. "More advanced and flexible version of ListView"
Components:
- RecyclerView -- The actual view
- LayoutManager (LinearLayoutManager, GridLayoutManager, or your own)
- Views are an extension of RecyclerView.ViewHolder
  - CardView -- a support view that gets used a lot
- Adapter is an extension of RecyclerView.Adapter
  - Uses an extension of RecyclerView.ViewHolder
    - onCreateViewHolder(ViewGroup parent, int viewType)
    - onBindViewHolder(ViewHolder holder, int position)
    - getItemCount() (standard adapter ...)
- CardView example from chapters 37 and 38.
- Another example: https://github.com/googlesamples/android-RecyclerView
Widgets in the chapter: ViewPager, PageAdapter, TabLayout, AppBarLayout

ViewPager -- android.support.v4.view.ViewPager

- Often used with Fragments
- Allows one to flip land and right through pages of data
- Lots of functionality, not used by this demo program
- Needs an adapter, PageAdapter is a typical one
  - setCurrentItem(item), setCurrentItem(item, smootheScroll)
  - addOnPageChangeListener()/removeOnPageChangeListener()
  - addView(child, index, layout_params)
  - should be set to be scrollable or not

PageAdapter -- used with the ViewPager

- Typically returns a fragment for each element, getItem() used
  - (I suspect it could just return views ... similar to a ListView use.)
Tabbed Interface (page 2)

- TabLayout -- Part of the design support library
- Typically used with the AppBar and ToolBar
- Can be used as part of the ViewPager
- Tabs can have text, views or icons.

```java
TabLayout tabLayout = ...;
    tabLayout.addTab(tabLayout.newTab().setText("Tab 1"));
    tabLayout.addTab(tabLayout.newTab().setText("Tab 2"));
    tabLayout.addTab(tabLayout.newTab().setText("Tab 3"));
    tabLayout.addTab(tabLayout.newTab().setView(view));
    tabLayout.addTab(tabLayout.newTab().setIcon(drawable));
```

- `addOnTabSelectedListener()/removeOnTabSelectedListener()`
- With enough tabs and setup, tab can be scrollable
Chapter 39 -- AppBar and Collapsing ToolBar Layouts

- AppBar: Status Bar, ToolBar, TabBar (in the example app)
  - Also can have a "flexible space area"
  - Flexible space can coordinate with a RecyclerView
  - Flexible space can have a background image that changes with movement
- android.support.design.widget.CollapsingToolbarLayout
  <android.support.design.widget.CollapsingToolbarLayout ... >
  <ImageView .... />
  <android.support.v7.widget.Toolbar ... />
</android.support.design.widget.CollapsingToolbarLayout>
Chapter 40 -- Navigation Drawer

- Panel that slides out from the left, like a big menu
- Provides for navigation and other functions
- `android.support.v4.widget.DrawerLayout`
- Main layout for the full UI
- `NavigationView` is a child of `DrawerLayout`
- Menu Resource for showing options on the `NavigationView`
- Optional View for nav drawer header region
- `ActionBarDrawerToggle` -- button on the AppBar to toggle the drawer
- drawer methods
  - `isDrawerOpen()`, `closeDrawer()`, `openDrawer()`
- Need an `onNavigationItemSelected()` method registered on `NavigationView`
Chapter 41: Master/Detail Flow

- This is an Android Studio template.
- Uses: RecyclerView and various other widgets for a more complex UI.
- Set up for use with tablet or phone, with a two-column view for landscape on the tablet.
- Portrait: ListActivity and a DetailActivity.
- Landscape: ListActivity and a DetailFragment.
- Uses the MVP - Model (data), View (UI), Presenter (code) design pattern.
- md-template - directly generated by Android Studio.
  - Notice placement of "DummyContent".
- Two activities: List for both orientations, detailActivity.
  - For landscape, there is a detailFragment.
- Assignment 2 somewhat follows the steps in Chapter 41.
- Example program from the book:
  - Adding a Manifest Permission and activity.
    - <uses-permission android:name="android.permission.INTERNET" />
  - 2nd activity in manifest.
  - Starting the 2nd activity.
Intents (Chapters 42 - 45)

- android.content.Intent -- a way to launch another activity
  - Also used to talk to services and content providers

