Mobile Application Development

Unit-1

Android Introduction

Android is an operating system based on Linux kernel, developed by Google. Linux is an Open Source and free operating system and with some modifications on the Linux operating system, android OS was developed. Linux OS is majorly used in server and Desktop operating system so the android operating system is focused on touch screen mobile devices like smartphones and tablets

Various Platform for Mobile

Android OS: The Android operating system is the most popular operating system today. It is a mobile OS based on the Linux Kernel and open-source software. The android operating system was developed by Google. The first Android device was launched in 2008.

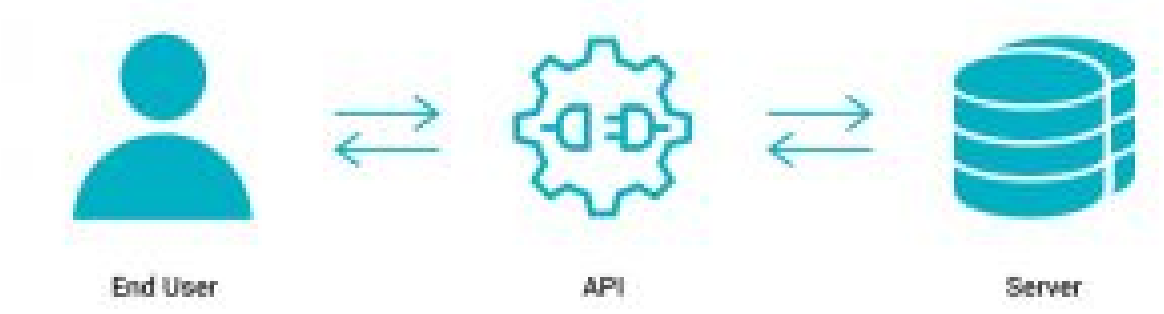
BlackBerry OS: The BlackBerry operating system is a mobile operating system developed by Research In Motion (RIM). This operating system was designed specifically for BlackBerry handheld devices.

iPhone OS / iOS: The iOS was developed by the Apple inc for the use on its device. The iOS operating system is the most popular operating system today. It is a very secure operating system. The iOS operating system is not available for any other mobiles.

Windows Mobile OS: The window mobile OS is a mobile operating system that was developed by Microsoft. It was designed for the pocket PCs and smart mobiles.

Android API

An **Application Programming Interface (API)** is a particular set of **rules ('code') and specifications that programs** can **follow to communicate with each other**. APIs are growing exponentially every year.
The world of software moves fast. In earlier days data entered and processed in the same system, but now the **origin of the data and processing place is entirely different.** We should be able to **access the data from anywhere at any time**, that's why we store this **data in cloud storage**.
For **sending and receiving data from/to the server**,we want a middle man who is platform independent.That **middleman handles the requests and serves the response to the user**.The below diagram illustrates this better than words.



The End user sends a request , **API executes the instruction then gets the data from the server and responds to the user.**

Types of API :

- SOAP API (Simple object Access Protocol)
- REST API (Representational state transfer)

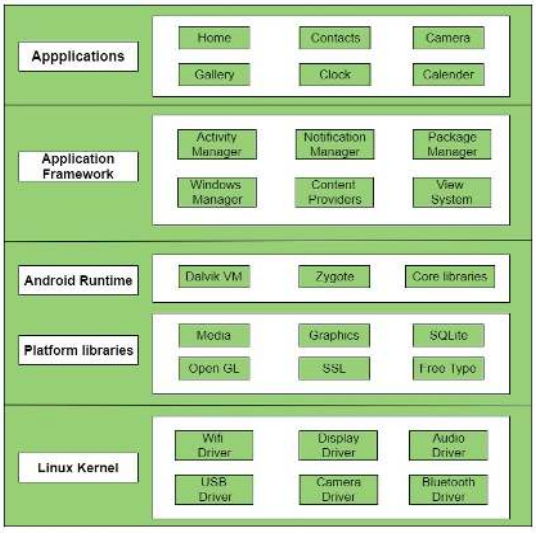
Andriod Architecture

Android architecture **contains a different number of components** to support any android device needs. Android software contains an **open-source Linux Kernel** having a **collection of C/C++ libraries** which are exposed through an **application framework services.**
Among all the **components Linux Kernel provides main functionality of operating system functions** to smartphones and Dalvik Virtual Machine (DVM) provides a platform for running an android application.

The main components of android architecture are following:-

- Applications
- Application Framework
- Android Runtime
- Platform Libraries
- Linux Kernel

Pictorial representation of android architecture with several main components and their sub components –



Applications –

Applications is the **top layer of android architecture**. The **pre-installed applications like home, contacts, camera, gallery** etc and third party applications downloaded from the play store like chat applications, games etc. will be installed on this layer only.
It runs within the Android run time with the help of the classes and services provided by the application framework.

Application framework –

Application Framework provides several important classes which are used to create an Android application. It provides a **generic abstraction for hardware access and also helps in managing the user interface with application resources.** Generally, it provides the services with the help of which we can create a particular class and make that class helpful for the Applications creation.
It includes **different types of services activity manager, notification manager, view system, package manager** etc. which are helpful for the development of our application according to the prerequisite.

Application runtime –

The Android **Runtime environment is one of the most important part of Android**. It contains **components like core libraries and the Dalvik virtual machine(DVM). Mainly**, it **provides the base** for the application framework and powers our application with the **help of the core libraries**.
Like Java Virtual Machine (JVM), Dalvik Virtual Machine (DVM). It depends on the layer Linux kernel for threading and low-level memory management. The core libraries enable us to implement android applications using the standard JAVA or Kotlin programming languages.

Platform libraries –

The Platform Libraries include various **C/C++ core libraries and Java based libraries such as Media, Graphics, Surface Manager, OpenGL** etc. to provide support for android development.

- **Media library** provides support to play and record audio and video formats.
- **Surface manager** responsible for managing access to the display subsystem.
- **SGL and OpenGL** both cross-language, cross-platform application program interface (API) are used for 2D and 3D computer graphics.
- **SQLite** provides database support and FreeType provides font support.
- **Web-Kit** This open source web browser engine provides all the functionality to display web content and to simplify page loading.
- **SSL (Secure Sockets Layer)** is security technology to establish an encrypted link between a web server and a web browser.

Linux Kernel –

Linux Kernel is the heart of the android architecture. It **manages all the available drivers such as display drivers, camera drivers, Bluetooth drivers, audio drivers, memory drivers, etc**. which are required during the runtime.
The Linux Kernel will provide an abstraction l**ayer between the device hardware and the other components** of android architecture. It is responsible for management of memory, power, devices etc.
The features of Linux kernel are:

- **Security:** The Linux kernel handles the security between the application and the system.
- **Memory Management:** It efficiently handles the memory management thereby providing the freedom to develop our apps.
- **Process Management**: It manages the process well, allocates resources to processes whenever they need them.
- **Network Stack**: It effectively handles the network communication.
- **Driver Model:** It ensures that the application works properly on the device and hardware manufacturers responsible for building their drivers into the Linux build.

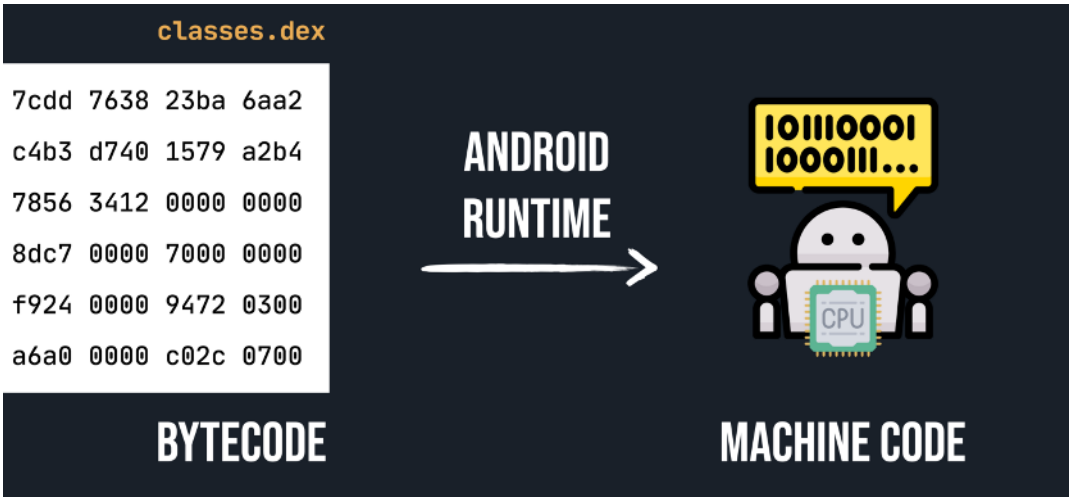
Android Runtime

Android Runtime (ART) is **an application runtime environment used by the Android operating system.**
ART performs the **translation of the application's bytecode into native instructions** that are later executed by the device's runtime environment.

When we build our app and generate APK, part of that APK are **.dex files**. Those **files contain the source code of our app including all libraries that we used in low-level code designed for a software interpreter — the bytecode**.

When a user runs our app the **bytecode written in .dex files is translated by Android Runtime into the machine code** — a set of instructions that can be directly understood by the machine and is processed by the CPU.

Android Runtime also **manages memory and garbage collection** but, to not make this article too long **I'll focus here only on a compilation**.



Dalvik Virtual Machine

The **Dalvik Virtual Machine (DVM)** is an android virtual machine optimized for mobile devices. It optimizes the virtual machine for *memory*, *battery life* and *performance*.

The **Dex compiler converts the class files into the .dex file that runs on the Dalvik VM.** Multiple class files are converted into one dex file.

There are 2 types of files:

- **.dex(Dalvik Executable file) file is an android's compiled code file**.
- These .dex files are then zipped into a single .apk file.
- .odex file is created by the Android operating system to **save space and increase the boot speed** of an Android app (a .apk file).

DALVIK VIRTUAL MACHINE	ANDROID RUNTIME
Faster Booting time	Rebooting is significantly longer
Cache builds up overtime	The cache is built during the first boot

Occupies less space due to JIT	Consumes a lot of storage space internally due to AOT
Works best for small storage devices	Works best for Large storage devices
Stable and tested virtual machine	Experimental and new – not much app support comparatively
Longer app loading time	Extremely Faster and smoother Faster and app loading time and lower processor usage
Uses JIT compiler(JIT: Just-In-Time) Thereby resulting in lower storage space consumption	Uses AOT compiler(Ahead-Of-Time) thereby compiling apps when installed
Application lagging due to garbage collector pauses and JIT	Reduced application lagging and better user experience
App installation time is comparatively lower as the compilation is performed later	App installation time is longer as compilation is done during installation
DVM converts bytecode every time you launch a specific app.	ART converts it just once at the time of app installation. That makes CPU execution easier. Improved battery life due to faster execution.

Features of Android

- **User Interface:** The user interface of the Android operating system is straight forward, and these features make it very user friendly.

- **Multiple Language Support:** Android supports multiple languages in its operating system and one can change the language very easily based on one's requirement, the International languages supported are English, Germany, Chinese, Dutch, French, German, Japanese, Korean, Russian, and many more also some native language of India is also Supported Like Hindi, Marathi, Gujarati, Punjabi and many more.

- **Multi-tasking:** Android provides support to run apps and services in the background with ease which allows the users to use multiple apps at the same time.

- **Connectivity**: Android has extensive support to the connectivity and it supports connectivity such as WiFi, Bluetooth, Hotspot, CDMA, GSM, NFC, VOLTE, UBB, VPN, 3G network band, and 4G Network Band.
- **Extensive Application Support**: Android have Play store which is used as the major tool to download and update applications on the operating system, however, one can download the installer(often called as APK file) and install it manually, but it is not much recommended as third party applications could be prone to some security breach in the smartphones.

ANDROID SDK

Android **SDK stands for Android Software Development Kit** which is developed by Google for Android Platform. With the help of Android SDK, we can create Android Apps easily.

About Android SDK

Android SDK is a **collection of libraries and Software Development tools** that are essential for Developing Android Applications. Whenever Google **releases a new version or update of Android Software, a corresponding SDK also releases with it**.

Components of Android SDK

Android SDK Components play a major role in the Development of Android applications. Below are the important components:

1. Android SDK Tools

Android SDK tool is an important component of Android SDK. It **consists of a complete set of development and debugging tools.** Below are the SDK developer tools:

- Android SDK Build tool.
- Android Emulator.

- Android SDK Platform-tools.
- Android SDK Tools.

2. Android SDK Build-Tools

Android SDK build tools are used for **building actual binaries of Android App**. The **main functions of Android SDK Build tools are built, debug, run and test Android applications.**

3. Android Emulator

An Android Emulator is a device that **simulates an Android device on your system.**. In **Android Emulator the virtual android device is shown on our system on which we run the Android application that we code.**

4. Android SDK Platform-tools

Android SDK Platform-tools is helpful when we are working on a Project and they will **show the error messages at the same time.** It is specifically **used for testing**. I

5. Android SDK Tools

Android SDK tool is a component of SDK tool. It consists of a set of tools which and other Utilities which are crucial for the development of Android Application. It contains the **complete set of Debugging and Development tools for android.**

6. SDK Platforms

For Each Android Software, one **SDK platform is available as shown below**:
Like in this **Android 11.0(R) is installed.**
These are numbered according to the android version.

