Applications

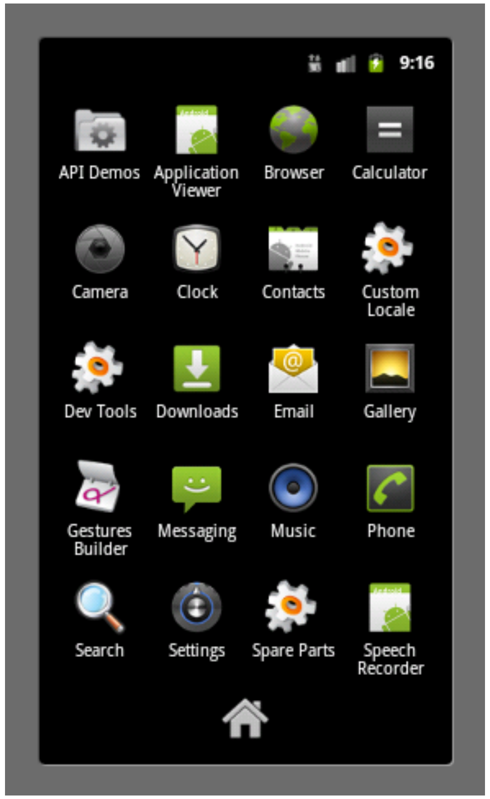
Marco Ronchetti
Università degli Studi di Trento

An Android *application* typically consists of one or more related, loosely bound activities for the user to interact with.

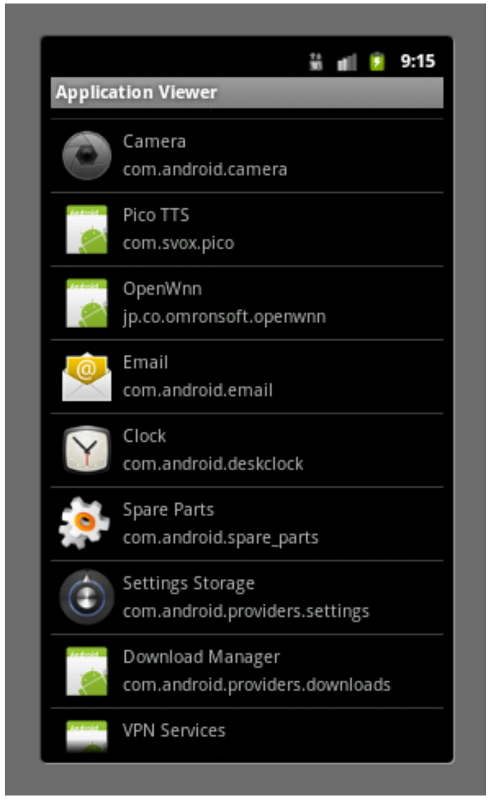
Android has an application launcher available at the Home screen, typically in a sliding drawer which displays applications as icons, which the user can pick to start an application.

Android ships with a rich set of applications that may include email, calendar, browser, maps, text messaging, contacts, camera, dialer, music player, settings and others.

Application Launcher



You can replace it


9:15

Application Viewer

Camera
com.android.camera

Pico TTS
com.svox.pico

OpenWnn
jp.co.omronsoft.openwnn

Email
com.android.email

Clock
com.android.deskclock

Spare Parts
com.android.spare_parts

Settings Storage
com.android.providers.settings

Download Manager
com.android.providers.downloads

VPN Services

See e.g. http://xjaphx.wordpress.com/2011/06/12/create-application-launcher-as-a-list/

Application package

An application is a single APK (application package) file. An APK file roughly has three main components.

- Dalvik executable: all your Java source code compiled down to Dalvik executable. This is the code that runs your application.
- Resources: everything that is not code (images, audio/video clips, XML files describing layouts, language packs, and so on.
- Native libraries: e.g. C/C++ libraries.

Signing applications

Android applications must be signed before they can be installed on a device

To distribute your application commercially, you'll want to sign it with your own key.

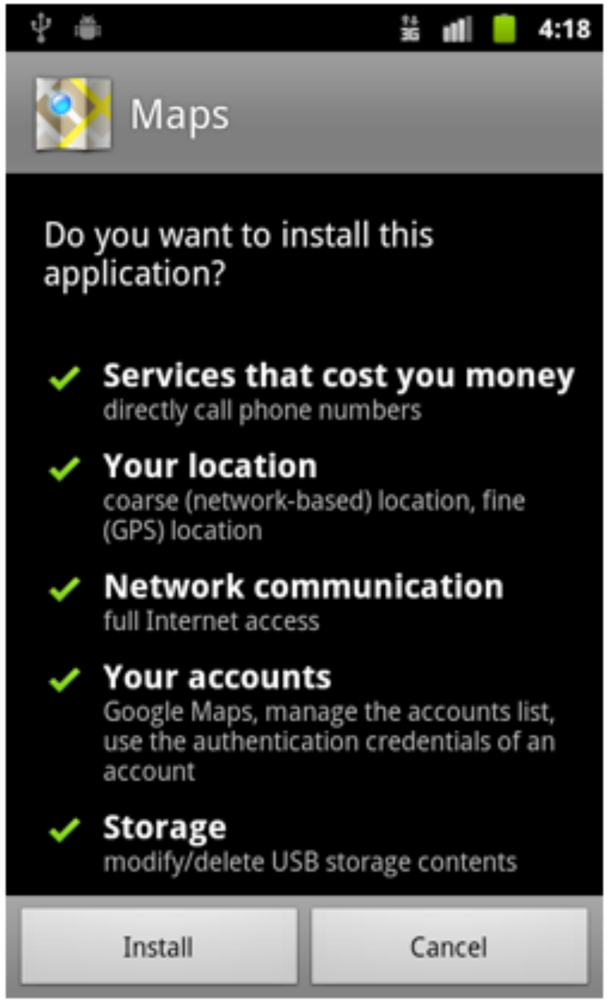
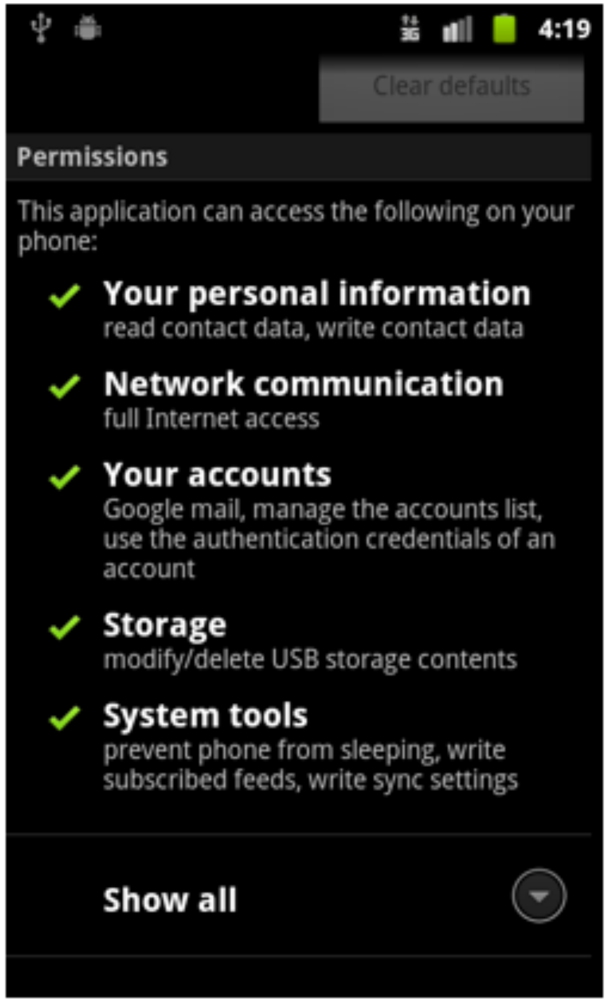
Distributing applications

Unlike the iPhone, on Android, there can be many different Android stores or markets. Each one can have its own set of policies with respect to what is allowed, how the revenue is split, and so on.

