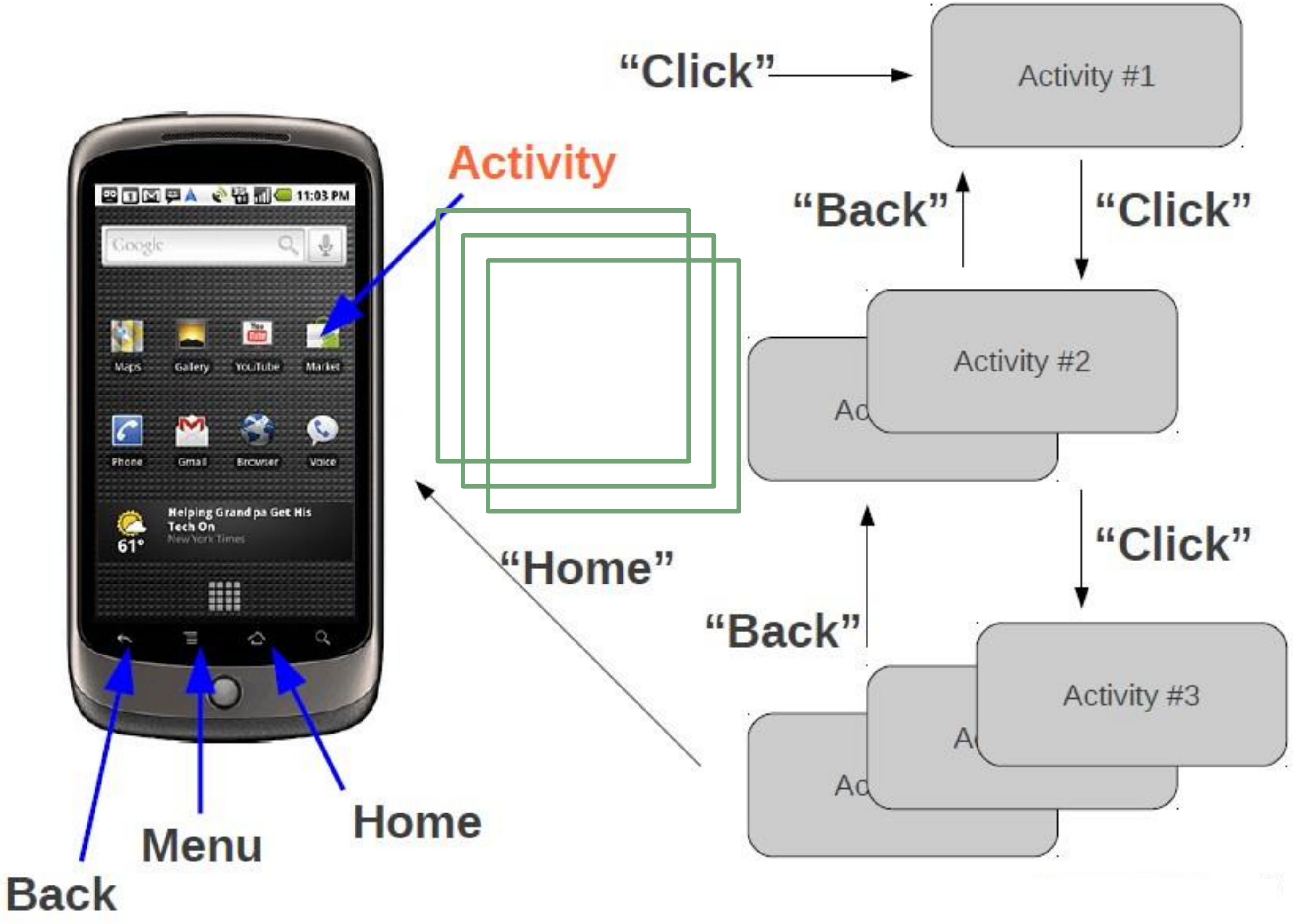
Android: Lifecycle and Menus

LEIM – DAM

App: User Interface



Activity: Lifecycle (I)

