UNIT-1

Introduction of Mobile Applications

1.1 Introduction

Talking about the mobile applications, the first thing that comes to mind are the apps like WhatsApp, Instagram, swigy, etc that we use in our everyday life.

Ever thought about how these apps are made? Which technology is used? Let's discuss what technologies or frameworks can be used to develop a mobile application.

Mobile Apps are majorly developed for 3 Operating System:

- Android
- IOS
- Windows

There are 3 different ways to develop Mobile apps: –

1. **1st Party Native App development**: These types of apps normally run in the native devices, that is, it runs only in the OS that it is specifically designed for it. These apps cannot be used on different devices using a different OS. The apps that are developed for android are normally coded using Java or Kotlin languages. The IDE normally used for android app development is Android Studio which provides all features and the apps that are developed for IOS are generally coded in Swift language or Objective-C.

   The IDE suggested for IOS App Development is XCode.

   Example:

   Here's an example of a 1st party native app:

   A retail company wants to improve the in-store shopping experience for its customers. They develop a 1st party native app that allows customers to:

   - Browse the store's inventory and product information
   - Create a shopping list
   - Scan barcodes to view product information and reviews
   - Locate items in the store using an interactive map
   - Pay for items directly through the app, without having to wait in line at the register
   - The app is only available to the company's customers and can only be used in their physical stores. The app is designed to integrate with the company's existing systems, such as inventory management and point-of-sale systems.
   - This app is developed by the retail company for their own use, to improve the in-store customer experience, increase sales and gain insights from the customer's behaviour.

In this example, the retail company is the 1st party, and the app is a native app, because it is developed for the specific platform (iOS or Android) and can take full advantage of the device's capabilities and features.

2. **Progressive web Application:** Progressive web apps are essentially a website which runs locally on your device. The technologies used are Microsoft Blazor, React, Angular JS, Native Script, Iconic. These technologies normally used for web development propose. The apps' UI is developed the same way as they are developed while developing the website. This category has many ups and downs let's start with the advantages of Progressive web apps.

   Example:

   Here's an example of a Progressive Web App:

   A news website wants to provide its users with a better mobile experience. They develop a Progressive Web App that:

   - Allows users to access the website offline by storing content on the user's device
   - Sends push notifications to users to alert them of breaking news
   - Can be installed on the user's home screen like a native app
   - Provides a fast and smooth browsing experience
   - Has a responsive design that adapts to different screen sizes
   - Users can access the PWA by visiting the website on their mobile browser. They are prompted to install the PWA on their home screen, which allows them to access the website offline and receive push notifications.

   In this example, the news website is the 1st party and the app is a Progressive web app, because it can be accessed through a web browser and can be installed on the user's device like a native app. It also allows users to access the content offline and have a fast and smooth experience.

3. **Cross-Platform Application:** These are frameworks that allow developing total native applications which have access to all the native features of IOS and Android but with the same code base. These apps run on both Android and IOS. So normally the development speeds of these apps are very fast and the maintenance cost is low. The performance speed is comparatively low to 1st party native apps but faster than PWA.

   Xamarin is Microsoft cross-platform solution that uses the programming languages like .NET, C#, F#. The IDE preferred is Visual Studio. The UI/UX is totally native giving access to all features. This technology is having a wide community. And whenever an update is released by Android and IOS the same updates are released by Microsoft through Visual Studio.

   React Native is Facebook's cross-platform solution which uses the language JavaScript And the preferred IDE is WebStrome & Visual Studio Code. Same like Xamarin React Native has totally native UI/UX and gives access to all features. And the updates are released the same day by Facebook as Android and IOS.

   Flutter is Google's cross-platform solution which uses the language, Dart. The IDE preferred is Android Studio, IntelliJ IDE, and Visual Studio Code. The UI/UX is bespoke and Flutters has to come up with their new libraries whenever Android and IOS comes up with an update to mimic those update. The community is fast growing.

Example:

Here's an example of a cross-platform application:

A project management company wants to create a project management tool that can be used by teams on different platforms. They develop a cross-platform application that Can be used on Windows, Mac, iOS, and Android devices:

- Allows users to create and assign tasks, set deadlines, and track progress
- Integrates with popular tools such as Google Calendar and Trello
- Has a user-friendly interface that works seamlessly across all platforms
- The application can be downloaded from the company's website or from different app stores such as App Store, Google Play Store, Microsoft Store, and Mac App Store, depending on the platform.

This example illustrates how the company developed a project management tool that can be used by teams on different platforms, Windows, Mac, iOS and Android, which is a cross-platform application. It allows teams to collaborate and manage their projects seamlessly, regardless of the platform they use.

1.2 Mobile Development Framework

A mobile development framework is a software framework that is designed to support mobile app development. It is a software library that provides a fundamental structure to support the development of applications for a specific environment.

Frameworks can be in three categories: native frameworks for platform-specific development, mobile web app frameworks, and hybrid apps, which combine the features of both native and mobile web app frameworks.

Mobile App Development Framework is a library that offers the required fundamental structure to create mobile applications for a specific environment. In short, it acts as a layout to support mobile app development. There are various advantages of Mobile App Development frameworks such as *cost-effectiveness, efficiency*, and many more. Moreover, mobile application frameworks can be classified majorly into 3 categories: *Native Apps, Web Apps* & *Hybrid Apps*.

Before moving further, let's take a quick look at all three categories of mobile development apps.

- **Native Apps:** A Native App is an application specifically designed for a particular platform or device.

- **Web Apps:** A Web App is concerned with an application that is designed to deliver web pages on different web platforms for any device.
- **Hybrid Apps:** A Hybrid App is a combination of both native & web applications. It can be developed for any platform from a single code base.

1.2.1 React Native

React Native is one of the most recommended Mobile App Frameworks in the development industry. The framework, created by Facebook, is an open-source framework that offers you to develop mobile applications for Android & iOS platforms. The React Native framework is based on React and JavaScript that aims to develop native applications over hybrid applications that run on a web view. Moreover, it is a cross-platform development framework that uses a single code base for both Android & iOS applications. Some of the major benefits of React Native are mentioned below:

- Code Re-usability & Cost-Effective
- Compatible with third-party plugins
- Re-usable components for optimal performance
- Provides hot deployment features
- Ease of Maintenance

1.2.2 Flutter

Flutter, developed by **Google**, is a **UI toolkit** to build native applications for mobile apps, desktop & web platforms. Flutter is a **cross-platform mobile app development framework** that works on one code base to develop Android as well as iOS applications. The framework provides a large range of fully customizable widgets that helps to build native applications in a shorter span. Moreover, Flutter uses the **2D rendering engine called Skia** for developing visuals and its layered architecture ensures the effective functioning of components. Some of the major benefits of Flutter are mentioned below:

- Provides Full Native Performance
- Flexible User interface (UI)
- Provides Strong Widget Support
- Offers Built-in Material Design
- Fast Application Development

1.2.3 Ionic

Ionic, developed in **2013**, is an open-source framework that allows you to build cross-platform for mobile apps using web technologies like **HTML**, **CSS** & **JavaScript**. The application built through the Ionic framework can work on *Android*, *iOS* & *Windows* platforms. The framework offers numerous default UI components such as *forms*, *action sheets*, *filters*, *navigation menus*, and many more for attractive and worthwhile design. Moreover, Ionic has its **own command-line interface** and various other in-built features such as **Ionic Native**, **Cordova-Based App packages**, etc. Some of the major benefits of Ionic for mobile development apps are mentioned below:

- Faster Application Development.

- Availability of Cordova Plugins
- Built-in UI components
- Platform Independent
- Based on AngularJS

Android Framework(Platform)

Android is an open source, Linux-based software stack created for a wide array of devices and form factors. Figure shows the major components of the Android Framework/Platform.

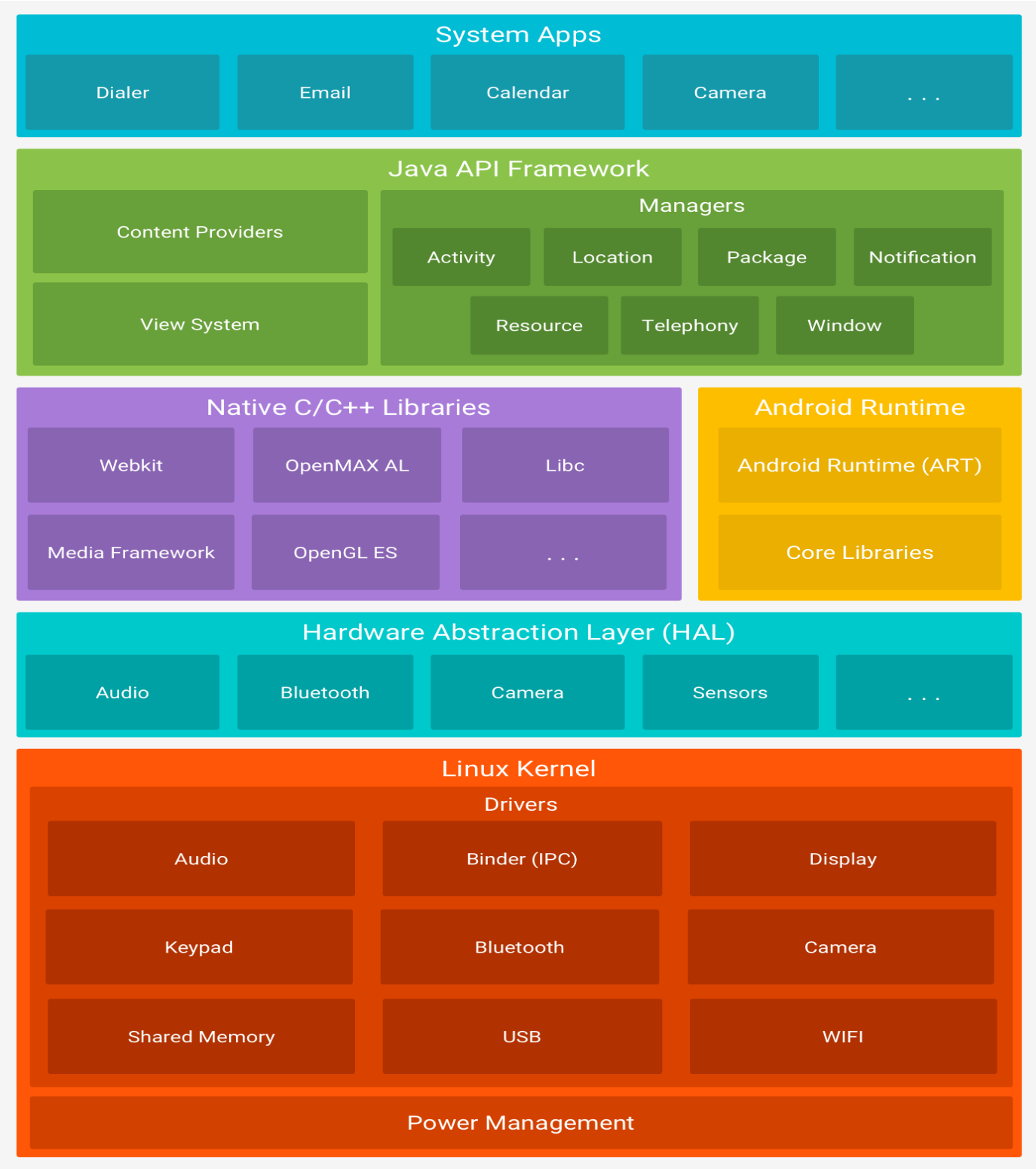


Figure. The Android software stack.

Linux kernel

The foundation of the Android platform is the Linux kernel. For example, the Android Runtime (ART) relies on the Linux kernel for underlying functionalities such as threading and low-level memory management.

Using a Linux kernel lets Android take advantage of key security features and lets device manufacturers develop hardware drivers for a well-known kernel.

Hardware abstraction layer (HAL)

The hardware abstraction layer (HAL) provides standard interfaces that expose device hardware capabilities to the higher-level Java API framework. The HAL consists of multiple library modules, each of which implements an interface for a specific type of hardware component, such as the camera or Bluetooth module. When a framework API makes a call to access device hardware, the Android system loads the library module for that hardware component.

Android runtime

For devices running Android version 5.0 (API level 21) or higher, each app runs in its own process and with its own instance of the Android Runtime (ART). ART is written to run multiple virtual machines on low-memory devices by executing Dalvik Executable format (DEX) files, a bytecode format designed specifically for Android that's optimized for a minimal memory footprint. Build tools, such as d8, compile Java sources into DEX bytecode, which can run on the Android platform.

Some of the major features of ART include the following:

- Ahead-of-time (AOT) and just-in-time (JIT) compilation
- Optimized garbage collection (GC)
- On Android 9 (API level 28) and higher, conversion of an app package's DEX files to more compact machine code
- Better debugging support, including a dedicated sampling profiler, detailed diagnostic exceptions and crash reporting, and the ability to set watchpoints to monitor specific fields

Prior to Android version 5.0 (API level 21), Dalvik was the Android runtime. If your app runs well on ART, then it can work on Dalvik as well, but the reverse might not be true.

