Pokhara University

Bachelor Of Computer Application

Lecture Notes

In

Mobile Application Development Technology

1.Introduction to mobile OS

A **mobile phone** (also known as a **cellular phone**, **cell phone**, **hand phone**, or simply a **phone**) is a phone that can make and receive telephone calls over a radio link while moving around a wide geographic area. It does so by connecting to a cellular network provided by a mobile phone operator, allowing access to the public telephone network. By contrast, a cordless telephone is used only within the short range of a single, private base station.

The common components found on all phones are:

- A battery, providing the power source for the phone functions.
- An input mechanism to allow the user to interact with the phone. The most common input mechanism is a keypad, but touch screens are also found in most smartphones.
- A screen which echoes the user's typing, displays text messages, contacts and more.
- Basic mobile phone services to allow users to make calls and send text messages.
- All GSM phones use a SIM card to allow an account to be swapped among devices. Some CDMA devices also have a similar card called a R-UIM.

History of Mobile Phones:

This history focuses on communication devices which connect wirelessly to the public switched telephone network. The transmission of speech by radio has a long and varied history going back to Reginald Fessenden's invention and shore-to-ship demonstration of radio telephony. The first mobile telephones were barely portable compared to today's compact hand-held devices. Along with the process of developing more portable technology, drastic changes have taken place in the networking of wireless communication and the prevalence of its use.

The world's first mobile phone call was made on April 3, 1973, when Martin Cooper, a senior engineer at Motorola, called a rival telecommunications company and informed them he was speaking via a mobile phone.

SIM card

GSM feature phones require a small microchip called a Subscriber Identity Module or SIM card, to function. The SIM card is approximately the size of a small postage stamp and is usually placed underneath the battery in the rear of the unit.

Introduction to Mobile OSes

Operating System:

An **operating system** (**OS**) is system software that manages computer hardware and software resources and provides common services for computer programs. The operating system is a component of the system software in a computer system. Application programs usually require an operating system to function.

For hardware functions such as input and output and memory allocation, the operating system acts as an intermediary between programs and the computer hardware, although the application code is usually executed directly by the hardware and frequently makes system calls to an OS function or be interrupted by it. Operating systems are found on many devices that contain a computer—from cellular phones and video game consoles to web servers and supercomputers.

Mobile Operating System:

Different types of Mobile Operating Systems are:

Android:

Android is a mobile operating system (OS) currently developed by Google, based on the Linux kernel and designed primarily for touchscreen mobile devices such as smartphones and tablets. Android's user interface is based on direct manipulation, using touch gestures that loosely correspond to real-world actions, such as swiping, tapping and pinching, to manipulate on-screen objects, along with a virtual keyboard for text input. In addition to touchscreen devices, Google has further developed Android TV for televisions, Android Auto for cars and Android Wear for wrist watches, each with a specialized user interface. Variants of Android are also used on notebooks, game consoles, digital cameras, and other electronics. As of 2015, Android has the largest installed base of all operating systems.

Initially developed by Android, Inc., which Google bought in 2005, Android was unveiled in 2007, along with the founding of the Open Handset Alliance – a consortium of hardware, software, and telecommunication companies devoted to advancing open standards for mobile devices

Java OS:

J2ME platform is a set of technologies, specifications and libraries developed for small devices like mobile phones, pagers, and personal organizers. Java ME was designed by Sun Microsystems. It is licensed under GNU General Public License.

It defines a minimum platform including the java language, virtual machine features and minimum class libraries for a grouping of devices.

It supports higher-level services common to a more specific class of devices. A profile builds on a configuration but adds more specific APIs to make a complete environment for building applications.

It includes two kinds of platforms:

- High-end platform for high-end consumer devices. E.g. TV set-top boxes, Internet TVs, auto-mobile navigation systems
- Low-end platform for low-end consumer devices. E.g.cell phones, and pagers.

iOS:

iOS (originally iPhone OS) is a mobile operating system created and developed by Apple Inc. and distributed exclusively for Apple hardware. It is the operating system that presently powers many of the company's mobile devices, including the iPhone, iPad, and iPod touch.

It is based on Mach Kernel and Drawing core as Mac OS X.

The Mac OS X kernel includes the following component:

– Mach Kernel

– BSD

– I/O component

– File Systems

– Networking components

Mac OS X has a preemptive multitasking environment. Preempting is the act of taking the control of operating system from one task and giving it to another task.

It supports real-time behavior.

In Mac OS X, each application has access to its own 4 GB address space.

Ubuntu Touch

Ubuntu Touch (also known as **Ubuntu Phone**) is a mobile version of the Ubuntu operating system developed by Canonical UK Ltd and Ubuntu Community. It is designed primarily for touchscreen mobile devices such as smartphones and tablet computers.

Mark Shuttleworth announced on 31 October 2011 that by Ubuntu 14.04, Ubuntu will support smartphones, tablets, smart TVs and other smart screens (as car head units and smart watches).

Ubuntu Touch was released to manufacturers on 16 September 2014. Bq Aquaris E4.5, the world's first Ubuntu-based smartphone went on sale in Europe on 9 February 2015.

Tizen

Tizen is an operating system based on the Linux kernel and the Linux API. It targets a very wide range of devices including smartphones, tablets, in-vehicle infotainment (IVI) devices, smart TVs, PCs, smart cameras, wearable computing (such as smart watches), Blu-ray players, printers and smart home appliances[1] (such as refrigerators, lighting, washing machines, air conditioners, ovens/microwaves and a robotic vacuum cleaner). Its purpose is to offer a consistent user experience across devices. Tizen is a project within the Linux Foundation and is governed by a Technical Steering Group (TSG) composed of Samsung and Intel among others.

The Tizen Association was formed to guide the industry role of Tizen, including requirements gathering, identifying and facilitating service models, and overall industry marketing and education.

Windows Phone (WP)

Windows Phone (**WP**) is a family of mobile operating systems developed by Microsoft for smartphones as the replacement successor to Windows Mobile and Zune. Windows Phone features a new user interface derived from Metro design language. Unlike Windows Mobile, it is primarily aimed at the consumer market rather than the enterprise market. It was first launched in October 2010 with Windows Phone 7.Windows Phone 8.1 is the latest public release of the operating system, released to manufacturing on April 14, 2014.

Windows 10 Mobile will succeed Windows Phone 8.1 in 2015; it emphasizes a larger amount of integration and unification with its PC counterpart—including a new, unified application ecosystem, along with an expansion of its scope to include small-screened tablets.

Firefox OS

Firefox OS is a Linux kernel-based open-source operating system for smartphones, tablet computers and smart TVs. It is being developed by Mozilla, the non-profit organization best known for the Firefox web browser.

Firefox OS is designed to provide a complete, community-based alternative system for mobile devices, using open standards and approaches such as HTML5 applications, JavaScript, a robust privilege model, open web APIs to communicate directly with cellphone hardware, and application marketplace. As such, it competes with commercially developed operating systems such as Apple's iOS, Google's Android, Microsoft's Windows Phone, BlackBerry's BlackBerry 10 and Jolla's Sailfish OS.

Firefox OS was publicly demonstrated in February 2012, on Android-compatible smartphones. In January 2013, at CES 2013, ZTE confirmed they would be shipping a smartphone with Firefox OS, and on July 2, 2013, Telefónica launched the first commercial Firefox OS based phone, ZTE Open, in Spain which was quickly followed by Geeks Phone's Peak+. As of December 16, 2014, Firefox OS phones are offered by 14 operators in 28 countries throughout the world.

Mozilla has also partnered with T2Mobile to make a Firefox OS reference phone dubbed "Flame" which is designed for developers to contribute to Firefox OS and to test apps.

Symbian

Symbian is a closed-source mobile operating system (OS) and computing platform designed for smartphones. Symbian was originally developed by Symbian Ltd. The current form of Symbian is an open-source platform developed by Symbian Foundation in 2009, as the successor of the original **Symbian OS**. Symbian was used by many major mobile phone brands, like Samsung, Motorola, Sony Ericsson, and above all by Nokia. It was the most popular smartphone OS on a worldwide average until the end of 2010, when it was overtaken by Android.

Symbian OS was created with three systems design principles in mind:

1. the integrity and security of user data is paramount
2. user time must not be wasted
3. all resources are scarce.

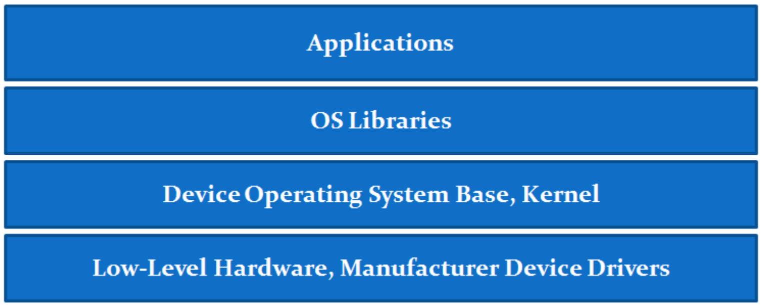
Blackberry:

BlackBerry OS is a proprietary mobile operating system developed by BlackBerry Ltd for its BlackBerry line of smartphone handheld devices. The operating system provides multitasking and supports specialized input devices that have been adopted by BlackBerry Ltd. for use in its handhelds, particularly the trackwheel, trackball, and most recently, the trackpad and touchscreen.

The BlackBerry platform is perhaps best known for its native support for corporate email, through MIDP 1.0 and, more recently, a subset of MIDP 2.0, which allows complete wireless activation and synchronization with Microsoft Exchange, Lotus Domino, or Novell GroupWise email, calendar, tasks, notes, and contacts, when used with BlackBerry Enterprise Server. The operating system also supports WAP 1.2.

Structure of Mobile OSes.

A mobile OS is a software platform on top of which other programs called application program, can run on mobile devices such as PDA, cellular phones, Smartphone and etc.


Applications
OS Libraries
Device Operating System Base, Kernel
Low-Level Hardware, Manufacturer Device Drivers

Low Level Hardware

-Mobile Processors

- Processors use RISC architecture.
- Mainly two types

  a. ARM Processors (32-bit RISC processor with Low power consumption)

  b. MIPS - Microprocessor without Interlocked Pipeline Stages used in embedded systems

- **Mobile Memory**

  The better the memory management offered by the OS, the wider the options available to applications developers. Mobile devices have two types of memories

  - ROM - For operating system and preinstalled programs
  - RAM - For user information

  Types of RAMs

  - DRAM (Dynamic RAM): cheapest, used in mobile devices
  - EDO (Enhanced Data Output): more expensive but offers a speed increase over DRAM
  - SDRAM (Synchronous Dynamic RAM): a further 50% speed (iPAQ)
  - DDR (Double Data Rate) SDRAM is twice as fast as SDRAM
  - OUM (Ovonics Unified Memory): experimental

Kernal

Responsible for services such as

- Security
- Memory Management
- Process Management

It include the following components

- I/O Components
- File Systems
- Networking Components

Introduction to Development Environment

Native Apps

Any app installed directly onto a device – usually from its associated app store – can be called a "native" app (although a native app can have HTML5 elements). Native apps are platform dependent – i.e. an app made for iPhone can work only on an iPhone, not an Android device.

Native apps have distinct advantages and disadvantages, such as:

Pros of Native Apps

- Better performance: A native app has complete access to a phone's hardware resources. The app also interacts directly with the phone without the mediation of a web browser. This lends it significantly better performance, particularly when rendering graphics and animation.
- Easier development: Developers can easily leverage platform SDKs to create native apps. This makes for easier development, provided you already have access to experienced engineering talent.
- Better distribution: Native apps can be distributed directly through relevant app stores. This can be an incredibly powerful distribution channel that not only makes app installation and updates easy, but can also help cut down on marketing costs through app stores' built-in discovery features.
- Better monetization: Since native apps are associated with app stores, they can take advantage of the store's built-in monetization features, such as one-click payments.

Cons of Native Apps

- Increased development time and costs: Since developers must build separate apps for each platform, total development time might be significantly higher than an equivalent HTML5 app. This also increases costs since developing for different platforms demands mastery over different languages and development environments. App developers themselves tend to be more expensive than their HTML5/JavaScript counterparts. Higher maintenance costs: From the above, it follows that maintenance costs for native apps are also higher. A business might require the services of separate iOS, Android, and Windows Phone developers to keep its apps up and running, eating into profits.
- App store content restriction: Every app distributed through the app store must adhere to strict content guidelines. For certain businesses (such as gambling), this may be a deal-breaker.
- App store fees: App store fees (up to 30 percent) can eat into your profit margins and make an already costly undertaking prohibitively expensive.

HTML5 Apps

A web app delivered through an HTML5 enabled browser may be called an “HTML5 app.” This means instead of downloading your app from the app store, a user would open her/his browser and navigate to a URL to access your app.

HTML5 apps offer various advantages and disadvantages, such as:

Pros of HTML5 Apps

- Platform independent: HTML5 apps are completely platform independent. It does not matter whether your users are on Android, iOS, or Windows Phone, they can all access your app as long as they have a web browser.
- Easier updates: Native apps update much like desktop applications – the user has to download each update individually. HTML5 apps, on the other hand, support centralised updates. Every time the user visits your app through her/his browser, she/he sees the latest version of your app with no additional download requirements.
- Faster development: HTML5 apps are written in HTML, CSS, JavaScript, and server-side languages such as ASP.NET. This cuts down on development time since there are already vast libraries for these languages (such as jQuery for JavaScript) and a huge body of trained developers.
- Cheaper: Since HTML5 apps are built on relatively simple web technologies such as HTML5 and JavaScript, you will find it easier to hire affordable HTML5 developers than similarly talented native app developers.
- No content restrictions: Since HTML5 apps need not seek approval from a closed app store, they can carry whatever content you want.
- No fees: HTML5 apps are delivered directly through the browser. This cuts out the app store and saves you on the 30 percent app store fees.
- Better data: Any customer data you collect through a native app is subject to the rules of the app store in question. There are no such restrictions with HTML5 apps; you can collect whatever customer data you want. In fact, the Apple App Store withholding subscriber data was one of the reasons why Financial Times switched to a web app.
- Distribution through app store: Until a couple of years ago, there was no easy method to distribute a HTML5 app through the app store. However, new frameworks such as Phonegap and Trigger.io have eased the distribution problem considerably bundling up HTML5 apps as native apps (which can then be distributed through the app store). Although performance remains a concern and Phonegap still doesn’t support every mobile OS fully, it’s a capable alternative to building separate native apps for every platform.

Cons of HTML5 Apps

- Poorer performance: An HTML5 app has limited access to a phone’s hardware. This leads to poorer performance, especially when dealing with heavy graphics, although new platforms such as Famo.us are trying to mitigate this problem.

- Device fragmentation: Device fragmentation within web browsers is a real thing. Different devices might render the same app differently. This means your developers will need extensive testing and fine-tuning to get the UI right, especially when working on more complex apps.
- Technical limitations: An HTML5 does not have complete access to a device's features and events. This poses a limit on what you can do with an HTML5 app. The UI options for HTML5 apps are also limited.
- Limited monetisation opportunities: A native app can make money through direct app sales and in-app purchases. These monetisation options are not available for HTML5 apps. This can be particularly challenging for "freemium" apps.

Introduction to Android:

It is a platform and an operating system for mobile devices based on the Linux operating system. It allows developers design applications in a java-like language using Google-developed java libraries.

API Levels and versions of Android:

The version history of the Android mobile operating system began with the release of the Android beta in November 2007. The first commercial version, Android 1.0, was released in September 2008. Android is under ongoing development by Google and the Open Handset Alliance (OHA), and has seen a number of updates to its base operating system since its initial release.

- Android 1.0 (API level 1)
- Android 1.1 (API level 2)
- Android 1.5 Cupcake (API level 3)
- Android 1.6 Donut (API level 4)
- Android 2.0 Eclair (API level 5)
- Android 2.0.1 Eclair (API level 6)
- Android 2.1 Eclair (API level 7)
- Android 2.2–2.2.3 Froyo (API level 8)
- Android 2.3–2.3.2 Gingerbread (API level 9)
- Android 2.3.3–2.3.7 Gingerbread (API level 10)
- Android 3.0 Honeycomb (API level 11)
- Android 3.1 Honeycomb (API level 12)
- Android 3.2–3.2.6 Honeycomb (API level 13)
- Android 4.0–4.0.2 Ice Cream Sandwich (API level 14)
- Android 4.0.3–4.0.4 Ice Cream Sandwich (API level 15)
- Android 4.1–4.1.2 Jelly Bean (API level 16)
- Android 4.2–4.2.2 Jelly Bean (API level 17)
- Android 4.3–4.3.1 Jelly Bean (API level 18)
- Android 4.4–4.4.4 KitKat (API level 19)
- Android 4.4W–4.4W.2 KitKat, with wearable extensions (API level 20)

- Android 5.0–5.0.2 Lollipop (API level 21)
- Android 5.1–5.1.1 Lollipop (API level 22)

Pros and Cons of Android:

Pros:

- Android is open source Operating System and is customizable.
- Android has large number of OEM's (Original Equipment Manufacturers)
- It solves user's problems through Google support.
- Android has large number of apps in its play store.

Cons:

-Not as secure as iOS and windowsOS.

-Too many apps creating mess in the store, heavy OS in itself

-Most of the apps are dolphin browsers, wallpapers, malwares (considerable part) or are not even downloaded once.

-App compatibility issues for various versions of android. Not all phones run the same version at a given time.

-Over time phones become slow in operation.

-Quickly eat up the battery

Comparison of Android with Other OSes:

- The number of apps of Android in its store (Google play , Mobango) is about 1.5 million, apple's store is about 1.4 million and windows has got about 0.4 million till date.
- Android was the first to support Wi-Fi Direct, multiple user accounts and screen mirroring support.
- The number of OEM for android OS is more than 100, for windows only about 25 while iOS has only one i.e. Apple Inc.
- Android is open source and customizable where iOS and windows are closed source.

Introduction to android VM and Runtime (Dalvik and ART)

Dalvik

Dalvik is a process virtual machine (VM) in Google's Android operating system that executes applications written for Android. This makes Dalvik an integral part of the Android software stack (in Android versions 4.4 "KitKat" and earlier) that is typically used on mobile devices such as mobile phones and tablet computers, as well as more recently on devices such as smart TVs and wearables.

Programs are commonly written in Java and compiled to bytecode for the Java virtual machine, which is then translated to Dalvik bytecode and stored in **.dex** (**Dalvik EXecutable**) and **.odex** (**Optimized Dalvik EXecutable**) files; related terms **odex** and **de-odex** are associated with respective bytecode conversions. The compact Dalvik Executable format is designed for systems that are constrained in terms of memory and processor speed.

Unlike Java VMs, which are stack machines, the Dalvik VM uses a register-based architecture that requires fewer, typically more complex virtual machine instructions. Dalvik programs are written in Java using the Android application programming interface (API), compiled to Java bytecode, and converted to Dalvik instructions as necessary.

A tool called **dx** is used to convert Java .class files into the .dex format. Multiple classes are included in a single .dex file. Duplicate strings and other constants used in multiple class files are included only once in the .dex output to conserve space. Java bytecode is also converted into an alternative instruction set used by the Dalvik VM. An uncompressed .dex file is typically a few percent smaller in size than a compressed Java archive (JAR) derived from the same .class files.

The Dalvik executables may be modified again when installed onto a mobile device. In order to gain further optimizations, byte order may be swapped in certain data, simple data structures and function libraries may be linked inline, and empty class objects may be short-circuited, for example.

Being optimized for low memory requirements, Dalvik has some specific characteristics that differentiate it from other standard VMs.

- The VM was slimmed down to use less space….The constant pool has been modified to use only 32-bit indices to simplify the interpreter
- Standard Java bytecode executes eight-bit stack instructions. Local variables must be copied to or from the operand stack by separate instructions.
- Dalvik instead uses its own 16-bit instruction set that works directly on local variables. The local variable is commonly picked by a four-bit "virtual register" field. This lowers Dalvik's instruction count and raises its interpreter speed.

Android 2.2 brought trace-based just-in-time (JIT) compilation into Dalvik, optimizing the execution of applications by continually profiling applications each time they run and dynamically compiling frequently executed short segments of their bytecode into native machine code. While Dalvik interprets the rest of application's bytecode, native execution of those short bytecode segments, called "traces", provides significant performance improvements.

ART (Android Runtime)

Android Runtime (ART) is an application runtime environment used by the Android operating system. ART replaces Dalvik, which is the process virtual machine originally used by Android, and performs transformation of the application's bytecode into native instructions that are later executed by the device's runtime environment.

ART and Dalvik are compatible runtimes running Dex bytecode, so apps developed for Dalvik should work when running with ART. However, some techniques that work on Dalvik do not work on ART.

Unlike Dalvik, ART introduces the use of ahead-of-time (AOT) compilation by compiling entire applications into native machine code upon their installation. By eliminating Dalvik's interpretation and trace-based JIT compilation, ART improves the overall execution efficiency and reduces power consumption, which results in improved battery autonomy on mobile devices. At the same time, ART brings faster execution of applications, improved memory allocation and garbage collection (GC) mechanisms, new applications debugging features, and more accurate high-level profiling of applications.