The biggest market currently is Android Market run by Google

Applications can also be distributed via the web. When you download an APK file from a website by using the Browser, the application represented by the APK file automatically gets installed on your phone.

Granting and checking permissions

Permissions at Application Install -- Google Maps	Permissions of an Installed Application -- gMail
	

Permissions at Application Install -- Google Maps

Do you want to install this application?

- ✓ **Services that cost you money** directly call phone numbers
- ✓ **Your location** coarse (network-based) location, fine (GPS) location
- ✓ **Network communication** full Internet access
- ✓ **Your accounts** Google Maps, manage the accounts list, use the authentication credentials of an account
- ✓ **Storage** modify/delete USB storage contents

Permissions of an Installed Application -- gMail

Permissions

This application can access the following on your phone:

- ✓ **Your personal information** read contact data, write contact data
- ✓ **Network communication** full Internet access
- ✓ **Your accounts** Google mail, manage the accounts list, use the authentication credentials of an account
- ✓ **Storage** modify/delete USB storage contents
- ✓ **System tools** prevent phone from sleeping, write subscribed feeds, write sync settings

Security

Android has a security framework.

http://source.android.com/tech/security/index.html

The Android File System can be encrypted.

Encryption on Android uses the dm-crypt layer in the Linux kernel.

Security model

Android OS is a multi-user Linux in which each application is a different user.

By default, the system assigns each application a unique Linux user ID (the ID is unknown to the application). The system sets permissions for all the files in an application so that only the user ID assigned to that application can access them.

Each process has its own virtual machine (VM), so an application's code runs in isolation from other applications.

By default, every application runs in its own Linux process.

Principle of least privilege

Principle of least privilege (or "need to know")

Each application, by default, has access only to the components that it requires to do its work and no more.

Data sharing

It's possible to arrange for two applications to share the same Linux user ID, in which case they are able to access each other's files.

Applications with the same user ID can also arrange to run in the same Linux process and share the same VM (the applications must also be signed with the same certificate).

An application can request permission to access device data such as the user's contacts, SMS messages, the mountable storage (SD card), camera, Bluetooth, and more. All application permissions must be granted by the user at install time.

Process lifetime

Android

- starts the process when any of the application's components need to be executed,
- shuts down the process when
  - it's no longer needed
  - the system must recover memory for other applications.

The fundamental components

Marco Ronchetti
Università degli Studi di Trento

The fundamental components

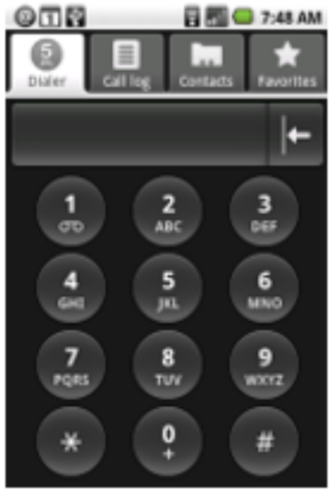
- Activity
  - an application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map.
- Fragment (since 3.0)
  - a behavior or a portion of user interface in an Activity
- View
  - equivalent to Swing Component
- Service
  - an application component that can perform long-running operations in the background and does not provide a user interface
- Intent
  - a passive data structure holding an abstract description of an operation to be performed. It activates an activity or a service. It can also be (as often in the case of broadcasts) a description of something that has happened and is being announced.
- Broadcast receiver
  - component that enables an application to receive intents that are broadcast by the system or by other applications.
- Content Provider
  - component that manages access to a structured set of data.
- AndroidManifest.xml
- Android Virtual Devices

Activity

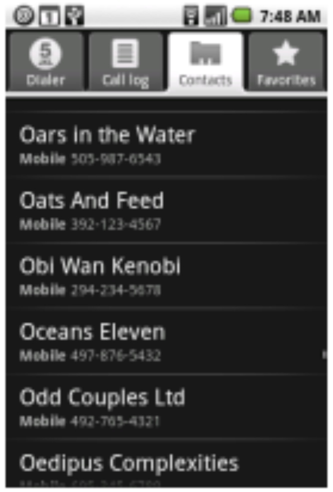
An application component that provides a screen with which users can interact in order to do something, such as dial the phone, take a photo, send an email, or view a map.

Each activity is given a window in which to draw its user interface. The window typically fills the screen, but may be smaller than the screen and float on top of other windows, or be embedded in another activity (activityGroup).

Activities of the dialer application

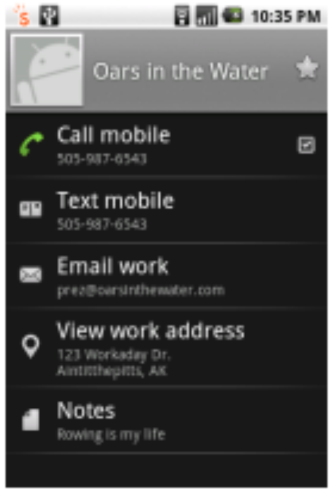


Dialer



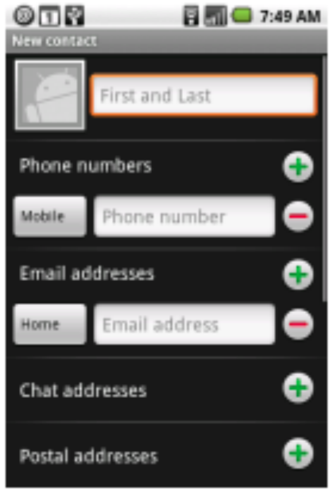
Oars in the Water
Mobile 505-987-6543
Oats And Feed
Mobile 392-123-4567
Obi Wan Kenobi
Mobile 294-234-5678
Oceans Eleven
Mobile 497-876-5432
Odd Couples Ltd
Mobile 492-765-4321
Oedipus Complexities

Contacts



Call mobile
505-987-6543
Text mobile
505-987-6543
Email work
prez@oarsinthewater.com
View work address
123 Workaday Dr.
Aintitthepitts, AK
Notes
Rowing is my life

View Contact



New Contact

Activity

Typically, one activity in an application is specified as the "main" activity, which is presented to the user when launching the application for the first time.

Each activity can then start another activity in order to perform different actions.