Android also includes a set of core runtime libraries that provide most of the functionality of the Java programming language, including some Java 8 language features, that the Java API framework uses.

Native C/C++ libraries

Many core Android system components and services, such as ART and HAL, are built from native code that requires native libraries written in C and C++. The Android platform provides Java framework APIs to expose the functionality of some of these native libraries to apps. For example, you can access OpenGL ES through the Android framework's Java OpenGL API to add support for drawing and manipulating 2D and 3D graphics in your app.

If you are developing an app that requires C or C++ code, you can use the Android NDK to access some of these native platform libraries directly from your native code.

Java API framework

The entire feature-set of the Android OS is available to you through APIs written in the Java language. These APIs form the building blocks you need to create Android apps by simplifying the reuse of core, modular system components and services, which include the following:

- A rich and extensible view system you can use to build an app's UI, including lists, grids, text boxes, buttons, and even an embeddable web browser
- A resource manager, providing access to non-code resources such as localized strings, graphics, and layout files
- A notification manager that enables all apps to display custom alerts in the status bar
- An activity manager that manages the lifecycle of apps and provides a common navigation back stack
- Content providers that enable apps to access data from other apps, such as the Contacts app, or to share their own data

Developers have full access to the same framework APIs that Android system apps use.

System apps

Android comes with a set of core apps for email, SMS messaging, calendars, internet browsing, contacts, and more. Apps included with the platform have no special status among the apps the user chooses to install. So, a third-party app can become the user's default web browser, SMS messenger, or even the default keyboard. Some exceptions apply, such as the system's Settings app.

The system apps function both as apps for users and to provide key capabilities that developers can access from their own app. For example, if you want your app to deliver SMS messages, you don't need to build that functionality yourself. You can instead invoke whichever SMS app is already installed to deliver a message to the recipient you specify.

1.3 Components of Android Application

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 hardware 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.

```
class MainActivity : AppCompatActivity() {

}
```

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:

```
class ServiceName : Service() {

}
```

3. Content Providers:

It is used to manage and persist the application data 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 sub-class of the class ContentProvider.

```
class contentProviderName : ContentProvider() {

  override fun onCreate(): Boolean {}

}
```

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.

5. Intents:

It is a powerful inter-application message-passing framework. They are extensively used throughout Android. Intents can be used to start and stop Activities and Services, to broadcast messages system-wide or to an explicit Activity, Service or Broadcast Receiver or to request action be performed on a particular piece of data.

6. Widgets:

These are the small visual application components that you can find on the home screen of the devices. They are a special variation of Broadcast Receivers that allow us to create dynamic, interactive application components for users to embed on their Home Screen.

7. Notifications:

Notifications are the application alerts that are used to draw the user's attention to some particular app event without stealing focus or interrupting the current activity of the user. They are generally used to grab user's attention when the application is not visible or active, particularly from within a Service or Broadcast Receiver. Examples: E-mail popups, Messenger popups, etc.

1.4 Android Manifest File in Android

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 Receivers 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 top-level nodes can specify any required permissions, and unit tests, and define hardware, screen, or platform requirements. The manifest comprises a root manifest tag 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. We use 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.

We can also specify whether our app should install on an SD card of the internal memory using the installLocation attribute. A typical manifest file looks as:

Example: 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.application"
   android:versionCode="1"
   android:versionName="1.0"
   android:installLocation="preferExternal">

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

   <application
      android:allowBackup="true"
      android:dataExtractionRules="@xml/data_extraction_rules"
      android:fullBackupContent="@xml/backup_rules"
      android:icon="@mipmap/ic_launcher"
      android:label="@string/app_name"
```

```
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/Theme.MyApplication"
    tools:targetApi="31">
    <activity>
      android:name=".MainActivity"
      android:exported="true">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
      </intent-filter>
    </activity>
  </application>

</manifest>
```

A manifest file includes the nodes that define the application components, security settings, test classes, and requirements that make up the application. Some of the manifest sub-node tags that are mainly used are:

1. manifest

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

```
<?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.application">
  <!-- manifest nodes -->
  <application>
  </application>
</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.

```
<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.

```
<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.

```
<application
    android:name=".Application"
    android:allowBackup="true"
    android:dataExtractionRules="@xml/data_extraction_rules"
    android:fullBackupContent="@xml/backup_rules"
    android:icon="@drawable/gfgIcon"
    android:label="@string/app_name"
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@android:style/Theme.Light"
    android:debuggable="true"
    tools:targetApi="31">
    <!-- 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.

```
<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.

<activity

android:name=".MainActivity"

android:exported="true">

</activity>.

7. intent-filter

It is the sub-element of activity that specifies 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.

<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.

<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.

<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.

<uses-configuration

android:reqTouchScreen="finger"

android:reqNavigation="trackball"

