Chapter 3

Android Components

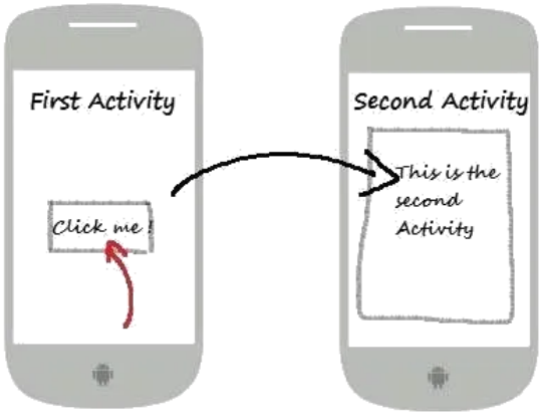
3.1 Android Application Components

- In android, application components are the basic building blocks of an application and these components will act as an entry point to allow system or user to access our app.
- Some components depend on others.
- Each type serves a distinct purpose and has a distinct lifecycle that defines how a component is created and destroyed.
- The following are the basic core application components that can be used in Android application.
  - **Activities**
  - **Services**
  - **Broadcast Receivers**
  - **Content Providers**
  - **Intents**
- All these application components are defined in the android app description file **(AndroidMainfest.xml)** like as shown below.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest …..>
    <application android:allowBackup="true" android:icon="@mipmap/ic_launcher" ……>
        <activity android:name=".MainActivity" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
       …….
    </application>
</manifest>
```

3.2 Activity

- Activity is the major component of the android app.
- An activity is the entry point for interacting with the user.
- It represents a single screen with a user interface.
- It contains all the user interface components (Button, TextInput etc.) in one screen.
- An application usually consists of multiple activities that are loosely bound to each other.



- For example, an email app might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails.
- Although the activities work together to form a cohesive user experience in the email app, each one is independent of the others.
- A different app can start any one of these activities if the email app allows it.
- For example, a camera app might start the activity in the email app for composing a new email to let the user share a picture.
- An activity facilitates the following key interactions between system and app:
  - Keeping track of what the user currently cares about—what is on-screen—so that the system keeps running the process that is hosting the activity.
  - Knowing which previously used processes contain stopped activities the user might return to and prioritizing those processes more highly to keep them available.
  - Helping the app handle having its process killed so the user can return to activities with their previous state restored.
  - Providing a way for apps to implement user flows between each other, and for the system to coordinate these flows. The primary example of this is sharing.

3.2.1 Key Characteristics of Activities

1. **User Interface (UI)**: Each activity typically has its own user interface, which is defined in an XML layout file. The UI consists of various elements like buttons, text views, images, and more.
2. **Lifecycle**: Activities have a well-defined lifecycle consisting of various states, including "created," "started," "resumed," "paused," "stopped," and "destroyed." These states allow you to manage the behavior of your app as it interacts with the user and the system.
3. **Intents**: Activities are typically started and navigated to using intents. An intent is a messaging object that specifies the action to be performed, such as launching a new activity.
4. **Stack**: Android maintains a stack of activities, known as the "activity stack," to keep track of the navigation history. Activities are pushed onto the stack when started and popped off when finished.

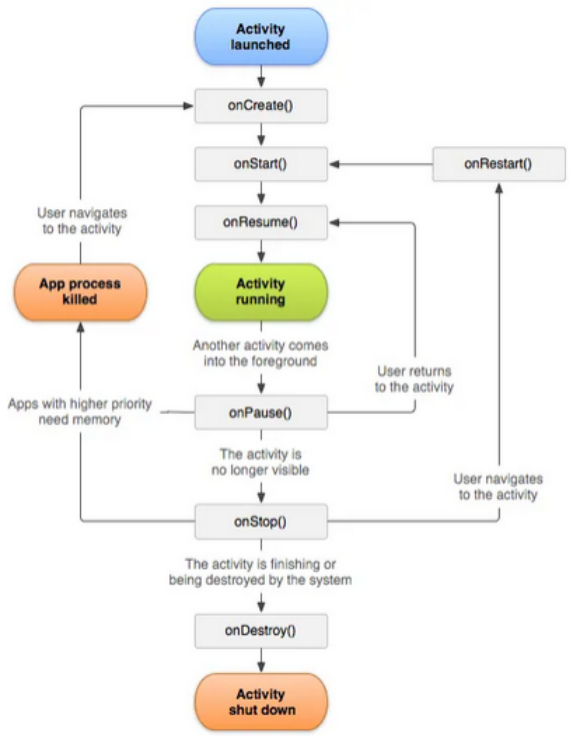
3.2.2 Common Uses of Activities

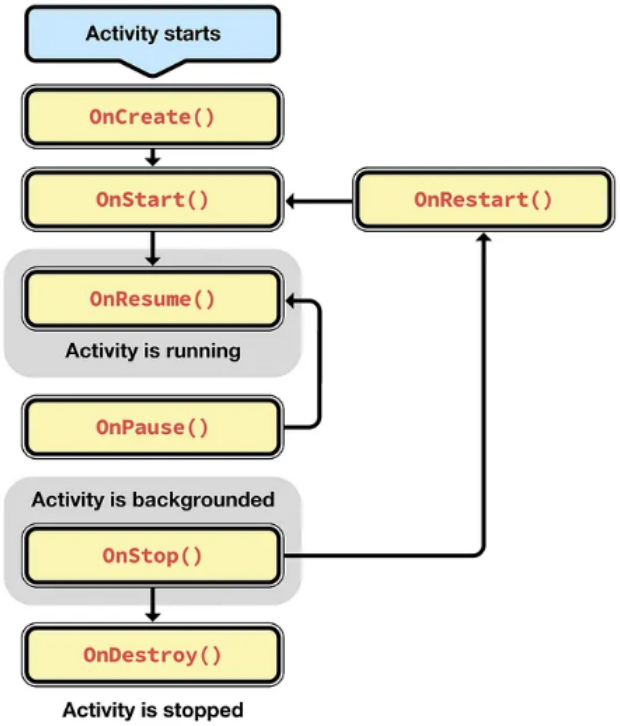
1. **Main Screen**: The entry point of many Android apps is an activity, often referred to as the main or launcher activity. It's the first screen users see when they launch the app.
2. **UI Interaction**: Activities are used for displaying forms, accepting user input, and responding to user actions, such as button clicks.
3. **Subscreens**: In complex apps, multiple activities are used to represent different screens or subscreens, making it easier to organize and manage the app's functionality.
4. **Navigation**: Activities are used for navigation between different parts of the app. Users can move from one activity to another to perform specific tasks.

3.2.3 Activity Lifecycle

- Typically, one activity in an application is specified as the "main" activity, which is presented to the user when launching the application for the first time.
- Each activity can then start another activity in order to perform different actions. Each time a new activity starts, the previous activity is stopped, but the system preserves the activity in a stack (the "back stack").
- When a new activity starts, it is pushed onto the back stack and takes user focus. The back stack abides to the basic "last in, first out" stack mechanism, so, when the user is done with the current activity and presses the Back button, it is popped from the stack (and destroyed) and the previous activity resumes. (The back stack is discussed more in the Tasks and Back Stack document.)
- When an activity is stopped because a new activity starts, it is notified of this change in state through the activity's lifecycle callback methods.
- There are several callback methods that an activity might receive, due to a change in its state—whether the system is creating it, stopping it, resuming it, or destroying it—and each callback provides you the opportunity to perform specific work that's appropriate to that state change.
- For instance, when stopped, your activity should release any large objects, such as network or database connections. When the activity resumes, you can reacquire the necessary resources and resume actions that were interrupted. These state transitions are all part of the activity lifecycle.
