



Android SDK

Lecture 2

Security of Mobile Devices

2023



SMD

Android Manifest

Resources

Activities

Services

Bibliography

Android Manifest

Resources

Activities

Services

Bibliography

- ▶ `AndroidManifest.xml` file
- ▶ In the root of an app's directory
- ▶ Describes app components and resources

- ▶ Permissions needed by the app
- ▶ Activities, Services, Broadcast Receivers, Content Providers
 - ▶ Intent filters
 - ▶ Main (default) activity
 - ▶ Permissions needed by other apps to access your app components
- ▶ Hardware and software features needed by the app
- ▶ Package name, minSdk, targetSdk, external libraries - moved in `build.gradle`

- ▶ Request access to resources and APIs for the app
- ▶ Provide security through sandboxing
- ▶ Declared in the Manifest
- ▶ `<uses-permission
android:name="android.permission.INTERNET"/>`
- ▶ `<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE"/>`
- ▶ Runtime & special permissions need to be requested at runtime

- ▶ 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

Android Manifest

Resources

Activities

Services

Bibliography

- ▶ Images, layouts, strings, etc.
- ▶ res/ directory
- ▶ Each resource type in a different subdirectory
 - ▶ drawable/ - images, xml files
 - ▶ layout/ - describe the UI
 - ▶ values/ - strings, integers, colors
 - ▶ menu/ - app menus
 - ▶ xml/ - configuration files
- ▶ An ID is generated for each resource name in gen/

- ▶ 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.id.start_button`
- ▶ 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 , I am a TextView" />
  <Button android:id="@+id/button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Hello , I am a Button" />
</LinearLayout>
```

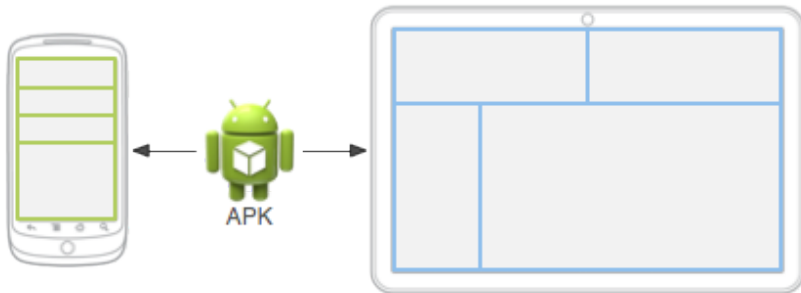
```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main_activity);

