Contents

1 The Manifest file

A 'list' declaring all public (visible) applications components. Uses:

- Specify entry points: How are they started, what can access it (inside and outside the application)
- Information about the application: What is it allowed to do, what are its requirements (minimum, maximum SDK versions)

Manifest file should always be contained inside .apk (AndroidManifest.xml)

2 Activities

Have to be sub-classes of `android.app.Activity` and they present a visual UI.
Each activity has its own 'window'

- Only one 'window' on screen at once
- Granular management of the resources of an application
- UI layout: a 'View' specified in a separate XML file and constructed programmatically

Apps can have several activities, typically, one activity in an app is specified as the **main activity**, which is the **first screen to appear** when the user launches the app. Each activity can then start another activity in order to perform different actions.
For example, the main activity in a simple e-mail app may provide the screen that shows an e-mail inbox. From there, the main activity might launch other activities that provide screens for tasks like writing e-mails and opening individual e-mails.

2.1 Android UI

Usually activities have a full screen window, which:

- Can hover over another activity
- Can be transparent
- Can be (Multi-window / picture-in-picture)

Within the window there is a hierarchy of View objects, which can be

- Set with setContentView()
- Inflated to fill the available window

Usually specified via an XML resource (/res/layout/mylayout.xml)

2.2 View Hierarchy

Types of View subclasses

- Those that display something (Views)
- Those that do something (Widgets)
- Those that layout subviews (ViewGroups)

Shallow layout hierarchies are preferred: wide over deep with as few nested layouts as possible. We can alter the view hierarchy programmatically as the application runs:

- addView, removeView
- find a particular view by ID generated from XML layout definition

We can also bind views to data

3 Views

3.1 ViewGroups - Layouts

- **FrameLayout:** Simplest, contains a single object
- **LinearLayout:** Aligns all children in a single direction, based on the orientation attribute (Lists, left to right, top to bottom)
- **TableLayout:** Positions children into rows and columns
- **ConstraintLayout / RelativeLayout:** Lets the child views specify their position relative to the parent view or to each other. Has alignment constraints
- **ScrollView:** A vertically scrolling view, like FrameLayout only contains a single element (e.g. a LinearLayout)
- **SwipeRefreshLayout:** Detects the vertical swipe, displays a progress bar, and triggers callback methods.
- **AbsoluteLayout:** Positions and sizes child elements proportional to its own size and position or by absolute values.

3.2 Views - Widgets

A child **View** that the user can (optionally) interact with

- Button (a button) EditText (text entry)
- CalendarViewer (a calendar widget)
- ImageView (displays an image)

Handle appropriate UI events

- In code, register setOnClickListener()
- In XML layout, set android:onClick parameter

Properties / parameters, which are set via XML at build-time. Equivalent to using set get methods for modifying at run-time.

3.3 Views - Parameters

Parameters specify the details of particular Views

- Width, height: generally not in terms of absolute pixels. For example:
  `android:layout_width="match_parent"`
  `android:layout_height="wrap_content"`
- Id: used to generate a Java member variable we can refer to programmatically. For example: `android:id="@+id/my_button"`
- Methods: used to automatically bind UI events to code. For example: `android:onClick="myMethod"`

4 Tasks, Activities and Processes

4.1 Overview

Activity

- An application component that defines a screen of information
- An app is a collection of activities, consisting of both the activities you create and those you re-use from other apps

Task

- The sequence of activities a user follows to accomplish a goal.
- A single task can make use of activities from just one app, or may draw on activities from a number of different apps

Process

- Created to host components belonging to a particular app
- I.e. a task can span multiple processes

4.2 Tasks vs Activities

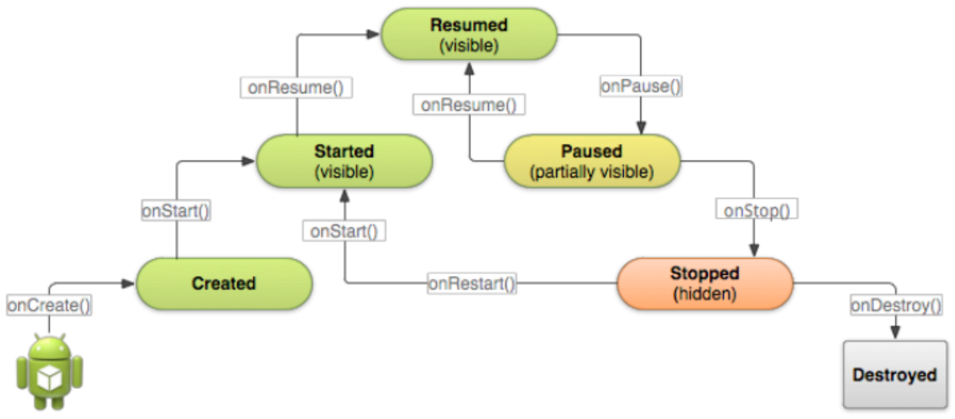
- Activities can start other activities. Forms a stack of Activities, where current activity is on the top (The back stack)
- Task is an API that represents asynchronous method calls
- Multiple activities form a Task as a task might span multiple applications.
- An activity should be an *atomic* part of a particular task. This supports reuse and integration into multiple tasks.
- Activities in a task move as a unit from foreground to background and vice versa. Switching task = switching stack