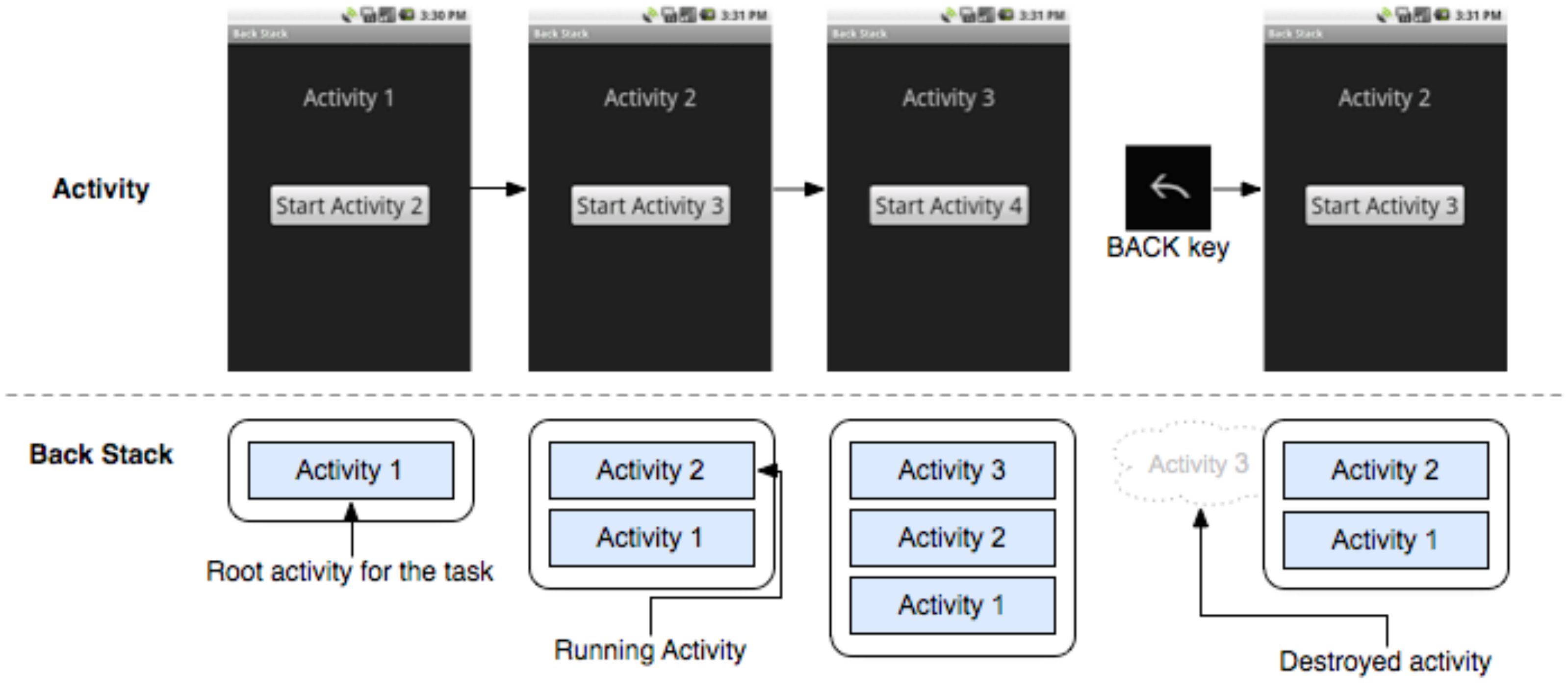
Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a LIFO stack (the "activity stack" or "back stack").

When a new activity starts, it is pushed onto the back stack and takes user focus.

When the user is done with the current activity and presses the *Back* button, it is popped from the stack (and destroyed) and the previous activity resumes.