- An activity goes through different number of states, and you can use some callbacks to manage all these transitions between state. There are seven key lifecycle methods through which every Activity goes depending upon its state. They are:

  1. **onCreate()**
  2. **onStart()**
  3. **onResume()**
  4. **onPause()**
  5. **onStop()**
  6. **onDestroy()**
  7. **onRestart()**


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

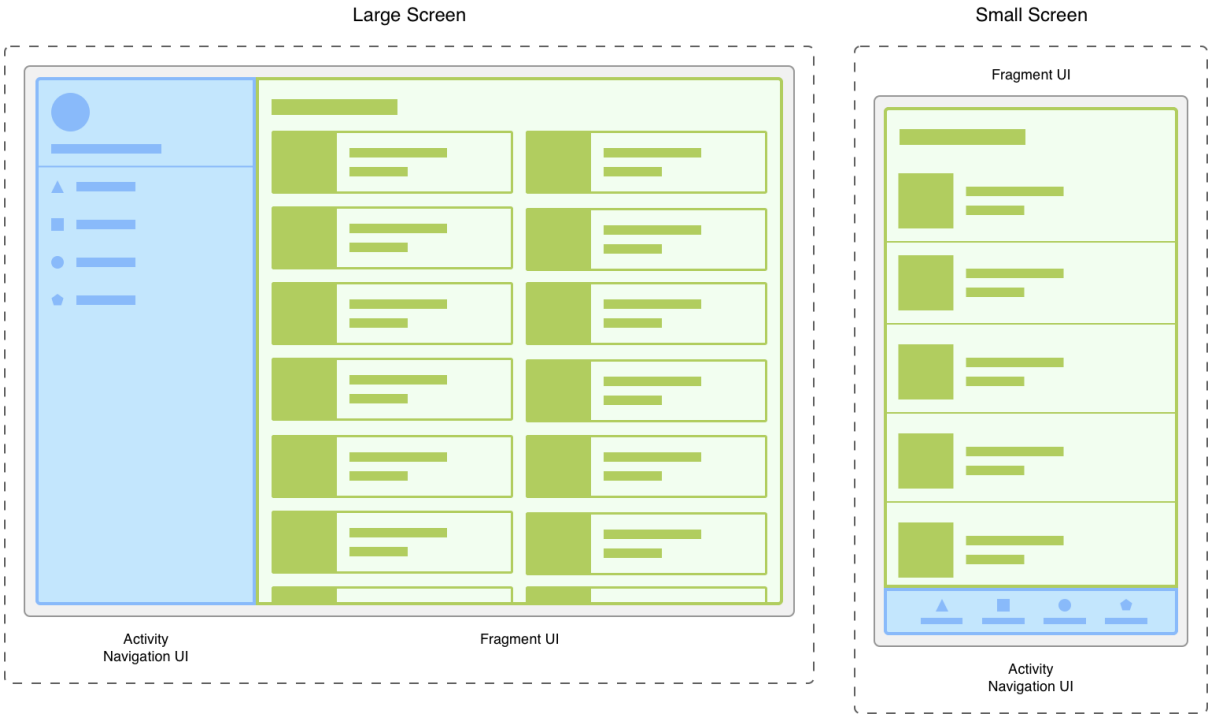

Activity starts
OnCreate()
OnStart()
OnRestart()
OnResume()
Activity is running
OnPause()
Activity is backgrounded
OnStop()
OnDestroy()
Activity is stopped

- **onCreate()**:
  - Whenever an Activity starts running, the first method to get executed is onCreate().
  - This method is executed only once during the lifetime or This method is called when the activity is first created.
  - It is typically used for initialization, setting up the UI, and performing one-time setup.
  - After onCreate() method, the onStart() method is executed. onStart()
- **onStart()**:
  - During the execution of onStart() method, the Activity is not yet rendered on screen but is about to become visible to the user.
  - In this method, we can perform any operation related to UI components.
  - Called when the activity becomes visible but is not yet in the foreground. At this stage, the activity is "started" but not "resumed."
- **onResume()**:
  - The activity is in the foreground and ready to interact with the user.
  - This is where you should start animations, play audio, and handle user input.
  - When the Activity finally gets rendered on the screen, onResume() method is invoked.
  - At this point, the Activity is in the active state and is interacting with the user.
- **onPause()**:
  - Called when the activity loses focus but is still visible. This is a good place to save data, stop animations, and release resources.
  - If the activity loses its focus and is only partially visible to the user, it enters the paused state.
  - During this transition, the onPause() method is invoked.
  - In the onPause() method, we may commit database transactions or perform light-weight processing before the Activity goes to the background.
- **onStop()**
  - The activity is no longer visible. This can happen when another activity comes to the foreground or when the app is minimized.
  - From the active state, if we hit the Home key, the Activity goes to the background and the Home Screen of the device is made visible.
  - During this event, the Activity enters the stopped state. Both onPause() and onStop() methods are executed.
- **onDestroy()**
  - Called when the activity is being destroyed. This is where you should release any resources that won't be needed anymore.
  - When an activity is destroyed by a user or Android system, onDestroy() function is called.
- **onRestart()**:
  - Called when the activity is restarting after being stopped.
  - This is an opportunity to reinitialize components.

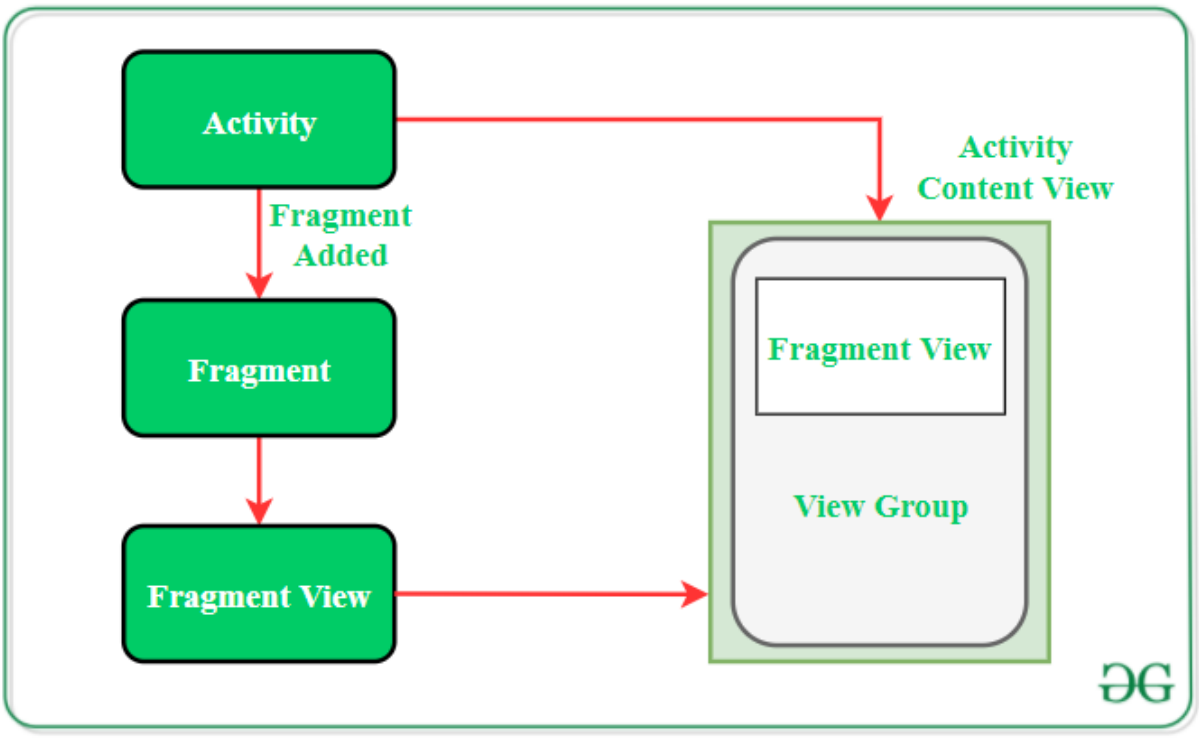
1. When the Activity comes back to focus from the paused state, onResume() is invoked.
2. When we reopen any app(after pressing Home key), Activity now transits from stopped state to the active state. Thus, onStart() and onResume() methods are invoked.