5 Activity Lifecycle

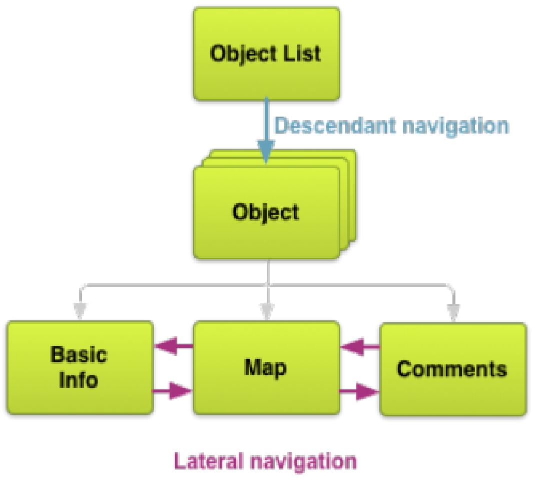
Essentially in one of three states: **Active, Paused, Stopped**. If paused or stopped, the system can drop the Activity from memory:

- Stopped activities are suspended in memory. Consume no processing resources
- Inactive activities are destroyed if memory is required. Least recently used

Transitioning between events generates events: setup ui, save state



6 Hierarchical Activity Navigation



- **Descendant** navigation
  - Users descend down an activity hierarchy
  - From parent to child activity
- **Lateral** navigation
  - Collection-related siblings: Items in a collection
  - Section-related siblings: Different sections of information about the parent
- Forms of navigation
  - Lists selecting an activity
  - Tab between screens
  - Swiping between screens
  - Buttons to start an activity

6.1 Back and up

Undo lateral and descendant navigation:

- Back:
  - Finishes the current activity
  - Resumes the next activity on the stack
  - NB tabs and swipes change information shown in the current screen, not the screen activity, so should not affect history
- Up
  - Finishes the current activity
  - Starts (or resumes) the appropriate parent activity
  - As specified in the manifest: If its in the back stack, bring it forward
  - Can create a "fake" back stack

7 Intents

Dont initantiate the Activity sub-class. Android works by passing Intent objects around:

- Intent is used to describe an operation: action and the data to operate on (as a URI)
- Allows for late runtime binding
- Glue multiple activities together

Android figures out how to honour the intent by inspecting registered manifests.

- Starting an Activity
  - Create a new Intent object
  - Specify what you want to send it to. Either implicitly, or explicitly
  - Pass the Intent object to startActivity() and new activity then started by the runtime
- Stopping an Activity
  - The called Activity can return to the original one by destroying itself, by calling the method finish()
  - Or when the user presses the back button

7.1 Intent types

7.1.1 Explicit

Provide fully qualified classname of Activity to start

```
Intent myIntent =
  new Intent(context,otherActivity.class);
startActivity(myIntent);
```

7.1.2 Implicit

Data and action are used to resolve the most appropriate activity. Usually handled by the OS

```
Intent myIntent =
  new Intent(context,otherActivity.class);
startActivity(myIntent);
Uri webpage =
    Uri.parse("http://www.cs.nott.ac.uk");
Intent myIntent =
  new Intent(Intent.ACTION_VIEW, webpage);
Uri number = Uri.parse("tel:01151234567");
Intent myIntent =
  new Intent(Intent.ACTION_DIAL, number);
startActivity(myIntent);
```

7.2 Intent Filters

Manifest specifies intent filter:

- Determine which Intents should be handled by the Activity class
- Match on action, schema of the URI
- When having multiple matches, user can choose the application.

```
<activity android:name=".BrowserActivitiy"
          android:label="@string/app_name">
  <intent-filter>
     <action android:name="android.intent.action.VIEW" />
     <category android:name="android.intent.category.DEFAULT" />
     <data android:scheme="http"/>
  </intent-filter>
</activity>
```

8 Inter-Activity Communication

- `startActivity()` doesnt allow the Activity to return a result
  - Applications usually want to maintain state
  - Remember what the user has done across all activities
  - We could store state in the broader Application context
  - But activities may be communicating between processes (IPC)
  - Entry point for other applications
- `startActivityForResult()`
  - Still takes an Intent object, but also a numerical request code
- `onActivityResult()` then called on the calling activity.
  - Data can be packaged up in an Intent / Bundle
    - Activity creates an Intent object containing the result
    - Use a Bundle to bundle complicated objects
  - Intent object then passed to `onActivityResult()` on finish
    - Returns an integer result code, set with `setResult()`
    - Returns the request code so we know which Activity we are receiving a result from

9 Saving UI state

Shouldnt rely on an Activity storing UI state

- E.g. rotating the device will destroy and recreate activity
- Aggressive OS management: application context handles various onTrimMemory calls

Before `onStop()` is called, Android will call `onSaveInstanceState()`

- To restore the UI to its previous state on restore. This is a cascading call into UI components

This allows you to save any UI state into a Bundle object

- It's essentially a key/value store, where key is string id of the element and value is any serializable data
- When the Activity is recreated, the Bundle is passed to `onCreate()` and `onRestoreInstanceState()`, giving the activity a chance to restore its state

Save other state to more **persistent storage**: SQLite / user preferences.

Reference section

atomic

Something that ”appears to the rest of the system to occur instantaneously” (One operation at a time).**Atomic operation** means an operation that appears to be instantaneous from the perspective of all other threads. You don’t need to worry about a partly complete operation when the guarantee applies.