Android SDK

Lecture 2

Security of Mobile Devices

2020

Applications

- Activities
- Services
- Intents
- Broadcast Receivers
- Content Providers
- Tools

Outline

Android Manifest

- `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

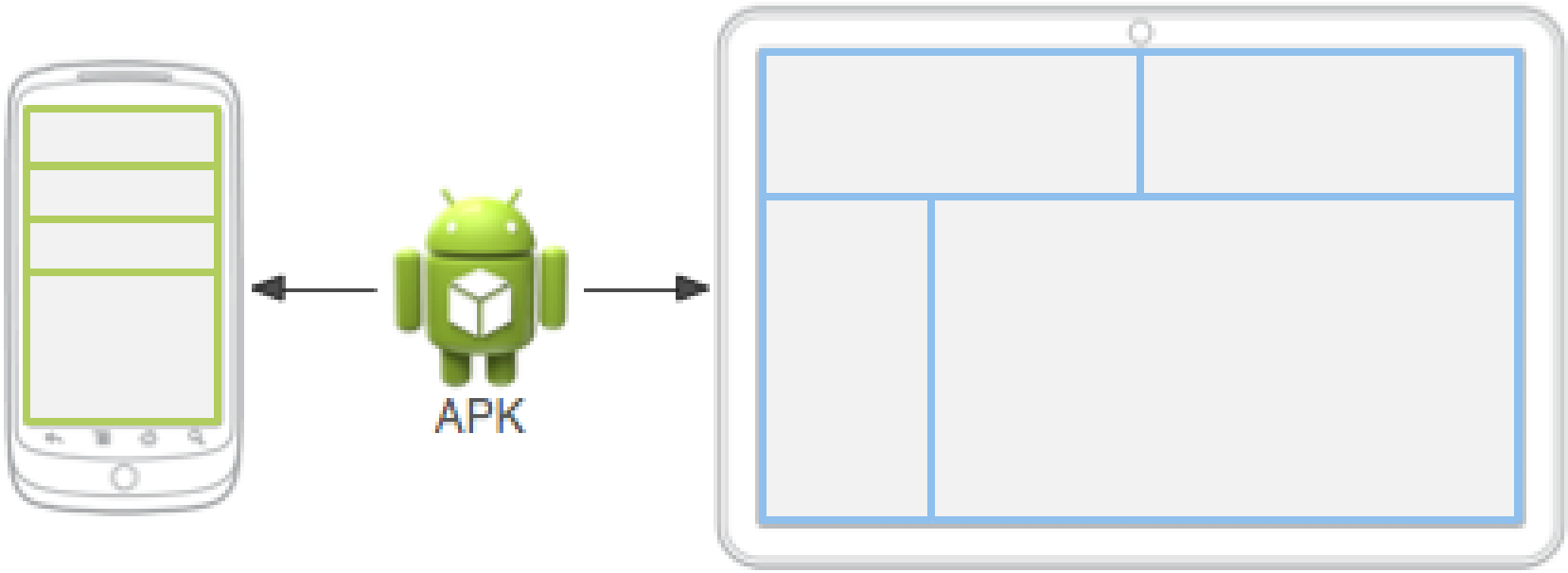
Permissions

- 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

Resources

- `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/`

Resources - Different Configurations



Source: http://developer.android.com

Layouts

- 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

Layouts - Example

```
<?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);
}
```

