Module - 1

Android Key Statistics:

- Android has over three billion active users
- Three quarters of all smartphones in the world run Android
- Over 1.5 billion Android smartphones were shipped last year
- Samsung is the largest Android smartphone manufacturer, followed by Xiaomi and Oppo

Trends in MAD:

- Internet of Things (IoT) App Integration
- Apps For Foldable Devices
- 5G Technology
- Development For Wearable Devices
- Mobile Commerce
- Mobile Wallets
- Augmented reality (aka virtual reality) **AR / VR Will Become More Impressive**
- Predictive Analytics
- Superior App Security
- Chatbots

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

Demand for mobile developers:

- Statistics shows that in India alone there are **Over 20000 MAD Job Openings**
- Companies are moving more heavily into creating a **Consummate App Experience**, driven by projections of smartphone penetration globally
- Smartphone users have transformed many industries, with companies now targeting them as their main audience.

On Demand Industry:

- Laundry service
- Doctors on-demand
- Virtual tutors and coaches
- Food delivery
- House cleaning
- Maintenance services
- Fitness on-demand
- Pet care
- Barber and beauty salon

History of Mobile Operating System:

- Nokia – Symbian OS – 1977
- Microsoft – Windows OS – 1985
- Samsung – Bada OS – 2009
- Blackberry – 1999 – RIM (Research in Motion)
- Apple – IOS – 2007

Android: History (Glimpse):

- Mobile phones are very common today among people of various age groups.
- There are many Operating Systems to power the mobile phones, we will focus on Android.
- Android is based on Linux family of OSs and written in C, C++ and Java Programming language.
- Java lies on the top of its all-development layers and therefore its applications are written in Java.
- To develop an Android application, Android Software Development Kit (Android SDK) is available with all essential tools (Complier, debugger, device emulator and VM to run Android programs).
- Emulator is a virtual device that is used to test or run the application without the availability of hardware.

Android: History (Actual):

- Andy Rubin founded Android Incorporation in Palo Alto, California, United States in October, 2003.
- In 17th August 2005, Google acquired android Incorporation. Since then, it is in subsidiary of Google
- The key employees of Android Incorporation are Andy Rubin, Rich Miner, Chris White and Nick Sears.
- Originally intended for camera but shifted to smart phones later because of low market for camera only.
- In 2007, Google announces the development of android OS.
- In 2008, HTC launched the first android mobile

Android Versions:

Android Version	Android Name	Launch Date
Android 1.0	No codename	September 23th 2008
Android 1.1	No codename	February 9th 2009
Android 1.5	Cupcake	April 27th 2009
Android 1.6	Donut	September 15th 2009
Android 2.0	Eclair	October 26th 2009
Android 2.2	Froyo	May 20th 2010
Android 2.3	Gingerbread	December 6th 2010
Android 3.0	Honeycomb	February 22th 2011
Android 4.0	Ice Cream Sandwich	October 18th 2011
Android 4.1	Jelly Bean	July 9th 2012
Android 4.4	KitKat	October 31th 2013
Android 5.0	Lollipop	November 12th 2014
Android 6.0	Marshmallow	October 5th 2015
Android 7.0	Nougat	August 22th 2016
Android 8.0	Oreo	August 21th 2017
Android 9.0	Pie	August 6th 2018
Android 10	Android Q	September 3th 2019
Android 11	Red Velvet Cake	September 8th 2020
Android 12	Snow Cone	October 17th 2021
Android 13	Tiramisu	August 15th 2022
Android 14	Upside Down Cake	October 4th 2023
Android 15	Vanilla Ice Cream	February 16th 2024

Android: Open Handset Alliance [OHA]:

- Group of companies, who have come together to accelerate innovation in mobile.
- Offer consumers a richer, less expensive, and better mobile experience.
- Android™ - the first complete, open, and free mobile platform.
- Building a better phone for consumers: Building a better mobile phone would enrich the lives of countless people across the globe.
- Innovating in the open: Increased openness will enable everyone in our industry to innovate more rapidly and respond better to consumer's demands.
- Making the vision a reality: commercially deploy handsets and services using the Android Platform

Members

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

Android Features:

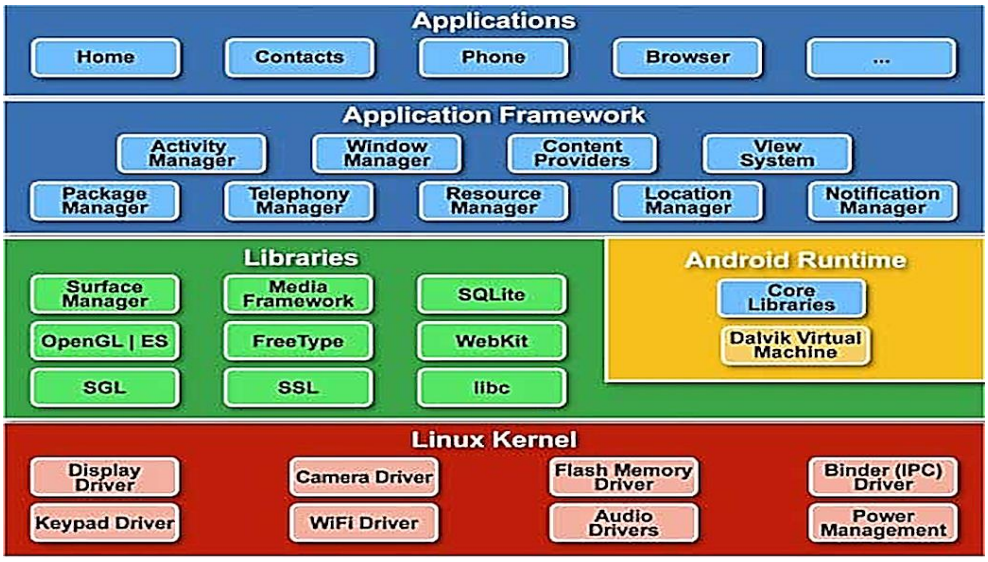
- **Storage** — Uses SQLite, a lightweight relational database, for data storage.
- **Connectivity** — Supports GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth (includes A2DP and AVRCP), Wi-Fi, LTE, and WiMAX.
- **Messaging** — Supports both SMS and MMS.
- **Web browser** — Based on the open source WebKit, 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)

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.

Application Components:

- **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

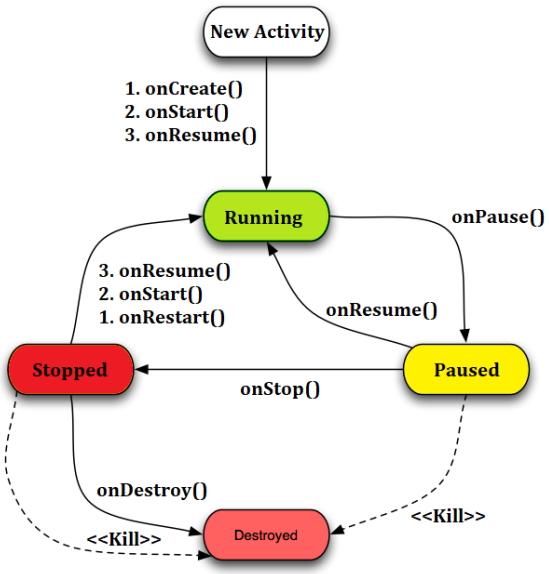
1. **Activity**
   - An Activity object is similar to a WindowsForm. It usually presents a single graphical visual interface (GUI) which in addition to the displaying/collecting of data, provides some kind of 'code-behind' functionality.
   - A typical Android application contains one or more Activity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