android:reqHardKeyboard="true"

```
   android:reqKeyboardType=”qwerty”/>


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

11. uses-features

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

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

12. permission

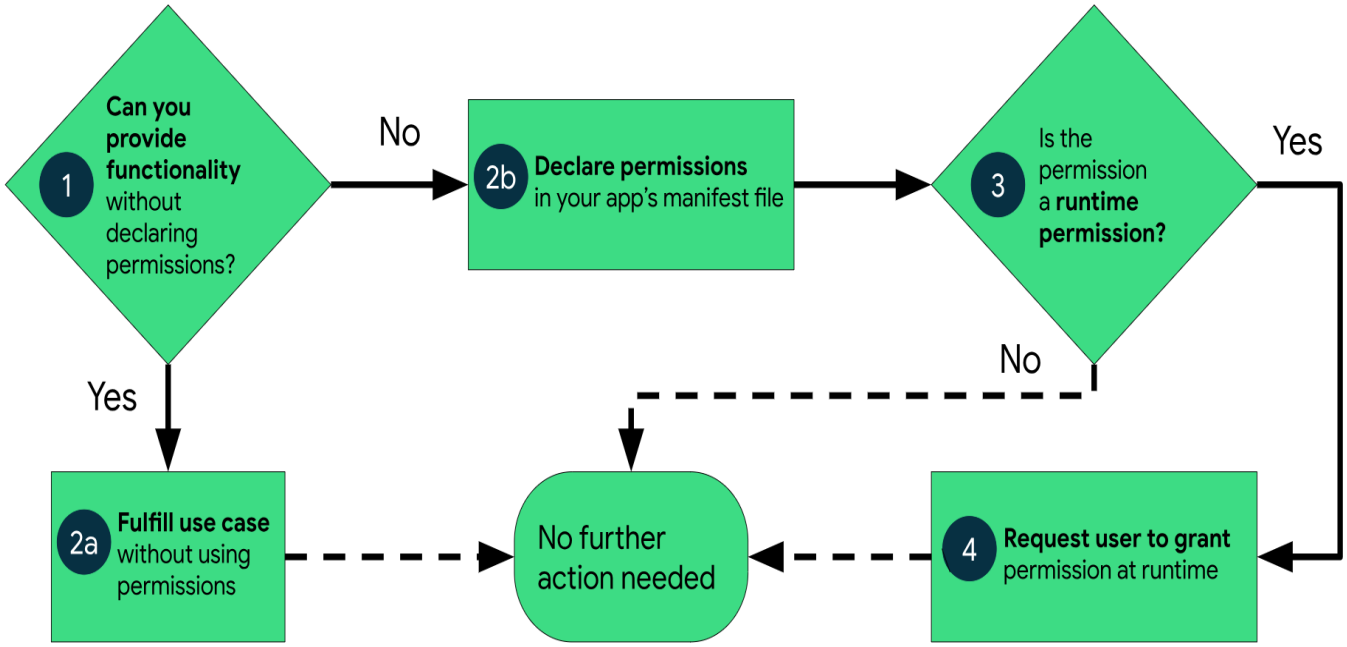
It is used to create permissions to restrict access to shared application components. We can also use the existing platform permissions for this purpose or define your own permissions in the manifest.

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

1.5. Permission Model of Android

If your app offers functionality that might require access to restricted data or restricted actions, determine whether you can get the information or perform the actions without needing to declare permissions. You can fulfill many use cases in your app, such as taking photos, pausing media playback, and displaying relevant ads, without needing to declare any permissions.

If you decide that your app must access restricted data or perform restricted actions to fulfill a use case, declare the appropriate permissions. Some permissions, known as install-time permissions, are automatically granted when your app is installed. Other permissions, known as runtime permissions, require your app to go a step further and request the permission at runtime.



Types of permission:

1. Install-time permissions

Install-time permissions give your app limited access to restricted data or let your app perform restricted actions that minimally affect the system or other apps. When you declare install-time permissions in your app, an app store presents an install-time permission notice to the user when they view an app's details page, as shown in figure 2. The system automatically grants your app the permissions when the user installs your app.

2.Normal permissions

These permissions allow access to data and actions that extend beyond your app's sandbox but present very little risk to the user's privacy and the operation of other apps.

3. Signature permissions

The system grants a signature permission to an app only when the app is signed by the same certificate as the app or the OS that defines the permission.

Applications that implement privileged services, such as autofill or VPN services, also make use of signature permissions. These apps require service-binding signature permissions so that only the system can bind to the services.

4. Runtime permissions

Runtime permissions, also known as dangerous permissions, give your app additional access to restricted data or let your app perform restricted actions that more substantially affect the system and other apps. Therefore, you need to request runtime permissions in your app before you can access the restricted data or perform restricted actions. Don't assume that these permissions have been previously granted—check them and, if needed, request them before each access.

When your app requests a runtime permission, the system presents a runtime permission prompt.

Many runtime permissions access *private user data*, a special type of restricted data that includes potentially sensitive information. Examples of private user data include location and contact information.

5.Special permissions

Special permissions correspond to particular app operations. Only the platform and OEMs can define special permissions. Additionally, the platform and OEMs usually define special permissions when they want to protect access to particularly powerful actions, such as drawing over other apps.

The Special app access page in system settings contains a set of user-toggleable operations. Many of these operations are implemented as special permissions.

6. Permission groups

Permissions can belong to permission groups. Permission groups consist of a set of logically related permissions. For example, permissions to send and receive SMS messages might belong to the same group, as they both relate to the application's interaction with SMS.

Permission groups help the system minimize the number of system dialogs that are presented to the user when an app requests closely related permissions. When a user is presented with a prompt to grant permissions for an application, permissions belonging to the same group are presented in the same interface. However, permissions can change groups without notice, so don't assume that a particular permission is grouped with any other permission.

1.6. Downloading and Installing SDK

Install the SDK

1. Click Tools > SDK Manager.
2. In the SDK Platforms tab, select Android UpsideDownCake Preview.
3. In the SDK Tools tab, select Android SDK Build-Tools 29.
4. Click OK to install the SDK.

1.7. Exploring the Development Environment

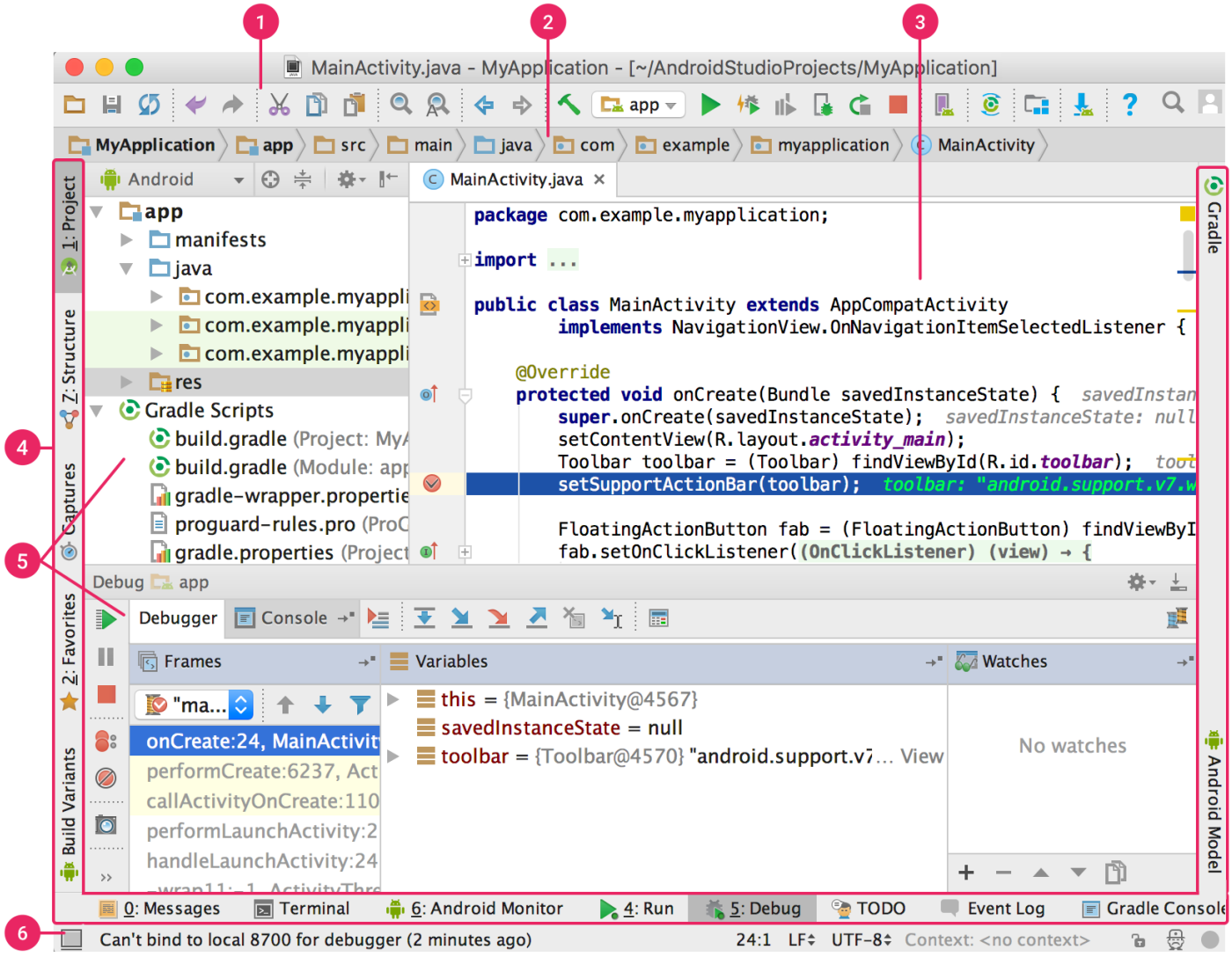
Android Studio is the official Integrated Development Environment (IDE) for Android app development.

Android Studio offers even more features that enhance your productivity when building Android apps, such as:

- A flexible Gradle-based build system
- A fast and feature-rich emulator
- A unified environment where you can develop for all Android devices
- Live Edit to update composables in emulators and physical devices in real time

- Code templates and GitHub integration to help you build common app features and import sample code
- Extensive testing tools and frameworks
- Lint tools to catch performance, usability, version compatibility, and other problems
- C++ and NDK support
- Built-in support for Google Cloud Platform, making it easy to integrate Google Cloud Messaging and App Engine.

Get to know the Android Studio UI



1. **Toolbar:** Carry out a wide range of actions, including running your app and launching Android tools.
2. **Navigation bar:** Navigate through your project and open files for editing. It provides a more compact view of the structure visible in the **Project** window.