3. Note: onCreate() method is not called, as it is executed only once during the Activity life-cycle.
4. To destroy the Activity on the screen, we can hit the Back key. This moves the Activity into the destroyed state. During this event, onPause(), onStop() and onDestroy() methods are invoked.
5. Note: Whenever we change the orientation of the screen i.e from portrait to landscape or vice-versa, lifecycle methods start from the start i.e from onCreate() method. This is because the complete spacing and other visual appearance gets changed and adjusted.

3.3 Fragment

- In Android, the fragment is the part of Activity which represents a portion of User Interface(UI) on the screen.
- Fragments were added in Honeycomb version of Android i.e API version 11.
- It is the modular section of the android activity that is very helpful in creating UI designs that are flexible in nature and auto-adjustable based on the device screen size.
- The UI flexibility on all devices improves the user experience and adaptability of the application.
- An activity can contain multiple fragments. In other words, we can also combine multiple Fragments in a single activity to build a multi-plane UI.
- A Fragment can be used in multiple activities.
- Fragments can be added, removed, or replaced dynamically i.e., while activity is running.
- Consider an app that responds to various screen sizes. On larger screens, you might want the app to display a static navigation drawer and a list in a grid layout. On smaller screens, you might want the app to display a bottom navigation bar and a list in a linear layout.
- Managing these variations in the activity is unwieldy. Separating the navigation elements from the content can make this process more manageable. The activity is then responsible for displaying the correct navigation UI, while the fragment displays the list with the proper layout.



- In the above figure, two versions of the same screen on different screen sizes. On the left, a large screen contains a navigation drawer that is controlled by the activity and a grid list that is controlled by the fragment. On the right, a small screen contains a bottom navigation bar that is controlled by the activity and a linear list that is controlled by the fragment.
- Fragments can exist only inside an activity as its lifecycle is dependent on the lifecycle of host activity.



- For example, if the host activity is paused, then all the methods and operations of the fragment related to that activity will stop functioning, thus fragment is also termed as **sub-activity**.
- Fragments has its own layout and its own behaviour with its own life cycle callbacks.

3.3.1 Need of Fragments in Android

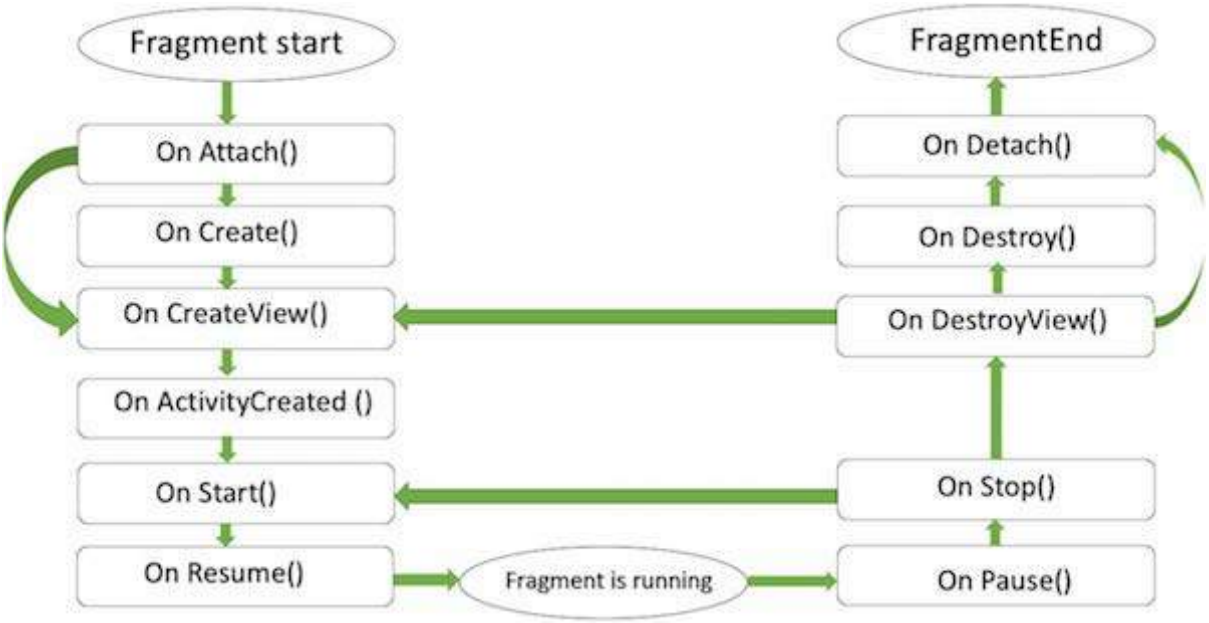
- Before the introduction of Fragment's we can only show a single Activity on the screen at one given point of time so we were not able to divide the screen and control different parts separately. With the help of Fragment's we can divide the screens in different parts and controls different parts separately.
- By using Fragments we can comprise multiple Fragments in a single Activity. Fragments have their own events, layouts and complete life cycle. It provide flexibility and also removed the limitation of single Activity on the screen at a time.

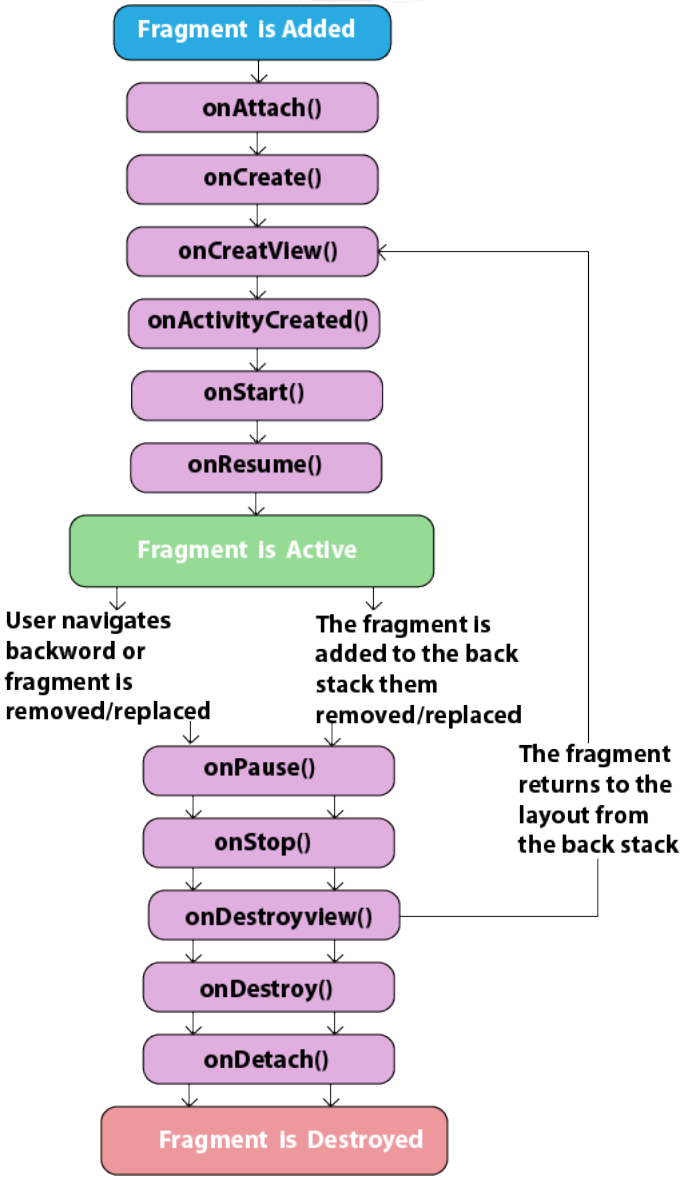
3.3.2 Types of Android Fragments

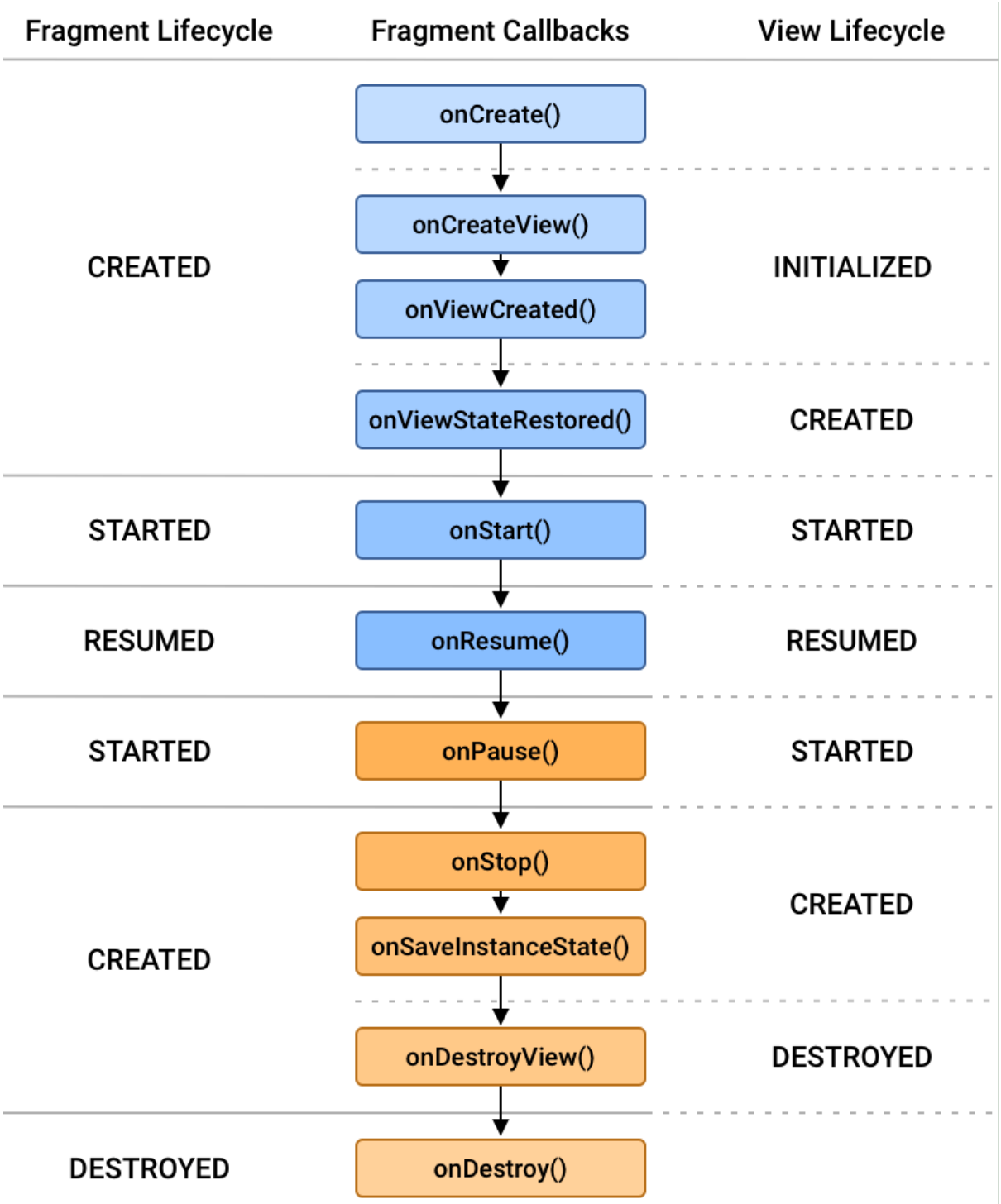
- **Single Fragment**: Display only one single view on the device screen. This type of fragment is mostly used for mobile phones.