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

    button.setOnClickListener(new View.OnClickListener() {
        public void onClick(View v) {
            //do somethig
        }
    });
}
```

- ▶ 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

- ▶ 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



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

Android Manifest

Resources

Activities

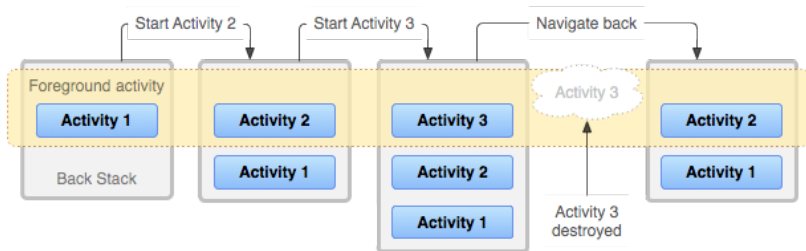
Services

Bibliography

- ▶ Activities
- ▶ Services
- ▶ Broadcast Receivers
- ▶ Content Providers

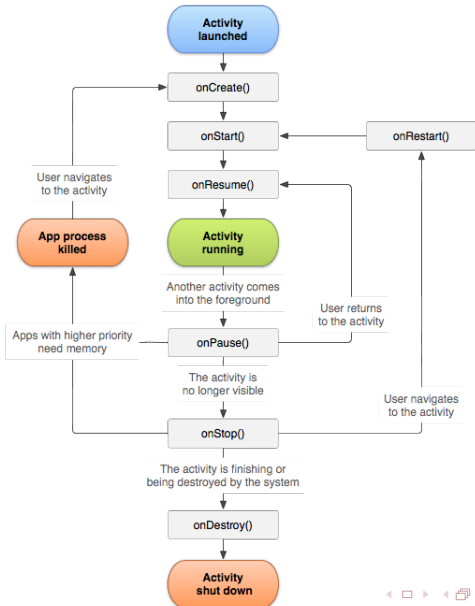
- ▶ App component
- ▶ User interface window, provide user interaction
- ▶ Require a layout
- ▶ Can only draw and change UI from the main UI thread
 - ▶ Computationally intensive or wait based tasks on separate threads

- ▶ An app 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



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

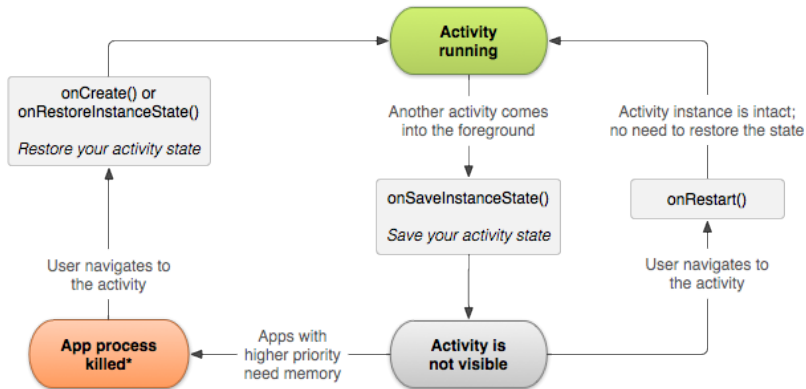
```
<manifest ... >
  <application ... >
    <activity android:name=".ExampleActivity" />
    ...
  </application ... >
  ...
</manifest >
```



```
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
 - ▶ UI 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

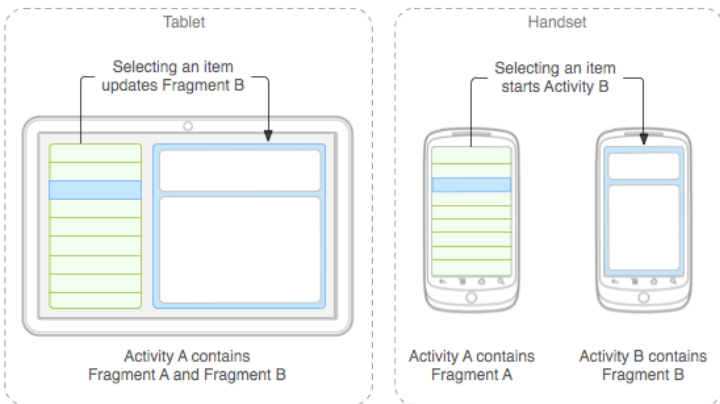


*Activity instance is destroyed, but the state from `onSaveInstanceState()` is saved

- ▶ A new activity is started using an Intent
 - ▶ `startActivity()`
 - ▶ `startActivityForResult()`

```
Intent intent = new Intent(this, SignInActivity.class);
startActivity(intent);
```

- ▶ 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
- ▶ Modularity and reusability



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

Android Manifest

Resources

Activities

Services

Bibliography

- ▶ Perform operations in the background
- ▶ Do not provide a UI
- ▶ Continue to run even if another app is in foreground
- ▶ Able to perform network transactions, file I/O operations, interact with content providers, etc.