- Explicit and Implicit intents
  - Explicit -- often used to start another activity in the current app
    ```java
    Context context = view.getContext();
    Intent intent = new Intent(context, NewActivity.class);
    intent.putExtra(key, value);
    context.startActivity(intent);
    ```
  - use "this" instead of the context up there if an Activity, fragment needs to get the context
  - putExtra adds to a bundle in the Intent
  - specifically talks about the NewActivity class in this app
  - In new Activity, access to bundle:
    ```java
    Bundle extras = getIntent().getExtras();
    if (extras != null) {
        ... extras.getString(key)...
        ... extras.getInt(key)...
    }
    ```
Returning data from an activity

`startActivityForResult(intent, REQ_CODE)`

- `REQ_CODE` -- identifies result later

`onActivityResult(requestCode: Int, resultCode: Int, data: Intent)`

- called to return result
- `requestCode` is `REQ_CODE`
- `resultCode` is `RESULT_OK` or `RESULT_CANCELLED`

Called activity can prepare return data as:

```java
finish()

Intent data = newintent;
data.putExtra(key,value);
setResult (RESULT_OK, data);
super.finish()
```

- `finish()` is called by your code when your Activity is done
- It will cause the activity to be killed
- That is why `super.finish()` is last.
Implicit Intents

Intent i = new Intent(Intent.ACTION_XXX)


System wide search for an activity to do that

Example:  ACTION_VIEW is normally associated with a URL

Intent i = new Intent(Intent.ACTION_VIEW, Uri.parse("http://...."));

If only one app responds to the intent, the app will be run

If more than one app responds to the intent, it will ask the user

Intent availability -- app crash if activity does not exist

Intent intent = new Intent(action);

List<ResolveInfo> list = getContext().getPackageManager().queryIntentActivities(intent, PackageManager.MATCH_DEFAULT_ONLY);

If list.size() greater than 0, there exists such Activities

Book has a good example function (code is different, getContext() is a View method)
Intent Filter

- "Advertisement" that an activity services an intent
- This is an <intent-filter> under the <activity> in the AndroidManifest.xml
- When starting up an activity that needs permissions
  - caller needs permissions
  - new activity needs permissions
- permissions are in the manifest
  - <uses-permission android:name="android.permission.INTERNET"/>
  - android.permissions.READ_CONTACTS
  - .CAMERA, .WRITE_EXTERNAL_STORAGE, SEND_SMS, ...

Intent filter: WebViewActivity

```
<intent-filter>
  <action android:name="android.intent.action.VIEW" />
  <category android:name="android.intent.category.DEFAULT" />
  <data android:scheme="http" />
</intent-filter>
```

- category: _LAUNCHER, _INFO, ...
- Examples from book: ExplicitIntent, ImplicitIntents
Broadcast Intents

- Sends and receives messages to the whole system

- Example send:
  ```java
  Intent msg = new Intent();
  msg.setAction("com.example.Broadcast");
  msg.putExtra(Key, data);
  sentBroadcast(msg);
  ```

- Action: Many standard strings ACTION_MEDIA_BAD_REMOVAL, ACTION_POWER_CONNECTED, ACTION_AIRPLANE_MODE_CHANGED ...

- putExtra() -- puts data in the bundle

- Sent to "running receivers" ... after 3.0, can’t start "stopped applications"
  ```java
  msg.addFlags(Intent.FLAG_INCLUDE_STOPPED_PACKAGES);
  ```

- allows applications to be started
Broadcast Receivers

- Receives a message on a matching broadcast

- Class BroadcastReceiver needs to be extended for your app
  - `onReceive(Context, Intent)` -- the receive method
  - No UI is associated with the Receiver
  - Receiver may start an activity

- Class needs to be registered
  - IntentFilter helps select broadcast
    - `IntentFilter filter = new IntentFilter("com.example.Broadcast")`
  - via code:
    - `MyReceiver receiver = new MyReceiver();`
    - `registerReceiver(receiver, filter)`
  - via XML -- in the manifest.
    - `<receiver android:name="MyReceiver"/>` (In the `<application ...>` tag)
    - `<intent-filter>` `<action android:name="com.example.Broadcast"/>` `</action>` `</intent-filter>`

- some broadcasts can not be received except if registered in code (see developer.android.com)

- Book’s SendBroadcast Example