

Introduction to Android

Android Platform

For mobile devices

- Including Sensors, Maps, location, wearables, etc.

Some Android Trivia ...

- Google Bought Android in 2005
- Android was officially open sourced in 2007 (Open Handset Alliance)
- Android is built on top of Linux
- Android security concerns are handled by Linux
- All Android applications run as separate Linux processes

- Dalvik used to be the process virtual machine used by Android
 - Java:
Java source code -> Java byte code -> Java VM executes
 - Dalvik:
Java source code -> Java byte code -> Dalvik byte code -> Dalvik VM executes
 - Dalvik uses JIT (Just-In-Time) compilation.
- Dalvik had been replaced by ART (Android RunTime).
 - ART uses ahead-of-time (AOT) compilation by compiling entire applications into native machine code upon installation.
- User Interface: Java user interface libraries (AWT and Swing) were replaced with Android user interface libraries.

Android Versions

for development

for users

Android Versions:

- Cupcake (API level 3, Android 1.5)
- Donut
- Éclair
- Froyo
- Gingerbread (Android 2.3, added OpenGL ES 2.0)
- Honeycomb
- Ice Cream Sandwich
- Jelly Bean (API level 18)
- KitKat (API level 19)
- Lollipop (API level 21, Android 5.0)
- Marshmallow (API level 23, Android 6.0)
- Nougat (API level 24, Android 7.0)
- Oreo (API level 26, Android 8.0)

Dalvik replaced by ART

Android SDK


- Download from
<https://developer.android.com/sdk/index.html>

Google Studio is the SDK to use.
Google is no longer supporting Eclipse.




Android Studio


Version 2.3.1

 Start a new Android Studio project

 Open an existing Android Studio project

 Check out project from Version Control ▾

 Import project (Eclipse ADT, Gradle, etc.)

 Import an Android code sample

 Configure ▾  Get Help ▾



New Project

Android Studio

Configure your new project

Application name:

Company Domain:

Package name:

[Edit](#)

Project location:

Name that will appear
on Google Play Store

The SDK automatically
concatenates the
Application Name into
one word and use it as
the Project Name.
Reverse of company
domain + Project
Name = Package Name

Cancel

Previous

Next

Finish

The Android OS requires that every app installed on a device have a package name that is at least 2 levels deep.

- myapp is not acceptable
- mycompany.myapp is acceptable



Target Android Devices

Select the form factors your app will run on

Different platforms require separate SDKs

☒ Phone and Tablet

Minimum SDK

API 17: Android 4.2 (Jelly Bean)

Lower API levels target more devices, but have fewer features available. By targeting API 17 and later, your app will run on approximately **61.3%** of the devices that are active on the Google Play Store. [Help me choose.](#)

☐ TV

Minimum SDK

API 21: Android 5.0 (Lollipop)

☐ Wear

Minimum SDK

API 21: Android 5.0 (Lollipop)

☐ Glass (Not Installed)

Minimum SDK

Cancel

Previous

Next

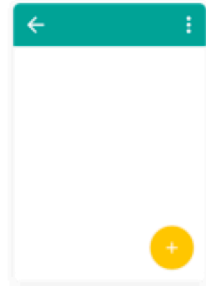
Finish



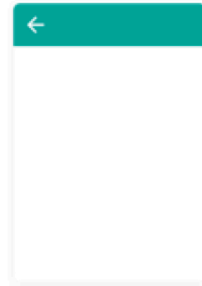
Add an activity to Mobile



Add No Activity



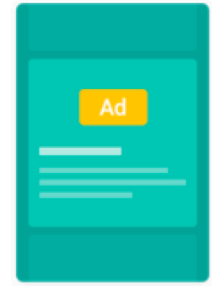
Blank Activity



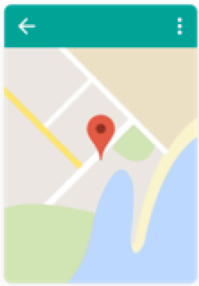
Empty Activity



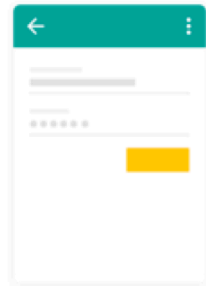
Fullscreen Activity



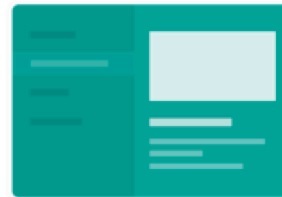
Google AdMob Ads Activity



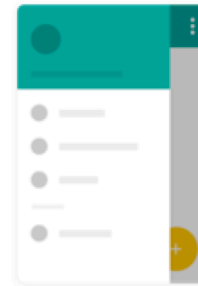
Google Maps Activity



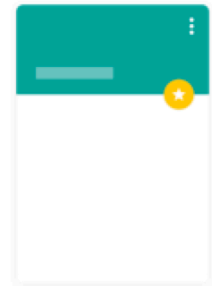
Login Activity



Master/Detail Flow



Navigation Drawer Activity



Scrolling Activity

Cancel

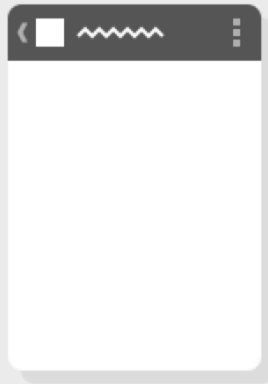
Previous

Next

Finish



Customize the Activity



Blank Activity

Creates a new blank activity with an action bar.

Activity Name:	<input type="text" value="MainActivity"/>
Layout Name:	<input type="text" value="activity_main"/>
Title:	<input type="text" value="MainActivity"/>
Menu Resource Name:	<input type="text" value="menu_main"/>

