Mobila applikationer och trådlösa nät, HI1033, HT2013

Today:
- User Interface basics
- View components
- Event driven applications and callbacks
- Menu and Context Menu
- ListView and Adapters
- Android Application life cycle,
- Activity life cycle

Expect this when developing software for limited devices

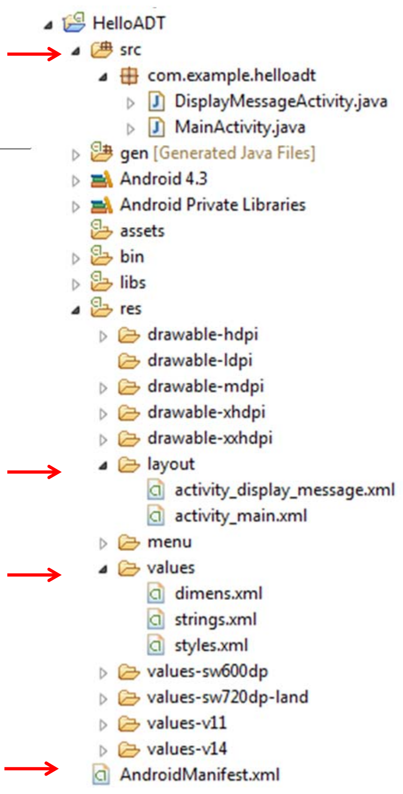
- Limited memory capacity and processor speed
  Limited battery capacity
- Network: High latency, low speeds, possibly interrupted
  Communication (might) be associated with a cost!
- Small screens, of different sizes and different densities
- Application might get interrupted at any time!
- Hardware-imposed design considerations
  Fragmentation of platforms
- *Design with this in mind:*
  Be efficient and be responsive

Android SDK

- API, Compiler, Debugger, …
- Android Virtual Device and Emulator
- Deployment tool
- Documentation, sample code, … at http://developer.android.com/
- The Eclipse IDE plugin adds:
  - Project management, project wizard
  - Editors for layouts and other resources
  - Automated building of projects
  - AVD manager
  - Debugger interface (Dalvik Debug Monitor Service, DDMS)

Anatomy of an Android Application

- Classes
- Resources, e.g. strings, layouts, data files, …
- Application Manifest, defining the entry point (e.g. an Activity), and other application settings
- On installation, packaged into an installation bundle, an Android Package (APK) file.
  - META-INF
  - res
  - AndroidManifest.xml
  - classes.dex



```
HelloADT
  src
    com.example.helloadt
      DisplayMessageActivity.java
      MainActivity.java
  gen [Generated Java Files]
  Android 4.3
  Android Private Libraries
  assets
  bin
  libs
  res
    drawable-hdpi
    drawable-ldpi
    drawable-mdpi
    drawable-xhdpi
    drawable-xxhdpi
    layout
      activity_display_message.xml
      activity_main.xml
    menu
    values
      dimens.xml
      strings.xml
      styles.xml
    values-sw600dp
    values-sw720dp-land
    values-v11
    values-v14
  AndroidManifest.xml
```

The Android Manifest file

- Contains information on application components, entry point, permissions, …

-
```
<?xml version="1.0" encoding="utf-8"?>
<manifest . . . >
    <application . . . >
        <activity
          android:name="com.example.project.GameActivity"
          android:icon="@drawable/small_pic.png"
          android:label="@string/myLabel"
            . . .  >
        </activity>
        . . .
    </application>
</manifest>
```

The Android Manifest file

- Multiple application components

-
```
<activity
    android:name="com.example.project.GameActivity” >
    <intent-filter>
          <action
                android:name="android.intent.action.MAIN" />
         <category
                android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>

<activity
    android:name="com.example.project.OtherActivity”
. . .  >
</activity>
```

