Module 1

Challenges of Android app development

- Building for a multi-screen world
- Getting performance right
- Keeping your code and your users secure
- Remaining compatible with older platform versions
- Understanding the market and the user.

Android: Open Handset Alliance[OHA]

Members:

- Mobile Operators
- Handset Manufacturers
- Semiconductor Companies
- Software Companies
- Commercialization Companies

Android: Features

- **Storage** — Uses SQLite, a lightweight relational database, for data storage.
- **Connectivity** — Supports GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth (includes A2DP and AVRCP), Wi-Fi, LTE, and WiMAX.
- **Messaging** — Supports both SMS and MMS.
- **Web browser** — Based on the open source WebKit, together with Chrome's V8 JavaScript engine
- **Media support** — Includes support for the following media: H.263, H.264 (in 3GP or MP4 container), MPEG-4 SP, AMR, AMR-WB (in 3GP container), AAC, HE-AAC (in MP4 or 3GP container), MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP.
- **Hardware support** — Accelerometer Sensor, Camera, Digital Compass, Proximity Sensor, and GPS
- **Multi-touch** — Supports multi-touch screens
- **Multi-tasking** — Supports multi-tasking applications

- **Flash support** — Android 2.3 supports Flash 10.1.
- **Tethering** — Supports sharing of Internet connections as a wired/wireless hotspot

Android: Development Tools

Top Hybrid Mobile App Frameworks

- Mobile Angular UI
- Appcelerator Titanium
- Kendo UI
- Sencha Touch
- IONIC
- PhoneGap (distribution of Apache Cordova)
- Apache Cordova (formerly PhoneGap)
- Flutter (early-stage funding from Y Combinator & was acquired by Google in October 2013)

Android: Architecture

1. Linux kernel

- It is the heart of android architecture that exists at the root of android architecture.
- All the hardware drivers are handled by this kernel.
- Drivers are the small programs written specifically to communicate with hardware.
- It also serves as the abstraction layer for other layers.

2. Native Libraries

- This layer provides the device, which has the capability to handle different types of data and information.
- Code in C and C++ languages for fast procedural transactions of data.
- Surface Manager, Media Frame work, WebKit, OpenGL, FreeType, SQLite, Media, C runtime library (libc) etc.

3. Android Runtime

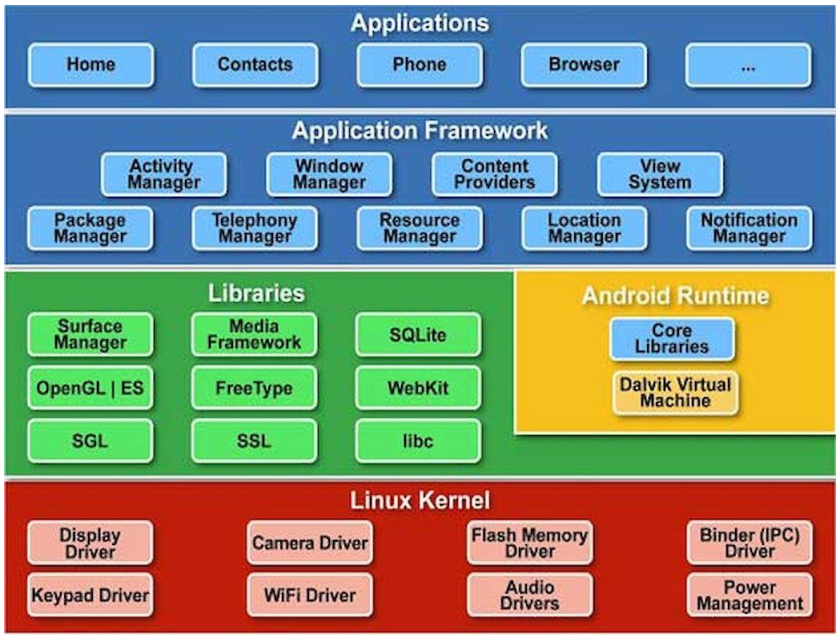
- Dalvik Virtual Machine (DVM) which is responsible to run android application.
- DVM is like JVM but it is optimized for mobile devices.
- It consumes less memory and provides fast performance.
- It executes the files with the .dex extension (generated implicitly from the .class file)

4. Android Framework

- The important blocks of Application framework are UI (User Interface), telephony, resources, locations, Content Providers (data) and package managers.
- It provides a lot of classes and interfaces for android application development.

