Unit-4

Content Providers:- Creating Content Providers, Accessing Content Providers, using Native Android Content Providers.

Working in the Background:- Creating and Controlling Services, Binding Services to Activities.

Expanding the User Experience:- Introducing the Action Bar ,Creating and Using Menus and Action Bar Action Items.

Content Providers

- Creating Content Providers,
- Accessing Content Providers,
- Using Native Android Content Providers.

Content Providers in Android

- Content providers provide data such as contacts information, call logs, browser history, settings etc…
- In Android, access to a database is restricted to the application that created it, so Content Providers offer a standard interface your applications can use to share data with, and consume data from, other applications—including many of the native databases.
- **Shared Preferences, files and databases** wouldn't support sharing of data between packages. They are accessible only to applications that created them. Content Providers are a way of sharing data between packages.
- The Android framework includes content providers that manage data such as audio, video, images, and personal contact information.
- Content Providers can be built in or user defined one.
- **Built in content providers in Android:**

  Browser — Stores data such as browser bookmarks, browser history, and so on

  CallLog — Stores data such as missed calls, call details, and so on

  Contacts — Stores contact details

  MediaStore — Stores media files such as audio, video and images

  Settings — Stores the device's settings and preferences

- Applications registers themselves as a provider of data. Other applications can request android to read/write data through a fixed API.
- Content Providers are used extensively by the Android framework, giving you the opportunity to enhance your own applications with native Content Providers, including contacts, calendar, and the Media Store.
- Content Providers offer rich search options, including how to provide real-time search suggestions using the Search View.

facilitate data sharing across applications

A content provider can be used to manage access to a variety of data storage sources, including both structured data, such as a SQLite relational database, or unstructured data such as image files, audio and video files