To maintain backward compatibility, ART uses the same input bytecode as Dalvik, supplied through standard .dex files as part of APK files, while the .odex files are replaced with Executable and Linkable Format (ELF) executables. Once an application is compiled by using ART's on-device *dex2oat* utility, it is run solely from the compiled ELF executable; as a result, ART eliminates various application execution overheads associated with Dalvik's interpretation and trace-based JIT compilation. As a downside, ART requires additional time for the compilation when an application is installed, and applications take up slightly larger amounts of secondary storage (which is usually flash memory) to store the compiled code.

Android 4.4 "KitKat" brought a technology preview of ART, including it as an alternative runtime environment and keeping Dalvik as the default virtual machine. In the next major Android release, Android 5.0 "Lollipop", Dalvik was entirely replaced by ART.

Some of the major features implemented by ART are:

1. Ahead-of-time (AOT) compilation
2. Improved garbage collection
3. Development and debugging improvements like

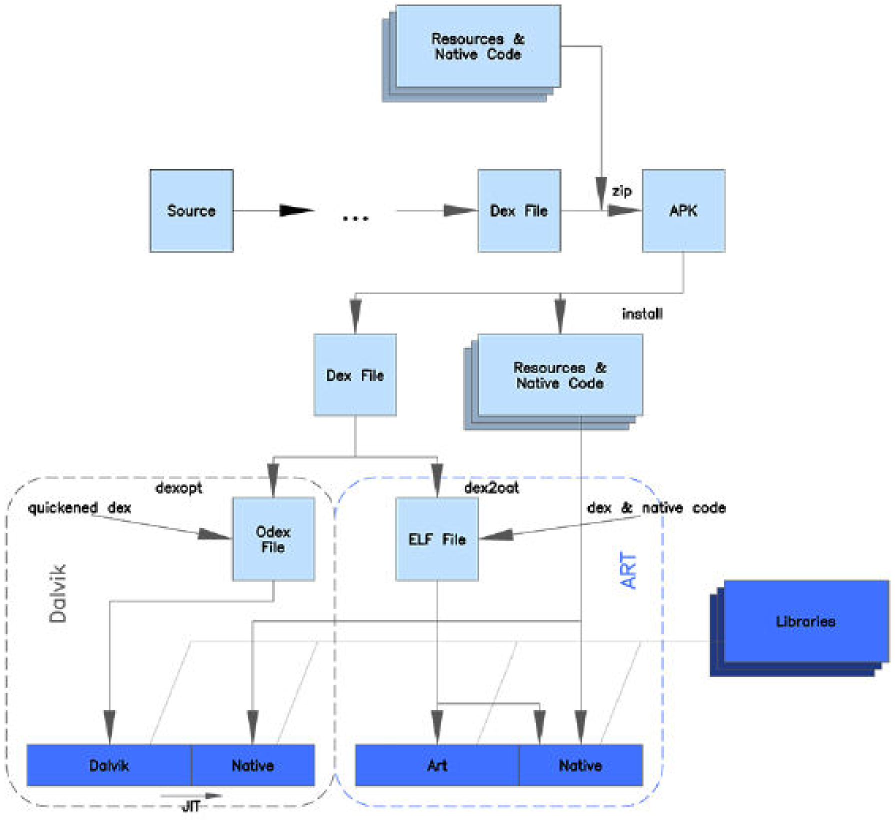


Figure : Architecture of Dalvik and ART Installation and configuration of Android SDK and Eclipse IDE

Eclipse:

From the moment the Android SDK was released, Eclipse has been the standard IDE for Android development and remains so to this day. From our perspective, these are the main reasons for this:

- With the release of the Android SDK, Google immediately made available the extensive Android Development Tools (ADT) plugin for Eclipse.
- ADT is used and maintained by the Google Android platform developers themselves.
- Eclipse/ADT, like the Android SDK itself, is open source and available free of charge.

Eclipse Home and Download Area

- *http://www.eclipse.org*
- *http://www.eclipse.org/downloads/*

Android Development Tools Plugin for Eclipse ADT

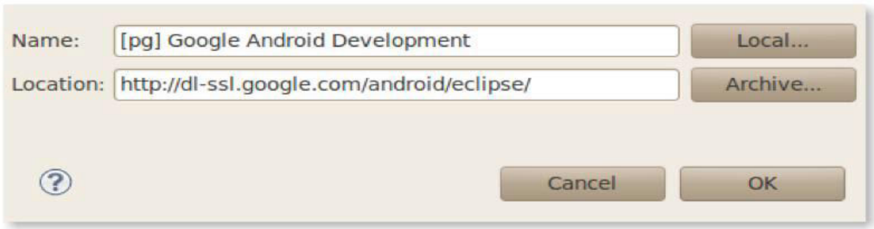
- *http://developer.android.com/sdk/eclipse-adt.html*

Official Google ADT Eclipse Update Site

- *https://dl-ssl.google.com/android/eclipse/*
- *http://dl-ssl.google.com/android/eclipse/*

Installing ADT

- To install the Eclipse ADT plugin, go to the Eclipse Help
- Install New Software menu and click the Add (a New Software Site) button.



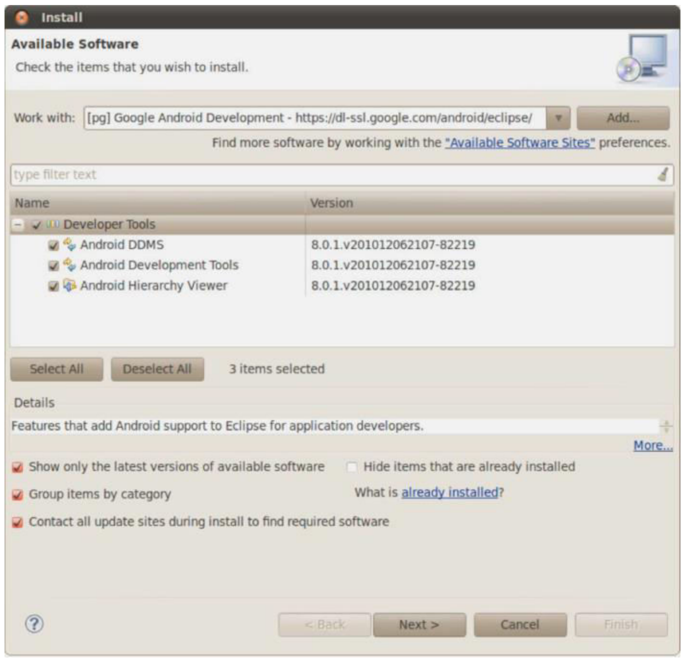
Name: [pg] Google Android Development

Location: http://dl-ssl.google.com/android/eclipse/

- This should display the dialog shown.
- Enter your own preferred Name and in the Location use either of the following resource locators:

  https://dl-ssl.google.com/android/eclipse/

  http://dl-ssl.google.com/android/eclipse/
- The Eclipse Available Software (dialog shown) displays with the ADT listed. Select all the tools and click next or Finish. Continue with the setup workflow until the installation is complete.



Install

Available Software

Check the items that you wish to install.

Work with: [pg] Google Android Development - https://dl-ssl.google.com/android/eclipse/

Find more software by working with the "Available Software Sites" preferences.

Name	Version
✓ Developer Tools	
✓ Android DDMS	8.0.1.v201012062107-82219
✓ Android Development Tools	8.0.1.v201012062107-82219
✓ Android Hierarchy Viewer	8.0.1.v201012062107-82219

3 items selected

Details

Features that add Android support to Eclipse for application developers.

✓ Show only the latest versions of available software

☐ Hide items that are already installed

✓ Group items by category

What is already installed?

✓ Contact all update sites during install to find required software

Net beans:

Over the last years, the NetBeans IDE has become a fully viable alternative to Eclipse in the IDE. Not only that, but the NetBeans platform is also making a serious challenge to Eclipse for a stake in the rich client platform development space.

NetBeans Home and Download Area

- *http://netbeans.org/*
- *http://netbeans.org/downloads/*

Android Plugin for NetBeans: NBAndroid

- *http://kenai.com/projects/nbandroid/*
- *http://kenai.com/projects/nbandroid/pages/Install*

Android Development In NetBeans, with NBAndroid:

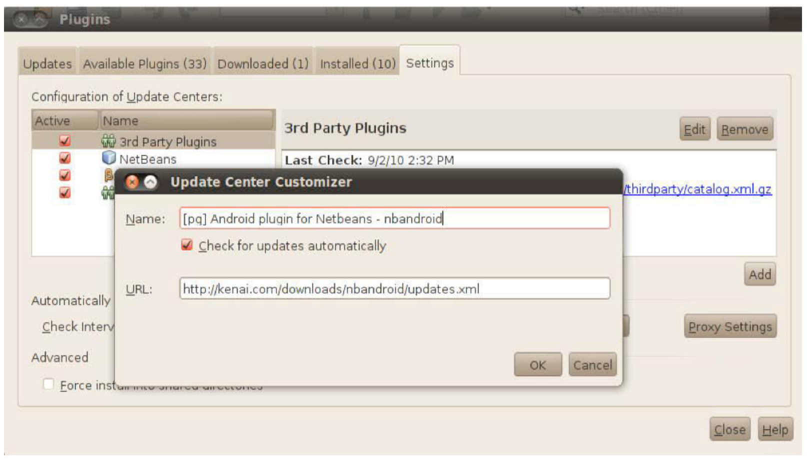
- *http://wiki.netbeans.org/IntroAndroidDevNetBeans*

Official NBAndroid NetBeans Update Site:

- http://kenai.com/downloads/nbandroid/updates.xml

Installing ADT:

- To install the NetBeans NBAndroid plugin, go to the NetBeans Tools > Plugins menu and click the Settings tab.
- Click the Add button to show the Update Center Customizer.
- Enter your own preferred Name for the Update Center and in the URL text box enter the following:



Name: [pq] Android plugin for Netbeans - nbandroid

[✓] Check for updates automatically

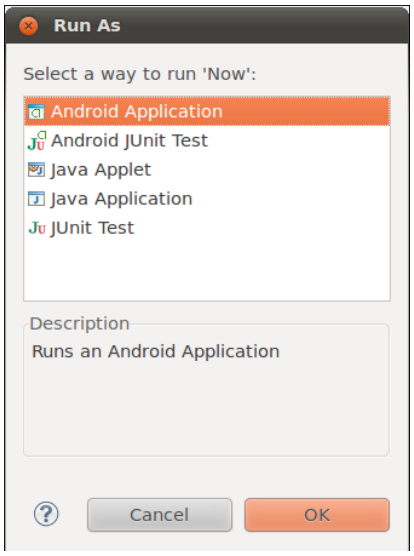
URL: http://kenai.com/downloads/nbandroid/updates.xml

http://kenai.com/downloads/nbandroid/updates.xml

- Click the OK button.
- Click the Available Plugin tab. Find and select Android. The description should appear on the right side.
- Now click the Install button on the bottom left of the dialog.
- Click next, accept the license agreement, and let the installation continue until it is complete.

Running the Emulator:

- With your project selected in the Package Explorer pane, click the green "play" button in the Eclipse toolbar to run your project.
- The first time you do this, you will have to go through a few steps to set up a "run configurations", and so Eclipse knows what you want to do.
- First, in the "Run As" list, choose "Android Application": If you have more than one emulator AVD or device available, you will then get an option to choose which you wish to run the application on. Otherwise, if you do not have a device plugged in, the emulator will start up with the AVD you created earlier.
- Then, Eclipse will install the application on your device or emulator and start it up.



Select a way to run 'Now':

Description

Runs an Android Application

Using adb command lines

This makes at least two assumptions:

1. You have the Android SDK installed.
2. You have an Android emulator (or physical device) running.

At the adb shell prompt we can enter a variety of commands to interact with your Android emulator or device.

The brief list of frequently used commands is:

adb devices	list the installed devices
adb pull <remote> <local>	copy a file or directory to the emulator or device

adb install Foo.apk	install the APK file/app
adb install -r Foo.apk	update the already installed app
adb uninstall <pkg>	uninstall the app given by pkg
ls	list files and directories
logcat	view the system's log buffers
reboot	reboot the device

This command lets us uninstall an Android application, where the argument given to the uninstall command is the root package name of the app:

$adb uninstall com.MyPackageName

2.Java Architecture And OOPS

Java Architecture

1. Compilation in Java

Java combines both the approaches of compilation and interpretation. First, java compiler compiles the source code into byte code. At the run time, Java Virtual Machine (JVM) converts this byte code and generates machine code which will be directly executed by the machine in which java program runs.

2. Java Virtual Machine (JVM)

JVM is a component which provides an environment for running Java programs. JVM converts the byte code into machine code which will be executed the machine in which the Java program runs.

3. Why Java is Platform Independent?

Platform independence is one of the main advantages of Java. In another words, java is portable because the same java program can be executed in multiple platforms without making any changes in the source code. You just need to write the java code for one platform and the same program will run in any platforms. But how does Java make this possible?

First the Java code is compiled by the Java compiler and generates the byte code. This byte code will be stored in class files. Java Virtual Machine (JVM) is unique for each platform. Though JVM is unique for each platform, all response the same byte code and convert it into machine code required for its own platform and this machine code will be directly executed by the machine in which java program runs. This makes Java platform independent and portable.

4. Java Runtime Environment (JRE) and Java Architecture in Detail

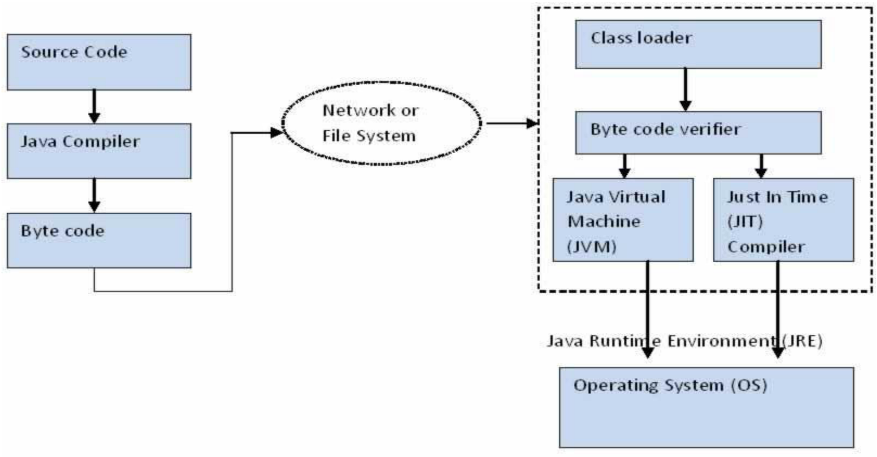
Java Runtime Environment contains JVM, class libraries and other supporting components.

As you know the Java source code is compiled into byte code by Java compiler. This byte code will be stored in class files. During runtime, this byte code will be loaded, verified and JVM interprets the byte code into machine code which will be executed in the machine in which the Java program runs.

A Java Runtime Environment performs the following main tasks respectively.

1. Loads the class
   This is done by the class loader
2. Verifies the byte code
   This is done by byte code verifier.
3. Interprets the byte code
   This is done by the JVM

These tasks are described in detail in the subsequent sessions. A detailed Java architecture can be drawn as given below.



4.1. Class loader

Class loader loads all the class files required to execute the program. Class loader makes the program secure by separating the namespace for the classes obtained through the network from the classes available locally. Once the byte code is loaded successfully, then next step is byte code verification by byte code verifier.

4.2. Byte code verifier

The byte code verifier verifies the byte code to see if any security problems are there in the code. It checks the byte code and ensures the followings.

1. The code follows JVM specifications.
2. There is no unauthorized access to memory.
3. The code does not cause any stack overflows.
4. There are no illegal data conversions in the code such as float to object references.

Once this code is verified and proven that there is no security issues with the code, JVM will convert the byte code into machine code which will be directly executed by the machine in which the Java program runs.

4.3. Just in Time Compiler

When the Java program is executed, the byte code is executed by JVM. But this interpretation is a slower process. To overcome this difficulty, JRE include the component JIT compiler. JIT makes the execution faster.

If the JIT Compiler library exists, when a particular byte code is executed first time, JIT complier compiles it into native machine code which can be directly executed by the machine in which the Java program runs. Once the byte code is recompiled by JIT compiler, the execution time needed will be much lesser. This compilation happens when the byte code is about to be executed and hence the name "Just in Time".

Once the byte code is compiled into that particular machine code, it is cached by the JIT compiler and will be reused for the future needs. Hence the main performance improvement by using JIT compiler can be seen when the same code is executed again and again because JIT make use of the machine code which is cached and stored.

5. Why Java is Secure?

As you have noticed in the prior session "Java Runtime Environment (JRE) and Java Architecture in Detail", the byte code is inspected carefully before execution by Java Runtime Environment (JRE). This is mainly done by the "Class loader" and "Byte code verifier". Hence a high level of security is achieved.

6. Garbage Collection

Garbage collection is a process by which Java achieves better memory management. Whenever an object is created, there will be some memory allocated for this object. This memory will remain as allocated until there are some references to this object. When there is no reference to this object, Java will assume that this object is not used anymore. When garbage collection process happens, these objects will be destroyed and memory will be reclaimed.

Java Classes and Objects:

Real-world objects share two characteristics: They all have *state* and *behavior*. Dogs have state (name, color, breed, hungry) and behavior (barking, fetching, wagging tail). Bicycles also have state (current gear, current pedal cadence, current speed) and behavior (changing gear, changing pedal cadence, applying brakes). Identifying the state and behavior for real-world objects is a great way to begin thinking in terms of object-oriented programming.

Object-oriented programming (OOP) is a programming paradigm based on the concept of "objects", which are data structures that contain data, in the form of fields, often known as *attributes;* and code, in the form of procedures, often known as *methods.* A distinguishing feature of objects is that an object's procedures can access and often modify the data fields of the object with which they are associated.

An entity that has state and behavior is known as an object e.g. chair, bike, marker, pen, table, car etc. It can be physical or logical (tangible and intangible). The example of intangible object is banking system.

An object has three characteristics:

- **State:** represents data (value) of an object.
- **Behavior:** represents the behavior (functionality) of an object such as deposit, withdraw etc.
- **Identity:** Object identity is typically implemented via a unique ID. The value of the ID is not visible to the external user. But, it is used internally by the JVM to identify each object uniquely.

For Example: Pen is an object. Its name is Reynolds, color is white etc. known as its state. It is used to write, so writing is its behavior.

Each object is said to be an instance of a particular class (for example, an object with its name field set to "Mary" might be an instance of class Employee). Procedures in object-oriented programming are known as methods, variables are also known as fields, members, attributes, or properties. This leads to the following terms:

- Class variables - belong to the *class as a whole*; there is only one copy of each one
- Instance variables or attributes - data that belongs to individual *objects*; every object has its own copy of each one
- Member variables - refers to both the class and instance variables that are defined by a particular class
- Class methods - belong to the *class as a whole* and have access only to class variables and inputs from the procedure call
- Instance methods - belong to *individual objects*, and have access to instance variables for the specific object they are called on, inputs, and class variables

Objects are accessed somewhat like variables with complex internal structure, and in many languages are effectively pointers, serving as actual references to an single instance of said object in memory within a heap or stack. They provide a layer of abstraction which can be used to separate internal from external code. External code can use an object by calling a specific instance method with a certain set of input parameters, read an instance variable, or write to an instance variable. Objects are created by calling a special type of method in the class known as a constructor. A program may create many instances of the same class as it runs, which operate independently. This is an easy way for the same procedures to be used on different sets of data.

Object-oriented programming that uses classes is sometimes called class-based programming, while prototype-based programming does not typically use classes. As a result, a significantly different yet analogous terminology is used to define the concepts of *object* and *instance*.

Class Methods and Instances:

Data is encapsulated in a class by declaring variables inside the class declaration. Variables declared in these scopes are known as **instance variables**. Instance variables are declared in the same way as local variables except that, they are declared outside any particular method.

Class variables are global to class and to all the instances of class. They are useful in communicating between different objects of the same class for keeping track of global states.

Methods are functions that operate on instances of classes in which they are defined. Objects can communicate with each other using methods and can call methods in other classes.

Method definition has four parts:

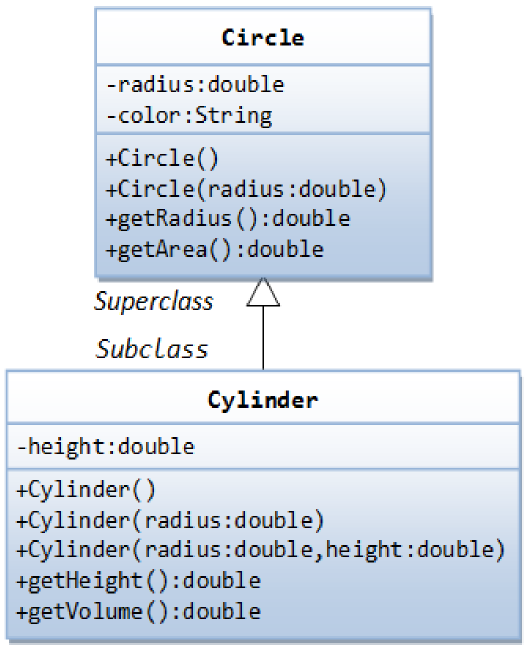
- Name of the method
- Type of object
- List of parameters
- Body of method

The methods which apply and operate on an instance are called instance methods and the methods which apply and operate on a class are called class methods.

Class methods, like class variables, are available to instances of the class and can be made available to other classes. Class methods can be used anywhere regardless of whether an instance of the class exists or not. Methods that provide some general utility but do not directly affect an instance of the class are declared as **class methods.**

Inheritance:

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time.



