ANDROID COMPONENTS & RESOURCE HANDLING

Prof. Nilesh B. Ghavate
Assistant Professor
CSE IEP Department

Lesson Plan

Subject/Course	Mobile App Development
Lesson Title	Android Components & Resource Handling

Lesson Objectives
Introduction, Android Architecture
Android Development Tools
Directory Structure of Android Application
Android Manifest file

Outline

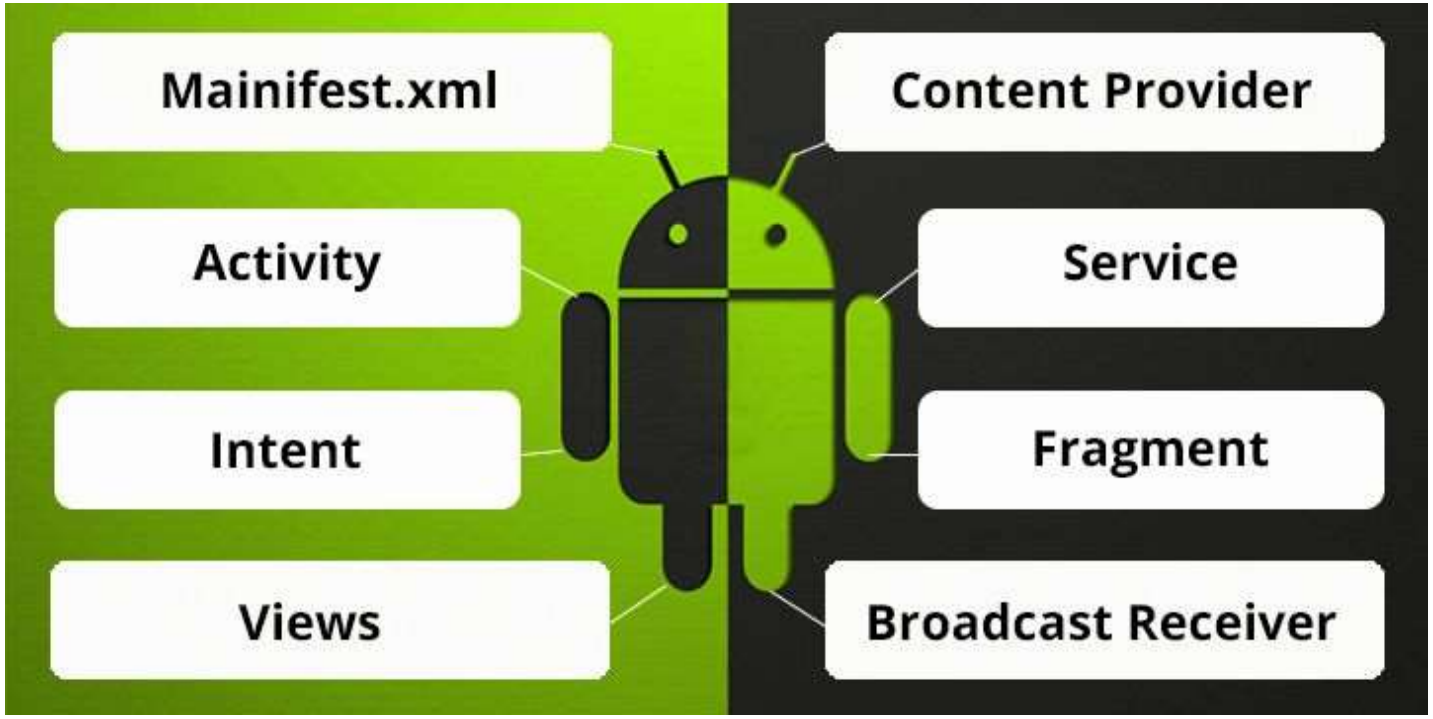
- Context
- Activity
- Intent
- Service
- Broadcast Receiver
- String
- Color
- Drawable
- Theme
- Prepare Application for Localization

Android Components

- An android component is simply a piece of code that has a well defined life cycle e.g. Activity, Receiver, Service etc.
- The core building blocks or fundamental components of android are activities, views, intents, services, content providers, fragments and **AndroidManifest. xml.**

Android Components

- Activities
- Services
- Content Providers
- Broadcast Receivers
- Intents
- Widgets
- Notification



Context

- The Contex class(android.content.context) is a fundamental building block of any android application and provides access to application-wide features such as the application's private files and device resources, as well as system-wide services.
- The application wide Context object is instantiated as an Application object(android.app.Application).