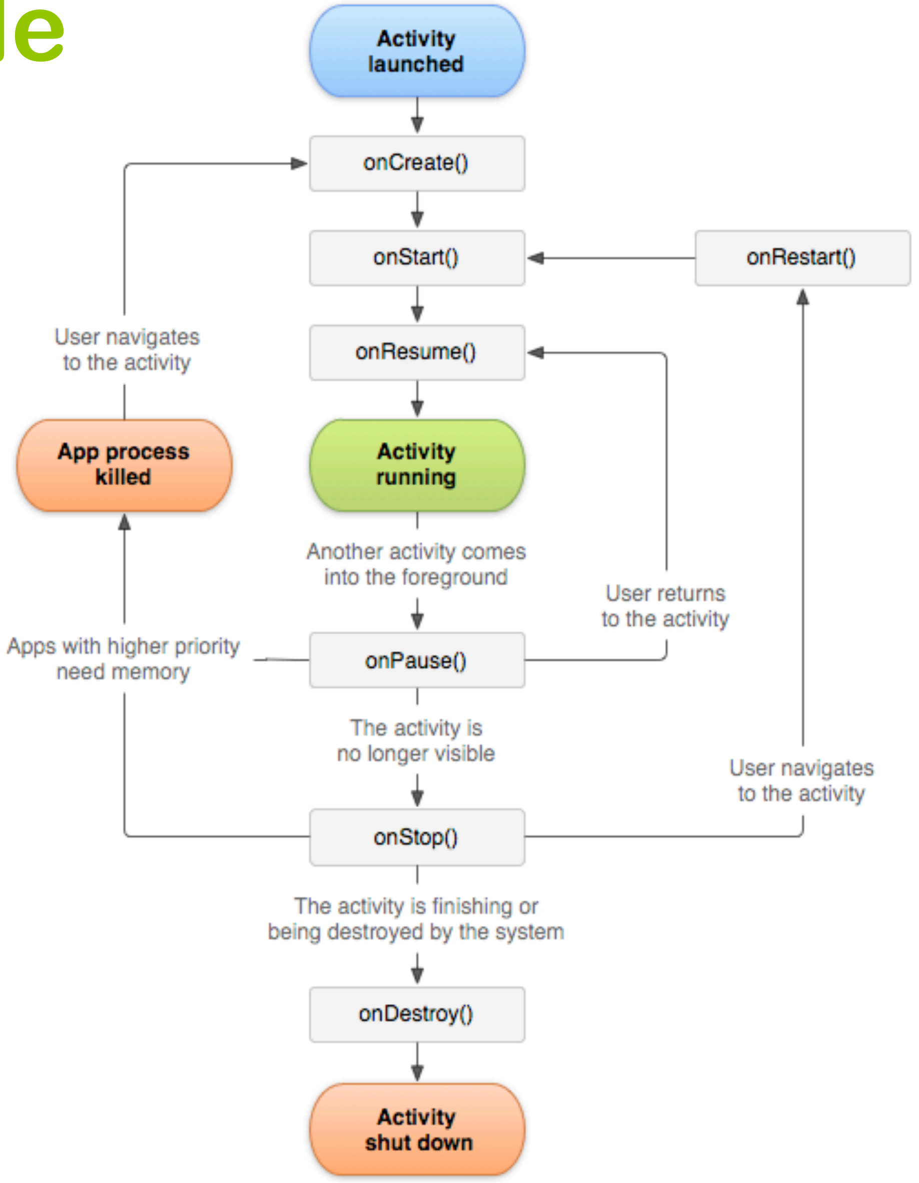
The activity stack

It's similar to the function stack in ordinary programming, with some difference



Activity lifecycle

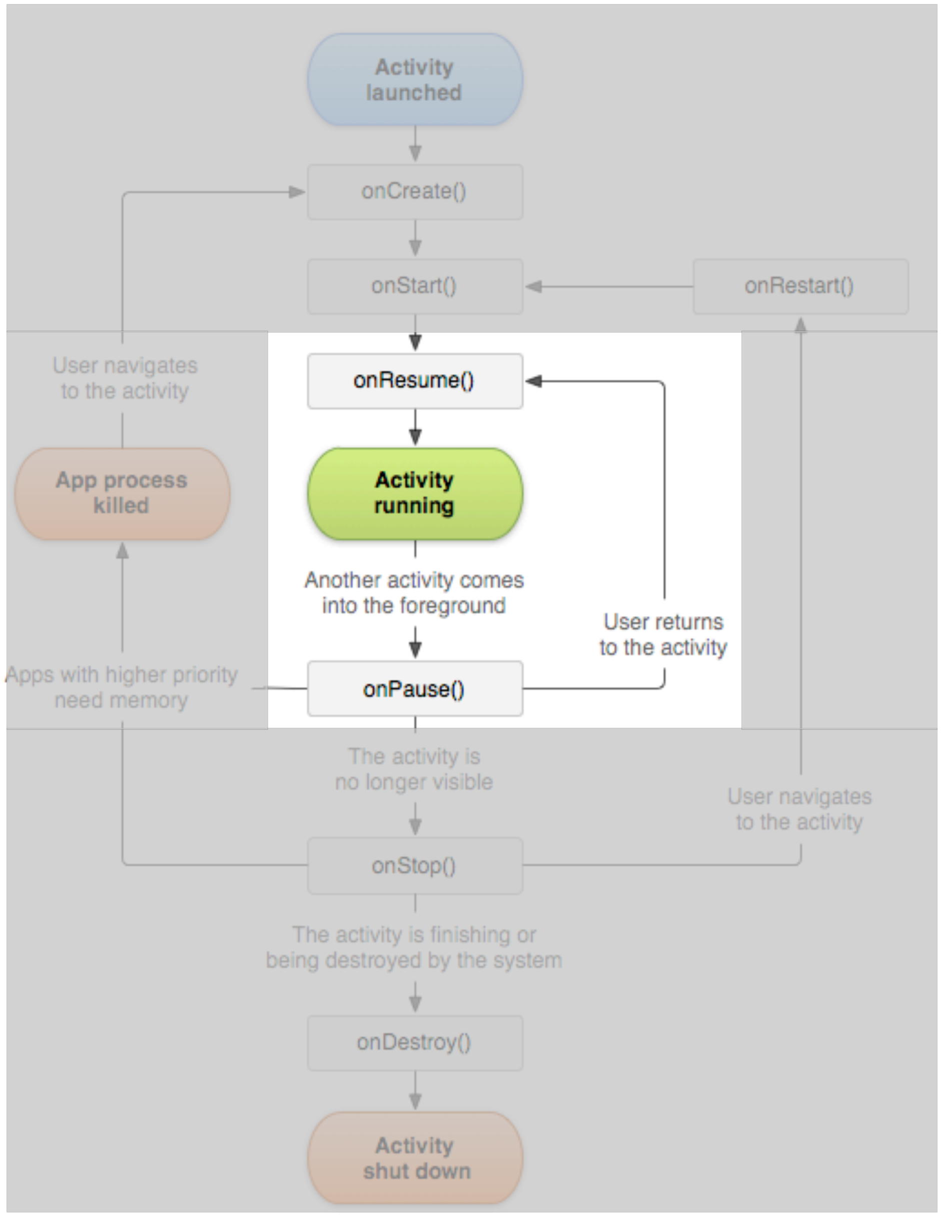
States (colored),
and
Callbacks (gray)



Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Activity lifecycle

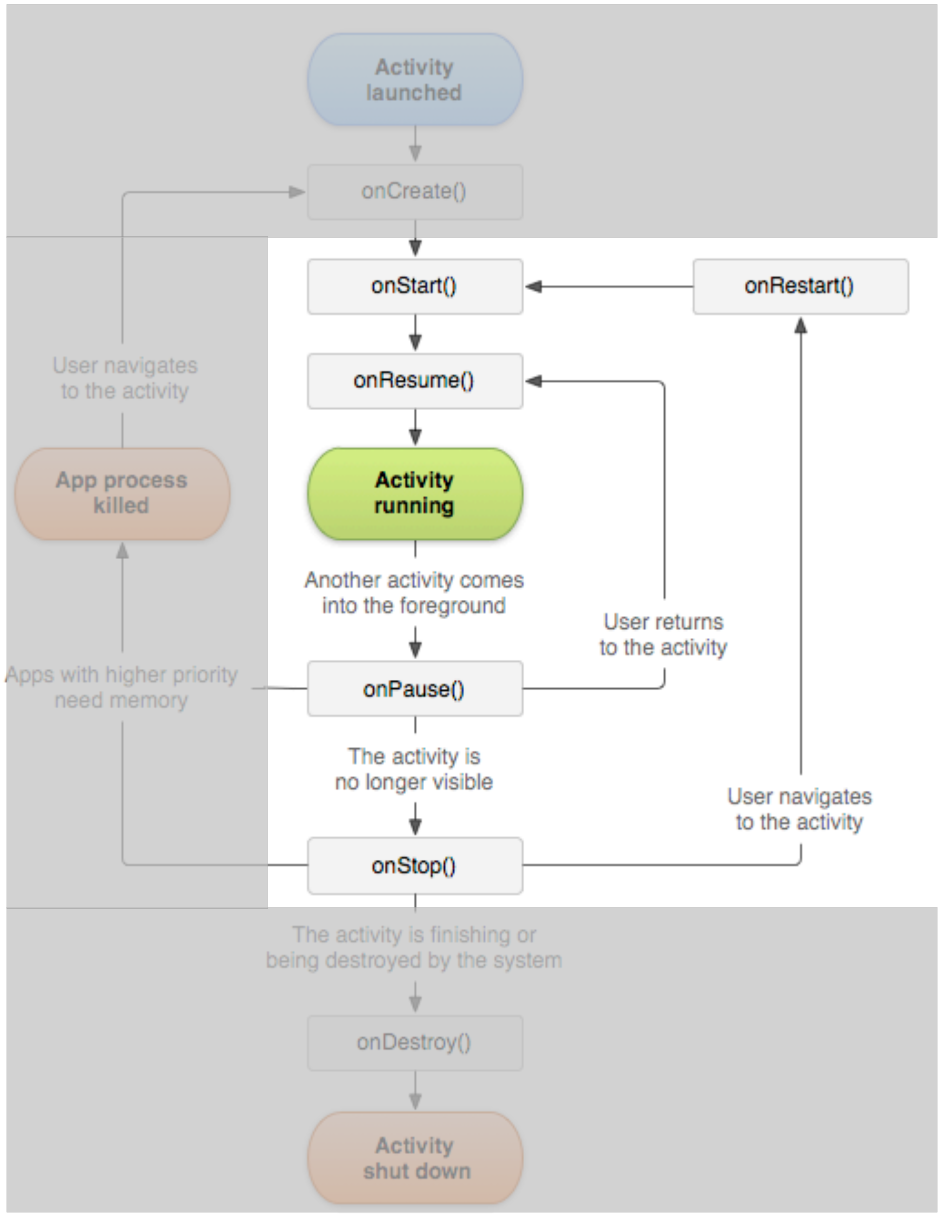
The FOREGROUND lifetime



Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Activity lifecycle

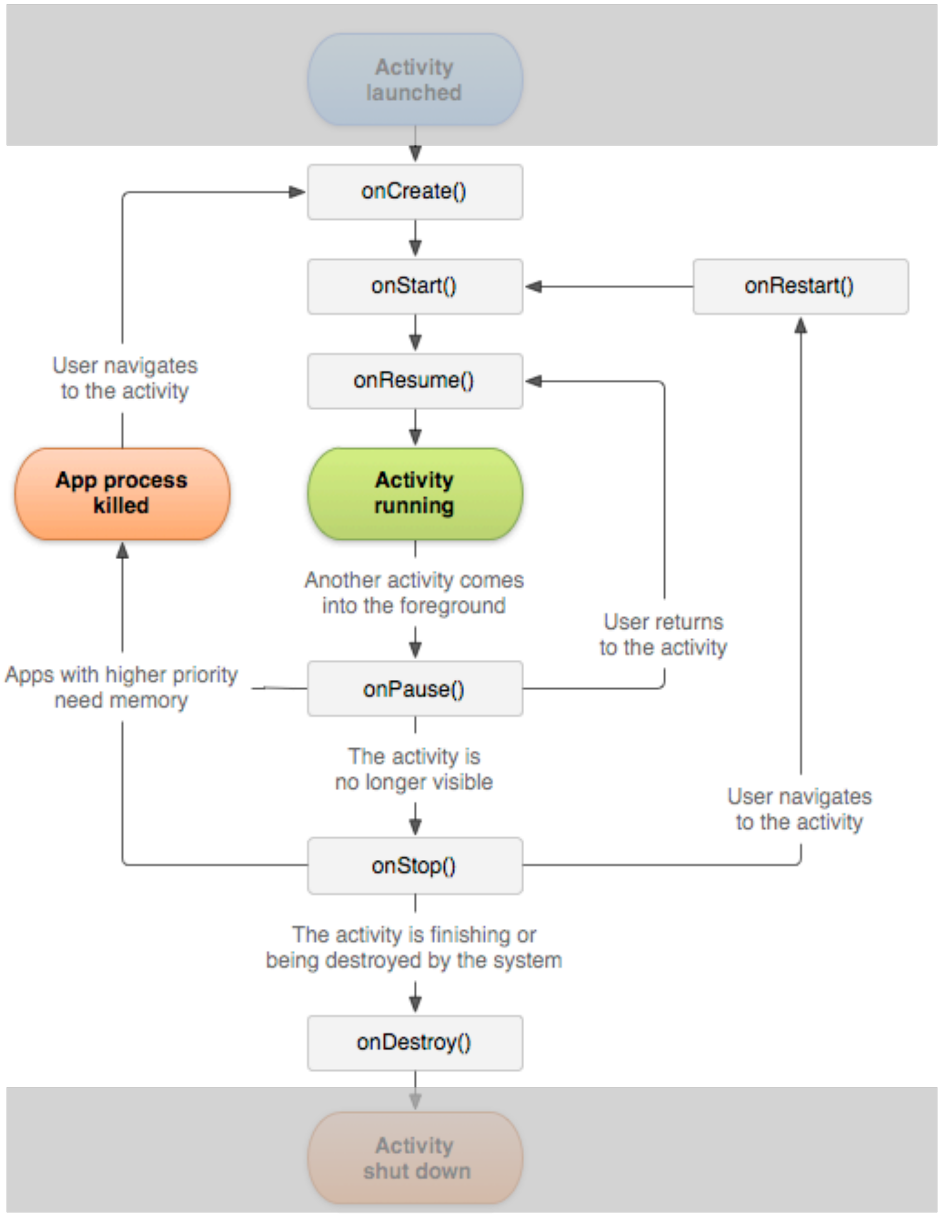
The VISIBLE lifetime



Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Activity lifecycle

The ENTIRE lifetime



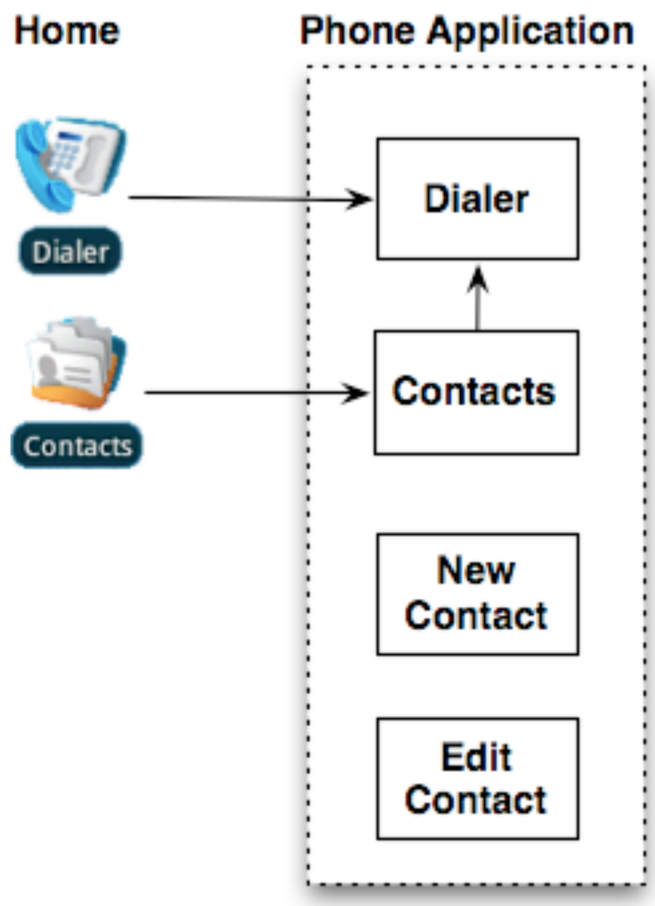
Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

When stopped, your activity should release any large objects, such as network or database connections. When the activity resumes, you can reacquire the necessary resources and resume actions that were interrupted. These state transitions are all part of the activity lifecycle.

When you create an application, you can assemble it from activities that you create and from activities you re-use from other applications. These activities are bound at runtime, so that newly installed applications can take advantage of already installed activities

Multiple entry-point for an app

An application can have multiple entry points



Fragment

A Fragment represents a behavior or a portion of user interface in an Activity.

You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.

You can think of a fragment as a modular section of an activity, which has its own lifecycle, receives its own input events, and which you can add or remove while the activity is running (sort of like a "sub activity" that you can reuse in different activities).

View

the basic building block for user interface components, similar to the Java AWT Component.

A View occupies a rectangular area on the screen and is responsible for drawing and event handling. View is the base class for *widgets*, which are used to create interactive UI components (buttons, text fields, etc.)

Service

A Service is an application component that can perform long-running operations in the background and does not provide a user interface.

Another application component can start a service and it will continue to run in the background even if the user switches to another application.