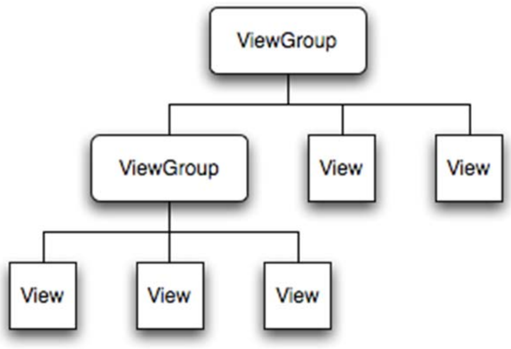
The Android Manifest file

- By default, an application has not permission to perform operations considered potentially harmful to other applications, the operating system, or the user.
- Reading or writing another application's files, performing network access, keeping the device awake, etc requires permissions

-
  ```
  <manifest
  xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.android.app.myapp" >
      <uses-permission
           android:name="android.permission.RECEIVE_SMS”
           android:name="android.permission.INTERNET” />
      ...
  </manifest>
  ```

User Interfaces, View components

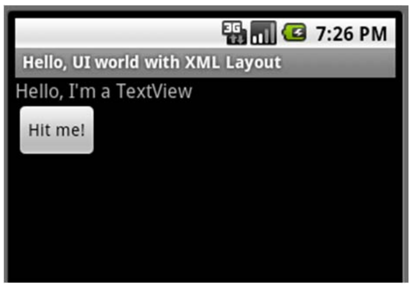
- Super class View
  - ViewGroup extends View
- ViewGroup sub classes:
  LinearLayout, RelativeLayout, ListView, GridView, RadioGroup, …
- View sub classes (Widgets):
  TextView, EditText, Button, RadioButton, … , DatePicker, Clock, + custom widgets



User Interface, View Components

- The root node should(?) be of type ViewGroup
- The Activity calls **`setContentView()`** to attach the root node to the screen
- Occupies a rectangular area of the screen
- Each view component has a set of properties representing layout parameters, text configuration, preferred size, …
- *As an object in the user interface, a View is also a point of interaction for the user and the receiver of the interaction events*

UI example



`A LinearLayout containing a TextView and a Button`

- Separate presentation and actions
- Layout and widgets and together with their properties in an XML file, e.g. res/activity_main.xml
- The file must contain exactly one root component, that may contain child components
- Multiple layout files allowed
  At least one layout file per Activity

Layout file, res/main.xml

```
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
     android:orientation="vertical"
     android:layout_width="fill_parent"
     android:layout_height="fill_parent”  >

    <TextView android:id="@+id/text"
       android:layout_height="wrap_content"
       android:layout_width="fill_parent"
       android:text="Hello, I'm a TextView"
       android:textSize="16sp">
    </TextView>

    <Button android:id="@+id/button"
         . . . >
    </Button>
</LinearLayout>
```

Layout file

- The corresponding Activity

```
public class MainActivity extends Activity {
    private TextView textView;
    private Button button;

    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);

        textView = (TextView) findViewById(R.id.text);
        button = (Button) findViewById(R.id.button);
    }
}
```

- NB: setContentView should be called *before* findViewById

User Interaction

- Event-based application - the flow of control of the program is determined by events
- Callbacks: The (Android) system detects such events and calls the appropriate method defined in or registered on the View component

User interaction – listener interfaces

- onClick() from View.OnClickListener
- onLongClick() from View.OnLongClickListener
- onFocusChange() from View.OnFocusChangeListener
- onKey() from View.OnKeyListener.
  Called when the view has focus and a key is pressed
- onTouch() from View.OnTouchListener – gestures