Drawables

- 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

Outline

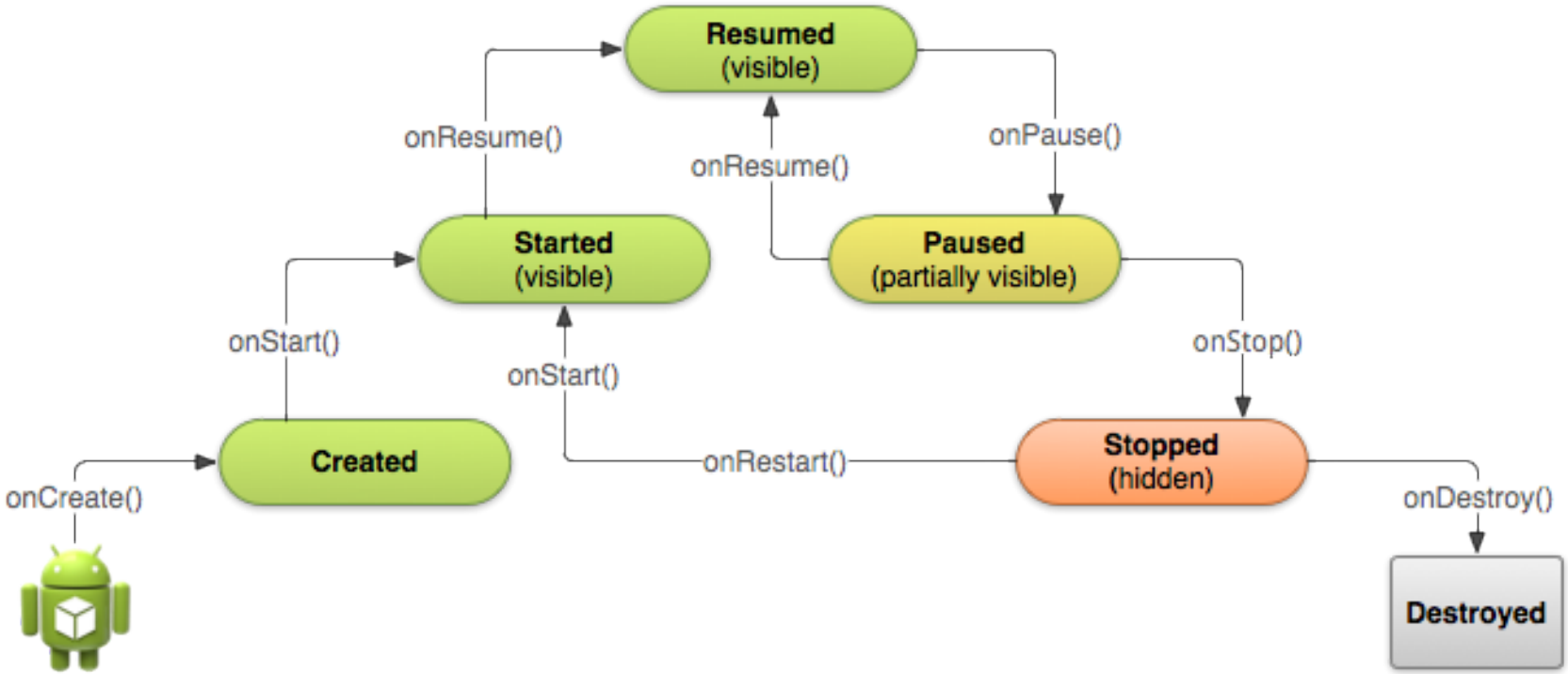
Activity

- 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

Activity Declaration in Manifest

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

Activity Lifecycle



Source: http://developer.android.com

Activity Lifecycle 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").
    }
[...]
```

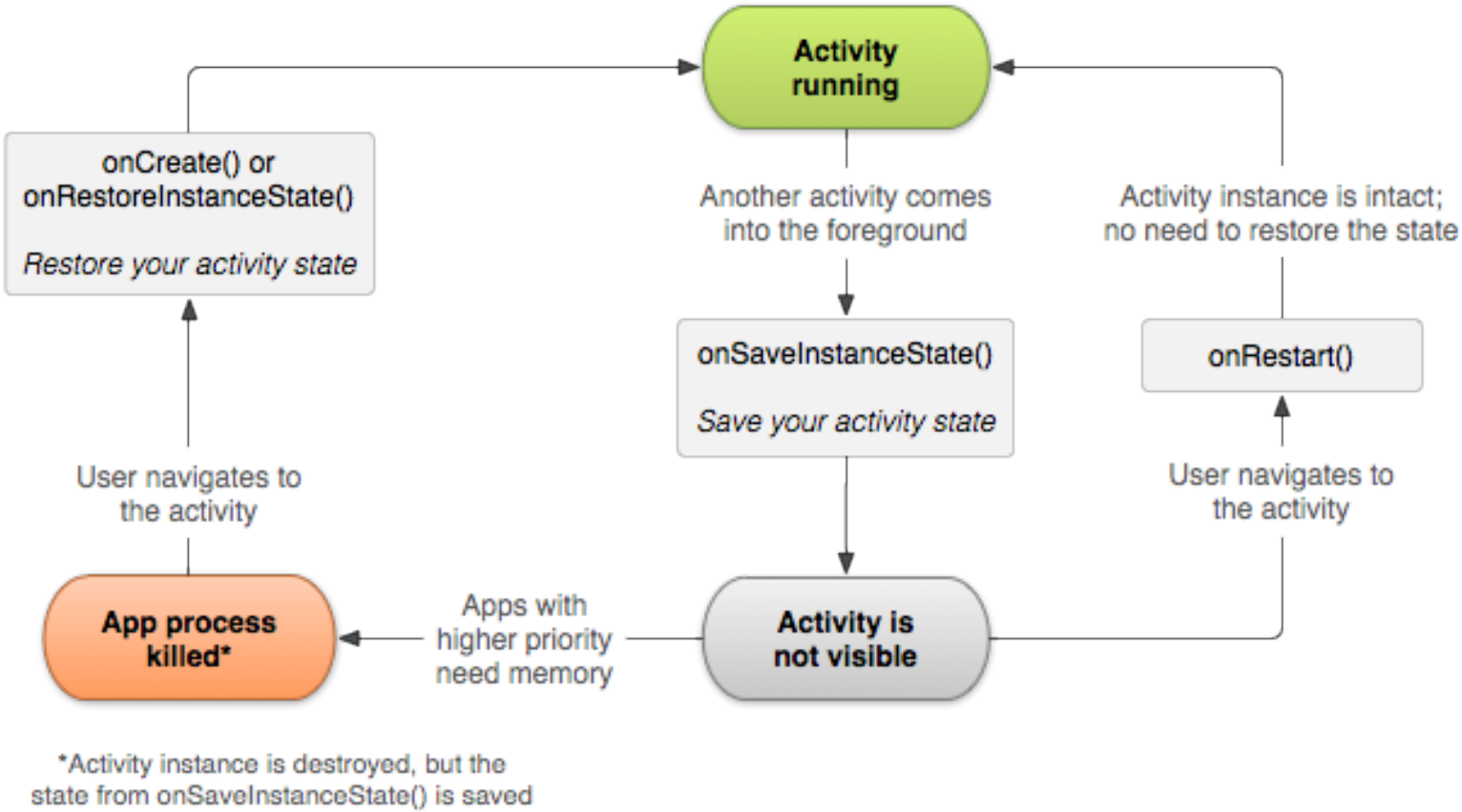
Activity Lifecycle Callbacks (2)