3. **Editor window:** Create and modify code. Depending on the current file type, the editor can change. For example, when viewing a layout file, the editor displays the Layout Editor.
4. **Tool window bar:** Use the buttons on the outside of the IDE window to expand or collapse individual tool windows.

5. **Tool windows:** Access specific tasks like project management, search, version control, and more. You can expand them and collapse them.
6. **Status bar:** Display the status of your project and the IDE itself, as well as any warnings or messages.

Tool windows

To expand or collapse a tool window, click the tool's name in the tool window bar. You can also drag, pin, unpin, attach, and detach tool windows.

To return to the default layout of the current tool window, click Window > Restore Default Layout. To customize your default layout, click Window > Store Current Layout as Default.

To show or hide the entire tool window bar, click the window in the bottom left-hand corner of the Android Studio window.

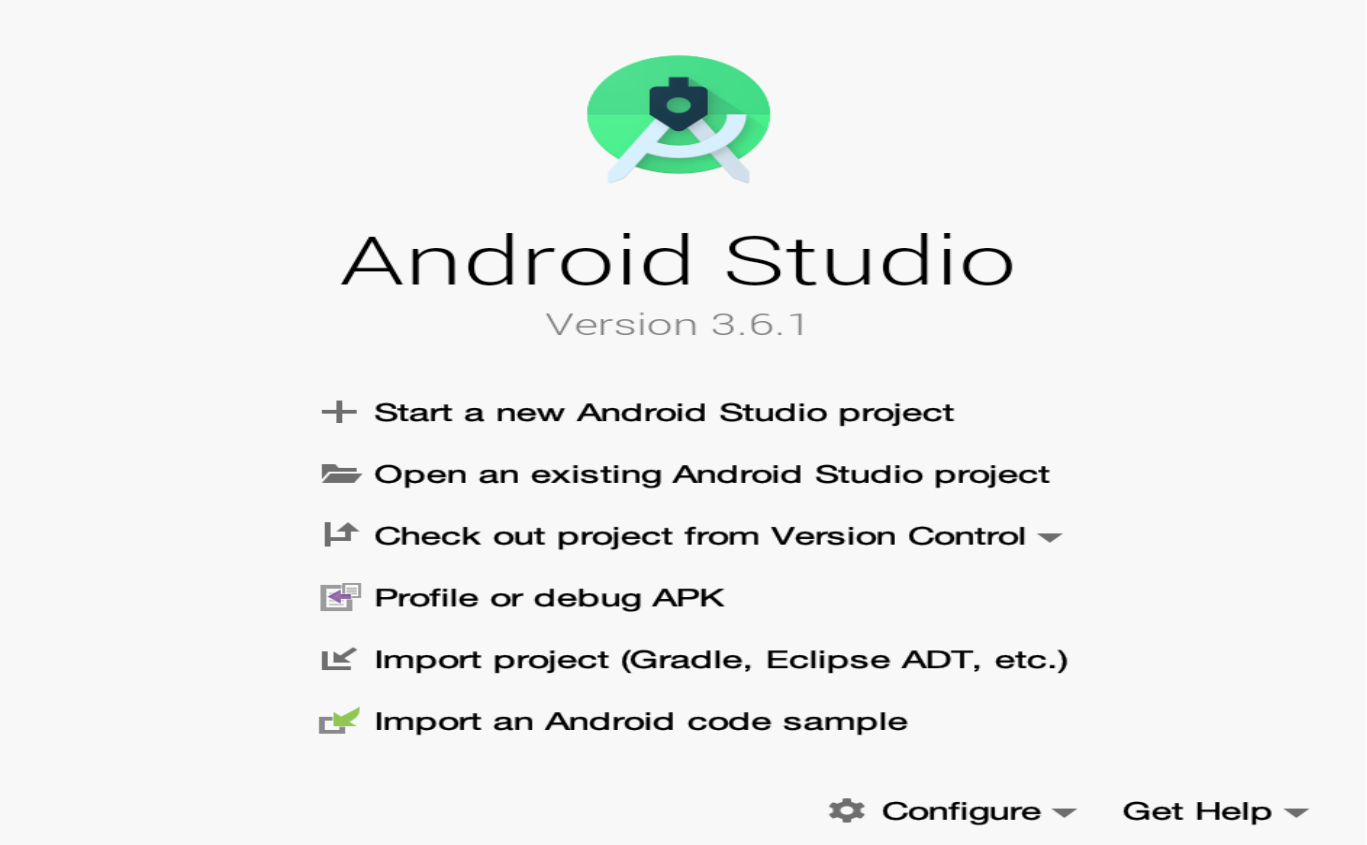
To locate a specific tool window, hover over the window icon and select the tool window from the menu.

1.8 Build Your First Android App in Kotlin

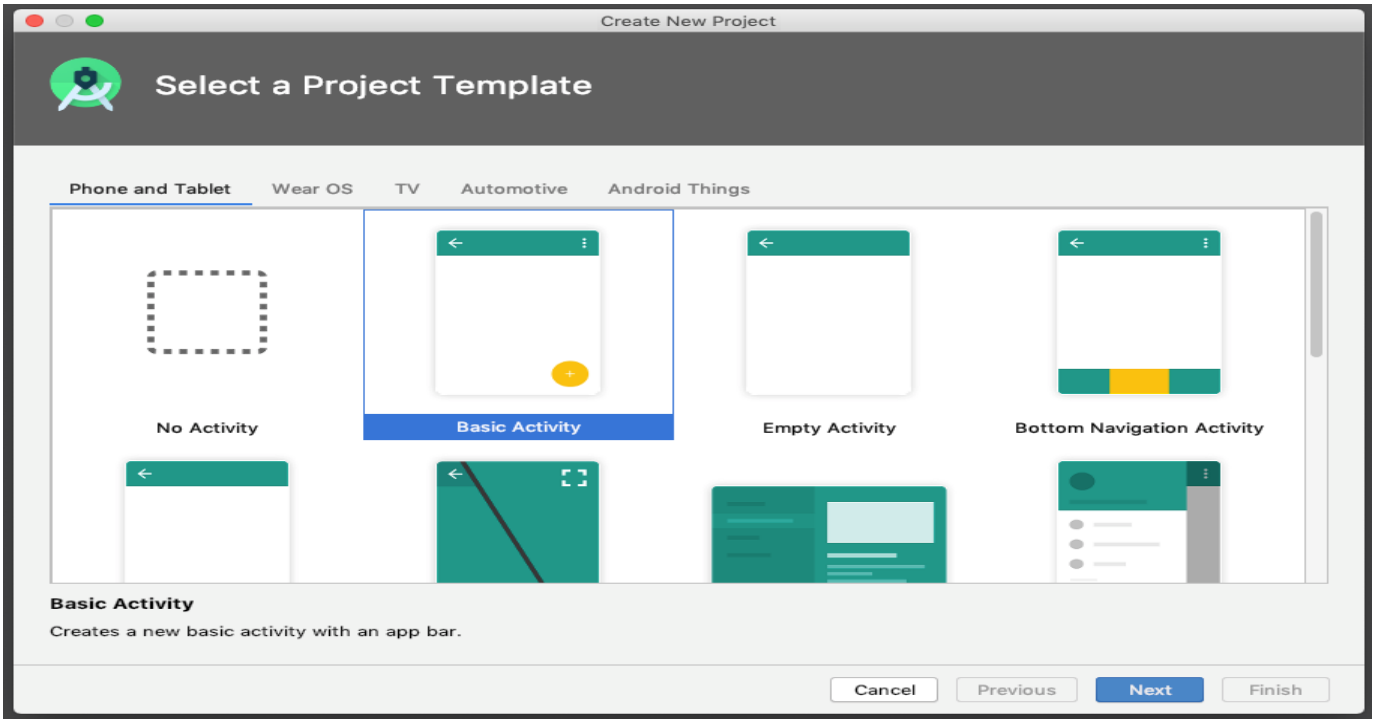
1.Install Android Studio

2.Create your first project

click Start a new Android Studio project.



Select Basic Activity (not the default). Click Next.



Give your application a name, such as My First App.

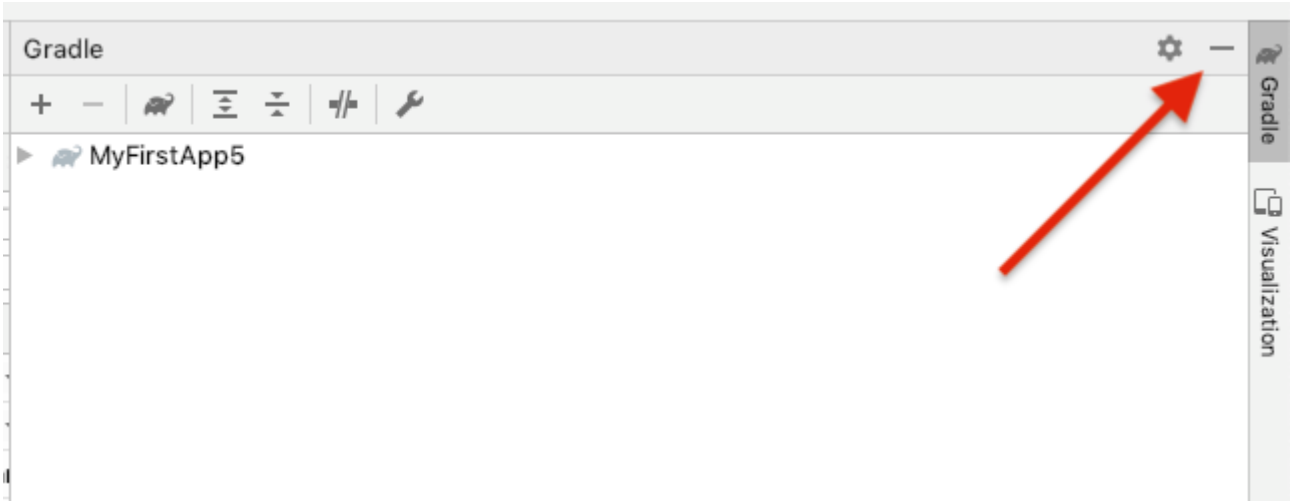
Make sure the Language is set to Kotlin.

Leave the defaults for the other fields.

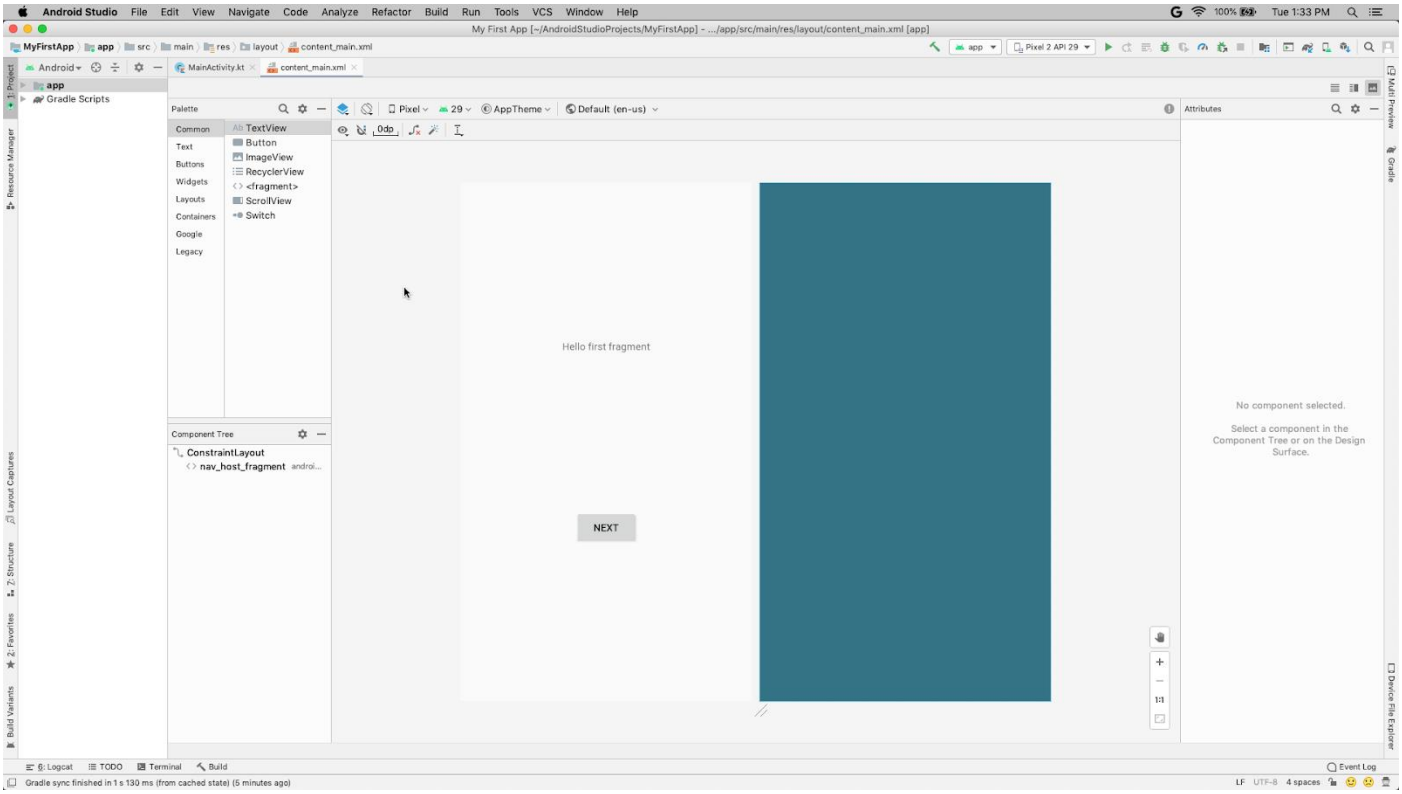
Click Finish.

Get your screen set up

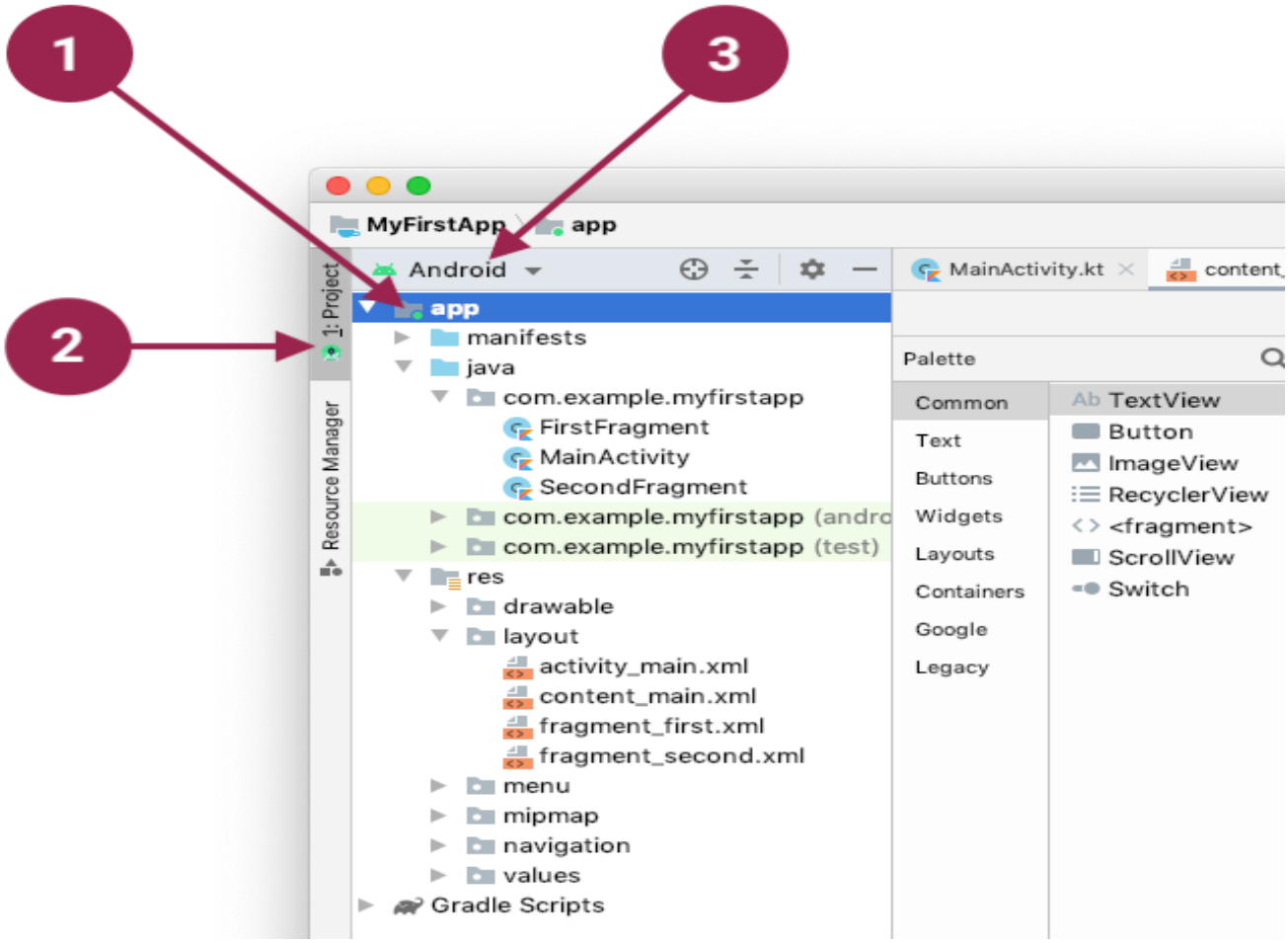
If there's a **Gradle** window open on the right side, click on the minimize button (—) in the upper right corner.



Depending on the size of your screen, consider resizing the pane on the left showing the project folders to take up less space.



Explore the project structure and layout



Double-click the app (1) folder to expand the hierarchy of app files. (See (1) in the screenshot.)

If you click Project (2), you can hide or show the Project view.

The current Project view selection is Project > Android.

Create a virtual device (emulator)

In Android Studio, select Tools > AVD Manager, or click the AVD Manager icon in the toolbar. 1ef215721ed1bd47.png

Click +Create Virtual Device. (If you have created a virtual device before, the window shows all of your existing devices and the +Create Virtual Device button is at the bottom.) The Select Hardware window shows a list of pre-configured hardware device definitions.

Choose a device definition, such as Pixel 2, and click Next. (For this codelab, it really doesn't matter which device definition you pick).

In the System Image dialog, from the Recommended tab, choose the latest release. (This does matter.)

If a Download link is visible next to a latest release, it is not installed yet, and you need to download it first. If necessary, click the link to start the download, and click Next when it's done. This may take a while depending on your connection speed.

In the next dialog box, accept the defaults, and click Finish.

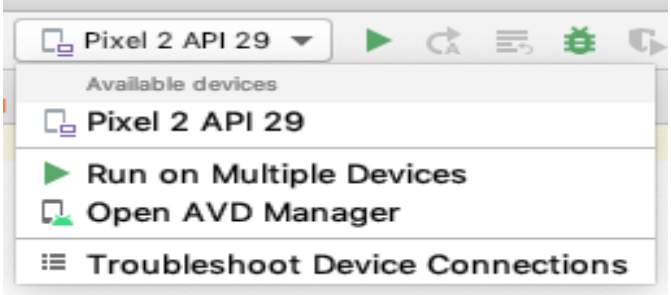
The AVD Manager now shows the virtual device you added.

If the Your Virtual Devices AVD Manager window is still open, go ahead and close it.

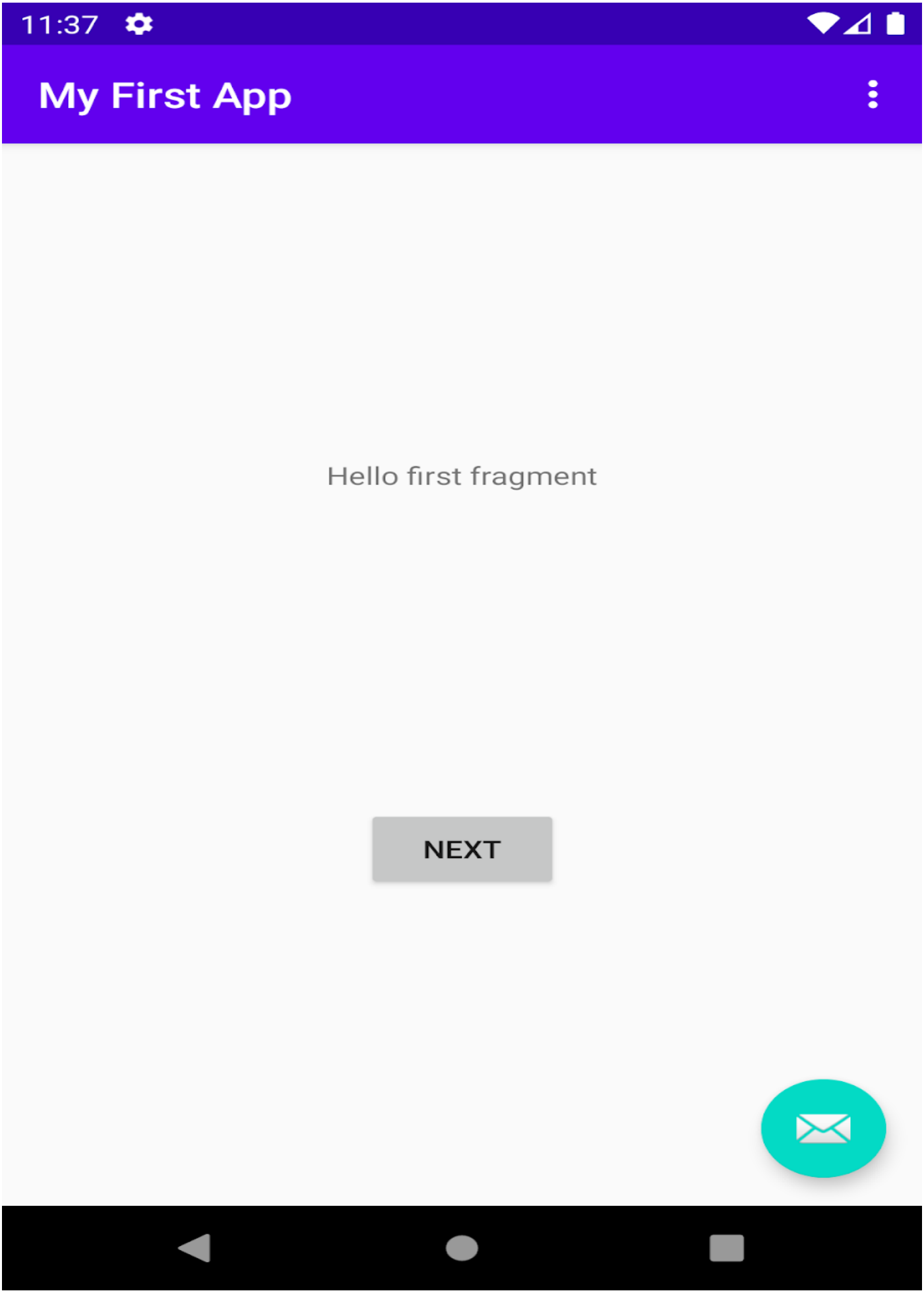
Run your app on your new emulator]

In Android Studio, select Run > Run ‘app', or click the Run icon in the toolbar.  The icon changes once your app is running 

In Run > Select Device, under Available devices, select the virtual device that you just configured. A dropdown menu also appears in the toolbar.



Once your app builds and the emulator is ready, Android Studio uploads the app to the emulator and runs it. You should see your app as shown in the following screenshot.


11:37
My First App
Hello first fragment
NEXT

UNIT-2

Designing Application

2.1 Working with activity

In Android, an activity is referred to as one screen in an application. It is very similar to a single window of any desktop application. An Android app consists of one or more screens or activities.

Each activity goes through various stages or a lifecycle and is managed by activity stacks. So when a new activity starts, the previous one always remains below it. There are four stages of an activity.

If an activity is in the foreground of the screen i.e at the top of the stack, then it is said to be active or running. This is usually the activity that the user is currently interacting with.

If an activity has lost focus and a non-full-sized or transparent activity has focused on top of your activity. In such a case either another activity has a higher position in multi-window mode or the activity itself is not focusable in the current window mode. Such activity is completely alive.

If an activity is completely hidden by another activity, it is stopped or hidden. It still retains all the information, and as its window is hidden thus it will often be killed by the system when memory is needed elsewhere.

The system can destroy the activity from memory by either asking it to finish or simply killing its process. When it is displayed again to the user, it must be completely restarted and restored to its previous state.

For each stage, android provides us with a set of 7 methods that have their own significance for each stage in the life cycle. The image shows a path of migration whenever an app switches from one state to another.

Activity Lifecycle in Android:

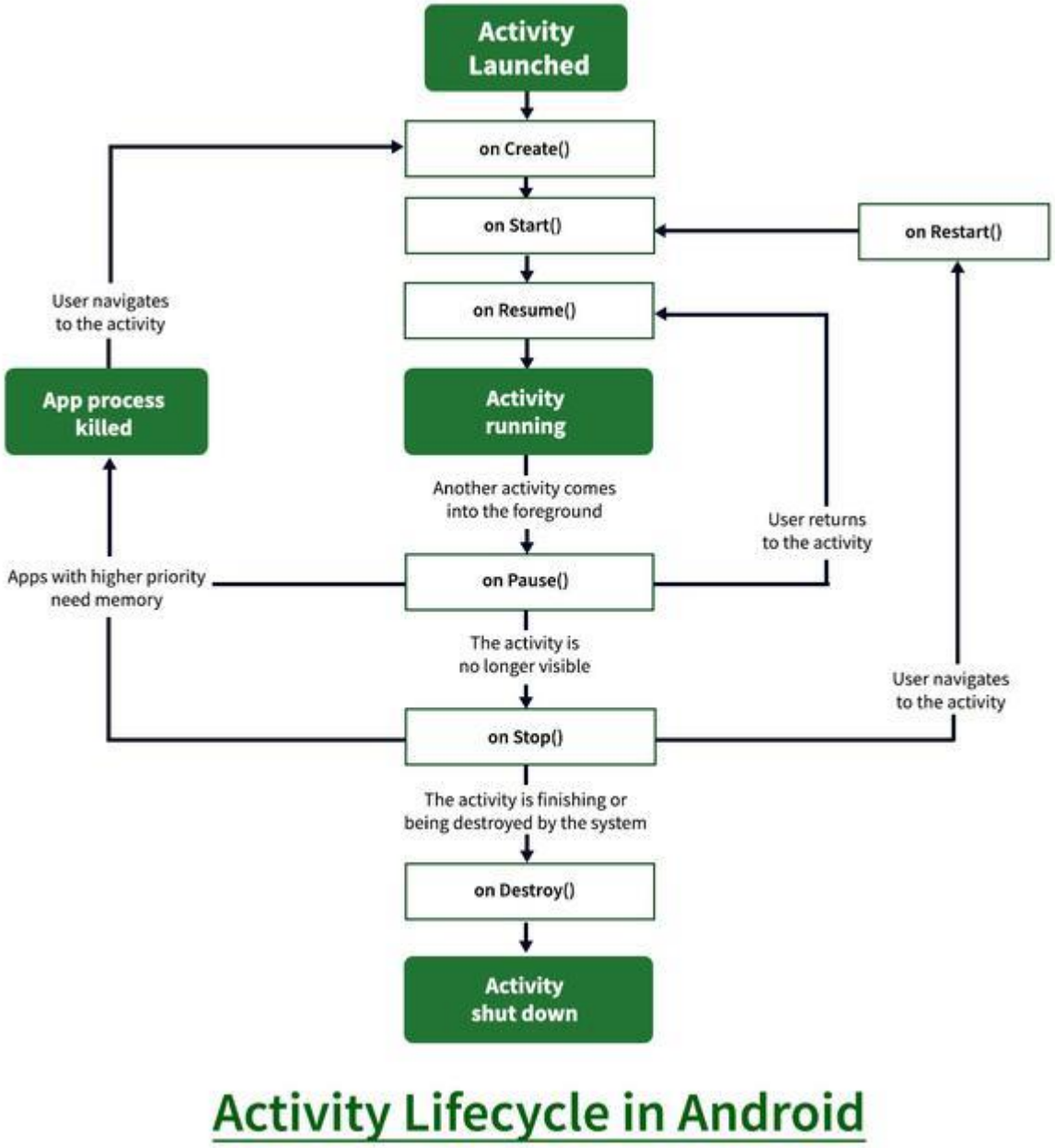
There are 7 different method in activity lifecycle: –

1. **onCreate()**: It is called when the activity is first created. This is where all the static work is done like creating views, binding data to lists, etc. This method also provides a Bundle containing its previous frozen state, if there was one.

   Example:

   ```
   import android.os.Bundle