Android Emulator

An Android emulator is a tool that creates virtual Android devices (with software and hardware) on your computer. Note that:

- It is a program (a process that runs on your computer's operating system).
- It works by mimicking the guest device's architecture (more on that in a bit).

Capabilities

Data transfer is faster on a virtual device (than a physical device connected via USB).
The drag-and-drop file upload lets you place .apk files from your computer to the

virtual mobile device. It's particularly great when developers need to quickly test apps under context.
The emulator is also pretty useful when you're working with physical sensors like the accelerometer. If you were testing a specific app feature that relies on the sensors, it'll be easier to configure the settings through the visual, extended controls of the emulator.

- Limitations
  The most popular chipset for Android smartphones out there is ARM v7a. Most PCs/laptops run on Intel (x86). Recall that guest and host CPU architectures need to match for faster emulation. Basically, without a computer equipped with an ARM processor, you're stuck with poor emulation of most of the commercially-available Android devices.
  The AVD Manager creates separate directories to store each virtual device's user data, SD card data, and cache. A single virtual device can take as much as 3.5GB of your disk space. Over time, a library of virtual devices will clam up your workstation.
  Virtual devices' performance is affected by that of your workstation. The emulator will crash and burn if you don't have enough free disk space at launch.
  Enabling hardware-acceleration takes care of performance issues. But setting up hardware-acceleration is a complex process that even experienced developers struggle with. The results often lead to complete system failure.

Requirement and recommendations

- SDK Tools 26.1.1 or higher
- 64-bit processor
- Windows: CPU with UG (unrestricted guest) support
- HAXM 6.2.1 or later (recommended HAXM 7.2.0 or later)

UNIT - 2

Main Components of Andriod APP

There are some necessary building blocks that an Android application consists of. These loosely **coupled components are bound by the application manifest file** which contains the **description of each component and how they interact**. The **manifest file also contains the app's metadata**, its h**ardware configuration, and platform requirements, external libraries, and required permissions.**

There are the following **main components of an android app:**

1. Activities

Activities are said to be the **presentation layer of our applications**. The **UI of our application is built around one or more extensions of the Activity class**. By using **Fragments and Views, activities set the layout and display the output and also respond to the user's actions**.
An activity is implemented as a subclass of class Activity.

- Java
- Kotlin

Tags :

```
public class MainActivity extends Activity {
}
```

2. Services

Services are **like invisible workers of our app.** These components run at the **backend, updating your data sources and Activities, triggering Notification, and also broadcast Intents**. They also **perform some tasks when applications are not active.**
A service can be used as a subclass of class Service:

- Java
- Kotlin

```
public class ServiceName extends Service {
}
```

3. Content Providers

It is used to **manage and persist the application data and also typically interacts with the SQL database.** They are also **responsible for sharing the data beyond the application boundaries**. The **Content Providers of a particular application can be configured to allow access from other applications,** and the Content Providers exposed by other applications can also be configured.
A content provider should be a subclass of the class ContentProvider.

- Java
- Kotlin

```
public class contentProviderName extends  ContentProvider {
   public void onCreate(){}
}
```

4. Broadcast Receivers

They are known to be **intent listeners as they enable your application to listen to the Intents** that satisfy the matching criteria specified by us. Broadcast Receivers make our application react to any received Intent thereby making them perfect for creating event-driven applications.

REPO (Android Tool Repository)

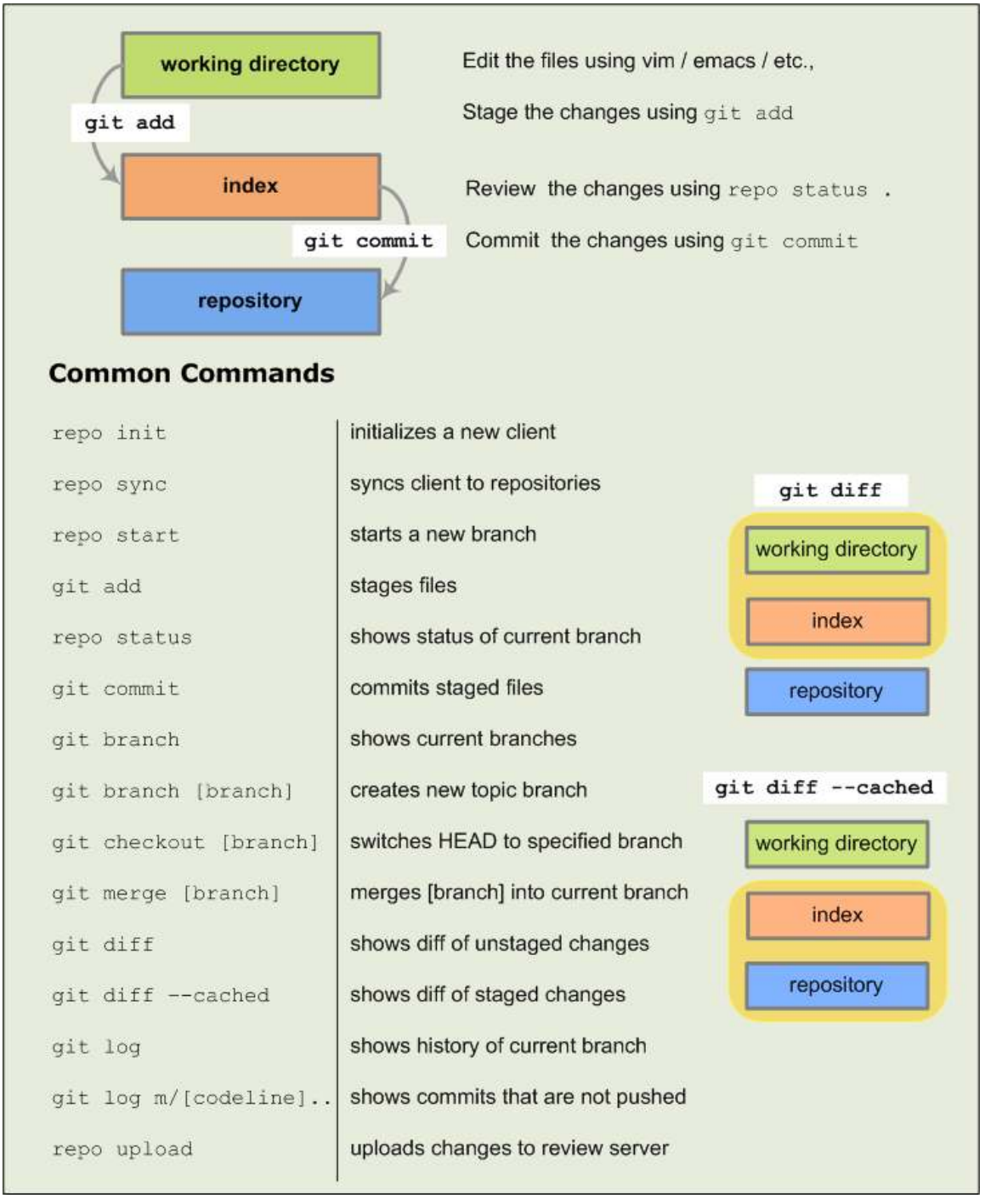
Repo is a **tool that we built on top of Git**. Repo helps us **manage the many Git repositories, does the uploads to our revision control system**, and automates parts of the Android development workflow. Repo is not meant to replace Git, only to make it e**asier to work with Git in the context of Android**. The repo command is an executable Python script that you can put anywhere in your path.

Different in REPO and GIT

Repo uses manifest files to aggregate Git projects into the Android superproject. You can put the repo command, which is an **executable Python script, anywhere in your path**. In working with t**he Android source files, you can use Repo for across-network operations** such as with a single Repo working directory.

Git is an open-source version-control system designed to handle very large projects that are **distributed over multiple repositories.** In the context of Android, we use Git for local operations such as local branching, commits, diffs, and edits.

Click on the cheatsheet to open it in a new window for easier printing.



Application manifest file

Every project in Android includes a Manifest XML file, which is AndroidManifest.xml, **located in the root directory** of its project hierarchy. The manifest file is an important part of our app because **it defines the structure and metadata of our application, its components, and its requirements**.

This file includes nodes for each of the **Activities, Services, Content Providers, and Broadcast Receiver** that make the application, and using **Intent Filters and Permissions determines how they coordinate with each other and other applications.**

The manifest file also **specifies the application metadata,** which includes **its icon, version number, themes, etc**., and additional t**op-level nodes can specify any required permissions, unit tests, and define hardware, screen, or platform requirements.** The manifest comprises a root manifest t**ag with a package attribute set to the project's package**. It should also include an xmls:android attribute that will supply several system attributes used within the file. The **versionCode attribute is used to define the current application version** in the form of an integer that increments itself with the iteration of the version due to update. Also, the versionName attribute is used to specify a public version that will be displayed to the users.

A **manifest file includes the nodes that define the application components**, security settings, test classes, and requirements that make up the application.

Some of Its Nodes are

1. manifest

The **main component of the AndroidManifest.xml** file is known as manifest. Additionally, the **package field describes the activity class's package name.** It must contain an <application> element with the xmlns:android and package attribute specified.

XML :

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    package="com.example.geeksforgeeks">

    <!-- manifest nodes -->
 <applicataion>

</applicataion>
 </manifest>
```

2. Uses-sdk

It is used to **define a minimum and maximum SDK version** by means of an **API Level integer** that must be available on a device so that our **application functions**

properly, and the target SDK for which it has been designed using a combination of minSdkVersion, maxSdkVersion, and targetSdkVersion attributes, respectively. It is **contained within the <manifest> element.**

- **XML**

```
<uses-sdk
    android:minSdkVersion="18"
    android:targetSdkVersion="27" />
```

3. uses-permission

It **outlines a system permission that must be granted by the user** for the app to function properly and is contained within the <manifest> element. When an application is installed (on Android 5.1 and lower devices or Android 6.0 and higher), the **user must grant the application permissions.**

- **XML**

```
<uses-permission
    android:name="android.permission.CAMERA"
    android:maxSdkVersion="18" />
```

4. Application

A manifest can **contain only one application node.** It uses attributes to specify the **metadata for your application (including its title, icon, and theme)**. During development, we should include a debuggable attribute set to true to enable debugging, then be sure to disable it for your release builds. The **application node also acts as a container for the Activity, Service, Content Provider, and Broadcast Receiver nodes that specify the application components**. The name of our custom application class can be specified using the **android:name attribute.**

- XML

```
<application
   <android:name=".GeeksForGeeks"

   android:icon="@drawable/gfgIcon"
  >
 <!-- application nodes -->

</application>
```

5. uses-library

It defines a **shared library against which the application must be linked**. This **element instructs the system to add the library's code to the package's class loader.** It is contained within the <application> element.

- **XML**

```
<uses-library
    android:name="android.test.runner"
    android:required="true" />
```

6. Activity

The **Activity sub-element of an application refers to an activity that needs to be specified in the AndroidManifest.xml file**. It has various characteristics, like **label, name, theme, launchMode, and others**. In the manifest file, all elements must be **represented by <activity>.** Any activity that is not declared there won't run and won't be visible to the system. It is contained within the <application> element.

- XML

```
<activity
    android:name=".MainActivity"
    android:exported="true">
</activity>
```

7. Intent-filter

It is the sub-element of activity that s**pecifies the type of intent to which the activity, service, or broadcast receiver can send a response.** It allows the component to receive intents of a **certain type while filtering out those that are not useful for the component**. The intent filter must contain at least one <action> element.

- XML

```
<intent-filter>
    <action android:name="android.intent.action.MAIN" />

    <category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
```

8. Action

It adds an **action for the intent-filter**. It is contained within the <intent-filter> element.

- XML

```
<action android:name="android.intent.action.MAIN" />
```

9. Category

It adds a **category name to an intent-filter**. It is contained within the <intent-filter> element.

- XML

```
<category android:name="android.intent.category.LAUNCHER" />
```

10. Uses-configuration

The uses-configuration components are used to specify the **combination of input mechanisms that are supported by our application**. It is **useful for games that require particular input controls.**

- **XML**

```
<uses-configuration
  android:reqTouchScreen=”finger”
  android:reqNavigation=”trackball”
  android:reqHardKeyboard=”true”
  android:reqKeyboardType=”qwerty”/>
```

11. uses-features

It is used to specify which **hardware features your application requires**. This will prevent our application from b**eing installed on a device that does not include a required piece of hardware** such as NFC hardware, as follows:

- XML

```
<uses-feature android:name=”android.hardware.nfc” />
```

12. permission

It is used to **create permissions to restrict access to shared application components**. We can also use the existing platform permissions for this purpose or define your own permissions in the manifest.

- **XML**

