Unit - 1: Introduction to the Android Platform and Application Development - I

Reference Books

1. Professional Android 4 Application Development – Reto Meier
2. Beginning Android 4 Application Development – Wei-Meng Lee
3. Beginning Android Programming with Android Studio – J. F. DiMarzio (4th Edition)
4. Android Cookbook – Ian F. Darwin , O’REILLY (2nd Edition)
5. Web References

Prior to Android ....

- Mobile Application Developers of low level programming with C and C++ needed to understand the specific hardware.
- Platforms such as Symbian were later created to provide developers with a wider target audience.
  - Although these platforms make heavy use of proprietary APIs that are difficult to work with.
- Java hosted MIDlets (Mobile Information Device) were also introduced, which are executed on JVM.
  - MIDlets provided abstraction for faster development, but device hardware access became restricted.
  - Third party applications development and deployment is difficult

What is Android?

- Android is a mobile operating system that is based on a modified version of Linux.
  - Originally developed by Android, Inc. In 2005
  - Later, Google purchased Android and took over its development work (July-2005)
  - The first Android mobile handset, the T-Mobile G1, was reseased in the US in October, 2008.
  - Android is continually developed by Google and the Open Handset Alliance (OHA)

What is Android?

- OHA is a collection of more than 80 technology companies, including H/W manufacturers, mobile carriers, S/W developers, semiconductor companies and commercialization companies.
- The core Android source code is known as Android Open Source Project (AOSP), and is primarily licensed under the Apache License.
- Vendors like hardware manufactures can add their own proprietary extensions to Android and customize Android to differentiate their products from others.

What is Android?

- Android is also associated with a suite of proprietary software developed by Google, called Google Mobile Services (GMS) that very frequently comes pre-installed in devices
  - which usually includes
    - Google Chrome web browser
    - Google Search and
    - always includes core apps for services such as Gmail, as well as the application store and digital distribution platform Google Play
  - These apps are licensed by manufacturers of Android devices certified under standards imposed by Google

Android v/s Others

- Platforms like Microsoft's Windows Phone and the Apple iPhone also provide a richer, simplified development environment for mobile applications.
- However, unlike Android, they are built on proprietary operating systems. In some cases,
  - they prioritize native applications over those created by third parties
  - restrict communication among applications and native phone data
  - restrict or control the distribution of third party applications to their platforms

Android v/s Others

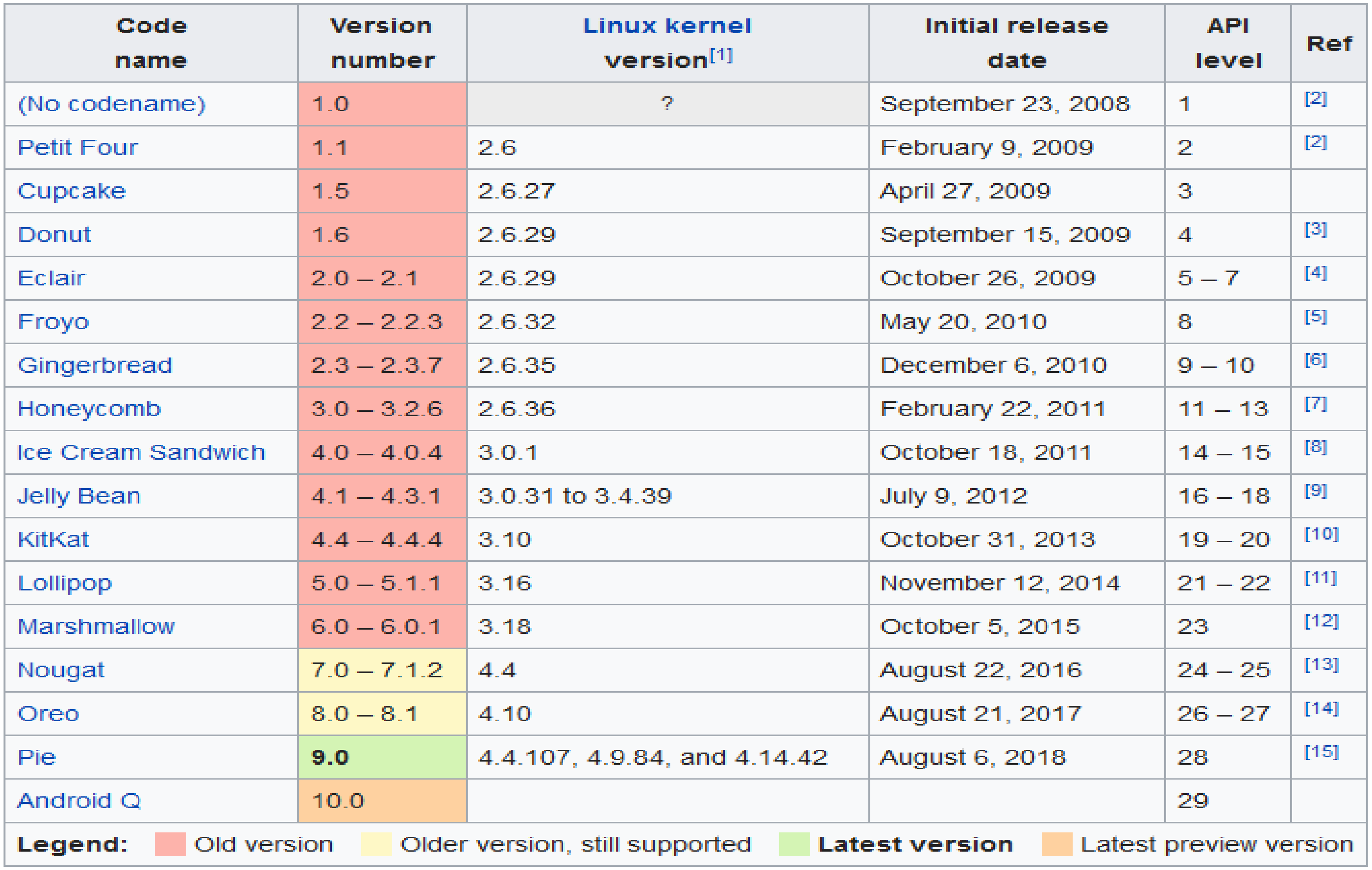
- Android offers new possibilities for mobile applications by offering an open development environment built on open source Linux kernel.
- In Android, H/W access is available to all applications through a series of API libraries.
- Application interaction is fully supported in Android.
- In Android, all applications, third party or native, have equal standing, as they are written with the same APIs and are executed on the same run time.
  - User can remove and replace any native application with a third party developer's alternative.