   import android.widget.Toast

   import androidx.appcompat.app.AppCompatActivity


   class MainActivity : AppCompatActivity() {


      override fun onCreate(savedInstanceState: Bundle?) {
         super.onCreate(savedInstanceState)


         // Bundle containing previous frozen state
   ```

```
        setContentView(R.layout.activity_main)


        // The content view pointing to the id of layout

        // in the file activity_main.xml

        val toast = Toast.makeText(applicationContext, "onCreate Called",
Toast.LENGTH_LONG).show()

        }

}
```



Activity Lifecycle in Android

```
}
```

2. **onStart():** It is invoked when the activity is visible to the user. It is followed by onResume() if the activity is invoked from the background. It is also invoked after onCreate() when the activity is first started.

Example:

```
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity


class MainActivity : AppCompatActivity() {


   override fun onCreate(savedInstanceState: Bundle?) {
      super.onCreate(savedInstanceState)
      setContentView(R.layout.activity_main)


      val toast = Toast.makeText(applicationContext, "onCreate Called",
Toast.LENGTH_LONG).show()
      }


   override fun onStart() {
      super.onStart()
      // It will show a message on the screen
      // then onStart is invoked
      val toast  = Toast.makeText(applicationContext, "onStart Called",
Toast.LENGTH_LONG).show()
   }
}
```

3. **onRestart():** It is invoked after the activity has been stopped and prior to its starting stage and thus is always followed by onStart() when any activity is revived from background to on-screen.

Example:

```
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity


class MainActivity : AppCompatActivity() {
```

```
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val toast = Toast.makeText(applicationContext, "onCreate Called",
Toast.LENGTH_LONG).show()
        }

    override fun onRestart() {
        super.onRestart()
        // It will show a message on the screen
        // then onRestart is invoked
        val toast = Toast.makeText(applicationContext, "onRestart Called",
Toast.LENGTH_LONG).show()
    }
}
```

4. **onResume():** It is invoked when the activity starts interacting with the user. At this point, the activity is at the top of the activity stack, with a user interacting with it. Always followed by onPause() when the activity goes into the background or is closed by the user.

   Example:

```
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val toast = Toast.makeText(applicationContext, "onCreate Called",
Toast.LENGTH_LONG).show()
        }
```

```
    override fun onResume() {
        super.onResume()
        // It will show a message on the screen
        // then onResume is invoked
        val toast = Toast.makeText(applicationContext, "onResume Called",
Toast.LENGTH_LONG).show()
    }
}
```

5. **onPause():** It is invoked when an activity is going into the background but has not yet been killed. It is a counterpart to onResume(). When an activity is launched in front of another activity, this callback will be invoked on the top activity (currently on screen). The activity, under the active activity, will not be created until the active activity's onPause() returns, so it is recommended that heavy processing should not be done in this part.

   Example:

```
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val toast = Toast.makeText(applicationContext, "onCreate Called",
Toast.LENGTH_LONG).show()
        }

    override fun onPause() {
        super.onPause()
        // It will show a message on the screen
        // then onPause is invoked
```

```
        val toast = Toast.makeText(applicationContext, "onPause Called",
Toast.LENGTH_LONG).show()
    }
}
```

6. **onStop():** It is invoked when the activity is not visible to the user. It is followed by onRestart() when the activity is revoked from the background, followed by onDestroy() when the activity is closed or finished, and nothing when the activity remains on the background only.

   Example:

```
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        val toast = Toast.makeText(applicationContext, "onCreate Called",
Toast.LENGTH_LONG).show()
        }

    override fun onStop() {
        super.onStop()
        // It will show a message on the screen
        // then onStop is invoked
        val toast = Toast.makeText(applicationContext, "onStop Called",
Toast.LENGTH_LONG).show()
    }
}
```

7. **onDestroy():**The final call received before the activity is destroyed. This can happen either because the activity is finishing (when finish() is invoked) or because the system is temporarily destroying this instance of the activity to save space.

Example:

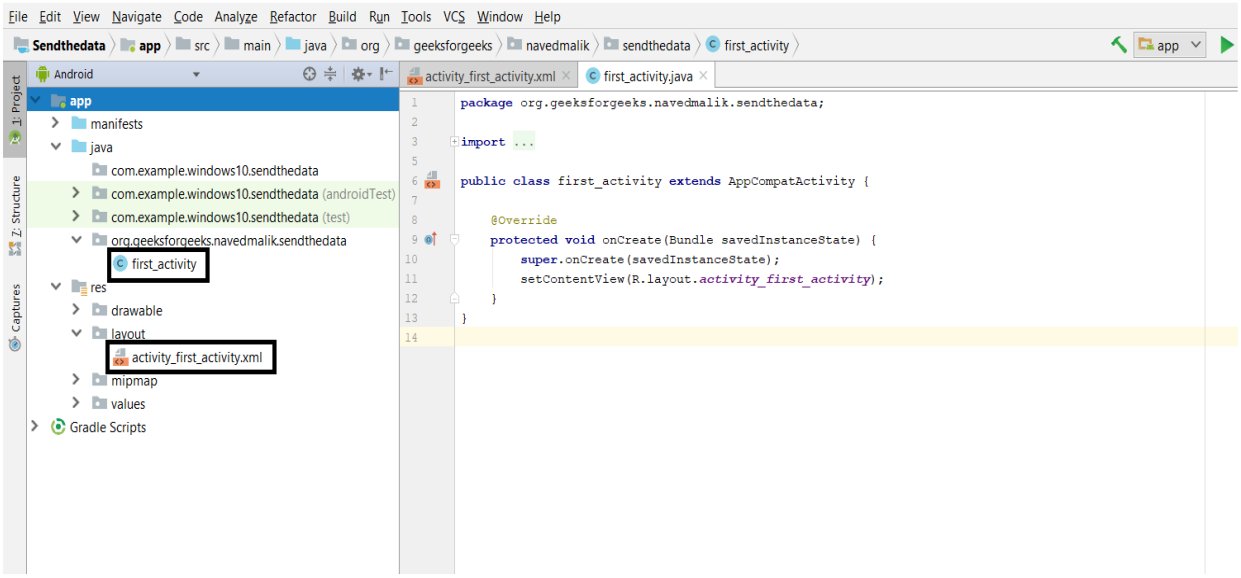
```
import android.os.Bundle
import android.widget.Toast
import androidx.appcompat.app.AppCompatActivity


class MainActivity : AppCompatActivity() {


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)


        val toast = Toast.makeText(applicationContext, "onCreate Called",
Toast.LENGTH_LONG).show()
        }