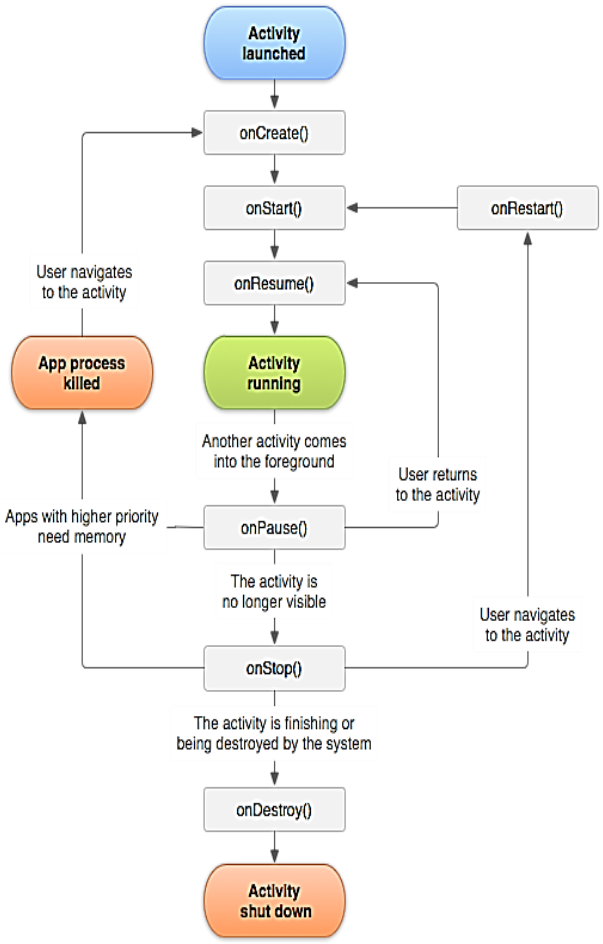
4. **Content Provider**
   - A content provider is a data-centric service that makes persistent datasets available to any number of applications.
   - Common global datasets include: contacts, pictures, messages, audio files, emails.
   - The global datasets are usually stored in a SQLite database (however 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 Lifecycle:



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:

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

Creation of Activity:

- Created using 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:

- A View usually draws something the user can see and interact with.
- The View objects are usually called **"widgets"** and can be one of many subclasses, such as Button or TextView**.**

View Group:

- A View Group is an **invisible container that defines the layout structure** for View and other View Group objects.
- The ViewGroup objects are usually called **"layouts"** can be one of many types that provide a different layout structure, such as **Linear Layout** or **Constraint Layout**

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.

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 Layout Editor to build your XML layout using a drag-and-drop interface.
- ***Instantiate layout elements at runtime.*** Your app can create View and View Group objects (and manipulate their properties) programmatically.

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
- Vertical list will only have one child per row, no matter how wide they are
- horizontal list will only be one row high (the height of the tallest child, plus padding).
- A Linear Layout respects margins between children and the gravity (right, center, or left alignment) of each child.

Relative Layout:

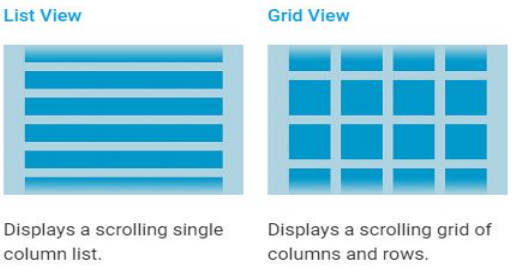
- Relative Layout is a view group that displays child views in relative positions.
- The position of each view can be specified as relative to sibling elements (such as to the left-of or below another view) or in positions relative to the parent Relative Layout area (such as aligned to the bottom, left 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.
- Several nested Linear Layout groups can be replaced by 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
- 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.
- One can build layout with Constraint Layout entirely by drag-and-dropping instead of editing the XML.

Building Layouts with an Adapter

- Used when content of layout is dynamic / not pre-determined to populate 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.
- Used when app needs to display a scrolling list of elements based on large data sets (or data that frequently changes)
- 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.
- 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
- Receive a result from the activity when it finishes, call **startActivityForResult ()**
- Activity receives the result as a separate Intent object in activity's **onActivityResult ()** callback

Intent Types:

1. **Explicit intents:**
    - They specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name.
    - Used to start a component in own app, because the class name of the activity or service that has to be started is known
    - For example: start a new activity within the app in response to a user action, or start a service to download a file in the background.

