



Android SDK

Lecture 2

Security of Mobile Devices

2018



SMD Applications

Activities

Services

Intents

Broadcast Receivers

Content Providers

Tools

Applications

Activities

Services

Intents

Broadcast Receivers

Content Providers

Tools

- ▶ `AndroidManifest.xml` file
- ▶ In the root of an app's directory
- ▶ Describes application components and resources
 - ▶ Application name and Java package name (unique)
 - ▶ Activities, Services, Broadcast Receivers, Content Providers
 - ▶ Main(default) activity
 - ▶ Permissions
 - ▶ Libraries
 - ▶ Target/Minimum API level

- ▶ Request access to resources and APIs for the application
- ▶ Provide security through sandboxing
- ▶ Declared in the Manifest
 - ▶ `<uses-permission android:name="android.permission.INTERNET" />`
- ▶ Control who can access your components and resources
 - ▶ Start Activity, start/bind Service, send broadcasts, access data in Content Providers
 - ▶ `<activity android:name=".ExampleActivity" android.permission="com.example.perm.START">`
...
`</activity>`
 - ▶ URI permissions

- ▶ res/ directory
- ▶ Each resource type in a different subdirectory
 - ▶ Specific name
 - ▶ drawable/, layout/, values/, menu/, xml/, etc.
- ▶ Different configurations may require different resources
 - ▶ Bigger screen -> different layout
 - ▶ Different language -> different strings
 - ▶ Subdirectory for each alternative set of resources
 - ▶ `<resources_name>-<config_qualifier>`
 - ▶ `drawable-hdpi/` for High Density Screens
 - ▶ Resource chosen at runtime based on device configuration
- ▶ An ID is generated for each resource name in `gen/`



Source: <http://developer.android.com>

- ▶ Resources from `res/layouts/`
- ▶ Describe the UI of an activity or part of the UI
- ▶ UI elements
 - ▶ Button, TextView, etc.
- ▶ `res/layout/filename.xml`
 - ▶ `filename` is used as resource ID
 - ▶ `R.layout.filename`
 - ▶ `R.java` includes all resource IDs
- ▶ Can be edited as xml or using graphical tools



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res
/android"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                android:orientation="vertical" >
    <TextView android:id="@+id/text"
              android:layout_width="wrap_content"
              android:layout_height="wrap_content"
              android:text="Hello , _l _am _a _TextView" />
    <Button android:id="@+id/button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Hello , _l _am _a _Button" />
</LinearLayout>
```

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_activity);
}
```

- ▶ Resources from `res/drawables/`
- ▶ Element that can be drawn on the screen
- ▶ Can be images (.png, .jpg, or .gif) or xmls
- ▶ xmls describe how an UI element reacts to input (pressed, focused)
- ▶ xmls point to images
- ▶ Visual feedback for interaction

Applications

Activities

Services

Intents

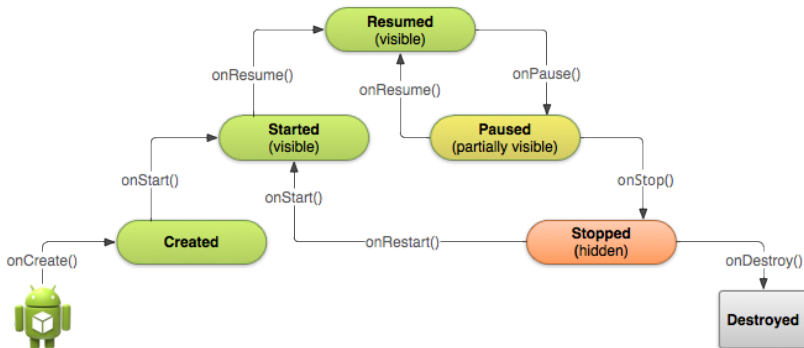
Broadcast Receivers

Content Providers

Tools

- ▶ Application component
- ▶ User interface window, provide user interaction
- ▶ Require a layout
- ▶ Can only draw and change UI from the Looper thread
 - ▶ Computationally intensive or wait based tasks on separate threads
- ▶ An application may include multiple activities
 - ▶ Only one is the main activity
 - ▶ Activities can start each other -> the previous one is stopped
 - ▶ Activity stack ("back stack")
 - ▶ Back -> activity destroyed and previous one resumed

```
<manifest ... >
  <application ... >
    <activity android:name=".ExampleActivity" />
    ...
  </application ... >
  ...
</manifest >
```