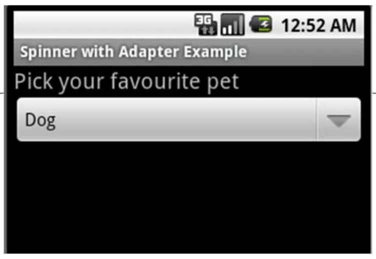
User interaction - Event Handlers

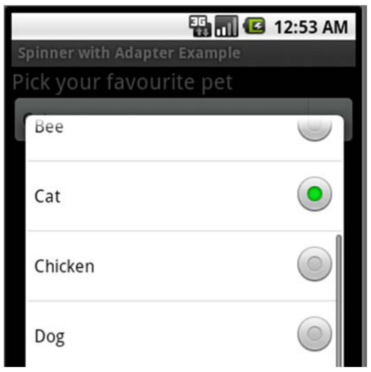
- View components also have their own callback methods for handling events, e.g.
- onKeyDown(int, KeyEvent) - Called when a new key event occurs.
- onKeyUp(int, KeyEvent) - Called when a key up event occurs.
- onTouchEvent(MotionEvent) - Called when a touch screen motion event occurs.
- onFocusChanged(boolean, int, Rect)
- Extend the View class and override the appropriate method

User interaction: OnClickListener

```
public class MainActivity extends Activity {

    public void onCreate(Bundle savedInstanceState) {
        . . .
        textView = (TextView) findViewById(R.id.text);
        button = (Button) findViewById(R.id.button);

        OnClickListener listener = new OnButtonClickListener();
        button.setOnClickListener(listener);
     }

     private class OnButtonClickListener implements OnClickListener {
         public void onClick(View v) {
            textView.setText("Hit number " + ++clicks);
         }
     }
```

Lists and Adapters





- An Adapter represents a bridge between
  - data, such as an array, a List or data from a ContentProvider,
  - and a View, such as a ListView or a Spinner.
- The Adapter creates the child views representing individual data
- The adapter updates the view(s) if the underlying data is changed
- - ArrayAdapter (array, List)
  - CursorApdapter (ContentProvider via a Cursor)

Adapters

- The Spinner, defined in res/main.xml

```
<RelativeLayout … >

    <TextView … />

    <Spinner
        android:id="@+id/PetSpinner"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/PetText"
        android:layout_below="@+id/PetText” />

</RelativeLayout>
```

Adapters

- Create a list or array:
```
private static final String[] PETS = { "Aardvark", "Ant", "Bee",… };
```

- In the Activity's onCreate(), bind the array PETS to the Spinner:
```
spinner = (Spinner) findViewById(R.id.PetSpinner);

ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
    android.R.layout.simple_spinner_item, PETS);

adapter.setDropDownViewResource(
    android.R.layout.simple_spinner_dropdown_item);

spinner.setAdapter(adapter);
spinner.setOnItemSelectedListener(new SpinnerListener());
```

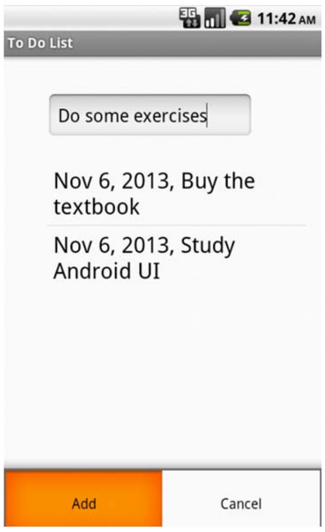
Adapters

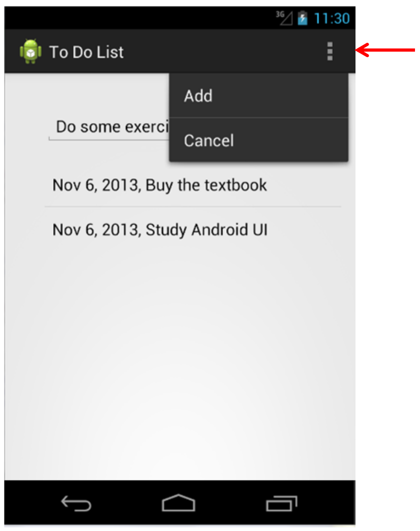
- The listener can access underlying data via the adapter

-
```
private class SpinnerListener implements OnItemSelectedListener {

    public void onItemSelected(AdapterView<?> parent, View view, int
                         pos, long id) {

        String pet = (String) spinner.getSelectedItem();
        showToast("Your favourite pet is a(n) " + pet);
    }

    public void onNothingSelected(AdapterView<?> parent) {}
}
```