- **List Fragment**: This Fragment is used to display a list-view from which the user can select the desired sub-activity. The menu drawer of apps like Gmail is the best example of this kind of fragment.
- **Fragment Transaction**: This kind of fragments supports the transition from one fragment to another at run time. Users can switch between multiple fragments like switching tabs.

3.3.3 Lifecycle of Fragments

The fragment is contained inside the activity and links closely to the lifecycle of an activity. The main difference between fragment and activity lifecycles is that activity creates only one view for the entire lifetime but fragment views can be recreated and even dynamically changed during the lifetime.


Fragment start
On Attach()
On Create()
On CreateView()
On ActivityCreated ()
On Start()
On Resume()
Fragment is running
FragmentEnd
On Detach()
On Destroy()
On DestroyView()
On Stop()
On Pause()


Fragment Is Added
onAttach()
onCreate()
onCreatView()
onActivityCreated()
onStart()
onResume()
Fragment Is Active
User navigates backword or fragment is removed/replaced
The fragment is added to the back stack them removed/replaced
onPause()
onStop()
onDestroyview()
onDestroy()
onDetach()
Fragment Is Destroyed
The fragment returns to the layout from the back stack


Fragment Lifecycle
Fragment Callbacks
View Lifecycle
onCreate()
onCreateView()
onViewCreated()
CREATED
INITIALIZED
onViewStateRestored()
CREATED
STARTED
onStart()
STARTED
RESUMED
onResume()
RESUMED
STARTED
onPause()
STARTED
onStop()
onSaveInstanceState()
CREATED
CREATED
onDestroyView()
DESTROYED
DESTROYED
onDestroy()

3.3.4 Fragment Lifecycle Callback Methods and Description

- **onAttach()**: The very first method to be called when the fragment has been associated with the activity. This method executes only once during the lifetime of a fragment.

  When we attach fragment(child) to Main(parent) activity then it call first and then not call this method any time(like you run an app and close and reopen) simple means that this method call only one time.

- **onCreate()** This method initializes the fragment by adding all the required attributes and components.

- **onCreateView()** System calls this method to create the user interface of the fragment. The root of the fragment's layout is returned as the View component by this method to draw the UI.

  You should inflate your layout in onCreateView but shouldn't initialize other views using findViewById in onCreateView.

- **onViewCreated()** It indicates that the activity has been created in which the fragment exists. View hierarchy of the fragment also instantiated before this function call.

- **onStart()** The system invokes this method to make the fragment visible on the user's device.

- **onResume()** This method is called to make the visible fragment interactive.

- **onPause()** It indicates that the user is leaving the fragment. System call this method to commit the changes made to the fragment.

- **onStop()** Method to terminate the functioning and visibility of fragment from the user's screen.

  onDestroyView() System calls this method to clean up all kinds of resources as well as view hierarchy associated with the fragment. It will call when you can attach new fragment and destroy existing fragment Resoruce

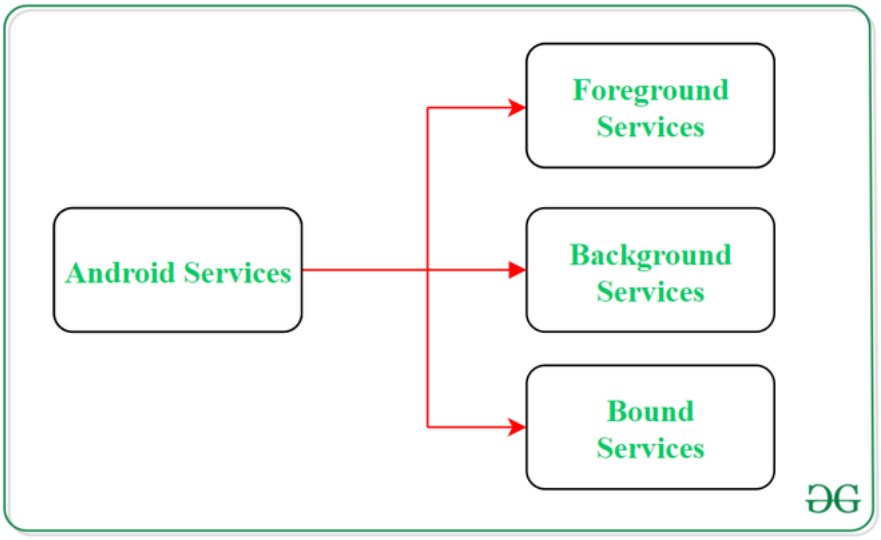
- **onDestroy()** It is called to perform the final clean up of fragment's state and its lifecycle.