In OOP, we often organize classes in hierarchy to avoid duplication and reduce redundancy. The classes in the lower hierarchy inherit all the variables (static attributes) and methods (dynamic behaviors) from the higher hierarchies. A class in the lower hierarchy is called a subclass (or derived, child, extended class). A class in the upper hierarchy is called a superclass (or base, parent class). By pulling out all the common variables and methods into the superclasses, and leave the specialized variables and methods in the subclasses, redundancy can be greatly reduced or eliminated as these common variables and methods do not need to be repeated in all the subclasses. For example,

A subclass inherits all the variables and methods from its superclasses, including its immediate parent as well as all the ancestors. It is important to note that a subclass is not a "subset" of a superclass. In contrast, subclass is a "superset" of a superclass. It is because a subclass inherits all the variables and methods of the superclass; in addition, it extends the superclass by providing more variables and methods.

In this example, we derive a subclass called Cylinder from the superclass Circle. It is important to note that we reuse the class Circle. The Class Cylinder inherits all the member variables (radius and color) and methods (getRadius() , getArea(), among others) from its superclass Circle. It further defines a variable called height, two public methods - getHeight() and getVolume() and its own constructors.

Type of Inheritance:

When deriving a class from a base class, the base class may be inherited through public, protected or private inheritance.

We hardly use protected or private inheritance, but public inheritance is commonly used. While using different type of inheritance, following rules are applied:

- **Public Inheritance**:

When deriving a class from a public base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class.

- **Protected Inheritance:**

When deriving from a protected base class, public and protected members of the base class become protected members of the derived class.

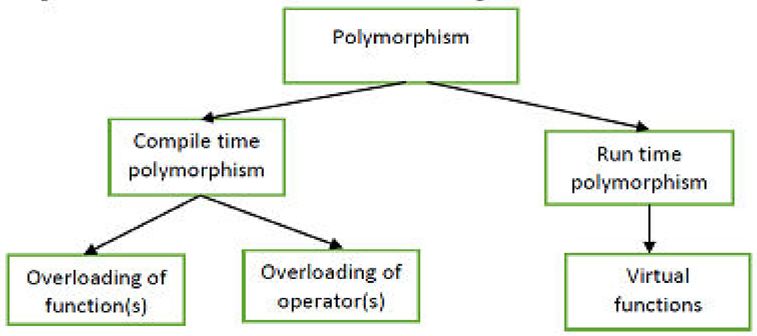
- **Private Inheritance:**

When deriving from a private base class, public and protected members of the base class become private members of the derived class.

Polymorphism:

Polymorphism is an important OOP concept. Polymorphism, a Greek term, means the ability to take more than one form. Polymorphism means, “One name, multiple forms”.

Polymorphism plays an important role in allowing objects having different internal structures to share the same external interface. This means that a general class of operations may be accessed in the same manner even though specific action associated with each operation may differ. Polymorphism is extensively used in implementing inheritance.


Polymorphism
Compile time polymorphism
Run time polymorphism
Overloading of function(s)
Overloading of operator(s)
Virtual functions

Compile time/Static polymorphism refers to the binding of functions on the basis of their signature (number, type and sequence of parameters). It is also called early binding. In this the compiler selects the appropriate function during the compile time. The examples of compile time polymorphism are Function overloading (use the same function name to create functions that perform a variety of different tasks) and Operator overloading (assign multiple meanings to the operators).

Dynamic or Subtype or Runtime Polymorphism means the change of form by entity depending on the situation. If a member function is selected while the program is running, then it is called run time polymorphism. This feature makes the program more flexible as a function can be called, depending on the context. This is also called late binding. The example of run time polymorphism is virtual function.

Interface and abstract Class:

An interface is a contract for what the classes can do. It, however, does not specify how the classes should do it. When a class implements a certain interface, it promises to provide implementation to all the abstract methods declared in the interface. Interface defines a set of common behaviors. The classes implement the interface agree to these behaviors and provide their own implementation to the behaviors. This allows you to program at the interface, instead of the actual implementation. One of the main usages of interface is provide a communication contract between two objects. If you know a class implements an interface, then you know that class contains concrete implementations of the methods declared in that interface, and you are guaranteed to be able to invoke these methods safely. In other words, two objects can communicate based on the contract defined in the interface, instead of their specific implementation.

A class containing one or more abstract methods is called an **abstract class**. An abstract class is incomplete in its definition, since the implementation of its abstract methods is missing. Therefore, an abstract class cannot be instantiated. In other words, you cannot create instances from an abstract class (otherwise, you will have an incomplete instance with missing method's body).

To use an abstract class, you have to derive a subclass from the abstract class. In the derived subclass, you have to override the abstract methods and provide implementation to all the abstract methods. The subclass derived is now complete, and can be instantiated. (If a subclass does not provide implementation to all the abstract methods of the superclass, the subclass remains abstract.)

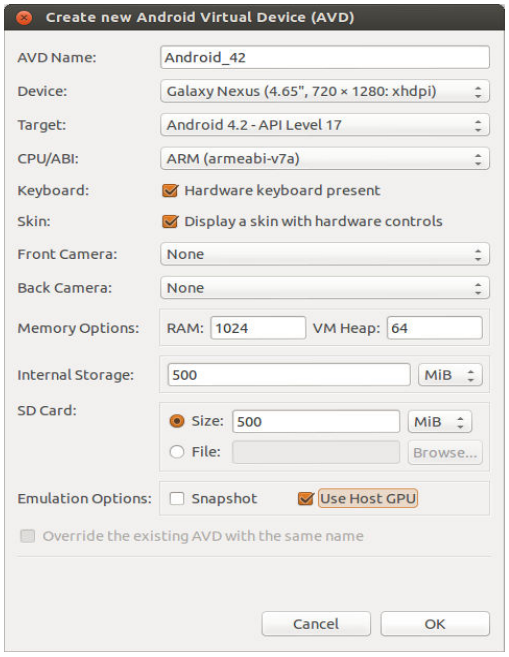
3. Android Classes and Basics

Android Fundamentals

Creating an Android App:

Step 1: To define an Android Virtual Device (ADV)

- Open the *AVD Manager* dialog via *Window → Android Virtual Device Manager* and press the *New* button.
- Enter the values similar to the following screenshot.



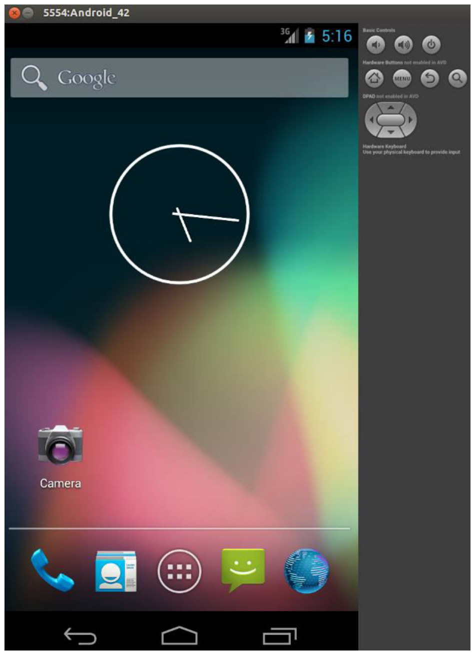
Create new Android Virtual Device (AVD)

AVD Name:	Android_42
Device:	Galaxy Nexus (4.65", 720 × 1280: xhdpi)
Target:	Android 4.2 - API Level 17
CPU/ABI:	ARM (armeabi-v7a)
Keyboard:	✓ Hardware keyboard present
Skin:	✓ Display a skin with hardware controls
Front Camera:	None
Back Camera:	None
Memory Options:	RAM: 1024 VM Heap: 64
Internal Storage:	500 MiB
SD Card:	● Size: 500 MiB ○ File: Browse...
Emulation Options:	☐ Snapshot ✓ Use Host GPU

☐ Override the existing AVD with the same name

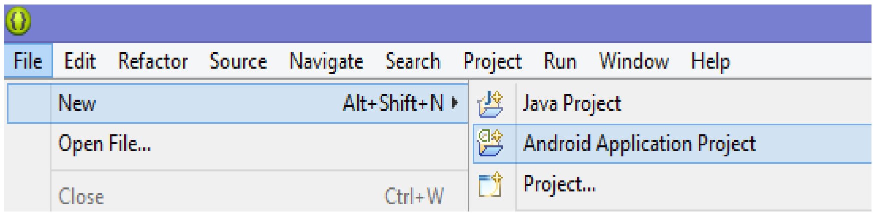
-

- Afterwards press the *OK* button. This will create the AVD configuration and display it under the list of available virtual devices. To test if setup is correct, select new entry and press the *Start* button.
  After the AVD runs, the following AVD appears.

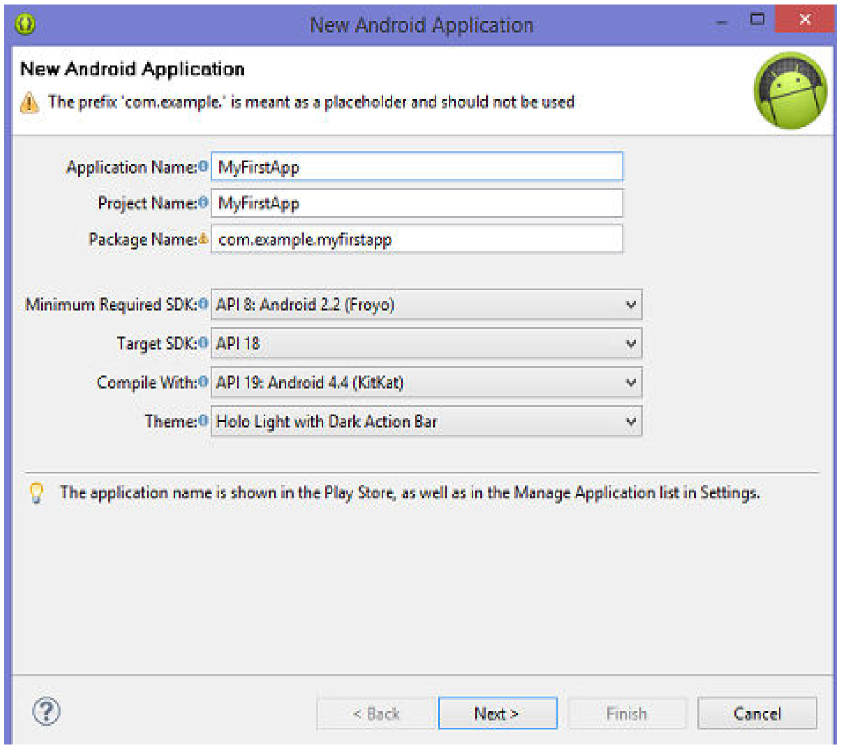


Step 2: Creating the Android App

- To create a new Android project select *File* → *New* → *Other* → *Android* → *Android Project* or *File* → *New* → *Android Application Project* from the menu



- Fill up the data similar to below screen shot.


New Android Application

The prefix 'com.example.' is meant as a placeholder and should not be used

Application Name: MyFirstApp

Project Name: MyFirstApp

Package Name: com.example.myfirstapp

Minimum Required SDK: API 8: Android 2.2 (Froyo)

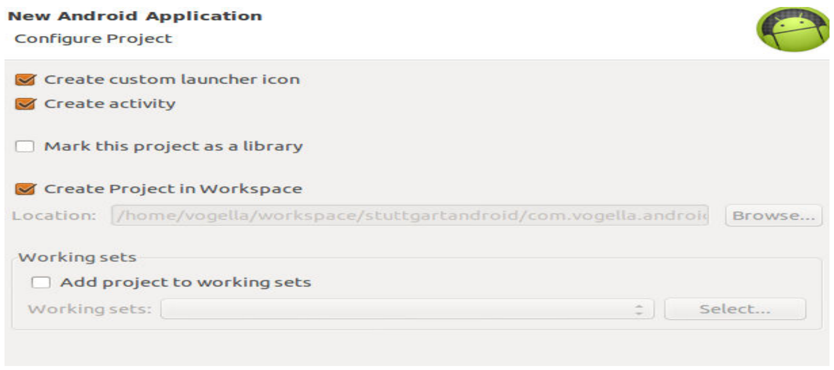
Target SDK: API 18

Compile With: API 19: Android 4.4 (KitKat)

Theme: Holo Light with Dark Action Bar

The application name is shown in the Play Store, as well as in the Manage Application list in Settings.

- Press the *Next* button and ensure that you have selected to create a launcher icon and an *activity*.


New Android Application

Configure Project

☑ Create custom launcher icon

☑ Create activity

☐ Mark this project as a library

☑ Create Project in Workspace

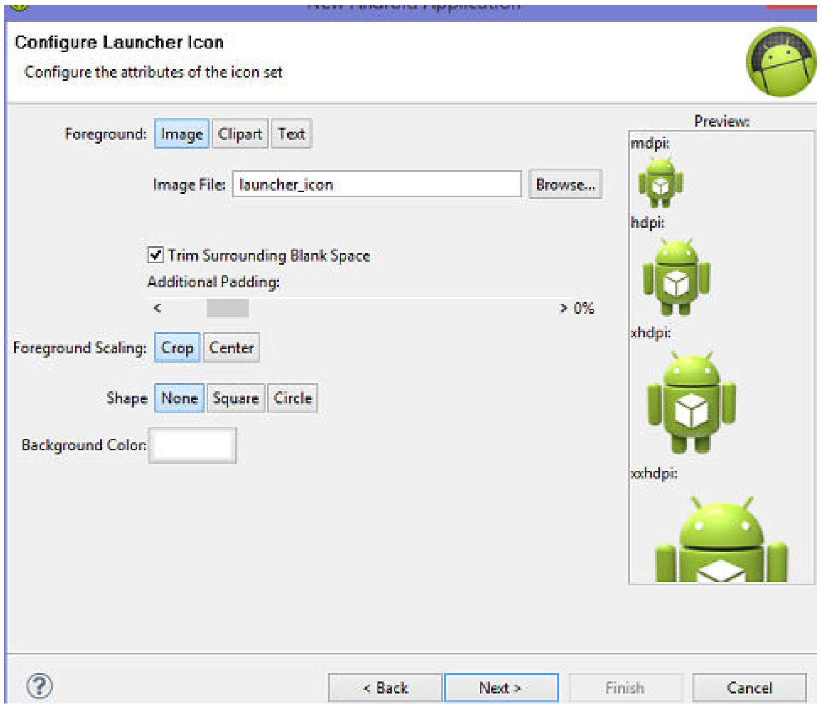
Location: /home/vogella/workspace/stuttgartandroid/com.vogella.androi

Working sets

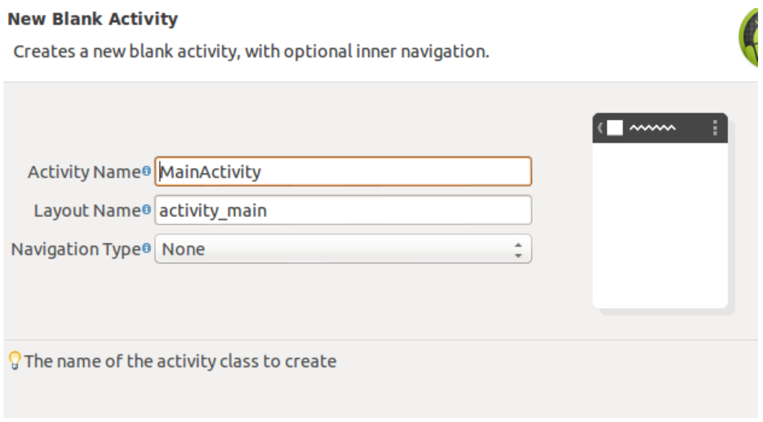
☐ Add project to working sets

Working sets:

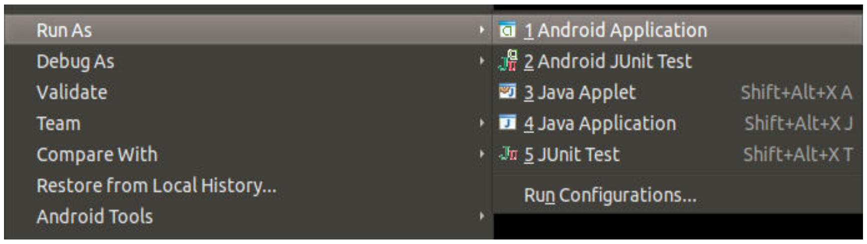
- On the wizard page for the launcher icon, create a nice looking icon. The following screenshot shows an example.


Configure Launcher Icon
Configure the attributes of the icon set
Foreground: Image Clipart Text
Image File: launcher_icon
Browse...
Trim Surrounding Blank Space
Additional Padding:
0%
Foreground Scaling: Crop Center
Shape None Square Circle
Background Color:
Preview:
mdpi:
hdpi:
xhdpi:
xxhdpi:
< Back
Next >
Finish
Cancel

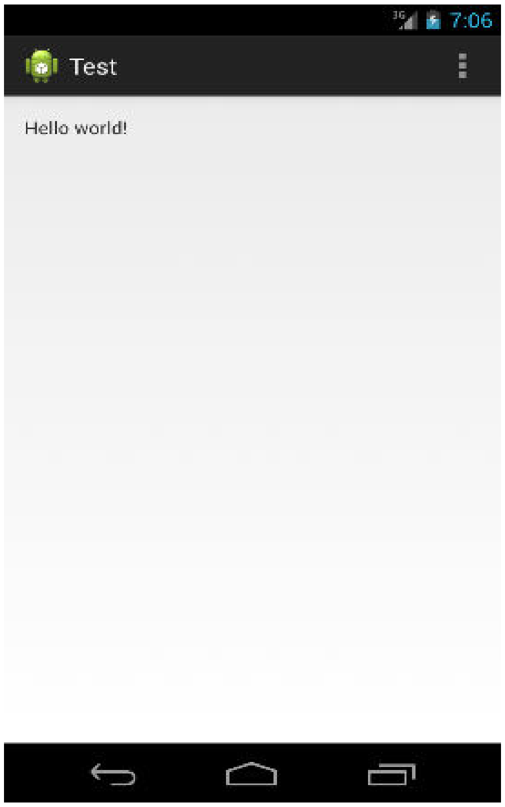
- Press the *Next* button and select on the next page the *Blank Activity* template. Press the *Next* button


New Blank Activity
Creates a new blank activity, with optional inner navigation.
Activity Name
MainActivity
Layout Name
activity_main
Navigation Type
None
The name of the activity class to create

- Start your Android application on the emulator. To build, install and run your application the Android Application, select your project, right click on it, and select *Run-As → Android Application*.


Run As
Debug As
Validate
Team
Compare With
Restore from Local History...
Android Tools
1 Android Application
2 Android JUnit Test
3 Java Applet Shift+Alt+X A
4 Java Application Shift+Alt+X J
5 JUnit Test Shift+Alt+X T
Run Configurations...

- This starts application on the AVD. The started application is a simple *Hello, world* application.



Hello world!

Android Manifests File

Every application must have an “AndroidManifest.xml” file (with precisely that name) in its root directory. The manifest file presents essential information about your app to the Android system, information the system must have before it can run any of the app's code.

Some of the works that are done by manifests are:

- It names the Java package for the application. The package name serves as a unique identifier for the application.
- It describes the components of the application — the activities, services, broadcast receivers, and content providers that the application is composed of. It names the classes that implement each of the components and publishes their capabilities (for example, which Intent messages they can handle). These declarations let the Android system know what the components are and under what conditions they can be launched.
- It determines which processes will host application components.
- It declares which permissions the application must have in order to access protected parts of the API and interact with other applications.
- It also declares the permissions that others are required to have in order to interact with the application's components.
- It lists the Instrumentation classes that provide profiling and other information as the application is running. These declarations are present in the manifest only while the application is being developed and tested; they're removed before the application is published.
- It declares the minimum level of the Android API that the application requires.

The Activity Class

Activity

An activity is a window that contains the user interface of the application. It is an application component that provides a screen with which users can interact in order to do something. Typically, applications have one or more activities; and the main purpose of an activity is to interact with the user.

Creating an activity

To create an activity, we must create a Java class that extends the Activity base class.

Example:

```
package com.example.Activity101;
import android.app.Activity;
import android.os.Bundle;
import com.example.Activity101.R;
 public class MainActivity extends Activity {
 /** Called when the activity is first created. */
@Override public void onCreate(Bundle savedInstanceState) { super.onCreate(savedInstanceState);
 setContentView(R.layout. main);
 }
 }
```

Here, "MainActivity" is the Java class that extends the Activity base class. (It is shown in line 5.)

The activity loads its user interface (UI) component using the XML file defined in "res/layout" folder.