```
[...]
    @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.
    }
}
```

Activity Lifecycle

- 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 graciously
    - In `onPause()` threads should be signaled to stop

Save Activity State

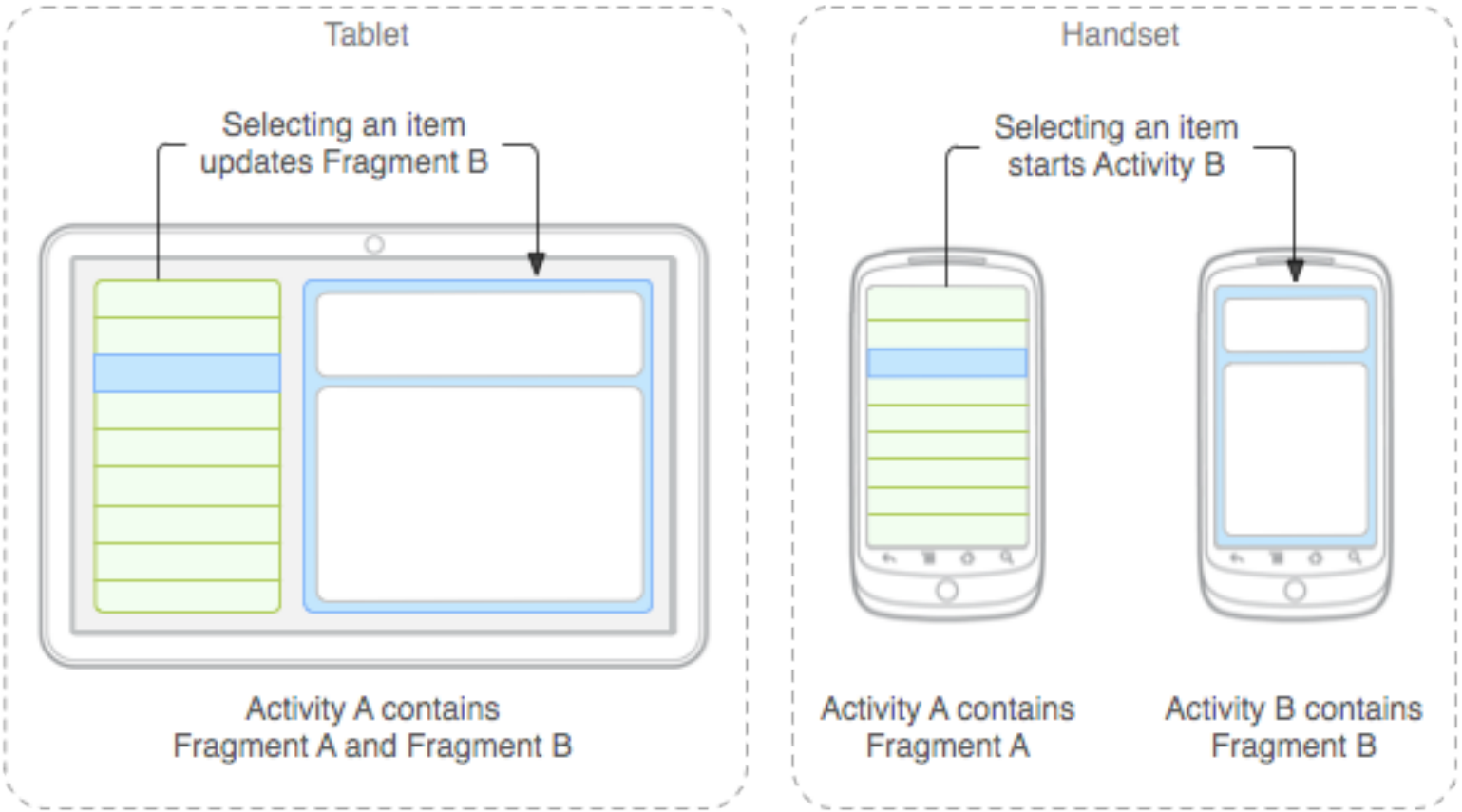


Source: http://developer.android.com

Fragments

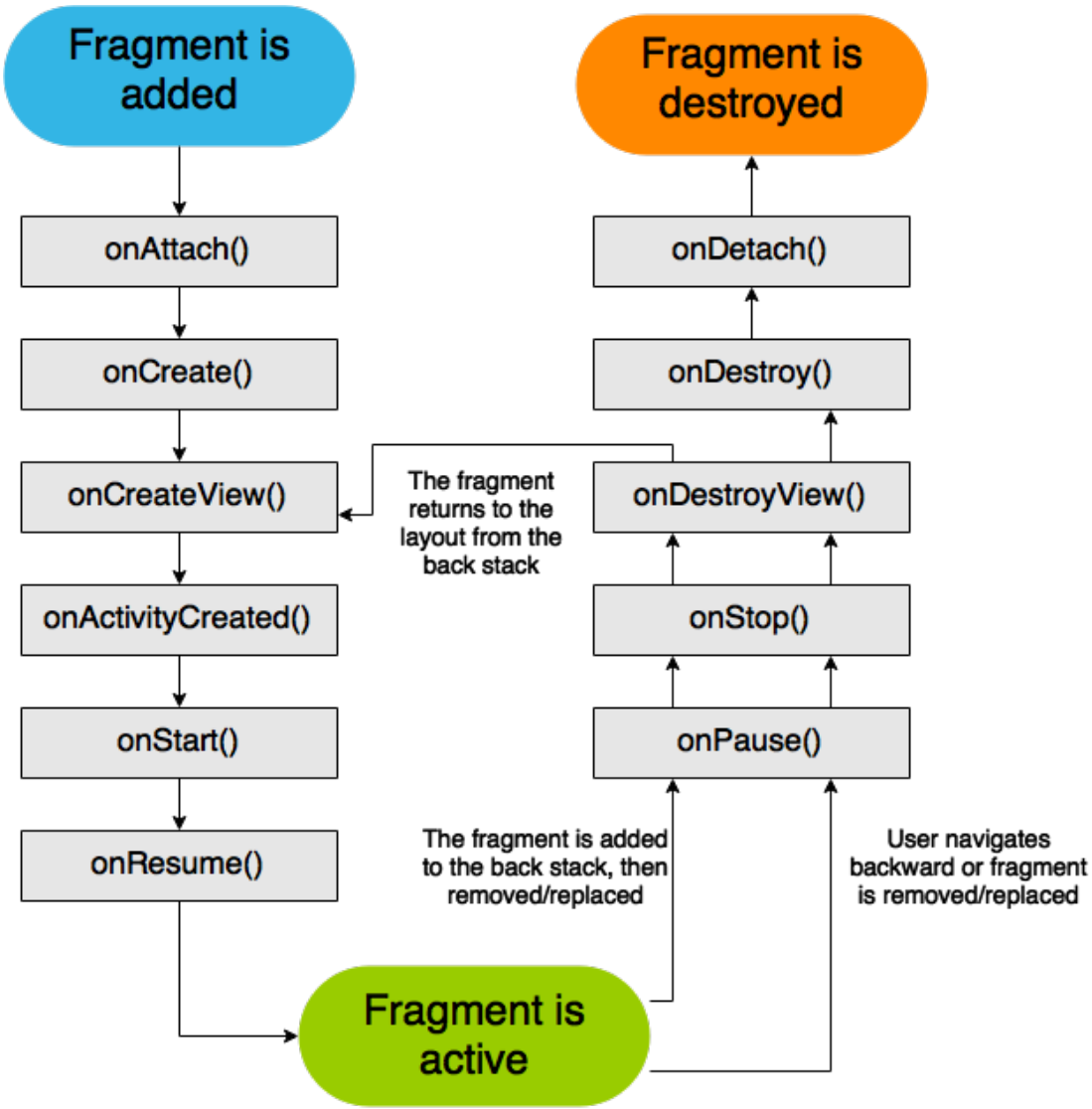
- Represent portions of UI in an Activity
- Can be combined to build a multi-pane UI
  - Same code, different layout for phone / tablet
- Can be reused in multiple Activities

