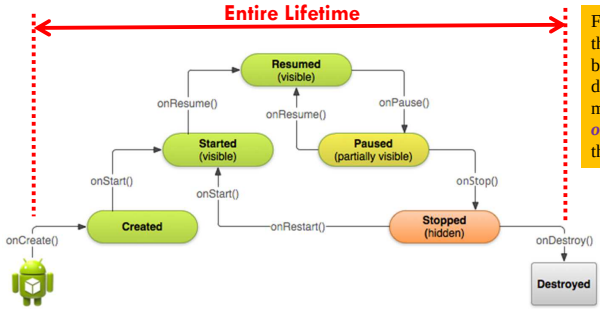
# ANDROID APPS DEVELOPMENT FOR MOBILE GAME

Lecture 3: Android Life Cycle and Intent

Peter Lo

## Entire Lifetime

- An activity begins its lifecycle when entering the ***onCreate()*** state
- If not interrupted or dismissed, the activity performs its job and finally terminates and releases its acquired resources when reaching the ***onDestroy()*** event.

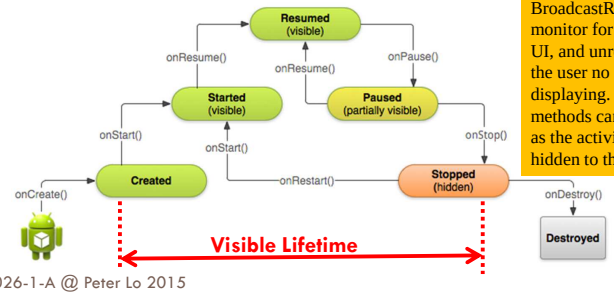



For example, if it has a thread running in the background to download data from the network, it may create that thread in ***onCreate()*** and then stop the thread in ***onDestroy()***.



## Visible Lifetime

- It happens between a call to ***onStart()*** until a corresponding call to ***onStop()***.
- During this time the user can see the activity on-screen, though it may not be in the foreground and interacting with the user.
- You can maintain resources that are needed to show the activity to the user between these two methods.

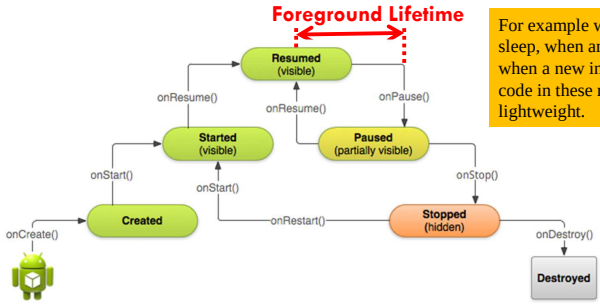



For example, you can register a BroadcastReceiver in ***onStart()*** to monitor for changes that impact your UI, and unregister it in ***onStop()*** when the user no longer sees what you are displaying. The ***onStart()*** and ***onStop()*** methods can be called multiple times, as the activity becomes visible and hidden to the user.



## Foreground Lifetime

- It happens between a call to ***onResume()*** until a corresponding call to ***onPause()***.
- During this time the activity is in front of all other activities and interacting with the user. An activity can frequently go between the resumed and paused states




For example when the device goes to sleep, when an activity result is delivered, when a new intent is delivered – so the code in these methods should be fairly lightweight.

# Life Cycle Callbacks

```
public class ExampleActivity extends Activity {
    @Override
    public void onCreate (Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        // The activity is being created.
    }
    @Override
    protected void onStart() {
        super.onStart();
        // The activity is about to become visible.
    }
    @Override
    protected void onResume() {
        super.onResume();
        // The activity has become visible (it is now "resumed").
    }
    @Override
    protected void onPause() {
        super.onPause();
        // Another activity is taking focus (this activity is about to be "paused").
    }
    @Override
    protected void onStop() {
        super.onStop();
        // The activity is no longer visible (it is now "stopped")
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        // The activity is about to be destroyed.
    }
}
```

All activities must implement ***onCreate()*** to do the initial setup when the object is first instantiated

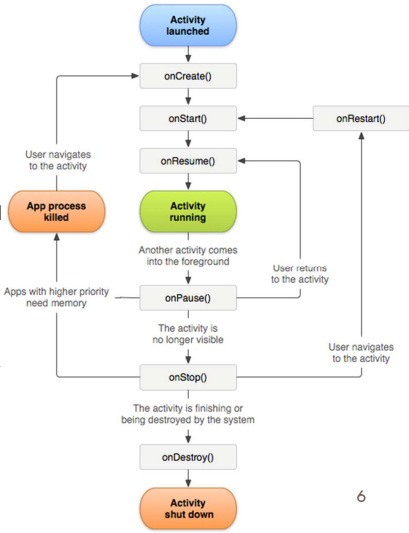
Activities should implement ***onPause()*** to commit data changes in anticipation to stop interacting with the user