    override fun onDestroy() {
        super.onDestroy()
        // It will show a message on the screen
        // then onDestroy is invoked
        val toast = Toast.makeText(applicationContext, "onDestroy Called",
Toast.LENGTH_LONG).show()
    }
}
```

2.2 Redirecting to another activity and passing data

How to "**Send the data from one activity to second activity using Intent**". In this example, we have two activities, activity_first which are the source activity, and activity_second which is the destination activity. We can send the data using the putExtra() method from one activity and get the data from the second activity using the getStringExtra() method.

Step by Step Implementation:

Step1: Create a New Project in Android Studio



Step2: Working with the XML Files

Next, go to the **activity_main.xml file**, which represents the UI of the project. Below is the code for the **activity_main.xml** file. Comments are added inside the code to understand the code in more detail. Open the "activity_first_activity.xml" file and add the following widgets in a **Relative Layout.**

- An **EditText** to Input the message
- A **Button** to send the data

Also, Assign the **ID** to each component along with other attributes as shown in the image and the code below. The assigned ID on a component helps that component to be easily found and used in the Java/Kotlin files.

Syntax:

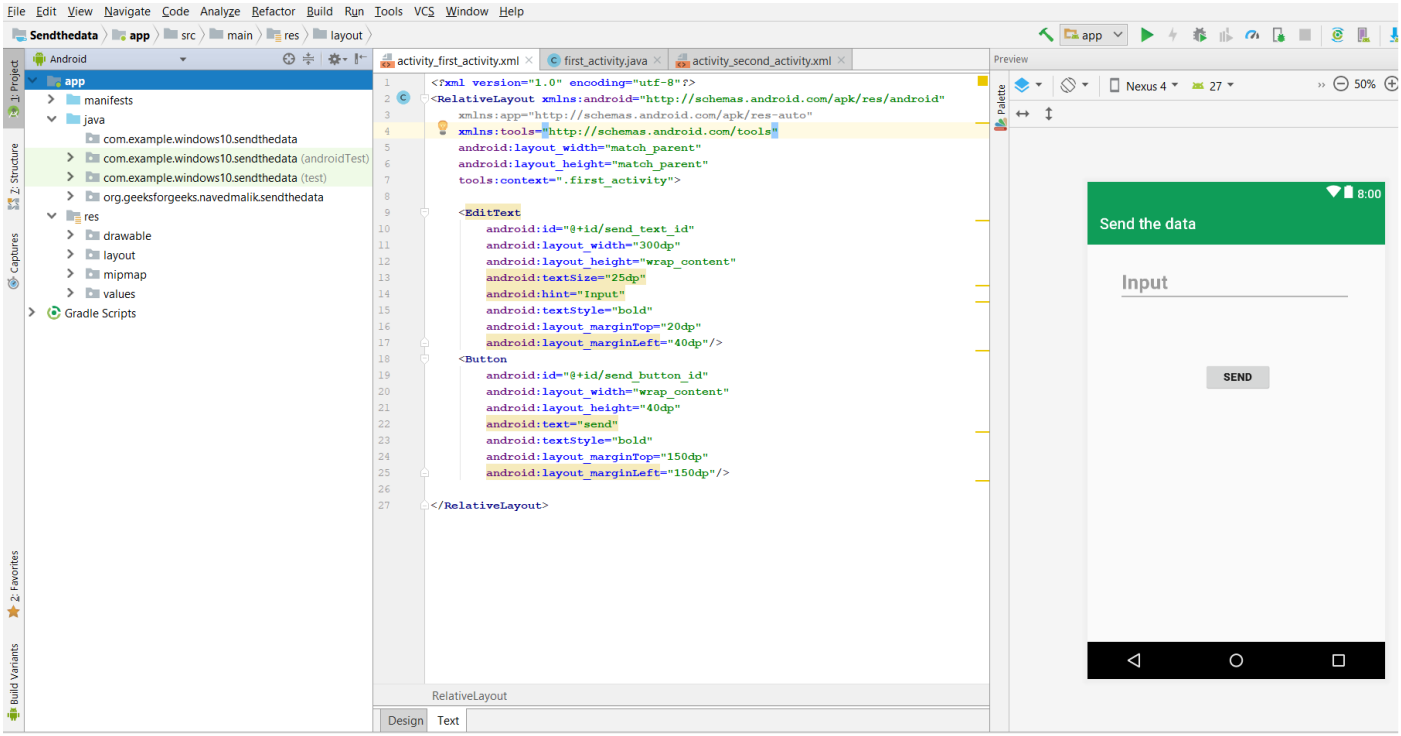
- android:id="@+id/id_name"
- Send Button: send_button_id
- input EditText: send_text_id

XML:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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"
    tools:context=".first_activity">

    <EditText
        android:id="@+id/send_text_id"
```

```
        android:layout_width="300dp"
        android:layout_height="wrap_content"
        android:layout_marginLeft="40dp"
        android:layout_marginTop="20dp"
        android:hint="Input"
        android:textSize="25dp"
        android:textStyle="bold" />

    <Button
        android:id="@+id/send_button_id"
        android:layout_width="wrap_content"
        android:layout_height="40dp"
        android:layout_marginLeft="150dp"
        android:layout_marginTop="150dp"
        android:text="send"
        android:textStyle="bold" />
</RelativeLayout>
```

This will make the UI of the Application



Step3: Working with the MainActivity File

Go to the MainActivity File and refer to the following code. Below is the code for the MainActivity File. Comments are added inside the code to understand the code in more detail. Now, after the UI, this step will create the Backend of the App. For this, open the "first_activity" file and instantiate the components made in the XML file (EditText, send Button) using findViewById() method. This method binds the created object to the UI Components with the help of the assigned ID.

Syntax: General

ComponentType object = (ComponentType)findViewById(R.id.IdOfTheComponent);

Syntax: for components used is as follows:

Button send_button= findViewById(R.id.send_button_id);

send_text = findViewById(R.id.send_text_id);

Setting up the Operations for the Sending and Receiving of Data.

These Operations are as follows:

Add the listener to the send button and this button will send the data.

This is done as follows:

send_button.setOnClickListener(v -> {}

after clicking this button following operation will be performed. Now create the String type variable to store the value of EditText which is input by the user. Get the value and convert it to a string.

This is done as follows:

String str = send_text.getText().toString();

Now create the Intent object First_activity.java class to Second_activity class.

This is done as follows:

Intent intent = new Intent(getApplicationContext(), Second_activity.class);

where getApplicationContext() will fetch the current activity. Put the value in the putExtra method in the key-value pair then start the activity.

This is done as follows:

intent.putExtra("message_key", str);

startActivity(intent);

where "str" is the string value and the key is "message_key" this key will use to get the str value

Example:

import android.content.Intent

import android.os.Bundle

import android.widget.Button

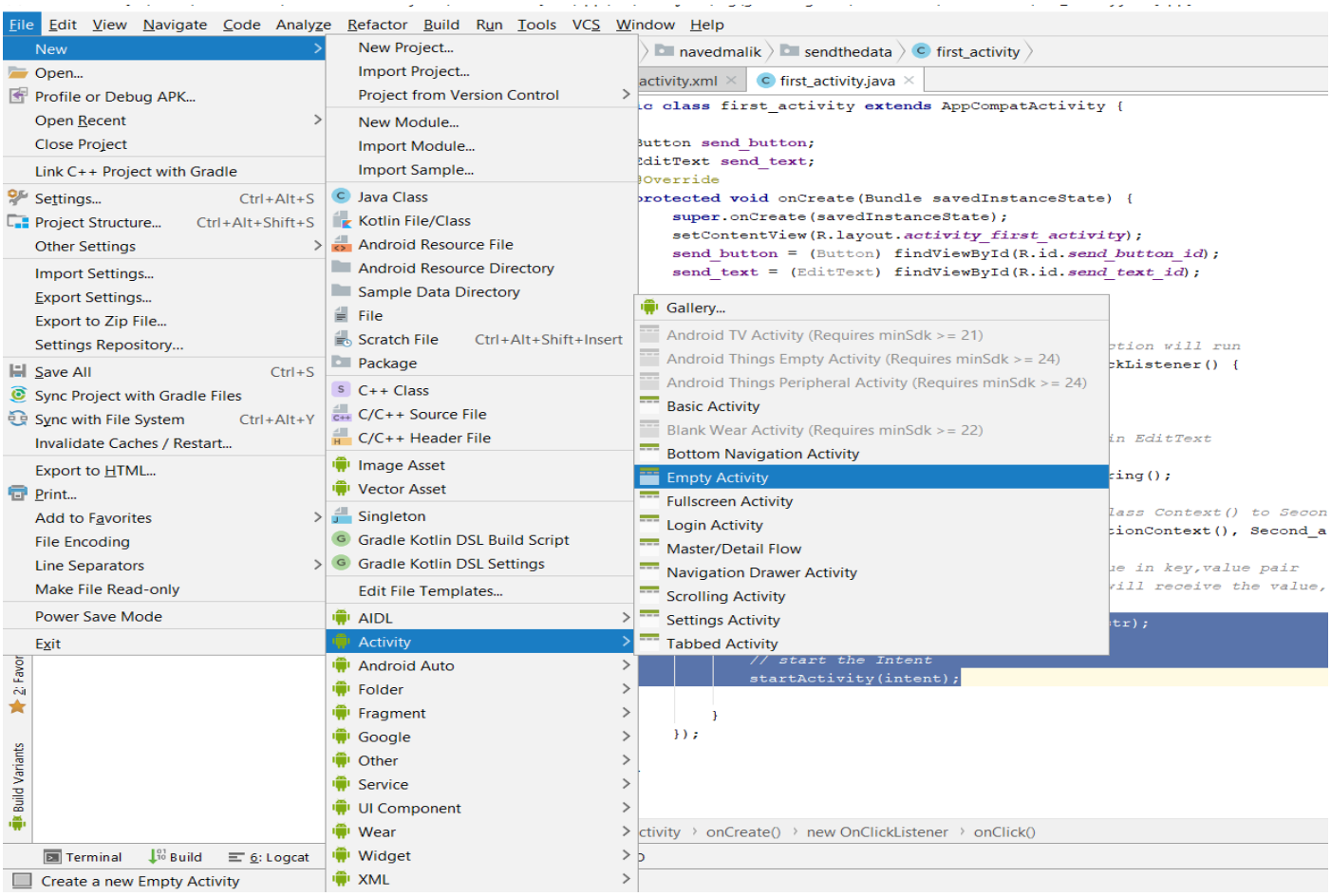
import android.widget.EditText

```
import androidx.appcompat.app.AppCompatActivity

class first_activity : AppCompatActivity() {

    // define the variable
    lateinit var send_button: Button
    lateinit var send_text: EditText

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_first_activity)

        send_button = findViewById(R.id.send_button_id)
        send_text = findViewById(R.id.send_text_id)

        // add the OnClickListener in sender button after clicked this button following Instruction will run
        send_button.setOnClickListener {
            // get the value which input by user in EditText and convert it to string
            val str = send_text.text.toString()
            // Create the Intent object of this class Context() to Second_activity class
            val intent = Intent(applicationContext, Second_activity::class.java)
            // now by putExtra method put the value in key, value pair key is
            // message_key by this key we will receive the value, and put the string
            intent.putExtra("message_key", str)
            // start the Intent
            startActivity(intent)
        }
    }
}
```

Step4: Creating Second_Activity to Receive the Data.

The steps to create the second activity are as follows:

android project > File > new > Activity > Empty Activity



Step5: Working with the Second XML File

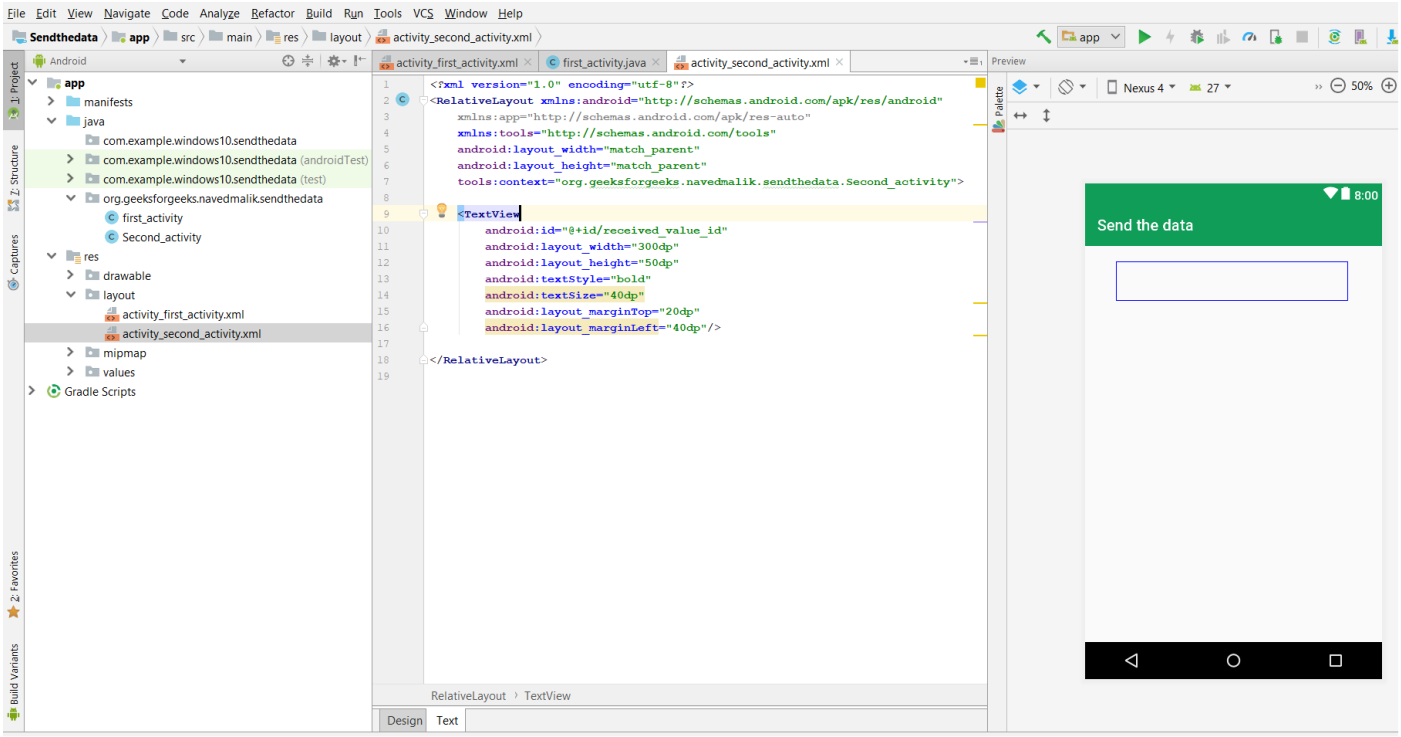
Add TextView to display the received messages. assign an ID to Textview.