The name of the activity class to create

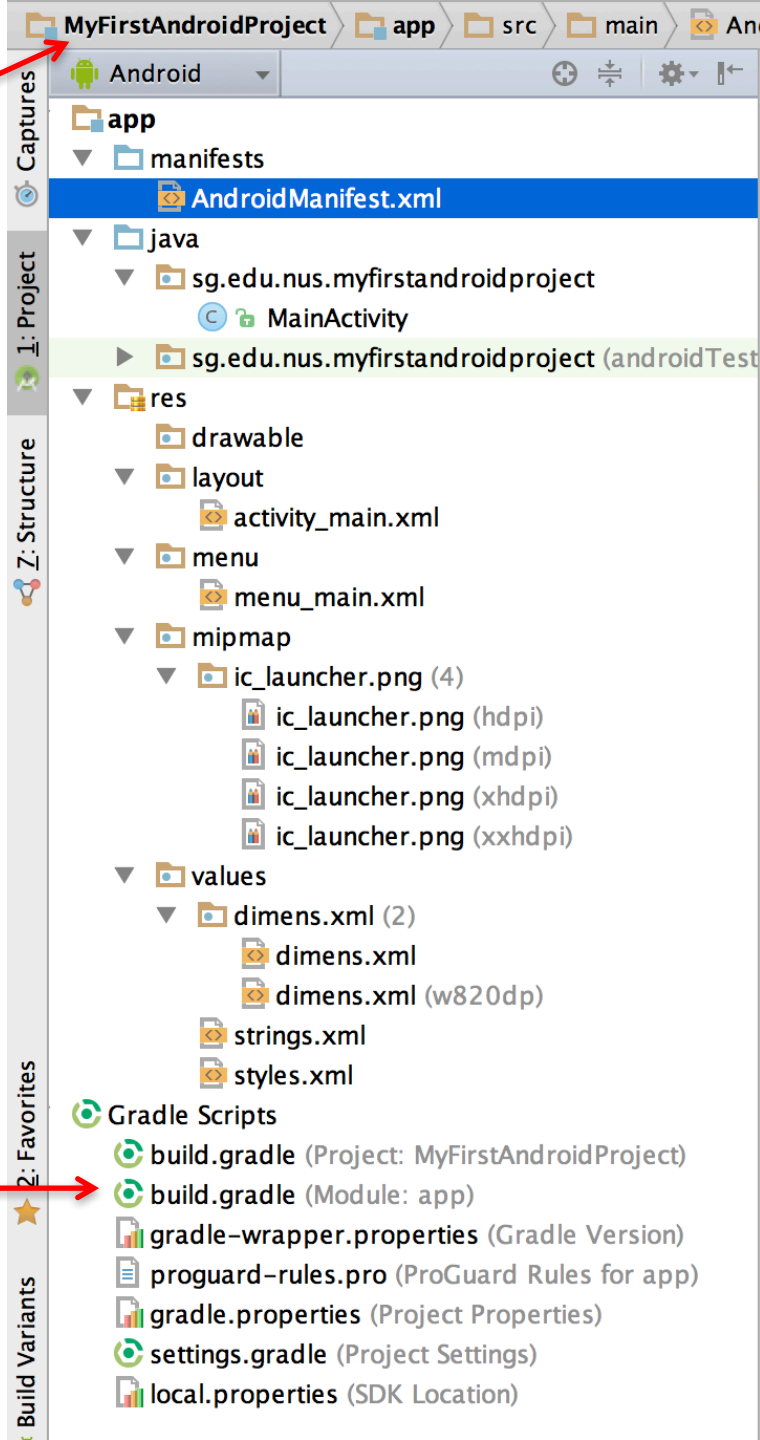
Cancel

Previous

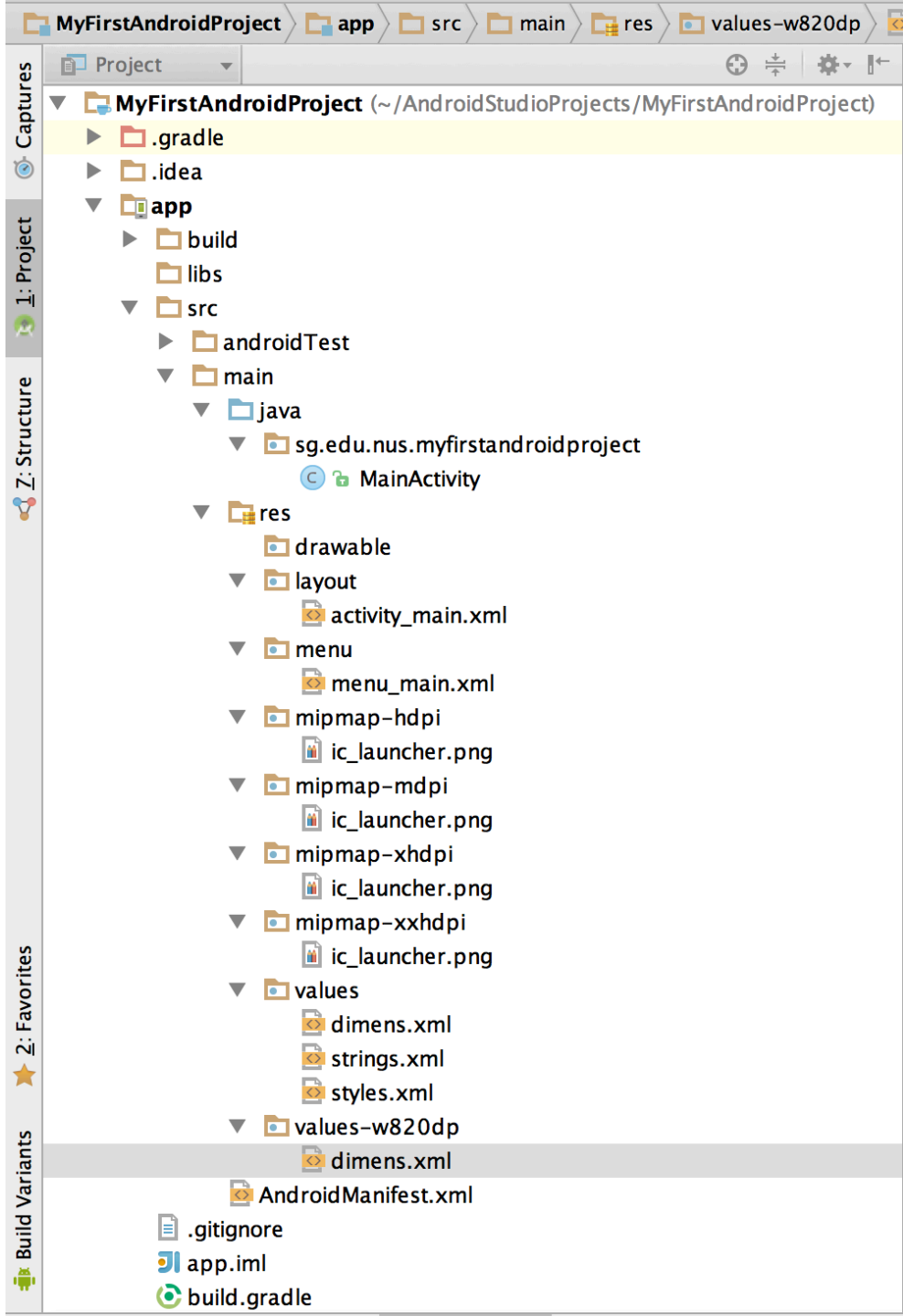
Next

Finish

Project Name



Gradle is a build tool that Google use for Android



- **AndroidManifest.xml**
 - specify permissions, intents, etc.
- Layout XML file eg. **res/layout/main.xml**
 - specifies the graphical layout of your screen
- Strings XML file **res/values/strings.xml**
 - for localization
 - maps the text resource ID with actual text that will be displayed
- Java Source Code

Android Platform Overview

Applications

Home Screen, Contacts, Phone, Browser, etc

Application Framework

Activity Manager, Window Manager, Content Providers, etc

Libraries

Media Framework, SQLite,
OpenGL, WebKit etc.