Android Versions and their Names

1. Android 1.5: Android Cupcake
2. Android 1.6: Android Donut
3. Android 2.0: Android Eclair
4. Android 2.2: Android Froyo
5. Android 2.3: Android Gingerbread
6. Android 3.0: Android Honeycomb
7. Android 4.0: Android Ice Cream Sandwich
8. Android 4.1 to 4.3.1: Android Jelly Bean
9. Android 4.4 to 4.4.4: Android KitKat
10. Android 5.0 to 5.1.1: Android Lollipop
11. Android 6.0 to 6.0.1: Android Marshmallow
12. Android 7.0 to 7.1: Android Nougat
13. Android 8.0 to Android 8.1: Android Oreo
14. Android 9.0: Android Pie

Android Version History

(https://en.wikipedia.org/wiki/Android_version_history)



Code name	Version number	Linux kernel version[1]	Initial release date	API level	Ref
(No codename)	1.0	?	September 23, 2008	1	[2]
Petit Four	1.1	2.6	February 9, 2009	2	[2]
Cupcake	1.5	2.6.27	April 27, 2009	3	
Donut	1.6	2.6.29	September 15, 2009	4	[3]
Eclair	2.0 – 2.1	2.6.29	October 26, 2009	5 – 7	[4]
Froyo	2.2 – 2.2.3	2.6.32	May 20, 2010	8	[5]
Gingerbread	2.3 – 2.3.7	2.6.35	December 6, 2010	9 – 10	[6]
Honeycomb	3.0 – 3.2.6	2.6.36	February 22, 2011	11 – 13	[7]
Ice Cream Sandwich	4.0 – 4.0.4	3.0.1	October 18, 2011	14 – 15	[8]
Jelly Bean	4.1 – 4.3.1	3.0.31 to 3.4.39	July 9, 2012	16 – 18	[9]
KitKat	4.4 – 4.4.4	3.10	October 31, 2013	19 – 20	[10]
Lollipop	5.0 – 5.1.1	3.16	November 12, 2014	21 – 22	[11]
Marshmallow	6.0 – 6.0.1	3.18	October 5, 2015	23	[12]
Nougat	7.0 – 7.1.2	4.4	August 22, 2016	24 – 25	[13]
Oreo	8.0 – 8.1	4.10	August 21, 2017	26 – 27	[14]
Pie	**9.0**	4.4.107, 4.9.84, and 4.14.42	August 6, 2018	28	[15]
Android Q	10.0			29	

Legend: 🟥 Old version 🟨 Older version, still supported 🟩 **Latest version** 🟧 Latest preview version

Android API Level

- Each Android device runs at exactly *one* API level – this API level is guaranteed to be unique per Android platform version.
- The API level precisely identifies the version of the API set that your app can call into; it identifies the combination of manifest elements, permissions, etc. that you code against as a developer.
- Android's system of API levels helps Android determine whether an application is compatible with an Android system image prior to installing the application on a device.
- When an application is built, it contains the following API level information:
  - The *target* API level of Android that the app is built to run on.
  - The *minimum* Android API level that an Android device must have to run your app.

Android API Level

(https://docs.microsoft.com/en-us/xamarin/android/app-fundamentals/android-api-levels?tabs=windows)

- These settings are used to ensure that the functionality needed to run the app correctly is available on the Android device at installation time.
- If not, the app is blocked from running on that device.
- For example, if the API level of an Android device is lower than the minimum API level that you specify for your app, the Android device will prevent the user from installing your app.