Fragments



Source: http://developer.android.com

Fragment Lifecycle



Source: http://developer.android.com

UI

- 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

Outline

Services

- 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

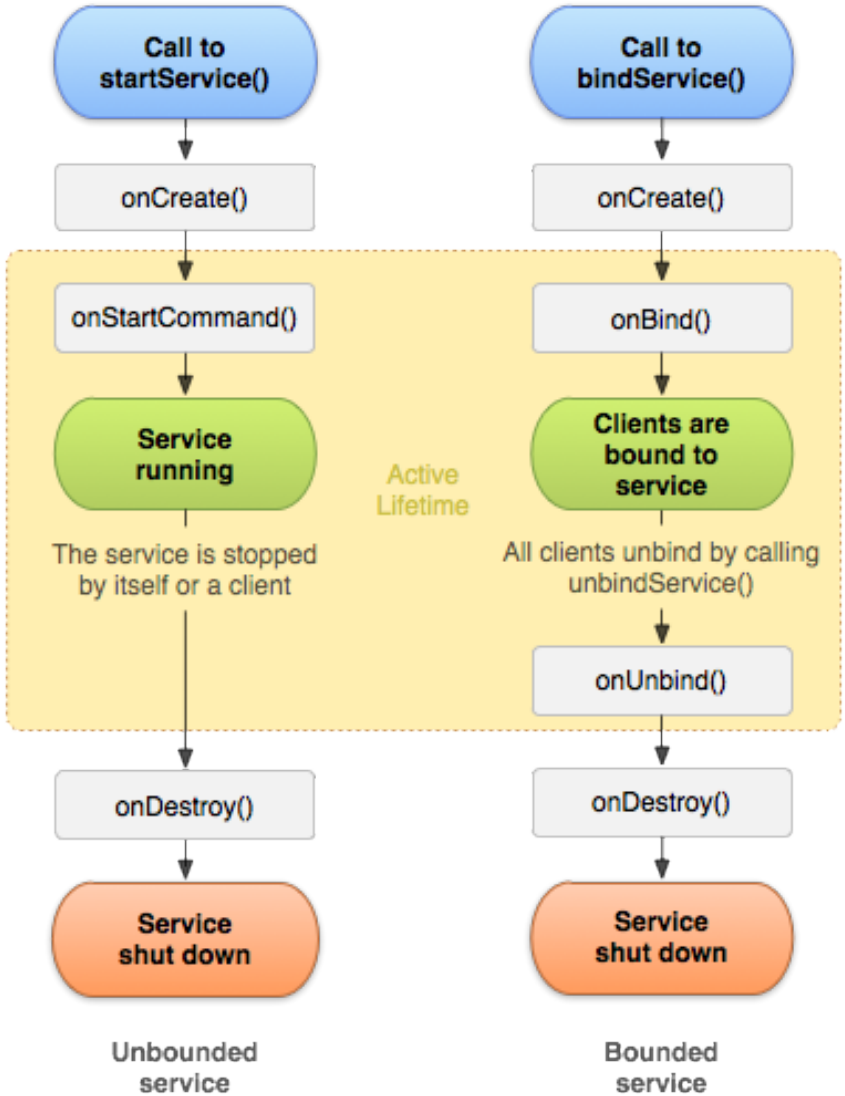
Service Declaration in Manifest

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

Types of services

- 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

Service Lifecycle



Source: http://developer.android.com

Service Lifecycle Callbacks