Android RunTime

Core libraries

Linux Kernel

Display driver, camera driver, audio drivers, power management,
WiFi driver, USB driver, Flash memory driver, etc.

Linux Kernel : Lowest level of software in Android Platform

- Provides generic operating system services
 - Security
 - Memory management
 - Process management
 - File and Network I/O
 - Allows device drivers plug in (eg. display, camera, audio, WiFi, USB drivers)
- Provides also the following which are critical for mobile platform:
 - Power management
 - Low memory killer
 - Interprocess communication (share data and services)

Libraries

- Native Libraries (often written in C/C++)
- System C library
 - eg. Libc
- Surface Manager
 - Display management
- Media Framework
 - audio and video
- WebKit
 - Browser engine
- OpenGL
 - Graphics
- SQLite
 - Relational database engine

Android Runtime

- Supporting writing and running android applications
- Core java libraries
 - Basic java classes – java.* javax.*
 - App lifecycle – Android.*
 - Internet/web services – org.*
 - Unit testing – junit.*
- Android formerly used Dalvik virtual machine but Dalvik is now obsolete

More on Dalvik Virtual Machine (Dalvik is obsolete)

- Designed for resource-constrained environments
 - Slower CPU
 - Less RAM
 - Limited battery life
- Dalvik is the one that executes the app
- Not the standard java virtual machine
- App written in java → compiled to multiple java bytecode files → a tool called DX converts java bytecode files to a single dex bytecode file (classes.dex) → dex file packaged with other resources and installed on device
- When launching the app, Dalvik executes DEX bytecode file

Application Framework

- keeps reusable and commonly used software
- 10 components:
 - Package manager
 - Keeps track of app packages on device
 - Allows packages to communicate with each other
 - Window manager
 - manages system notification bar, main application window, subwindows (eg. menu, dialogue)

- View System
 - Provides user interface elements
- Resource Manager
 - Manages non-compiled resources (eg. Strings, graphics, layout files)
- Activity Manager
 - Manages app lifecycle and navigation stack
- Content Provider
 - Essentially databases that allow applications to store and share structured information

- Location Manager
 - Provides location and movement information
- Notification Manager
 - Place notification icons in the status bar when important events occur
- Telephony Manager
- XMPP (Extensible Messaging and Presence Protocol) Service
 - For instant messaging and online presence detection
 - Google Talk uses XMPP

Application Layer

- Standard apps include:
 - Home screen
 - Contacts
 - Phone
 - Browser
 - Email
 - etc.
- Note that none of these apps is hard-coded into the Android system. You can substitute your own or 3rd party app for any of these standard apps.

Android project

Build

Gradle

Byte Code

External
Resources eg.
Images,
strings etc.

Manifest

APK (specially formatted zip file)

Jar Signer

Sign

Install on Device using ADB

Main Building Blocks of Android

Main Building Blocks in Android App

- All building blocks live in Application Context
- All 4 building blocks are implemented as Java classes:
 - Activities (this is the only one that has GUI)
 - Services
 - Content Providers
 - Broadcast Receivers
- Activities and Services extend the Context class. Therefore, they can be used directly to access the Context.

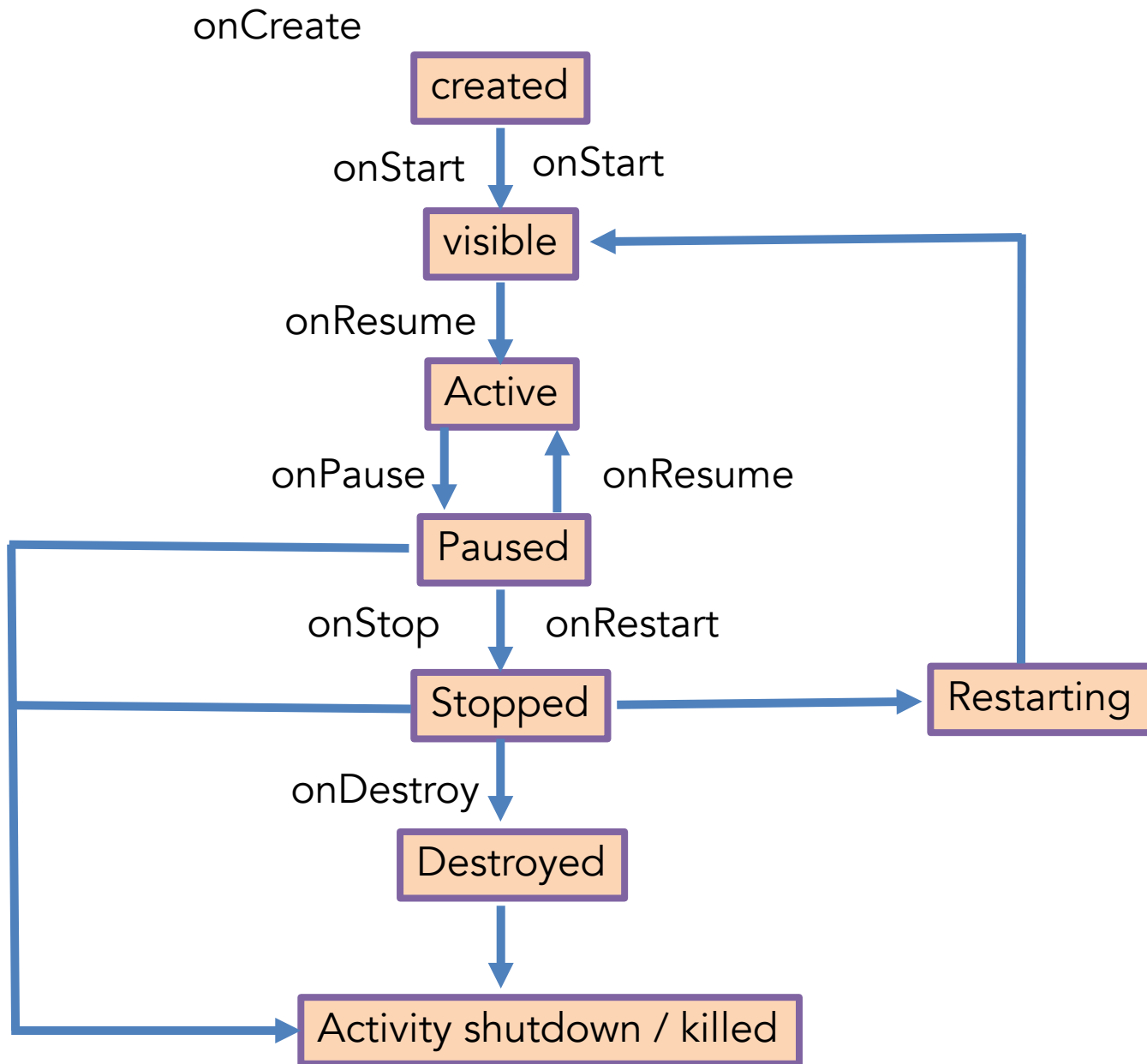
Activity

- An activity is a single screen
- Provides user interface
- Activity should be modular – should focus on doing one thing
- An application usually comprises multiple activities
 - Navigation among activities is done in several ways
 - Tasks
 - A set of related activities
 - Most tasks start at the home screen
 - Tasks backstack
 - When an activity is launched, it gets pushed on top of the backstack
 - When activity is destroyed, it is popped off the backstack
 - Suspending and resuming activities