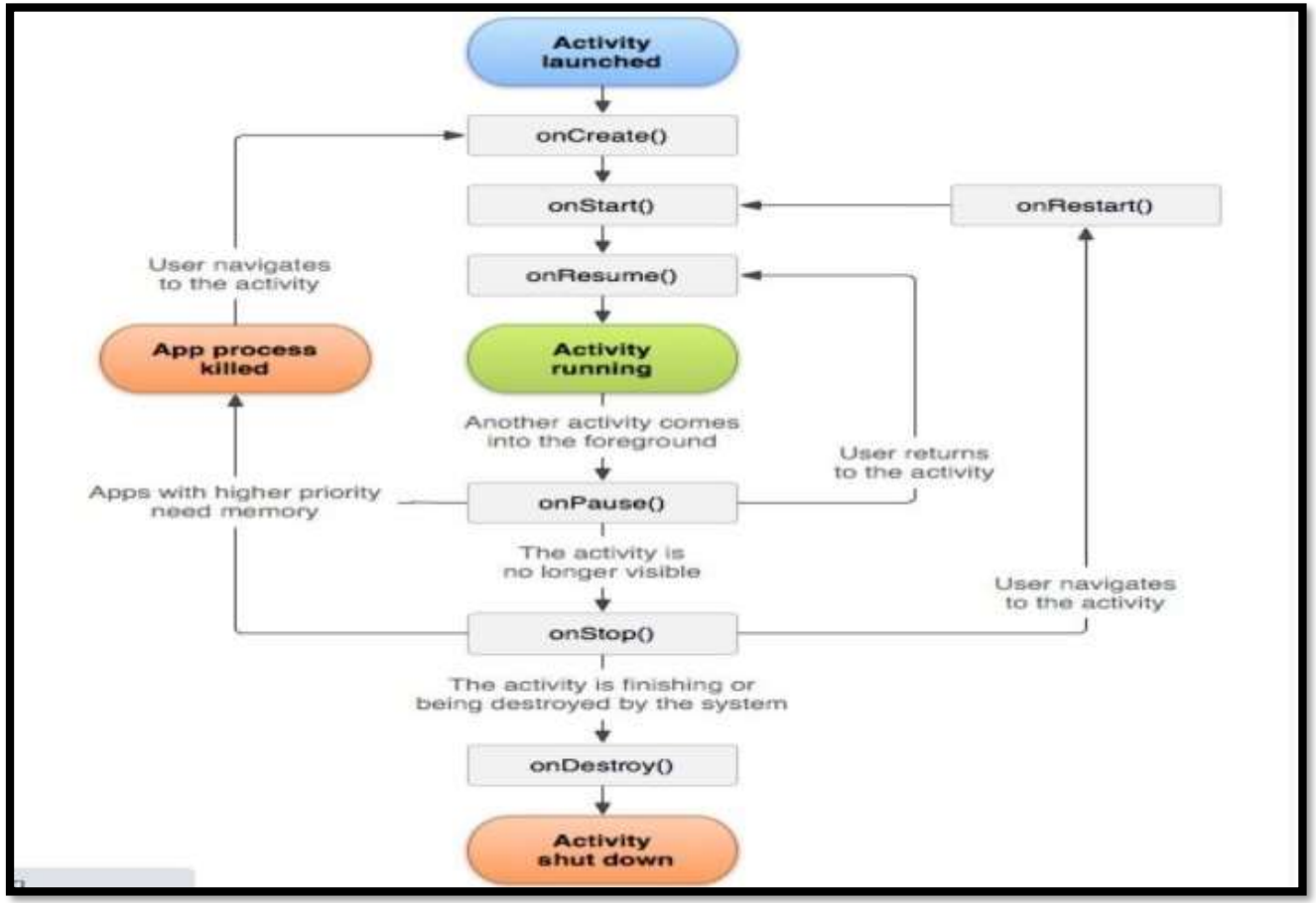
Activity

- Activity class is the core of any Android Application.
- You define and implement Activity class for each screen in your application.
- For example a simple game application might have the following five activities:
  - A starting screen: displays the application name and version
  - A main menu screen: users choose what they want to do within the application.
  - A game play screen
  - A high score screen
  - A help/about screen

Activity

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 is not visible to the user.
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.

Activity


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

Running State

- An activity is in the running state if it's shown in the foreground of the users' screen
- When an Activity is in active state, it means it is active and running.
- It is visible to the user and the user is able to interact with it.
- Android Runtime treats the Activity in this state with the highest priority and never tries to kill it.