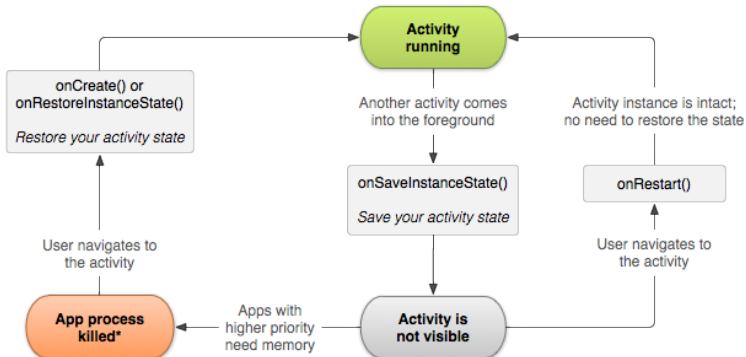


Source: <http://developer.android.com>

```
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 (is now "stopped")  
}  
@Override  
protected void onDestroy() {  
    super.onDestroy();  
    // The activity is about to be destroyed.  
}  
}
```

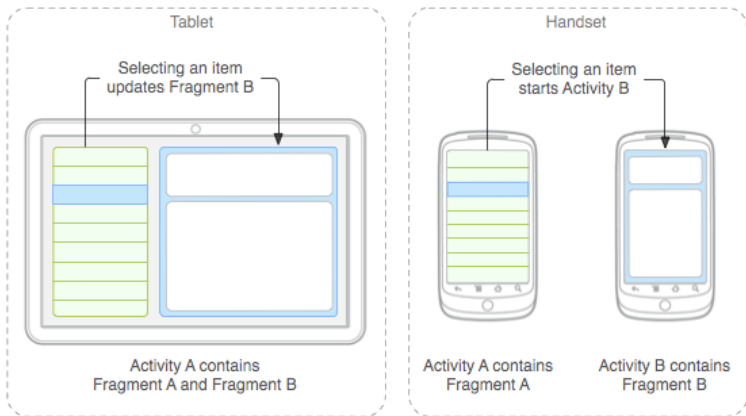

- ▶ Activities can be killed after `onPause()`, `onStop()` in low memory situations
 - ▶ The activity state (objects) are lost
 - ▶ Can preserve state by saving objects
 - ▶ User interaction can be saved and restored
 - ▶ `onSaveInstanceState()` callback
 - ▶ Save information in a `Bundle`
 - ▶ `onCreate()`, `onRestoreInstanceState()`
 - ▶ Restore the activity state
 - ▶ Threads can be stopped gracefully
 - ▶ In `onPause()` threads should be signaled to stop



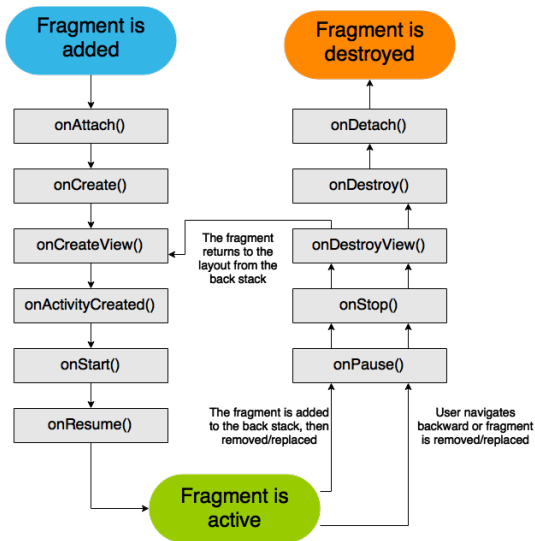
*Activity instance is destroyed, but the state from `onSaveInstanceState()` is saved

Source: <http://developer.android.com>

- ▶ Represent portions of UI in an Activity
- ▶ Can be combined to build a multi-pane UI
 - ▶ Same code, different layout for phone / tablet
- ▶ Can be reused in multiple Activities



Source: <http://developer.android.com>



Source: <http://developer.android.com>

- ▶ UI is a hierarchy of views
- ▶ View: rectangular space, provides user interaction
- ▶ Buttons, Lists, Images, TextViews, EditTexts
- ▶ Callbacks for actions
 - ▶ `onTouch()`, `onClick()`, `onLongClick()`
- ▶ A `ViewGroup` is a container for other Views or `ViewGroups`
- ▶ View / `ViewGroup` classes can be extended to create complex views
- ▶ Adapters allows for more complex data types to be displayed