```
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;
    }
[...]
```

Service Lifecycle Callbacks (2)

```
[...]
    @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
    }
}
```

Outline

Intent

- 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>`

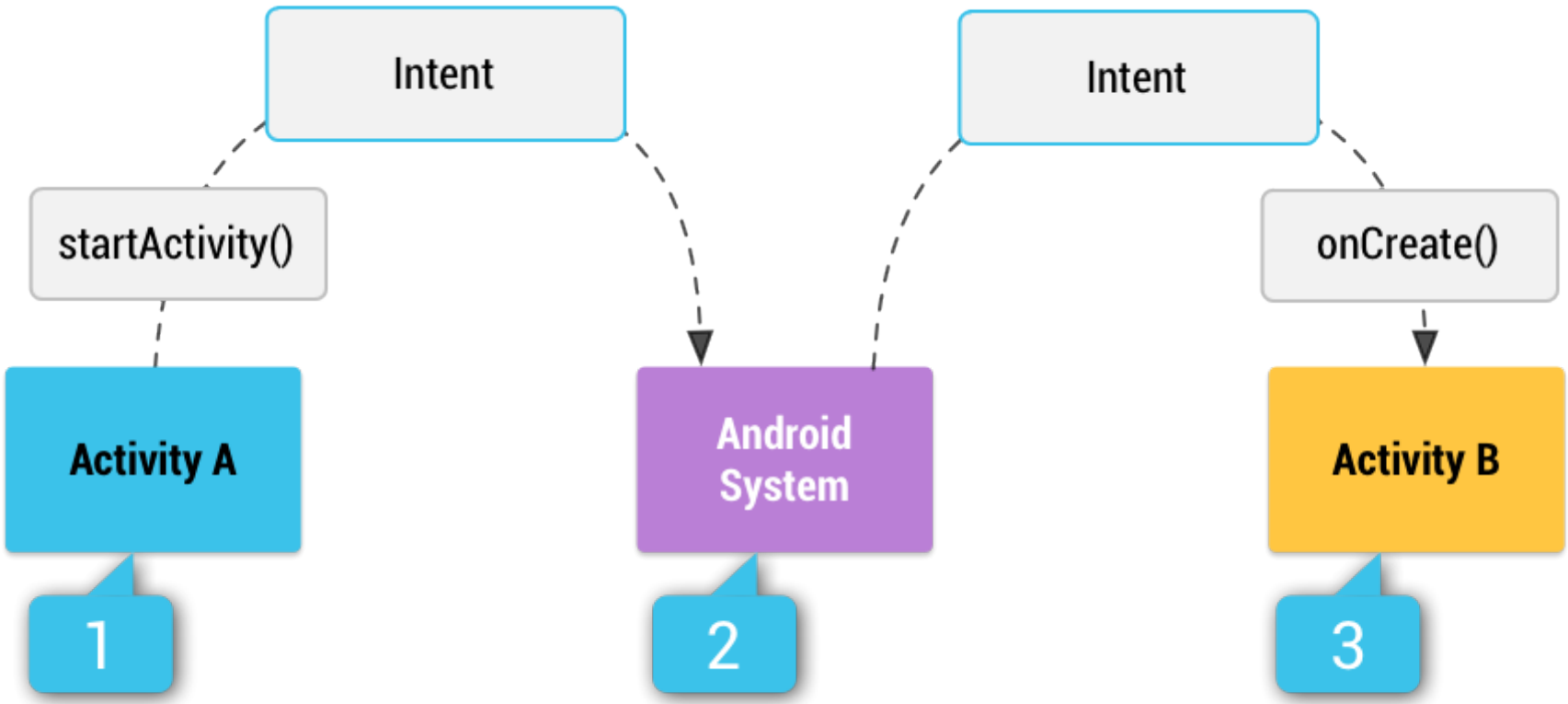
Intents - Use Cases

- 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()` or `sendOrderedBroadcast()`

Types of Intents

- 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

Implicit Intents



Source: http://developer.android.com

Implicit Intents - Example