- How Android launches an App?
  - Callback method (OnCreate) of an instance of the Activity class
  - Sequence of callbacks that initiate an Activity and another sequence of callbacks that terminate Activity
- “Callbacks” define the lifecycle of an Activity

Activity: Lifecycle (II)

- Activity started for the first time
  - In "foreground" you receive the "focus" of the user
  - Android executes lifecycle methods
- Another Activity Started
  - Initial activity changes to "background"
  - Another set of life cycle methods is run
- Defining the lifecycle methods you can define the behavior of the Activity

Activity: States

- **Resumed**
  - In "foreground" and has the "focus" of the user (also referred to as "Running")
- **Paused**
  - Another activity is in "foreground" and has the "focus" but the "Paused" activity is visible (at least partially)
- **Stopped**
  - Activity is in “background” and is not visible
- **Created and Started**
  - Transient states, the system quickly switches to "Resumed"

Activity: “Paused” versus “Stopped”

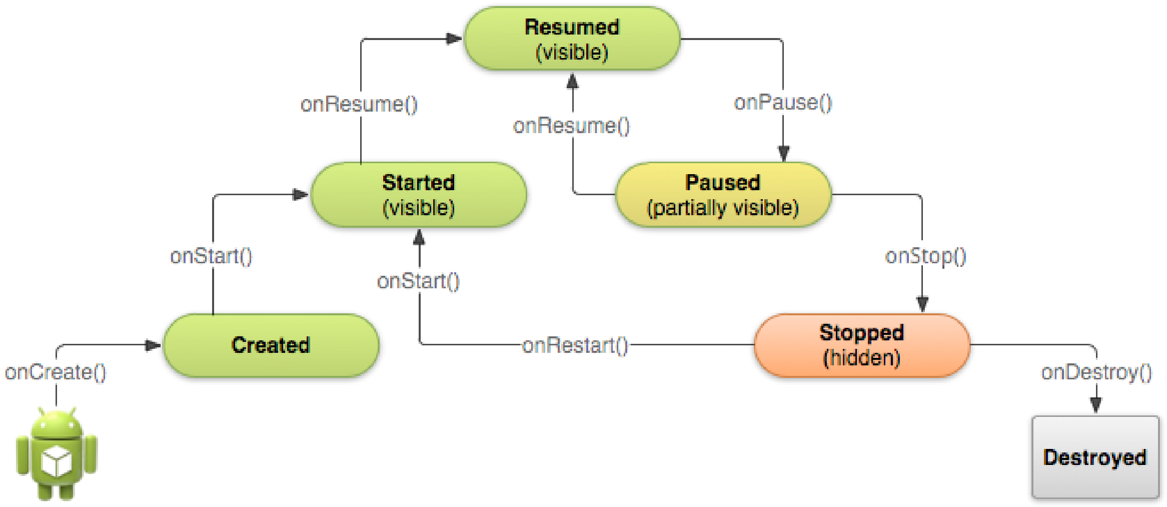
- Paused
  - Totally alive
  - Retained in memory
  - Keeps all your status and member information
  - Connected to window manager
  - Eliminated in an extremely low memory situation
- Stopped
  - Totally alive
  - Retained in memory
  - Keeps all your status and member information
  - Not connected to window manager
  - Deleted when memory is required

Activity: Callbacks (Lifecycle)