In above example, UI is loaded from "main.xml" file via,

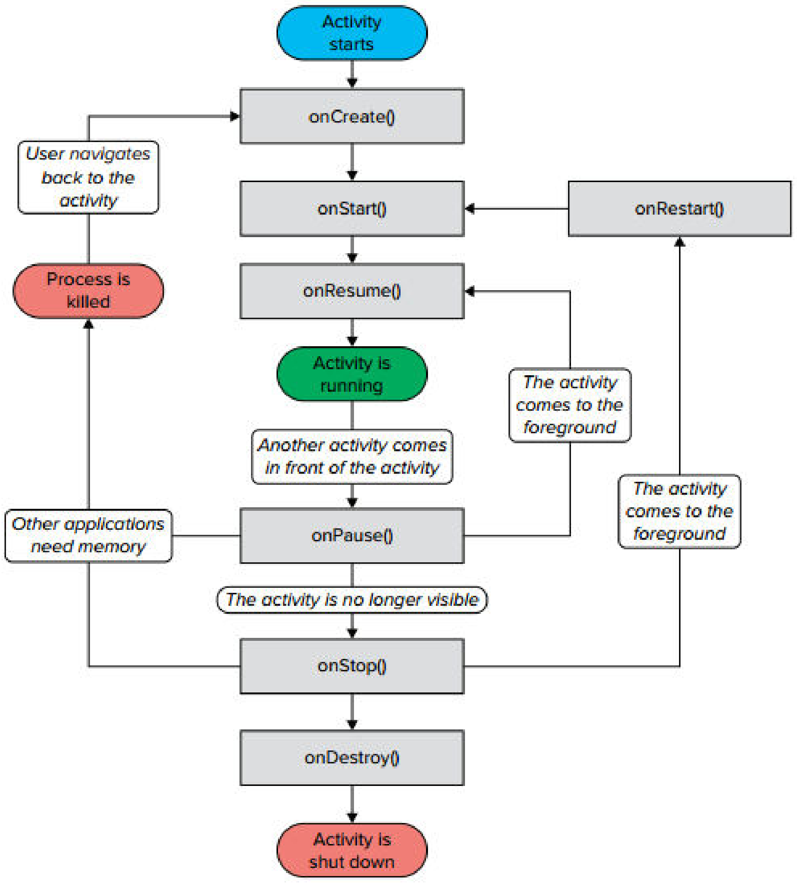
setContentView(R.layout. main); (It is shown in line 10.)

Note:

-setContentView() simply changes the display layout; it doesn't create a new activity. So, while switching between activities always use Intent.

Life cycle of Activity

There are various states of activity like Running, Paused, Stopped and Killed. The process to represent these states of activity can be defined as Activity life cycle.


Activity starts
onCreate()
User navigates back to the activity
onStart()
onRestart()
Process is killed
onResume()
Activity is running
The activity comes to the foreground
Another activity comes in front of the activity
The activity comes to the foreground
Other applications need memory
onPause()
The activity is no longer visible
onStop()
onDestroy()
Activity is shut down

The major event that governs the life cycle of an activity is:

1. onCreate( ) : called when the activity is first created.

2. onStart ( ) : called when the activity becomes visible to the user.

3. onResume( ): called when the activity starts interacting with the user.

4. onPause( ) : called when the current activity is being paused and previous activity is being resumed i.e. activity is going into the background but has not yet been killed.

5. onStop( ) : called when the activity is no longer visible to the user.

6. onRestart( ) : called when the activity has been stopped and is restarting again.

7. onDestroy( ) : called before the activity is destroyed by the system.

Default Activity and Launcher Activity

The activity which is created by default while making our Android project is known as default activity.

The activity via which we launch our application is known as launcher activity.

The first activity on an Android project by default becomes the launcher activity. To change the launcher activity manually we need to make some changes in the file "AndroidManifest.xml".

Example:

```
<activity android:name="com.example.myapp.Activity1"
android:label="@string/first_act" >
<intent-filter>
<action android:name="android.intent.action.MAIN" />
```

```
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<activity android:name=" com.example.myapp.Activity2"
android:label="@string/second_act" >
</activity>
```

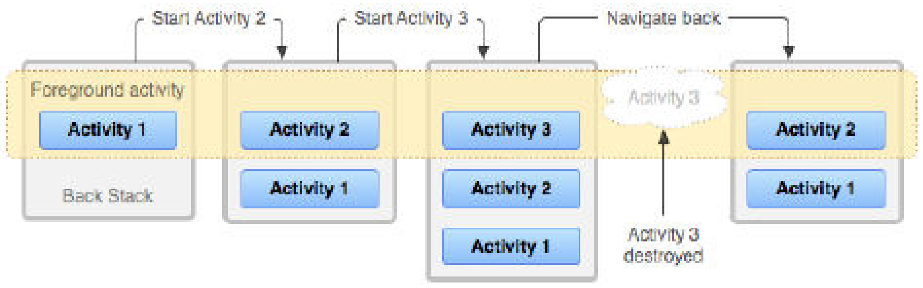
Here, in this example we have two activities i.e. 'Activity1' and 'Activity2'. Here Activity1 is the launcher activity. The launcher activity can be distinguished from other activities with the help of "<intent-filter>". Here, "<intent-filter>" is inside of 'Activity1' so it is the launcher activity.

We can also have both the activities as the launcher activities if we have the code of "<intent-filter>" inside of both activities. On doing so, both of the activities will be seen on the application launcher screen and behaves as a launcher activity.

Activity Stack (Task and Back Stack)

An application usually contains multiple activities. The activities can be designed to start the activities of the same application or that of the other applications. Even though the activities may be from different applications, Android maintains this seamless user experience keeping both activities in the same task.

A task is a collection of activities that users interact with when performing a certain job. The activities are arranged in a stack (the *back stack*), in the order in which each activity is opened.



The device Home screen is the starting place for most tasks. When the user touches an icon in the application launcher (or a shortcut on the Home screen), that application's task comes to the foreground. If no task exists for the application (the application has not been used recently), then a new task is created and the "main" activity for that application opens as the root activity in the stack.

When the current activity starts another, the new activity is pushed on the top of the stack and takes focus. The previous activity remains in the stack, but is stopped. When an activity stops, the system retains the current state of its user interface. When the user presses the *Back* button, the current activity is popped from the top of the stack (the activity is destroyed) and the previous activity resumes (the previous state of its UI is restored).

Activities in the stack are never rearranged, only pushed and popped from the stack—pushed onto the stack when started by the current activity and popped off when the user leaves it using the *Back* button. As such, the back stack operates as a "last in, first out" object structure.

Creating Splash and Login Activities

…………………..

The Intent Class

Intent is an abstract description of an operation to be performed. It can be used with

- startActivity to launch an Activity,
- broadcastIntent to send it to any interested BroadcastReceiver components, and
- startService(Intent) or bindService(Intent, ServiceConnection, int) to communicate with a background Service.

Intent provides a facility for performing late runtime binding between the codes in different applications. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities. It is basically a passive data structure holding an abstract description of an action to be performed.

There are two primary forms of intents we will use.

- **Explicit Intents** have specified a component (via setComponent(ComponentName) or setClass(Context, Class), which provides the exact class to be run. Often these will not include any other information, simply being a way for an application to launch various internal activities it has as the user interacts with the application.
- **Implicit Intents** have not specified a component; instead, they must include enough information for the system to determine which of the available components is best to run for that intent.

When using implicit intents, given such an arbitrary intent we need to know what to do with it. This is handled by the process of *Intent resolution*, which maps an Intent to an Activity, BroadcastReceiver, or Service (or sometimes two or more activities/receivers) that can handle it.

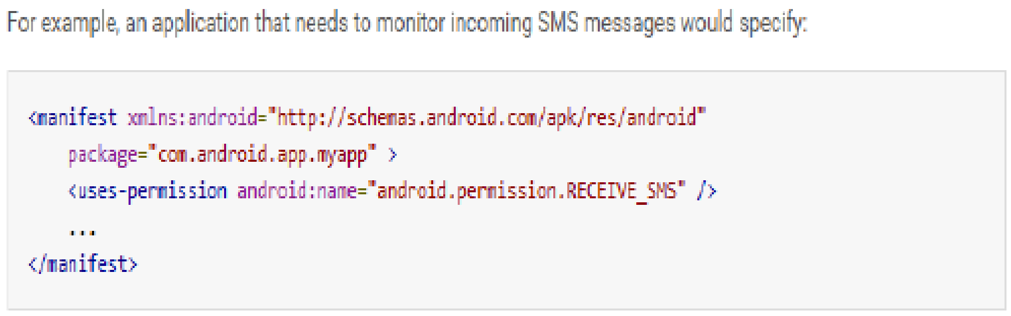
Permission

Permission is a security mechanism that enforces restrictions on the specific operations that a particular process can perform, and per-URI permissions for granting ad hoc access to specific pieces of data.

A permission is a restriction limiting access to a part of the code or to data on the device. The limitation is imposed to protect critical data and code that could be misused to distort or damage the user experience.

A basic Android application has no permissions associated with it by default, meaning it cannot do anything that would adversely impact the user experience or any data on the device. To make use of protected features of the device, we must include one or more <uses-permission> tags declaring the permissions in the "AndroidManifest.xml" file.

Then, when the application is installed on the device, the installer determines whether or not to grant the requested permission by checking the authorities that signed the application's certificates and, in some cases, asking the user. If the permission is granted, the application is able to use the protected features. If not, its attempts to access those features will simply fail without any notification to the user.