```
// 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);
}
```

Intent Filter - Example

```
<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>
```

Outline

Broadcast Receiver

- 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

Broadcast Receiver

- 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()`

Types of Broadcasts

- 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()`

Broadcast Receiver Declaration in Manifest

```
<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 >
```

Broadcast Receiver Example

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

Outline

Content Provider

- 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

Content Provider

- 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

Content URIs

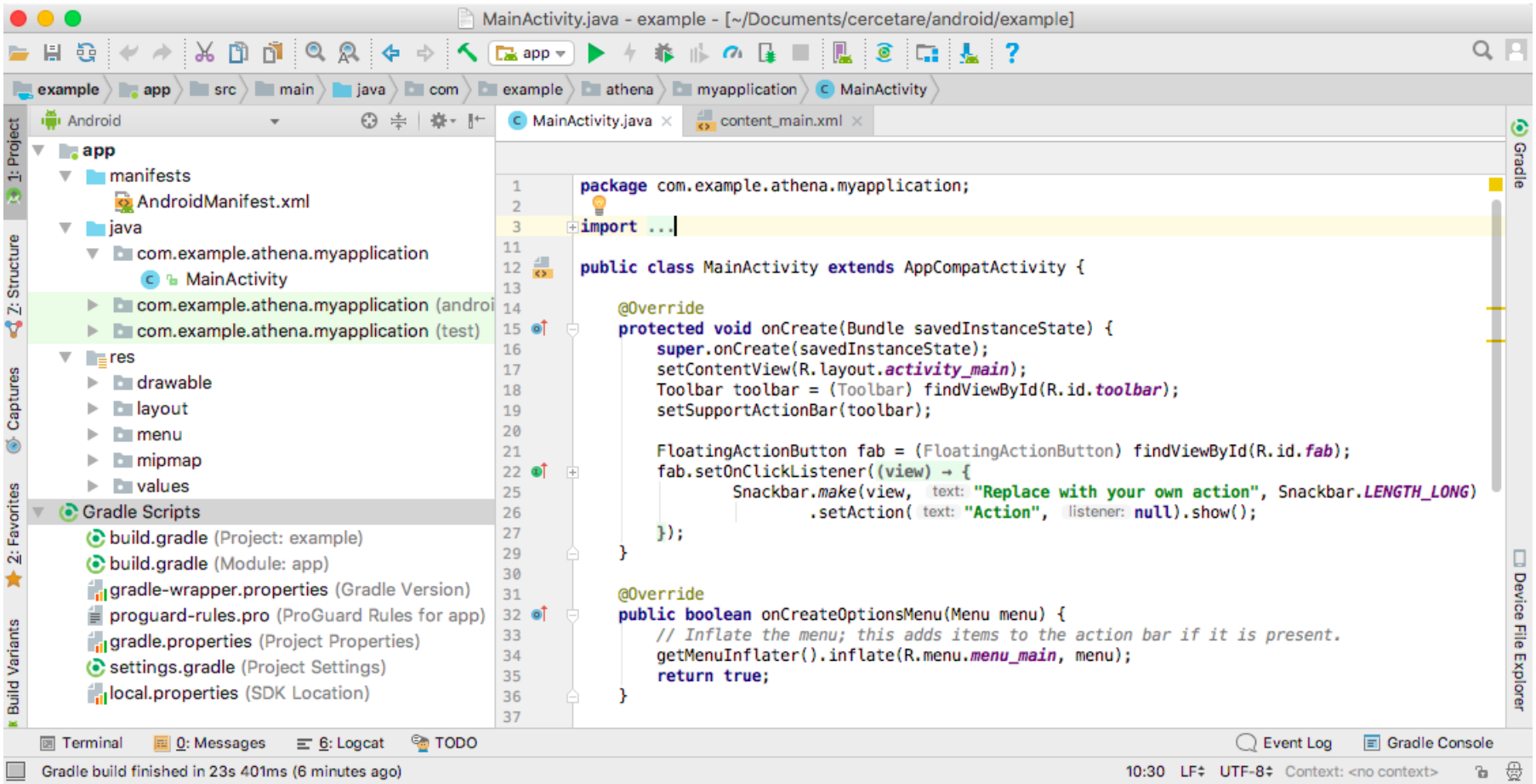
- Identify data in the provider
- Include a symbolic name for the provider (*authority*) and a name for the table (*path*)
  - Example: `content://user_dictionary/words`
  - The *ContentResolver* uses the *authority* for identifying the provider
  - From a system table with all known providers
  - The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

Content Provider - Example