Activity

Paused State

- An activity being in this state means that the user can still see the Activity in the background such as behind a transparent window or a dialog box but it is partially visible.
- The user cannot interact with the Activity until he/she is done with the current view.
- Android Runtime usually does not kill an Activity in this state but may do so in an extreme case of resource crunch.

Resumed State

- It is when an activity goes from the paused state to the foreground that is an **active** state

Activity

Stopped State:

- When a new Activity is started on top of the current one or when a user hits the Home key, the activity is brought to Stopped state.
- The activity in this state is invisible, but it is not destroyed.
- Android Runtime may kill such an Activity in case of resource crunch.

Intent

- An **Intent** is a messaging object that allows components (such as activities, services, and broadcast receivers) to communicate with each other.
- Intents are primarily used to **start new activities**, **pass data**, **invoke system services**, or **broadcast messages** across the system.
- 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.

Intent

Types of Intent:

- **Explicit Intent:** Used when you explicitly specify the target component (e.g., starting a specific activity or service).
- Example: Navigating from one activity to another within the same application.
- **Implicit Intent:** Used when you don't specify the target component directly. Instead, you declare an action, and the system determines the appropriate component (e.g., opening a web page or sharing content).
- Example: Opening a web page in the browser.

Intent

- To implement activities in our application, we need to register them in the Manifest file. For the activities to work properly, we must manage their lifecycle properly.
  - Declare Activities
  - Declare Intent Filters
  - Declare Permission

```
<activity android:name="MainActivity">
    <!-- This activity is the main entry, should appear in app launcher -->
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Service

- A service is a component that runs in the background, it acts as an invisible worker of our application. It keeps updating data sources and activities. It also broadcasts intents and performs tasks when applications are not active. An example of service is we can surf the internet or use any other application while listening to music.