- Activity life cycle:
 - create a new Linux process
 - allocating memory for all the UI objects
 - inflating all the objects from XML layouts
 - setting up the whole screen
 - activity life cycle is managed via Activity Manager
- Activity Manager is responsible for creating, destroying, and managing activities
- User can change life cycle, Android can change life cycle too (eg. Low in memory or other resources, Android can kill activities that are currently suspended)

Android announces activity lifecycle state changes to activity by calling specific activity methods

- Eg. onCreate, onStart, onResume, onPause, onStop, onDestroy



Note: after onStart, it is visible but user cannot interact yet. After onResume, then user can interact

- starting state: when an activity does not exist in memory, it is in a starting state.
- The transition from starting state to running state is very computationally intensive and affects battery life. This is why we don't automatically destroy activities in case user needs them later.
- running state: The activity in a running state is the one currently on the screen and interacting with the user. There is only one running activity at any given time

- The running activity is the one that has priority in terms of getting memory and resources
- paused state: when an activity is not in focus (i.e. not interacting with the user) but still visible on the screen, we say it is in a paused state (imagine a dialog box scenario). All activities go through a paused state before being stopped. Paused activities still have high priority in terms of getting memory and other resources

- stopped state: when an activity is not visible, but still in memory, we say it is in a stopped state.
- restarting a stopped activity is much cheaper than starting an activity from scratch
- destroyed state: a destroyed activity is no longer in memory

Note: if you need to save your activity data, don't do it in `onStop` or `onDestroy` as they may not be called at all

- Eg. `onStop` may not be called if Android killed the application process when it has low memory. So don't save persistence data in `onStop`. Should do that in `onPause` instead. Same thing for `onDestroy`. `onDestroy` may not be called at all when Android kills the application due to, for example, low memory

Activities – in the Java code...

- extends Activity means subclass of Activity
- Entry point is onCreate()
- onCreate() has to call super.onCreate();
- super.onCreate(savedInstanceState) is to create based on what was previously saved when the last time this activity was running.
- for returning result to calling activity
 - RESULTCODE (an int)
 - RESULT_CANCELED (Eg. When user choose to end abnormally)
 - RESULT_OK
 - RESULT_FIRST_USER (after this one, can add custom result code)

Summary of Activity Life Cycle

Starting State :

- Activity about to occupy memory
- Activity about to go through a set of callback methods

Computationally Intensive
Eats up Battery

Running State:

- Activity “in Focus”
- Only 1 running activity at any one time
- Enjoys priority in using memory and resources

Paused State :

- Activity visible on screen but not in focus
- eg. paused by dialog boxes
- All activities must go through paused state before being stopped
- Still enjoys high priority in using memory and other resources

Stopped State:

- Activity not visible but still in memory
- Could be brought back to running again
- Restarting stopped activity much cheaper than starting an activity from scratch
- Can be removed from memory

Destroyed State :

- no longer in memory
- not necessarily must pass through stopped state before destroy
- can be destroyed directly from paused state

not Visible

Visible

starting

running

Expensive

stopped

paused

destroyed

- onPause is guaranteed executed before killed, but that is pre-Honeycomb era. For post-Honeycomb, onStop is guaranteed to be the last to be executed.
- "Active" -> onSaveInstanceState -> "onPause" -> "Terminated"
- "Terminated" -> onCreate -> "onRestoreInstanceState"
(note: this only happens when you re-create)

Fragments

- Fragments have their own life cycles but their life cycle are also tied to the life cycle of the activity that contains them
- Fragment can be statically bound to the hosting activity, or they can be dynamically added
- Fragments Life Cycle states
 - Resumed – fragment is visible
 - Paused – hosting activity is visible but not in foreground
 - Stopped – fragment is not visible
- Fragment Life Cycle callback methods (called in the following order):
 - `onAttach()` – fragment is first attached to hosting activity
 - `onCreate()` – `fragment.onCreate`, not the one for activity – initialize the fragment, but don't set up user interface like in `activity.onCreate`
 - `onCreateView()` – sets up and returns the user interface
 - `onActivityCreated()` – when fragment has been installed

Fragment life cycle is dependent on hosting activity life cycle

Example:

When hosting activity is about to become visible, activity `onStart()` will cause fragment's `onStart()` method to be called as well.

When activity `onResume()`, the fragment's `onResume` will be called as well.

When activity is visible but not in focus (i.e. pause), the fragment's `onPause()` method will also be called.

When activity `onStop()`, fragment's `onStop()` will also be called.

When activity gets destroyed, the fragment's view gets detached from the activity, and this will cause the fragment's `onDestroyView()` to be called. When fragment is no longer in use, it receives the `onDestroy()` method.

When fragment is no longer attached to its hosting activity, `onDetach()` will be called.

Services

- services run in the background and don't have any user interface components
- services can perform the same actions as activities, but without user interface
- services have a simpler life cycle than activities:
 - onCreate(), onStart(), onDestroy()
- services are used for repetitive or long running operations that do not require user interface eg. internet download, data processing, updating content providers, checking for new data

- service runs in a **background**, but that does not mean it runs on a separate thread (although you can choose to run on a separate thread using something called IntentServices). **Services and activities run on the same main application thread**, called the **UI thread**, so don't use services for heavy duty tasks like playing MP3 music. If must play music, then use a thread different from the main UI thread.
- services can be **accessed by other applications** than the one that started them.
- Services support interaction with remote processes (eg. Share data)
- services are different from AsyncTasks: it is designed for longer events and can **run even when application Activity is not opened**.

Content Providers

- content providers are interfaces for storing and sharing data across applications
- Android runs each application in its own sandbox (i.e. data in one application is totally isolated from other applications in the system).
- note that small amount of data can be passed between applications via intents.

- Content Provider API adheres to the CRUD principle (Create, Read, Update, Delete are the 4 basic functions of persistent storage)
- Uses database-style interface but also handles interprocess communication
- relatively simple interfaces:
 - insert()
 - update()
 - delete()
 - query()

Examples of Content Providers

- Contacts Provider – exposes all user data to various applications
- Settings Provider – exposes system settings to various applications
- Media Store – storing and sharing photos and music across various applications

Broadcast Receivers

- publish/subscribe mechanism
- the receiver is a dormant code that gets activated once an event to which it is subscribed happens

Application Context

where Activities, Services, Content Providers, Broadcast Receivers live together.....