```
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);
```

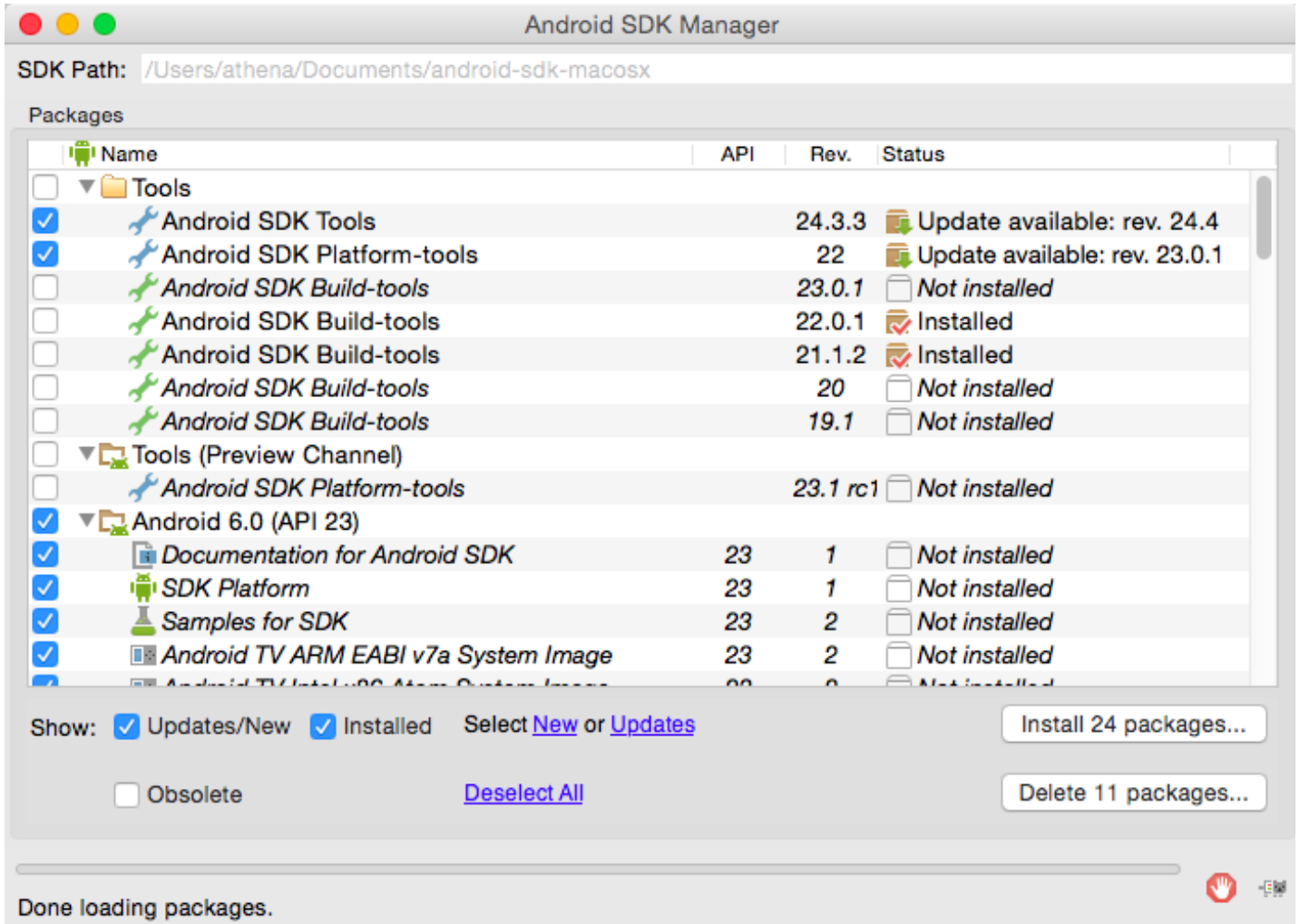
Outline

- Official IDE
- Gradle-based build system

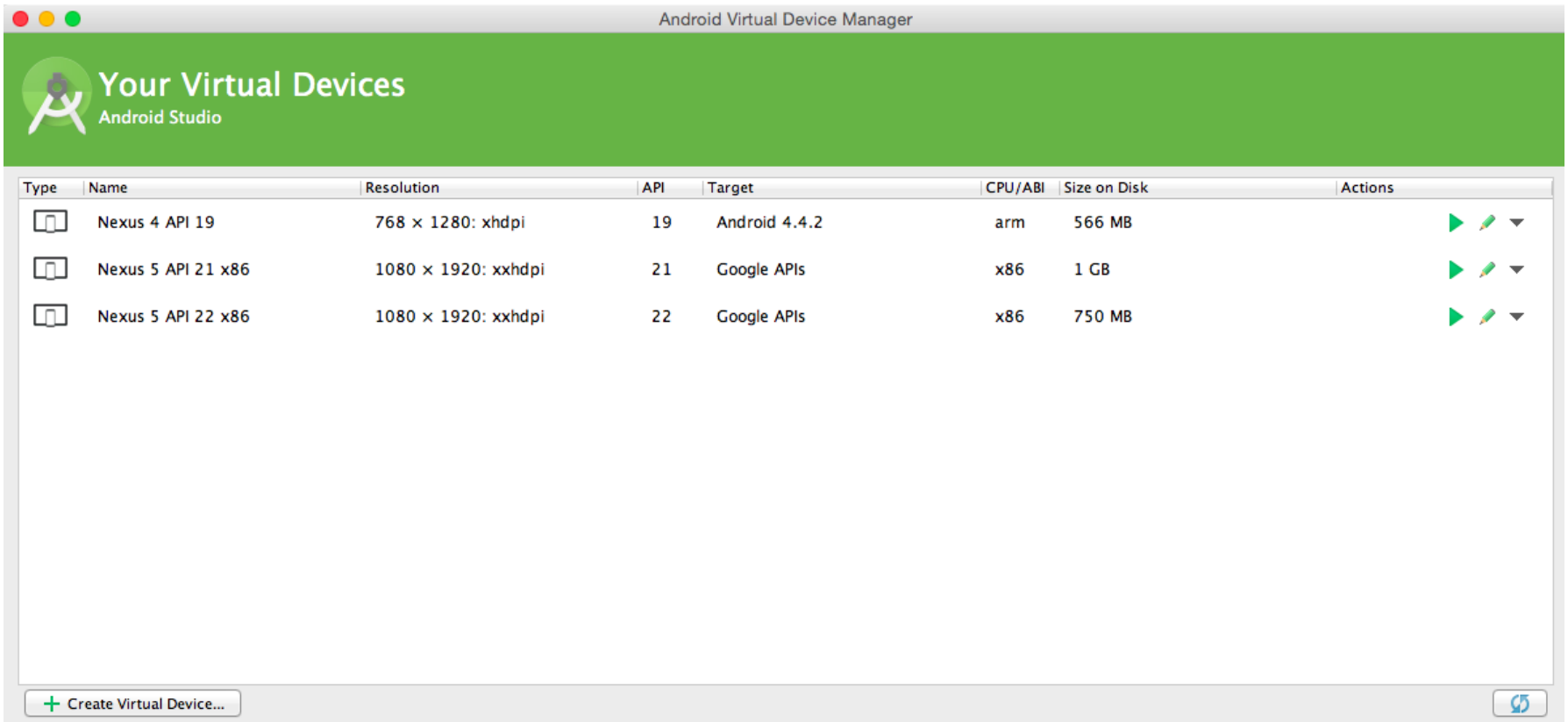


Android SDK Manager

- Download SDK packages, samples, emulator images, tools



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



Emulator

- 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

Android Debug Bridge

- 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`)

- 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

Keywords

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