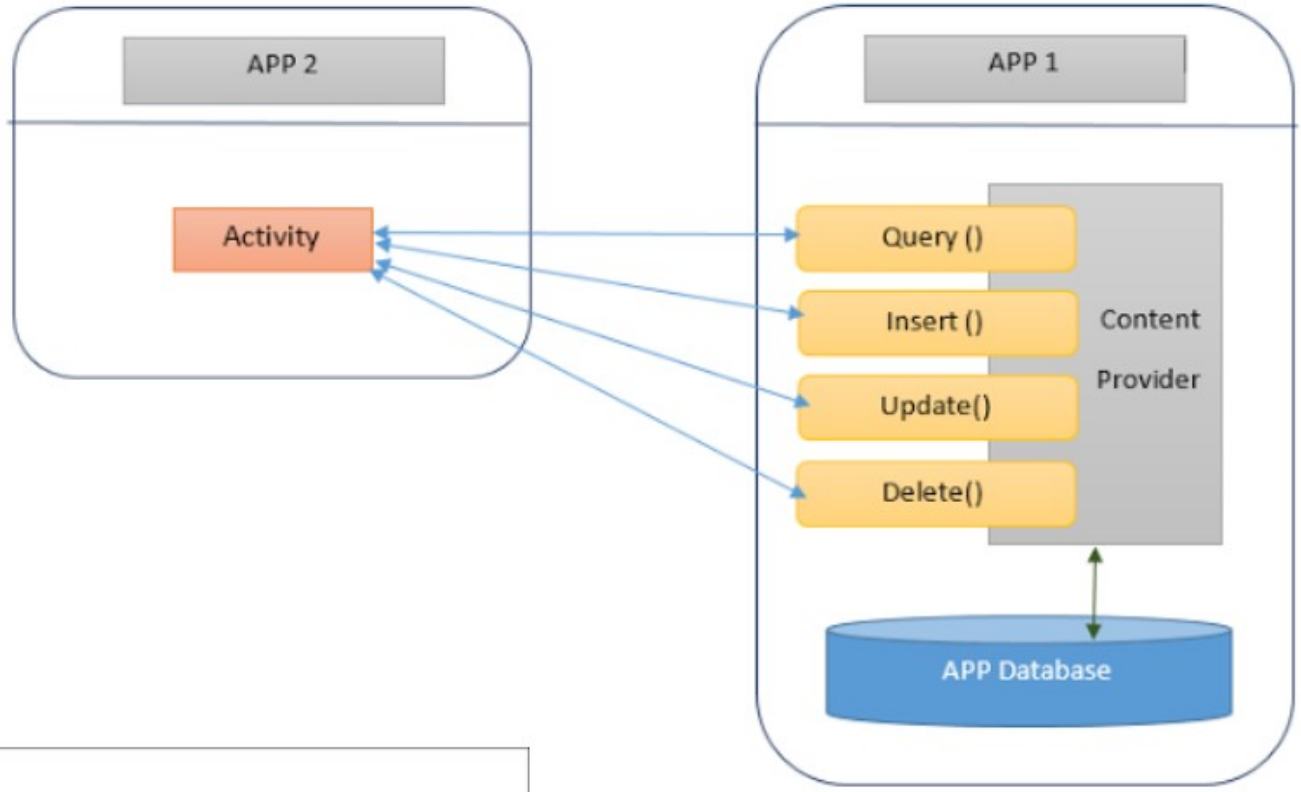
Content Providers in Android

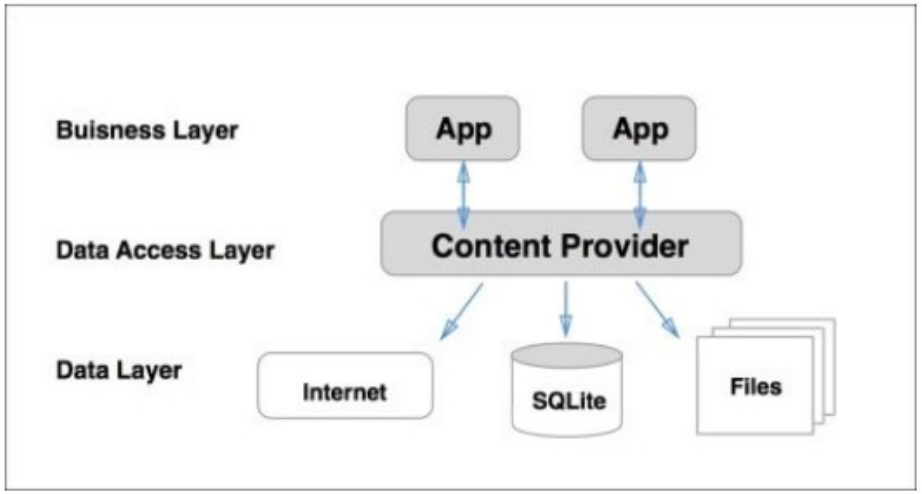
- Any application—with the appropriate permissions—can potentially query, add, remove, or update data provided by another application through a Content Provider, including the native Android Content Providers.
- By publishing your own Content Providers, you make it possible for you (and potentially other developers) to incorporate and **extend your data in other applications.**
- Content Providers are also the mechanism used to provide **search results for the Search View**, and to generate real-time **search suggestions**.
- Content Providers are also useful as a way to **decouple your application layers from the underlying data layers**, by encapsulating and abstracting your underlying database. This approach is useful in making your applications datasource agnostic, such that your data storage mechanism can be modified or replaced without affecting your application layer.

Example Content Providers:

- Music Playlist
- Calls Logs
- Contact List
- Message Threads
- Gallery Application
- File Manager, etc

Content Provider





Content Provider Operations

Content providers have four basic operations: read, create, update, and delete. These tasks are often referred to as CRUD operations.

Create: Used for creating data in the content provider.

Read: Requesting data from the content provider.

Update: Modify existing data.

Delete: Deletes the existing data from the store.

Creating Content Providers in Android

1. Registering Content Providers:

- Like Activities and Services, Content Providers are application components that must be registered in your application manifest before the Content Resolver can discover and use them. You do this using a **provider tag** that includes a name attribute, describing the Provider's class name, and an authorities tag.
- Use the **authorities tag** to define the base URI of the Provider; a Content Provider's authority is used by the Content Resolver as an address to find the database to interact with.
- Each Content Provider authority must be unique, so it's good practice to base the URI path on your package name. The general form for defining a Content Provider's authority is as follows:

com.<CompanyName>.provider.<ApplicationName>

Example: in AndroidManifest.xml file, add

```
<provider android:name=".MyHoardContentProvider"
android:authorities="com.professionalandroid.provider.hoarder"/>
```

Creating Content Providers in Android

2. Publishing Your Content Provider's URI Address

- Each Content Provider should expose its authority using a public static CONTENT_URI property that includes a data path to the primary content.

  public static final Uri CONTENT_URI = Uri.parse("content://com.
  professionalandroid.provider.hoarder/lairs

- These content URIs will be used when accessing your Content Provider using a **Content Resolver**
- A query made using this form represents **a request for all rows**, whereas an appended trailing /<rownumber>, as shown in the following snippet, represents **a request for a specific single record:**

  content://com.professionalandroid.provider.hoarder/lairs/5

- add a **UriMatcher** to your Content Provider implementation to parse URIs, determine their forms, and extract the provided details.