Options Menu

- Activated from the device's (hardware) menu button (v < 3.0), or at top right of screen (v > 2.3)





Options Menu

- Create a menu by overriding the Activity's onCreateOptionMenu method calling menu.add(…) for each item
- Define the menus widgets in a layout file, res/menu/somemenu.xml, and, in onCreateOptionMenu, use a MenuInflater to inflate the menu from XML
- Event Handler/Listener
  - Override Activity's onOptionsItemSelected, or
  - Create and add an OnMenuItemClickListener – inefficient, discouraged

Option Menu in Activity

```
public class ToDoActivity extends Activity {

  private EditText toDoInput;
  private ListView toDoListView;
  . . .


  public boolean onCreateOptionsMenu(Menu menu) {
      . . .
  }

  public boolean onOptionsItemSelected(MenuItem item) {
     super.onOptionsItemSelected(item);
     . . .
  }
```

Option Menu, onCreateOptionsMenu

- The layout file (here: res/menu/main_activity_actions)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android" >

    <item android:id="@+id/action_add_todo"
          android:title="@string/menu_item_add"
          android:showAsAction="never" />

    <item android:id="@+id/action_cancel"
          android:title="@string/menu_item_cancel"
          android:showAsAction="never" />
</menu>
```

Option Menu, onCreateOptionsMenu

- The corresponding Activity

```
public class MainActivity extends Activity {

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

Option Menu, onCreateOptionsMenu, alternative

```
public boolean onCreateOptionsMenu(Menu menu) {

   menu.add(0, ADD_ITEM, Menu.NONE, "Add To-Do-item");
   menu.add(0, CANCEL_ITEM, Menu.NONE, "Cancel");

   return true;
 }
```

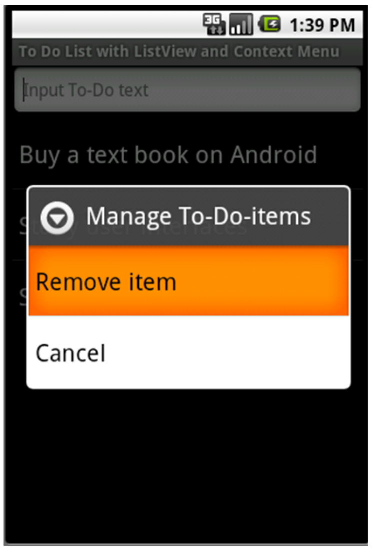
- menu.add(. . .) arguments:
  - Menu group (for example for radio buttons),
  - unique id for this menu item (an integer),
  - order,
  - title, a text

Options Menu, the Event Handler

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {

  switch (item.getItemId()) {
    case R.id.action_add_todo:
      String text = toDoInput.getText().toString();
      …
      return true;
    case R.id.action_cancel:
      toDoInput.setText("");
      return true;
    default:
      // Other alternatives -> default behavior
      return super.onOptionsItemSelected(item);
  }
}
```

Context Menu

- Associated with a specific widget, e.g. an item in a list view.
- Triggered on user event
  - long-click on widget or,
  - pressing the middle D-pad button or trackball (widget must be focused)


1:39 PM
To Do List with ListView and Context Menu
Input To-Do text
Buy a text book on Android
Manage To-Do-items
Remove item
Cancel

Context menu

- Create a context menu by overriding the *Activity'*s onCreateContextMenu method, or
- Define the menus widgets in a layout file, res/menu/somemenu.xml, and, in onCreateContextMenu, use a MenuInflater to inflate the menu from XML
- Event Handler/Listener
  - Override Activity's onContextItemSelected, or
  - Create and add an OnMenuItemClickListener – inefficient, discouraged

Context Menu, onCreateContextMenu

The layout file (here: res/menu/todolist_context_actions)

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

    <item
        android:id="@+id/action_remove_todo"
        android:showAsAction="always"
        android:title="@string/menu_item_remove"/>
    <item
        android:id="@+id/action_cancel"
        android:showAsAction="always"
        android:title="@string/menu_item_cancel"/>

</menu>
```