Applications

Activities

Services

Intents

Broadcast Receivers

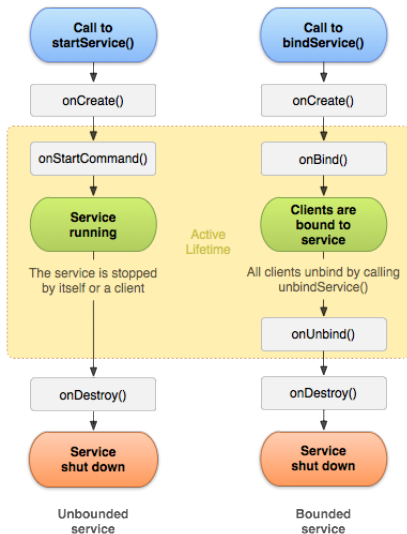
Content Providers

Tools

- ▶ Perform operations in the background
- ▶ Do not provide a UI
- ▶ Continue to run even if another application is in foreground
- ▶ Able to perform network transactions, file I/O operations, interact with content providers, etc.
- ▶ Run in the main thread of the hosting process
 - ▶ A separate thread should be created if the service performs CPU intensive or blocking operations
- ▶ Start using Intents
- ▶ Private service


```
<manifest ... >
  ...
  <application ... >
    <service android:name=". ExampleService"
             android:exported=" false" />
    ...
  </application>
</manifest>
```

- ▶ Started
 - ▶ An application component calls `startService()`
 - ▶ Performs a single operation, then stops itself and does not return a result to the caller
 - ▶ Runs even if the caller component is destroyed
- ▶ Bound
 - ▶ An application component binds to it by calling `bindService()`
 - ▶ Provides a client-server interface - send requests, return results
 - ▶ Runs as long as the application component is bound to it
 - ▶ Check for null service
 - ▶ Multiple components can bind to a service at once
 - ▶ Service destroyed after all components unbind



Source: <http://developer.android.com>

```
public class ExampleService extends Service {
    int mStartMode;           // indicates how to behave
                             // if the service is killed
    IBinder mBinder;         // interface for clients that bind
    boolean mAllowRebind;    // indicates whether onRebind
                             // should be used

    @Override
    public void onCreate() {
        // The service is being created
    }

    @Override
    public int onStartCommand(Intent intent, int flags,
                              int startId) {
        // The service is starting,
        // due to a call to startService()
        return mStartMode;
    }
    [...]
}
```



```
[...]  
@Override  
public IBinder onBind(Intent intent) {  
    // A client is binding to the service with bindService()  
    return mBinder;  
}  
@Override  
public boolean onUnbind(Intent intent) {  
    // All clients have unbound with unbindService()  
    return mAllowRebind;  
}  
@Override  
public void onRebind(Intent intent) {  
    // A client is binding to the service with bindService(),  
    // after onUnbind() has already been called  
}  
@Override  
public void onDestroy() {  
    // The service is no longer used and is being destroyed  
}  
}
```

Applications

Activities

Services

Intents

Broadcast Receivers

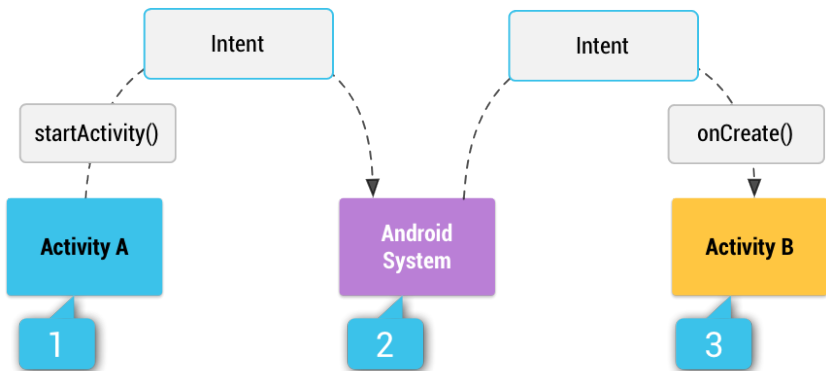
Content Providers

Tools

- ▶ An object used for delivering a message
- ▶ Includes: target, action and data
- ▶ Intent filters
 - ▶ Declare the types of intents that a component can receive
 - ▶ Specified in the manifest - `<intent-filter>`
 - ▶ `<action>`, `<data>`

- ▶ Starting an activity
 - ▶ Pass Intent to `startActivity()` or `startActivityForResult()`
- ▶ Starting or binding a service
 - ▶ Pass Intent to `startService()` or `bindService()`
- ▶ Delivering a broadcast message
 - ▶ Pass Intent to `sendBroadcast()`, `sendOrderedBroadcast()`, or `sendStickyBroadcast()`