For example, an application that needs to monitor incoming SMS messages would specify:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.android.app.myapp" >
    <uses-permission android:name="android.permission.RECEIVE_SMS" />
    ...
</manifest>
```

Here, “RECEIVE_SMS” is the permission for receiving SMS.

Some app permissions are:

ACCOUNT_MANAGER

BATTERY_STATS

BLUETOOTH

CAMERA

CALL_PHONE

DEVICE_POWER

FLASHLIGHT

INTERNET

READ_CALENDAR

READ_CONTACTS

READ_SMS

REBOOT

RECEIVE_MMS

RECEIVE_SMS

RECORD_AUDIO

SEND_SMS

SET_ALARM

SET_WALLPAPER

VIBRATE

WRITE_CALENDAR

WRITE_CONTACTS

WRITE_SMS etc.

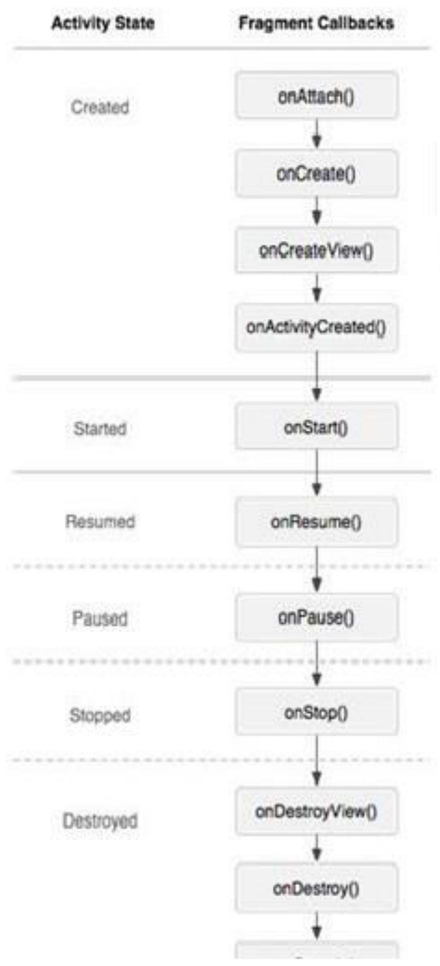
Fragments

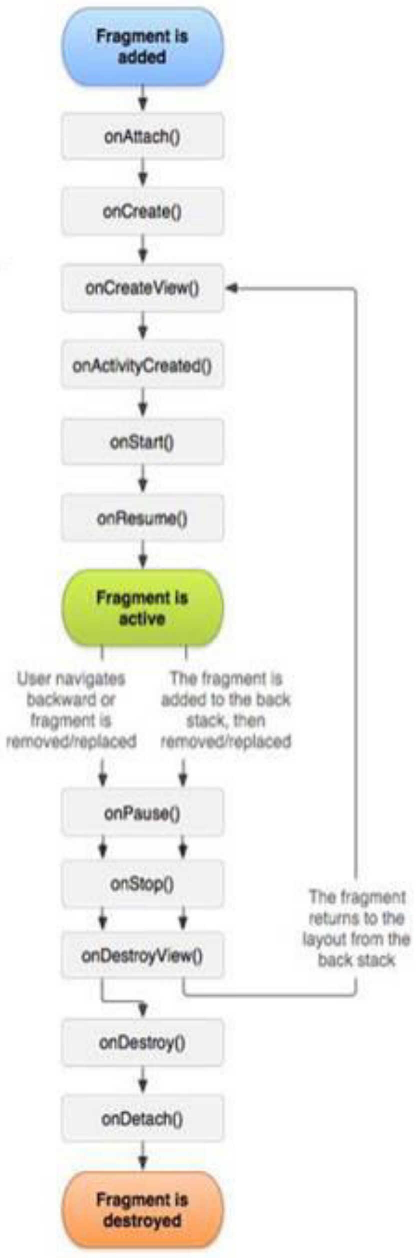
A fragment represents a behavior or a portion of user interface in an activity. Multiple fragments can be combines in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.

A fragment must always be embedded in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle. For example, when the activity is paused, so are all fragments in it, and when the activity is destroyed, so are all fragments.

Life cycle of Fragment

To create a fragment, you must create a subclass of Fragment (or an existing subclass of it). The Fragment class has code that looks a lot like an Activity. It contains callback methods similar to an activity, such as onCreate(), onStart(), onPause(), and onStop()





Usually, you should implement at least the following lifecycle methods:

onCreate()

The system calls this when creating the fragment. Within your implementation, you should initialize essential components of the fragment that you want to retain when the fragment is paused or stopped, then resumed.

onCreateView()

The system calls this when it's time for the fragment to draw its user interface for the first time. To draw a UI for your fragment, you must return a View from this method that is the root of your fragment's layout. You can return null if the fragment does not provide a UI.

onPause()

The system calls this method as the first indication that the user is leaving the fragment (though it does not always mean the fragment is being destroyed). This is usually where you should commit any changes that should be persisted beyond the current user session (because the user might not come back).

Most applications should implement at least these three methods for every fragment, but there are several other callback methods we should also use to handle various stages of the fragment lifecycle. These additional callback methods are:

onAttach()

Called when the fragment has been associated with the activity (the Activity is passed in here).

onActivityCreated()

Called when the activity's onCreate() method has returned.

onDestroyView()

Called when the view hierarchy associated with the fragment is being removed.

onDetach()

Called when the fragment is being disassociated from the activity.

Activity vs Fragment

Activity	**Fragment**
1. An activity is a window with which the user interacts with.	1. A fragment is a part of an activity, which contributes its own UI to that Activity
2. An activity may contain zero or multiple number of fragments.	2. A fragment can be reused in multiple activities, so it acts like a reusable component in activities.
3. An activity can exist without any fragment in it.	3. A fragment has to live inside the activity. It should always be the part of an activity.
6. Activity is needed to be declared in "AndroidManifest.xml"	6. Fragment is not needed to be declared in "AndroidManifest.xml"
7. We cannot have nested activities.	7. We can have nested fragments.

4. Android User Interface

Multiple screen sizes and Orientation Interfaces

For supporting multiple screen sizes and orientation following terms has to be considered:

Screen size:

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

For simplicity, Android groups all actual screen sizes into four generalized sizes: small, normal, large, and extra-large.

Screen density

The quantity of pixels within a physical area of the screen; usually referred to as dpi (dots per inch). For example, a "low" density screen has fewer pixels within a given physical area, compared to a "normal" or "high" density screen.

For simplicity, Android groups all actual screen densities into six generalized densities: low, medium, high, extra-high, extra-extra-high, and extra-extra-extra-high.

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. Be aware that not only do different devices operate in different orientations by default, but the orientation can change at runtime when the user rotates the device.

Resolution

Resolution is 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.

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.

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. At runtime, the system transparently handles any scaling of the dp units, as necessary, based on the actual density of the screen in use. The conversion of dp units to screen pixels is simple:

$$px = dp * (dpi / 160).$$

For example, on a 240 dpi screen, 1 dp equals 1.5 physical pixels. You should always use dp units when defining your application's UI, to ensure proper display of your UI on screens with different densities.

How to Support Multiple Screens

The foundation of Android's support for multiple screens is its ability to manage the rendering of an application's layout and bitmap drawables in an appropriate way for the current screen configuration. The system handles most of the work to render your application properly on each screen configuration by scaling layouts to fit the screen size/density and scaling bitmap drawables for the screen density, as appropriate. To more gracefully handle different screen configurations, however, you should also:

- **Explicitly declare in the manifest which screen sizes your application supports**

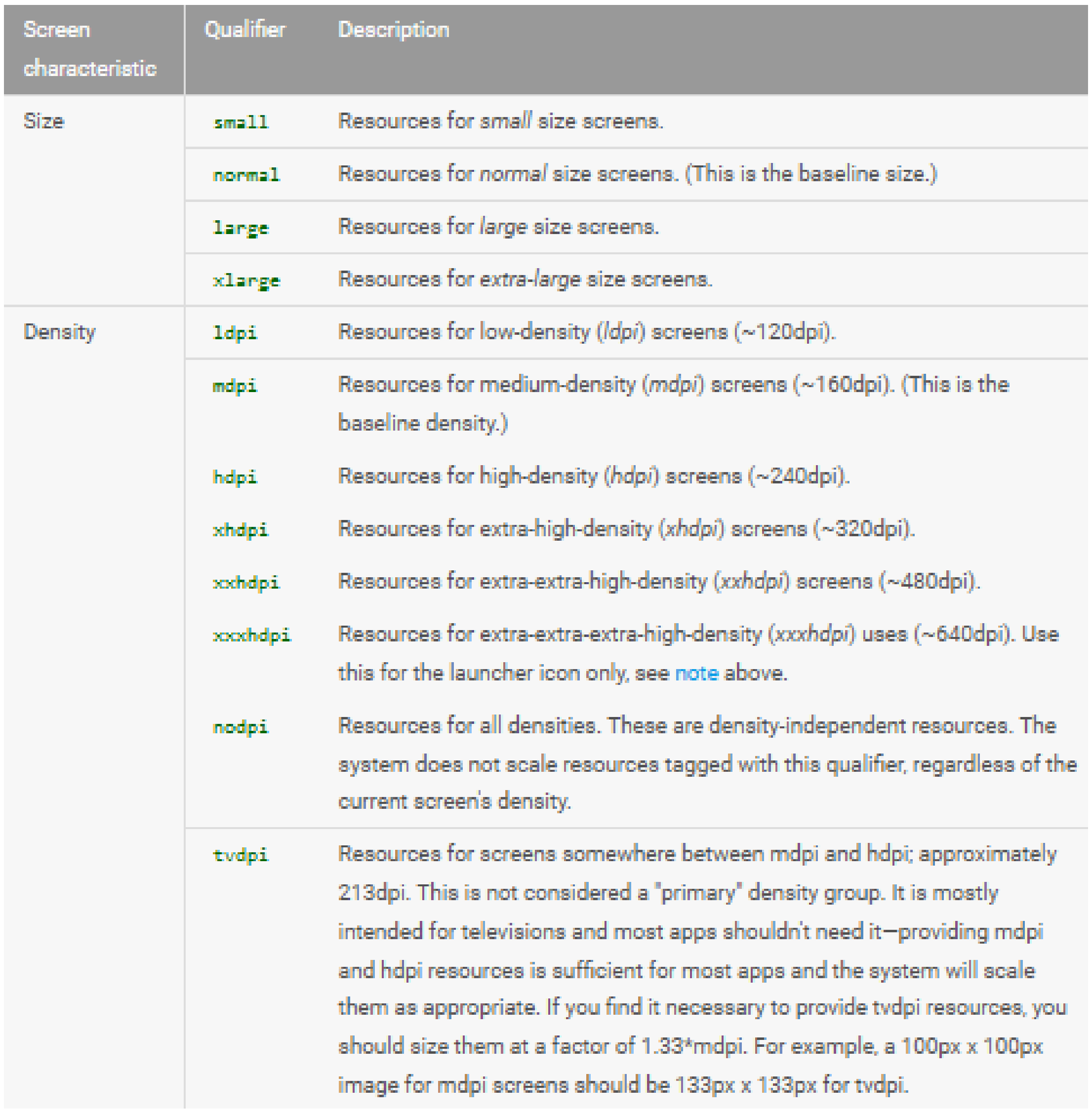
  By declaring which screen sizes your application supports, you can ensure that only devices with the screens you support can download your application. Declaring support for different screen sizes can also affect how the system draws your application on larger screens—specifically, whether your application runs in screen compatibility mode.

  To declare the screen sizes your application supports, you should include the <supports-screens> element in your manifest file.
- **Provide different layouts for different screen sizes**

  By default, Android resizes your application layout to fit the current device screen. In most cases, this works fine. In other cases, your UI might not look as good and might need adjustments for different screen sizes. For example, on a larger screen, you might want to adjust the position and size of some elements to take advantage of the additional screen space, or on a smaller screen, you might need to adjust sizes so that everything can fit on the screen.

  The configuration qualifiers you can use to provide size-specific resources are small, normal, large, and xlarge. For example, layouts for an extra-large screen should go in layout-xlarge/.
- **Provide different bitmap drawables for different screen densities** By default, Android scales your bitmap drawables (.png, .jpg, and .gif files) and Nine-Patch drawables (.9.png files) so that they render at the appropriate physical size on each device. For example, if your application provides bitmap drawables only for the baseline, medium screen density (mdpi), then the system scales them up when on a high-density screen, and scales them down when on a low-density screen. This scaling can cause artifacts in the bitmaps. To ensure your bitmaps look their best, you should include alternative versions at different resolutions for different screen densities.

  The configuration qualifiers that you can use for density-specific resources are ldpi (low), mdpi (medium), hdpi (high), xhdpi extra-high), xxhdpi (extra-extra-high), and xxxhdpi (extra-extra-extra-high). For example, bitmaps for high-density screens should go in drawable-hdpi/.



Screen characteristic	Qualifier	Description
Size	`small`	Resources for *small* size screens.
	`normal`	Resources for *normal* size screens. (This is the baseline size.)
	`large`	Resources for *large* size screens.
	`xlarge`	Resources for *extra-large* size screens.
Density	`ldpi`	Resources for low-density (*ldpi*) screens (~120dpi).
	`mdpi`	Resources for medium-density (*mdpi*) screens (~160dpi). (This is the baseline density.)
	`hdpi`	Resources for high-density (*hdpi*) screens (~240dpi).
	`xhdpi`	Resources for extra-high-density (*xhdpi*) screens (~320dpi).
	`xxhdpi`	Resources for extra-extra-high-density (*xxhdpi*) screens (~480dpi).
	`xxxhdpi`	Resources for extra-extra-extra-high-density (*xxxhdpi*) uses (~640dpi). Use this for the launcher icon only, see note above.
	`nodpi`	Resources for all densities. These are density-independent resources. The system does not scale resources tagged with this qualifier, regardless of the current screen's density.
	`tvdpi`	Resources for screens somewhere between mdpi and hdpi; approximately 213dpi. This is not considered a "primary" density group. It is mostly intended for televisions and most apps shouldn't need it—providing mdpi and hdpi resources is sufficient for most apps and the system will scale them as appropriate. If you find it necessary to provide tvdpi resources, you should size them at a factor of 1.33*mdpi. For example, a 100px x 100px image for mdpi screens should be 133px x 133px for tvdpi.

We can also use following methods to support multiple screen sizes and orientation:

1. Use “wrap_content” and “match_parent”
2. Use Relative Layout
3. Use Size Qualifiers
4. Use the Smallest-width Qualifier
5. Use Layout Aliases
6. Use Orientation Qualifiers
7. Use Nine-patch Bitmaps

1. Use “wrap_content” and “match_parent”

To ensure that your layout is flexible and adapts to different screen sizes, you should use "wrap_content" and "match_parent" for the width and height of some view components. If you use "wrap_content", the width or height of the view is set to the minimum size necessary to fit the content within that view, while "match_parent" (also known as "fill_parent" before API level 8) makes the component expand to match the size of its parent view.

By using the "wrap_content" and "match_parent" size values instead of hard-coded sizes, your views either use only the space required for that view or expand to fill the available space, respectively.
For example, this is what this layout looks like in portrait and landscape mode. Notice that the sizes of the components adapt automatically to the width and height:

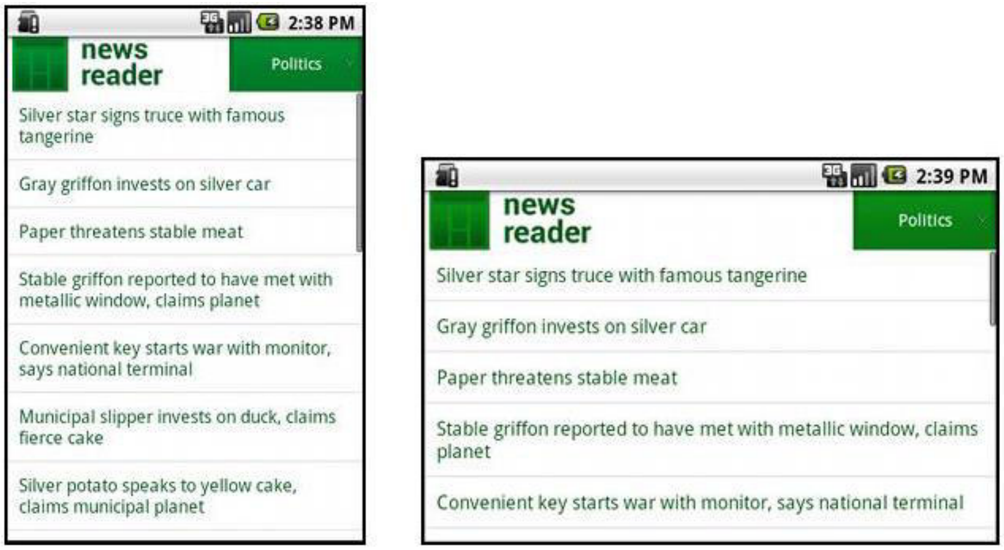

2:38 PM
news reader
Politics
Silver star signs truce with famous tangerine
Gray griffon invests on silver car
Paper threatens stable meat
Stable griffon reported to have met with metallic window, claims planet
Convenient key starts war with monitor, says national terminal
Municipal slipper invests on duck, claims fierce cake
Silver potato speaks to yellow cake, claims municipal planet
2:39 PM

Figure 1. The News Reader sample app in portrait (left) and landscape (right).

2. Use Relative Layout

You can construct fairly complex layouts using nested instances of LinearLayout and combinations of "wrap_content" and "match_parent" sizes. However, LinearLayout does not allow you to precisely control the spacial relationships of child views; views in a LinearLayout simply line up side-by-side. If you need child views to be oriented in variations other than a straight line, a better solution is often to use a RelativeLayout, which allows you to specify your layout in terms of the spacial relationships between components. For instance, you can align one child view on the left side and another view on the right side of the screen.

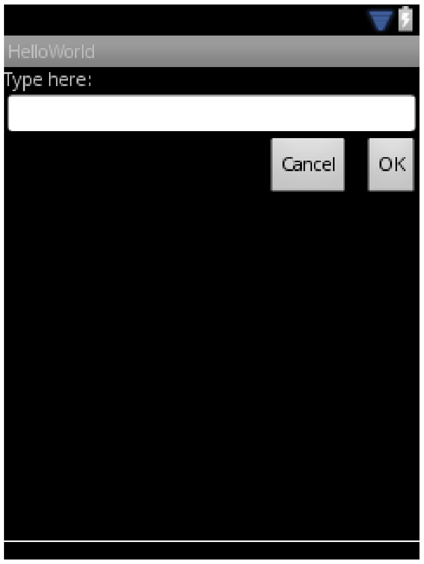


Figure. Screenshot on a small screen

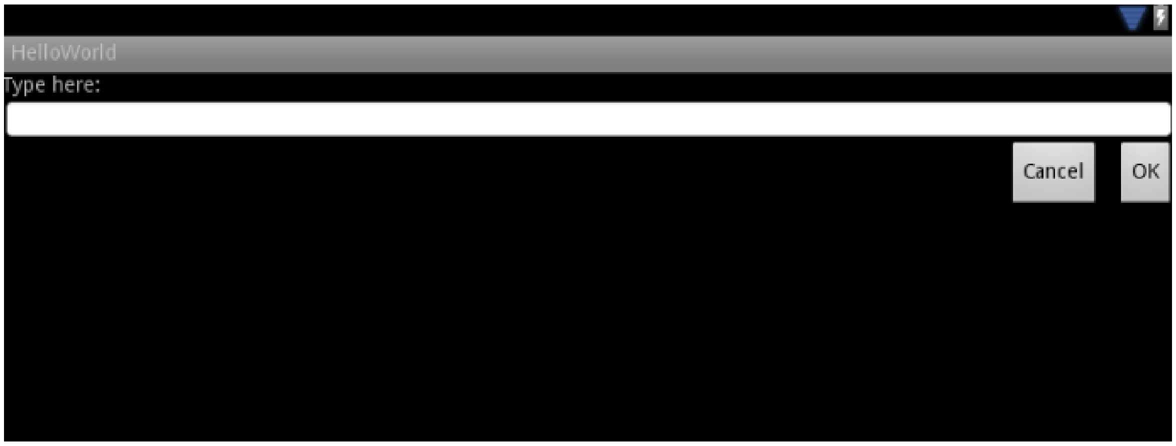


Figure. Screenshot on a large screen

User Interface (UI)

All user interface elements in an Android app are built using View and ViewGroup objects.

A View is a widget that has an appearance on screen that the user can interact with. Examples of widgets are buttons, labels, text boxes, etc. A View derives from the base class

android.view.View

A ViewGroup (which is by itself is a special type of View) provides the layout in which you can order the appearance and sequence of views. It provides invisible container that hold other Views or other ViewGroups and define their layout properties. Examples of Viewgroups are LinearLayout, FrameLayout, etc. A ViewGroup derives from the base class

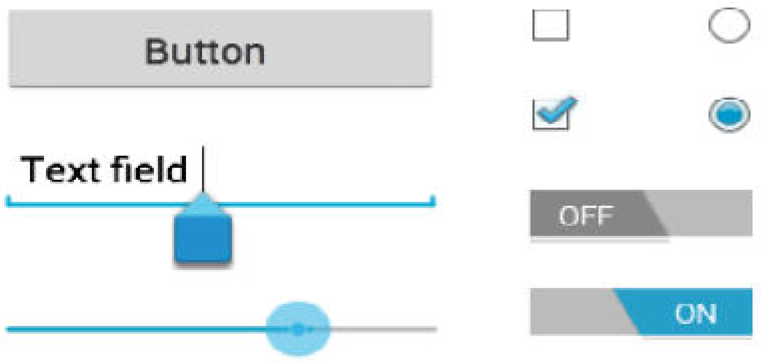
android.view.ViewGroup.

UI Controls/Elements

There are number of UI controls provided by Android that allow you to build the graphical user interface for your app. Some of them are as follows:

- TextView

  This control is used to display text to the user.



- EditText

  EditText is a predefined subclass of TextView that includes rich editing capabilities.
- Button

  A push-button that can be pressed, or clicked, by the user to perform an action.
- ImageButton

  AbsoluteLayout enables you to specify the exact location of its children.
- CheckBox

An on/off switch that can be toggled by the user. You should use check box when presenting users with a group of selectable options that are not mutually exclusive.

- ToggleButton

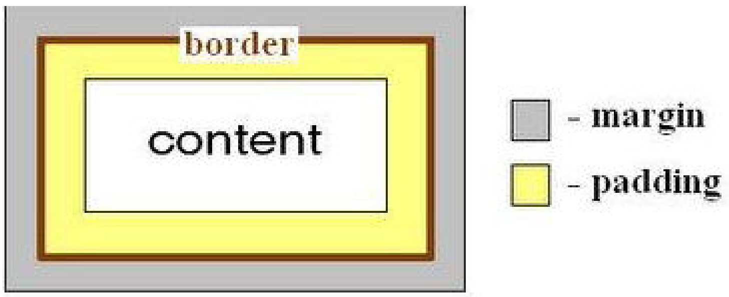
  An on/off button with a light indicator.
- RadioGroup

  A RadioGroup is used to group together one or more RadioButtons.
- ProgressBar The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.

Differentiate between:

1. Padding vs. Margin

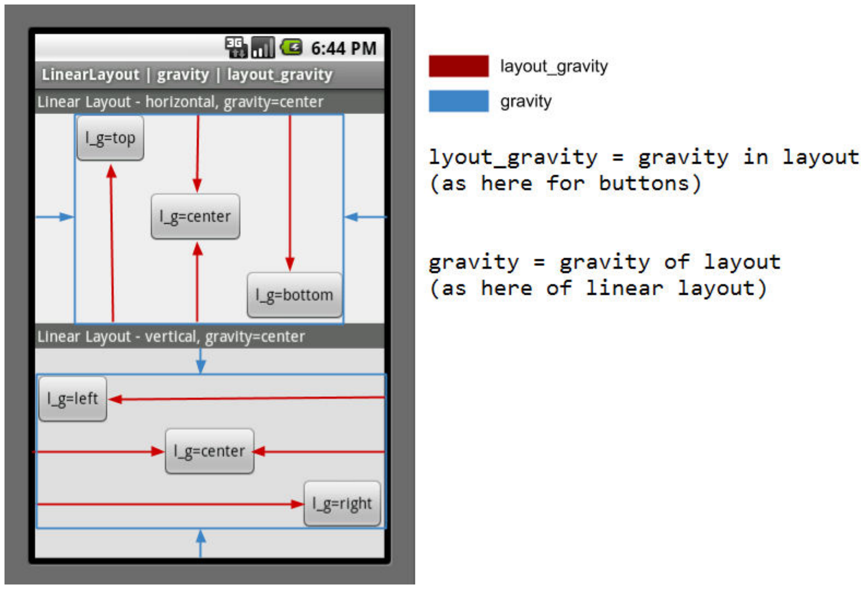
- **Padding** is the space inside the border, between the border and the actual view's content. Note that padding goes completely around the content: there is padding on the top, bottom, right and left sides (which can be independent).



- **Margins** are the spaces outside the border, between the border and the other elements next to this view. In the image, the margin is the grey area outside the entire object. Note that, like the padding, the margin goes completely around the content: there are margins on the top, bottom, right, and left sides.

2. Gravity vs. Layout_Gravity

- **android:gravity** : sets the gravity of the content of the View its used on.
- **android:layout_gravity** : sets the gravity of the View or Layout in its parent.


6:44 PM
LinearLayout | gravity | layout_gravity
Linear Layout - horizontal, gravity=center
l_g=top
l_g=center
l_g=bottom
Linear Layout - vertical, gravity=center
l_g=left
l_g=center
l_g=right
layout_gravity
gravity
lyout_gravity = gravity in layout
(as here for buttons)
gravity = gravity of layout
(as here of linear layout)

Android XML

XML stands for Extensible Mark-up Language.XML is a very popular format and commonly used for sharing data on the internet.

Android provides three types of XML parsers which are **DOM,SAX and XMLPullParser**. Among all of them android recommend XMLPullParser because it is efficient and easy to use. So we are going to use XMLPullParser for parsing XML

The first step is to identify the fields in the XML data in which you are interested in. For example. In the XML given below we interested in getting temperature only.

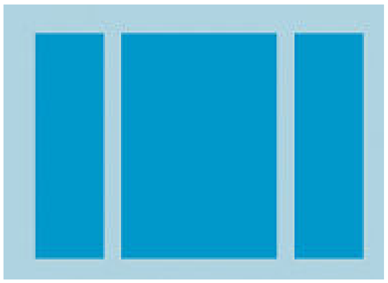
```
<?xml version="1.0"?>
<current>

  <city id="2643743" name="London">
    <coord lon="-0.12574" lat="51.50853"/>
    <country>GB</country>
    <sun rise="2013-10-08T06:13:56" set="2013-10-08T17:21:45"/>
  </city>

  <temperature value="289.54" min="289.15" max="290.15" unit="kelvin"/>
  <humidity value="77" unit="%"/>
  <pressure value="1025" unit="hPa"/>
</country>
```

Android Layout:

Android supports the following layouts models:

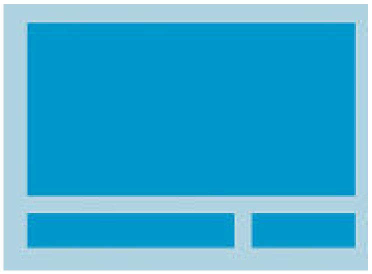
1. Linear Layout

A Layout that arranges its children in a single column or a single row. The direction of the row can be set by calling setOrientation(). You can also specify gravity, which specifies the alignment of all the child elements by calling setGravity() or specify that specific children grow to fill up any remaining space in the layout by setting the *weight* member of Linear Layout. Layout Params. The default orientation is horizontal.



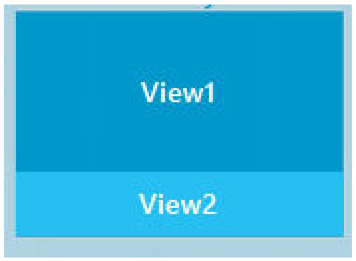
2. Relative Layout

Relative Layout is a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent Relative Layout area (such as aligned to the bottom, left or center).



3. Frame Layout

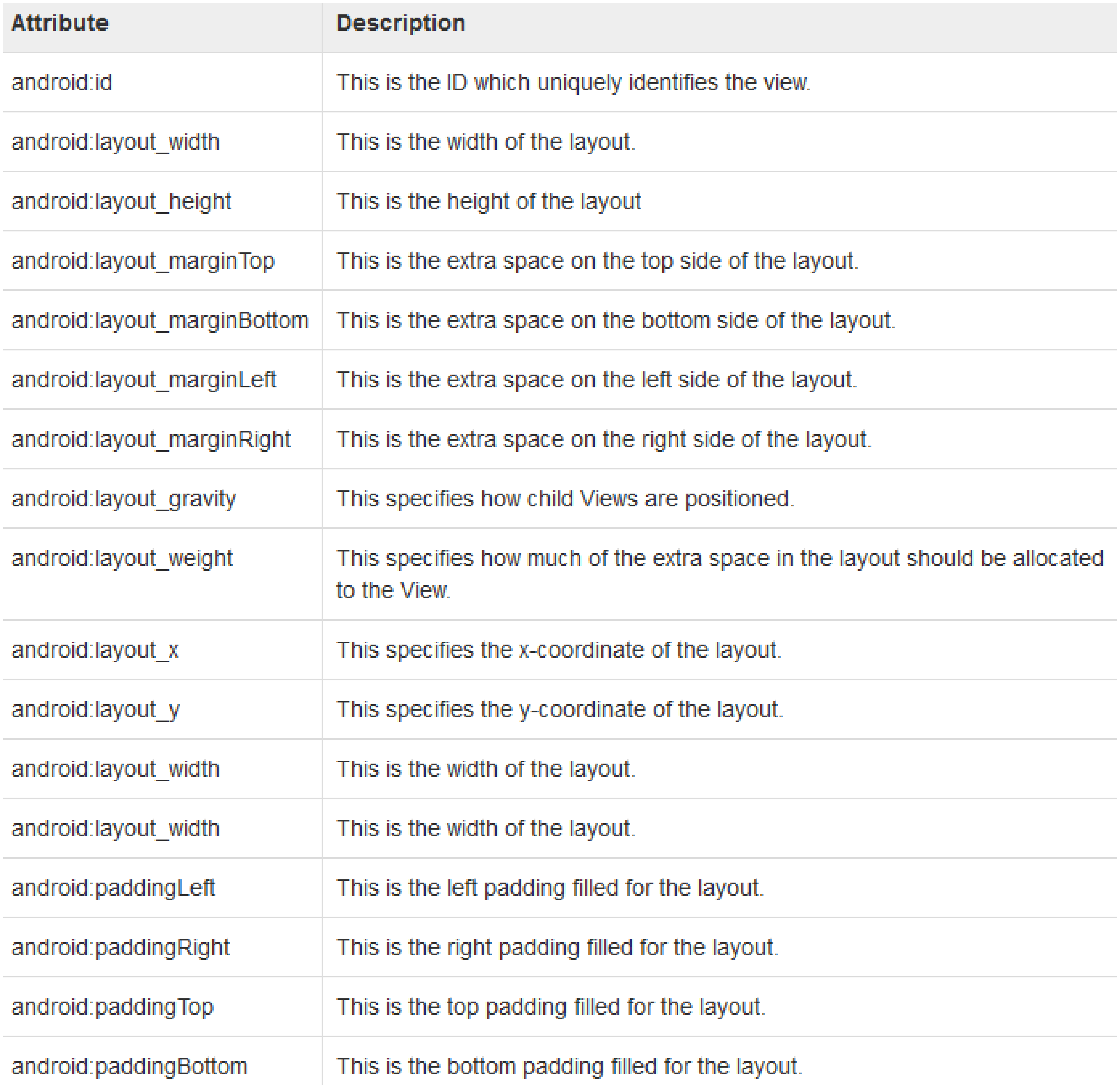
Frame Layout is designed to block out an area on the screen to display a single item. Generally, Frame Layout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other.



The Other layout android supports are:

Sr.No	Layout & Description
1	**Linear Layout** LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	**Relative Layout** RelativeLayout is a view group that displays child views in relative positions.
3	**Table Layout** TableLayout is a view that groups views into rows and columns.
4	**Absolute Layout** AbsoluteLayout enables you to specify the exact location of its children.
5	**Frame Layout** The FrameLayout is a placeholder on screen that you can use to display a single view.
6	**List View** ListView is a view group that displays a list of scrollable items.
7	**Grid View** GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

Different types of layout properties are:



Attribute	Description
android:id	This is the ID which uniquely identifies the view.
android:layout_width	This is the width of the layout.
android:layout_height	This is the height of the layout
android:layout_marginTop	This is the extra space on the top side of the layout.
android:layout_marginBottom	This is the extra space on the bottom side of the layout.
android:layout_marginLeft	This is the extra space on the left side of the layout.
android:layout_marginRight	This is the extra space on the right side of the layout.
android:layout_gravity	This specifies how child Views are positioned.
android:layout_weight	This specifies how much of the extra space in the layout should be allocated to the View.
android:layout_x	This specifies the x-coordinate of the layout.
android:layout_y	This specifies the y-coordinate of the layout.
android:layout_width	This is the width of the layout.
android:layout_width	This is the width of the layout.
android:paddingLeft	This is the left padding filled for the layout.
android:paddingRight	This is the right padding filled for the layout.
android:paddingTop	This is the top padding filled for the layout.
android:paddingBottom	This is the bottom padding filled for the layout.

Resources

Resources are the additional files and static content that your code uses, such as bitmaps, layout definitions, user interface strings and animation instructions. The Android resource system keeps track of all non-code assets associated with an application.

For any type of resource, you can specify *default* and multiple *alternative* resources for your application:

- Default resources are those that should be used regardless of the device configuration or when there are no alternative resources that match the current configuration.

- Alternative resources are those that you've designed for use with a specific configuration. To specify that a group of resources are for a specific configuration, append an appropriate configuration qualifier to the directory name.

The **res/** directory contains all the resources in various sub directories. Here we have an image resource, two layout resources, and a string resource file. Following table gives a detail about the resource directories supported inside project res/ directory.

Directory	Resource Type
anim/	XML files that define property animations. They are saved in res/anim/ folder and accessed from the **R.anim** class.
color/	XML files that define a state list of colors. They are saved in res/color/ and accessed from the **R.color** class.
drawable/	Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawable. They are saved in res/drawable/ and accessed from the **R.drawable** class.
layout/	XML files that define a user interface layout. They are saved in res/layout/ and accessed from the **R.layout** class.
menu/	XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. They are saved in res/menu/ and accessed from the **R.menu** class.
raw/	Arbitrary files to save in their raw form. You need to call*Resources.openRawResource()* with the resource ID, which is*R.raw.filename* to open such raw files.
values/	XML files that contain simple values, such as strings, integers, and colors. For example, here are some filename conventions for resources you can create in this directory − • arrays.xml for resource arrays, and accessed from the **R.array** class. • integers.xml for resource integers, and accessed from the **R.integer** class. • bools.xml for resource boolean, and accessed from the **R.bool** class. • colors.xml for color values, and accessed from the**R.color** class. • dimens.xml for dimension values, and accessed from the **R.dimen** class. • strings.xml for string values, and accessed from the**R.string** class. • styles.xml for styles, and accessed from the **R.style**class.
xml/	Arbitrary XML files that can be read at runtime by calling*Resources.getXML()*. You can save various configuration files here which will be used at run time.

Android Style

A **style** is a collection of properties that specify the look and format for a View or window. A style can specify properties such as height, padding, font color, font size, background color, and much more. A style is defined in an XML resource that is separate from the XML that specifies the layout.

Styles in Android share a similar philosophy to cascading stylesheets in web design—they allow you to separate the design from the content.

For example, by using a style, you can take this layout XML:

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:textColor="#00FF00"
    android:typeface="monospace"
    android:text="@string/hello" />
```

A **theme** is a style applied to an entire Activity or application, rather than an individual View (as in the example above). When a style is applied as a theme, every View in the Activity or application will apply each style property that it supports.