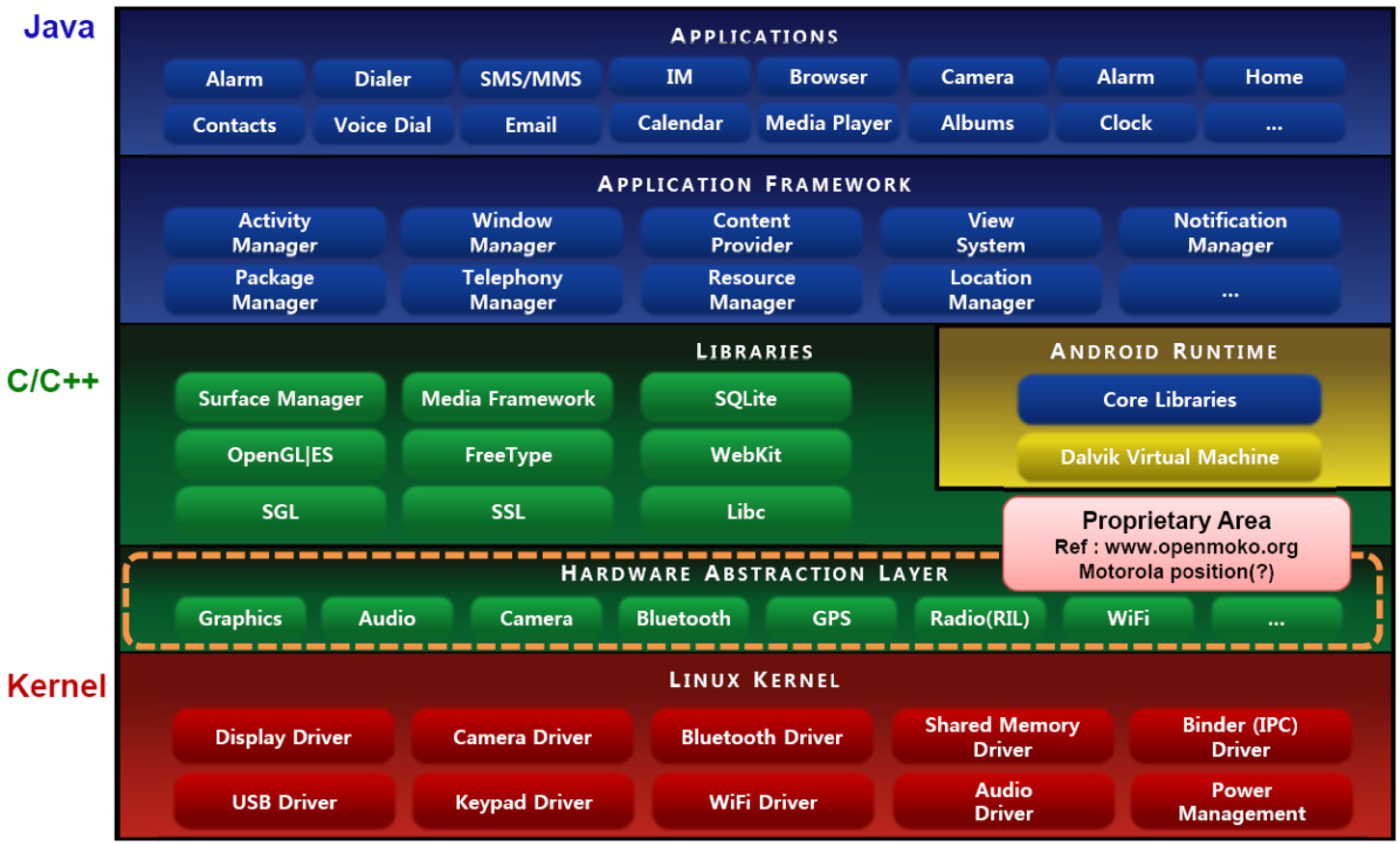
Features of Android

- Storage: Uses SQLite, a lightweight relational database, for data storage.
- Connectivity: Supports GSM/EDGE, IDEN CDMA, EV-DO, Bluetooth, Wi-Fi, LTE, NFC, and WiMAX.
- Messageing: Supports both SMS and MMS.
- Web browser: Based on the open source WebKit

Features of Android

- Media support: Includes support for the media in form of MPEG-4, MP4, MP3, MIDI, WAV, JPEG, PNG, GIF, BMP, and many more.
- Hardware support: Accelerometer Sensor, Camera, Digital Compass, Proximity Sensor, and GPS.
- Multi-touch:
- Multi-tasking:
- Flash support:
- Tethering: supports sharing of Internet connections as a wired/wireless hotspot.

Architecture of Android



Architecture of Android

- The Android OS is made up various layers.
- The Android OS is roughly divided into five sections in four main layers.
  1. Linux Kernel
  2. Libraries
  3. Android Runtime
  4. Applications Framework
  5. Applications

Linux Kernel

- This is the kernel on which Android is based.
- This layer contains all the low-level device drivers for the various hardware components of an Android device.
- Kernel also provides an abstraction layer between the hardware and the remainder of the stack.

Libraries

- Running on top of the kernel, Android includes various C/C++ core libraries.
- These contain all the code that provides the main features of an Android OS.
- For example,
  - The SQLite library for native database support
  - The SSL and WebKit for integrated web browser and Internet security.
  - Media library for playback of audio and video media
  - Surface manager to provide display management

Android Runtime

- Android runtime includes core libraries and Dalvik VM.

1. At the same layer as the libraries, the Android runtime provides a set of ***core libraries*** that enable developers to write Android apps using the Java programming language.
2. It includes ***Dalvik Virtual Machine***, which enables every Android application to run in its own process, with its own instance of the Dalvik Virtual Machine.
   - Android applications are compiled into Dalvik executables.
   - Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU.