2. **Implicit intents:**
    - They 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, used to show the user a location on a map, one can use an implicit intent to request that another capable app to show a specified location on a map

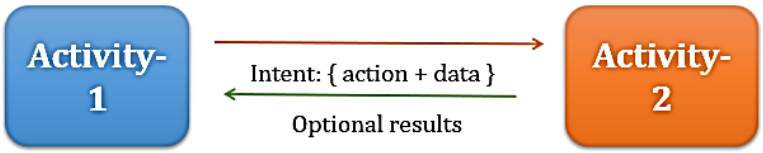
Intent Components:

1. **Action:**

    The built-in action to be performed, such as ***ACTION_VIEW, ACTION_EDIT, ACTION_CALL, ACTION_SENDTO***, …… or a *user-created-activity*

2. **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.



Components of Intent: Android Actions:

List of common actions that Intents can use for launching built-in activities [usually through ***startActivity(Intent)***]

- ACTION_MAIN
- ACTION_VIEW
- ACTION_ATTACH_DATA
- ACTION_EDIT
- ACTION_PICK
- ACTION_CHOOSER
- ACTION_GET_CONTENT
- ACTION_DIAL
- ACTION_CALL
- ACTION_SEND
- ACTION_SENDTO
- ACTION_ANSWER
- ACTION_INSERT
- ACTION_DELETE
- ACTION_RUN
- ACTION_SYNC
- ACTION_PICK_ACTIVITY
- ACTION_SEARCH
- ACTION_WEB_SEARCH
- ACTION_FACTORY_TEST
- ACTION_RUN

Components of Intent: Data:

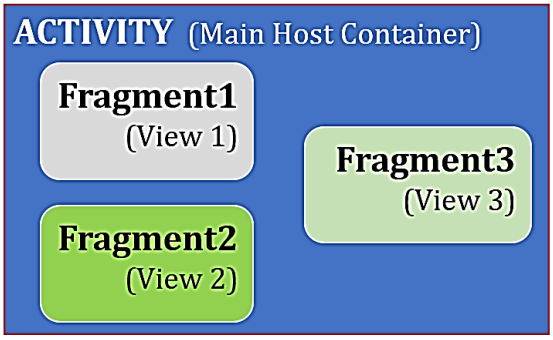
Data is supplied as an URI, i.e. a string whose prefix indicates the composition of the data item.

For instance:

- tel://,
- http://,
- mailto://,
- file://,
- content://,
- geo:,
- audio/,
- media/,
- vnd.android.cursor.dir

Fragments:

- A Fragment in a Fragment Activity represents a specific behaviour or part of the UI.
- Multiple fragments can be combined within a single activity to create a multi-pane UI and reused across different activities.
- Fragments are always hosted within an activity, and their lifecycle is tied to the host activity's lifecycle.
- Each fragment can be manipulated independently, allowing actions like addition or removal.



- Fragments within an activity are paused or destroyed along with the activity itself.
- Fragments live within the activity's view hierarchy, each with its own layout definition.
- Fragments can be added to the activity's layout via XML declaration or programmatically from your application code by adding it to an existing View Group

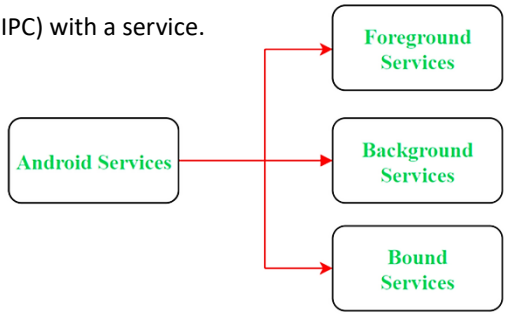
Fragment's Lifecycle:

- **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.

Module – 3

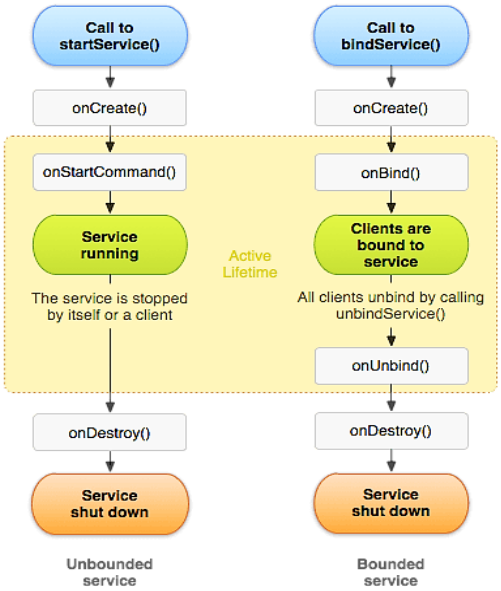
Services:

- A Service is an application component that can perform long-running operations in the background.
- It does not provide a user interface.
- Once started, a service might continue running for some time, even after the user switches to another application.
- A component can bind to a service to interact with it.
- Components can perform inter process communication (IPC) with a service.
- **Examples:**
  - A service can handle network transactions.
  - A service can play music.
  - A service can perform file I/O.
  - A service can interact with a content provider.



- **Types:**
  1. **Foreground Services:**
     - Notify users about ongoing operations through notifications.
     - Users can interact with the service via the notifications.
  2. **Background Services:**
     - Run in the background without user intervention.
     - Don't notify users about their tasks.
     - Users cannot directly access them.
  3. **Bound Services:**
     - Allow other application components (like activities) to bind to them.
     - Perform tasks as long as at least one component is bound.
     - Use **bindService ()** method for binding components to the service.

Lifecycle of Android Services:



1. **onStartCommand ():**
   - Called when a component requests to start a service using **startService ().**
   - Allows explicit stopping of the service using **stopService ()** or **stopSelf ().**
2. **onCreate ():**
   - Called when a service is created, either via **onStartCommand ()** or **onBind ().**
   - Performs one-time setup necessary for the service.

3. **onBind ():**
    - Mandatory method to implement in Android service.
    - Invoked when an application component calls **bindService ()** to bind with the service.
    - Provides user-interface for effective communication by returning an IBinder object.
    - If binding is not required, returns null.
4. **onUnbind ():**
    - Invoked when all clients disconnect from a particular service interface.
5. **onDestroy ():**
    - Called before a service is destroyed, as a final cleanup call.
    - Services must implement this method to clean up resources like registered listeners, threads, receivers, etc.

Broadcasts:

- Apps can send or receive broadcast messages, similar to publish-subscribe design pattern.
- Broadcasts are triggered by specific events, such as system boot-up or device charging, and can be sent by the Android system or custom apps.
- Apps can register to receive specific broadcasts relevant to their functionality.
- The system automatically delivers broadcasts to subscribed apps.
- Broadcasts facilitate inter-app communication and tasks outside the regular user flow.
- Caution is advised to avoid overusing broadcast handling, which can lead to background jobs and performance issues.

System Broadcasts:

- System broadcasts are sent automatically for various system events like switching in /out of airplane mode.
- These broadcasts are received by all apps subscribed to receive the event.
- The **BROADCAST_ACTIONS.TXT** file in the Android SDK lists all system broadcast actions.
- Each broadcast action has a constant field associated with it.
- **ACTION_AIRPLANE_MODE_CHANGED** → android . intent . action . AIRPLANE_MODE

Changes to System Broadcasts:

- Android platform changes affect system broadcasts.
- **Android 9:** The **NETWORK_STATE_CHANGED_ACTION** broadcast no longer includes location or personally identifiable data.
- **Android 8.0:** Additional restrictions are imposed on manifest-declared receivers.
- Apps targeting Android 8.0+ cannot declare receivers for most implicit broadcasts in manifest.
- **Android 7.0+:** Certain broadcasts like **ACTION_NEW_PICTUR**E and **ACTION_NEW_VIDEO** no longer sent.
- Apps targeting Android 7.0+ must register the **CONNECTIVITY_ACTION** broadcast dynamically using **registerReceiver (BroadcastReceiver, IntentFilter)** Declaring receiver manifest won't work.

Receiving System Broadcasts:

- Apps can receive broadcasts through manifest-declared or context-registered receivers.
- Manifest-declared receivers launch the app when the broadcast is sent.
- Declare a receiver in the manifest with the <receiver> element.
- Subclass BroadcastReceiver and implement **onReceive (Context, Intent)** to handle broadcasts.
- Context-registered receivers receive broadcasts as long as their context is valid.
- Create an instance of BroadcastReceiver and IntentFilter for context-registered receivers.
- Register the receiver using **registerReceiver (BroadcastReceiver, IntentFilter).**
- Unregister the receiver using **unregisterReceiver (android.content.BroadcastReceiver**) when no longer needed.