```
<permission
   android: name=”com.paad.DETONATE_DEVICE”
   android:protectionLevel=“dangerous”
   android:label=”Self Destruct”
   android:description=”@string/detonate_description”>
</permission>
```

DDMS (Dalvik Debug Monitor Server)

Android ships with a **debugging too**l called the Dalvik Debug Monitor Server (DDMS), which provides **port-forwarding services, screen capture on the device, thread and heap information on the device, logcat, process, and radio state information, incoming call and SMS spoofing, location data spoofing, and more**. it is not an exhaustive exploration of all the features and capabilities.
DDMS ships in the **tools/ directory of the SDK**. Enter this directory from a terminal/console and type ddms (or ./ddms on Mac/Linux) to run it. DDMS will **work with both the emulator and a connected device**. If both are connected and running simultaneously, DDMS defaults to the emulator.
DDMS stands for Dalvik Debug Monitor Server. It gives the wide array of **debugging features:**

1. Port forwarding services
2. Screen capture
3. Thread and heap information
4. Network traffic tracking
5. Location data spoofing

File Explorer

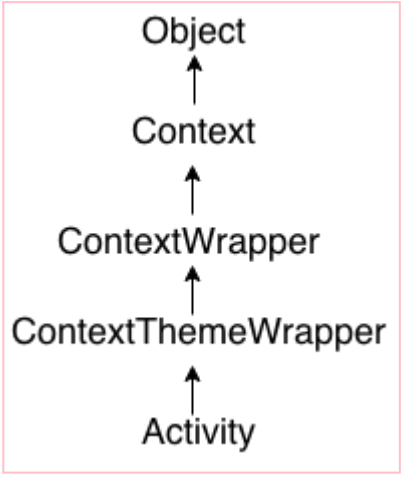
With the File Explorer, you can vi**ew the device file system and perform basic management, like pushing and pulling files**. This circumvents **using the adb push and pull commands, with a GUI experience.**

With DDMS open, select Device > File Explorer... to open the File Explorer window. You can drag-and-drop into the device directories, but cannot drag out of them. To copy files from the device, select the file and click the Pull File from Device button in the toolbar. To delete files, use the Delete button in the toolbar.

If you're interested in using an SD card image on the emulator, you're still required to use the mksdcard command to create an image, and then mount it during emulator bootup. For example, from the /tools directory, execute:
$ mksdcard 1024M ./img
$ emulator -sdcard ./img

Now, when the emulator is running, the DDMS File Explorer will be able to read and write to the sdcard directory. However, your files may not appear automatically. For example, if you add an MP3 file to the sdcard, the media player won't see them until you restart the emulator. (When restarting the emulator from command line, be sure to mount the sdcard again.)

Android Activity Lifecycle

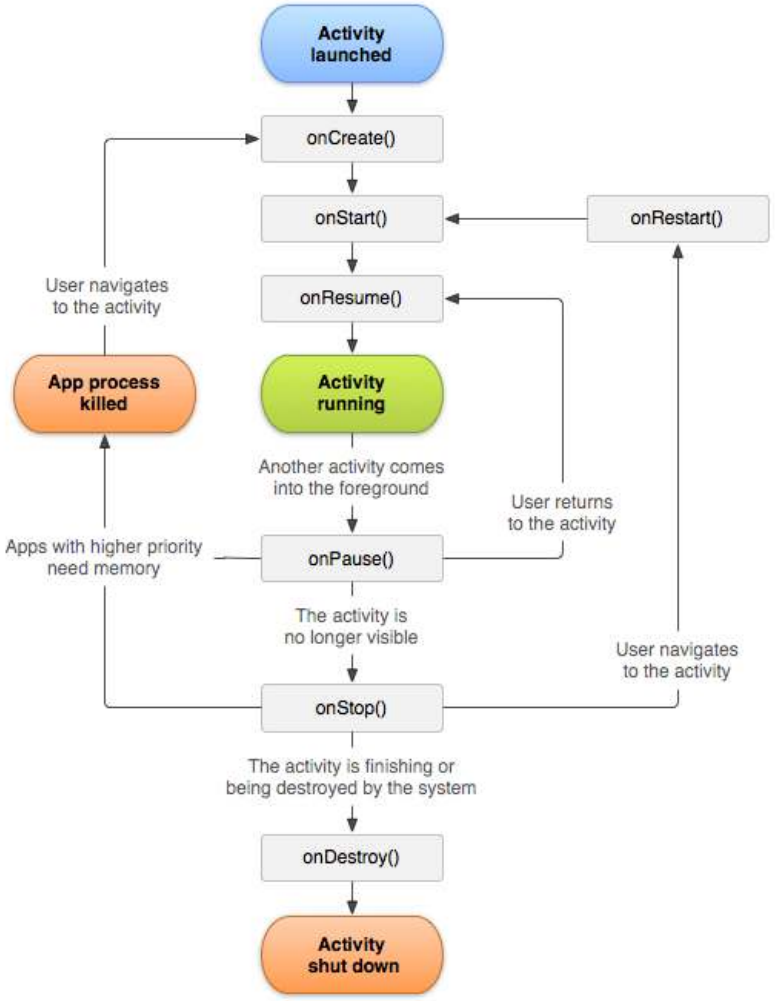


Android Activity Lifecycle is **controlled by 7 methods of android.app**.
Activity class. The android Activity is the **subclass of ContextThemeWrapper class**.
An activity is the **single screen in android.**
It is like a **window or frame of Java.**
By the help of activity, you can place all your **UI components or widgets in a single screen.**
The **7 lifecycle method of Activity describes how activity will behave at different states.**

Android Activity Lifecycle methods

Let's see the **7 lifecycle methods of android 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.



Android Activity Lifecycle Example

It provides the details about the invocation of **life cycle methods of activity**. In this example, we are displaying the content on the logcat.

```
package example.javatpoint.com.activity lifecycle;

import android.app.Activity;
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Log.d("lifecycle","onCreate invoked");
    }
    @Override
    protected void onStart() {
        super.onStart();
        Log.d("lifecycle","onStart invoked");
    }
    @Override
    protected void onResume() {
        super.onResume();
        Log.d("lifecycle","onResume invoked");
    }
    @Override
    protected void onPause() {
        super.onPause();
        Log.d("lifecycle","onPause invoked");
    }
    @Override
    protected void onStop() {
        super.onStop();
        Log.d("lifecycle","onStop invoked");
    }
    @Override
    protected void onRestart() {
        super.onRestart();
        Log.d("lifecycle","onRestart invoked");
    }
    @Override
    protected void onDestroy() {
        super.onDestroy();
        Log.d("lifecycle","onDestroy invoked");      }  }
```

LOGCAT WINDOW

LogCat Window is the **place where various messages can be printed when an application runs**. Suppose, you are running your application and the program crashes, unfortunately. Then, **Logcat Window is going to help you to debug the output by collecting and viewing all the messages that your emulator throws.**

So, this is a **very useful component for the app development because this Logcat dumps a lot of system messages and these messages are actually thrown by the emulator.**

This means, that when you run your app in your emulator, then you will see a lot of messages which i**nclude all the information, all the verbose messages, and all the errors that you are getting in your application**. Suppose, an application of about 10000 lines of code gets an error. So, in that 10000 line codes, to detect the error Logcat helps by displaying the error messages.

Errors in different modules and methods can be easily detected with the help of the LogCat window.

Using LogCat window:

The **LogCat prints an error using a Log Class**. The class which is used to print the log messages is actually known as a Log Class. So, this class is responsible to **print messages in the Logcat terminal.**

There are lots of methods that are present in the log class:

v(String, String)	verbose
d(String, String)	debug
i(String, String)	information
w(String, String)	warning
e(String, String)	error

All these methods contain two parameters in them. The **first and second both are the string. When you print the log messages with these different methods**, then you will get a corresponding color according to the method.

Method	The output will be printed in
verbose	black
debug	blue
information	green
warning	red
error	orange

The **verbose method is of very lesser priority and erro**r is of higher priority. Thus, the method's priorit**y increases from verbose to error.**

Syntax:

```
// for verbose Log.v("TAG", "MESSAGE");
// for debug Log.d("TAG", "MESSAGE");
// for information Log.i("TAG", "MESSAGE");
// for warning Log.w("TAG", "MESSAGE");
// for error Log.e("TAG", "MESSAGE");
```

For Example, A verbose log message can be written as:

```
Log.v("MainActivity", "We are under the Main Activity");
```

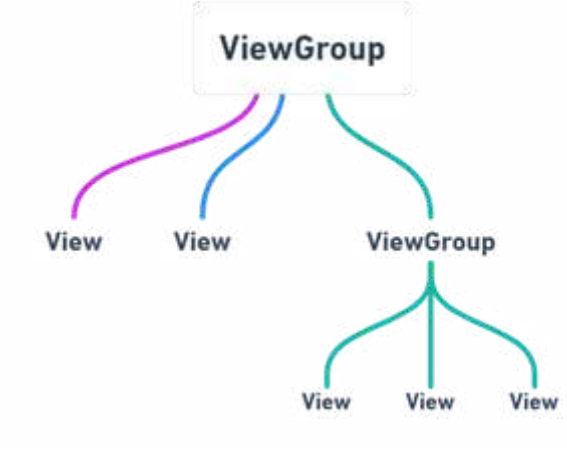
UNIT - 3

Andriod layouts

Android Layout is used to define the user interface that holds the UI controls or widgets that will appear on the screen of an android application or activity screen. Generally, every application is a combination of View and ViewGroup. As we know, an android application contains a large number of activities and we can say each activity is one page of the application. So, each activity contains multiple user interface components and those components are the instances of the View and ViewGroup. All the elements in a layout are built using a hierarchy of View and ViewGroup objects.

A View is defined as the user interface which is used to create interactive UI components such as TextView, ImageView, EditText, RadioButton, etc., and is responsible for event handling and drawing. They are Generally Called Widgets.

A ViewGroup act as a base class for layouts and layouts parameters that hold other Views or ViewGroups and to define the layout properties. They are Generally Called layouts.



The Android framework will allow us to use UI elements or widgets in two ways:

Use UI elements in the XML file
Create elements in the Kotlin file dynamically

Types of Android Layout

- Android Linear Layout: LinearLayout is a ViewGroup subclass, used to provide child View elements one by one either in a particular direction either horizontally or vertically based on the orientation property.
- Android Relative Layout: RelativeLayout is a ViewGroup subclass, used to specify the position of child View elements relative to each other like (A to the right of B) or relative to the parent (fix to the top of the parent).
- Android Constraint Layout: ConstraintLayout is a ViewGroup subclass, used to specify the position of layout constraints for every child View relative to other views present. A ConstraintLayout is similar to a RelativeLayout, but having more power.
- Android Frame Layout: FrameLayout is a ViewGroup subclass, used to specify the position of View elements it contains on the top of each other to display only a single View inside the FrameLayout.
- Android Table Layout: TableLayout is a ViewGroup subclass, used to display the child View elements in rows and columns.
- Android Web View: WebView is a browser that is used to display the web pages in our activity layout.
- Android ListView: ListView is a ViewGroup, used to display scrollable lists of items in a single column.
- Android Grid View: GridView is a ViewGroup that is used to display a scrollable list of items in a grid view of rows and columns.

UI resource

Resources are the additional files and static content that your code uses, such as bitmaps, layout definitions, user interface strings, animation instructions, and more. You should always externalize app resources such as images and strings from your code, so that you can maintain them independently

Android UI Controls

There are number of UI controls provided by Android that allow you to build the graphical user interface for your app.

Sr.No.	UI Control & Description
1	TextView This control is used to display text to the user.
2	EditText EditText is a predefined subclass of TextView that includes rich editing capabilities.
3	AutoCompleteTextView The AutoCompleteTextView is a view that is similar to EditText, except that it shows a list of completion suggestions automatically while the user is typing.
4	Button A push-button that can be pressed, or clicked, by the user to perform an action.

5	ImageButton An ImageButton is an AbsoluteLayout which enables you to specify the exact location of its children. This shows a button with an image (instead of text) that can be pressed or clicked by the user.
6	CheckBox An on/off switch that can be toggled by the user. You should use check box when presenting users with a group of selectable options that are not mutually exclusive.
7	ToggleButton An on/off button with a light indicator.
8	RadioButton The RadioButton has two states: either checked or unchecked.
9	RadioGroup A RadioGroup is used to group together one or more RadioButtons.
10	ProgressBar The ProgressBar view provides visual feedback about some ongoing tasks, such as when you are performing a task in the background.
11	Spinner A drop-down list that allows users to select one value from a set.

12	TimePicker The TimePicker view enables users to select a time of the day, in either 24-hour mode or AM/PM mode.
13	DatePicker The DatePicker view enables users to select a date of the day.

Create UI Controls

Input controls are the interactive components in your app's user interface. Android provides a wide variety of controls you can use in your UI, such as buttons, text fields, seek bars, check box, zoom buttons, toggle buttons, and many more.

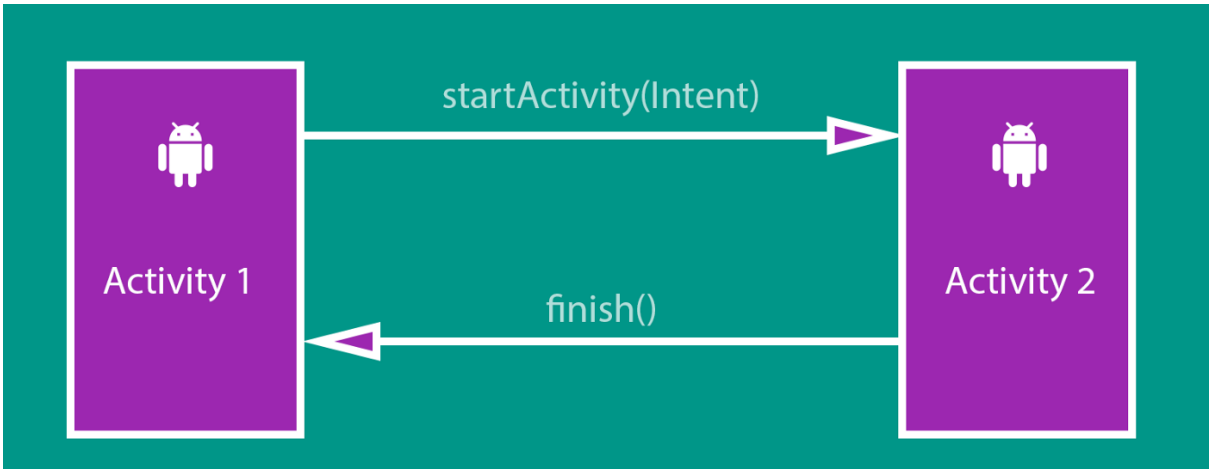
As explained in previous chapter, a view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is –