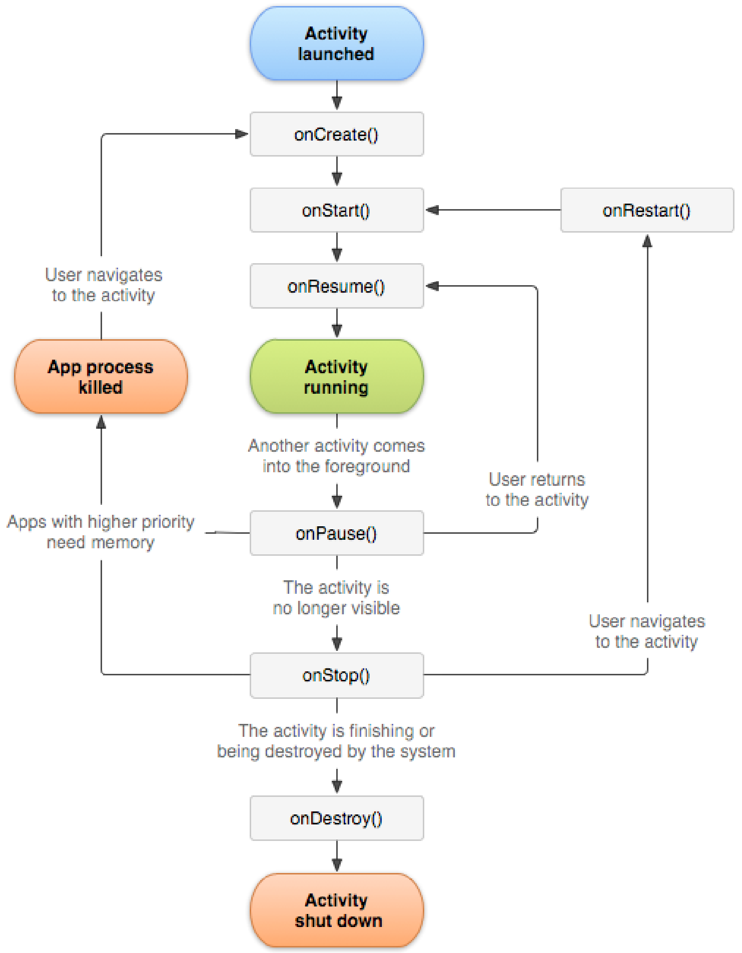
```
public class ActivityA extends Activity {

   // The activity is being created
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      …
   }

   // The activity is about to become visible
   protected void onStart() { super.onStart(); … }

   // The activity is about to be started again
   protected void onRestart() { super.onRestart(); … }

   // The activity has become visible (“resumed”)
   protected void onResume() { super.onResume(); … }

   // Another activity is taking focus (“paused”)
   protected void onPause() { super.onPause(); … }

   // The activity is no longer visible (“stopped”)
   protected void onStop() { super.onStop(); … }

   // The activity is about to be destroyed
   protected void onDestroy() { super.onDestroy(); … }
   . . .
}
```

Activity: States and Pyramid Lifecycle

- Pyramid lifecycle
  - Set of callbacks to get to the top ("Resumed")
  - Set of callbacks to destroy ("Destroyed")



Activity: Lifecycle States Diagram


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Activity: When to Save Crucial Data?

- Activity Destroyed (Killed)
  - onPause()
  - onStop()
  - onDestroyed()

- onPause()
  - Last method to be executed (if the system needs memory, OnStop() and OnDestroy () will not run)
  - Crucial data should be saved in this callback
  - Data written in this callback must be properly selected in order to not to slow down the user experience

Activity: “Paused” and “Resumed”

- Paused
  - Stop animations/videos or other actions that are consuming resources
  - Save permanent data
  - Free up system resources (Broadcast receivers, GPS or any other feature that affects the battery)
- Resumed
  - Doing the opposite of what was done in Paused
  - Do not forget that this state also happens when the app is Created