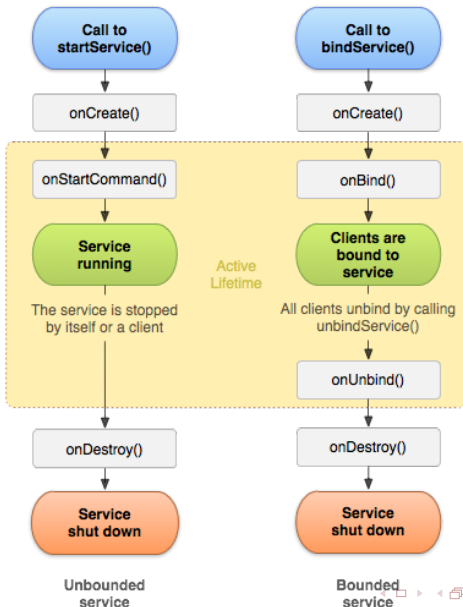
- ▶ Runs by default in the main thread of the hosting process
- ▶ Performing CPU intensive or I/O operations may block the UI
 - ▶ Run in a separate thread
 - ▶ Run in a separate process
- ▶ Start using Intents
- ▶ Private service

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

- ▶ Started
 - ▶ An app 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 app component binds to it by calling `bindService()`
 - ▶ Provides a client-server interface - send requests, return results
 - ▶ Runs as long as the app component is bound to it
 - ▶ Multiple components can bind to a service at once
 - ▶ Service destroyed after all components unbind
- ▶ A service can be both started and bound

- ▶ 3 ways to define the interface:
 - ▶ Extending the Binder class
 - ▶ When the service is private
 - ▶ Using Messengers
 - ▶ 2 Messenger objects for two-way communication
 - ▶ Simple IPC
 - ▶ Using AIDL
 - ▶ Similar to other RPC solutions
 - ▶ .aidl file



```
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  
}  
}
```


- ▶ Operation noticeable to the user
- ▶ Examples: music player, fitness app
- ▶ Must display a notification that can be dismissed by the user
- ▶ May configure non-dismissable notification
- ▶ App must have `android.permission.FOREGROUND_SERVICE`

- ▶ Starting a foreground service:
 - ▶ Start the service with `startForegroundService()`
 - ▶ Call `startForeground()` in the callback `onStartCommand()`
- ▶ Remove a service from foreground:
 - ▶ Call `stopForeground()`
 - ▶ The service will keep running in the background

- ▶ Performs an operation that is not noticed by the user
- ▶ From API 26, are not allowed to perform certain operations
 - ▶ When the app is not in foreground
 - ▶ e.g. access location in a background is not allowed

- ▶ The system may kill services when memory is low
- ▶ A **started** long running service is more likely to be killed
- ▶ A service **bound** to a foreground activity is less likely to be killed
- ▶ A **foreground** service is rarely killed
- ▶ Check for null service
- ▶ Killed services may be restarted by the system when memory is available
 - ▶ Design to handle restarts from the system

- ▶ A new service is started/bound using an Intent
 - ▶ `startService()`
 - ▶ `startForegroundService()`
 - ▶ `bindService()`

```
Intent intent = new Intent(this, HelloService.class);
startService(intent);
```

Android Manifest

Resources

Activities

Services

Bibliography



- ▶ <http://developer.android.com/guide/topics/manifest/manifest-intro.html>
- ▶ <https://developer.android.com/guide/topics/permissions/overview>
- ▶ <https://developer.android.com/guide/topics/resources/providing-resources>
- ▶ <https://developer.android.com/guide/components/activities/intro-activities>
- ▶ <https://developer.android.com/guide/components/activities/activity-lifecycle>



- ▶ <https://developer.android.com/guide/components/activities/tasks-and-back-stack>
- ▶ <https://developer.android.com/guide/fragments>
- ▶ <https://developer.android.com/guide/components/services>
- ▶ <https://developer.android.com/guide/components/foreground-services>
- ▶ <https://developer.android.com/guide/components/bound-services>

- ▶ Manifest file
- ▶ Permissions
- ▶ Resources
- ▶ Layouts
- ▶ Drawables
- ▶ Activity
- ▶ Service
- ▶ Started Service
- ▶ Bound Service
- ▶ Foreground Service