- **onDetach()** The system executes this method to disassociate the fragment from its host activity.

  It will call when your fragment Destroy(app crash or attach new fragment with existing fragment)

3.4 Services

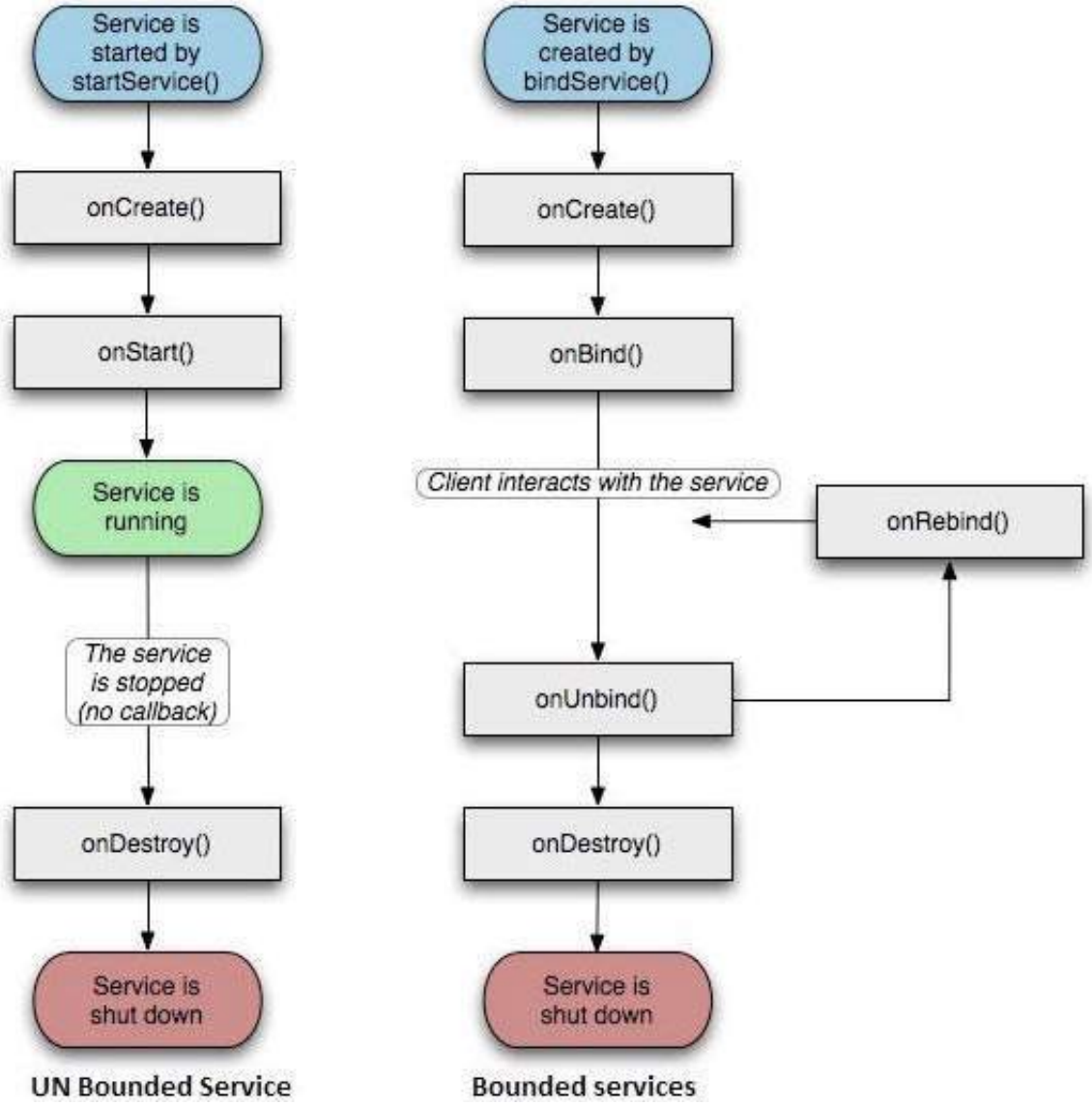
- A service is a general-purpose entry point for keeping an app running in the background for all kinds of reasons.
- It is a component that runs in the background to perform long-running operations or to perform work for remote processes.
- A service does not provide a user interface.
- **For example**: A service might play music in the background while the user is in a different app, or it might fetch data over the network without blocking user interaction with an activity.
- Another component, such as an activity, can start the service and let it run or bind to it to interact with it.



- There are two types of services that tell the system how to manage an app:
  - **Started services**
    - Foreground Services
    - Background Services
  - **Bound services**
- **Started services** tell the system to keep them running until their work is completed. This might be to sync some data in the background or play music even after the user leaves the app.
- Syncing data in the background or playing music represent different types of started services, which the system handles differently called **Foreground Service**.
- Services that notify the user about its ongoing operations are termed as **Foreground Services**.
- Users can interact with the service by the notifications provided about the ongoing task.
- For Example: Such as in downloading a file, the user can keep track of the progress in downloading and can also pause and resume the process.
- Music playback is something the user is directly aware of, and the app communicates this to the system by indicating that it wants to be in the foreground, with a notification to tell the user that it is running.
- In this case, the system prioritizes keeping that service's process running, because the user has a bad experience if it goes away.
- A regular **background service** is not something the user is directly aware of, so the system has more freedom in managing its process.

- **Background services** do not require any user intervention. These services do not notify the user about ongoing background tasks and users also cannot access them. The process like schedule syncing of data or storing of data fall under this service.
- It might let it be killed, restarting the service sometime later, if it needs RAM for things that are of more immediate concern to the user.
- **Bound services** run because some other app (or the system) has said that it wants to make use of the service. A bound service provides an API to another process, and the system knows there is a dependency between these processes. So if process A is bound to a service in process B, the system knows that it needs to keep process B and its service running for A. Further, if process A is something the user cares about, then it knows to treat process B as something the user also cares about.
- Bound service allows the components of the application like activity to bound themselves with it. Bound services perform their task as long as any application component is bound to it. More than one component is allowed to bind themselves with a service at a time. In order to bind an application component with a service bindService() method is used.
- Because of their flexibility, services are useful building blocks for all kinds of higher-level system concepts. Live wallpapers, notification listeners, screen savers, input methods, accessibility services, and many other core system features are all built as services that applications implement and the system binds to when they run.

3.4.1 Services Lifecycle



- To create an service, you create a Java class that extends the Service base class or one of its existing subclasses.
- The Service base class defines various callback methods and the most important are given below.
- You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.
- **Callback Methods and Description**
  - **onStartCommand()**: The system calls this method when another component, such as an activity, requests that the service be started, by calling startService(). If you implement this method, it is your responsibility to stop the service when its work is done, by calling stopSelf() or stopService() methods.
  - **onBind()**: The system calls this method when another component wants to bind with the service by calling bindService(). If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an IBinder object. You must always implement this method, but if you don't want to allow binding, then you should return null.
  - **onUnbind()**: The system calls this method when all clients have disconnected from a particular interface published by the service.
  - **onRebind()**: The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its onUnbind(Intent).
  - **onCreate()**: The system calls this method when the service is first created using onStartCommand() or onBind(). This call is required to perform one-time set-up.
  - **onDestroy()**: The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.