Sending Broadcasts:

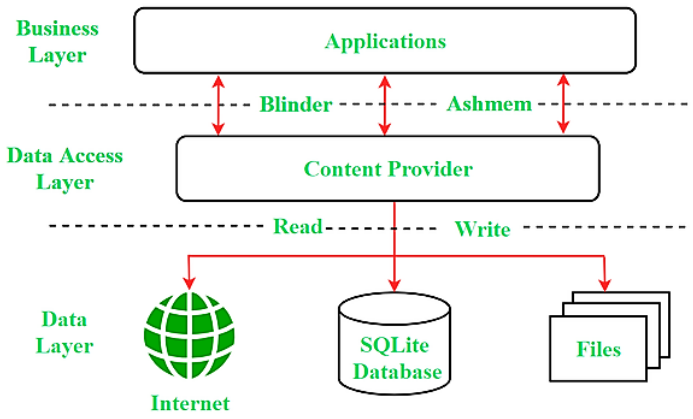
1. **sendOrderedBroadcast (Intent, String):**
    - Sends broadcasts to one receiver at a time.
    - Receivers can propagate results or abort the broadcast.
    - Order of receivers can be controlled with android attribute.
2. **sendBroadcast (Intent):**
    - Sends broadcasts to all receivers in an undefined order. Known as Normal Broadcast.
    - Efficient but receivers can't read results from others or abort the broadcast.
3. **LocalBroadcastManager.sendBroadcast:**
    - Sends broadcasts to receivers within the same app.
    - More efficient, no inter process communication needed.

Delivering Broadcasts:

- A broadcast is a message received by any app.
- The system sends broadcasts for events like boot-up or charging.
- Use **sendBroadcast ()** or **sendOrderedBroadcast ()** to deliver broadcasts.

Content Providers:

- Act as a central repository for application data.
- Allow other applications to securely access and modify data.
- Can store data in various ways like SQLite databases, files, or networks.
- Use permissions to control access to data.



Content URI:

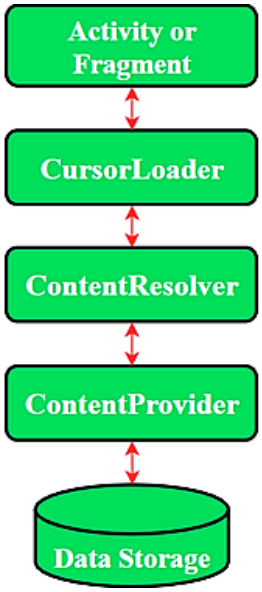
- Unique identifier for accessing data from a content provider.
- **Structure:** content://authority/optionalPath/optionalID
- **content://:** Identifies it as a content URI.
- **authority:** Name of the content provider (e.g., contacts, browser).
- **optionalPath:** Type of data (e.g., audio, video).
- **optionalID:** Specific record identifier (numeric).

CRUD Operations:

- **Create:** Add new data to the content provider.
- **Read:** Fetch data from the content provider.
- **Update:** Modify existing data.
- **Delete:** Remove data from the content provider.

Working of Content Provider:

- App components (Activities, Fragments) use CursorLoader to send queries.
- CursorLoader sends requests (CRUD) to ContentResolver.
- ContentResolver forwards the request to the specific content provider.
- Content provider processes the request and returns the result.



Creating a Content Provider:

- Create a class extending ContentProvider.
- Define a content URI for accessing the provider.
- Create a database (optional) to store data.

Implement six abstract methods of ContentProvider.

- Register the content provider in the manifest (AndroidManifest.xml).
- Abstract Methods of ContentProvider:
    - **Query ():** Fetches data based on arguments and returns a Cursor.
    - **Insert ():** Inserts a new row and returns its content URI.
    - **Update ():** Updates existing rows and returns the number of rows updated.
    - **Delete ():** Deletes rows and returns the number of rows deleted.
    - **getType ():** Returns the MIME type of data for a given URI.
    - **onCreate ():** Initializes the provider when it's created.