Application Framework

- Application framework exposes the various capabilities of the Android OS to application developers so that they can make use of them in their applications.
- It provides the classes used to create Android applications.
- It also provides a generic abstraction for hardware access and manages the user interface and application resources.

Applications

- At this top layer, we find applications that ship with the Android device, such as Phone, Contacts, Browser, etc.
- The applications that one can download and install from the play store, are also found in this layer.
- Any application that one writes is located at this layer.
- The application layer runs within the Android runtime, using the classes and services made available from the application framework.

Types of Android Applications

- **Foreground**: An application that is useful only when it is in the foreground and is effectively suspended when it is not visible.
  - For example, Games
- **Background**: An application with limited interaction that spends most of its lifetime hidden.
  - These applications are less common
  - For example, SMS auto-responders, alarm clock

Types of Android Applications

- **Intermittent**: most well designed applications fall in this category;
  - At one extreme are applications that expect limited interactivity but do most of their work in the background. For example, Media Player.
  - At the other extreme are applications that are typically used as foreground applications but that do important work in the background . For example, E-mail, News applications.
- **Widgets and Live Wallpapers**:
  - An application may consist entirely of a Widget or Live Wallpaper.
  - By creating widgets and live wallpapers, one provides interactive visual components that can add functionality to user's home screen
  - Widgets are commonly used to display dynamic information, such as battery level, weather forecasts, or the date and time.
  - Widgets are also used to invoke an activity when tapped.

Dalvik Virtual Machine

- Android uses its own custom VM designed to ensure that multiple instances run efficiently on a single device.
- The DVM uses the device's underlying Linux kernel to handle low-level functionality including;
  - Security
  - Threading
  - Process & Memory management
- If speed and efficiency of C/C++ is required for an application, Android provides a Native Development Kit (NDK).
  - NDK enables developer to create C++ libraries using the ***libc*** and ***libm*** libraries
  - NDK enables access of OpenGL

Dalvik Virtual Machine

- All Android hardware and system service access is managed using Dalvik as a middle tier.
- By using VM to host application execution, developers have an abstraction layer that ensures they should never have to worry about a particular hardware implementation.
- The DVM executes Dalvik executable files (.dex), a format optimized to ensure minimal memory footprint.
- .dex is created by transforming Java language compiled classes using the tools supplied within the SDK.

Difference between .dex and .apk

- **DEX (Dalvik Executable)**
  - DEX is binary file format, so it is compiled.
  - We could say, that *.dex* file is for DVM (Dalvik Virtual Machine) something like *.class* files for JVM.
  - DEX file format is generated from java CLASS files by dex compiler from Android SDK.
  - This compiler translates JVM bytecode to DVM bytecode and put all class files to one .dex file.
- **APK (Android Application Package)**
  - APK is file format designed for distributing Android application on its platform.
  - It has some similarities with JAR format. Again it is simple just ZIP archive, but APK files have pre-defined specific structure.
  - For example, it always must contains file named *AndroidManifest.xml* and many more.
  - Also this package aggregates compiled classes in dex format.

Key Android Concepts

- Android SDK : The Android Software Development Kit (SDK) contains a Android APIs, Development tools, AVD Manager and emulator, debugger, libraries, documentation, sample code, and tutorials.
- Android SDK Manager: manages the various versions of the Android SDK currently installed on your computer.
- Android Virtual Devices (AVDs): used for testing Android application.
  - An AVD is an emulator instance that enables you to model an actual device.
  - You can create as many AVDs as you want in order to test your applications with several different configurations.

Key Android Concepts

- Android Development Tools (ADT): is an extension to the Eclipse IDE that supports the creation and debugging of Android applications.
- One can do following using ADT:
  - Create new Android application projects
  - Access the tools for accessing your Android emulators and devices
  - Compile and debug Android applications
  - Export Android applications into ***Android Packages (APKs)***
  - Creating digital certificates for code-signing your APK.

Android Application Components

- Intent
- Activity
- Service
- Broadcast Receiver
- Content Provider
- Synch Adapters

Activity

- An activity is a window that contains the user interface of your application.
- An activity comprises the visual components (“views”) for one screen as well as the code that displays data into that screen and can respond to user events on that screen.
- Almost every application has at least one activity.

Activity

- An Android application may consists of a number of activities.
- Each activity is a single user interface screen.
- One activity may invoke another activity.
- Android maintains an Activity Stack.
- The activity that the user is currently interacting with is always at the top of the Activity Stack.
- When that activity starts another activity, the newly started activity is pushed on the top of the stack.
- When the newly started activity finishes, it is popped from the Activity Stack, bringing the previous activity to the top of the stack again.

Service

- A service is a component that has no user interface, and can run for a longer period of time than an Activity.
- Two main uses for services are
  - for long running tasks such as a music player, and
  - running medium-length tasks without tying up the user-interface thread.
- Services are used for performing functions that are to be performed periodically without the user invoking the application or for taking some action when particular event occurs.
- Services are also used to take some action in response to some event

Broadcast Receivers

- Broadcast receivers are less common, and are used to respond to system-wide events such as
  - the network losing or regaining connectivity
  - the battery running low
  - the system rebooting, and so on.
- Broadcast Receivers are consumers of broadcast messages.