Context Menu, onCreateContextMenu

- The corresponding Activity

```
@Override
public void onCreate(Bundle savedInstanceState) {
      …
      // Register the ListView for this Activity's ContextMenu
      this.registerForContextMenu(toDoListView);
}

@Override
public void onCreateContextMenu(ContextMenu menu, View view,
                                ContextMenu.ContextMenuInfo menuInfo) {
      super.onCreateContextMenu(menu, view, menuInfo);
      MenuInflater inflater = getMenuInflater();
      inflater.inflate(R.menu.todolist_context_actions, menu);
}
```

Context Menu, onCreateContextMenu

- The corresponding Activity

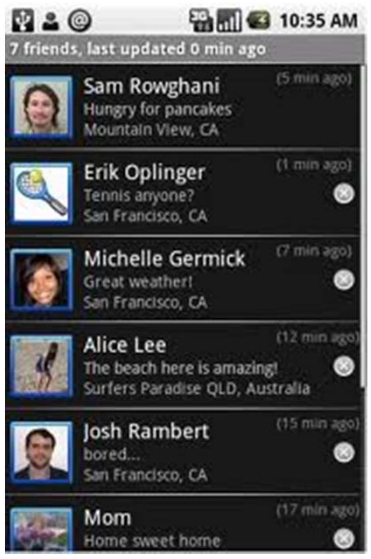
```
@Override
public boolean onContextItemSelected(MenuItem item) {

   switch (item.getItemId()) {
      case R.id.action_remove_todo:
         …
         return true;
      case R.id.action_cancel:
         // Do nothing
         return true;
      default:
         // Other alternatives -> default behavior
         return super.onContextItemSelected(item);
   }
}
```

Alternative: Contextual action mode

- Contextual action mode, displays a *contextual action bar* at the top of the screen with action items that affect the selected item(s)
- It is possible select multiple items, e.g. from a list in this mode
- The contextual action mode is available on Android 3.0 (API level 11) and higher

ListView

- Shows items in a vertically scrolling list
- Items come from a ListAdapter (or ArrayAdapter) associated with the list view.
- Simple list items: the data objects toString() method is called, or
- Customized items



10:35 AM

7 friends, last updated 0 min ago

Sam Rowghani (5 min ago)
Hungry for pancakes
Mountain View, CA

Erik Oplinger (1 min ago)
Tennis anyone?
San Francisco, CA

Michelle Germick (7 min ago)
Great weather!
San Francisco, CA

Alice Lee (12 min ago)
The beach here is amazing!
Surfers Paradise QLD, Australia

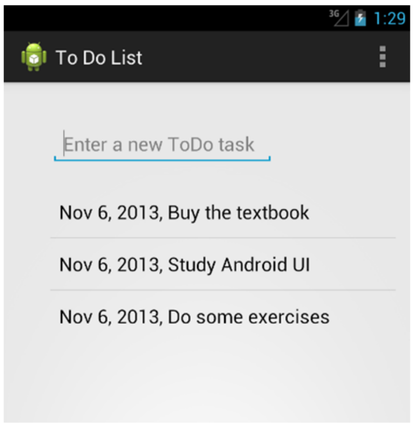
Josh Rambert (15 min ago)
bored...
San Francisco, CA

Mom (17 min ago)
Home sweet home

ListView, the To-Do list example

The layout file

```
<RelativeLayout . . . >

  <EditText
       android:id="@+id/InputText"
       . . .
  </EditText>

  <ListView
      android:id="@+id/ToDoListView"
      android:layout_width="fill_parent"
      android:layout_height="wrap_conten">
  </ListView>

</RelativeLayout>
```