Additionally, a component can bind to a service to interact with it and even perform interprocess communication (IPC). For example, a service might handle network transactions, play music, perform file I/O, or interact with a content provider, all from the background.

Service

A service can essentially take two forms:

Started

A service is "started" when an application component (such as an activity) starts it by calling startService(). Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.

For example, it might download or upload a file over the network. When the operation is done, the service should stop itself.

Bound

A service is "bound" when an application component binds to it by calling bindService(). A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).

A bound service runs only as long as another application component is bound to it.

Multiple components can bind to the service at once, but when all of them unbind, the service is destroyed.

dedicated to user interaction with your activities.

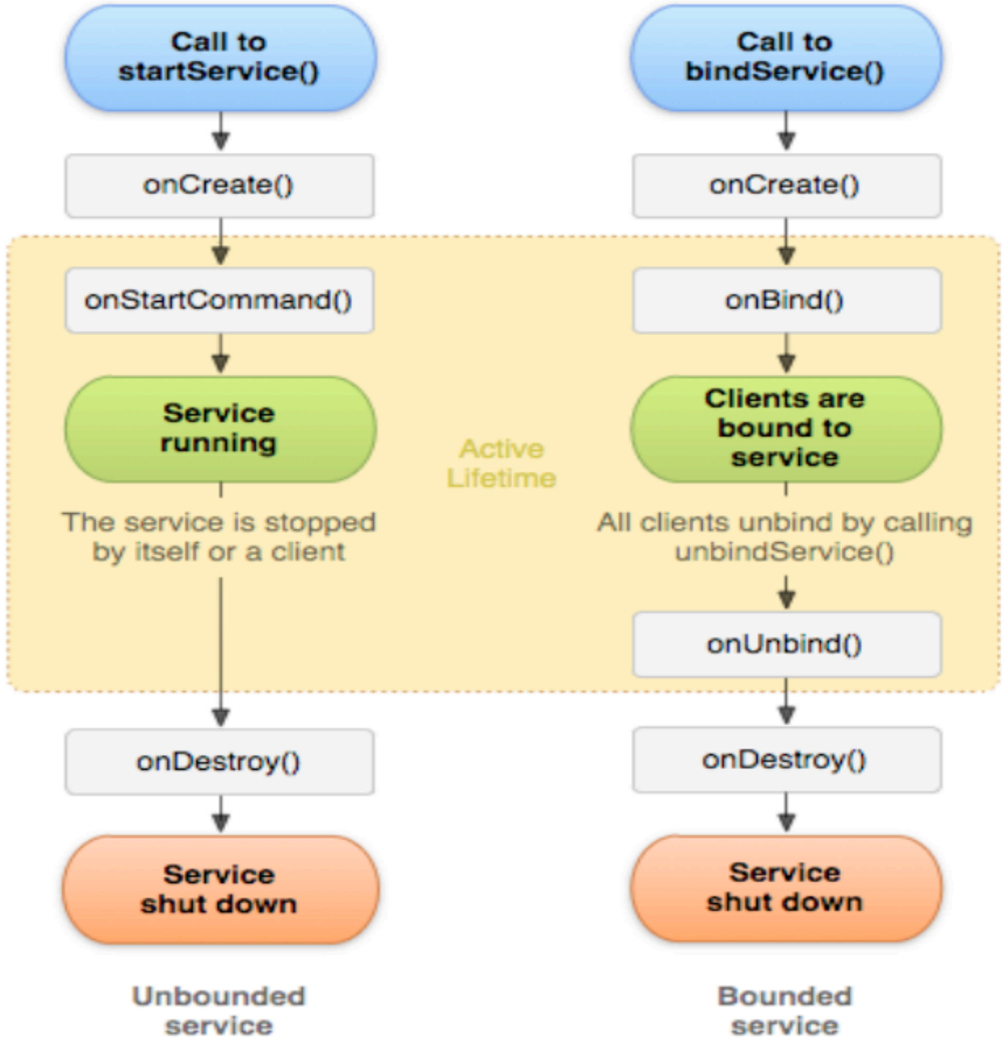
Service

You can declare the service as private, in the manifest file, and block access from other applications.

Caution: A service runs in the main thread of its hosting process—the service does not create its own thread and does not run in a separate process (unless you specify otherwise). This means that, if your service is going to do any CPU intensive work or blocking operations (such as MP3 playback or networking), you should create a new thread within the service to do that work.

By using a separate thread, you will reduce the risk of Application Not Responding (ANR) errors and the application's main thread can remain dedicated to user interaction with your activities.

Service lifecycle



Broadcast receiver

A broadcast receiver is a component that responds to system-wide broadcast announcements.

Many broadcasts originate from the system—for example, a broadcast announcing that the screen has turned off, the battery is low, or a picture was captured. Applications can also initiate broadcasts—for example, to let other applications know that some data has been downloaded to the device and is available for them to use.

Although broadcast receivers don't display a user interface, they may create a status bar notification to alert the user when a broadcast event occurs.

Content Provider

Content providers manage access to a structured set of data. They encapsulate the data, and provide mechanisms for defining data security. Content providers are the standard interface that connects data in one process with code running in another process.

Android itself includes content providers that manage data such as audio, video, images, and personal contact information.

You can see some of them listed in the reference documentation for the android.provider package.

Intents

Three of the core components of an application — activities, services, and broadcast receivers — are activated through messages, called intents.

Intent messaging is a facility for late run-time binding between components in the same or different applications. The intent itself, an Intent object, is a passive data structure holding an abstract description of an operation to be performed — or, often in the case of broadcasts, a description of something that has happened and is being announced.

In each case, the Android system finds the appropriate activity, service, or set of broadcast receivers to respond to the intent, instantiating them if necessary.

There is no overlap within these messaging systems:

- Broadcast intents are delivered only to broadcast receivers, never to activities or services.
- An intent passed to startActivity() is delivered only to an activity, never to a service or broadcast receiver, and so on.

Android Java packages

Marco Ronchetti
Università degli Studi di Trento

Basic components

android.app

- implements the Application model for Android

android.content

- implements the concept of Content providers

android content.pm

- Package manager: permissions, installed {packages, services, provider, applications, components}

android.content.res

- Access to resources

android.provider

- Contacts, MediaStore, Browser, Setting

GUI basics

android.view
- Menu, View, ViewGroup + listeners

android.view.animation

android.view.inputmethod
- Input methods framework

android.widget
- UI controls derived from View (Button, Checkbox…)

android.gesture
- create, recognize, load and save gestures

Graphics

android.graphics

- low level graphics tools such as canvases, color filters, points, and rectangles that let you handle drawing to the screen directly.
- Bitmap, Canvas, Camera (3D transformation, not the camera!) , Color, Matrix, Movie, Paint, Path, Rasterizer, Shader, SweepGradient, Typeface

android.graphics.drawable

- variety of visual elements that are intended for display only, such as bitmaps and gradients

android.graphics.drawable.shapes

android.opengl

- opengl-related utility classes, not the opengl!

javax.microedition.khronos.opengles
javax.microedition.khronos.egl
javax.microedition.khronos.nio

android.renderscript

- low-level, high performance means of carrying out mathematical calculations and 3D graphics rendering

Text rendering

android.text

- classes used to render or track text and text spans on the screen

android.text.method

- Classes that monitor or modify keypad input.