Content providers

- Content providers are also relatively rare, and are used when one application needs to share its data with other applications; they can also be used with sync-adapters.

Sync adapters

- Sync adapters synchronize data with cloud services.
- The best-known examples are the contacts and calendar apps on the device, which can easily be synchronized to your Google account.

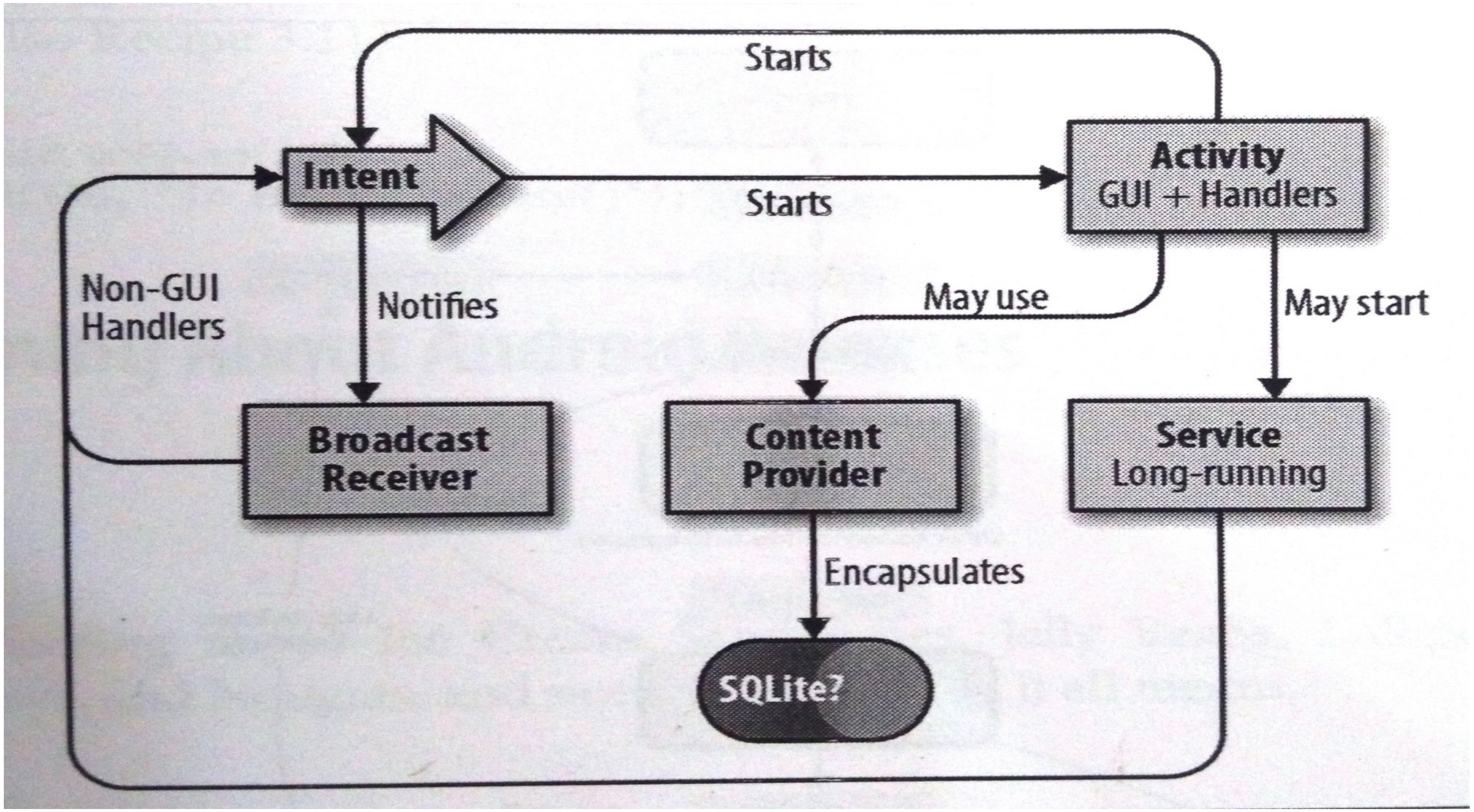
Intent

- Your code does not create these objects using the “new” operator, as in conventional Java, but requests the invocation of Activities, Services, etc. using an ***Intent.***
- Intent is an object that specifies your intention to have something done.
- Intent can start,
  - Activities within your application by class name.
  - Activities in other applications by specifying content type and other information,
  - Services, and request other operations.
- The interactions among the components of android application are shown in following figure.