Nov 6, 2013, Buy the textbook

Nov 6, 2013, Study Android UI

Nov 6, 2013, Do some exercises

ListView + Adapter

- Create an ArrayAdapter to bind the ListView to the data

```
public class ToDoActivity extends Activity {

   private ListView toDoListView;

   private ArrayList<ToDoItem> toDoItems;
   private ArrayList<ToDoItem> adapter;

    public void onCreate(Bundle savedInstanceState) {
             . . .

           toDoItems = new ArrayList<ToDoItem>();

           adapter = new ArrayAdapter<ToDoItem>(
                 this, android.R.layout.simple_list_item_1,
                 toDoItems);
          toDoListView.setAdapter(adapter);
    }
```

ListView + Adapter

```
public boolean onContextItemSelected(MenuItem item) {
   switch(item.getItemId()) {
    case REMOVE_ITEM:
       AdapterView.AdapterContextMenuInfo menuInfo;
       menuInfo = (AdapterView.AdapterContextMenuInfo) item.getMenuInfo();
       int info = menuInfo.position;
       toDoItems.remove(info);
       adapter.notifyDataSetChanged();
       return true;
       …
```

- Complete examples with ListView, Adpater, OptionsMenu and ContextMenu: ToDoListOld, menues defined in code; ToDoList, menus defined in XML, ToDoListWithCustomAdapter (custom row-view + custom adapter)

The Dalvik VM and applications

- Every Android application runs in its own process, with its own instance of the Dalvik virtual machine.
  - A device can run multiple VMs efficiently.
- The Dalvik VM executes files in the Dalvik Executable (.dex) format which is optimized for minimal memory footprint
- JIT, Just-In-Time compilation enhance performance
- Android starts the process when any of the application's code needs to be executed.
  The process is shut down when it's no longer needed *and system resources are required by other applications(!)*

Android application components

- Android applications don't have a single entry point (no main method) Instead: Consists of essential *components* that the system can instantiate and run as needed
- ***Activities** presents a visual user interface (holding View components)*
- **Services** doesn't have a visual user interface, but rather runs in the background
- **Broadcast receivers** receive and react to broadcast announcements, e.g. battery is low
- **Content providers** makes a specific set of the application's data available to other applications

Android Application Life Cycle

- *The process remains running until it is no longer needed and the system needs to reclaim its memory*
- Priorities
  1. A **foreground process**, holding an Activity at the top of the screen
  2. A **visible process**, holding an Activity that is visible
  3. A **service process**, holding a Service that has been started
  4. A **background process**, holding an Activity that is not currently visible to the user
  5. An **empty process**, that doesn't hold any active application components
- http://developer.android.com/videos/index.html#v=fL6gSd4ugSI

Android *Activity* Life Cycle

- An activity has essentially 4 states

1. *Active* or *running* when in the foreground
2. *Paused* if visible but not in focus.
   A paused activity is alive, it maintains all state and member information and remains attached to the window manager
3. *Stopped* if completely obscured by another activity.
   It still retains all state and member information
4. The Activity might be dropped from memory.
   When it is displayed again to the user, it must be completely restarted and restored to its previous state.

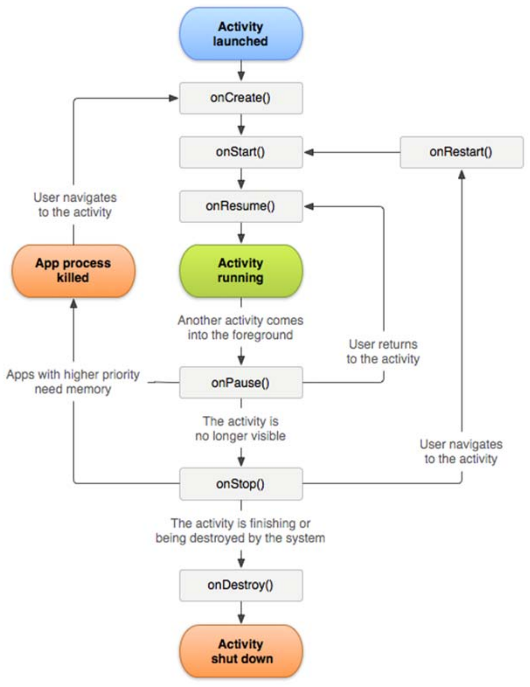
- *If an activity is paused or stopped, it's "killable".*
  The system can drop the activity from memory by either asking it to finish, or simply killing its process