- To execute services, extend the Services class in your sub-class:

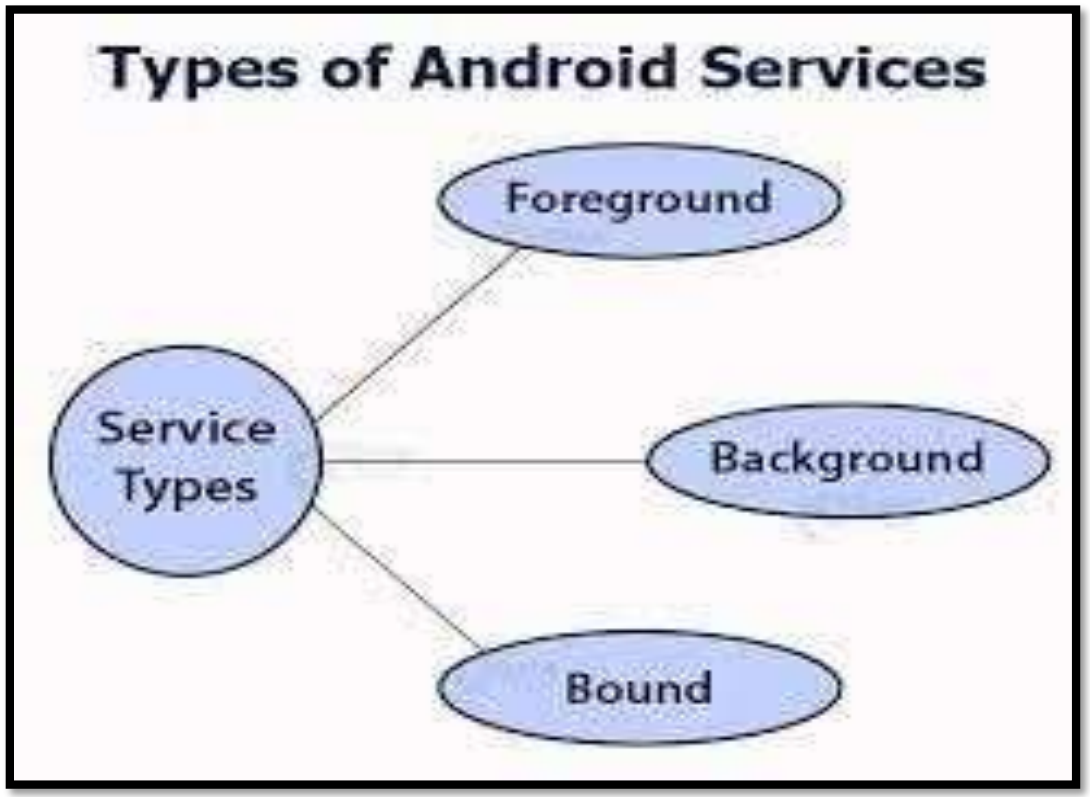
```
public class MyService extends Services {
//code
}
```

Service

- Registering Services in Android Manifest

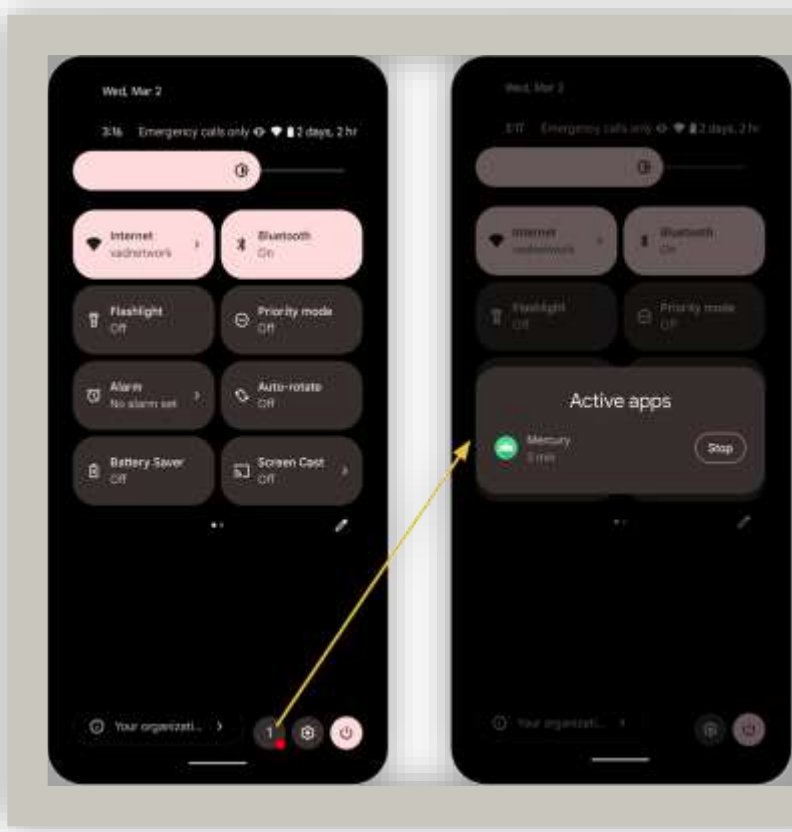
```
<service android:description="string resource"
         android:directBootAware=["true" | "false"]
         android:enabled=["true" | "false"]
         android:exported=["true" | "false"]
         android:foregroundServiceType=["camera" | "connectedDevice" |
                                        "dataSync" | "location" | "mediaPlayback" |
                                        "mediaProjection" | "microphone" | "phoneCall"]
         android:icon="drawable resource"
         android:isolatedProcess=["true" | "false"]
         android:label="string resource"
         android:name="string"
         android:permission="string"
         android:process="string" >
    . . .
</service>
```

Service


Types of Android Services
Foreground
Service Types
Background
Bound

Service – Foreground services

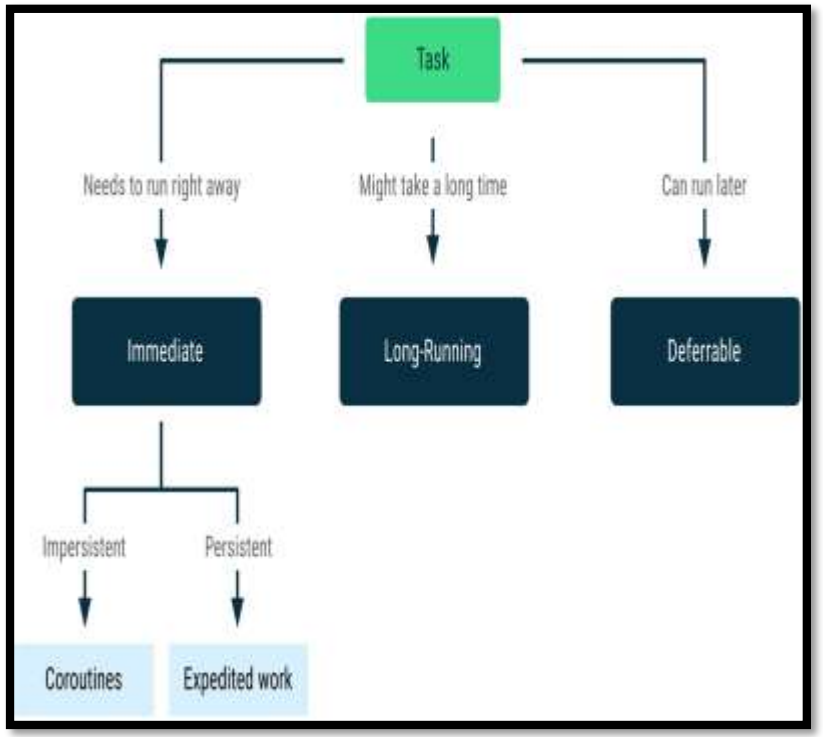
- Foreground services are those services that are visible to the users.
- The users can interact with them at ease and track what's happening.
- These services continue to run even when users are using other applications
- *The perfect example of this is Music Player and Downloading.*



Service – background services

- These services run in the background, such that the user can't see or access them.
- These are the tasks that don't need the user to know them.
- 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.

Service – background services



- Background work falls into one of three primary categories:
  - **Immediate:**
    - Needs to execute right away and complete soon.
  - **Long Running:**
    - May take some time to complete.
  - **Deferrable:**
    - Does not need to run right away

- Likewise, background work in each of these three categories can be either persistent or impersistent:
  - **Persistent work**:
    - Remains scheduled through app restarts and device reboots.
  - **Impersistent work:**