Applications do not need to implement each of the transition methods, however there are mandatory and recommended states to consider



# Activity Lifecycle

- If an activity in the foreground of the screen, it is active or running.
- If an activity has lost focus but is still visible, it is paused. A paused activity is completely alive, but can be killed by the system in extreme low memory situations.
- If an activity is completely obscured by another activity, it is stopped. It still retains all state and member information, but no longer visible to the user so its window is hidden and it will often be killed by the system when memory is needed elsewhere.
- If an activity is paused or stopped, the system can drop the activity from memory by either asking it to finish, or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.

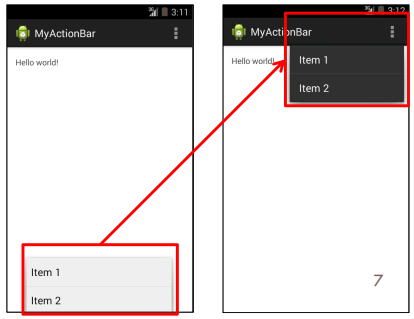





# Option Menu

- 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.

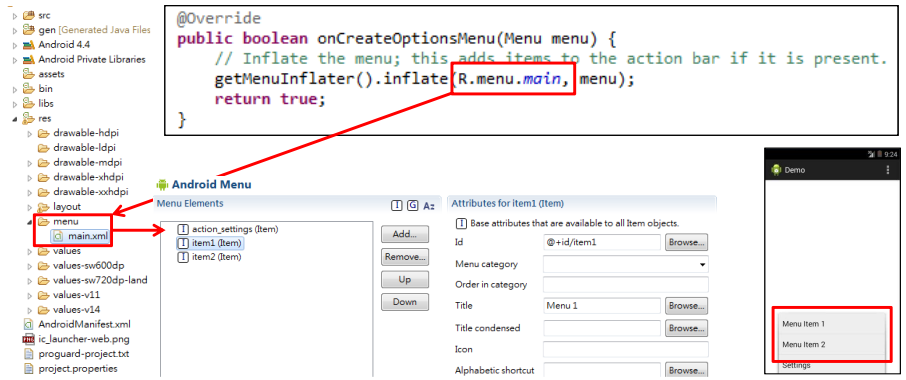
Beginning with Android 3.0 (API level 11), Android-powered devices are no longer required to provide a dedicated Menu button. With this change, Android apps should migrate away from a dependence on the traditional 6-item menu panel and instead provide an action bar to present common user actions.






# 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.



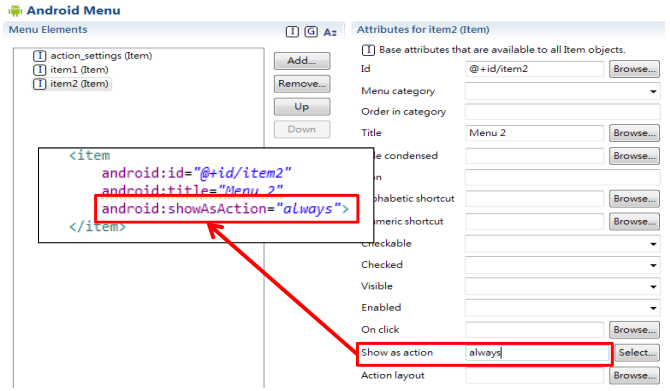
# Action Bar

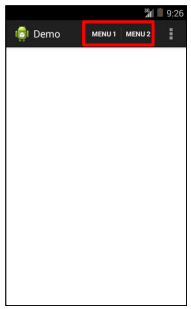
- Located at the top of the activity.
- Can display the activity title, icon, actions which can be triggered, additional views and other interactive items.
- Provides several features that make your app immediately familiar to users by offering consistency between other Android apps:
  1. A dedicated 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).