5. Application

- It is the top layer in the Android Architecture.
- This is the palace where your application is finally deployed over a device.
- All applications such as home, contact, settings, games, browsers are using android framework that uses android runtime and libraries.
- Android runtime and native libraries are using Linux kernel.
- Pre – installed applications can be replaced by newly written applications.
- Makes Oss so flexible.



<u>Application Components</u>

Activity is a single screen with a user interface

Service performs long-running tasks in background

Broadcast receiver responds to system-wide announcements

Content provider manages shared set of data

1. **Activity**

- An Activity object is similar to a WindowsForm. It usually presents a single graphical visual interface (GUI) which in addition to the displaying/collecting of data, provides some kind of ‘code-behind‘ functionality.
- A typical Android application contains one or more Activity objects.

- Applications must designate one activity as their main task or entry point. That activty is the first to be executed when the app is launched.
- An activity may transfer control and data to another activity through an interprocess communication protocol called intents.
- For example, a login activity may show a screen to enter user name and password. After clicking a button some authentication process is applied on the data, and before the login activity ends some other activity is called.

2. **Services**

- Services are a special type of activity that do not have a visual user interface. A service object may be active without the user noticing its presence.
- Services are analogous to secondary threads, usually running some kind of background 'busy-work' for an indefinite period of time.
- Applications start their own services or connect to services already active.
- Examples:
- Your background GPS service could be set to quietly run in the backgroud detecting location information from satellites, phone towers or wi-fi routers. The service could periodically broadcast location coordinates to any app listening for that kind of data. An application may opt for binding to the running GPS service and use the data that it supplies.

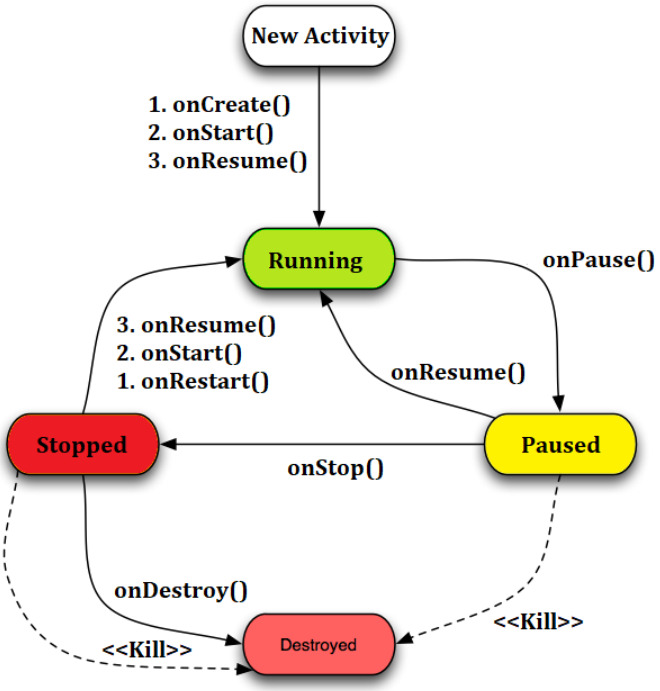
3. Broadcast Receiver

- A BroadcastReceiver is a dedicated listener that waits for a triggering system-wide message to do some work. The message could be something like: low-battery, wi-fi connection available, earth-quakes in California, speed-camera nearby.
- Broadcast receivers do not display a user interface.
- They tipically register with the system by means of a filter acting as a key. When the broadcasted message matches the key the receiver is activated.
- A broadcast receiver could respond by either executing a specific activity or use the notification mechanism to request the user's attention.

4. Content Provider

- A content provider is a data-centric service that makes persistent datasets available to any number of applications.
- Common global datasets include: contacts, pictures, messages, audio files, emails.
- The global datasets are usually stored in a SQLite database (however the developer does not need to be an SQL expert)
- The content provider class offers a standard set of parametric methods to enable other applications to retrieve, delete, update, and insert data items.