    - No longer scheduled after the process ends.

Service – background services

Immediate Working

- Immediate work includes tasks that must be completed right away. These are tasks that are important to the user or that you cannot otherwise schedule for later execution. They are significant enough that they may need to be scheduled for immediate execution even if the app is closed or the device restarted.
- A data source must be used to load data into an app. Making such a request on the main thread, on the other hand, will block it and cause UI jank. Instead, the app executes the request in a coroutine outside of the main thread.

Service – background services

Long Running Work

- A long running task is a task that requires more than 30 seconds to complete and involves a large amount of data.
- Work is long running if it is likely to take more than ten minutes to complete.
- It allows you to handle such tasks using a long Eg:
- An apps needs to download a large file which you cannot junk it creates a long running work and computing the download.

Service - Bound Services

- Bound service runs as long as some other *application component* is bound to it. Many components can bind to one service at a time, but once they all unbind, the service will destroy.

Service

- Android services life cycle can have two forms of services and they follow two paths, that are

1. **Started Service**

- A service becomes started only when an application component calls startService(). It performs a single operation and doesn't return any result to the caller. Once this service starts, it runs in the background even if the component that created it destroys. This service can be stopped only in one of the two cases
  - By using the stopService() method.
  - By stopping itself using the stopSelf() method.

Service

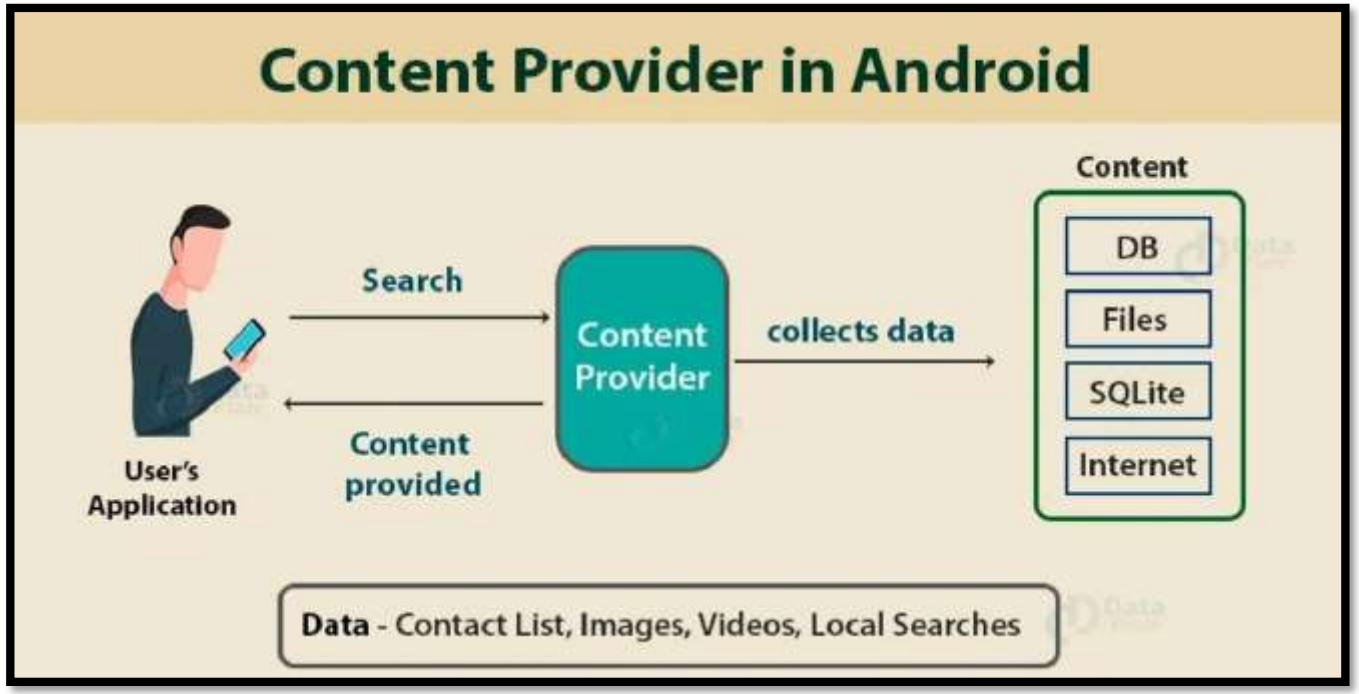
2. Bounded Service

- A service is bound only if an application component binds to it using bindService(). It gives a client-server relation that lets the components interact with the service. The components can send requests to services and get results.
- This service runs in the background as long as another application is bound to it. Or it can be unbound according to our requirement by using the unbindService() method.