android.text.style

- Text styling mechanisms

android.service.textservice

- Provides classes that allow you to create spell checkers

android.view.textservice

- Use spelling checkers

Database, Web and location

android.database

- classes to explore data returned through a content provider.

android.datebase.sqlite

- the SQLite database management classes that an application would use to manage its own private database. Applications use these classes to manage private databases.

android.webkit

- tools for browsing the web.

android.location

- Address, Geocoder, Location, LocationManager, LocationProvider

com.google.android.maps

Network and telephony

android.net

- Socket-level network API - help with network access, beyond the normal java.net.* APIs.

android.net.wifi

android.bluetooth

android.nfc

- Near Field Communication (NFC) is a set of short-range wireless technologies, typically requiring a distance of 4cm or less to initiate a connection. NFC allows you to share small payloads of data between an NFC tag and an Android-powered device, or between two Android-powered devices.

android.telephony

- monitoring the basic phone information, plus utilities for manipulating phone number strings, SMS
- CellLocation, PhoneNumberUtils, TelephonyManager

android.telephony.gsm

- Obtain Cell location of GSM

android.telephony.cdma

- Obtain Cell location of CDMA - CDMA2000 is a family of 3G mobile technology standards

Media and speech

android.media

- manage various media interfaces in audio and video
- MediaPlayer, MediaRecorder, Ringtone, AudioManager, FaceDetector.

android.media.effect

- apply a variety of visual effects to images and videos

android.hardware

- support for hardware features, such as the camera and other sensors

android.drm

- Digital right management

android.mtp

- interact directly with connected cameras and other devices, using the PTP (Picture Transfer Protocol)

android.speech

- base class for recognition service implementations

android.speech.tts

- Text to Speech

General utilities

android.utils

- date/time manipulation, base64 encoders and decoders, string and number conversion methods, and XML utilities.

android.sax

- XML parsing

android.test

- A framework for writing Android test cases and suites

android.preference

- manage application preferences and implement the preferences UI. Using these ensures that all the preferences within each application are maintained in the same manner and the user experience is consistent with that of the system and other applications

android.os

- basic operating system services, message passing, and inter-process communication
- Binder (ipc), FileObserver (changes in files) Handler e Looper (for dealing with message threads), BatteryManager, PowerManager

Still useful java packages

java.lang (e subpackages)

java.math

java.net + javax.net

java.io

java.nio

java.sql+javax.sql

- (android.database preferable if possible)

java.util

Other still useful packages

javax.crypto

javax.security

javax.xml

org.w3c.dom

org.xml.sax

org.apache.http (e subpackages)

Screen properties

Marco Ronchetti
Università degli Studi di Trento

Screen related terms and concepts

Resolution The total number of physical pixels on a screen. When adding support for multiple screens, applications do not work directly with resolution; applications should be concerned only with screen size and density, as specified by the generalized size and density groups.

Screen size Actual physical size, measured as the screen's diagonal.

Screen density The quantity of pixels within a physical area of the screen; usually referred to as dpi (dots per inch).

Orientation The orientation of the screen from the user's point of view. This is either landscape or portrait, meaning that the screen's aspect ratio is either wide or tall, respectively. Not only do different devices operate in different orientations by default, but the orientation can change at runtime when the user rotates the device.

Screen Sizes and Densities

Android divides the range of actual screen sizes and densities into:

A set of four generalized **sizes**:
xlarge screens are at least 960dp x 720dp
large screens are at least 640dp x 480dp
normal screens are at least 470dp x 320dp
small screens are at least 426dp x 320dp

A set of generalized **densities**:
ldpi (low), *mdpi* (medium), *tdpi* (only Google TV, Nexus 7), *hdpi* (high), *xhdpi* (extra high) and *xxhdpi* (extra extra high)

Density-independent pixel

Density-independent pixel (dp) A virtual pixel unit that you should use when defining UI layout, to express layout dimensions or position in a density-independent way.

At runtime, the system transparently handles any scaling of the dp units, as necessary, based on the actual density of the screen in use.

You should always use dp units when defining your application's UI, to ensure proper display of your UI on screens with different densities.

Pixel densities

The density-independent pixel is equivalent to one physical pixel on a 160 dpi screen, which is the baseline density assumed by the system for a "medium" density screen.

ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	xxxhdpi
120	160	213	240	320	480	640

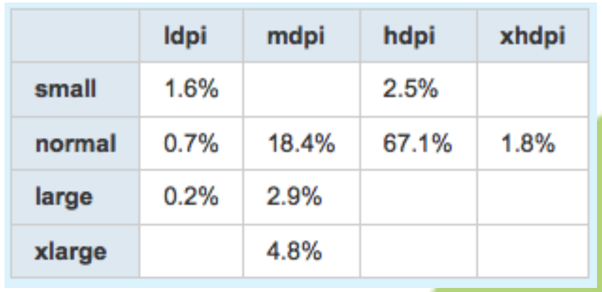
The ratio for asset scaling is:

ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	xxxhdpi
0.75	1	1.33	1.5	2	3	4