XML:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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"
    tools:context="org.geeksforgeeks.navedmalik.sendthedata.Second_activity">
    <TextView
        android:id="@+id/received_value_id"
        android:layout_width="300dp"
```

```
    android:layout_height="50dp"
    android:layout_marginLeft="40dp"
    android:layout_marginTop="20dp"
    android:textSize="40sp"
    android:textStyle="bold"
    android:layout_marginStart="40dp" />
</RelativeLayout>
```

The Second Activity is shown below:



Step6: Working with the SecondActivity File

Define the TextView variable, use findViewById() to get the TextView as shown above.

receiver_msg = (TextView) findViewById(R.id.received_value_id);

Now In the second_activity.java file create the object of getIntent to receive the value in String type variable by the getStringExtra method using message_key.

Intent intent = getIntent();

String str = intent.getStringExtra("message_key");

The received value set in the TextView object of the second activity XML file

receiver_msg.setText(str);

Kotlin:

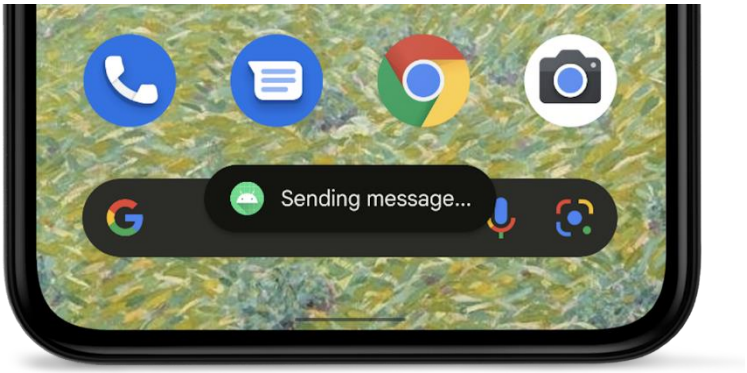
import android.os.Bundle

```
import android.widget.TextView
import androidx.appcompat.app.AppCompatActivity

class Second_activity : AppCompatActivity() {

    lateinit var receiver_msg: TextView

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_second_activity)

        receiver_msg = findViewById(R.id.received_value_id)
        // create the get Intent object
        val intent = intent
        // receive the value by getStringExtra() method and
        // key must be same which is send by first activity
        val str = intent.getStringExtra("message_key")
        // display the string into textView
        receiver_msg.text = str
    }
}
```

Toast

A toast provides simple feedback about an operation in a small popup. It only fills the amount of space required for the message and the current activity remains visible and interactive. Toasts automatically disappear after a timeout.

For example, clicking Send on an email triggers a "Sending message..." toast, as shown in the following screen capture:



If your app targets Android 12 (API level 31) or higher, its toast is limited to two lines of text and shows the application icon next to the text. Be aware that the line length of this text varies by screen size, so it's good to make the text as short as possible.

Instantiate a Toast object

Use the makeText() method, which takes the following parameters:

1. The activity Context.
2. The text that should appear to the user.
3. The duration that the toast should remain on the screen.

The makeText() method returns a properly initialized Toast object.

Show the toast

To display the toast, call the show() method, as demonstrated in the following example:

```
val text = "Hello toast!"
val duration = Toast.LENGTH_SHORT

val toast = Toast.makeText(this, text, duration) // in Activity
toast.show()
```

2.3 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.

Size, padding, and margins

The size of a view is expressed with a width and height. A view has two pairs of width and height values.

The first pair is known as *measured width* and *measured height*. These dimensions define how big a view wants to be within its parent. You can obtain the measured dimensions by calling getMeasuredWidth() and getMeasuredHeight().

The second pair is known as *width* and *height*, or sometimes *drawing width* and *drawing height*. These dimensions define the actual size of the view on screen, at drawing time and after layout. These values might, but don't have to, differ from the measured width and height. You can obtain the width and height by calling getWidth() and getHeight().

To measure its dimensions, a view takes into account its padding. The padding is expressed in pixels for the left, top, right and bottom parts of the view. You can use padding to offset the content of the view by a specific number of pixels. For instance, a left padding of two pushes the view's content two pixels to the right of the left edge. You can set padding using the setPadding(int, int, int, int) method and query it by calling getPaddingLeft(), getPaddingTop(), getPaddingRight(), and getPaddingBottom().

Although a view can define a padding, it doesn't support margins. However, view groups do support margins. See ViewGroup and ViewGroup.MarginLayoutParams for more information.

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.

1.LinearLayout:

Android **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. We can specify the linear layout orientation using **android:orientation** attribute.

All the child elements arranged one by one in multiple rows and multiple columns.

1. **Horizontal list:** One row, multiple columns.
2. **Vertical list:** One column, multiple rows.

LinearLayout in activity_main.xml file:

<?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:orientation="vertical"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        tools:context=".MainActivity">

        <TextView
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_margin="16dp"
                android:text="Enter your name here:"
                android:textSize="24dp"
                android:id="@+id/txtVw"/>

        <EditText
                android:id="@+id/editText"
                android:layout_width="match_parent"
                android:layout_height="wrap_content"
                android:layout_margin="16dp"
                android:hint="Name"
                android:inputType="text"/>

        <Button
                android:id="@+id/showInput"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center_horizontal"
```

```
android:text="show"

android:backgroundTint="@color/colorPrimary"

android:textColor="@android:color/white"/>


</LinearLayout>
```

2.RelativeLayout:

Android **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).

Instead of using LinearLayout, we have to use RelativeLayout to design the user interface and keep our hierarchy flat because it improves the performance of the application.

Attributes	Description
layout_alignParentLeft	It is set “true” to match the left edge of view to the left edge of parent.
layout_alignParentRight	It is set “true” to match the right edge of view to the right edge of parent.
layout_alignParentTop	It is set “true” to match the top edge of view to the top edge of parent.
layout_alignParentBottom	It is set “true” to match the bottom edge of view to the bottom edge of parent.
layout_alignLeft	It accepts another sibling view id and align the view to the left of the specified view id
layout_alignRight	It accepts another sibling view id and align the view to the right of the specified view id.
layout_alignStart	It accepts another sibling view id and align the view to start of the specified view id.
layout_alignEnd	It accepts another sibling view id and align the view to end of specified view id.
layout_centerInParent	When it is set “true”, the view will be aligned to the center of parent.
layout_centerHorizontal	When it is set “true”, the view will be horizontally centre aligned within its parent.
layout_centerVertical	When it is set “true”, the view will be vertically centre aligned within its parent.
layout_toLeftOf	It accepts another sibling view id and places the view left of the specified view id.
layout_toRightOf	It accepts another sibling view id and places the view right of the specified view id.
layout_toStartOf	It accepts another sibling view id and places the view to start of the specified view id.
layout_toEndOf	It accepts another sibling view id and places the view to end of the specified view id.
layout_above	It accepts another sibling view id and places the view above the specified view id.
layout_below	It accepts another sibling view id and places the view below the specified view id.

RelativeLayout in activity_main.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:paddingLeft="10dp"
    android:paddingRight="10dp">
    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:text="First name:"
        android:layout_marginTop="20dp"
        android:textSize="20dp"/>
    <EditText
        android:id="@+id/editText1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_toRightOf="@id/textView1"
        android:layout_marginTop="8dp"/>
    <TextView
        android:id="@+id/textView2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/textView1"
        android:layout_marginTop="10dp"
        android:text="Last name:"
        android:textSize="20dp"/>
```

```
    <EditText
        android:id="@+id/editText2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_toRightOf="@id/textView2"
        android:layout_marginTop="45dp"/>
    <Button
        android:id="@+id/btn4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_below="@id/textView2"
        android:layout_marginTop="20dp"
        android:text="Submit" />
</RelativeLayout>
```

3.ConstraintLayout:

ConstraintLayout is similar to that of other View Groups which we have seen in Android such as RelativeLayout, LinearLayout, and many more. In this article, we will take a look at using ConstraintLayout in Android.

Attributes	Description
android:id	This is used to give a unique id to the layout.
app:layout_constraintBottom_toBottomOf	This is used to constrain the view with respect to the bottom position.
app:layout_constraintLeft_toLeftOf	This attribute is used to constrain the view with respect to the left position.
app:layout_constraintRight_toRightOf	This attribute is used to constrain the view with respect to the right position.
app:layout_constraintTop_toTopOf	This attribute is used to constrain the view with respect to the top position.

Advantages of using ConstraintLayout in Android:

- ConstraintLayout provides you the ability to completely design your UI with the drag and drop feature provided by the Android Studio design editor.

- It helps to improve the UI performance over other layouts.
- With the help of ConstraintLayout, we can control the group of widgets through a single line of code.
- With the help of ConstraintLayout, we can easily add animations to the UI components which we used in our app.

Disadvantages of using ConstraintLayout:

- When we use the Constraint Layout in our app, the XML code generated becomes a bit difficult to understand.
- In most of the cases, the result obtain will not be the same as we got to see in the design editor.
- Sometimes we have to create a separate layout file for handling the UI for the landscape mode.

Working with the activity_main.xml file:

```
<?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=".MyActivity">

    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="16dp"
        android:layout_marginRight="16dp"
        android:gravity="center"
        android:padding="10dp"
        android:text="Geeks for Geeks"
        android:textColor="@color/black"
        android:textSize="20sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
```

</androidx.constraintlayout.widget.ConstraintLayout>

4.Table Layout:

Android **TableLayout** is a ViewGroup subclass which is used to display the child View elements in rows and columns. It will arrange all the children elements into rows and columns and does not display any border lines in between rows, columns or cells.

The working of TableLayout is almost similar to HTML table and it contains as many columns as row with the most cells.

Attributes	Description
android:id	This is the ID which uniquely identifies the layout.
android:collapseColumns	This specifies the zero-based index of the columns to collapse. The column indices must be separated by a comma: 1, 2, 5.
android:shrinkColumns	The zero-based index of the columns to shrink. The column indices must be separated by a comma: 1, 2, 5.
android:stretchColumns	The zero-based index of the columns to stretch. The column indices must be separated by a comma: 1, 2, 5.

How to declare TableLayout and TableRow?

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginTop="10dp"
        android:paddingLeft="5dp"
        android:paddingRight="5dp">

        // Add Table rows here
</TableLayout>
<TableRow android:background="#51B435" android:padding="10dp">
<TextView
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_weight="1"
android:text="Rank" />
</TableRow>
```

activity_main.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_marginTop="10dp"
        android:paddingLeft="5dp"
        android:paddingRight="5dp">
        <TextView
                android:id="@+id/txt"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="ICC Ranking of Players:"
                android:textSize = "20dp"
                android:textStyle="bold">
        </TextView>
        <TableRow android:background="#51B435" android:padding="10dp">
                <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:text="Rank" />
                <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:text="Player" />
                <TextView
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_weight="1"
                        android:text="Team" />
```

```
        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="Points" />
    </TableRow>
    <TableRow android:background="#F0F7F7" android:padding="5dp">
        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="1" />
        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="Virat Kohli" />
        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="IND" />
        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_weight="1"
                android:text="895" />
    </TableRow>
    <TableRow android:background="#F0F7F7" android:padding="5dp">
        <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
```

```
            android:layout_weight="1"
            android:text="2" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Rohit Sharma" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="IND" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="863" />
    </TableRow>
    <TableRow android:background="#F0F7F7" android:padding="5dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="3" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Faf du Plessis" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
```

```
            android:layout_weight="1"
            android:text="PAK" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="834" />
    </TableRow>
    <TableRow android:background="#F0F7F7" android:padding="5dp">
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="4" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Steven Smith" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="AUS" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
        android:text="820" />
    </TableRow>
    <TableRow android:background="#F0F7F7" android:padding="5dp">
        <TextView
```