Activity: onDestroy

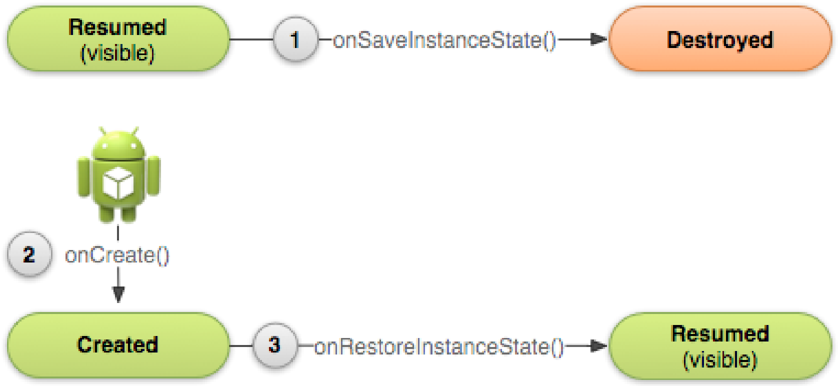
- When this callback occurs?
  - The **finish**() method was executed
  - System need space (must use **isFinishing**() to distinguish)
  - User pressed the “Back” button

- Must be implemented to free resources

- In some situations the system eliminates "activity" without calling onDestroy
  - Should not be used to store data

Activity: Save Activity State (I)

- (1) To save additional data should be redefined **onSaveInstanceState**() method
- System calls this method and passes it the Bundle object
- Creating (again) the activity the system passes the Bundle object (with the information saved) on the (2) onCreate () and the (3) **OnRestoreInstanceState** ()



Activity: Save Activity State (II)

- Save app current execution state in **onSaveInstanceState**
  - Save state to Bundle in pairs (key, value), using:
    - putInt, …, putString, putIntArray, …, putIntegerArrayList, … putStringArrayList, …
  - The Views that have input (and **id**) are saved as part of the View hierarchy state in super.onSaveInstanceState

```
static final String STATE_SCORE = "playerScore";
static final String STATE_LEVEL = "playerLevel";
...

@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
   // Call super class method to save view hierarchy state
   super.onSaveInstanceState(savedInstanceState);

   // saving the current app state
   savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
   savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);
}
```

Activity: Restoring Activity State (II)

- Restore state in **`onRestoreInstanceState`**
  - Get state from Bundle, in pairs (key, value), using:
    - getInt, …, getString, getIntArray, …, getIntegerArrayList, … getStringArrayList, …
  - Restore state in onCreate, it is also possible, but it is more restrictive

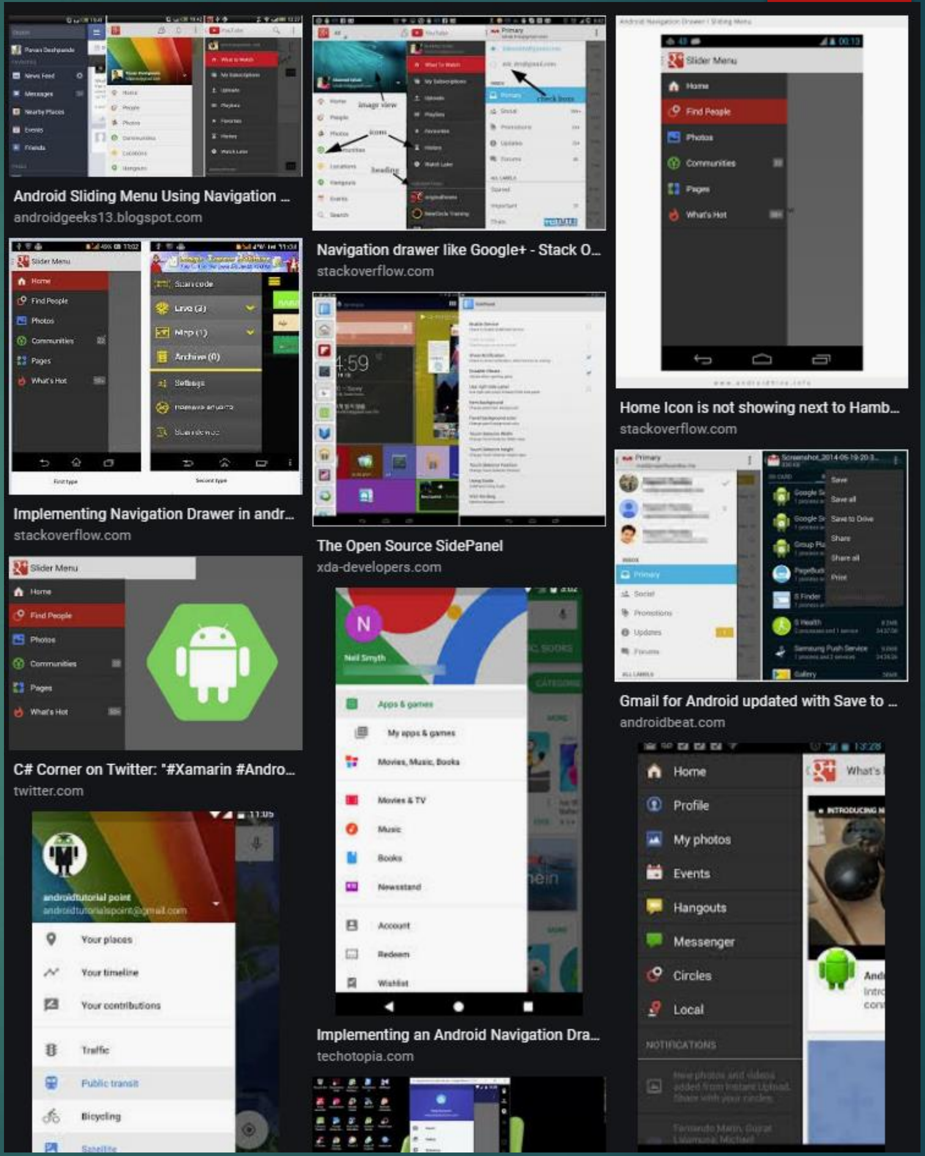
```java
@Override
public void onRestoreInstanceState(Bundle savedInstanceState) {
    // call superclass method to restore view hierarchy state
    super.onRestoreInstanceState(savedInstanceState);

    // Restore app state
    mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
    mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
}
```