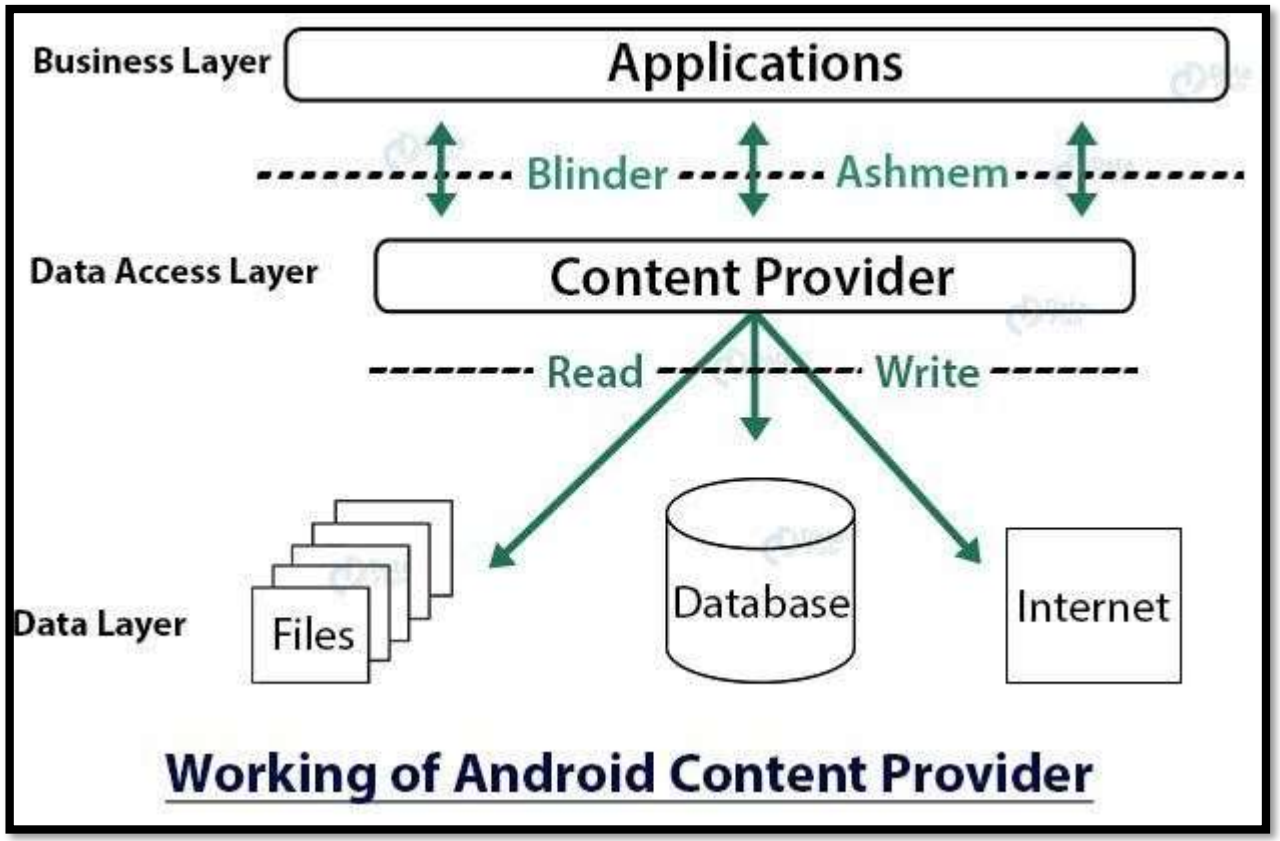
Content Provider

- Content Providers are an important component of Android.
- They handle the access to the central repository and supply data from one application to another on request.
- This task of handling is done by methods of ContentResolver class. So, content providers can store data in various ways such as files, database or over the internet.
- It acts as the same as database and also we can query it to add, delete, insert or update the data.
- It can be understood that a content provider hides the database details and also, it lets an application share data among other applications. Content providers are not limited to texts, but also contains images and videos as well.

Content Provider



Content Provider



Content Provider

- We need to use the ContentResolver object in our application, in order to communicate with the content providers for data access.
- Now to enable communication between the user interface and ContentResolver, we use another object, CursorLoader to run query asynchronously. This CursorLoader will be called using Activity/Fragment of the application.
- Then, the content provider receives the query from the client and executes and returns the result.

Content Provider

- **onCreate() –** This method in Android initializes the provider as soon as the receiver is created.
- **query() –** It receives a request in the form of a query from the user and responds with a cursor in Android.
- **insert() –** This method is used to insert the data into our content provider.
- **update() –** This method is used to update existing data in a row and return the updated row data.
- **delete() –** This method deletes existing data from the content provider.
- **getType()–** It returns the Multipurpose Internet Mail Extension type of data to the given Content URI.

Broadcast Receiver

- Broadcast Receiver is a component that responds to broadcast messages from another application or the same system.
- It can also deliver broadcasts to applications that are not running. For example – notify the user that the battery is low.
- Android developers can use broadcast messages in the application or outside the normal flow.
- To implement this, extend Broadcast Receiver to your receiver:

```
public class Broadcast_Name extendsBroadcastReceiver {
//code
}
```

Broadcast Receiver

- Registering Broadcast Receiver in Android Manifest

```
<receiver android:directBootAware=["true" | "false"]
          android:enabled=["true" | "false"]
          android:exported=["true" | "false"]
          android:icon="drawable resource"
          android:label="string resource"
          android:name="string"
          android:permission="string"
          android:process="string" >
    . . .
</receiver>
```

Resource (res folder structure)

```
MyProject/
   app/
      manifest/
         AndroidManifest.xml
   java/
      MyActivity.java
      res/
         drawable/
            icon.png
         layout/
            activity_main.xml
            info.xml
         values/
            strings.xml
```

String

- One of the most important as well as widely used values file is the strings.xml due to its applicability in the Android project.
- Basic function of the strings.xml is to define the strings in one file so that it is easy to use same string in different positions in the android project plus it makes the project looks less messy. We can also define an array in this file as well.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string  name="hello">Hello, World!</string>
</resources>
```

Color

- The *colors.xml* is an XML file that is used to store the colors for the resources. An Android project contains 3 essential colors namely:

➢ colorPrimary

➢ colorPrimaryDark

➢ colorAccent

- These colors are used in some predefined resources of the Android studio as well. These colors need to be set opaque otherwise it could result in some exceptions to arise.

Color

```xml
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="purple_200">#FFBB86FC</color>
    <color name="purple_500">#FF6200EE</color>
    <color name="purple_700">#FF3700B3</color>
    <color name="teal_200">#FF03DAC5</color>
    <color name="teal_700">#FF018786</color>
    <color name="black">#FF000000</color>
    <color name="white">#FFFFFFFF</color>
    <color name="pietColor">#FAAE6B</color>
</resources>
```

Drawable

- The **resource folder** is the most important folder because it contains all the non-code sources like images, XML layouts, UI strings for the android application.
- Among them, the drawable folder contains the different types of images used for the development of the application.
- We need to add all the images to the drawable folder for the application development. Images are used in android applications to provide more user-friendly behavior & functionality.