Activity Life Cycle

The activity is

- Visible between onStart and onStop
- In the foreground , and can receive user input events, between onPause and onResume
- Killable after onPause

Override the appropriate life cycle methods to handle state changes


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 Life Cycle Callbacks

- Start activity
  - onCreate
  - onStart
  - onResume
- Other activity started (e.g. a phone call)
  - onPause
  - onStop
  (then onRestart, onStart and onResume)
- Press home key
  - onPause
  - onStop
- Press back key (!)
  - onPause
  - onStop
  - onDestroy
  i.e. object deallocated

Activity Life Cycle Callbacks

Full lifetime

- onCreate – override to initialize your Activity, e.g. the user interface, create services and threads, connect to database,…
- onDestroy – override to free resources created in onCreate, close external connections, …

Visible lifetime

- onStart - override to resume/restart the below mentioned
- onRestart
- *onStop – override to release all resources that aren't needed while the user is not using it*

Activity Life Cycle Callbacks

Active lifetime (Activity in foreground)

- *A callback to onPause is an indication that the user is leaving the activity and it will soon enter the stopped state*
- onResume – restore/restart the below mentioned
- onPause –
  - Stop animations or other ongoing actions that could consume CPU
  - Release system resources, such as broadcast receivers, handles to sensors (like GPS), or any resources that may affect battery life while your activity is paused and the user does not need them
  - Commit unsaved changes, but only if users expect such changes to be permanently saved when they leave (such as a draft email)
- onStop/onStart or onPause/onResume? The latter is called more frequently – put larger, more CPU intensive, shut-down operations in the former

Saving UI and similar state

- An Activity is destroyed when
  - user presses back button; UI state lost
  - Activity calls finish() on itself
  - The system kills the process to reclaim resources; major part of UI state is saved before destroy is called
- By default, the system uses a Bundle instance state to save information about each View object in your activity layout (UI state)
- Callbacks:
  - onSaveInstanceState(Bundle savedInstanceState), before the system stops the process
  - onRestoreInstanceState(Bundle savedInstanceState), called after onCreate, if a Bundle was saved
- NB!
  - Your activity will be destroyed and recreated each time the user rotates the screen

Saving UI and similar state state

```
@Override
public void onSaveInstanceState(Bundle savedInstanceState) {
    // Save the user's current game state
    savedInstanceState.putInt(STATE_SCORE, mCurrentScore);
    savedInstanceState.putInt(STATE_LEVEL, mCurrentLevel);

    // Save the view hierarchy state
    super.onSaveInstanceState(savedInstanceState);
}
```

Restoring UI and similar state

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState); // Restore UI state

    // Are we recreating a previously destroyed instance?
    if (savedInstanceState != null) {
        // Restore value of members from saved state
        mCurrentScore = savedInstanceState.getInt(STATE_SCORE);
        mCurrentLevel = savedInstanceState.getInt(STATE_LEVEL);
    } else {
        // Initialize members with default values for a new instance
    }
    ...
}
```

- Alternative: Override onRestoreInstanceState (only called if a Bundle was saved)

Where to go from here?

- Introduction to User Interfaces
  http://developer.android.com/guide/topics/ui/index.html
- Application resources and application anatomy
  http://developer.android.com/guide/topics/resources/index.html
- Managing application lifecycle
  http://developer.android.com/training/basics/activity-lifecycle/index.html
- The API documentation
  http://developer.android.com/reference/packages.html
- The textbook(!)