App storage

- **App-specific storage:** Store files that are meant for your app's use only, either in dedicated directories within an internal storage volume or different dedicated directories within external storage. Use the directories within internal storage to save sensitive information that other apps shouldn't access.
- **Shared storage:** Store files that your app intends to share with other apps, including media, documents, and other files.
- **Preferences:** Store private, primitive data in key-value pairs.
- **Databases:** Store structured data in a private database using the Room persistence library.

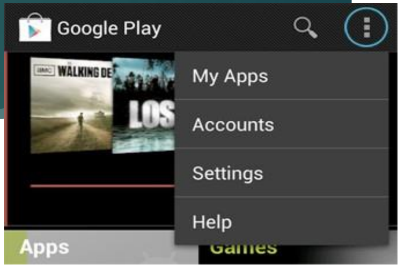
https://developer.android.com/training/data-storage

Menus

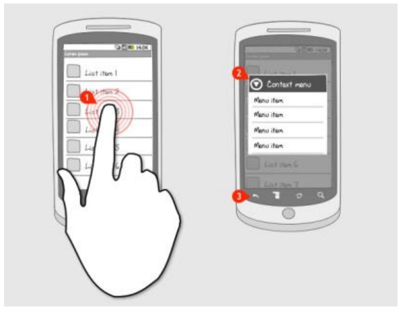

Android Sliding Menu Using Navigation ...
androidgeeks13.blogspot.com
Navigation drawer like Google+ - Stack O...
stackoverflow.com
Home Icon is not showing next to Hamb...
stackoverflow.com
Implementing Navigation Drawer in andr...
stackoverflow.com
The Open Source SidePanel
xda-developers.com
Gmail for Android updated with Save to ...
androidbeat.com
C# Corner on Twitter: "#Xamarin #Andro...
twitter.com
Implementing an Android Navigation Dra...
techotopia.com
Slider Menu
Home
Find People
Photos
Communities
Pages
What's Hot
Profile
My photos
Events
Hangouts
Messenger
Circles
Local