Styles

- Another important file in the values folder is the *styles.xml* where all the themes of the Android project are defined.
- The base theme is given by default having the option to customize or make changes to the customized theme as well.
- Every theme has a parent attribute which defines the base of the theme. There are a lot of options to choose from depending on the need of the Android project.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <style name="CustomFontStyle">
        <item name="android:layout_width">fill_parent</item>
        <item name="android:layout_height">wrap_content</item>
        <item name="android:capitalize">characters</item>
        <item name="android:typeface">monospace</item>
        <item name="android:textSize">12pt</item>
        <item name="android:textColor">#00FF00</item>/>
    </style>
</resources>
```

Styles – Using styles

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/text_id"
        style="@style/CustomFontStyle"
        android:text="@string/hello_world" />

</LinearLayout>
```

Theme

- A theme is nothing but an Android style applied to an entire Activity or application, rather than an individual View.
- Thus, when a style is applied as a theme, every **View** in the Activity or application will apply each style property that it supports.
- For example, you can apply the same **CustomFontStyle** style as a theme for an Activity and then all text inside that **Activity** will have green monospace font.
- To set a theme for all the activities of your application, open the **AndroidManifest.xml** file and edit the **<application>** tag to include the **android:theme** attribute with the style name. For example –

Localization: Preparing an Application for Localization

- Localization is the process of adapting your application to support multiple languages and regions to reach a global audience.
- This involves translating text, formatting dates, times, numbers, currencies, and other region-specific content.
- In Android, localization is achieved by organizing application resources (like strings, layouts, images) into region-specific directories and allowing the system to load the appropriate resources based on the user's language and locale settings.

Steps to Prepare an Android Application for Localization

1. Use Resource Files for Text (Avoid Hardcoding)All user-visible strings should be defined in the res/values/strings.xml file. Avoid hardcoding strings directly in the code or layout files.

- **XML code**

- **<!-- res/values/strings.xml -->**
- **<string name="app_name">My Application</string>**
- **<string name="welcome_message">Welcome to My Application</string>**

Steps to Prepare an Android Application for Localization

- Access these strings in

Java code

- **String welcomeMessage = getString(R.string.welcome_message);**

XML code

- **<TextView**
- **android:text="@string/welcome_message"**
- **android:layout_width="wrap_content"**
- **android:layout_height="wrap_content" />**

Steps to Prepare an Android Application for Localization

2. Create Language-Specific Resource FilesAndroid uses a flexible resource structure to support multiple languages.

For each language, create a separate strings.xml file in a language-specific folder.

Default language (English): ***res/values/strings.xml***

<string name="welcome_message">Welcome</string>

Steps to Prepare an Android Application for Localization

French localization: res/values-fr/strings.xml

<string name="welcome_message">Bienvenue</string>

Spanish localization: res/values-es/strings.xml

<string name="welcome_message">Bienvenido</string>

When a user's device is set to a specific language, Android automatically loads the corresponding ***strings.xml*** file.

Steps to Prepare an Android Application for Localization

3. Organize Region-Specific Resources Localization isn't just about translating text. Images, layouts, or other resources might also need adaptation for different regions.

Example of localized resources:

- Default layout: res/layout/activity_main.xml
- Arabic layout (Right-to-Left): res/layout-ar/activity_main.xml
- Default drawable: res/drawable/logo.png
- Chinese drawable: res/drawable-zh/logo.png

Steps to Prepare an Android Application for Localization

4. Localize Date, Time, and Numbers

Android provides utilities to format dates, times, and numbers in a localized manner using the user's locale settings.

- **Example of localized date and time formatting**:

```
Locale currentLocale = Locale.getDefault();
DateFormat dateFormat = DateFormat.getDateInstance(DateFormat.LONG,
currentLocale);
String formattedDate = dateFormat.format(new Date());
```

Steps to Prepare an Android Application for Localization

- **Example of localized currency formatting**:

```
NumberFormat currencyFormat = NumberFormat.getCurrencyInstance(currentLocale);
String formattedCurrency = currencyFormat.format(1234.56);
```

Steps to Prepare an Android Application for Localization

5. **Test Localization** Use Android's built-in Emulator or a real device to test localization by switching the language in device settings:

- Settings > System > Languages & Input > Languages.
- Simulate Locale Changes:
- Use Android Studio's Layout Editor to preview layouts in different locales.
- Go to Design View > Tools > Preview Locale.

Benefits of Localization

- **Wider Audience:** Reach users from different countries and cultures.
- **Improved User Experience:** Users feel more comfortable using apps in their native language.
- **Higher Engagement:** Users are more likely to engage and spend time in apps tailored to their region.