```
android:id="@+id/text_id"
```

To create a UI Control/View/Widget you will have to define a view/widget in the layout file and assign it a unique ID as follows –

```
<?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"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="I am a TextView" />
</LinearLayout>
```

Navigate Between Activities in Android Studio



Switching between pages in an application is one of the basic features of an app. We can do that by adding few lines of code.

For that open android studio and create a new project.

create new project > Empty Activity >Next > Enter name of the project > Finish

Create two activies that we can navigate in using a button.

(We can navigate between pages using pretty much every element, not necessarily button).

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>

<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:app="http://schemas.android.com/apk/res-auto"

    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    tools:context=".MainActivity">



    <Button
```

```
        android:id="@+id/page1"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:text="This Is the First Page"

        app:layout_constraintBottom_toBottomOf="parent"

        app:layout_constraintLeft_toLeftOf="parent"

        app:layout_constraintRight_toRightOf="parent"

        app:layout_constraintTop_toTopOf="parent" />



</androidx.constraintlayout.widget.ConstraintLayout>
```

activity_main2.xml

```
<?xml version="1.0" encoding="utf-8"?>
```

```
<androidx.constraintlayout.widget.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"

    xmlns:app="http://schemas.android.com/apk/res-auto"

    xmlns:tools="http://schemas.android.com/tools"

    android:layout_width="match_parent"

    android:layout_height="match_parent"

    tools:context=".MainActivity2">









        <Button

            android:id="@+id/page2"

            android:layout_width="wrap_content"

            android:layout_height="wrap_content"
```

```
            android:text="This Is the Second Page"

            app:layout_constraintBottom_toBottomOf="parent"

            app:layout_constraintLeft_toLeftOf="parent"

            app:layout_constraintRight_toRightOf="parent"

            app:layout_constraintTop_toTopOf="parent" />




</androidx.constraintlayout.widget.ConstraintLayout>
```

Don’t forget to give **id’s** to the buttons which we will need in MainActivity.java file.

To navigate from **activity_main.xml** to **activity_main2.xml** we have to write the code in **MainActivity.java file**.

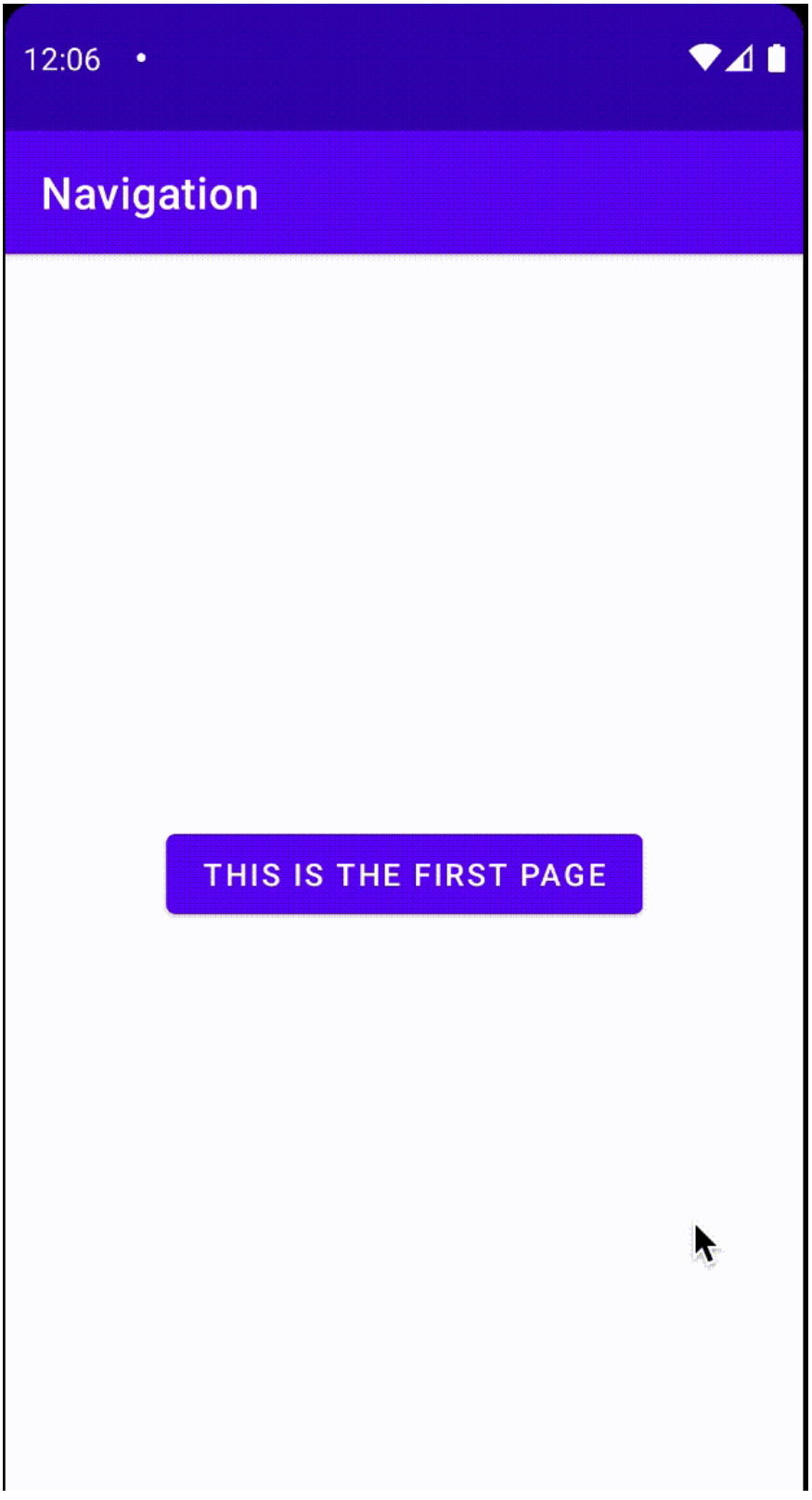
We have to set an **onClickListner** to the element which we are going to use to navigate between pages (in this case button).

```
package com.example.navigation;




import android.content.Intent;

import android.view.View;

import android.widget.Button;

import androidx.appcompat.app.AppCompatActivity;




import android.os.Bundle;




public class MainActivity extends AppCompatActivity {
```

```
  Button b1  ;


   @Override

   protected void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.activity_main);



        b1 = findViewById(R.id.page1);

        b1.setOnClickListener(

                new View.OnClickListener() {

                    @Override
```

```
                    public void onClick(View v) {

                Intent i = new
Intent(MainActivity.this,MainActivity2.class);

                        startActivity(i);

                    }

                }

        );



    }

}
```

Now after clicking the button in the first page the second page is opened.



Thank You , Happy Coding!

ACTion bar

n Android applications, **ActionBar** is the element present at the top of the activity screen. It is a salient feature of a mobile application that has a consistent presence over all its activities. 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)**. Before that, the name of this top most visual element was **AppBar**. AppBar contains only the name of the application or current activity. It was not very much useful for the users and developers also have negligible option to customize it.

Google announced a **support library** along with the introduction of ActionBar. This library is a part of **AppCompat** and its purpose is to provide backward compatibility for older versions of Android and to support tabbed interfaces. All applications that use the default theme provided by the Android(Theme.AppCompat.Light.DarkActionBar), contains an ActionBar by default. However, developers can customize it in several ways depending upon their needs. Components included in the ActionBar are:

- **App Icon:** Display the branding logo/icon of the application.
- **View Controls:** Section that displays the name of the application or current activity. Developers can also include spinner or tabbed navigation for switching between views.
- **Action Button:** Contains some important actions/elements of the app that may be required to the users frequently.
- **Action Overflow:** Include other actions that will be displayed as a menu.

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.

Step 1: Default ActionBar

As mentioned earlier, 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. The string value of the application's title is provided by **@string/app_name** resource present under the **application nodes**.

Output:

Event Handling in Android

Events are the actions performed by the user in order to interact with the application, for e.g. pressing a button or touching the screen. The events are managed by the android framework in the **FIFO** manner i.e. First In – First Out. Handling such actions or events by performing the desired task is called **Event Handling.**

Overview of the input event management

- **Event Listeners:** It is an interface in the View class. It contains a single callback method. Once the view to which the listener is associated is triggered due to user interaction, the callback methods are called.
- **Event Handlers:** It is responsible for dealing with the event that the event listeners registered for and performing the desired action for that respective event.
- **Event Listeners Registration**: Event Registration is the process in which an Event Handler gets associated with an Event Listener so that this handler is called when the respective Event Listener fires the event.
- **Touch Mode:** When using an app with physical keys it becomes necessary to give focus to buttons on which the user wants to perform the action but if the device is touch-enabled and the user interacts with the interface by touching it, then it is no longer necessary to highlight items or give focus to particular View. In such cases, the device enters touch mode and in such scenarios, only

those views for which the isFocusableInTouchMode() is true will be focusable, e.g. plain text widget.

For e.g. if a button is pressed then this action or event gets registered by the event listener and then the task to be performed by that button press is handled by the event handler, it can be anything like changing the color of the text on a button press or changing the text itself, etc.

Listeners

An event listener is **an interface in the View class that contains a single callback method**. These methods will be called by the Android framework when the View to which the listener has been registered is triggered by user interaction with the item in the UI.

Event Listeners and their respective event handlers

- **OnClickListener()** – This method is called when the user clicks, touches, or focuses on any view (widget) like Button, ImageButton, Image, etc. **Event handler used for this is onClick().**
- **OnLongClickListener()** – This method is called when the user presses and holds a particular widget for one or more seconds. **Event handler used for this is onLongClick().**
- **OnMenuItemClickListener()** – This method is called when the user selects a menu item. **Event handler used for this is onMenuItemClick().**
- **OnTouch()** – This method is called either for a movement gesture on the screen or a press and release of an on-screen key. **Event handler used for this is onTouch().**

Event Listeners & Event Handlers

Event Handler	Event Listener & Description
onClick()	OnClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.
onLongClick()	OnLongClickListener() This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	OnFocusChangeListener() This is called when the widget looses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event.
onKey()	OnFocusChangeListener() This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.

onTouch()	OnTouchListener() This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.
onMenuItemClick()	OnMenuItemClickListener() This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event.
onCreateContextMenu()	onCreateContextMenuItemListener() This is called when the context menu is being built(as the result of a sustained "long click)

What is Toast in Android?

A Toast is a feedback message. It takes a very little space for displaying while overall activity is interactive and visible to the user. It disappears after a few seconds. It disappears automatically. If user wants permanent visible message, Notification can be used.

Toast class: Toast class provides a simple popup message that is displayed on the current activity UI screen (e.g. Main Activity).

Constants of Toast class

Constants	Description

public static final int LENGTH_LONG	**displays for a long time**
public static final int LENGTH_SHORT	**displays for a short time**

Methods of Toast class

Methods	Description
public static Toast makeText(Context context, CharSequence text, int duration)	**makes the toast message consisted of text and time duration**
public void show()	**displays a toast message**
public void setMargin (float horizontalMargin, float verticalMargin)	**changes the horizontal and vertical differences**

Full code of activity class displaying Toast

Let's see the code to display the toast.