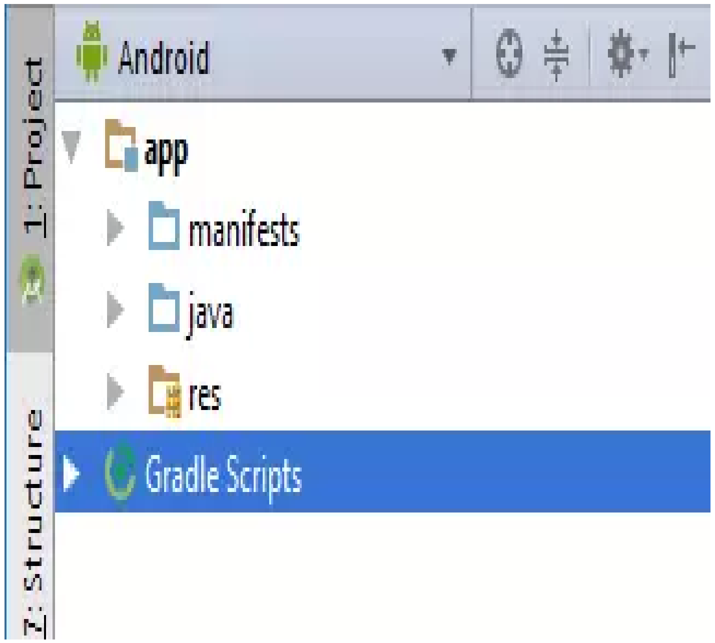
Intent

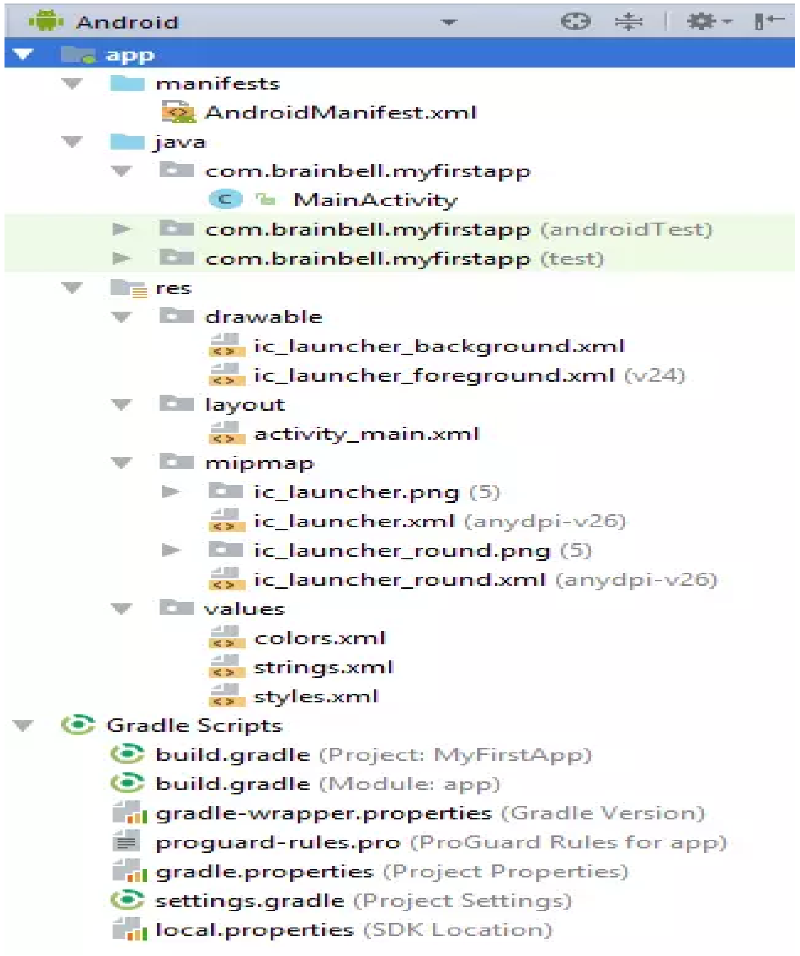
- An Intent is basically the “glue” that enables different activities from different applications to work together seamlessly, ensuring that tasks can be performed as though they all belong to one single application.
- When an application has more than one activity, you need to navigate from one to another.
- In android, you navigate between activities through intent.

Android Application Components



Anatomy of Android Project





```
Android
app
    manifests
        AndroidManifest.xml
    java
        com.brainbell.myfirstapp
            MainActivity
        com.brainbell.myfirstapp (androidTest)
        com.brainbell.myfirstapp (test)
    res
        drawable
            ic_launcher_background.xml
            ic_launcher_foreground.xml (v24)
        layout
            activity_main.xml
        mipmap
            ic_launcher.png (5)
            ic_launcher.xml (anydpi-v26)
            ic_launcher_round.png (5)
            ic_launcher_round.xml (anydpi-v26)
        values
            colors.xml
            strings.xml
            styles.xml
Gradle Scripts
    build.gradle (Project: MyFirstApp)
    build.gradle (Module: app)
    gradle-wrapper.properties (Gradle Version)
    proguard-rules.pro (ProGuard Rules for app)
    gradle.properties (Project Properties)
    settings.gradle (Project Settings)
    local.properties (SDK Location)
```

Anatomy of Android Project

- The Project tool window has several views, including the
  - Android view
  - Project view
  - Packages view
- Each view shows the same stuff, but each view organizes this stuff a bit differently
- Android and Project are the most useful views, though the Android view may hide certain folders from you.
- By default, Android Studio will set the view to Android.
- The Project tool window provides a simple tree interface with files and nested folders that you can toggle.

Anatomy of Android Project

- When you create an application in Android Studio, you find that the project is divided into an App folder and Gradle scripts.
- The App folder contains three subfolders (manifests, java and res) that make up your application.

Anatomy of Android Project
1. Manifest Folder

- This is where you would put your manifest files.
- Most Android apps have single manifest file. But an app may have several manifest files due to application versioning, or for supporting specific hardware.
- An “AndroidManifest.xml” file is generated inside manifest folder by Android Studio when you create a project.
- This file contains the configuration parameters of the project such as permissions, services and additional libraries.
- A manifest file also provides information to the OS and Google Play store about your app.