Module – 4

Notification:

- Message displayed outside the app's UI for reminders, communication, or timely information.
- Users can interact with notifications by tapping to open or take actions.

Appearances on Device:

- Notifications appear in various locations and formats for user convenience.
- Initially, notifications show as icons in the status bar.
- Users can access more details by swiping down to open the notification drawer.
- In the notification drawer, users can view additional information and take actions.
- **Heads-up Notification:**
    - Introduced in Android 5.0. Appears as a floating window for urgent notifications.
    - Appears only if the device is unlocked.
- **Lock Screen Notifications:**
    - Introduced in Android 5.0. Notifications can appear on the lock screen.
    - Developers can control level of detail or visibility on secure lock screens.
- **App Icon Badge:**
    - Available on Android 8.0 and higher.
    - Coloured badge on app launcher icon indicates new notifications.
    - Long-pressing app icon shows notifications for the app.
    - Users can dismiss or act on notifications from this menu.

Notification Anatomy: **(Additional Details May Appear in Expanded Window)**

- Notification design follows system templates; app defines content.
- Key components include:
    - **Small icon:** Set with **setSmallIcon().**
    - **App name:** Provided by system.
    - **Time stamp:** Provided by system, customizable with **setWhen()** or hidden with **setShowWhen(false).**
    - **Large icon:** Optional, typically for contact photos, set with **setLargeIcon().**
    - **Title:** Optional, set with **setContentTitle().**
    - **Text:** Optional, set with **setContentText().**

Shared Preferences:

- Shared Preferences stores key-value pairs in a file.
- Simple methods for reading and writing data.
- Managed by the framework; files can be private or shared.

Handle Shared Preferences:

- **getSharedPreferences():** Use for multiple shared preference files identified by name, callable from any Context.
- **getPreferences():** Use from an Activity for a single shared preference file belonging to the activity, no need to specify a name.

Writing into Shared Preferences:

- Call **edit ()** on SharedPreferences to create a SharedPreferences.Editor.
- Use methods like **putInt ()** and **putString ()** to pass keys and values. Finally, call **apply ()** or **commit ()** to save the changes.

Read from Shared Preferences:

- Call methods like **getInt ()** and **getString ()** on SharedPreferences.
- Provide the key for the value you want and optionally a default value if the key isn't present.

Save Data using SQLite:

- Database storage is suitable for structured or repeating data like contacts.
- Android provides database APIs in the android.database.sqlite package.
- However, these APIs are low-level and require significant effort to use:
  - No compile-time verification of raw SQL queries.
  - Manual updating of SQL queries when data structure changes.
  - Boilerplate code needed for converting between SQL queries and data objects.

Create database using SQL Helper:

- Database schema defined with SQL statements like **CREATE TABLE** and **DROP TABLE.**
- Database stored in the app's private folder, ensuring data security.
- SQLiteOpenHelper class provides APIs for managing the database.
- Subclass SQLiteOpenHelper, override **onCreate ()** and **onUpgrade ()** methods.
- Optionally implement **onDowngrade ()** or **onOpen ()** methods.
- Instantiate your subclass of SQLiteOpenHelper to access the database.

Put Information into Database:

- Insert data into the database using **insert ()** method.
- Obtain the database in write mode using **getWritableDatabase ().**
- Create a ContentValues object and populate it with column-value pairs.
- Use **insert ()** method, passing table name and ContentValues.
- Optionally handle cases where ContentValues is empty.
- **Insert ()** returns the ID of the newly created row, or -1 on error.

Read Information from Database:

- Use **query ()** method to read from the database. Pass selection criteria, desired columns to **query ().**
- Results are returned as a Cursor object. Use Cursor move methods to navigate rows.
- Call **moveToNext ()** to move to the next row. Use Cursor get methods to read column values.
- Pass column index obtained from **getColumnIndex ()** or **getColumnIndexOrThrow ().**
- Call **close ()** on the cursor to release resources when finished.