- ▶ Explicit intents
 - ▶ Specify exactly which component to start (the class name)
 - ▶ Typically used to start components in your own app
 - ▶ Will be delivered even if there is no intent filter declared
- ▶ Implicit intents
 - ▶ Do not specify the exact component
 - ▶ Declare a general action to be performed
 - ▶ The Android system finds the appropriate component
 - ▶ Compares the intent to the intent filters in the manifest of the apps
 - ▶ Multiple components that match the intent
 - ▶ Intent filters are mandatory



Source: <http://developer.android.com>

```
// Create the text message with a string
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Verify that the intent will resolve to an activity
if (sendIntent.resolveActivity(getPackageManager()) != null) {
    startActivity(sendIntent);
}
```

```
<activity android:name=".ExampleActivity">
  <intent-filter>
    <action android:name="android.intent.action.SEND" />
    <category android:name="android.intent.category.DEFAULT" />
    <data android:mimeType="text/plain" />
  </intent-filter>
</activity>
```

Applications

Activities

Services

Intents

Broadcast Receivers

Content Providers

Tools

- ▶ Responds to system-wide broadcast announcements
- ▶ The system generates many broadcasts
 - ▶ Example: battery is low, screen has turned off, etc.
- ▶ Apps can generate broadcasts - send an announcement for other apps
- ▶ No UI, may create a notification in the status bar to alert the user
- ▶ The receiver lets other components perform the work based on the event

- ▶ Each broadcast is delivered as an *Intent*
 - ▶ Intent passed to `startBroadcast()` or `startOrderedBroadcast()`
- ▶ Local broadcasts using *LocalBroadcastManager*
 - ▶ More efficient
 - ▶ Data does not leave the app
 - ▶ Other apps cannot send the broadcast - no security holes
- ▶ Register a receiver in two ways
 - ▶ Statically in the manifest using the `<receiver>` tag
 - ▶ Dynamically using `Context.registerReceiver()`

- ▶ Normal broadcasts
 - ▶ Completely Asynchronous
 - ▶ All receivers run in an undefined order
 - ▶ `sendBroadcast()`
- ▶ Ordered broadcasts
 - ▶ Delivered to one receiver at a time
 - ▶ Each receiver executes and may propagate the result to the next or abort the broadcast
 - ▶ The order is determined using the `android:priority` in the `<intent-filter>` of the receiver
 - ▶ `sendOrderedBroadcast()`


```
<manifest ... >
  <uses-permission android:name=
    "android.permission.RECEIVE_BOOT_COMPLETED" />
  <application ... >
    <receiver android:name="ExampleReceiver" >
      <intent-filter>
        <action android:name=
          "android.intent.action.BOOT_COMPLETED" />
      </intent-filter>
    </receiver>
    ...
  </application ... >
  ...
</manifest >
```

```
public class ExampleReceiver extends BroadcastReceiver {
    @Override
    public void onReceive(Context context, Intent intent) {
        Intent intent = new Intent(context,
                                   ExampleService.class);
        context.startService(intent);
    }
}
```

Applications

Activities

Services

Intents

Broadcast Receivers

Content Providers

Tools

- ▶ Provides access to a repository of data
- ▶ System Content Providers
- ▶ To access a provider you have to request specific permissions (in the manifest)
 - ▶ `<uses-permission android:name="android.permission.READ_USER_DICTIONARY">`
- ▶ Two ways of storing data
 - ▶ File data - audio, video, photos
 - ▶ Structured data - database, array, etc.
 - ▶ Form compatible with tables of rows and columns
 - ▶ Usually a SQLite database

- ▶ Interface for accessing data in one process from another process
 - ▶ Provider and client
 - ▶ The application that owns the data includes the provider
 - ▶ The client application owns the client
- ▶ Access data using a *ContentResolver* client object
 - ▶ Its methods provide CRUD (create, retrieve, update, delete) functions
 - ▶ Calls the methods with the same name in the *ContentProvider* object

- ▶ Identify data in the provider
- ▶ Include a symbolic name for the provider (*authority*) and a name for the table (*path*)
 - ▶ Example: `content://user_dictionary/words`
 - ▶ The *ContentResolver* uses the *authority* for identifying the provider
 - ▶ From a system table with all known providers
 - ▶ The *ContentResolver* sends a query to the provider
 - ▶ The *ContentProvider* uses the *path* to identify the table

```
mCursor = getContentResolver().query(
    UserDictionary.Words.CONTENT_URI,
    mProjection,
    mSelectionClause,
    mSelectionArgs,
    mSortOrder);
[...]
```

```
mNewUri = getContentResolver().insert(
    UserDictionary.Words.CONTENT_URI,
    mNewValues);
[...]
```

```
mRowsUpdated = getContentResolver().update(
    UserDictionary.Words.CONTENT_URI,
    mUpdateValues,
    mSelectionClause,
    mSelectionArgs);
```