File: MainActivity.java

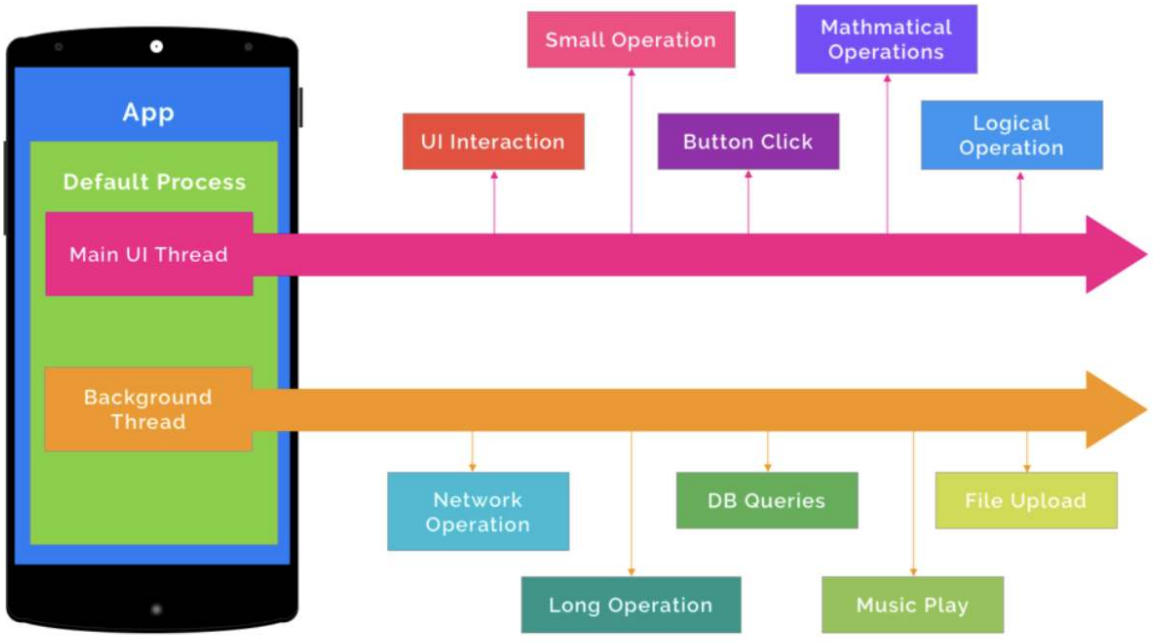
```
1. package example.javatpoint.com.toast;
2.
3. import android.support.v7.app.AppCompatActivity;
4. import android.os.Bundle;
5. import android.widget.Toast;
6.
7. public class MainActivity extends AppCompatActivity {
8.
9.     @Override
10.    protected void onCreate(Bundle savedInstanceState) {
11.        super.onCreate(savedInstanceState);
12.        setContentView(R.layout.activity_main);
13.
14.        //Displaying Toast with Hello Javatpoint message
15.        Toast.makeText(getApplicationContext(),"Hello
    Javatpoint",Toast.LENGTH_SHORT).show();
16.    }
17.}
```

Output:

UNIT - 4

Thread in Andriod

Thread is one of the important concepts in Android. Thread is a lightweight sub-process that provides us a way to do background operations without interrupting the User Interface (UI). When an app is launched, it creates a single thread in which all app components will run by default. The thread which is created by the runtime system is known as the main thread. The main thread's primary role is to handle the UI in terms of event handling and interaction with views in the UI. If there is a task that is time-consuming and that task is run on the main thread, then it will stop other tasks until it gets completed, which in turn may result in displaying a warning "Application is unresponsive" to the user by the operating system. So we need different threads for such tasks and some other tasks.



All threading components belong to one of two basic categories.

- The fragment or activity attached threads: This category of threads are bound to the lifecycle of the activity/fragment and these are terminated as soon as the activity/fragment is destroyed.

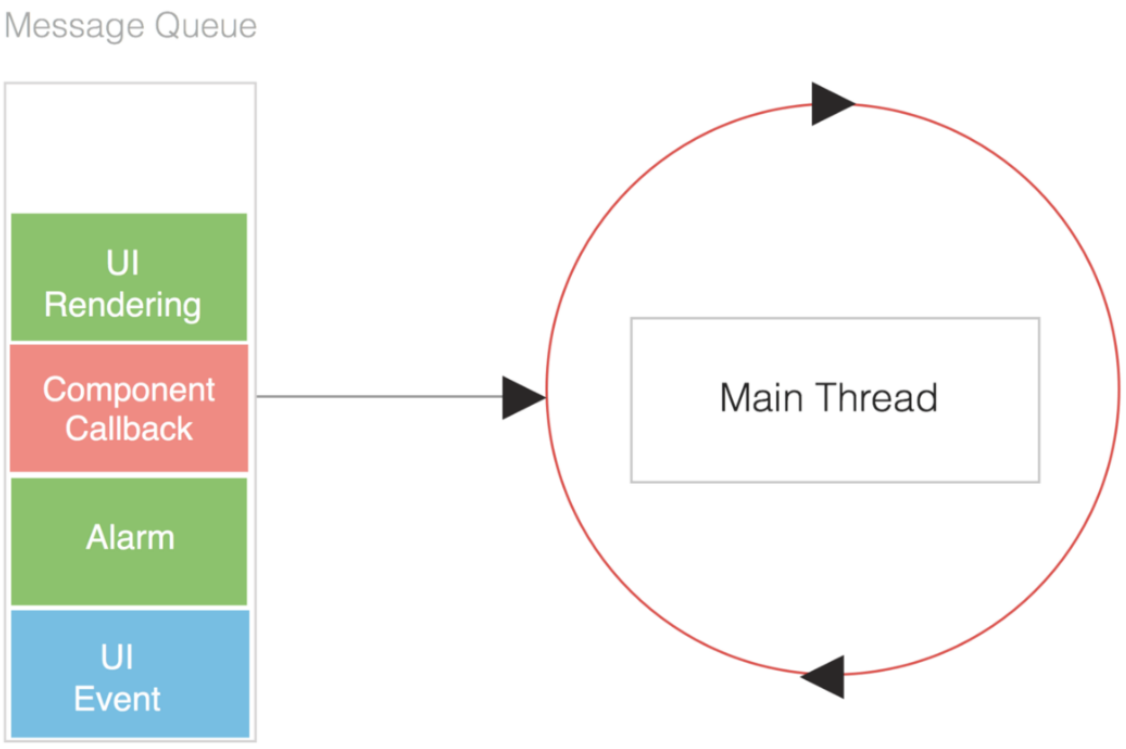
Thread components:
AsyncTask, Loaders.

- The fragment or activity not attached threads: These types of threads can continue to run beyond the lifetime of the activity or fragment from which they were spawned.

Threading Components: Service, Intent Service.

For the two threading components, there are five types of thread used in Android mobile development.

- Main Thread: When we launch our app on Android, it creates the first thread of execution called the “Main Thread”. The communication between the components from the Android UI toolkit and the dispatching of events to their appropriate UI widgets is handled by the main thread. We should avoid network operations, database calls, and the loading of certain components in the main thread. Because the main thread is called synchronously when executed, that means the user interface will remain completely unresponsive until the performance completes.



- UI Thread: Every app in Android has its own thread which is responsible for running the UI objects, like view objects. Such a thread is known as the UI thread. The UI thread is the main thread of execution for our app as this is where most of the app code is run. The UI thread is where all of our app components (like activities, services, content providers, and broadcast receivers) are created. This thread allows our tasks to perform their background work and then move the results to UI elements such as bitmaps. All objects running on our UI thread will be able to access other objects which are also running on the same UI thread. The tasks that we run on a thread from a thread pool do not run on our UI thread, so they will not have access to UI objects. The data moves from a background thread to the UI thread, using a handler that runs on the UI thread.

- Worker Thread: The worker thread is a background thread. The worker threads are created separately, other than threads like the UI thread. As we know from the rules, we cannot block a UI thread so this is where the worker thread comes into play since we can use them to run the child processes and tasks.

- Any Thread:

```
1
2
3
4
5
6
7
8
@Target([
  AnnotationTarget.FUNCTION, AnnotationTarget.PROPERTY_GETTER,
  AnnotationTarget.PROPERTY_SETTER,
AnnotationTarget.CONSTRUCTOR,
  AnnotationTarget.CLASS, AnnotationTarget.FILE,
  AnnotationTarget.VALUE_PARAMETER
])

class AnyThread
```

In Any thread, the annotated method can be called from any thread. If the annotated element is a class, then all methods in the class can be called from Any Thread.

- Binder Thread: Binder thread represents a separate thread of service. The binder is a mechanism that provides inter-process communication. The binder thread is used in service binding with interprocess communication. This concept is mainly related to service calls with interfaces defined by Android Interface Definition Language (AIDL).

MUltmedia

The Android multimedia framework includes support for playing variety of common media types, so that you can easily integrate audio, video and images into your applications. You can play audio or video from media files stored in your application's resources (raw resources), from standalone files in the filesystem, or from a data stream arriving over a network connection, all using MediaPlayer APIs.

Method	Description
public void setDataSource(String path)	sets the data source (file path or http url) to use.
public void prepare()	prepares the player for playback synchronously.
public void start()	it starts or resumes the playback.
public void stop()	it stops the playback.
public void pause()	it pauses the playback.
public boolean isPlaying()	checks if media player is playing.
public void seekTo(int millis)	seeks to specified time in miliseconds.
public void setLooping(boolean looping)	sets the player for looping or non-looping.
public boolean isLooping()	checks if the player is looping or non-looping.

public void selectTrack(int index)	it selects a track for the specified index.
public int getCurrentPosition()	returns the current playback position.
public int getDuration()	returns duration of the file.
public void setVolume(float leftVolume,float rightVolume)	sets the volume on this player.

```
//package com.java2s;
import java.io.File;

import android.content.Context;
import android.content.Intent;
import android.net.Uri;

public class Main {
    public static void PlayAudio(Context context, String audioPath) {

        Intent intent = new Intent();
        intent.setAction(android.content.Intent.ACTION_VIEW);
        File file = new File(audioPath.toString());
        intent.setDataAndType(Uri.fromFile(file), "audio/*");
        context.startActivity(intent);//from w ww. j ava 2s . c om
    }
```

Android Video Player

MediaController class

The **android.widget.MediaController** is a view that contains media controls like play/pause, previous, next, fast-forward, rewind etc.

VideoView class

The **android.widget.VideoView** class provides methods to play and control the video player. The commonly used methods of VideoView class are as follows:

Method	Description
public void setMediaController(MediaController controller)	sets the media controller to the video view.
public void setVideoURI (Uri uri)	sets the URI of the video file.
public void start()	starts the video view.
public void stopPlayback()	stops the playback.
public void pause()	pauses the playback.
public void suspend()	suspends the playback.
public void resume()	resumes the playback.
public void seekTo(int millis)	seeks to specified time in miliseconds.

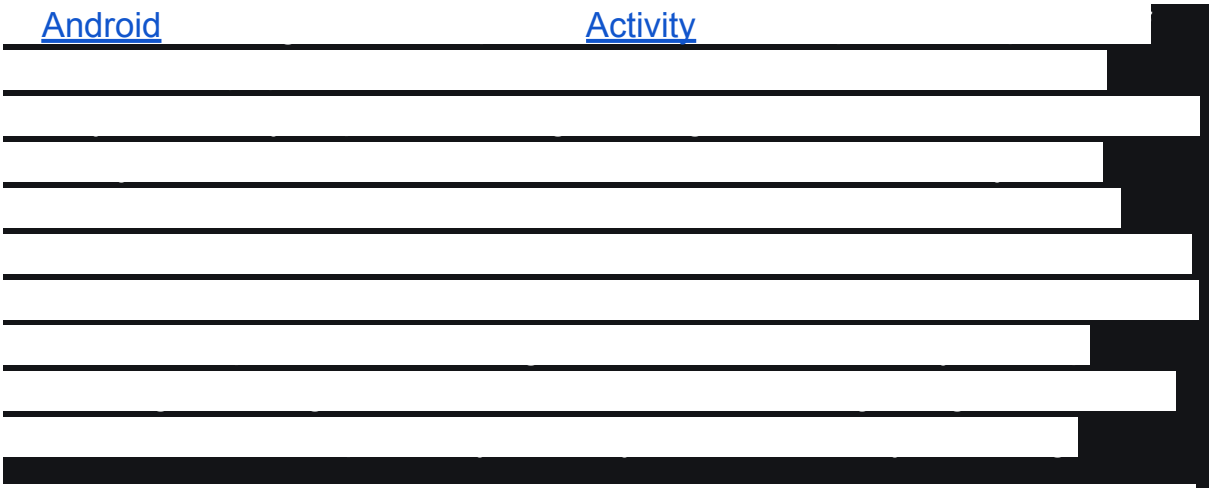
```
1. <RelativeLayout
   xmlns:androclass="http://schemas.android.com/apk/res/android"
2.     xmlns:tools="http://schemas.android.com/tools"
3.     android:layout_width="match_parent"
4.     android:layout_height="match_parent"
5.     tools:context=".MainActivity" >
6.
7.     <VideoView
8.         android:id="@+id/videoView1"
9.         android:layout_width="wrap_content"
10.        android:layout_height="wrap_content"
11.        android:layout_alignParentLeft="true"
```