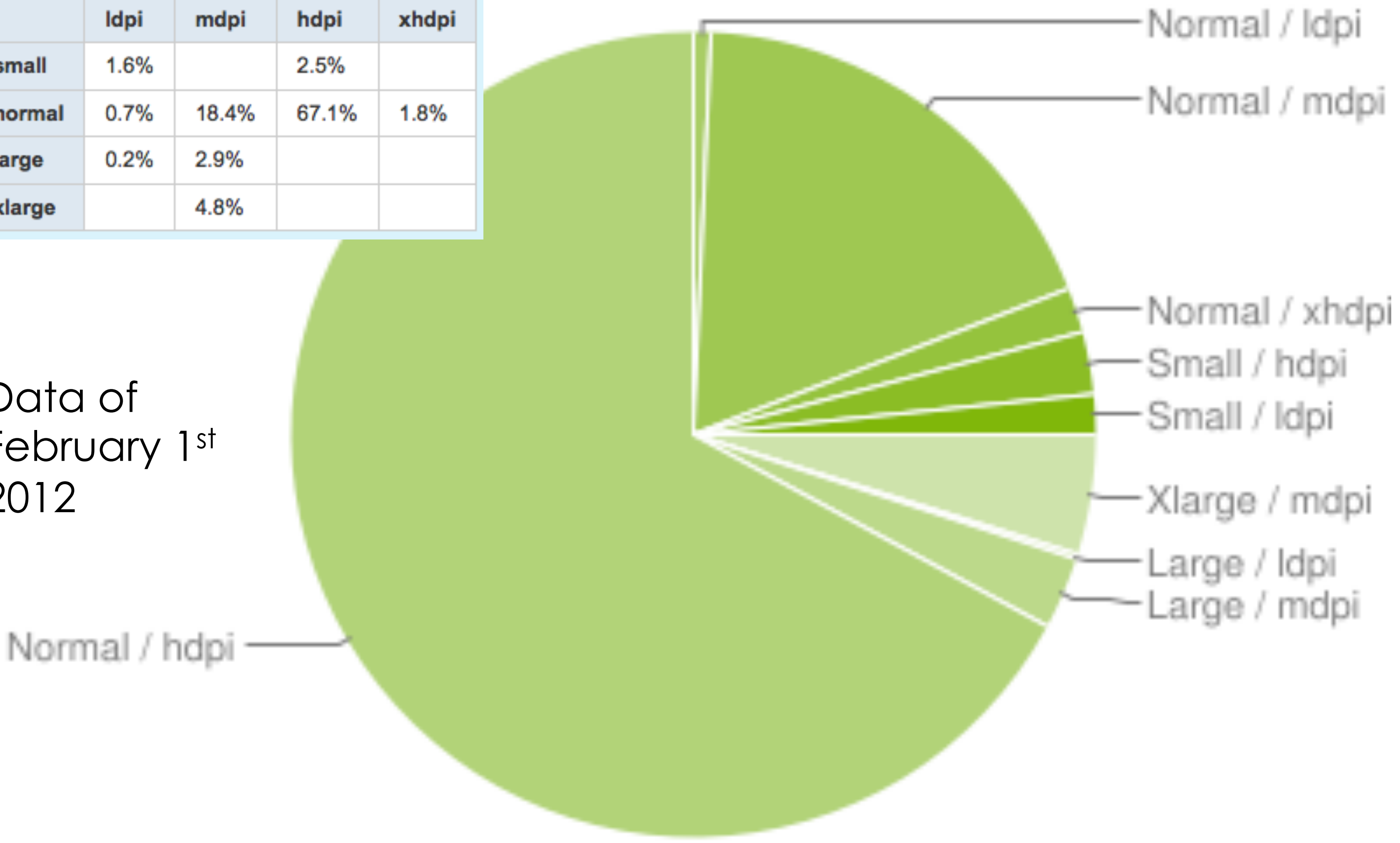
Example images

ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	xxxhdpi
36 x 36	48 x 48	64 x 64	72 x 72	96 x 96	144 x 144	192 x 192

Screen Sizes and Densities (2012)



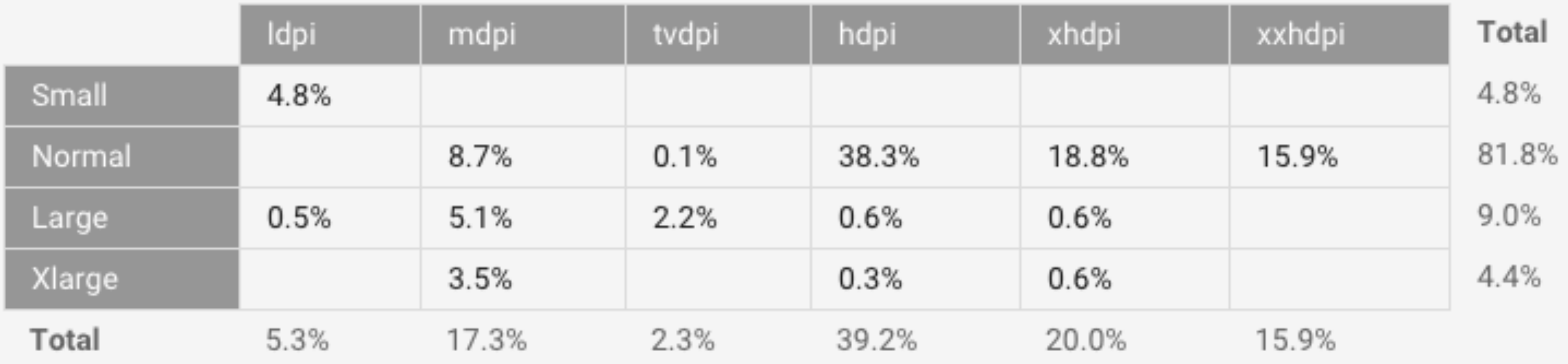
	ldpi	mdpi	hdpi	xhdpi
small	1.6%		2.5%	
normal	0.7%	18.4%	67.1%	1.8%
large	0.2%	2.9%		
xlarge		4.8%		



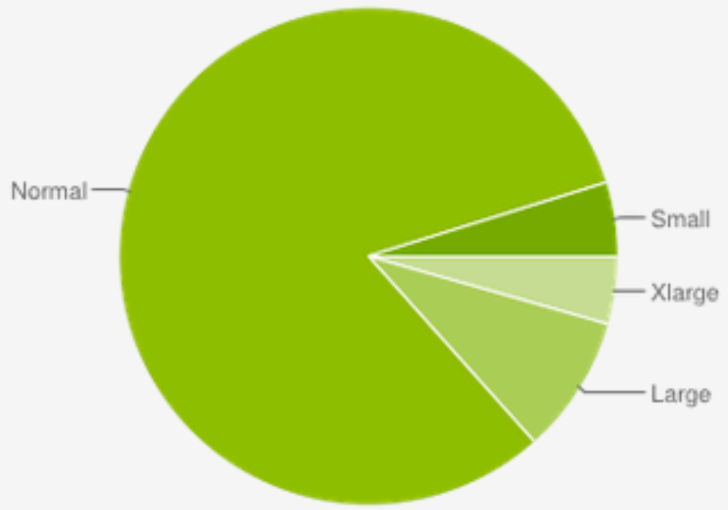
Data of February 1st 2012

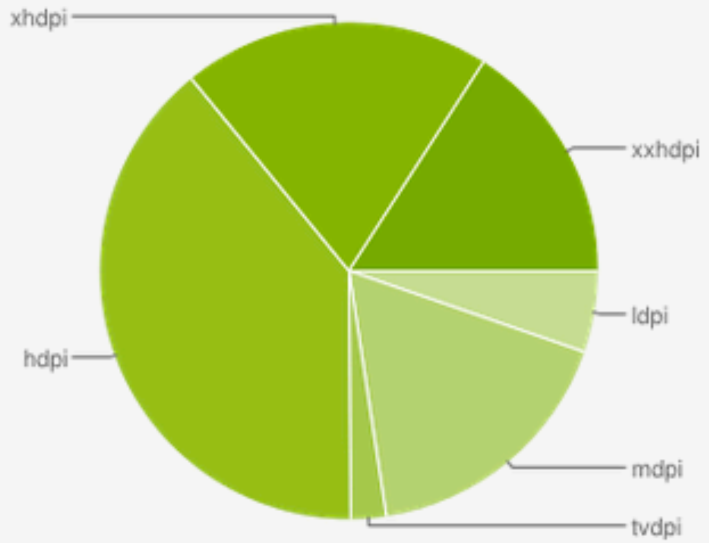
http://developer.android.com/resources/dashboard/screens.html

Screen Sizes and Densities (2015)



	ldpi	mdpi	tvdpi	hdpi	xhdpi	xxhdpi	Total
Small	4.8%						4.8%
Normal		8.7%	0.1%	38.3%	18.8%	15.9%	81.8%
Large	0.5%	5.1%	2.2%	0.6%	0.6%		9.0%
Xlarge		3.5%		0.3%	0.6%		4.4%
Total	5.3%	17.3%	2.3%	39.2%	20.0%	15.9%	





Data collected during a 7-day period ending on February 2, 2015.
Any screen configurations with less than 0.1% distribution are not shown.

Una lettura consigliata…

Marco Ronchetti
Università degli Studi di Trento

Android design

http://developer.android.com/design/index.html