Delete Information from Database:

- To delete rows, provide selection criteria to **delete ()** method.
- Selection criteria similar to those in **query ()** method.
- Selection criteria divided into clause and arguments.
- Clause defines columns and tests to be performed.
- Arguments are values tested against and bound into the clause. Result is immune to SQL injection.
- Example: Define selection criteria and arguments, then call **delete ()** method.

Update Information in Database:

- Use **update()** method to modify a subset of database values.
- Updating combines ContentValues syntax of **insert()** with WHERE syntax of delete().

Module – 5

Graphics:

- Android framework provides graphics rendering APIs for 2D and 3D.
- These APIs interact with manufacturer implementations of graphics drivers.
- Understanding these APIs at a higher level is important for efficient graphics rendering.

Android Rendering Options:

- **Canvas API:**
  - Standard rendering for typical SDK applications uses View objects.
  - Calls to each View's **onDraw ()** method with a Canvas parameter.
- **RenderScript:**
  - Introduced in Android 3.0, targeted high-performance 3D rendering and compute operations.
  - Executes native code on the device, still cross-platform.
  - Extensions placed into the application package.
- **OpenGL Wrappers:**
  - OpenGL APIs available at SDK Level, not recommended for complex scenes.
  - Music application in Android 3.0 used this approach.
- **NDK OpenGL:**
  - No access to View objects, events, or SDK APIs.
  - Suitable for specific purposes like game development.
  - Compiles applications to specific CPU architectures.

Animations:

- Animation API introduced in Android 3.0 enables changing object properties over time.
- Offers flexibility to create visually attractive animations and transitions.
- **Types:**
  - **View Animation:** Adds animation to specific views, performing tweened animation. Tweened animation calculates size, rotation, start point, and endpoint.
  - **Property Animation:** Robust framework for animating almost anything.
  - **Drawable Animation:** Animates one image over another. Involves loading a series of drawable to create an animation effect.
- **Methods**
  - **Start ():** Start the animation
  - **setDuration (long duration):** Sets the duration of an animation
  - **getDuration ():** Gets the duration
  - **End ():** Ends the animation
  - **Cancel ():** Cancels the animation

Animation File in Android Studio:

- Create an **"anim"** folder in Android Studio's resource directory.
- There are two options for resource types:
  - **animator/:** XML files defining property animations.
  - **anim/:** XML files defining tween animations.
- **Common Properties:**

Property	Description
alpha	Fade in or out
rotation, rotationX, rotationY	Spin or flip
scaleX ,scaleY	Grow or shrink
x,y,z	Position
translationX, translationY, translationZ (API 21+)	Offset from Position

Sending Emails:

- Email sending in Android utilizes the default Email app.
- Use **Intent.ACTION_SEND** action to launch the Email client.
- Specify recipient's URI with **setData ()** method and content type with **setType ()** method.
- Attachments and extras like TO, SUBJECT, CC, BCC can be included using **putExtra ()** method.

Sending SMS:

- **Two methods:** SMSManager API or Intents.
- SMSManager API sends directly from the app.
- Intents with **ACTION_VIEW** invoke built-in SMS app.

Sending SMS using SMSManager:

- Obtain SmsManager instance with **SmsManager.getDefault().**
- Use **sendTextMessage ()** method to send SMS.
- Requires **SEND_SMS** permission in manifest.

Sending SMS using Intent:

- Create Intent object with ACTION_VIEW.
- Set recipient's address and message body as extras.
- Set type to **"vnd.android-dir/mms-sms".**
- Requires SEND_SMS permission in manifest.

Location Based Services:

- Location-Based Services (LBS) in Android provide features like current location detection, nearby places display, and geofencing.
- LBS fetch’s location using device's GPS, WIFI, or Cellular Networks.
- To build an app with LBS, access the Google Play Services Module.
- Utilize the Location Framework, offering methods, classes, and interfaces for easier implementation.

Components of Location Based Services:

- **LocationManager Class:** Provides access to Location Service.
- **LocationListener Interface:** Receives updates from Location Manager.
- **LocationProvider Class:** Provides device location.
- **Location Class:** Carries information like latitude, longitude, accuracy, altitude, and speed.