Applications

Activities

Services

Intents

Broadcast Receivers

Content Providers

Tools



SMD

- ▶ Official IDE
- ▶ Gradle-based build system

```
1 package com.example.athena.myapplication;
2
3 import ...
4
11
12
13
14
15 @Override
16 protected void onCreate(Bundle savedInstanceState) {
17     super.onCreate(savedInstanceState);
18     setContentView(R.layout.activity_main);
19     Toolbar toolbar = (Toolbar) findViewById(R.id.toolbar);
20     setSupportActionBar(toolbar);
21
22     FloatingActionButton fab = (FloatingActionButton) findViewById(R.id.fab);
23     fab.setOnClickListener(new View.OnClickListener() {
24         @Override
25         public void onClick(View view) {
26             Snackbar.make(view, "Replace with your own action", Snackbar.LENGTH_LONG)
27                 .setAction("Action", null).show();
28         }
29     });
30
31 @Override
32 public boolean onCreateOptionsMenu(Menu menu) {
33     // Inflate the menu; this adds items to the action bar if it is present.
34     getMenuInflater().inflate(R.menu.menu_main, menu);
35     return true;
36 }
37 }
```



Android SDK Manager

- ▶ Download SDK packages, samples, emulator images, tools

Android SDK Manager

SDK Path: /Users/athena/Documents/android-sdk-macosx

Packages

Name	API	Rev.	Status
Tools			
Android SDK Tools		24.3.3	Update available: rev. 24.4
Android SDK Platform-tools		22	Update available: rev. 23.0.1
Android SDK Build-tools		23.0.1	Not installed
Android SDK Build-tools		22.0.1	Installed
Android SDK Build-tools		21.1.2	Installed
Android SDK Build-tools		20	Not installed
Android SDK Build-tools		19.1	Not installed
Tools (Preview Channel)			
Android SDK Platform-tools		23.1 rc1	Not installed
Android 6.0 (API 23)			
Documentation for Android SDK	23	1	Not installed
SDK Platform	23	1	Not installed
Samples for SDK	23	2	Not installed
Android TV ARM EABI v7a System Image	23	2	Not installed
Android TV Intel x86 Atom System Image	23	0	Not installed

Show: Updates/New Installed [Select New or Updates](#)

Obsolete [Deselect All](#)

Done loading packages.



SMD

AVD Manager

- ▶ Manages Android Virtual Devices (for emulator)
- ▶ Emulator
 - ▶ Virtual mobile devices running on a PC

Type	Name	Resolution	API	Target	CPU/ABI	Size on Disk	Actions
	Nexus 4 API 19	768 x 1280: xhdpi	19	Android 4.4.2	arm	566 MB	
	Nexus 5 API 21 x86	1080 x 1920: xxhdpi	21	Google APIs	x86	1 GB	
	Nexus 5 API 22 x86	1080 x 1920: xxhdpi	22	Google APIs	x86	750 MB	

+ Create Virtual Device...

- ▶ QEMU
- ▶ Screen, Keyboard, Network, Audio, GPS, Radio
- ▶ Can be accelerated through virtualization
 - ▶ x86 System Image
 - ▶ Intel Hardware Accelerated Execution Manager (HAXM) on Windows and MacOS
 - ▶ KVM on Linux
- ▶ GPU accelerated

- ▶ Communication between the development tools and (virtual) device
- ▶ Three components
 - ▶ Client: runs on the development machine
 - ▶ Server: background process on the development machine
 - ▶ Daemon: background process on the (virtual) device
- ▶ Copy files (`adb push`, `adb pull`)
- ▶ Install applications (`adb install`)
- ▶ Debug (`adb logcat`)
- ▶ Shell on the (virtual) device (`adb shell`)



SMD

- `http://developer.android.com/guide/topics/manifest/manifest-intro.html`
- ▶ `http://developer.android.com/guide/topics/resources/overview.html`
- ▶ `http://developer.android.com/guide/components/activities.html`
- ▶ `http://developer.android.com/guide/components/services.html`
- ▶ `http://developer.android.com/guide/topics/providers/content-providers.html`
- ▶ `http://developer.android.com/guide/components/intents-filters.html`
- ▶ `https://developer.android.com/studio/command-line/index.html`

- ▶ Manifest file
- ▶ Permissions
- ▶ Resources
- ▶ Layouts
- ▶ Drawables
- ▶ Activity
- ▶ Service
- ▶ Intent
- ▶ Broadcast Receiver
- ▶ Content Provider
- ▶ Content URI
- ▶ Tools