Android: Activity Life cycle



Android Activity Lifecycle methods

Method	Description
onCreate	called when activity is first created.
onStart	called when activity is becoming visible to the user.
onResume	called when activity will start interacting with the user.
onPause	called when activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.

onStop	called when activity is no longer visible to the user.
onRestart	called after your activity is stopped, prior to start.
onDestroy	called before the activity is destroyed by the system.

SDK (software development kit) Manager

- The first and most important piece of software you need to download is, of course, the Android
- SDK. The Android SDK contains a debugger, libraries, an emulator, documentation, sample code and tutorials.

Android Debug Bridge (ADB)

- ADB is a versatile command-line tool that lets you communicate with a device.
- The ADB command facilitates a variety of device actions, such as installing and debugging apps, and it provides access to a Unix shell that you can use to run a variety of commands on a device.
- It is a client-server program that includes three components:
- A client, which sends commands. The client runs on your development machine. You can invoke a client from a command-line terminal by issuing an ADB command.
- A daemon (ADBD), which runs commands on a device. The daemon runs as a background process on each device.
- A server, which manages communication between the client and the daemon. The server runs as a background process on your development machine.

Module 2

Introduction to Activity

What is Activity?

- An Activity is an application component that allows a user to perform an operation.
- Eg. Making call, capturing photos and sending messages etc.
- An Application is an collection of activities
- Main activity provides interface to the user and further invokes the other activity

Creating an Activity

- You can create an activity by creating a java class that extends the Activity class.
- Each activity needs to create a window in which its user interface components can be loaded.
- setContentView(View) method is invoked
- **onCreate() method** is called when Activity class is first created.

View, View Groups and Layouts

- A layout defines the structure for a user interface in your app, such as in an activity.
- All elements in the layout are built using a hierarchy of View and View Group objects.
- A View usually draws something the user can see and interact with.
- Whereas a View Group is an invisible container that defines the layout structure for View and other View Group objects.
- The View objects are usually called "widgets" and can be one of many subclasses, such as Button or TextView.
- The ViewGroup objects are usually called "layouts" can be one of many types that provide a different layout structure, such as LinearLayout or ConstraintLayout .

You can declare a layout in two ways:

- **Declare UI elements in XML**. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts. You can also use Android Studio's using a drag-and-drop interface. Layout Editor to build your XML layout
- **Instantiate layout elements at runtime**. Your app can create View and View Group objects (and manipulate their properties) programmatically.

View	ViewGroup
View is a simple rectangle box that responds to the user's actions.	ViewGroup is the invisible container. It holds View and ViewGroup
View is the SuperClass of All component like TextView, EditText, ListView, etc	ViewGroup is a collection of Views(TextView, EditText, ListView, etc..), somewhat like a container.
A View object is a component of the user interface (UI) like a button or a text box, and it's also called a widget.	A ViewGroup object is a layout, that is, a container of other ViewGroup objects (layouts) and View objects (widgets)
Examples are EditText, Button, CheckBox, etc.	For example, LinearLayout is the ViewGroup that contains Button(View), and other Layouts also.
android.view.View which is the base class of all UI classes.	ViewGroup is the base class for Layouts.

Common Layouts

Linear Layout

A layout that organizes its children into a single horizontal or vertical row. It creates a scrollbar if the length of the window exceeds the length of the screen.

Relative Layout

Enables you to specify the location of child objects relative to each other (child A to the left of child B) or to the parent (aligned to the top of the parent).

Web View

Displays web pages.

Linear Layout

- Linear Layout is a view group that aligns all children in a single direction, vertically or horizontally. You can specify the layout direction with the android:orientation attribute.
- All children of a Linear Layout are stacked one after the other, so a vertical list will only have one child per row, no matter how wide they are, and a horizontal list will only be one row high (the height of the tallest child, plus padding). A Linear Layout respects *margin*s between children and the *gravity* (right, center, or left alignment) of each child.