- it's the environment and process within which all its components are running
- allows applications to share data and resources between various building blocks
- Application context gets created as soon as the application gets started
- can get a reference to Application Context by `Context.getApplicationContext()` or `Activity.getApplication()`
- Activities and services are subclasses of context

- Application context is attached to application's life-cycle
- Activity context is attached to the Activity's life-cycle (can be destroyed when Activity is destroyed).
 - Eg. new Intent(**this**,)



Activity Context

Android User Interface

All graphical elements (i.e. widgets and layouts) has View class as their base class.

Layouts:

- Layouts extend the ViewGroup class, which in turn extends the View class
- eg. relative layout, linear layout

Widgets:

- Widgets directly extend the View class.
- Widgets can be individual or groups of UI elements.
- eg. buttons, text fields, labels.

View Class and View Events

- View class is basic building block
- View occupies a rectangular space on screen
- Responsible for drawing themselves and for handling events
- Some predefined views: Button, ToggleButton, CheckBox, RatingBar, AutoCompleteTextView

- View Event Sources

- User interaction
 - Touch
 - Keyboard/trackball/D-pad
- Android (system control)
 - Lifecycle changes



- Handling View Events

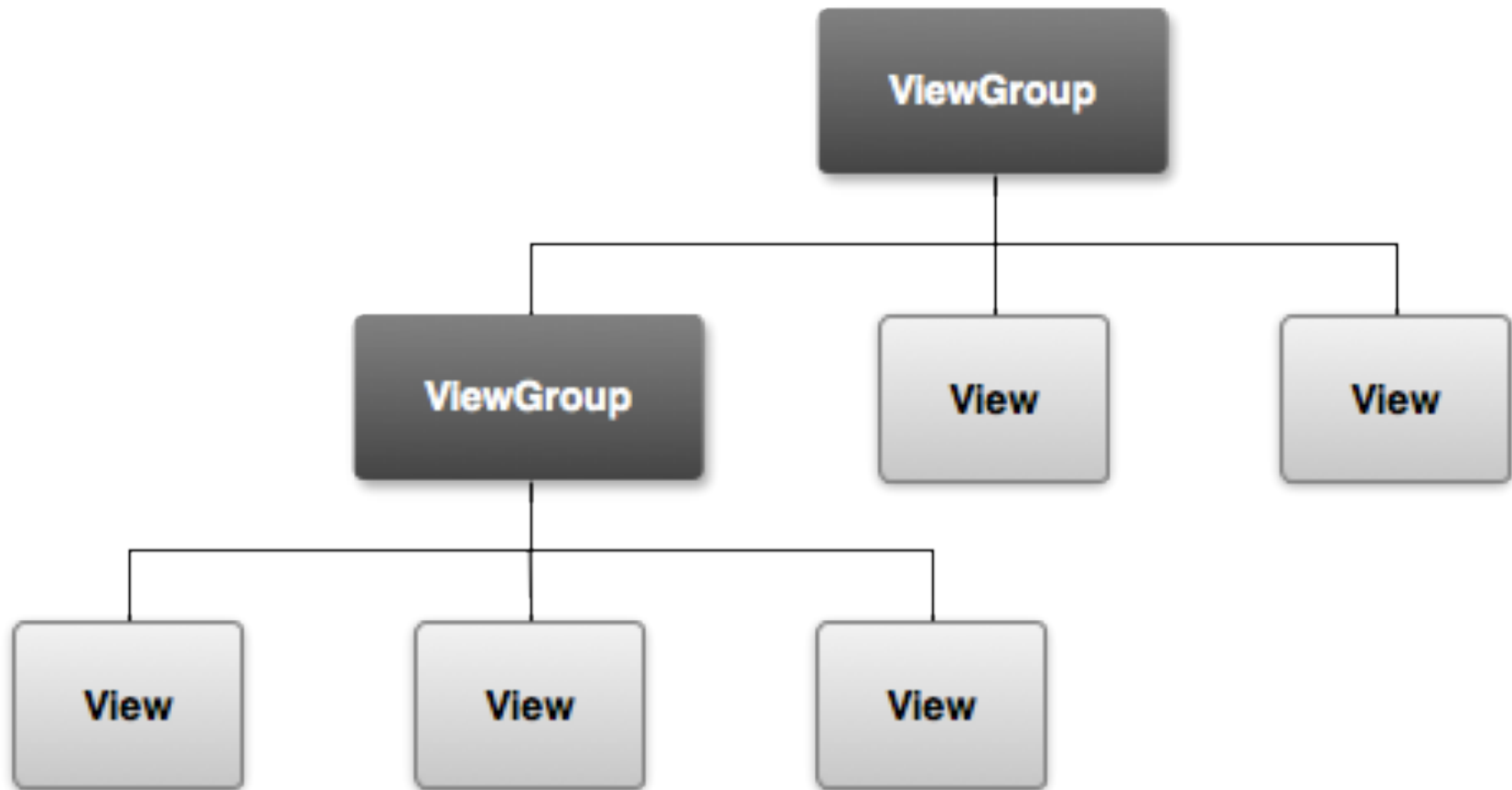
- OnClickListener.onClick()
- OnLongClickListener.onLongClick()
- onFocusChangeListener.onFocusChange()
- OnKeyListener.onKey() -- hardware key press
- onMeasure() – measures the size of view and its children.
- onLayout() – view must assign a size and position to all its children
- onDraw() – view should render its content
- onKeyUp(), onKeyDown(), onWindowVisibilityChanged()

Displaying views

- Views are organized in a tree
 - First pass : Measure (get dimensions of each view)
 - `onMeasure()`
 - Second pass : Layout – position each view
 - `onLayout()`
 - Third pass: Draw – draw each view
 - `onDraw()`
- Custom view subclass can override the above methods

ViewGroup

- ViewGroup is invisible view that contains other views
- ViewGroup is the base class for view containers and layouts
- Some predefined ViewGroups:
 - RadioGroup
 - Group of checkboxes (radio buttons)
 - TimePicker
 - DatePicker
 - WebView
 - MapView
 - Gallery (horizontally scrolling list. Items managed by SpinnerAdapter)
 - Note that Gallery was deprecated since API level 16.
 - Spinner (this is an AdapterView. Items managed by SpinnerAdapter)
- Adapters and AdapterViews
 - AdapterViews are views whose children and data are managed by an Adapter
 - Adapter manages the data and provides data when AdapterView asked for them
 - ListView is an AdapterView. Items managed by ListAdapter.