Content URI/ Query string

- **Content URI/ Query string** of content provider:

  <standard_prefix>://<authority>/<data_path>/<id>

Ex: content://media/internal/images

content://media/external/images

content://call_log/calls

content://browser/bookmarks

Standard_prefix : content://

Authority: specify name of the content provider

Data_path: specifies the kind of data requested.

Id: specifies the record requested.

Using Content Providers

- Add the following tag in AndroidManifest.xml file

```
<uses-permission
  android:name=”android.permission.READ_CONTACTS”>
```

- retrieve Managed Cursor by calling

```
Cursor c = managedQuery(allContacts, null, null, null, null);
```

- Create a Cursor adapter

```
SimpleCursorAdapter adapter =
        new SimpleCursorAdapter(this, R.layout.main, c, columns, views);
```

- Add created **SimpleCursorAdapter** to the **ListView** Control

```
this.setListAdapter(adapter);
```

Creating Content Providers

Steps:

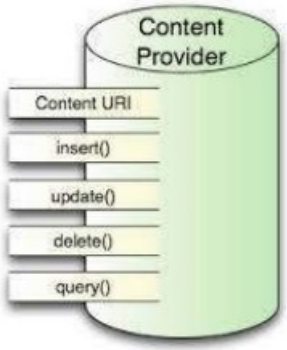
1) First of all you need to **create a ContentProvider class** that extends the *ContentProvider baseclass*.

2) Second, you need to **define your content provider URI address** which will be used to access the content.

3) Next you will need to **create your own database to keep the content**. Usually, Android uses SQLite database and framework needs to override *onCreate()* method which will use SQLite Open Helper method to create or open the provider's database. When your application is launched, the *onCreate()* handler of each of its Content Providers is called on the main application thread.

4) Next you will have to **implement Content Provider queries** to perform different database specific operations.

5) Finally **register your Content Provider** in your activity file using <provider> tag.



Content Providers

- **Content Provider class methods:**
- **onCreate()** This method is called when the provider is started.
- **query()** This method receives a request from a client. The result is returned as a Cursor object.
- **insert()**This method inserts a new record into the content provider.
- **delete()** This method deletes an existing record from the content provider.
- **update()** This method updates an existing record from the content provider.
- **getType()** This method returns the MIME type of the data at the given URI.

Working in the Background – Services in Android

Services

- Your application runs on a Java thread called "main" whose responsibility it is to draw the UI and respond to user actions.
- The main thread handles user action -- keep it fast!
- If the user has to wait for long-running tasks they will get frustrated, might even abandon your app
- If the UI waits too long for long running operations to finish, it becomes unresponsive.
- The framework shows the dreaded ANR dialog (Application not responding).
- Examples of long-running tasks:
  - Network operations
  - Long calculations
  - Downloading/uploading files
  - Processing images
  - Playing music

- To avoid UI meltdown, execute long running tasks on a **background thread**.

The Android SDK includes several ways to accomplish this:

- AsyncTask
- The Loader Framework
- **Services**

Services in Android

- A Service is an application in Android that runs in the background without needing to interact with the user.
- Ex: playing background music, logging continuously geographical coordinates.
- While short amounts of background work are best accomplished with a Broadcast Receiver, and the Job Service (or Work Manager) is preferred for batching background work, there are cases where your app will need to continue running well beyond the lifecycle of a particular Activity. These types of longer running operations that happen in the background are the focus for Services.
- Most Services have their life cycle tied to other components; these are called bound Services.

Difference between android services and threads:

- ✓ Thread is a feature provided by the Operating system to allow the user to perform operations in the background.
- ✓ While service is an android component that performs a long-running operation about which the user might not be aware of as it does not have UI.