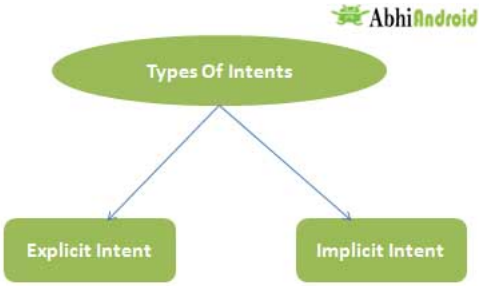
3.5 Intent

- The dictionary meaning of intent is intention or purpose. So, it can be described as the intention to do action.
- Intent are the objects which is used in android for passing the information among Activities in an Application and from one app to another also. Intent are used for communicating between the Application components and it also provides the connectivity between two apps.
- In Android, it is quite usual for users to witness a jump from one application to another as a part of the whole process,
- For example,
  - searching for a location on the browser and witnessing a direct jump into Google Maps
  - receiving payment links in Messages Application (SMS) and on clicking jumping to PayPal or GPay (Google Pay)
- Intent is the message communicating between the components of an Application and also from one application to another application such as activities, content providers, broadcast receivers, services etc.
- This process of taking users from one application to another is achieved by passing the Intent to the system.
- Intents, in general, are used for navigating among various activities within the **same application**, but note, is not limited to one single application, i.e., they can be utilized from **moving from one application to another as well**.
- Android intents are mainly used to:
  - Start the service
  - Launch an activity
  - Display a web page
  - Display a list of contacts
  - Broadcast a message
  - Dial a phone call etc.

3.5.1 Types of Android Intents

There are two types of intents in android

1. **Implicit** intent
2. **Explicit** intent

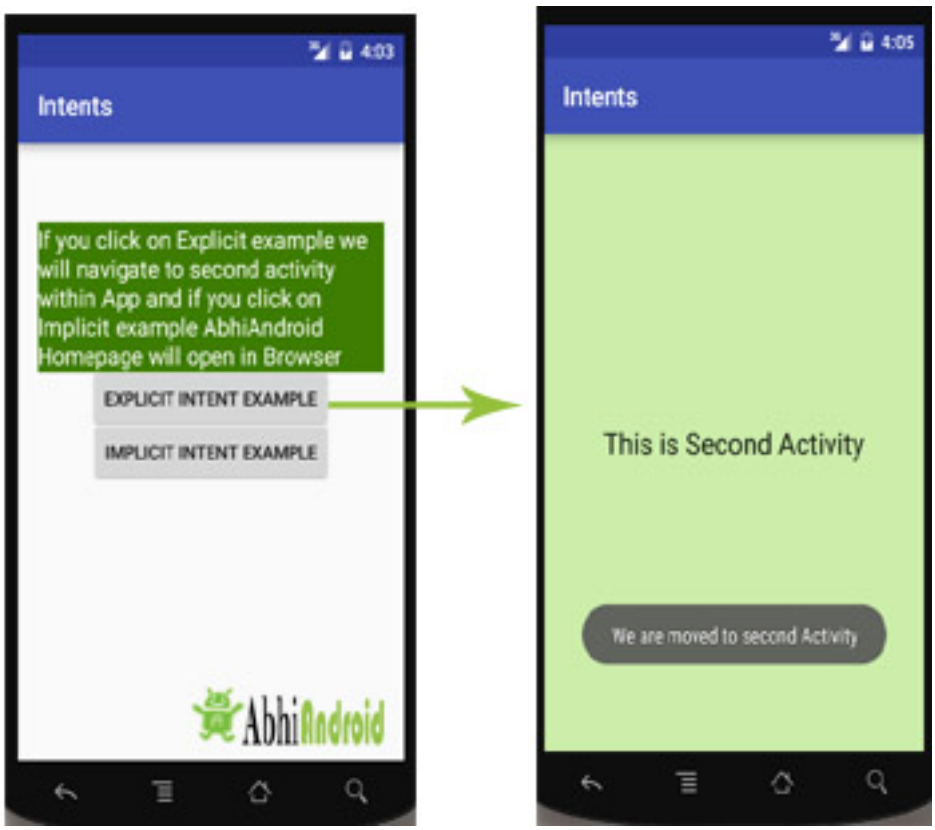


3.5.1.1 Explicit Intent

- **Explicit Intents** are used to connect the application internally. In Explicit we use the name of component which will be affected by Intent. For Example: If we know class name then we can navigate the app from One Activity to another activity using Intent. In the similar way we can start a service to download a file in background process.
- Explicit Intent work internally within an application to perform navigation and data transfer. The below given code snippet will help you understand the concept of Explicit Intents
- Explicit Intent work internally within an application to perform navigation and data transfer. The below given code snippet will help you understand the concept of Explicit Intents

```
Intent intent = new Intent(getApplicationContext(), SecondActivity.class);
startActivity(intent);
```

- Example, Here SecondActivity is the JAVA class name where the activity will now be navigated. Example with code in the end of this post will make it more clear.



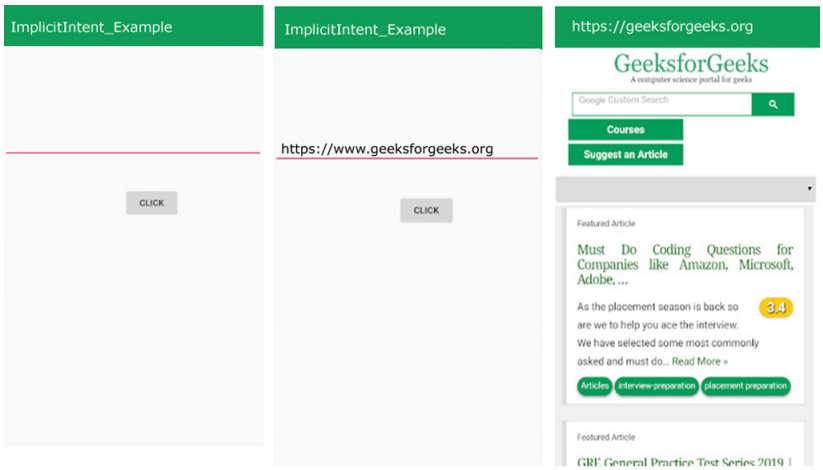
3.5.1.2 Implicit Intent

- Explicit Intents need to specify the name of the component.
- In Implicit intent, we just specify the Action which has to be performed and further this action is handled by the component of another application.
- Implicit Intent doesn't specify the component. In such a case, intent provides information on available components provided by the system that is to be invoked.
- For example, you may write the following code to view the webpage.

Syntax:

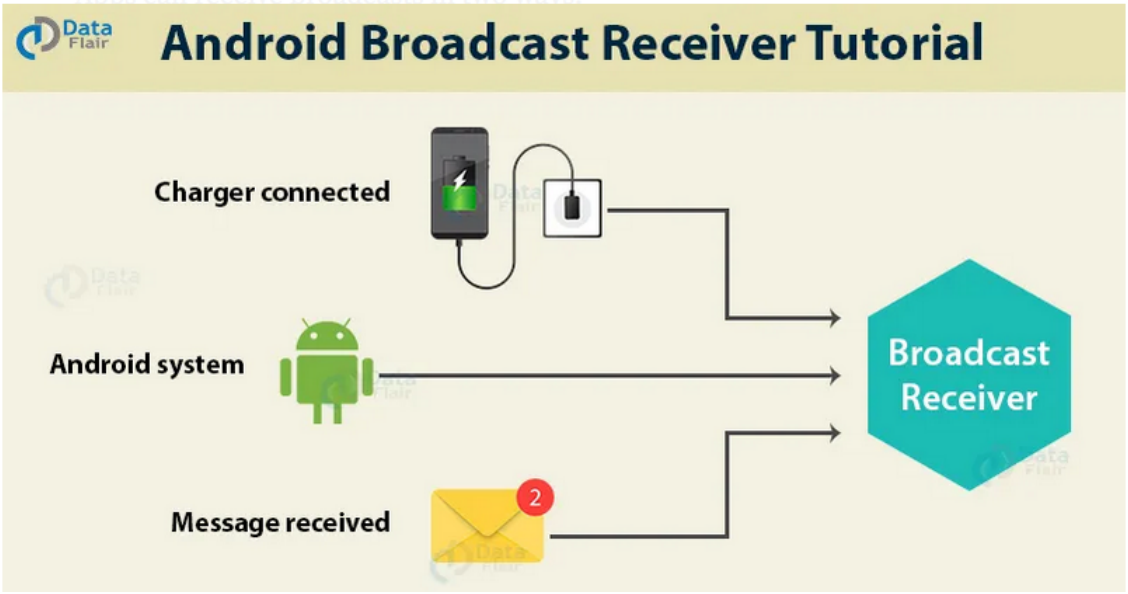
```
Intent intent=new Intent(Intent.ACTION_VIEW);
intent.setData(Uri.parse("https://www.geeksforgeeks.org/"));
startActivity(intent);
```

- For Example: In the below images, no component is specified, instead, an action is performed i.e. a webpage is going to be opened. As you type the name of your desired webpage and click on the 'CLICK' button. Your webpage is opened.