To create a set of styles, save an XML file in the res/values/ directory of android project. The name of the XML file is arbitrary, but it must use the .xml extension and be saved in the res/values/ folder.

The root node of the XML file must be <resources>.

For each style we want to create, add a <style> element to the file with a name that uniquely identifies the style (this attribute is required). Then add an <item> element for each property of that style, with a name that declares the style property and a value to go with it (this attribute is required). The value for the <item> can be a keyword string, a hex color, a reference to another resource type, or other value depending on the style property. Here's an example file with a single style:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CodeFont" parent="@android:style/TextAppearance.Medium">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:textColor">#00FF00</item>
        <item name="android:typeface">monospace</item>
    </style>
</resources>
```

Android Third party UI

Google wants Android to be in everything that has web connectivity and to that effect, the company has unveiled what third party apps might one day look like inside your car. Android for the car will be called Android Auto and the image you see here is what the apps might one day look like in the car, assuming your car supports Android Auto. Google did the same thing with Android Wear where the apps for Wear aren't complete apps, rather they are used to control what content from an app on your smartphone you can see on the wearable device.

There are many third-party libraries for Android but several of them are "must have" libraries that are extremely popular and are often used in almost any Android project. Each has different purposes but all of them make life as a developer much more pleasant.

5.Advanced Topics

Notifications

A notification is a message you can display to the user outside of your application's normal UI. When we tell the system to issue a notification, it first appears as an icon in the **notification area**. To see the details of the notification, the user opens the **notification drawer**. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

Creating a notification

The UI information and actions for a notification is specified in a NotificationCompat.Builder object. To create the notification itself, we call NotificationCompat.Builder.build(), which returns a Notification object containing your specifications. To issue the notification, you pass the Notification object to the system by calling NotificationManager.notify().

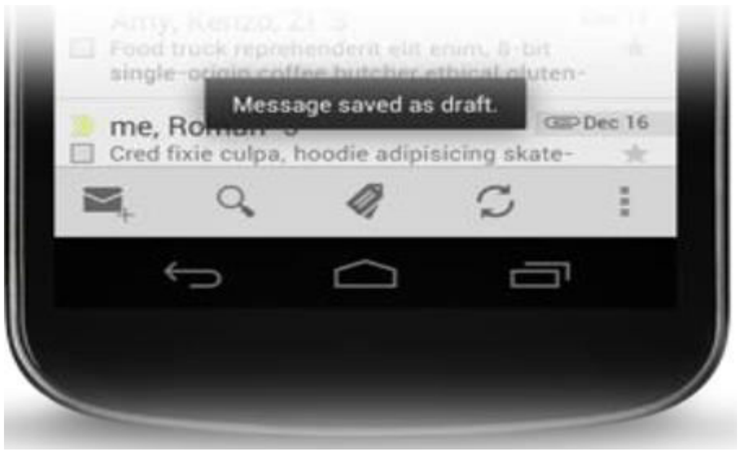
A Notification object *must* contain the following:

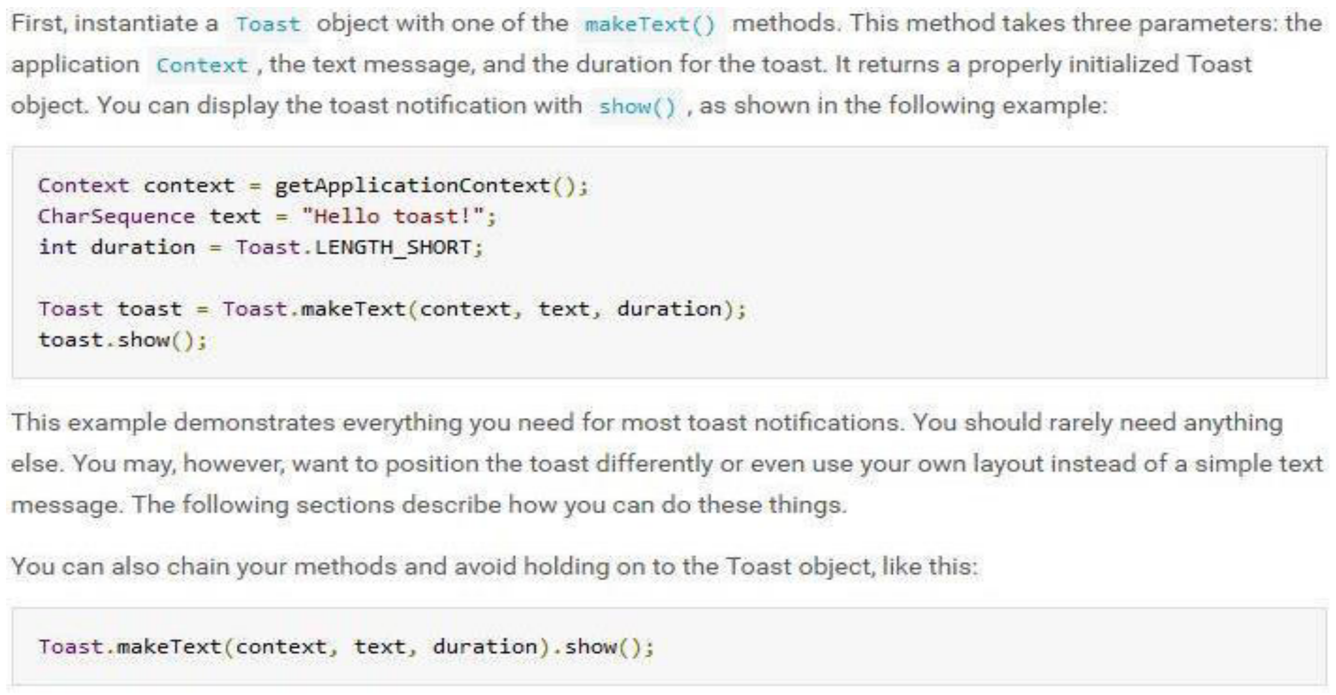
- A small icon, set by setSmallIcon()
- A title, set by setContentTitle()
- Detail text, set by setContentText()

```
NotificationCompat.Builder mBuilder =
    new NotificationCompat.Builder(this)
    .setSmallIcon(R.drawable.notification_icon)
    .setContentTitle("My notification")
    .setContentText("Hello World!");
```

Toast

A toast provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive. For example, navigating away from an email before you send it triggers a "Draft saved" toast to let you know that you can continue editing later. Toasts automatically disappear after a timeout.


Message saved as draft.

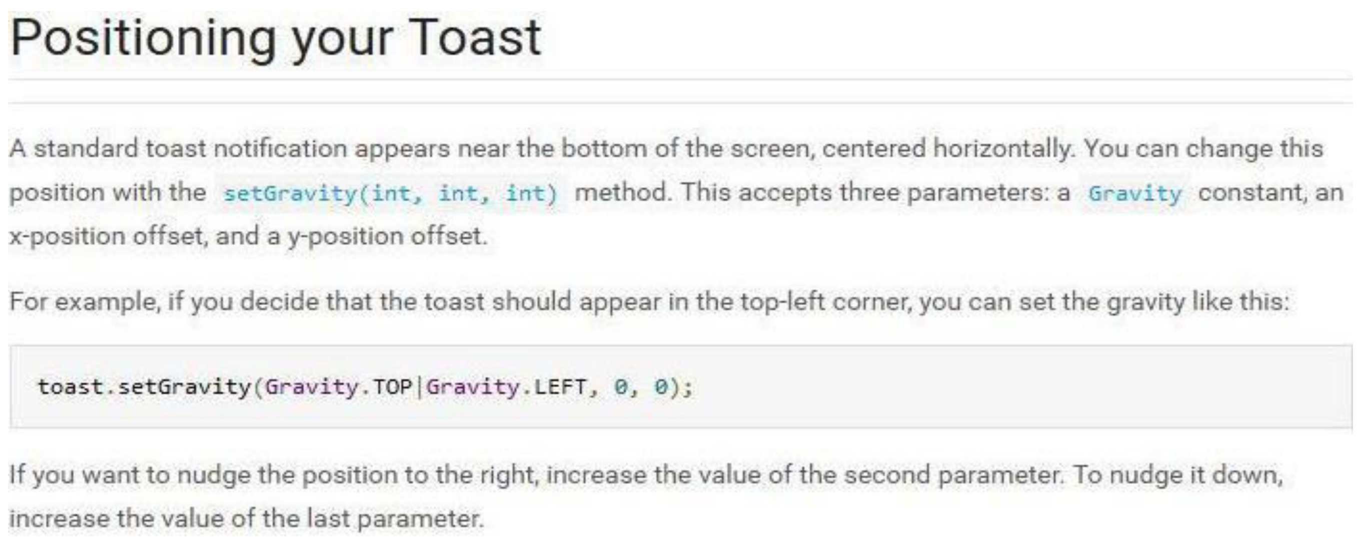

First, instantiate a `Toast` object with one of the `makeText()` methods. This method takes three parameters: the application `Context`, the text message, and the duration for the toast. It returns a properly initialized Toast object. You can display the toast notification with `show()`, as shown in the following example:

```
Context context = getApplicationContext();
CharSequence text = "Hello toast!";
int duration = Toast.LENGTH_SHORT;

Toast toast = Toast.makeText(context, text, duration);
toast.show();
```

This example demonstrates everything you need for most toast notifications. You should rarely need anything else. You may, however, want to position the toast differently or even use your own layout instead of a simple text message. The following sections describe how you can do these things.

You can also chain your methods and avoid holding on to the Toast object, like this:

```
Toast.makeText(context, text, duration).show();
```


Positioning your Toast

A standard toast notification appears near the bottom of the screen, centered horizontally. You can change this position with the `setGravity(int, int, int)` method. This accepts three parameters: a `Gravity` constant, an x-position offset, and a y-position offset.

For example, if you decide that the toast should appear in the top-left corner, you can set the gravity like this:

```
toast.setGravity(Gravity.TOP|Gravity.LEFT, 0, 0);
```

If you want to nudge the position to the right, increase the value of the second parameter. To nudge it down, increase the value of the last parameter.

Broadcast Receiver

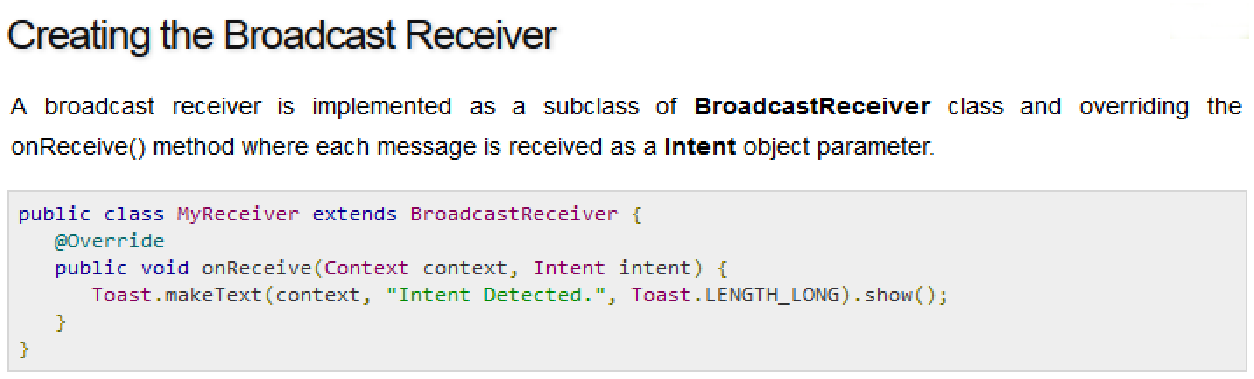
A broadcast receiver (short receiver) is an Android component which allows you to register for system or application events. All registered receivers for an event are notified by the Android runtime once this event happens.

For example, applications can register for the ACTION_BOOT_COMPLETED system event which is fired once the Android system has completed the boot process.

Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

There are following two important steps to make BroadcastReceiver works for the system broadcasted intents

- Creating the Broadcast Receiver

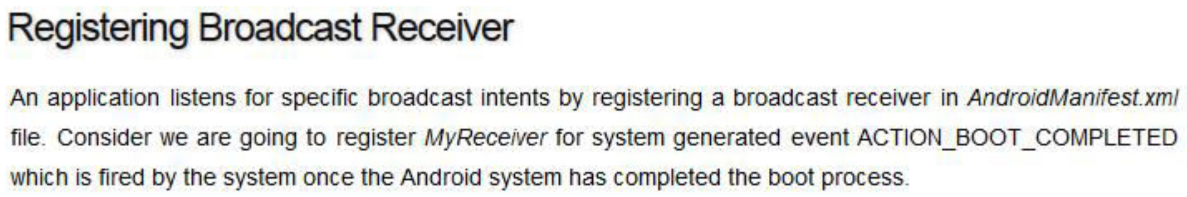


Creating the Broadcast Receiver

A broadcast receiver is implemented as a subclass of **BroadcastReceiver** class and overriding the onReceive() method where each message is received as a **Intent** object parameter.

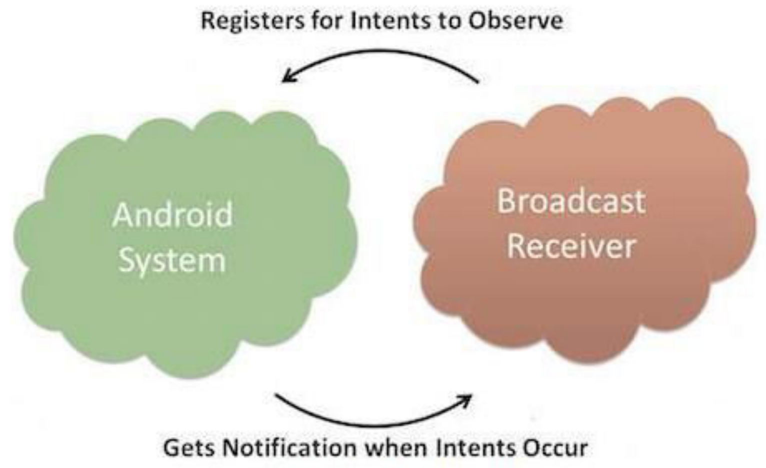
```
public class MyReceiver extends BroadcastReceiver {
   @Override
   public void onReceive(Context context, Intent intent) {
      Toast.makeText(context, "Intent Detected.", Toast.LENGTH_LONG).show();
   }
}
```

- Registering Broadcast Receiver



Registering Broadcast Receiver

An application listens for specific broadcast intents by registering a broadcast receiver in *AndroidManifest.xml* file. Consider we are going to register *MyReceiver* for system generated event ACTION_BOOT_COMPLETED which is fired by the system once the Android system has completed the boot process.


Registers for Intents to Observe
Android System
Broadcast Receiver
Gets Notification when Intents Occur

BROADCAST-RECEIVER

Threads and Handlers

Thread

A Thread is a concurrent unit of execution. It has its own call stack for methods being invoked, their arguments and local variables. Each application has at least one thread running when it is started, the main thread, in the main “ThreadGroup”. The runtime keeps its own threads in the system thread group.

There are two ways to execute code in a new thread.

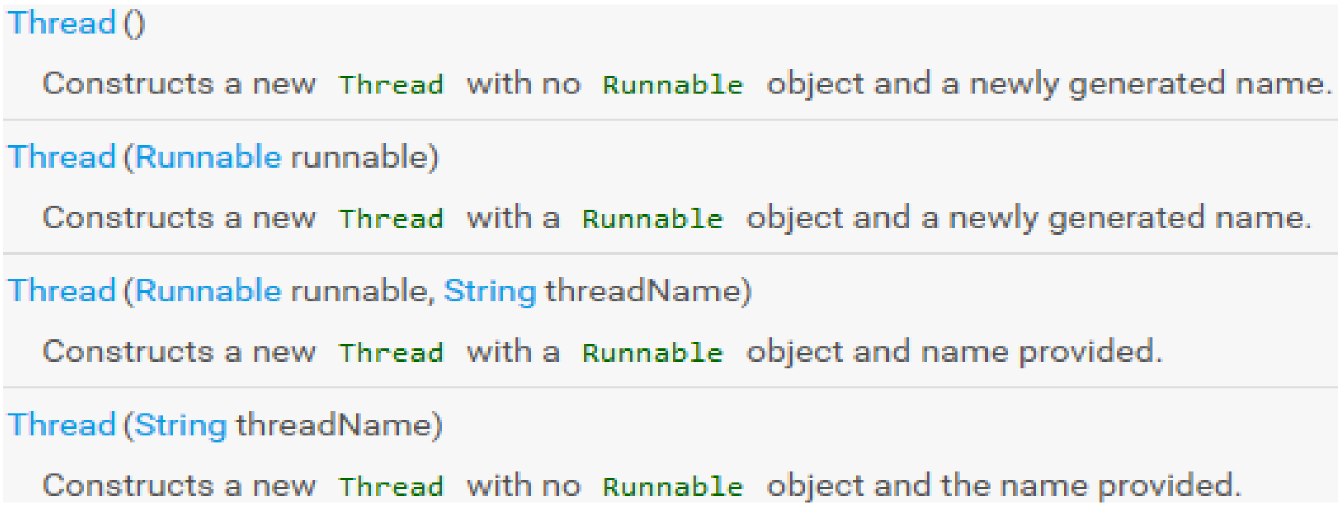
We can either subclass Thread and overriding its run() method.

Or

Construct a new Thread and pass a Runnable to the constructor.

In either case, the start() method must be called to actually execute the new Thread.

Some Thread Syntax:



Thread ()

Constructs a new `Thread` with no `Runnable` object and a newly generated name.

Thread (Runnable runnable)

Constructs a new `Thread` with a `Runnable` object and a newly generated name.

Thread (Runnable runnable, String threadName)

Constructs a new `Thread` with a `Runnable` object and name provided.

Thread (String threadName)

Constructs a new `Thread` with no `Runnable` object and the name provided.

Handler

Handler is part of the Android system's framework for managing threads. A Handler object receives messages and runs code to handle the messages. A Handler allows us to send and process Message and Runnable objects associated with a thread's MessageQueue.

There are two main uses for a Handler:

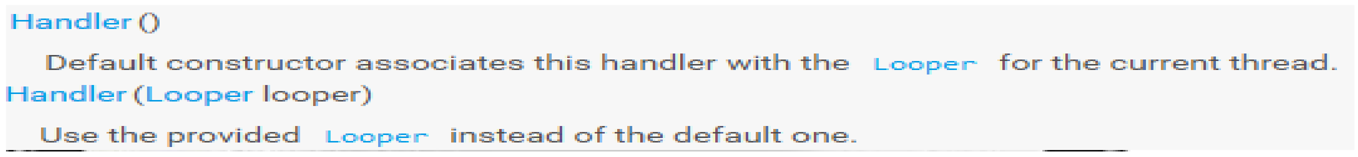
- to schedule messages and runnables to be executed as some point in the future
- to enqueue an action to be performed on a different thread than your own

Scheduling messages is accomplished with the following methods.

- post(Runnable),
- postAtTime(Runnable, long),
- postDelayed(Runnable, long),
- sendEmptyMessage(int),
- sendMessage(Message),
- sendMessageAtTime(Message, long), and
- sendMessageDelayed(Message, long)

The *post* versions allow you to enqueue Runnable objects to be called by the message queue when they are received; the *sendMessage* versions allow you to enqueue a Message object containing a bundle of data that will be processed by the Handler's handleMessage(Message) method (requiring that you implement a subclass of Handler). When posting or sending to a Handler, you can either allow the item to be processed as soon as the message queue is ready to do so, or specify a delay before it gets processed or absolute time for it to be processed. The latter two allow you to implement timeouts, ticks, and other timing-based behavior.

Some Handler Syntax:



Handler ()

Default constructor associates this handler with the `Looper` for the current thread.

Handler (Looper looper)

Use the provided `Looper` instead of the default one.

AsyncTask

AsyncTask enables proper and easy use of the UI thread. This class allows to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

AsyncTask is designed to be a helper class around `Thread` and `Handler` and does not constitute a generic threading framework. AsyncTasks should ideally be used for short operations (a few seconds at the most.) If you need to keep threads running for long periods of time, it is highly recommended you use the various APIs provided by the `java.util.concurrent` package such as `Executor`, `ThreadPoolExecutor` and `FutureTask`.

An asynchronous task is defined by a computation that runs on a background thread and whose result is published on the UI thread. An asynchronous task is defined by 3 generic types, called `Params`, `Progress` and `Result`, and 4 steps, called `onPreExecute`, `doInBackground`, `onProgressUpdate` and `onPostExecute`.

AsyncTask's generic types

The three types used by an asynchronous task are the following:

1. `Params`, the type of the parameters sent to the task upon execution.
2. `Progress`, the type of the progress units published during the background computation.
3. `Result`, the type of the result of the background computation.

Not all types are always used by an asynchronous task. To mark a type as unused, simply use the type `Void`:

```
private class MyTask extends AsyncTask<Void, Void, Void> { ... }
```

The 4 steps

When an asynchronous task is executed, the task goes through 4 steps:

1. `onPreExecute()`, invoked on the UI thread before the task is executed. This step is normally used to setup the task, for instance by showing a progress bar in the user interface.
2. `doInBackground(Params...)`, invoked on the background thread immediately after `onPreExecute()` finishes executing. This step is used to perform background computation that can take a long time. The parameters of the asynchronous task are passed to this step. The result of the computation must be returned by this step and will be passed back to the last step. This step can also use `publishProgress(Progress...)` to publish one or more units of progress. These values are published on the UI thread, in the `onProgressUpdate(Progress...)` step.
3. `onProgressUpdate(Progress...)`, invoked on the UI thread after a call to `publishProgress(Progress...)`. The timing of the execution is undefined. This method is used to display any form of progress in the user interface while the background computation is still executing. For instance, it can be used to animate a progress bar or show logs in a text field.
4. `onPostExecute(Result)`, invoked on the UI thread after the background computation finishes. The result of the background computation is passed to this step as a parameter.

Alarm Manager

Alarms (based on the AlarmManager class) gives a way to perform time-based operations outside the lifetime of your application. For example, we could use an alarm to initiate a long-running operation, such as starting a service once a day to download a weather forecast.

Alarms have these characteristics:

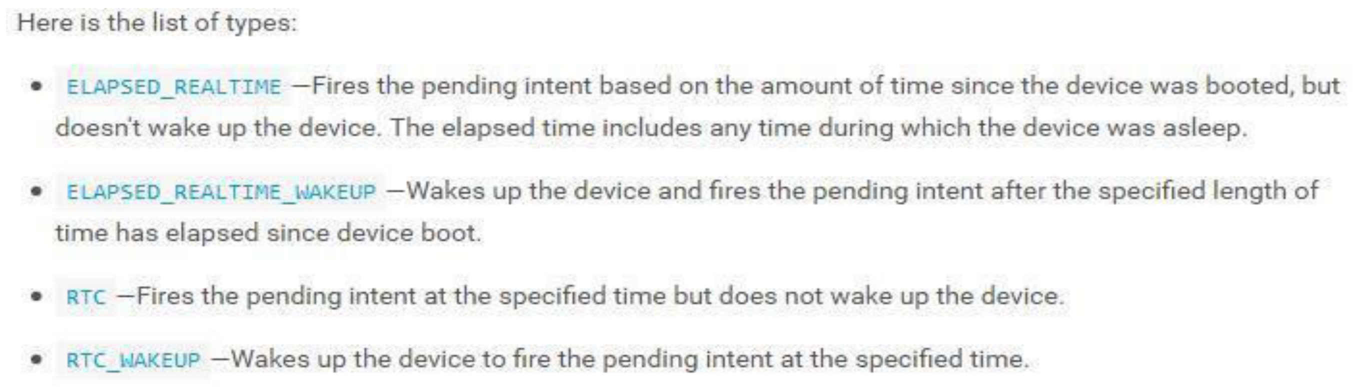
- They let us fire Intents at set times and/or intervals.
- We can use them in conjunction with broadcast receivers to start services and perform other operations.
- They operate outside of your application, so we can use them to trigger events or actions even when your app is not running, and even if the device itself is asleep.
- They help us to minimize your app's resource requirements. We can schedule operations without relying on timers or continuously running background services.

Note: For timing operations that are guaranteed to occur *during* the lifetime of your application, instead consider using the Handler class in conjunction with Timer and Thread. This approach gives Android better control over system resources.

Alarm Types:

There are two general clock types for alarms: "elapsed real time" and "real time clock" (RTC). Elapsed real time uses the "time since system boot" as a reference, and real time clock uses UTC (wall clock) time. This means that elapsed real time is suited to setting an alarm based on the passage of time (for example, an alarm that fires every 30 seconds) since it isn't affected by time zone/locale. The real time clock type is better suited for alarms that are dependent on current locale.

Both types have a "wakeup" version, which says to wake up the device's CPU if the screen is off. This ensures that the alarm will fire at the scheduled time. This is useful if your app has a time dependency—for example, if it has a limited window to perform a particular operation. If you don't use the wakeup version of your alarm type, then all the repeating alarms will fire when your device is next awake.



Here is the list of types:

- `ELAPSED_REALTIME` —Fires the pending intent based on the amount of time since the device was booted, but doesn't wake up the device. The elapsed time includes any time during which the device was asleep.
- `ELAPSED_REALTIME_WAKEUP` —Wakes up the device and fires the pending intent after the specified length of time has elapsed since device boot.
- `RTC` —Fires the pending intent at the specified time but does not wake up the device.
- `RTC_WAKEUP` —Wakes up the device to fire the pending intent at the specified time.

Networking in Android

Android lets your application connect to the internet or any other local network and allows you to perform network operations.

A device can have various types of network connections. This chapter focuses on using either a Wi-Fi or a mobile network connection.

Checking Network Connection

Before you perform any network operations, you must first check that are you connected to that network or internet e.t.c. For this android provides**ConnectivityManager** class. You need to instantiate an object of this class by calling **getSystemService()** method. Its syntax is given below −

```
ConnectivityManager check = (ConnectivityManager)
this.context.getSystemService(Context.CONNECTIVITY_SERVICE);
```

Once you instantiate the object of ConnectivityManager class, you can use**getAllNetworkInfo** method to get the information of all the networks. This method returns an array of **NetworkInfo**. So you have to receive it like this.

```
NetworkInfo[] info = check.getAllNetworkInfo();
```

The last thing you need to do is to check **Connected State** of the network. Its syntax is given below −

```
for (int i = 0; i<info.length; i++){
  if (info[i].getState() == NetworkInfo.State.CONNECTED){
    Toast.makeText(context, "Internet is connected
    Toast.LENGTH_SHORT).show();
  }
}
```

Apart from this connected states, there are other states a network can achieve. They are listed below:

Sr.No	State
1	Connecting
2	Disconnected
3	Disconnecting
4	Suspended
5	Unknown

Performing Network Operations

After checking that you are connected to the internet, you can perform any network operation. Here we are fetching the html of a website from a url.

Android provides **HttpURLConnection** and **URL** class to handle these operations. You need to instantiate an object of URL class by providing the link of website. Its syntax is as follows −

```
String link = "http://www.google.com";
URL url = new URL(link);
```

After that you need to call **openConnection** method of url class and receive it in a HttpURLConnection object. After that you need to call the **connect** method of HttpURLConnection class.

```
HttpURLConnection conn = (HttpURLConnection) url.openConnection();
conn.connect();
```

And the last thing you need to do is to fetch the HTML from the website. For this you will use **InputStream** and **BufferedReader** class. Its syntax is given below −

```
InputStream is = conn.getInputStream();
BufferedReader reader =new BufferedReader(new InputStreamReader(is, "UTF-8"));
String webPage = "",data="";

while ((data = reader.readLine()) != null){
  webPage += data + "\n";
}
```

Apart from this connect method, there are other methods available in HttpURLConnection class. They are listed below −

Sr.No	Method & description
1	**disconnect()** This method releases this connection so that its resources may be either reused or closed
2	**getRequestMethod()** This method returns the request method which will be used to make the request to the remote HTTP server
3	**getResponseCode()** This method returns response code returned by the remote HTTP server
4	**setRequestMethod(String method)** This method Sets the request command which will be sent to the remote HTTP server
5	**usingProxy()** This method returns whether this connection uses a proxy server or not

6. Graphics and Multimedia

Graphics

If one talks about standard components that we can be used in UI, this is good but it is not enough when we want to develop a game or an app that requires graphic contents. Android SDK provides a set of API for drawing custom 2D and 3D graphics. When we write an app that requires graphics, we should consider how intensive the graphic usage is. In other words, there could be an app that uses quite static graphics without complex effects and there could be other app that uses intensive graphical effects like games. According to this usage, there are different techniques we can adopt:

- **Canvas and Drawable:** In this case, we can extend the existing UI widgets so that we can customize their behavior or we can create custom 2D graphics using the standard method provided by the Canvas class.
- **Hardware acceleration:** We can use hardware acceleration when drawing with the Canvas API. This is possible from Android 3.0.
- **OpenGL:** Android supports OpenGL natively using NDK. This technique is very useful when we have an app that uses intensively graphic contents (i.e games).

The easiest way to use 2D graphics is extending the View class and overriding the onDraw method. We can use this technique when we do not need a graphics intensive app.

In this case, we can use the Canvas class to create 2D graphics. This class provides a set of method starting with draw that can be used to draw different shapes like:

- lines
- circle
- rectangle
- oval
- picture
- arc

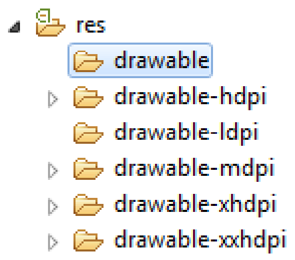
For example let us suppose we want do draw a rectangle. We create a custom view and then we override onDraw method. Here we draw the rectangle:

```
public class TestView extends View {
public TestView(Context context) {
super(context);
}
public TestView(Context context, AttributeSet attrs, int defStyle) {
super(context, attrs, defStyle);
}
public TestView(Context context, AttributeSet attrs) {
```

```
super(context, attrs);
}
@Override
protected void onDraw(Canvas canvas) {
super.onDraw(canvas);
Paint p = new Paint();
p.setColor(Color.GREEN);
p.setStrokeWidth(1);
```

Drawable

In Android, a Drawable is a graphical object that can be shown on the screen. From API point of view all the Drawable objects derive from *Drawable* class. They have an important role in Android programming and we can use XML to create them. They differ from standard widgets because they are not interactive, meaning that they do not react to user touch. Images, colors, shapes, objects that change their aspect according to their state, object that can be animated are all drawable objects. In Android under res directory, there is a sub-dir reserved for Drawable, it is called *res/drawable.*



Under the drawable dir we can add binary files like images or XML files.

We can create several directories according to the screen density we want to support. These directories have a name like drawable-<>. This is very useful when we use images; in this case, we have to create several image versions: for example, we can create an image for the high dpi screen or another one for medium dpi screen.

Once we have our file under drawable directory, we can reference it, in our class, using

R.drawable.file_name

While it is very easy add a binary file to one of these directory, it is a matter of copy and paste, if we want to use a XML file we have to create it.

There are several types of drawable:

- Bitmap
- Nine-patch
- State list
- Level list
- Transition drawable
- Inset drawable
- Clip drawable

• Scale drawable

• Shape drawable

Shape drawable

This is a generic shape. Using XML we have to create a file with shape element as root. This element as an attribute called

android:shape

where we define the type of shape like rectangle, oval, line and ring. We can customize the shape using child elements like:

For example, let us suppose we want to create an oval with solid background color. We create a XML file called for example *oval.xml:*

```
<shape xmlns:android="http://schemas.android.com/apk/res/android"
android:shape="oval" >
<solid android:color="#FF0000" />
<size
android:height="100dp"
android:width="120dp" />
</shape>
```

Multi touch and gestures

A multi-touch gesture is when multiple pointers (fingers) touch the screen at the same time. This lesson describes how to detect gestures that involve multiple pointers.

Track Multiple Pointers

When multiple pointers touch the screen at the same time, the system generates the following touch events:

- ACTION_DOWN—For the first pointer that touches the screen. This starts the gesture. The pointer data for this pointer is always at index 0 in the MotionEvent.
- ACTION_POINTER_DOWN—For extra pointers that enter the screen beyond the first. The pointer data for this pointer is at the index returned by getActionIndex().
- ACTION_MOVE—A change has happened during a press gesture.
- ACTION_POINTER_UP—Sent when a non-primary pointer goes up.
- ACTION_UP—Sent when the last pointer leaves the screen.

You keep track of individual pointers within a MotionEvent via each pointer's index and ID:

- **Index**: A MotionEvent effectively stores information about each pointer in an array. The index of a pointer is its position within this array. Most of the MotionEvent methods you use to interact with pointers take the pointer index as a parameter, not the pointer ID.
- **ID**: Each pointer also has an ID mapping that stays persistent across touch events to allow tracking an individual pointer across the entire gesture.

Multimedia

The Android SDK provides a set of APIs to handle multimedia files, such as audio, video and images. Moreover, the SDK provides other API sets that help developers to implement interesting graphics effects, like animations and so on.

The modern smart phones and tablets have an increasing storage capacity so that we can store music files, video files, images. etc. Not only the storage capacity is important, but also the high definition camera makes it possible to take impressive photos. In this context, the Multimedia API plays an important role.

Multimedia API

Android supports a wide list of audio, video and image formats. You can give a look here to have an idea; just to name a few formats supported:

Audio

- MP3
- MIDI
- Vorbis (es: mkv)

Video

- H.263
- MPEG-4 SP

Images

- JPEG
- GIF
- PNG

All the classes provided by the Android SDK that we can use to add multimedia capabilities to our apps are under the

android.media package.

In this package, the heart class is called MediaPlayer. This class has several methods that we can use to play audio and video file stored in our device or streamed from a remote server. This class implements a state machine with well-defined states and we have to know them before playing a file. Simplifying the state diagram, as shown in the official documentation, we can define these macro-states:

- **Idle state:** When we create a new instance of the MediaPlayer class.
- **Initialization state**: This state is triggered when we use setDataSource to set the information source that MediaPlayer has to use.
- **Prepared state**: In this state, the preparation work is completed. We can enter in this state calling prepare method or prepareAsync. In the first case after the method returns the state moves to Prepared. In the async way, we have to implement a listener to be notified when the system is ready and the state moves to Prepared. We have to keep in mind that when calling the prepare method, the entire app could hang before the method returns because the method can take a long time before it completes its work, especially when data is streamed from a remote server. We should avoid calling this method in the main thread because it

might cause a ANR (Application Not Responding) problem. Once the MediaPlayer is in prepared state we can play our file, pause it or stop it.

- **Completed state:** Te end of the stream is reached.

We can play a file in several ways:

```
// Raw audio file as resource
MediaPlayer mp = MediaPlayer.create(this, R.raw.audio_file);
// Local file
MediaPlayer mp1 = MediaPlayer.create(this, Uri.parse("file:///...."));
// Remote file
MediaPlayer mp2 = MediaPlayer.create(this, Uri.parse("http://website.com"));
```

or

we can use setDataSource in this way:

```
MediaPlayer mp3 = new MediaPlayer();
mp3.setDataSource("http://www.website.com");
```

Once we have created our MediaPlayer we can “prepare” it:

```
mp3.prepare();
```

and finally we can play it:

```
mp3.start();
```

Using Android Camera

If we want to add to our apps the capability to take photos using the integrated smart phone camera, then the best way is to use an Intent. For example, let us suppose we want to start the camera as soon as we press a button and show the result in our app.

In the onCreate method of our Activity, we have to setup a listener of the Button and when clicked to fire the intent:

```
Button b = (Button) findViewById(R.id.btn1);
b.setOnClickListener(new View.OnClickListener() {
@Override
public void onClick(View v) {
// Here we fire the intent to start the camera
Intent i = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);
startActivityForResult(i, 100);
}
});
```

In the onActivityResult method, we retrieve the picture taken and show the result:

```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
// This is called when we finish taking the photo
Bitmap bmp = (Bitmap) data.getExtras().get("data");
iv.setImageBitmap(bmp);
}
```

7. Packaging and Monetizing

Data Management

Android provides several options for you to save persistent application data. The solution you choose depends on your specific needs, such as whether the data should be private to your application or accessible to other applications (and the user) and how much space your data requires.

Your data storage options are the following:

1. Shared Preferences: Store private primitive data in key-value pairs.
2. Internal Storage: Store private data on the device memory.
3. External Storage: Store public data on the shared external storage.
4. SQLite Databases: Store structured data in a private database.
5. Network Connection: Store data on the web with your own network server.

Android provides a way for you to expose even your private data to other applications — with a content provider. A content provider is an optional component that exposes read/write access to your application data, subject to whatever restrictions you want to impose.

1. Shared Preferences

The SharedPreferences class provides a general framework that allows you to save and retrieve persistent key-value pairs of primitive data types. You can use SharedPreferences to save any primitive data: booleans, floats, ints, longs, and strings. This data will persist across user sessions (even if your application is killed).

To get a SharedPreferences object for your application, use one of two methods:

- getSharedPreferences() - Use this if you need multiple preferences files identified by name, which you specify with the first parameter.
- getPreferences() - Use this if you need only one preferences file for your Activity. Because this will be the only preferences file for your Activity, you don't supply a name.

To write values:

- Call edit() to get a SharedPreferences.Editor.
- Add values with methods such as putBoolean() and putString().
- Commit the new values with commit().

  To read values, use SharedPreferences methods such as getBoolean() and getString().

2. Internal Storage

You can save files directly on the device's internal storage. By default, files saved to the internal storage are private to your application and other applications cannot access them (nor can the user). When the user uninstalls your application, these files are removed.

To create and write a private file to the internal storage:

- Call openFileOutput() with the name of the file and the operating mode. This returns a FileOutputStream.

- Write to the file with write().
- Close the stream with close().

To read a file from internal storage:

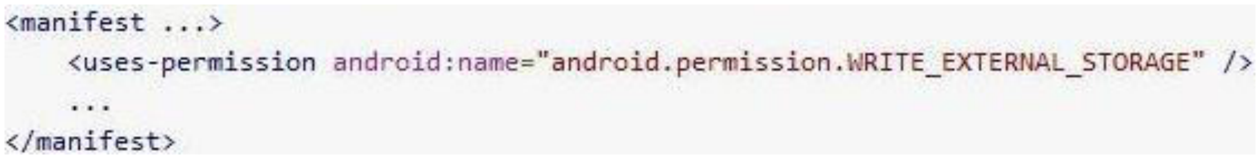
- Call openFileInput() and pass it the name of the file to read. This returns a FileInputStream.
- Read bytes from the file with read().
- Then close the stream with close().

3. External Storage

Every Android-compatible device supports a shared "external storage" that you can use to save files. This can be a removable storage media (such as an SD card) or an internal (non-removable) storage. Files saved to the external storage are world-readable and can be modified by the user when they enable USB mass storage to transfer files on a computer.

In order to read or write files on the external storage, your app must acquire the READ_EXTERNAL_STORAGE or WRITE_EXTERNAL_STORAGE system permissions.

For example:



```
<manifest ...>
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
    ...
</manifest>
```

If you need to both read and write files, then you need to request only the WRITE_EXTERNAL_STORAGE permission, because it implicitly requires read access as well.

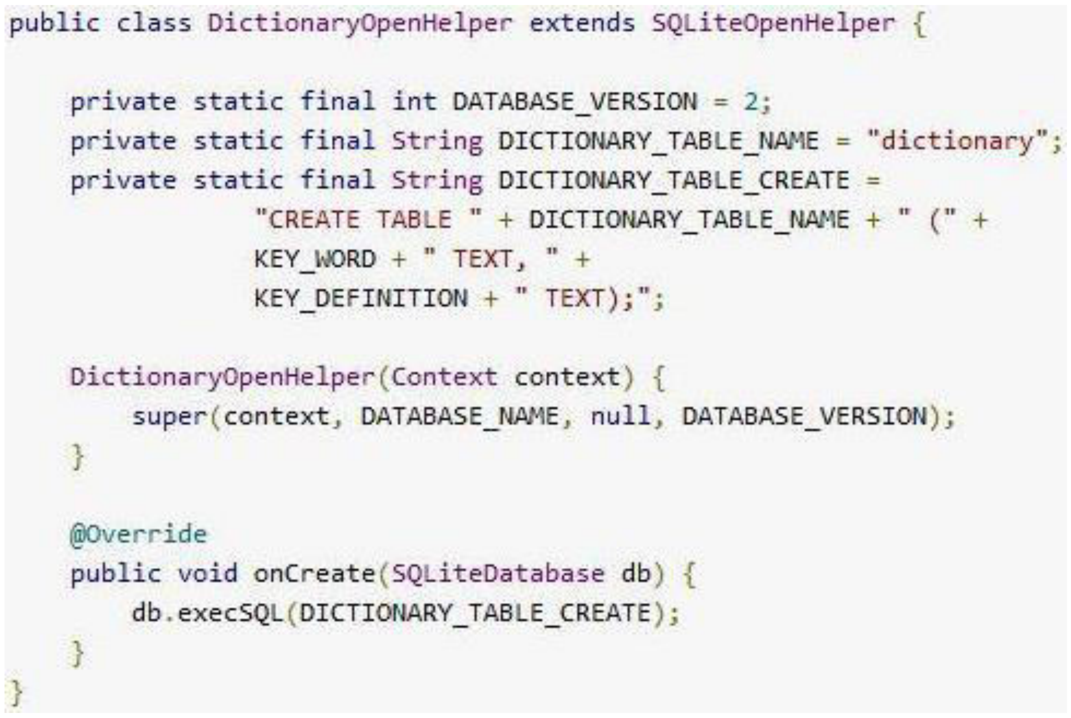
4. Databases

Android provides full support for SQLite databases. Any databases you create will be accessible by name to any class in the application, but not outside the application.

The recommended method to create a new SQLite database is to create a subclass of SQLiteOpenHelper and override the onCreate() method, in which you can execute a SQLite command to create tables in the database.

For example:

You can then get an instance of your SQLiteOpenHelper implementation using the constructor you've defined. To write to and read from the database, call getWritableDatabase() and getReadableDatabase(), respectively. These both return.