Services in Android

Steps in Creating a Service:

1. Create an Android App with blank activity
2. Create a class that extends 'Service'

```
Public class MyService extends Service{
        public Ibinder onBind(Intent arg0) {
            ----
         return null;
     }
     public int onStartCommand(Intent intent, int flags, int startId) {
            ----
     return START_STICKY;
}
public void onDestroy() {
    ----
}
```

Services in Android

- A user-defined service can be created through a normal class which is extending the **class Service**. Further, to carry out the operations of service on applications, there are certain callback methods which are needed to be **overridden**. The following are some of the important methods of Android Services:

Method	Description
onStartCommand()	The Android service calls this method when a component(eg: activity) requests to start a service using startService(). Once the service is started, it can be stopped explicitly using **stopService**() or **stopSelf**() methods.
onBind()	This method is mandatory to implement in android service and is invoked whenever an application component calls the **bindService**() method in order to bind itself with a service.
onDestroy()	When a service is no longer in use, the system invokes this method just before the service destroys as a final clean up call. Services must implement this method in order to clean up resources like registered listeners, threads, receivers, etc.
onCreate()	Whenever a service is created either using onStartCommand() or onBind(), the android system calls this method. This method is necessary to perform a one-time-set-up.
onUnbind()	The Android system invokes this method when all the clients get disconnected from a particular service interface.
onRebind()	When a service is no longer in use, the system invokes this method just before the service destroys as a final clean up call.

Types of Services

- Foreground Services
- Background Services
- Bound Services

1. Foreground Services:

- 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. Such as in downloading a file, the user can keep track of the progress in downloading and can also pause and resume the process.

2. Background Services:

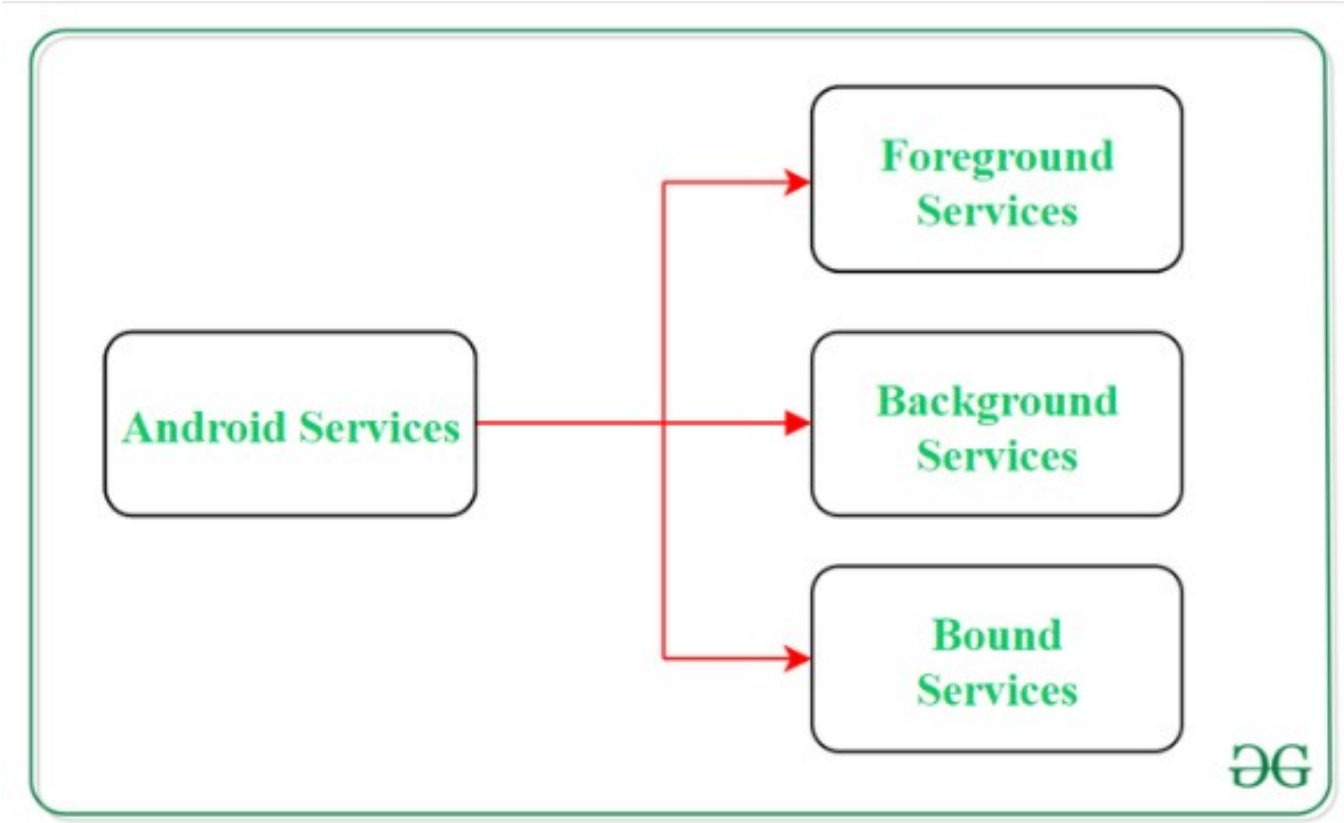
✓ Background services do not require any user intervention. These services do not notify the user about ongoing background tasks and users also cannot access them. The process like schedule **syncing of data or storing of data** fall under this service.