3.6 Broadcast Receiver

- Broadcast Receivers are one of the major component of the android.
- Android apps can send or receive broadcast messages from the Android system and other Android apps, similar to the publish-subscribe design pattern. These broadcasts are sent when an event of interest occurs.
- To explain briefly, Broadcast Receivers answer to broadcast messages from another applications or system.
- For example,
  - Sending a low battery message
  - Screen turned off the message to the app
  - Deliver other apps know that some data downloaded to the device and available for use.
  - The most common example is your alarm application. Whenever you keep an alarm, you leave that application or simple terms, close that application. Still, at a particular time, you receive the notification.



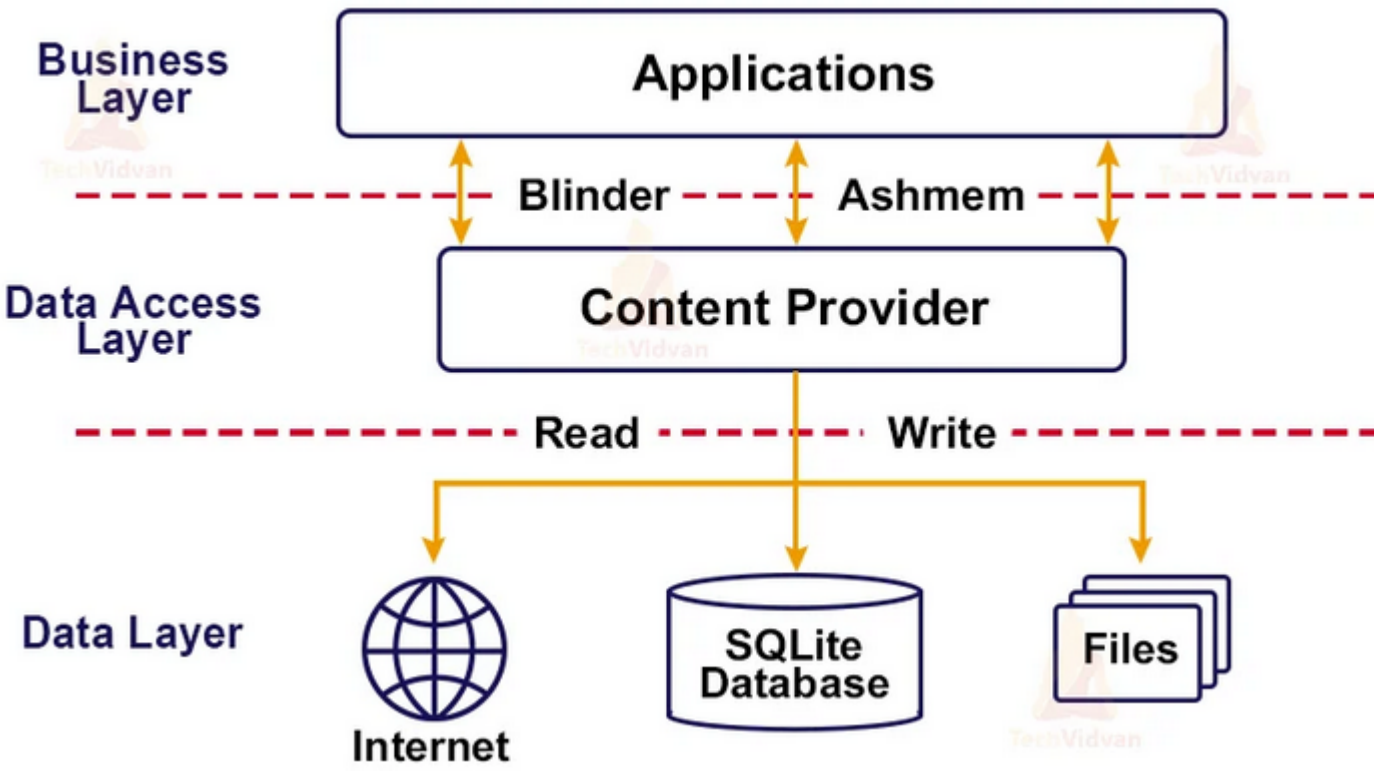
- It's not always necessary for your applications to be active to receive some notifications. Most of the time, these broadcasts are system originated and didn't require the app's user interface.
- Even though broadcast receivers do not show user interface, they might create status bar notification to user for events.
- There are mainly two types of Broadcast Receivers:
  - **Static Broadcast Receivers**: These types of Receivers are declared in the manifest file and works even if the app is closed.
  - **Dynamic Broadcast Receivers**: These types of receivers work only if the app is active or minimized.

- There are several system generated events defined as final static fields in the Intent class. The following table lists a few important system events.

Event	Description
android.intent.action.BOOT_COMPLETED	It raises an event, once boot completed.
android.intent.action.POWER_CONNECTED	It is used to trigger an event when power connected to the device.
android.intent.action.POWER_DISCONNECTED	It is used to trigger an event when the power got disconnected from the device.
android.intent.action.BATTERY_LOW	It is used to call an event when battery is low on device.
android.intent.action.BATTERY_OKAY	It is used to call an event when a battery is OKAY again.
android.intent.action.REBOOT	It call an event when the device rebooted again.

3.7 Content Providers

- In Android, Content Providers are a very important component that serves the purpose of a relational database to store the data of applications.
- For example,
  - Android Contacts application uses a content provider to store and retrieve contacts data.
  - The Contacts application provides a content provider that other applications can use to access contacts data.
  - Similarly, the Android Media Store application uses a content provider to store and retrieve media files.
- There are various examples of content providers in Android. Some of them are as follows:
  - Music Playlist
  - Calls Logs
  - Contact List
  - Message Threads
  - Gallery Application
  - File Manager, etc
- The role of the content provider in the android system is like a central repository in which data of the applications are stored, and it facilitates other applications to securely access and modifies that data based on the user requirements.
- Android system allows the content provider to store the application data in several ways.
- Users can manage to store the application data like images, audio, videos, and personal contact information by storing them in SQLite Database, in files, or even on a network.
- In order to share the data, content providers have certain permissions that are used to grant or restrict the rights to other applications to interfere with the data.



3.7.1 Content URIs

Content URI(Uniform Resource Identifier) is the key concept of Content providers. To access the data from a content provider, URI is used as a query string.

```
<prefix>://<authority>/<data_type>/<id>
```

Sr.No	Part & Description
1	**prefix** This is always set to content://
2	**authority** This specifies the name of the content provider, for example *contacts*, *browser* etc. For third-party content providers, this could be the fully qualified name, such as *com.tutorialspoint.statusprovider*
3	**data_type** This indicates the type of data that this particular provider provides. For example, if you are getting all the contacts from the *Contacts* content provider, then the data path would be *people* and URI would look like this*content://contacts/people*
4	**id** This specifies the specific record requested. For example, if you are looking for contact number 5 in the Contacts content provider then URI would look like this *content://contacts/people/5*.