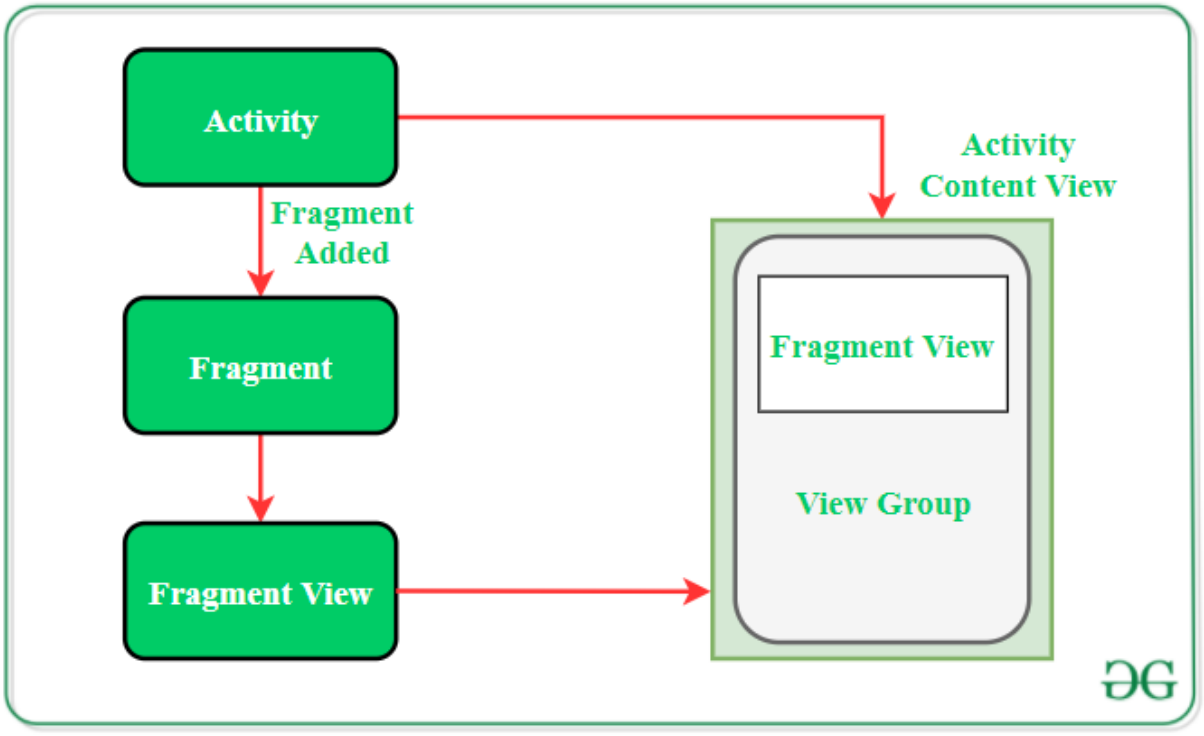
```
12.     android:layout_centerVertical="true" />
13.
14. </RelativeLayout>
```

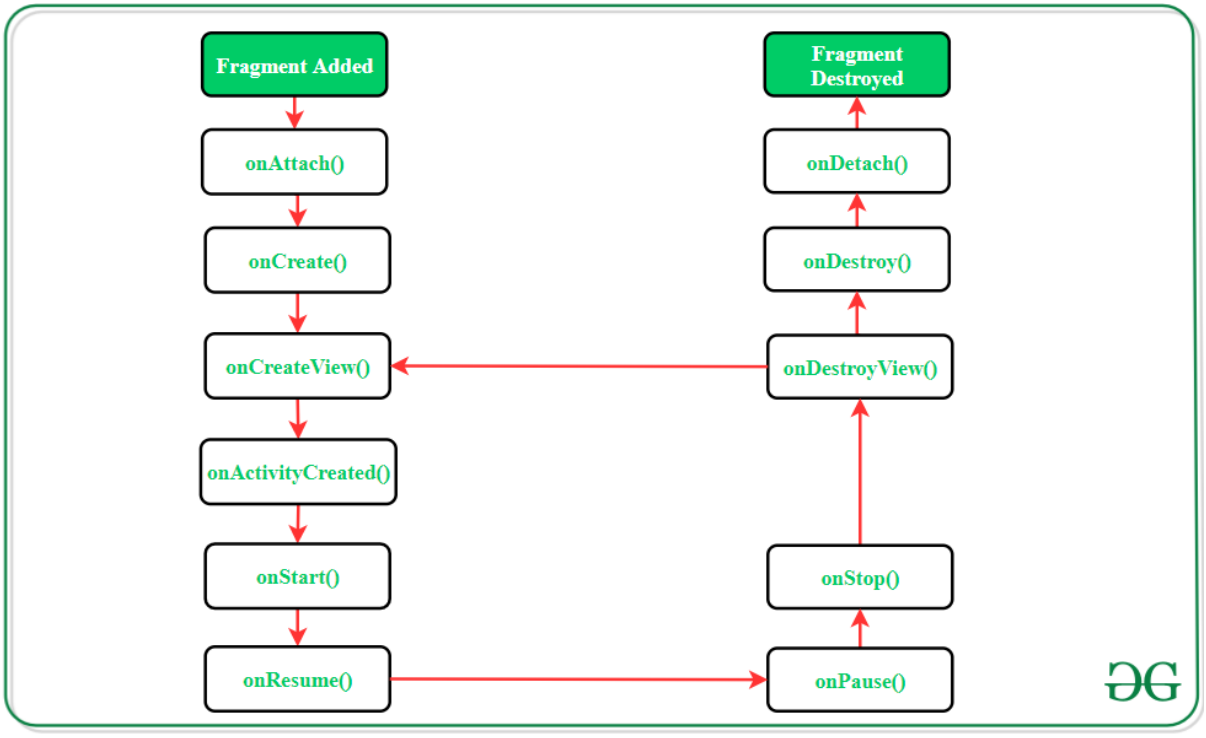
Fragment / cycle

A **Fragment** is a piece of an activity which enable more modular activity design. A fragment encapsulates functionality so that it is easier to reuse within activities and layouts.
Android devices exists in a variety of screen sizes and densities. Fragments simplify the reuse of components in different layouts and their logic. You can build single-pane layouts for handsets (phones) and multi-pane layouts for tablets. You can also use fragments also to support different layout for landscape and portrait orientation on a smartphone

Android Activity




Activity
Fragment
Added
Fragment
Fragment View
Activity
Content View
Fragment View
View Group



Methods	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.
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.
onActivityCreated()	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.
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.

UNIT - 5

SQL Lite Database

SQLite is another data storage available in Android where we can store data in the user's device and can use it any time when required. In this article, we will take a look at creating an SQLite database in the Android app and adding data to that database in the Android app. This is a series of 4 articles in which we are going to perform the basic **CRUD (Create, Read, Update, and Delete)** operation with **SQLite Database** in Android

What is SQLite Database?

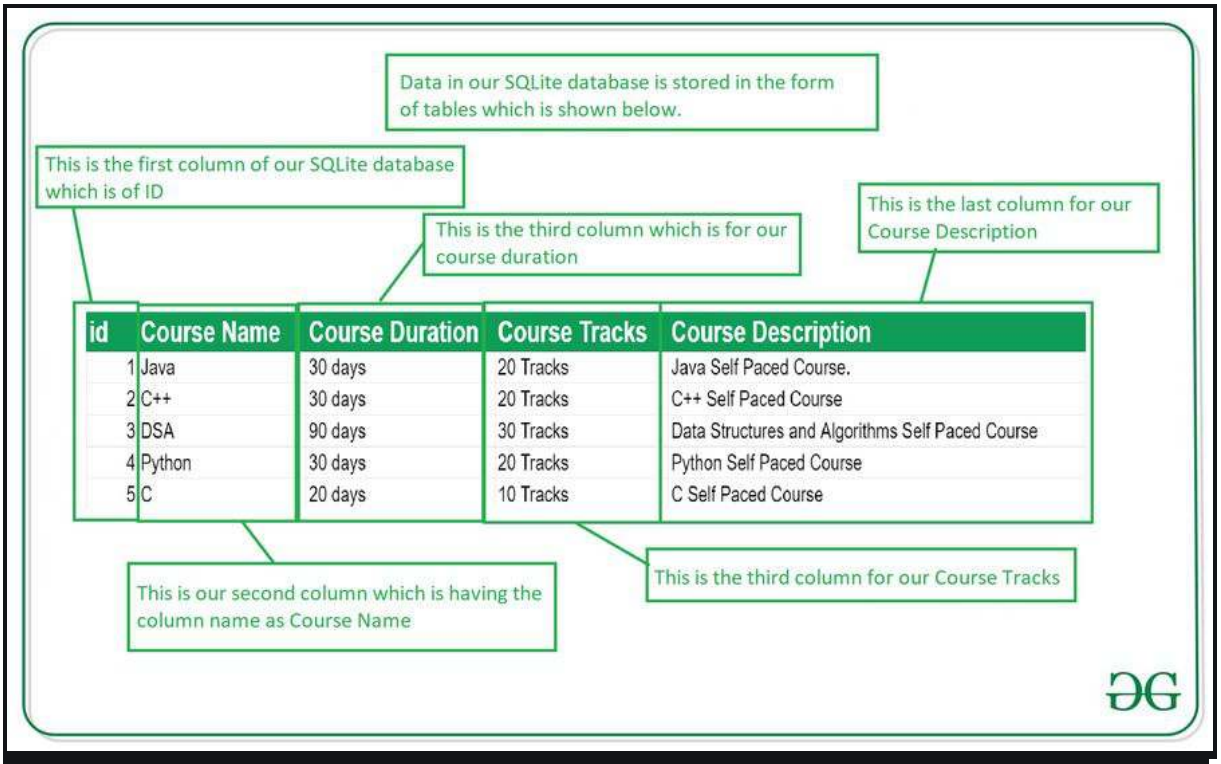
SQLite Database is an open-source database provided in Android which is used to store data inside the user's device in the form of a Text file. We can perform so many operations on this data such as adding new data, updating, reading, and deleting this data. SQLite is an offline database that is locally stored in the user's device and we do not have to create any connection to connect to this database.

What is SQLite Database?

SQLite Database is an open-source database provided in Android which is used to store data inside the user's device in the form of a Text file. We can perform so many operations on this data such as adding new data, updating, reading, and deleting this data. SQLite is an offline database that is locally stored in the user's device and we do not have to create any connection to connect to this database.

How Data is Being Stored in the SQLite Database?

Data is stored in the SQLite database in the form of **tables**. When we stored this data in our SQLite database it is arranged in the form of tables that are similar to that of an excel sheet. Below is the representation of our SQLite database which we are storing in our SQLite database.



Important Methods in SQLite Database

Below are the several important methods that we will be using in this SQLite database integration in Android.

Method	Description
getColumnNames()	This method is used to get the Array of column names of our SQLite table.
getCount()	This method will return the number of rows in the cursor.

isClosed()	This method returns a Boolean value when our cursor is closed.
getColumnCount()	This method returns the total number of columns present in our table.
getColumnName(int columnIndex)	This method will return the name of the column when we passed the index of our column in it.
getColumnIndex(String columnName)	This method will return the index of our column from the name of the column.
getPosition()	This method will return the current position of our cursor in our table.

What we are going to build in this article?

We will be building a simple application in which we will be adding data to the SQLite database. We will be creating a database for adding course name, course description, course duration, and course tracks. We will be saving all this data in our SQLite database. A sample video is given below to get an idea about what we are going to do in this article. Note that we are going to implement this project using the **Java** language.

Video Player

00:00
00:30

Step by Step Implementation

Step 1: Create a New Project

To create a new project in Android Studio please refer to How to Create/Start a New Project in Android Studio. Note that select **Java** as the programming language.

Step 2: Adding permissions to access the storage in the AndroidManifest.xml file

Navigate to the **app > AndroidManifest.xml** and add the below code to it.

- XML

```
<uses-permission
android:name="android.permission.READ_EXTERNAL_STORAGE" />
```

Step 3: Working with the activity_main.xml file

Navigate to the **app > res > layout > activity_main.xml** and add the below code to that file. Below is the code for the **activity_main.xml** file.