Anatomy of Android Project
2. Java Folder

- This is the folder in your project where you will be storing all of the source code files written in Java programming language.
- A MainActivity.java is automatically created in this folder by Android Studio.
- All of your classes will be available here
- Android Studio will even bundle together the package path so that you can work with the files without having to drill down through the folders that make up your package.

Anatomy of Android Project 3. Res Folder

- It contains folders that help you separate and sort the resources of your application.
- Resources basically mean all the needed files except the source code.
- For example, while developing an app, you need to include resource files such as the app-logo, photos, sounds, videos or animations.
- Each file type should be added to its own folder to comply with the Android development standards.

Resources

- Android follows the philosophy of keeping application logic separate from the user interface.
- Hence everything except the code is defined as resources separate from the code.
- Following are different types of resources:
  - Drawable
  - Layouts
  - Mipmap
  - Values

Resources/Drawable

- Drawables are graphics resources that can be painted on the screen. They include icons and images.
- To cater to different screen sizes, we may have folders like res/drawable-ldpi, res/drawable-mdpi, res/drawable-hdpi, res/drawable-xhdpi, res/drawable-xxhdpi, etc. corresponding to screens with low, medium, high, extra-high and extra-extra-high DPI (Dots Per Inch) screens respectively.
- Having different sized icons and images for different sized screens ensures that the application has nice looks on all types of devices.

Resources/Layouts

- Layouts are ViewGroups that are not rendered (displayed) themselves, but are used to hold and organize other views geometrically on the screen.
- layouts decide the sizes and arrangements of the views on screen at runtime.
- Because there may be a wide variation in screen sizes between different devices, it is not recommended to use fixed sizes and positioning.
- The typical way of constructing user interface screens is to define the layout in a layout file in XML format and then generate the necessary views at runtime from the XML definition. this process is called ***layout inflation.***
- Like drawables, we may define different layouts for different screen sizes and Android will pick the most suitable one based on the actual screen size at runtime.
- We may also create views programmatically at runtime without defining them in an XML file.
- Layouts can be nested one inside another to create a complex layout

Resources/mipmap

- Devices have different resolutions so the launcher app show the icons on different resolutions.
- Resource optimisation techniques sometimes removes resources for unused screen densities and when the launcher app has to upscale a lower-resolution icon for display it might look blurred.
  - **Launcher** is the name given to the part of the Android user interface that lets users customize the home screen.
- It makes sure launcher apps show a high-resolution icon for your app by moving all densities of your launcher icons to density-specific res/mipmap/ folders (for example res/mipmap-mdpi/ and res/mipmap-xxxhdpi/).
- App uses mipmap/ folders instead of the drawable/ folders for launcher icons.

Resources/values

- There are different types of values that are stored in XML files seperately from the code for easy changeability. They are:

1. **Colors**
   - Colors are used to define basic colouring scheme of the app globally
2. **Strings**
   - No string should be hard-coded anywhere in the application code or other resource files (e.g. layouts).
   - All such strings, including labels for view on the screen, titles, menu item titles, etc. should be defined in the strings XML file.
   - This enables internationalization, where the strings may be translated into different languages by non- developers or third parties without going through or having access to the source code.
   - The systems picks the correct strings XML file at runtime based on the locale (language) settings in effect on the user's device

Resources/values

3. **Styles**
   - Styles are used to define visual appearance of view.
   - They work very similarly to CSS.
   - To ensure maximum portability across different devices, one should never define any visual characterisitcs anywhere other than in styles.
   - One style may inherit from another style. There are several built-in styles and a programmer may define one's own style as well.
   - A style applied at the entire ap- plication level is called a theme.
   - Themes are used to ensure consistent look and feel throughout the application.
   - They also provide a central place for the styles for easy changeability and for avoiding duplication

Resources

- **Advantages of using Resources**
  - Separating the presentation from the logic
  - Providing different screen layouts, menus and graphic resources (e.g. images/icons) for different screen sizes
  - Ability to change the visual design of the app without affecting the code

Anatomy of Android Project
Gradle Scripts

- Apk files are built using the gradle build system, which is integrated in the Android Studio.
- When we start an app, the gradle build scripts are automatically created.
- If you have special requirements for your project, you can specify these requirements here.
- The gradle scripts folder contains the scripts used to build the app are:
  - configuration files
  - properties files
  - setting files

Android Activity Lifecycle

- The lifecycle of activities is managed by the Android system itself.
- To conserve resources, especially battery power and memory, Android may terminate a running activity at any time.
- The activity is supposed to save its state when an event signalling the possibility of such termination occurs.
- When the activity is restarted, it is passed the previously saved state as a Bundle.
- The activity should use it to restore itself such that the user has a seamless experience and does not realize that the activity was terminated and restarted in-between.

Android Activity Lifecycle

- At any time, an activity may be in one of the following states.