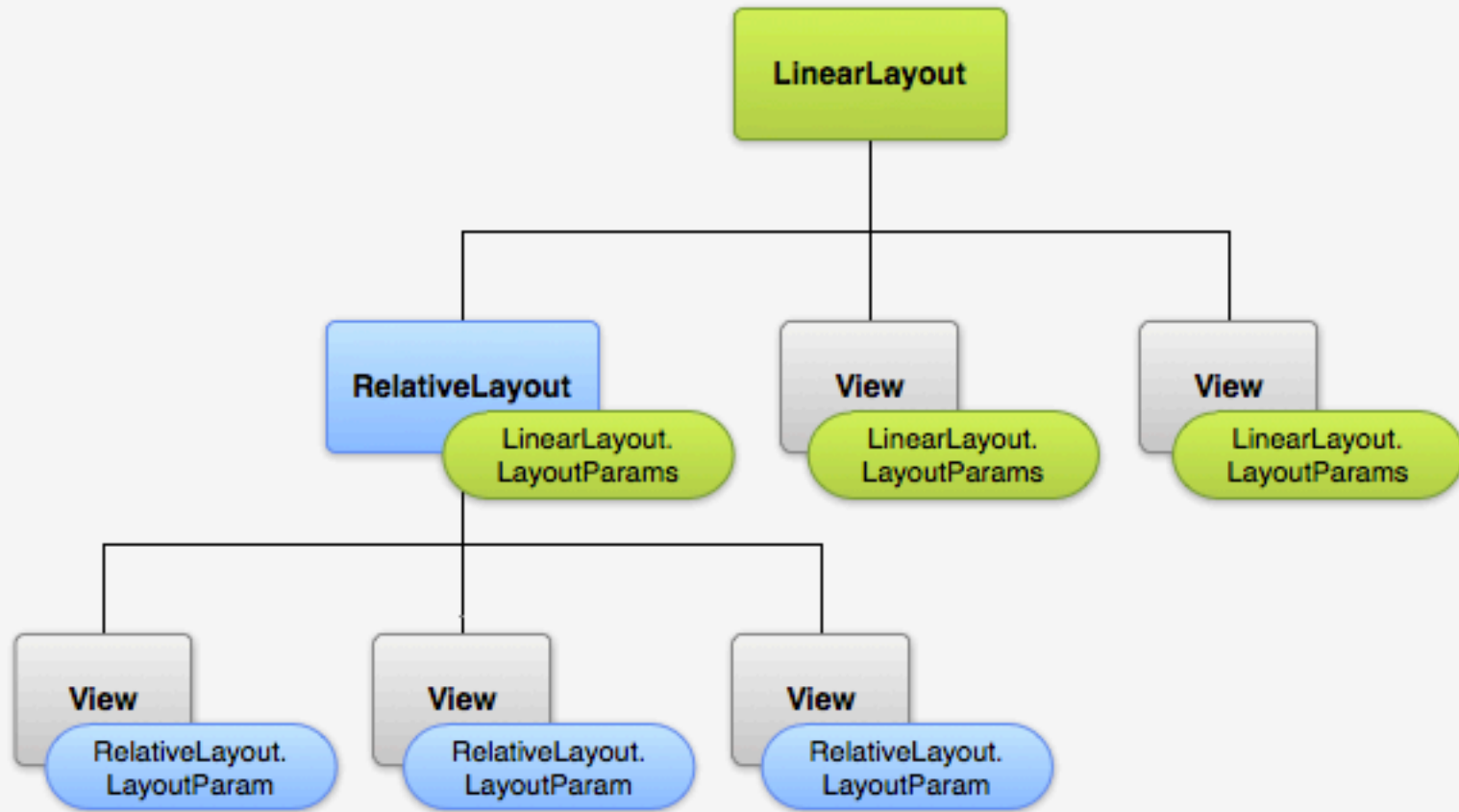


Figure 1. Visualization of a view hierarchy with layout parameters associated with each view.

UI Layout Files

- Layout extends View Group
- At compilation time, resources are used to generate the R.java class.
- Java code uses the R class to access the resources

Common types of Layout

- LinearLayout
 - lay out its children next to each other (horizontally or vertically)
 - the property `layout_orientation` can be set to vertical or horizontal
- RelativeLayout
 - each element's position is relative to other elements
- AbsoluteLayout
 - each element's position is specified by (x,y)
- TableLayout
 - displays elements in the form of table (rows and columns)
- GridView
 - Arrange its children in 2D scrollable grid
 - Eg. Use `ImageAdapter` which extends from `BaseAdapter`

Note:

Although we can nest one or more layouts within another layout, we should not nest too deeply. We should try to keep the layout hierarchy as shallow as possible.

This is because layout draws faster if it has fewer nested layouts.

A wide hierarchy is better than a deep hierarchy.

Layouts can be defined:

- dynamically (i.e. in your Java code)
- statically (in xml)
- any combination of Java code or XML
- XML will eventually be inflated to run as Java codes.

Note: Running on Android Device:

Connect your Android device to the computer using a USB cable.

Enable USB debugging on your device:

On most devices running Android 3.2 or older, you can find the option under **Settings > Applications > Development**.

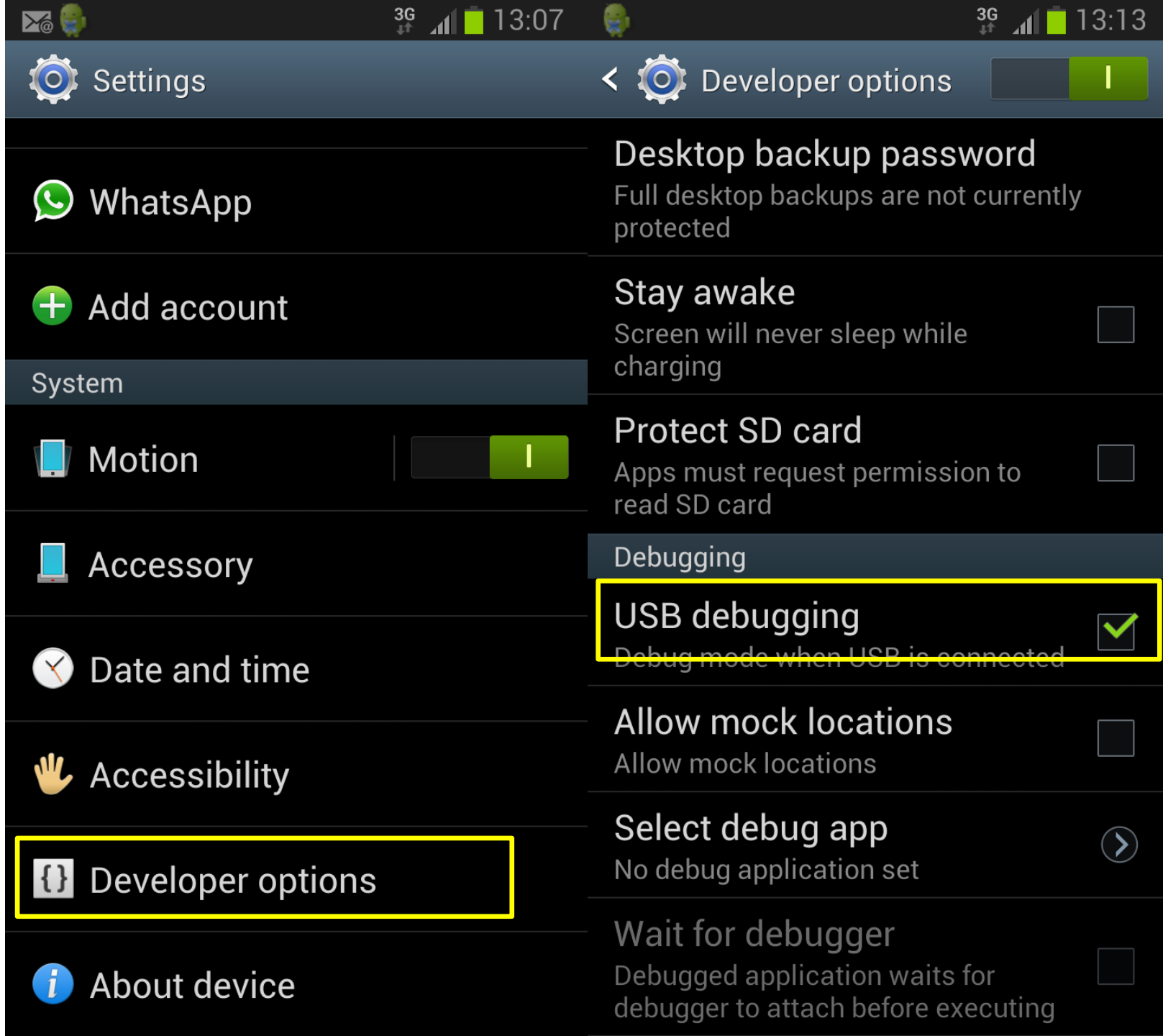
On Android 4.0 and newer, it's in **Settings > Developer options**.