Menus

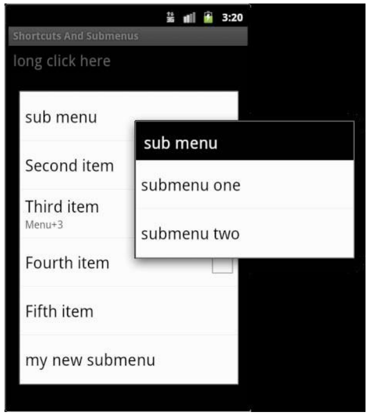
- Options Menu (more common)
  - Press the menu key on the device



- Context Menu (associated with an element)
  - Press and hold on an element



- Submenu
  - Used to create groups of options within a menu



Menu: How to Create a Menu using XML?

- Defining the XML file
  - Manually: Inside the folder “res/menu” create/edit “menu.xml” file
    - The filename identifies the menu
  - With wizard: Over the res directory selects New -> Android Resource File -> on Resource Type select ``Menu'', on File name write something like ``activityName_menu''
    - If the menu is shared between activities, choose an appropriated name
    - Add Menu items, or other elements from the Design Editor
- Upload the file programmatically
  - onCreateOptionsMenu(…)
- Advantages
  - It separates the menu from the application code
  - It is easy to visualize the Menu

Menu: XML

- Attributes of menu item: id, name, icon
  - Game_menu.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<menu
   xmlns:android="http://schemas.android.com/apk/res/android"

   <item
      android:id="@+id/menu_new_game"
      android:title="@string/new_game"
      android:icon="@drawable/ic_new_game" />
   <item
      android:id="@+id/menu_help"
      android:title="@string/help"
      android:icon="@drawable/ic_help" />
</menu>
```

Menu: Elements in the Menu.xml file

- <**menu**>
  - Defines a Menu that is a container for items of the menu
  - The "root" node of the file and may contain one or more <item> and <group> elements
- <**item**>
  - Creates a an item (option) in the menu
  - It can contains a <menu> element to create a sub-menu
- <**group**>
  - Invisible container and optional of <item> elements
  - Allows you to categorize menu items

Options Menu (I)

- Where should be included the actions of the Activity and the elements necessary for navigation
- The first visible part is the menu icon and then the first 6 options
- More than 6 options means that the 6th element is replaced by the "More" option which when pressed shows a menu with the remaining options

Setting the menu in the activity

- Set the menu in the activity by inflating the defined menu items to the menu passed on **onCreateOptionsMenu** method

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

React to User menu option selected

- When the user selects an option the system calls the activity method **onOptionsItemSelected** method

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   // Handle item selection
   switch(item.getItemId()) {

    case R.id.menu_new_game:
       newGame();
       return true;

   case R.id.menu_help:
       showHelp();
       return true;

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

Not identified options

Context Menu

- Similar to the menu displayed on the PC when the user clicks the right mouse button
- Should be used to allow access to specific **actions** of an interface element
- In Android system is displayed when the user presses and holds (**press and hold**) an interface element

How to Create a Context Menu?

- Registering a View for a Context Menu
  - View.registerForContextMenu(View view)
- Defining the appearance of the menu
  - OnCreateContextMenu()
- Defining Menu behavior
  - OnContextItemSelected()

Context Menu: Example

```
@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.menu_edit:
       editNote(info.id);
       return true;

   case R.id.menu_delete:
       deleteNote(info.id);
       return true;

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

Get the information of the selected item

Submenu

- **Menu** that the user can open by selecting an option from another menu
- You can add to each menu except a **submenu**
- Useful when the application has many functions that can be **organized into topics**

Submenu: XML

- A submenu is created by adding a <menu> element as the child of an <item>

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

Submenu: User Option Selected

- When a submenu option is selected, the **OnXXXItemSelected** callback associated with the <item> "parent" menu is executed

- For example, if the submenu is applied to a Options Menu the (**onOptionsItemSelected**) method is called

Bibliography

- Activity Lifecycle
- Menu