# Convert Menu Item into Action Bar

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






# Simple XML for Menu

By setting ***showAsAction = "always"***, this menu item and corresponding submenu items will show in Action Bar

Display as icon using customize icon (from drawable folder)

The menu is declared inside <menu> tag

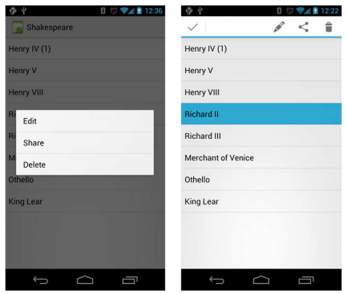
```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item android:id="@+id/item1" android:title="Menu Item 1"></item>

    <item android:id="@+id/item2" android:title="Menu Item 2"
          android:showAsAction="always" android:icon="@drawable/ic_menu_1">

    <item android:id="@+id/item2" android:title="Menu Item 2"
          android:showAsAction="always" android:icon="@android:drawable/ic_menu_2">

        <menu>
            <item android:id="@+id/item3" android:title="Sub Item 3" />
            <item android:id="@+id/item4" android:title="Sub Item 4" />
            <item android:id="@+id/item5" android:title="Sub Item 5" />
        </menu>
    </item>
</menu>
```

Display as icon using system icon

Submenu items are bounded in <menu> tag

## 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.






## 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

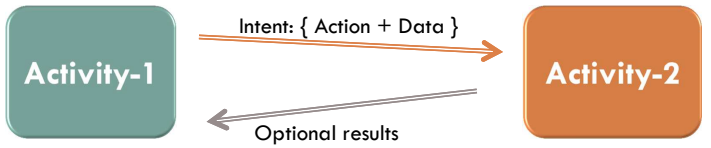
```
public boolean onContextItemSelected(MenuItem item) {
    switch(item.getItemId()) {
            case R.id.item1:
                // Context Menu Item 1 Action
                return true;

            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.



## Intents

- Intents are asynchronous messages which allow application components to request functionality from other Android components.
- Intents allow you to interact with components from the own and other applications.
  - For example an activity can start an external activity for taking a picture.



## Creating Simple Intent

- Typically an intent is called with Action/Data pair:

The built-in action to be performed, such as **ACTION_VIEW**, **ACTION_EDIT**, **ACTION_CALL**, or user-created-activity

```
Intent myIntent = new Intent( Intent.ACTION_VIEW,
                Uri.parse("www.polyu.edu.hk"));

startActivity (myIntent);
```

The *startActivity()* method is used to start a new activity, which will be placed at the top of the activity stack.

The primary data to operate on, such as a phone number to be called (expressed as a **Uri** such as **tel://** , **http://** , **sendto://**)



## Common Intent Action/Data Pairs

**Examples** of **action/data** pairs are:

| | |
|---|---|
| **ACTION_DIAL** | ***tel:27665111***<br>Display the phone dialer with the given number filled in. |
| **ACTION_SENDTO** | ***smsto:27665111***<br>Display the SMS editor with the given number filled in. |
| **ACTION_VIEW** | ***http://www.polyu.edu.hk***<br>Show specified web page in a browser view. |
| **ACTION_VIEW** | ***content://contacts/people/***<br>Display a list of people, which the user can browse through. Selecting a particular person to view would result in a new intent |
| **ACTION_EDIT** | ***content://contacts/people/2***<br>Edit information about the contact person whose identifier is "2". |

## Secondary Attributes

- In addition to the primary action/data attributes, there are secondary attributes that you can also include with an intent, such as: Category, Components, Type, and Extras.

**Type**
Set an explicit **MIME** data type
contacts/people
images/pictures
images/video
audio/mp3

*MIME - Multipurpose Internet Mail Extensions*

**Extras**
This is a Bundle of any additional information. Typical methods include:
`bundle.putInt(key, value)`
`bundle.getInt(key)`

**Category**
additional information about the action to execute


**Component**
Explicit name of a component class to use for the intent (eg. "MyMethod1")



## Creating Intent with Secondary Attribute

- We can pass the secondary attribute with the *putExtra()* method:

The built-in action to be performed, such as **ACTION_VIEW**, **ACTION_EDIT**, **ACTION_CALL**, or user-created-activity

```
Intent myIntent = new Intent( Intent.ACTION_WEB_SEARCH );

myIntent.putExtra( SearchManager.QUERY, "PolyU" );

startActivity(myIntent);
```

The ***startActivity()*** method is used to start a new activity, which will be placed at the top of the activity stack.

The secondary data to operate on, such as passing a string as an ***Extra*** argument for a Google Search. The string is a 'human' query with keywords.

## Starting Activities and Getting Results

- In order to get results back from the called activity we use the method
  - *startActivityForResult ( Intent, requestCodeID )*
- Where requestCodeID is an arbitrary value you choose to identify the call (similar to a 'nickname' ).
- The result sent by the sub-activity could be picked up through the listener-like asynchronous method
  - *onActivityResult ( requestCodeID, resultCode, Intent )*



## Starting Activities and Getting Results

- Before an invoked activity exits, it can call *setResult( )* to return a termination signal back to its parent.
- All of this information can be capture back on the parent's *onActivityResult()*.