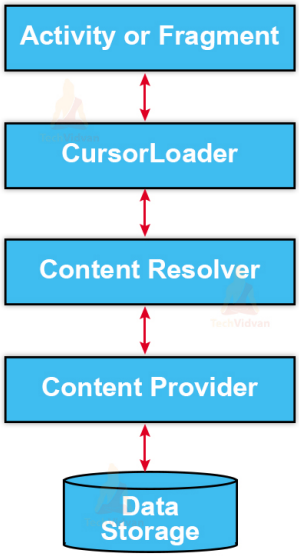
3.7.2 Operations in Content Provider

Four fundamental operations are possible in Content Provider namely Create, Read, Update, and Delete. These operations are often termed as CRUD operations.

- **Create**: Operation to create data in a content provider.
- **Read**: Used to fetch data from a content provider.
- **Update**: To modify existing data.
- **Delete**: To remove existing data from the storage.

3.7.3 Working of the Content Provider

- We know that the Content Provider provides us data, but we need to establish a connection to get this data.
- The link is made using the "**ContentResolver**" object. The ContentResolver objects help the application to communicate with the Content Provider.
- Now, in the next step, we need to connect our ContentResolver object with the user interface.
- By this step, the user can interact directly with the Content Provider by using the user interface. For this purpose, we require another object called **CursorLoader**.
- The CursorLoader task is to receive the user's query request and forward it to the Content Provider.
- The content Provider executes the query and provides the user the requested data.



3.7.4 Creating a Content Provider

To create a content provider in android applications we should follow below steps.

- We need to create a content provider class that extends the ContentProvider base class.
- We need to define our content provider URI to access the content.
- The ContentProvider class defines a six abstract methods (insert(), update(), delete(), query(), getType()) which we need to implement all these methods as a part of our subclass.
- We need to register our content provider in AndroidManifest.xml using <provider> tag.

  Following are the list of methods which need to implement as a part of ContentProvider class.

Abstract Method	Description
query()	A method that accepts arguments and fetches the data from the desired table. Data is retired as a cursor object.
insert()	To insert a new row in the database of the content provider. It returns the content URI of the inserted row.
update()	This method is used to update the fields of an existing row. It returns the number of rows updated.
delete()	This method is used to delete the existing rows. It returns the number of rows deleted.
getType()	This method returns the Multipurpose Internet Mail Extension(MIME) type of data to the given Content URI.
onCreate()	As the content provider is created, the android system calls this method immediately to initialise the provider.

3.8 Sample Questions

Marks	Sample Question
4	Please explain the "Activity life cycle" and its role in creating Android apps.
4	Explain the different types of Android intents and give some instances of each.
6	Could you compare and contrast Android's explicit and implicit intentions to show how they differ and what they have in common?
6	Find out what makes Android apps tick by digging into the inner workings of manifest files.
9	How important is the "Activity life cycle" in making Android apps run smoothly?
9	Examine how Android's explicit and implicit intentions serve to improve interprocess communication.
4	Explain how intent filters work in Android and how they facilitate interaction between different parts of the system.
12	What methods do you recommend for optimising the speed of apps that use a lot of memory on Android?
5	Explain why "application resources" are so crucial, and how they should be managed, while creating an Android app.
5	Explain how "content providers" facilitate sharing of information between Android apps.
10	Discover what "dependency injection" is and why it's useful in Android programming.
10	Describe how the "RecyclerView" helps Android apps efficiently show lists of data.
15	Analyse the challenges and solutions that come with localising Android apps.
15	Explain why "asynchronous programming" is important in Android development.
6	Outline what "view recycling" means in the context of Android's RecyclerView.
6	Explain why it's so important for Android apps to have "permissions" to restrict who can access what when.
12	Evaluate the difficulties of managing the execution of background tasks in Android apps and suggest ways to overcome them.
12	Analyse how Android's "Fragments" facilitate the development of flexible and extensible UIs.
18	Create a responsive interface for a weather app that works with different screen sizes and orientations.
18	Create an all-encompassing strategy to boost the functionality of an image-intensive Android app.
18	Is there a detailed tutorial on how to make Android recognise swipes as a means of switching between apps?

3.9 New Sample Questions

Marks	Sample Question
2	Can you provide a brief explanation of the role and significance of services in Android applications?
2	Could you highlight the differences and similarities between implicit and explicit intents in Android?
2	Please describe how Broadcast Receivers function within the Android system.
3	Assess the importance of Content Providers in the development of Android applications.
3	Can you elaborate on the procedure for initiating a new activity in Android?
3	Analyze the significance of exchanging data between different Android components.
4	Could you explain the lifecycle method that gets invoked when a fragment is no longer visible to the user? Additionally, provide insights into its purpose and importance.
4	Elaborate on the process of creating a customized view by extending the View class in Android.
8	Discuss the utilization of fragments for creating versatile and adaptable UI layouts in Android.
8	Analyze the methodology for handling touch events in Android and elucidate the implementation of touch event management in a custom view.
12	Evaluate the value of incorporating accessibility features in Android app design. How can developers ensure their applications are accessible to a diverse user base?
12	Analyze the factors and techniques for enhancing the performance of Android applications.
12	Please provide an explanation of data binding in Android and how it streamlines the development process.
5	Enumerate and elucidate the advantages of integrating Fragments into Android app development.
10	Assess the pros and cons of employing Fragments instead of traditional activities for UI design in Android apps.
10	Can you expound on the concept of implicit intents in Android and offer an example of scenarios when and why they might be employed?
10	Compare and contrast the roles of an AsyncTask and a Handler in managing background tasks in Android.
15	Discuss the considerations and challenges associated with designing Android applications for various screen sizes and resolutions.
15	Analyze the impact of integrating third-party libraries and frameworks into the Android app development process.
6	Describe the purpose and application of the ViewHolder pattern in RecyclerViews within the Android framework.
6	Explain the concept of data binding in Android and illustrate how it simplifies UI updates and event handling.
8	Detail the role of XML files in defining Android UI layouts and expound on how XML layouts bolster responsive UI design.
12	Compare the Kotlin and Java programming languages for Android app development, highlighting their differences and similarities.
12	Compare the methods used to handle configuration changes in Android, including onSaveInstanceState() and ViewModel.

3.10 Related Videos

- **Components of Android App**
  https://www.youtube.com/watch?v=W2Xn42Id2V4
- **Android Activity and Activity Lifecycle**
  https://www.youtube.com/watch?v=kp-oGImpO_c
  https://www.youtube.com/watch?v=2DYv7aiTTOA
- **Fragment** https://www.youtube.com/watch?v=7jcXWOXwd4Y
  https://www.youtube.com/watch?v=hD9Z4OZt_Wk
- **Android Service**
  https://www.youtube.com/watch?v=hMMeFhEenM8
  https://www.youtube.com/watch?v=oRmgM9xQLDE
  https://www.youtube.com/watch?v=tNO7izOZD5k
- **Intent** https://www.youtube.com/watch?v=JlQ5uaOWLaA
  - **Explicit Intent** https://www.youtube.com/watch?v=VeFPqmkXAGE
  - **Implicit Intent** https://www.youtube.com/watch?v=r-JNSsrIBds
- **Broadcast Receiver** https://www.youtube.com/watch?v=vIwj6BnH7yk
- **Content Provider** https://www.youtube.com/watch?v=UuUikmsBr_M
  https://www.youtube.com/watch?v=1-CAcdayL2I
  https://www.youtube.com/watch?v=B8koZxv9-0U
  https://www.youtube.com/watch?v=T4zq9nHBdlc
  https://www.youtube.com/watch?v=afB9cZZFKoc