Services in Android

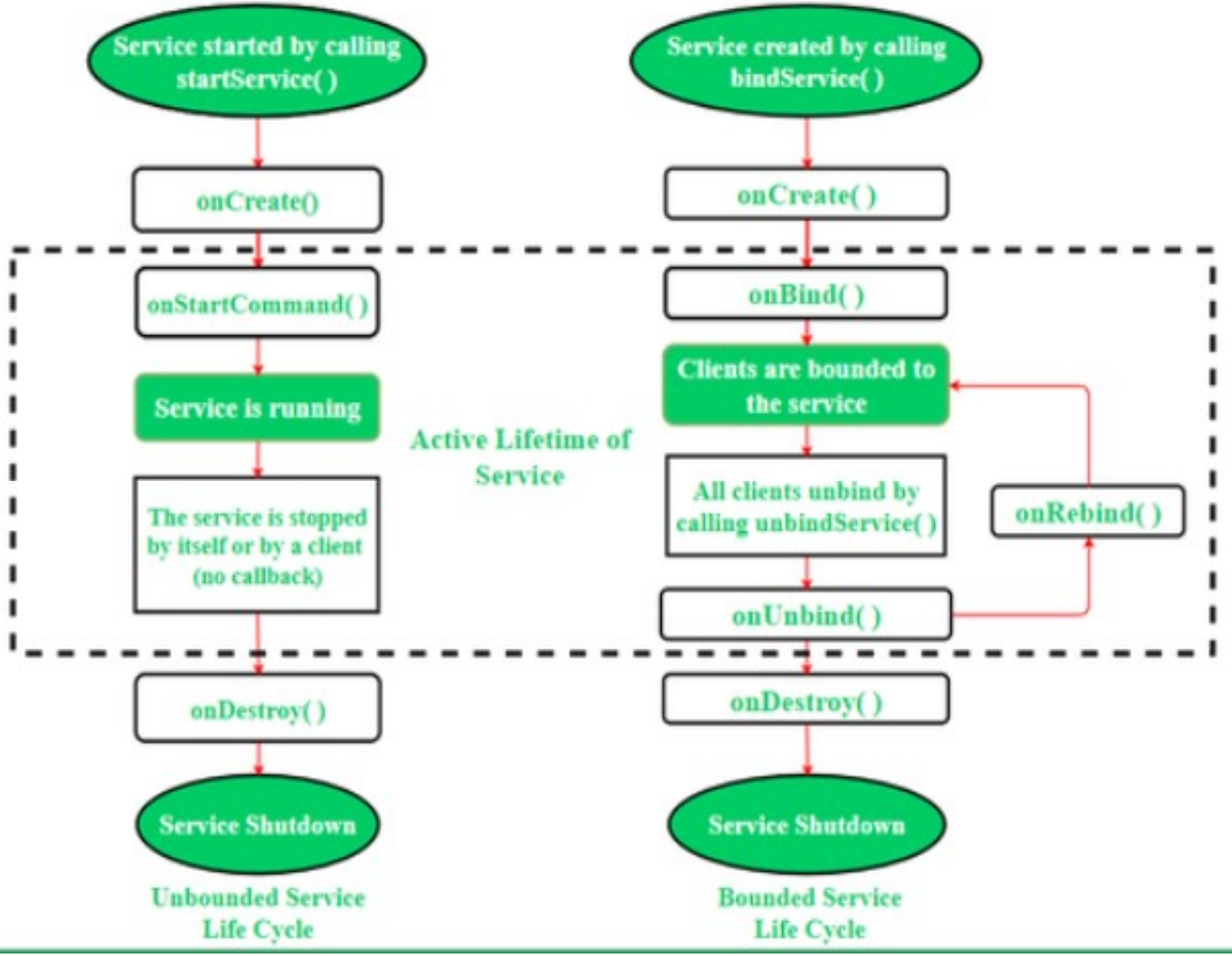
3. Bound Services:

- This type of android 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.

Services in Android



Services in Android



ActionBar

ActionBar

- Android ActionBar is a menu bar that runs across the top of the activity screen in android. Android ActionBar can contain menu items which become visible when the user clicks the “menu” button.
- In general an ActionBar consists of the following four components:
  - **App Icon**: App branding logo or icon will be displayed here
  - **View Control**: A dedicated space to display Application title. Also provides option to switch between views by adding spinner or tabbed navigation
  - **Action Buttons**: Some important actions of the app can be added here
  - **Action Overflow**: All unimportant action will be shown as a menu

- It provides a visual structure to the app and contains some of the frequently used elements for the users.
- Android ActionBar was launched by **Google in 2013** with the release of **Android 3.0(API 11)**.
- Beginning with Android 3.0 (API level 11), the action bar appears at the top of an activity's window when the activity uses the system's Holo theme (or one of its descendant themes), which is the default.
- Google announced a **support library** along with the introduction of ActionBar. This library is a part of **AppCompat**
- All applications that use the default theme provided by the Android (contains an ActionBar by default.
- EvTheme.AppCompat.Light.DarkActionBar), every android app contains an ActionBar by default. This pre-included ActionBar display title for the current activity that is managed by the **AncdroidManifest.xml** file.
- Beginning with Android L (API level 21), the action bar may be represented by any Toolbar widget within the application layout.

ActionBar

Advantages of ActionBar

- Provides a customized area to design the identity of an app
- Specify the location of the user in the app by displaying the title of the current Activity.
- Provides access to important and frequently used actions
- Support tabs and a drop-down list for view switching and navigation.

Disadvantages of ActionBar

- All features of the ActionaBar are not introduced at once but were introduced with the release of different API levels such as API 15, 17, and 19.
- The ActionBar behaves differently when it runs on different API levels.
- The features that were introduced with a particular API does not provide backward compatibility.

ActionBar

- For every MenuItem of ActionBares, the following attributes are needed to be configured:
- **android:title:** Its value contains the title of the menu item that will be displayed when a user clicks and holds that item in the app.
- **android:id:** A unique ID for the menu item that will be used to access it anywhere in the whole application files.
- **android:orderInCategory:** The value of this attribute specify the item's position in the ActionBar. There are two ways to define the position of different menu items. The first one is to provide the same value of this attribute for all items and the position will be defined in the same order as they are declared in the code.
- **app:showAsAction:** This attribute defines how the item is going to be present in the action bar. There are four possible flags to choose from:
  - **a. always:** To display the item in the ActionBar all the time.
  - **b. ifRoom:** To keep the item if space is available.
  - **c. never:** With this flag, the item will be not be displayed as an icon in ActionBar, but will be present in the overflow menu.
  - **d. withText:** To represent an item as both icon and the title, one can append this flag with the always or ifRoom flag(always | withText or ifRoom | withText).

android:icon: The icon of an item is referenced in the drawable directories through this attribute.

ActionBar

- **ActionBar methods:**
- **setTitle()** → Set the action bar's title
- **setSubTitle()** → Set the action bar's subtitle.
- **setIcon( )** → Set the icon to display in the 'home' section of the action bar.
- **setDisplayUseLogoEnabled(boolean useLogo)** → Set whether to display the activity logo rather than the activity icon. A logo is often a wider, more detailed image.
- **setDisplayShowHomeEnabled(boolean showHome)** → Set whether to include the application home affordance in the action bar. Home is presented as either an activity icon or logo.

ActionBar

ActionBar

ActionBar

ActionBar

ActionBar

ActionBar

ActionBar

ActionBar