Relative Layout

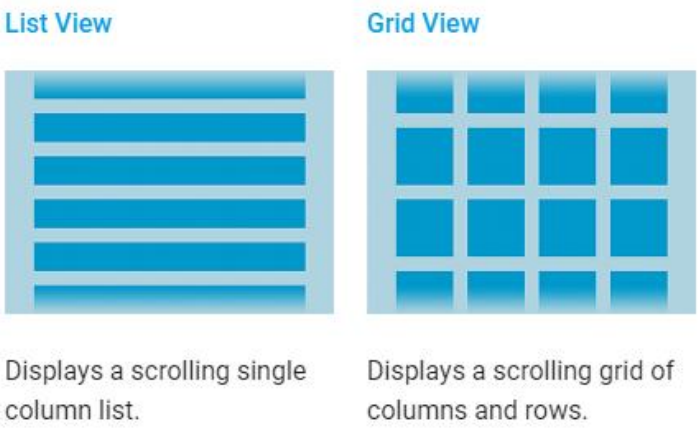
- Relative Layout is a view group that displays child views in relative positions. The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent Relative Layout area (such as aligned to the bottom, left or center).
- A Relative Layout is a very powerful utility for designing a user interface because it can eliminate nested view groups and keep your layout hierarchy flat, which improves performance. If you find yourself using several nested Linear Layout groups, you may be able to replace them with a single Relative Layout.

Constraint Layout

- Constraint Layout allows you to create large and complex layouts with a flat view hierarchy (no nested view groups). It's similar to Relative Layout in that all views are laid out according to relationships between sibling views and the parent layout, but it's more flexible than Relative Layout and easier to use with Android Studio's Layout Editor.
- All the power of Constraint Layout is available directly from the Layout Editor's visual tools, because the layout API and the Layout Editor were specially built for each other. So you can build your layout with Constraint Layout entirely by drag-and-dropping instead of editing the XML.

<u>Building Layouts with an Adapter</u>

- When the content for your layout is dynamic or not pre-determined, you can use a layout that subclasses Adapter View to populate the layout with views at runtime. A subclass of the Adapter View class uses an Adapter to bind data to its layout.



Recycler View

- The Recycler View widget is a more advanced and flexible version of List View.
- If your app needs to display a scrolling list of elements based on large data sets (or data that frequently changes), you should use Recycler View.
- Several different components work together to display your data.

Intent

- An Intent is a messaging object you can use to request an action from another app component.
- Although intents facilitate communication between components in several ways,
- There are three fundamental use cases:
    - Starting an activity
    - Starting a service
    - Delivering a broadcast

Starting an activity

- An Activity represents a single screen in an app.
- You can start a new instance of an Activity by passing an Intent to startActivity(). The Intent describes the activity to start and carries any necessary data.
- If you want to receive a result from the activity when it finishes, call startActivityForResult().
- Your activity receives the result as a separate Intent object in your activity's onActivityResult() callback.

Intent Types

- Explicit intents specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name. You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start.
    - For example, you might start a new activity within your app in response to a user action, or start a service to download a file in the background.
- Implicit intents do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it.

- For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

The two main components of an Intent are

- **Action**
- The built-in action to be performed, such as
  ACTION_VIEW, ACTION_EDIT, ACTION_CALL, ACTION_SENDTO,... or a *user-created-activity*
- **Data**
- Basic argument needed by the intent to work.
- For instance: a phone number to be called, a picture to be shown, a message to be sent, etc.

Fragments

- A Fragment represents a behavior or a portion of user interface in a FragmentActivity.
- You can combine multiple fragments in a single activity to build a multi-pane UI and reuse a fragment in multiple activities.
- A fragment must always be hosted in an activity and the fragment's lifecycle is directly affected by the host activity's lifecycle.
- you can manipulate each fragment independently, such as add or remove them.
- For example, when the activity is paused, so are all fragments in it, and when the activity is destroyed, so are all fragments. However, while an activity is running (it is in the *resumed* lifecycle state)
- When you add a fragment as a part of your activity layout, it lives in a ViewGroup inside the activity's view hierarchy and the fragment defines its own view layout.
- You can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a <fragment> element, or from your application code by adding it to an existing ViewGroup.

Fragment's Lifecycle

- **onAttach()** Invoked when the fragment has been connected to the host activity.
- **onCreate()** Used for initializing non-visual components needed by the fragment.
- **onCreateView()** *Most of the work is done here*. Called to create the view hierarchy representing the fragment. Usually inflates a layout, defines listeners, and populates the widgets in the inflated layout.
- **onPause()** The session is about to finish. Here you should commit state data changes that are needed in case the fragment is re-executed.
- **onDetach()** Called when the inactive fragment is disconnected from the activity.