```
public class DictionaryOpenHelper extends SQLiteOpenHelper {

    private static final int DATABASE_VERSION = 2;
    private static final String DICTIONARY_TABLE_NAME = "dictionary";
    private static final String DICTIONARY_TABLE_CREATE =
                "CREATE TABLE " + DICTIONARY_TABLE_NAME + " (" +
                KEY_WORD + " TEXT, " +
                KEY_DEFINITION + " TEXT);";

    DictionaryOpenHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
        db.execSQL(DICTIONARY_TABLE_CREATE);
    }
}
```

You can then get an instance of your SQLiteOpenHelper implementation using the constructor you've defined. To write to and read from the database, call getWritableDatabase() and getReadableDatabase(), respectively. These both return a SQLiteDatabase object that represents the database and provides methods for SQLite operations.

You can execute SQLite queries using the SQLiteDatabase query() methods, which accept various query parameters, such as the table to query, the projection, selection, columns, grouping, and others. For complex queries, such as those that require column aliases, you should use SQLiteQueryBuilder, which provides several convienent methods for building queries.

Every SQLite query will return a Cursor that points to all the rows found by the query. The Cursor is always the mechanism with which you can navigate results from a database query and read rows and columns.

Database debugging

The Android SDK includes a sqlite3 database tool that allows you to browse table contents, run SQL commands, and perform other useful functions on SQLite databases.

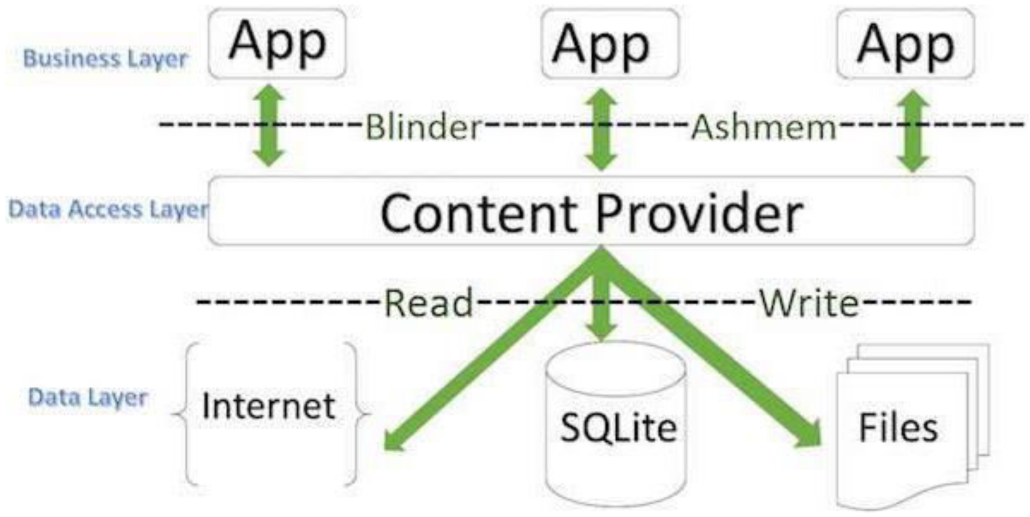
Content Provider Classes

Content providers are one of the primary building blocks of Android applications, providing content to applications. They encapsulate data and provide it to applications through the single ContentResolver interface.

A content provider is only required if you need to share data between multiple applications. For example, the contacts data is used by multiple applications and must be stored in a content provider. If you don't need to share data amongst multiple applications you can use a database directly via SQLiteDatabase.

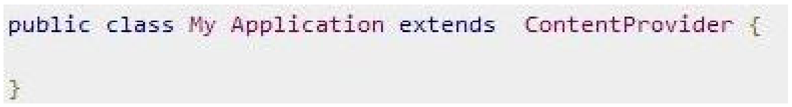
A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.

When a request is made via a ContentResolver the system inspects the authority of the given URI and passes the request to the content provider registered with the authority. The content provider can interpret the rest of the URI however it wants. The UriMatcher class is helpful for parsing URIs.



Content providers let you centralize content in one place and have many different applications access it as needed. A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content using insert(), update(), delete(), and query() methods. In most cases this data is stored in a **SQlite** database.

A content provider is implemented as a subclass of **ContentProvider** class and must implement a standard set of APIs that enable other applications to perform transactions.



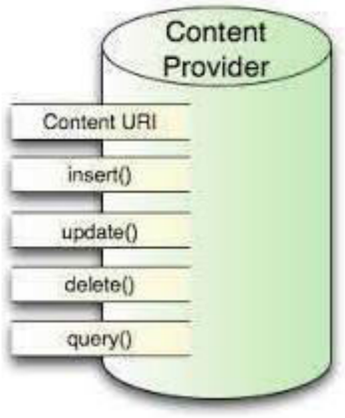
```
public class My Application extends  ContentProvider {

}
```

Requests to ContentResolver are automatically forwarded to the appropriate ContentProvider instance, so subclasses don't have to worry about the details of cross-process calls.

Creating Content Provider

This involves number of simple steps to create your own content provider.

- First of all you need to create a Content Provider class that extends the *ContentProviderbaseclass.*
- Second, you need to define your content provider URI address which will be used to access the content.
- Next you will need to create your own database to keep the content. Usually, Android uses SQLite database and framework needs to override *onCreate()* method which will use SQLite Open Helper method to create or open the provider's database. When your application is launched, the *onCreate()* handler of each of its Content Providers is called on the main application thread.
- Next you will have to implement Content Provider queries to perform different database specific operations.
- Finally register your Content Provider in your activity file using <provider> tag.



Signing and Exporting an App

Android requires that all apps be digitally signed with a certificate before they can be installed. Android uses this certificate to identify the author of an app, and the certificate does not need to be signed by a certificate authority. Android apps often use self-signed certificates. The app developer holds the certificate's private key.

You can sign an app in debug or release mode. You sign your app in debug mode during development and in release mode when you are ready to distribute your app. The Android SDK generates a certificate to sign apps in debug mode. To sign apps in release mode, you need to generate your own certificate.

Signing in Debug Mode

In debug mode, you sign your app with a debug certificate generated by the Android SDK tools. This certificate has a private key with a known password, so you can run and debug your app without typing the password every time you make a change to your project.

Android Studio signs your app in debug mode automatically when you run or debug your project from the IDE.

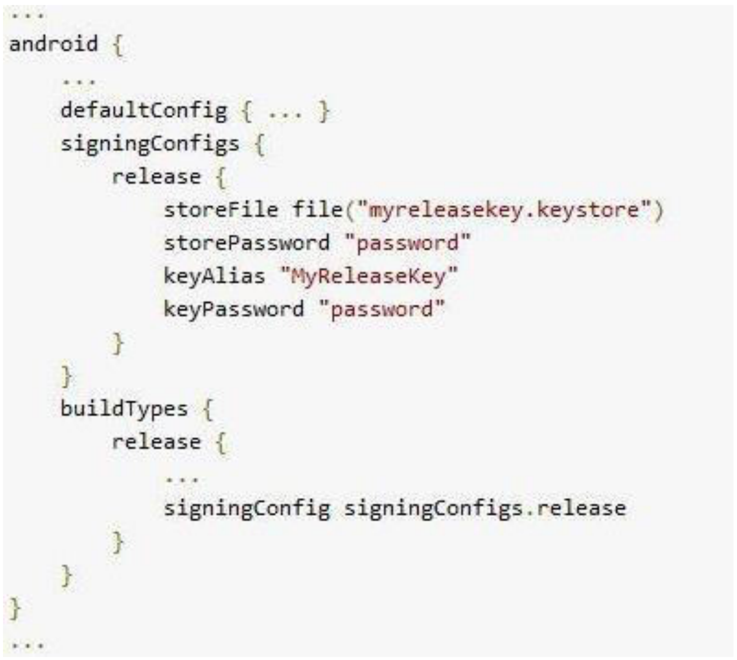
You can run and debug an app signed in debug mode on the emulator and on devices connected to your development machine through USB, but you cannot distribute an app signed in debug mode.

By default, the *debug* configuration uses a debug keystore, with a known password and a default key with a known password. The debug keystore is located in $HOME/.android/debug.keystore, and is created if not present. The debug build type is set to use this debug SigningConfig automatically.

Signing in Release Mode

In release mode, you sign your app with your own certificate:

- *Create a keystore.* A **keystore** is a binary file that contains a set of private keys. You must keep your keystore in a safe and secure place.
- *Create a private key.* A **private key** represents the entity to be identified with the app, such as a person or a company.
- Add the signing configuration to the build file for the app module:



```
...
android {
    ...
    defaultConfig { ... }
    signingConfigs {
        release {
            storeFile file("myreleasekey.keystore")
            storePassword "password"
            keyAlias "MyReleaseKey"
            keyPassword "password"
        }
    }
    buildTypes {
        release {
            ...
            signingConfig signingConfigs.release
        }
    }
}
...
```

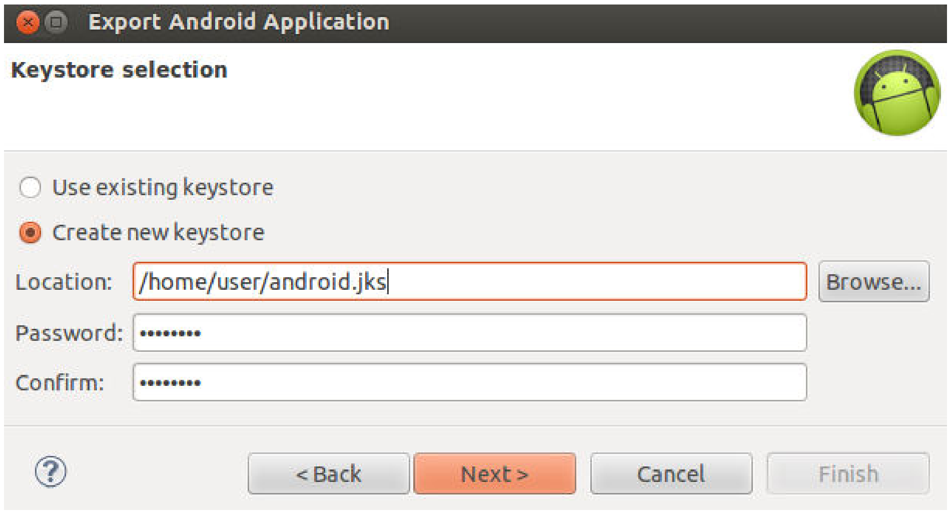
- Invoke the assembleRelease build task from Android Studio.

The package in app/build/apk/app-release.apk is now signed with your release key.

Exporting an app

To sign your app for release with ADT while exporting, follow these steps:

1. Select the project in the Package Explorer and select **File > Export**.
2. On the *Export* window, select **Export Android Application** and click **Next**.
3. On the *Export Android Application* window, select the project you want to sign and click **Next**.
4. Enter the location to create a keystore and a keystore password. If you already have a keystore, select **Use existing keystore**, enter your keystore's location and password, and go to step 6.



Export Android Application

Keystore selection

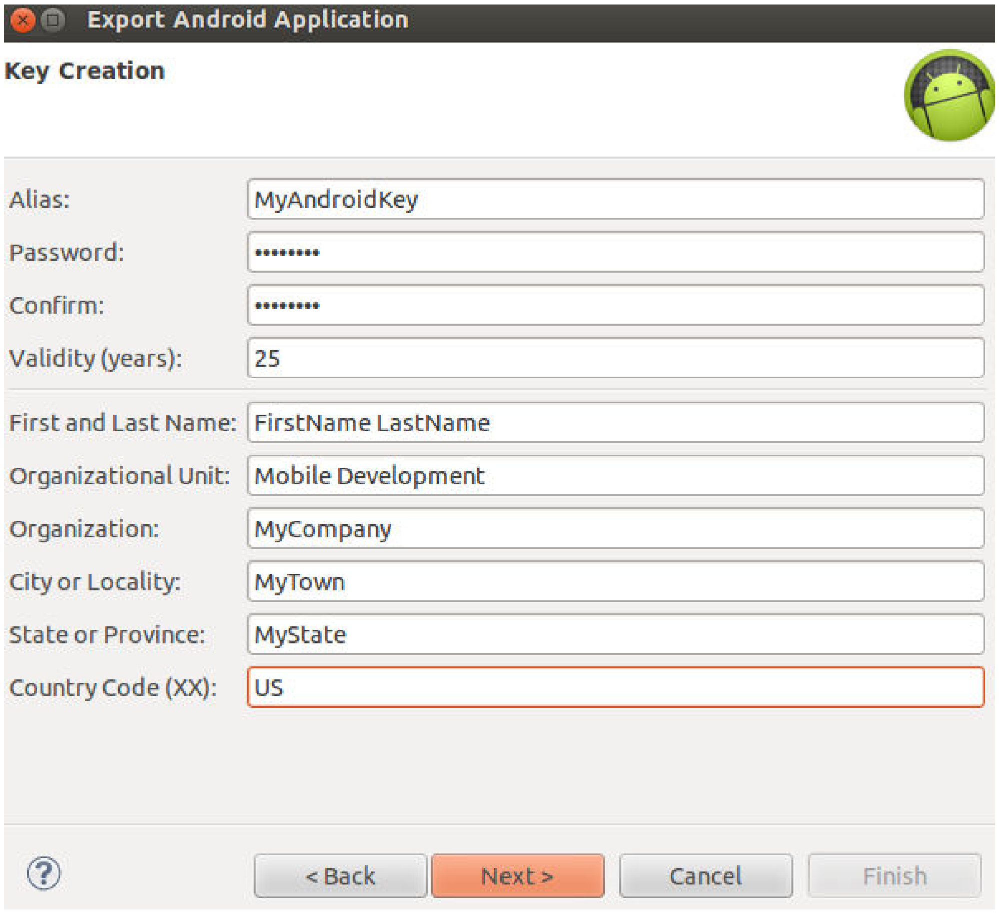
Use existing keystore [✓ Create new keystore]

Location: /home/user/android.jks

Password: ••••••••

Confirm: ••••••••

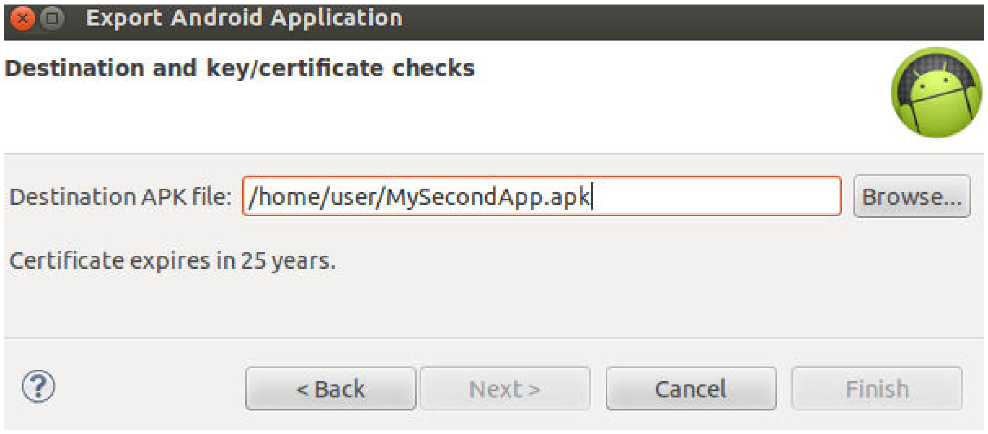
5. Provide the required information as shown in figure 5.Your key should be valid for at least 25 years, so you can sign app updates with the same key through the lifespan of your app.


Export Android Application

Key Creation

Alias:	MyAndroidKey
Password:	••••••••
Confirm:	••••••••
Validity (years):	25
First and Last Name:	FirstName LastName
Organizational Unit:	Mobile Development
Organization:	MyCompany
City or Locality:	MyTown
State or Province:	MyState
Country Code (XX):	US

6. Select the location to export the signed APK.


Export Android Application

Destination and key/certificate checks

Destination APK file: /home/user/MySecondApp.apk

Certificate expires in 25 years.

Publishing an App to the Play Store

There are basically three things to consider before publishing an android app.

- Registering for a Google Play publisher account
- Setting up a Google payments merchant account, if you will sell apps or in-app products.
- Exploring the Google Play Developer Console and publishing tools.

1. Register for a Publisher Account

1. Visit the Google Play Developer Console.

2. Enter basic information about your **developer identity** — name, email address, and so on. You can modify this information later.

3. Read and accept the **Developer Distribution Agreement** for your country or region. Note that apps and store listings that you publish on Google Play must comply with the Developer Program Policies and US export law.
4. Pay a **$25 USD registration fee** using Google payments. If you don't have a Google payments account, you can quickly set one up during the process.
5. When your registration is verified, you'll be notified at the email address you entered during registration.

Tips

_ You need a Google account to register. You can create one during the process.

_ If you are an organization, consider registering a new Google account rather than using a personal account.

_ Review the developer countries and merchant countries where you can distribute and sell apps.

2. Set Up a Google Payments Merchant Account

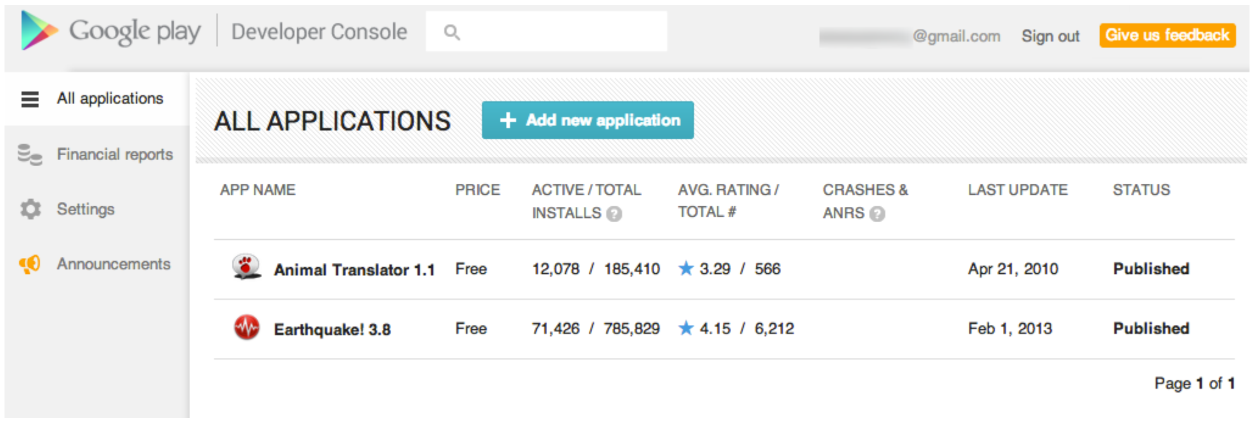
If you want to sell priced apps, in-app products, or subscriptions, you'll need a Google payments merchant account. You can set one up at any time, but first review the list of merchant countries.

To set up a Google payments merchant account:
1. **Sign in** to your Google Play Developer Console at https://play.google.com/apps/publish/.
2. Open **Financial reports** on the side navigation.
3. Click **Setup a Merchant Account now**.

This takes you to the Google payments site; you'll need information about your business to complete this step.

3. Explore the Developer Console

When your registration is verified, you can sign in to your Developer Console, which is the home for your app publishing operations and tools on Google Play



ALL APPLICATIONS

APP NAME	PRICE	ACTIVE / TOTAL INSTALLS	AVG. RATING / TOTAL #	CRASHES & ANRS	LAST UPDATE	STATUS
Animal Translator 1.1	Free	12,078 / 185,410	★ 3.29 / 566		Apr 21, 2010	Published
Earthquake! 3.8	Free	71,426 / 785,829	★ 4.15 / 6,212		Feb 1, 2013	Published

Page 1 of 1