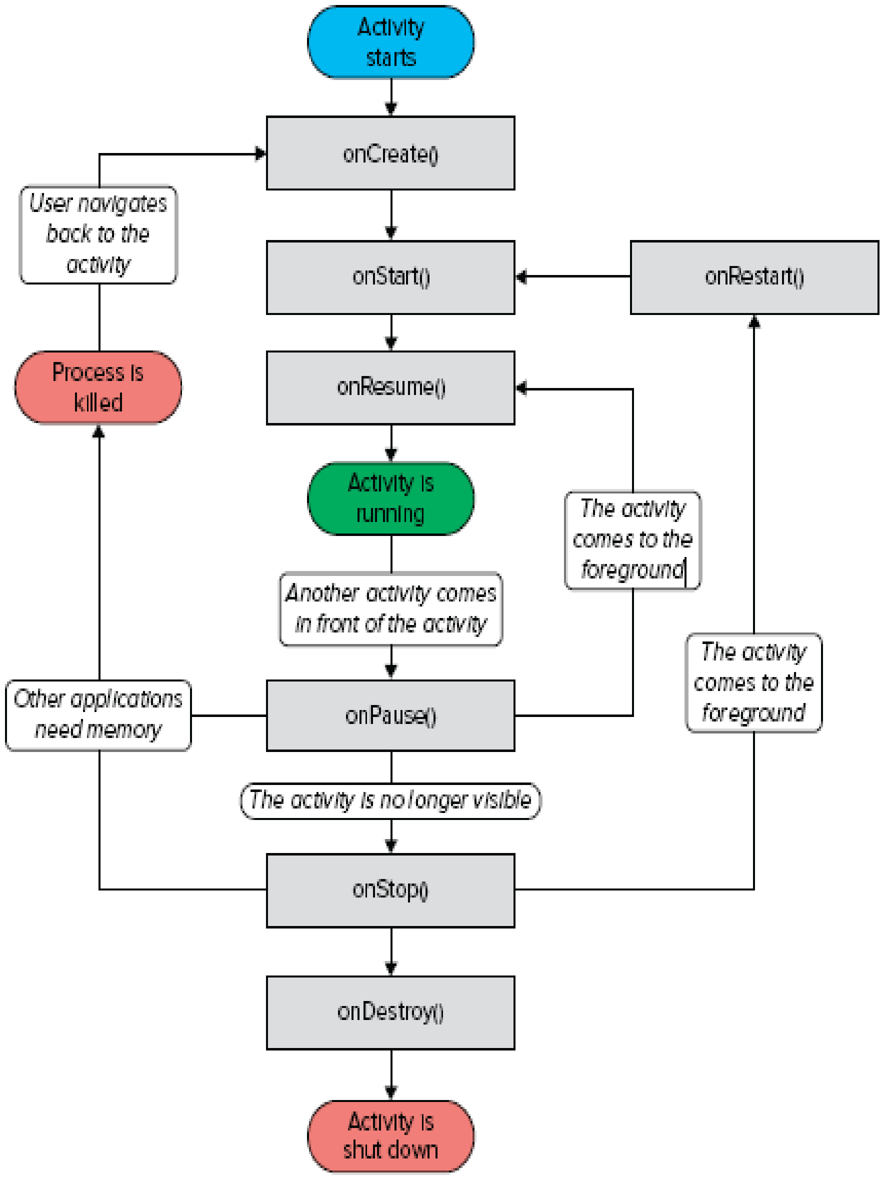
1. **Resumed (Running)** An activity in this state is currently interacting with the user and has user focus in it.
   – It is on the top of the Activity Stack.
   – Such activity must continue running and update the UI and accept user input when needed, so it is never killed automatically by the Android system
2. **Paused** An activity in this state is not currently interacting with the user and does not have user focus in it.
   – It is not on the top of the Activity Stack.
   – However, because the top-of-the-stack activity does not occupy the whole screen, this activity is at least partially visible.
   – It is desirable to keep such an activity running and updating its UI, so such activity is killed automatically by the Android system
3. **Stopped** An activity in this state is completely obscured by another activity (hidden behind that activity).
   – While it may need to preserve its state (e.g. display), there is no need to waste CPU cycles on keeping it running.
   – Such activity may be killed automatically by the Android system at any time

Android Activity Lifecycle

- From the moment an activity appears on the screen to the moment it is hidden, it goes through a number of stages, known as an activity's *life cycle.*
- The Activity base class defines a series of events that govern the life cycle of an activity.
- By default, the activity created for you contains the onCreate() event.
- Within this event handler is the code that helps to display the UI elements of your screen.

Activity Life Cycle

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 the previous activity is being resumed
5. onStop() — Called when the activity is no longer visible to the user
6. onDestroy() — Called before the activity is destroyed by the system (either manually or by the system to conserve memory)
7. onRestart() — Called when the activity has been stopped and is restarting again


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

Activity Life Cycle (Remember This!!)

- The onPause() method is called in both scenarios — when an activity is sent to the background, as well as when it is killed.
- When an activity is started, the onStart() and onResume()methods are always called, regardless of whether the activity is restored from the background or newly created.
- When an activity is created for the first time, the onCreate() method is called.

Activity Life Cycle (Remember This!!)

- Use the onCreate() method to create and instantiate the objects that you will be using in your application.
- Use the onResume() method to start any services or code that needs to run while your activity is in the foreground.
- Use the onPause() method to stop any services or code that does not need to run when your activity is not in the foreground.
- Use the onDestroy() method to free up resources before your activity is destroyed.
- ***Even if an application has only one activity and the activity is killed, the application will still be running in memory.***