- XML

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout

xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <!--Edit text to enter course name-->
    <EditText
        android:id="@+id/idEdtCourseName"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:hint="Enter course Name" />

    <!--edit text to enter course duration-->
    <EditText
        android:id="@+id/idEdtCourseDuration"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:hint="Enter Course Duration" />

    <!--edit text to display course tracks-->
    <EditText
        android:id="@+id/idEdtCourseTracks"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:hint="Enter Course Tracks" />

    <!--edit text for course description-->
    <EditText
        android:id="@+id/idEdtCourseDescription"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:hint="Enter Course Description" />
```

```
    <!--button for adding new course-->
    <Button
        android:id="@+id/idBtnAddCourse"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="10dp"
        android:text="Add Course"
        android:textAllCaps="false" />

</LinearLayout>
```

Step 4: Creating a new Java class for performing SQLite operations

Navigate to the **app > java > your app's package name > Right-click on it > New > Java class** and name it as **DBHandler** and add the below code to it. Comments are added inside the code to understand the code in more detail.

- Java

```
import android.content.ContentValues;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class DBHandler extends SQLiteOpenHelper {

    // creating a constant variables for our database.
    // below variable is for our database name.
    private static final String DB_NAME = "coursedb";

    // below int is our database version
    private static final int DB_VERSION = 1;

    // below variable is for our table name.
    private static final String TABLE_NAME = "mycourses";

    // below variable is for our id column.
    private static final String ID_COL = "id";

    // below variable is for our course name column
    private static final String NAME_COL = "name";

    // below variable id for our course duration column.
    private static final String DURATION_COL = "duration";

    // below variable for our course description column.
    private static final String DESCRIPTION_COL =
"description";

    // below variable is for our course tracks column.
    private static final String TRACKS_COL = "tracks";

    // creating a constructor for our database handler.
    public DBHandler(Context context) {
        super(context, DB_NAME, null, DB_VERSION);
    }

    // below method is for creating a database by running
a sqlite query
    @Override
```

```
    public void onCreate(SQLiteDatabase db) {
        // on below line we are creating
        // an sqlite query and we are
        // setting our column names
        // along with their data types.
        String query = "CREATE TABLE " + TABLE_NAME + " ("
                + ID_COL + " INTEGER PRIMARY KEY
AUTOINCREMENT, "
                + NAME_COL + " TEXT,"
                + DURATION_COL + " TEXT,"
                + DESCRIPTION_COL + " TEXT,"
                + TRACKS_COL + " TEXT)";

        // at last we are calling a exec sql
        // method to execute above sql query
        db.execSQL(query);
    }

    // this method is use to add new course to our sqlite
database.
    public void addNewCourse(String courseName, String
courseDuration, String courseDescription, String
courseTracks) {

        // on below line we are creating a variable for
        // our sqlite database and calling writable method
        // as we are writing data in our database.
        SQLiteDatabase db = this.getWritableDatabase();

        // on below line we are creating a
        // variable for content values.
        ContentValues values = new ContentValues();

        // on below line we are passing all values
        // along with its key and value pair.
        values.put(NAME_COL, courseName);
        values.put(DURATION_COL, courseDuration);
        values.put(DESCRIPTION_COL, courseDescription);
        values.put(TRACKS_COL, courseTracks);

        // after adding all values we are passing
```

```
        // content values to our table.
        db.insert(TABLE_NAME, null, values);

        // at last we are closing our
        // database after adding database.
        db.close();
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int
oldVersion, int newVersion) {
        // this method is called to check if the table
exists already.
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
    }
}
```

Step 5: Working with the MainActivity.java file

Go to the **MainActivity.java** file and refer to the following code. Below is the code for the **MainActivity.java** file. Comments are added inside the code to understand the code in more detail.

- Java

```
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

import androidx.appcompat.app.AppCompatActivity;

public class MainActivity extends AppCompatActivity {

    // creating variables for our edittext, button and
dbhandler
    private EditText courseNameEdt, courseTracksEdt,
courseDurationEdt, courseDescriptionEdt;
    private Button addCourseBtn;
    private DBHandler dbHandler;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        // initializing all our variables.
        courseNameEdt =
findViewById(R.id.idEdtCourseName);
        courseTracksEdt =
findViewById(R.id.idEdtCourseTracks);
        courseDurationEdt =
findViewById(R.id.idEdtCourseDuration);
        courseDescriptionEdt =
findViewById(R.id.idEdtCourseDescription);
        addCourseBtn = findViewById(R.id.idBtnAddCourse);

        // creating a new dbhandler class
        // and passing our context to it.
        dbHandler = new DBHandler(MainActivity.this);

        // below line is to add on click listener for our
add course button.
        addCourseBtn.setOnClickListener(new
View.OnClickListener() {
```

```
            @Override
            public void onClick(View v) {

                // below line is to get data from all edit
text fields.
                String courseName =
courseNameEdt.getText().toString();
                String courseTracks =
courseTracksEdt.getText().toString();
                String courseDuration =
courseDurationEdt.getText().toString();
                String courseDescription =
courseDescriptionEdt.getText().toString();

                // validating if the text fields are empty
or not.
                if (courseName.isEmpty() &&
courseTracks.isEmpty() && courseDuration.isEmpty() &&
courseDescription.isEmpty()) {
                    Toast.makeText(MainActivity.this,
"Please enter all the data..", Toast.LENGTH_SHORT).show();
                    return;
                }

                // on below line we are calling a method
to add new
                // course to sqlite data and pass all our
values to it.
                dbHandler.addNewCourse(courseName,
courseDuration, courseDescription, courseTracks);

                // after adding the data we are displaying
a toast message.
                Toast.makeText(MainActivity.this, "Course
has been added.", Toast.LENGTH_SHORT).show();
                courseNameEdt.setText("");
                courseDurationEdt.setText("");
                courseTracksEdt.setText("");
                courseDescriptionEdt.setText("");
            }
        });
```

```
        }
}
```

CRUD

Data from the app can be saved on users' devices in different ways. We can store data in the user's device in SQLite tables, shared preferences, and many more ways. In this article, we will take a look at **saving data, reading, updating, and deleting data in Room Database** on Android. We will perform CRUD operations using Room Database on Android. In this article, we will take a look at performing CRUD operations in Room Database in Android.

As the heading tells you here, we are going to learn the CRUD operation in SQLite Database.

But what is CRUD? CRUD is nothing but an abbreviation for the basic operations that we perform in any database. And the operations are

- **C**reate
- **R**ead
- **U**pdate
- **D**elete

Android SQLite Database Example App Apk

- Before moving ahead on this tutorial if you want to know what we will be building, you can get the final apk of this tutorial from the link given below.

Android SQLite Database Example App Apk Download

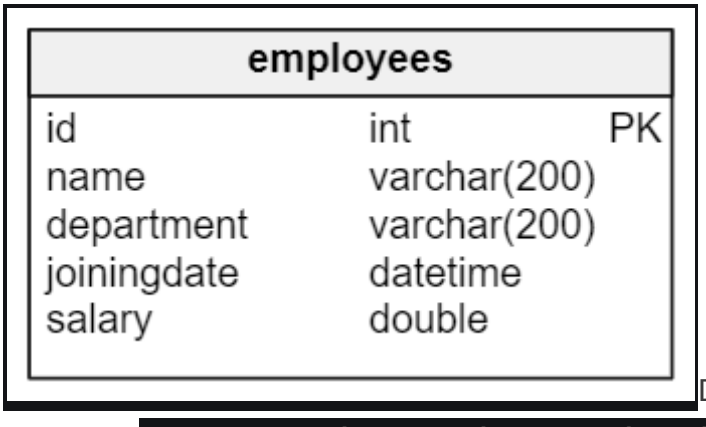
Android SQLite Database Example

Creating a new Android Studio Project

- As always we will create a new Android Studio Project. For this example, I have a new project named **SQLiteCRUDExample**.
- Once your project is loaded, we can start working on it.

The Database Structure

- The first thing needed is the database structure. We create database structure according to the system. But here we are not building an application, and it is only an example demonstrating the use of SQLite Database. So for this, I will use the following table structure.



Database Structure

- Now we have only a single table, but in real-world scenarios, you will have multiple tables with some complex relationships. Also, remember one thing whenever you create a table create a column named id with int as **PRIMARY KEY and AUTOINCREMENT. (If you**

are confused don't worry we will see now how do we create tables in database using SQL).

SQL Queries

- Now let's see how we can create the above table in our SQLite database.

Creating the Table

```

EATE TABLE employees (
  id INTEGER NOT NULL CONSTRAINT employees_pk PRIMARY KEY AUTOINCREMENT,
  name varchar(200) NOT NULL,
  department varchar(200) NOT NULL,
  joiningdate datetime NOT NULL,
  salary double NOT NULL
)

```

Creating a new Record

```

SERT INTO employees
ame, department, joiningdate, salary)
LUES
Belal Khan', 'Technical', '2017-09-30 10:00:00', '40000');

```

Reading All Existing Records

```

LECT * FROM employees;

```

Reading Specific Record

```
1
2  LECT * FROM employees WHERE id = 1;
3
```

Note: * means selecting all the columns, if you want a specific column or multiple columns but not all you can write names of the columns like **SELECT name, department.**

Updating a Record

```
1
2  DATE employees
3  T
4  me = 'Belal Haque',
5  partment = 'Research and Development',
6  lary = '100000'
7  ERE id = 1;
8
```

Deleting a Record

```
1
2  LETE FROM employees WHERE id = 1;
3
```

These are just some simple basics operations, but we can perform many tasks in our database. For this, you need to learn SQL in detail.

Note: SQL Queries are not case sensitive.

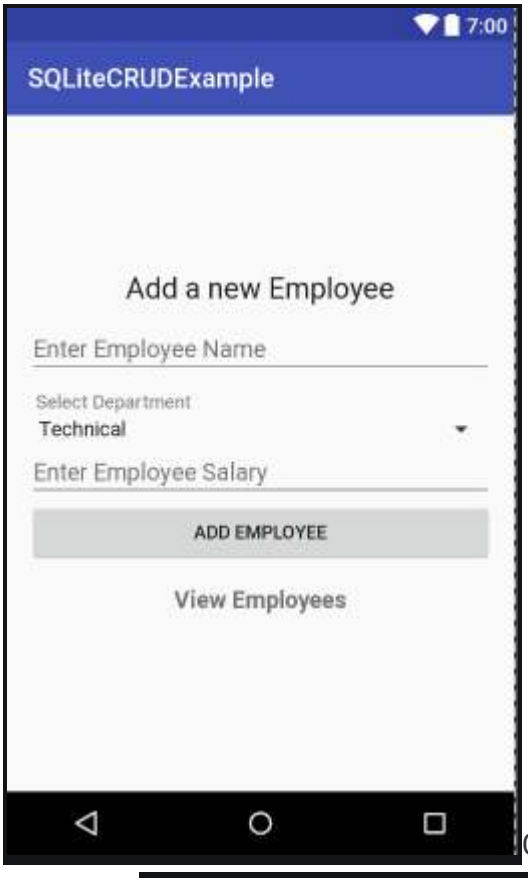
User Interface Design

- To implement all the above-given queries in our application, we need an Interface from where we can accomplish these tasks. Now, lets think about the

screens that we need to make an app that will perform all the above-given queries with user interaction.

Adding a new Employee

- The first thing is adding a new record to our database, and for this, we can use the following screen.



Creating Record

- As you can see we have **EditText, Button, Spinner and some TextViews.** To create the above interface, you can use the following XML code. You need to paste the following code inside activity_main.xml which is generated by default in any project because this will be the first screen for our application.

```
1
2   ml version="1.0" encoding="utf-8"?>
3   lativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
4    xmlns:app="http://schemas.android.com/apk/res-auto"
5    xmlns:tools="http://schemas.android.com/tools"
6    android:layout_width="match_parent"
7    android:layout_height="match_parent"
8    tools:context="net.simplifiedlearning.sqlitecrudexample.MainActivity">
9
10
11   <LinearLayout
12       android:layout_width="match_parent"
13       android:layout_height="wrap_content"
14       android:layout_centerVertical="true"
15       android:orientation="vertical"
16       android:padding="16dp">
17
18       <TextView
19           android:layout_width="match_parent"
20           android:layout_height="wrap_content"
21           android:layout_marginBottom="12dp"
22           android:text="Add a new Employee"
23           android:textAlignment="center"
24           android:textAppearance="@style/Base.TextAppearance.AppCompat.Large"
25
26
27       <EditText
28           android:id="@+id/editTextName"
29           android:layout_width="match_parent"
30           android:layout_height="wrap_content"
31           android:hint="Enter Employee Name" />
32
33       <TextView
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="10dp"
            android:paddingLeft="6dp"
            android:text="Select Department" />

        <Spinner
            android:id="@+id/spinnerDepartment"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:entries="@array/departments" />

        <EditText
            android:id="@+id/editTextSalary"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:digits="0123456789"
            android:hint="Enter Employee Salary"
            android:inputType="number" />

        <Button
            android:id="@+id/buttonAddEmployee"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:text="Add Employee" />

        <TextView
            android:id="@+id/textViewViewEmployees"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:padding="16dp"
            android:text="View Employees"
            android:textAlignment="center"
```

```
            android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"
            android:textStyle="bold" />

    </LinearLayout>

elativeLayout>
```

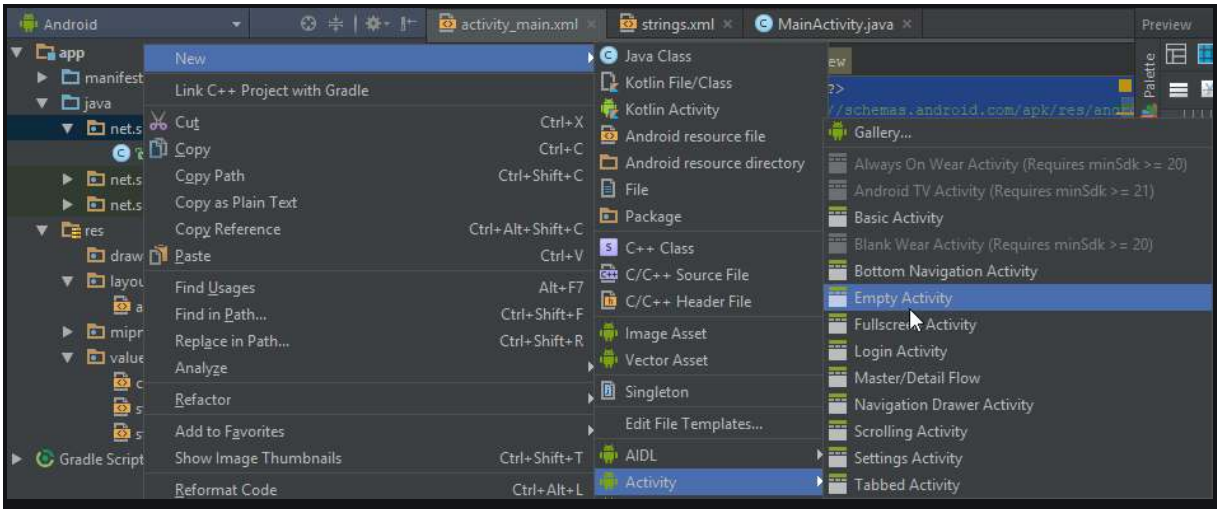
- For the spinner that we used in the above screen, we need to define an Array as the entries for the spinner. So go inside **values->strings.xml** and modify it as below.