Note that on Android 4.2 and newer, **Developer options** is hidden by default. To make it available, go to **Settings > About phone**, then tap Build number seven times. Return to the previous screen to find the **Developer options**.

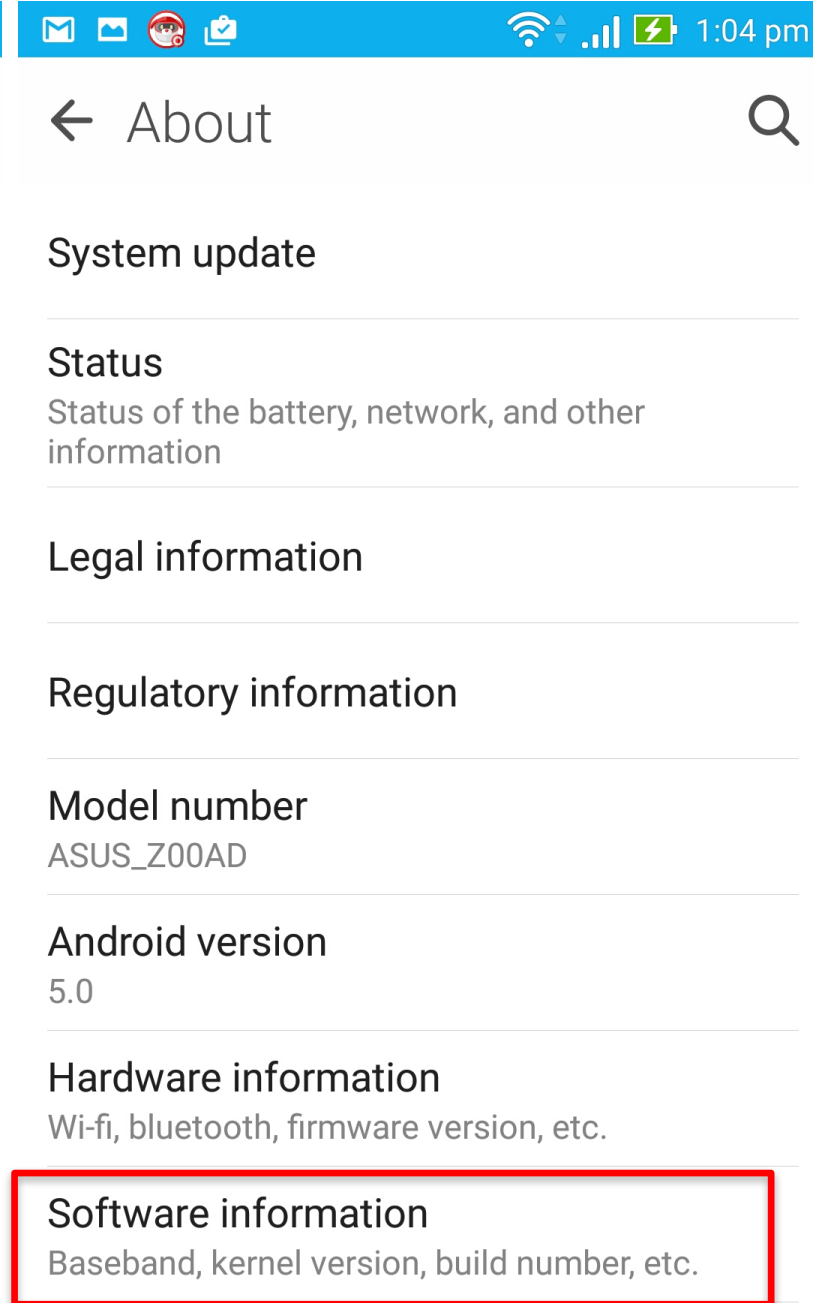
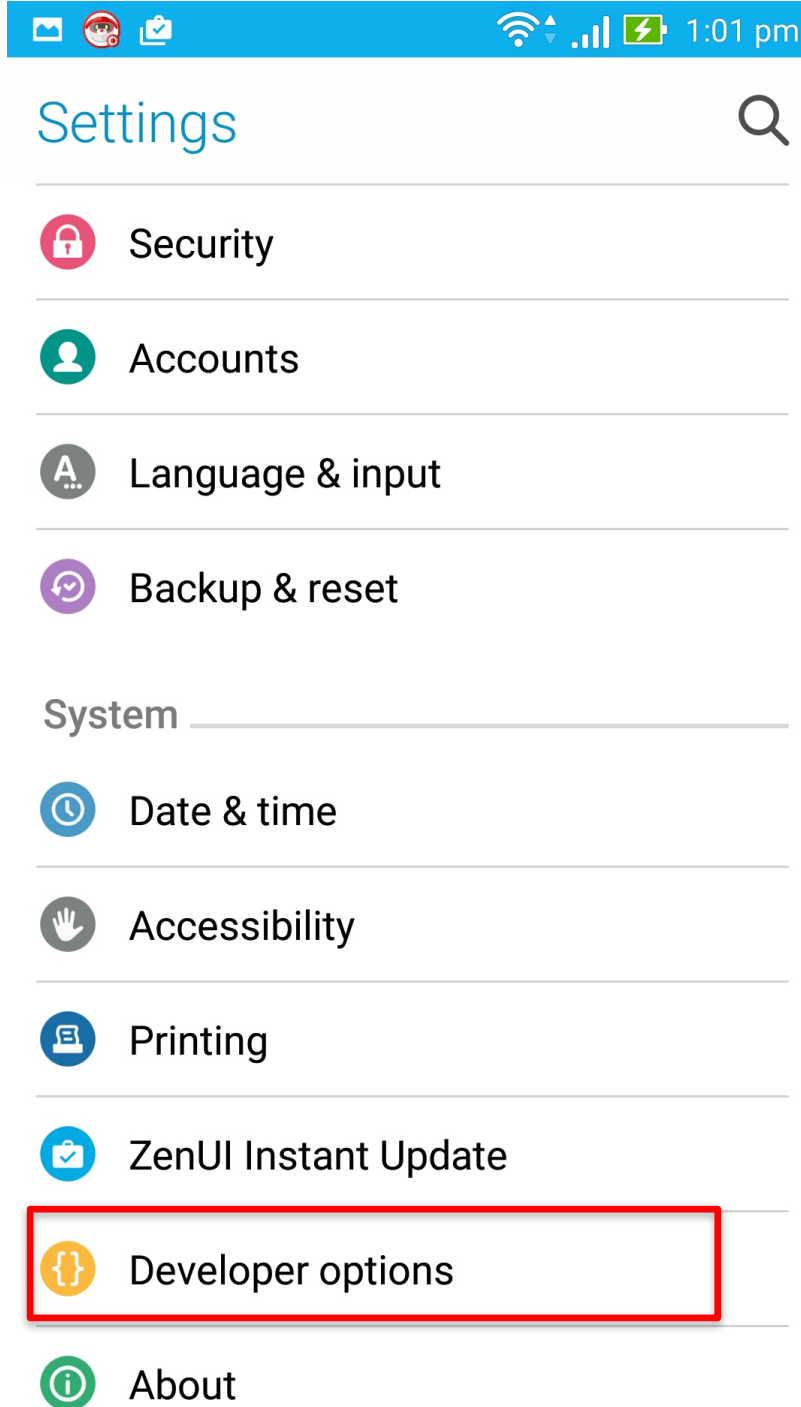
Once you get into **Developer options**, check the box for **USB Debugging**.