```

sources>
    <string name="app_name">SQLiteCRUDExample</string>

    <array name="departments">
        <item>Technical</item>
        <item>Support</item>
        <item>Research and Development</item>
        <item>Marketing</item>
        <item>Human Resource</item>
    </array>
esources>

```

Fetching All the Employees

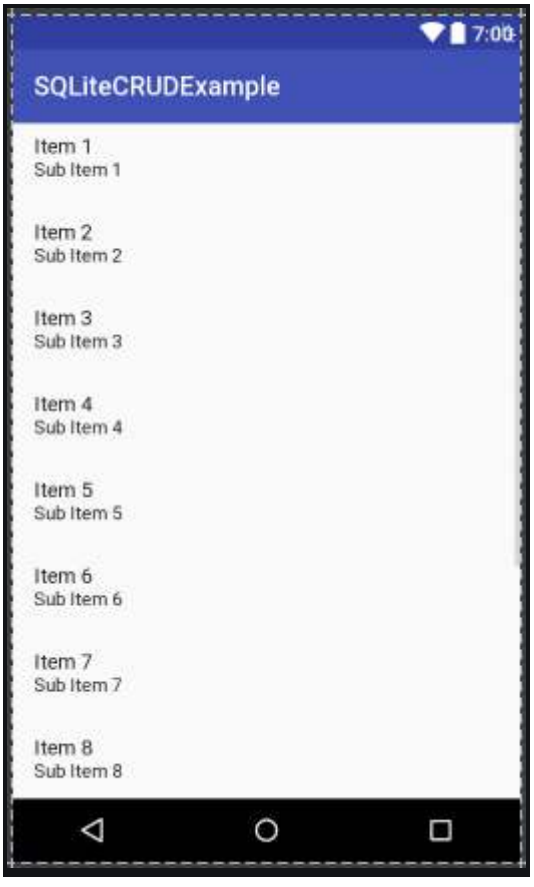
- Now after storing employee to the database, we also need to see all the stored employee from the database. For this, we can use a ListView.
- So, to create a new EmptyActivity in your project named **EmployeeActivity.** It will create a java file named

EmployeeActivity.java and a layout file called **activity_employee.xml**.



Creating an Empty Activity

- For this screen we can use the following design.



- This screen contains a only a **ListView**. The xml for the above screen is below

RESTful Web Services are basically REST Architecture based Web Services. In REST Architecture everything is a resource. RESTful web services are light weight, highly scalable and maintainable and are very commonly used to create APIs for web-based applications. This tutorial will teach you the basics of RESTful Web Services and contains chapters discussing all the basic components of RESTful Web Services with suitable examples.

Audience

This tutorial is designed for Software Professionals who are willing to learn RESTful Web Services in simple and easy steps. This tutorial will give you great understanding on RESTful Web Services concepts and after completing this tutorial you will be at intermediate level of expertise from where you can take yourself at higher level of expertise.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of Java Language, Text Editor, etc. Because we are going to develop web services applications using RESTful, so it will be good if you have understanding on other web technologies like HTML, CSS, AJAX, etc.

JSON or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange. The JSON format was originally

JSON

specified by Douglas Crockford, and is described in RFC 4627. The official Internet media type for JSON is application/json. The JSON filename extension is .json. This tutorial will help you understand JSON and its use within various programming languages such as PHP, PERL, Python, Ruby, Java, etc.

Audience

This tutorial has been designed to help beginners understand the basic functionality of JavaScript Object Notation (JSON) to develop the data interchange format. After completing this tutorial, you will have a good understanding of JSON and how to use it with JavaScript, Ajax, Perl, etc.

Prerequisites

Before proceeding with this tutorial, you should have a basic understanding of the web application's work over HTTP and we assume that you have a basic knowledge of JavaScript

Learn about Google Play services

Google Play services is core system software that enables key functionality on every certified Android device. There are three types of core device features Google Play services provides:

Security and reliability

Google Play services helps to ensure the security and reliability of an Android device, and keep devices updated with the latest security features. This includes:

- Google Play Protect, which can warn users if an app contains known malware

- Identification and validation of secure connections, such as allowing a device to safely and automatically recognize and connect to other devices, or share files or apps with Android devices nearby
- Protection of apps from fraud and security threats through SafetyNet
- End to end encrypted backup of your data when users have a lock screen passcode
- Management and protection of your passwords

Developer APIs

Google Play services provides developers thousands of continually updated APIs that enable them to deliver high-quality experiences in their apps, such as:

- Streaming media using Google Cast
- Integrating Google Maps to enhance app functionality
- Providing accurate location information through the fused location provider when apps have permission to access location
- Enabling services that allow developers to build advertising constructs according to user and app settings
- Sending timely notifications via a messaging transport layer

Core device services

Google Play services enables core services on Android devices. For example:

- When users make an emergency call to a supported emergency number, Google helps local emergency services directly receive the device's location
- Google's autofill services help users save time and reduce typing errors
- Nearby Share allows users to send and receive files with their contacts or anonymously
- Find My Device makes it easy to locate, lock, or wipe a lost device
- Fast Pair makes it easy to connect Bluetooth accessories using your Google account

Additionally, when a user signs into their Google account on their device, they are able to update their Google settings, manage the security of their account, and sync important data, such as their Google Contacts, across devices.

Why Google Play services collects data

Google Play services collects data on certified Android devices to support core device features. Collection of limited basic information, such as an IP address, is necessary to deliver content to a device, app, or browser. Device manufacturers also provide Google Play services with permission to access certain data on a device, such as location and contacts, to support these features.

Actual data collection varies depending on device settings configured by a user, the apps and services installed or used on a device, the device manufacturer, and a user's Google account settings. In many instances, Google Play services will access data locally on the device without collecting data off the device.

To support each of the functions described above, Google Play services may collect information for the following reasons:

Security and fraud prevention

Google collects data through Google Play services to help protect users, Google services, and third party developers' apps and services from fraud, spam, and abuse. This includes:

- Information to validate that a request is coming from a real user and information about installed apps, including the results of malware scans.
- Google Account and login information if a user is signed in on a device or moves their data to a new device.
- Google may collect a device's phone number to provide account recovery services and to log users into phone number based services (like Google Duo).
- Hardware identifiers such as IMEI, MAC addresses, and serial numbers, to update devices with the latest security patches and to monitor trends across the Android ecosystem, such as how long different types of devices stay in service. Google's Device Configuration Service, which collects data to ensure that devices remain up-to-date and are working as well as possible, is part of Google Play services.

Support and improve the Android ecosystem

As described above, Google Play services provides a number of APIs and core device services that enable Android to be a feature-rich, connected platform. Google may collect data about these services and APIs to help provide, maintain and improve them. Depending on device settings, Google may collect additional information about a device. Examples include:

- Google collects data to understand how these APIs are used and to help ensure that they function correctly.
- If Google Location Accuracy is enabled, in addition to providing more accurate location on a device, location information may be used in an anonymous way to improve location-based services.
- If a device's usage and diagnostics control is enabled, Google collects information about device usage and how well a device is working to improve products and services, like Google apps and Android devices.

Provide Google services

If a user uses Google apps and services on Android, Google collects data through Google Play services to provide and improve those apps and services. For example:

- Depending on a user's settings, Google collects data like contacts and bookmarks to sync them across devices and the cloud.
- Google Play services syncs a user's Google account settings across devices, and collects information to help protect their account.
- Google Play services may collect data to enable embedded app functionality like Google Maps.
- Google Play services are used to enable the COVID-19 Exposure Notifications system.
- Google Play services help users interact and send messages directly to businesses.
- When using Google Pay, Google Play services helps users manage their payment info, make contactless payments, or use a digital car key securely.

Location vased service

This becomes possible with the help of Google Play services, which facilitates adding location awareness to your app with automated location tracking, geofencing, and activity recognition.

This tutorial shows you how to use Location Services in your APP to get the current location, get periodic location updates, look up addresses etc.

The Location Object

The Location object represents a geographic location which can consist of a latitude, longitude, time stamp, and other information such as bearing, altitude and velocity. There are following important methods which you can use with Location object to get location specific information –

Sr.No.	Method & Description
1	float distanceTo(Location dest) Returns the approximate distance in meters between this location and the given location.
2	float getAccuracy() Get the estimated accuracy of this location, in meters.
3	double getAltitude() Get the altitude if available, in meters above sea level.

4	float getBearing() Get the bearing, in degrees.
5	double getLatitude() Get the latitude, in degrees.
6	double getLongitude() Get the longitude, in degrees.
7	float getSpeed() Get the speed if it is available, in meters/second over ground.
8	boolean hasAccuracy() True if this location has an accuracy.
9	boolean hasAltitude() True if this location has an altitude.
10	boolean hasBearing() True if this location has a bearing.
11	boolean hasSpeed() True if this location has a speed.

12	void reset() Clears the contents of the location.
13	void setAccuracy(float accuracy) Set the estimated accuracy of this location, meters.
14	void setAltitude(double altitude) Set the altitude, in meters above sea level.
15	void setBearing(float bearing) Set the bearing, in degrees.
16	void setLatitude(double latitude) Set the latitude, in degrees.
17	void setLongitude(double longitude) Set the longitude, in degrees.
18	void setSpeed(float speed) Set the speed, in meters/second over ground.
19	String toString()

	Returns a string containing a concise, human-readable description of this object.

Get the Current Location

To get the current location, create a location client which is LocationClient object, connect it to Location Services using connect() method, and then call its getLastLocation() method. This method returns the most recent location in the form of Location object that contains latitude and longitude coordinates and other information as explained above. To have location based functionality in your activity, you will have to implement two interfaces –

- GooglePlayServicesClient.ConnectionCallbacks
- GooglePlayServicesClient.OnConnectionFailedListener

These interfaces provide following important callback methods, which you need to implement in your activity class –

Sr.No.	Callback Methods & Description
1	abstract void onConnected(Bundle connectionHint) This callback method is called when location service is connected to the location client successfully. You will use connect() method to connect to the location client.

2	abstract void onDisconnected() This callback method is called when the client is disconnected. You will use disconnect() method to disconnect from the location client.
3	abstract void onConnectionFailed(ConnectionResult result) This callback method is called when there was an error connecting the client to the service.

You should create the location client in onCreate() method of your activity class, then connect it in onStart(), so that Location Services maintains the current location while your activity is fully visible. You should disconnect the client in onStop() method, so that when your app is not visible, Location Services is not maintaining the current location. This helps in saving battery power up-to a large extent.

Get the Updated Location

If you are willing to have location updates, then apart from above mentioned interfaces, you will need to implement LocationListener interface as well. This interface provide following callback method, which you need to implement in your activity class –

Sr.No	Callback Method & Description
1	abstract void onLocationChanged(Location location) This callback method is used for receiving notifications from the LocationClient when the location has changed.

Location Quality of Service

The LocationRequest object is used to request a quality of service (QoS) for location updates from the LocationClient. There are following useful setter methods which you can use to handle QoS. There are equivalent getter methods available which you can check in Android official documentation.

Sr.No	Method & Description
1	setExpirationDuration(long millis) Set the duration of this request, in milliseconds.
2	setExpirationTime(long millis) Set the request expiration time, in millisecond since boot.
3	setFastestInterval(long millis) Explicitly set the fastest interval for location updates, in milliseconds.
4	setInterval(long millis) Set the desired interval for active location updates, in milliseconds.
5	setNumUpdates(int numUpdates) Set the number of location updates.
6	setPriority(int priority) Set the priority of the request.

Now for example, if your application wants high accuracy location it should create a location request with setPriority(int) set to PRIORITY_HIGH_ACCURACY and setInterval(long) to 5 seconds. You can also use bigger interval and/or other priorities like PRIORITY_LOW_POWER for to request "city" level accuracy or PRIORITY_BALANCED_POWER_ACCURACY for "block" level accuracy.

Activities should strongly consider removing all location request when entering the background (for example at onPause()), or at least swap the request to a larger interval and lower quality to save power consumption.

Displaying a Location Address

Once you have Location object, you can use Geocoder.getFromLocation() method to get an address for a given latitude and longitude. This method is synchronous, and may take a long time to do its work, so you should call the method from the doInBackground() method of an AsyncTask class.

The AsyncTask must be subclassed to be used and the subclass will override doInBackground(Params...) method to perform a task in the background and onPostExecute(Result) method is invoked on the UI thread after the background computation finishes and at the time to display the result. There is one more important method available in AyncTask wh