Note: If you are using Samsung devices, and if Android Studio cannot detect your device, try to install Kies from <http://www.samsung.com/kies>. This fix the problem for Windows.

Example:
Samsung



Example:
Asus
Zenfone-2



← Software information 🔍

Baseband version

1520_5.0.27.3_0528,1517_7.0.13.0_0424

Kernel version

3.10.20-x86_64_moor-g4a549b3
jenkins@localserver #1
Fri May 29 00:32:56 CST 2015

Build number

MR7-LRX21V.WW-
ASUS_Z00A-2.18.40.12_20150529_9543_user
044030427_201501060109
0010000100001

Tap 7 times

Settings



- Security
- Accounts
- Language & input
- Backup & reset

System

- Date & time
- Accessibility
- Printing
- ZenUI Instant Update
- Developer options

- About

Developer options



On



Take bug report

Desktop backup password

Desktop full backups aren't currently protected

Stay awake

Screen will never sleep while charging



Enable Bluetooth HCI snoop log

Capture all bluetooth HCI packets in a file



Process Stats

Geeky stats about running processes

Debugging

USB debugging

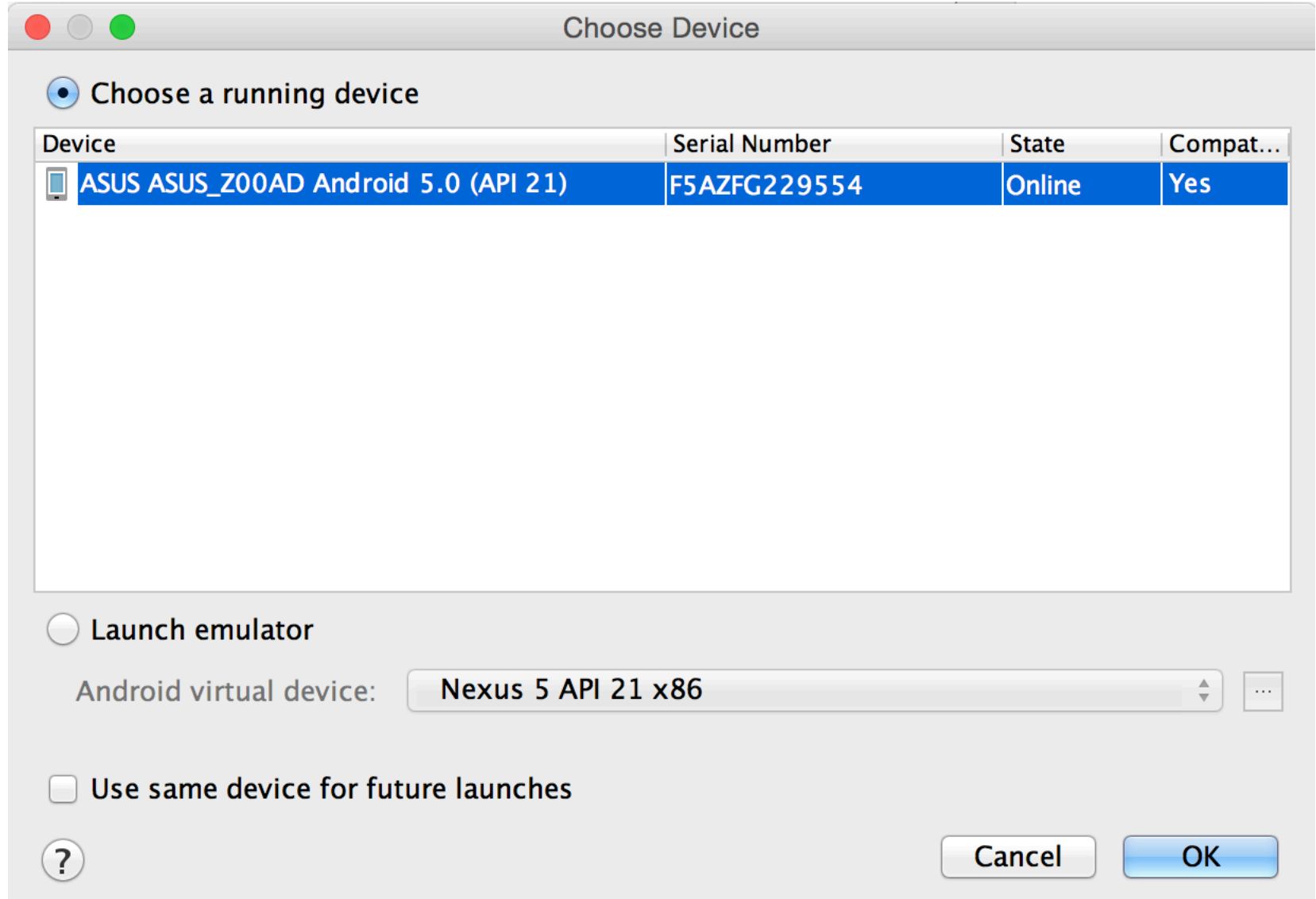
Debug mode when USB is connected



Revoke USB debugging authorizations

Bug report shortcut

In Android Studio,
Run -> Run 'app'



My First Android Project



Hello world!