```
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="5" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="Ross Taylor" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="NZ" />
        <TextView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="817" />
</TableRow> </TableLayout>
```

2.4 Navigation & Fragments

Navigation basically in mobile development refers to the interaction between different screens or pieces of contents within the application. Android Jetpack's architecture Component the Navigation is so powerful, providing a consistent and predictable user experience. The navigation component helps to implement the Navigation between activities and fragments or chunks of the data in the application by simple button clicks. In this article, it's been discussed from Navigation key properties to implementing sample Navigation between the fragments.

Three key properties of Navigation are:

1. **Navigation Graph:** This is the XML file containing all of the individual content areas, within the applications called destinations. These may be possible paths that users may take through the application.
2. **NavHost:** This is the XML file which is an empty container that displays the destinations as the user navigates, an. This basically contains the NavHostFragment, which displays the various destinations from Navigation Graph.

3. **NavController**: NavController is an object which manages the navigation of destinations with NavHost. This controls the swapping of destination content as the user navigates through the destinations through the application.

Benefits that developers get from Navigation Component are:

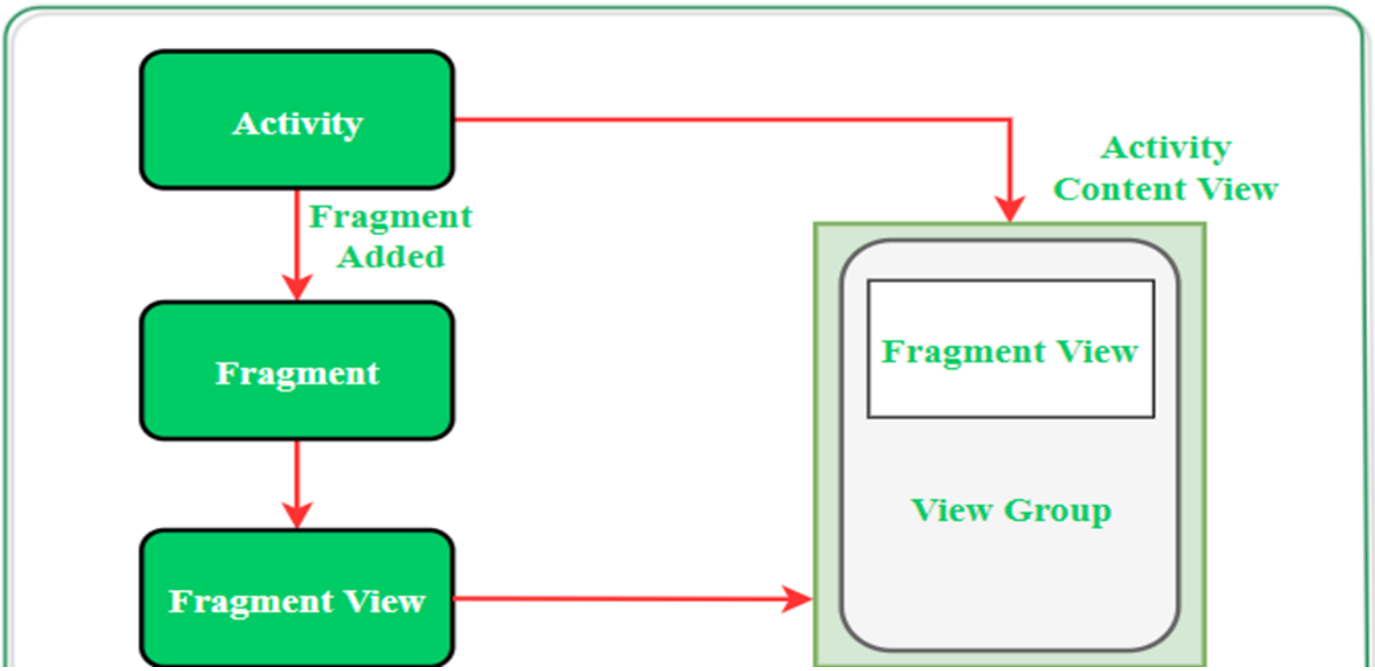
- Navigation Component handles the Fragment transaction.
- Handling Up and back actions, basically handling the Backstack.
- Provides standardized animation and transitions while switching the content of the NavHost.
- Easy handling and implementation of deep linking.
- Easy handling and implementation of Navigation UI patterns such as Navigation Drawer or Bottom Navigation Views etc.
- Safe Args, a Gradle plugin that provides type-safe data transfer between the destinations.
- Navigation Component also supports ViewModel. Can scope a view model to navigation graph to share UI related between destinations.

Fragments

In Android, the fragment is the part of Activity which represents a portion of User Interface(UI) on the screen. It is the modular section of the android activity that is very helpful in creating UI designs that are flexible in nature and auto-adjustable based on the device screen size. The UI flexibility on all devices improves the user experience and adaptability of the application. Fragments can exist only inside an activity as its lifecycle is dependent on the lifecycle of host activity. For example, if the host activity is paused, then all the methods and operations of the fragment related to that activity will stop functioning, thus fragment is also termed as **sub-activity**. Fragments can be added, removed, or replaced dynamically i.e., while activity is running.

<fragment> tag is used to insert the fragment in an android activity layout. By dividing the activity's layout multiple fragments can be added in it.

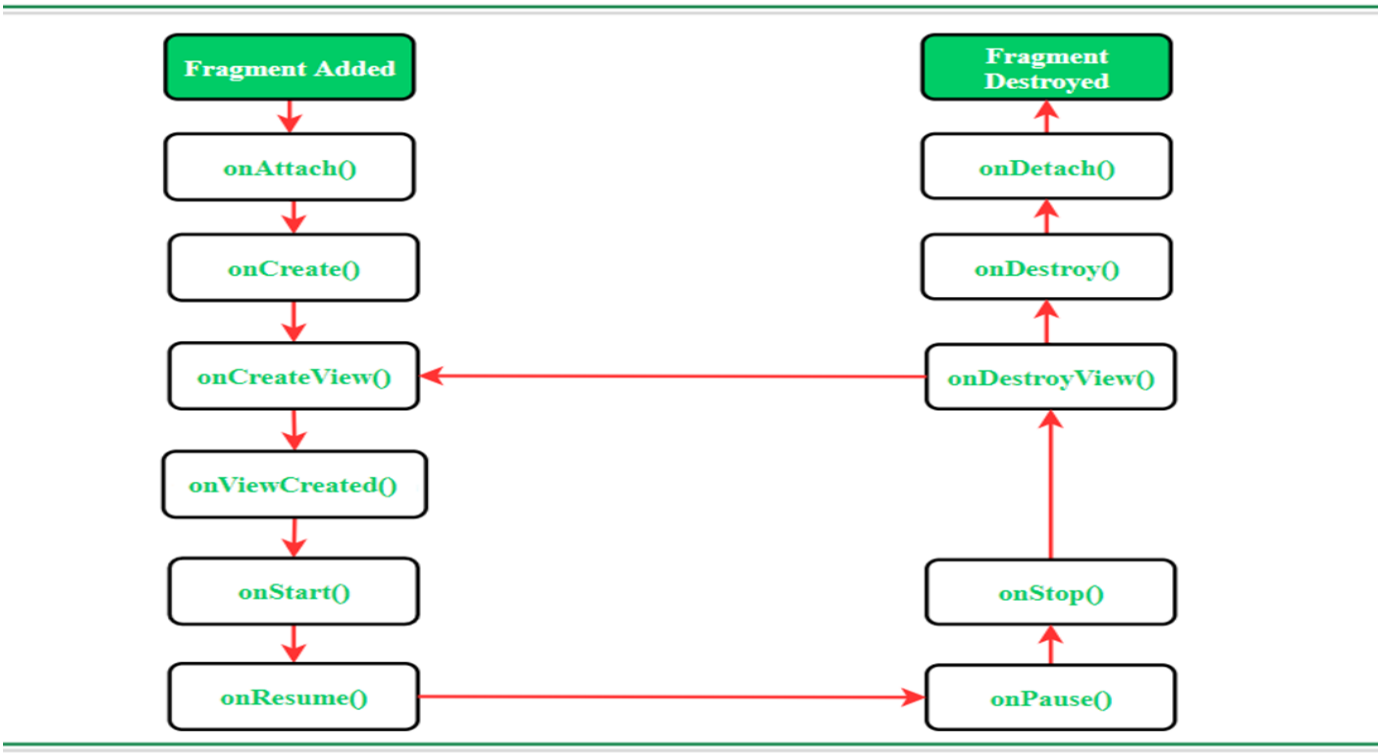
Below is the pictorial representation of fragment interaction with the activity:



Types of Android Fragments:

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

Fragment Lifecycle:



Each fragment has it's own lifecycle but due to the connection with the Activity it belongs to, the fragment lifecycle is influenced by the activity's lifecycle.

Methods of the Android Fragment

1. **onAttach():** once during the lifetime of a fragment.. When we attach fragment(child) to Main(parent) activity then it call first and then not call this method any time(like you run an app and close and reopen) simple means that this method call only one time.
2. **onCreate():** This method initializes the fragment by adding all the required attributes and components.
3. **onCreateView():** System calls this method to create the user interface of the fragment. The root of the fragment's layout is returned as the View component by this method to draw the UI. You should inflate your layout in onCreateView but shouldn't initialize other views using findViewById in onCreateView.
4. **onCreate():** It is called when the activity is first created. This is where all the static work is done like creating views, binding data to lists, etc. This method also provides a Bundle containing its previous frozen state, if there was one.

5. **onViewCreated():** It indicates that the activity has been created in which the fragment exists. View hierarchy of the fragment also instantiated before this function call.
6. **onStart():** The system invokes this method to make the fragment visible on the user's device.
7. **onResume():** This method is called to make the visible fragment interactive.
8. **onPause():** It indicates that the user is leaving the fragment. System call this method to commit the changes made to the fragment.
9. **onStop():** Method to terminate the functioning and visibility of fragment from the user's screen.
10. **onDestroyView():** System calls this method to clean up all kinds of resources as well as view hierarchy associated with the fragment. It will call when you can attach new fragment and destroy existing fragment Resoruce
11. **onDestroy():** It is called to perform the final clean up of fragment's state and its lifecycle.
12. **onDetach():** The system executes this method to disassociate the fragment from its host activity. It will call when your fragment Destroy(app crash or attach new fragment with existing fragment)

Example:

Step1: Create a new project

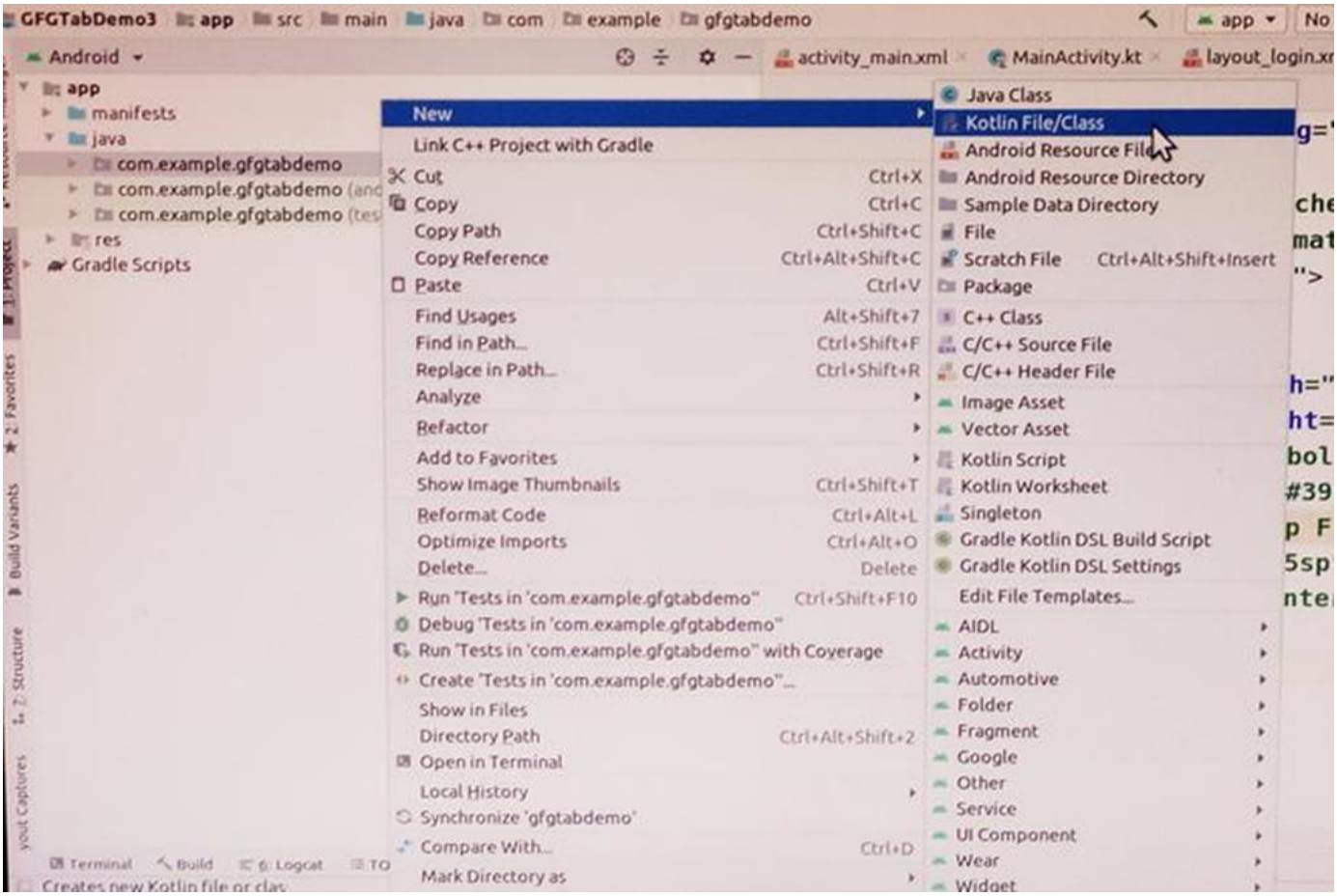
Step2: Create Fragments

- Go to app > res > layout > right-click > New > Layout Resource File and after that, it asks for the file name then gives "**layout_login**" as the name of that layout file.
- Use the same method to create another layout file "**layout_signup**".
- After that, use the following code for the "**layout_login.xml**" file. Here one TextView is shown.
- Below is the code for the layout_signup.xml file.

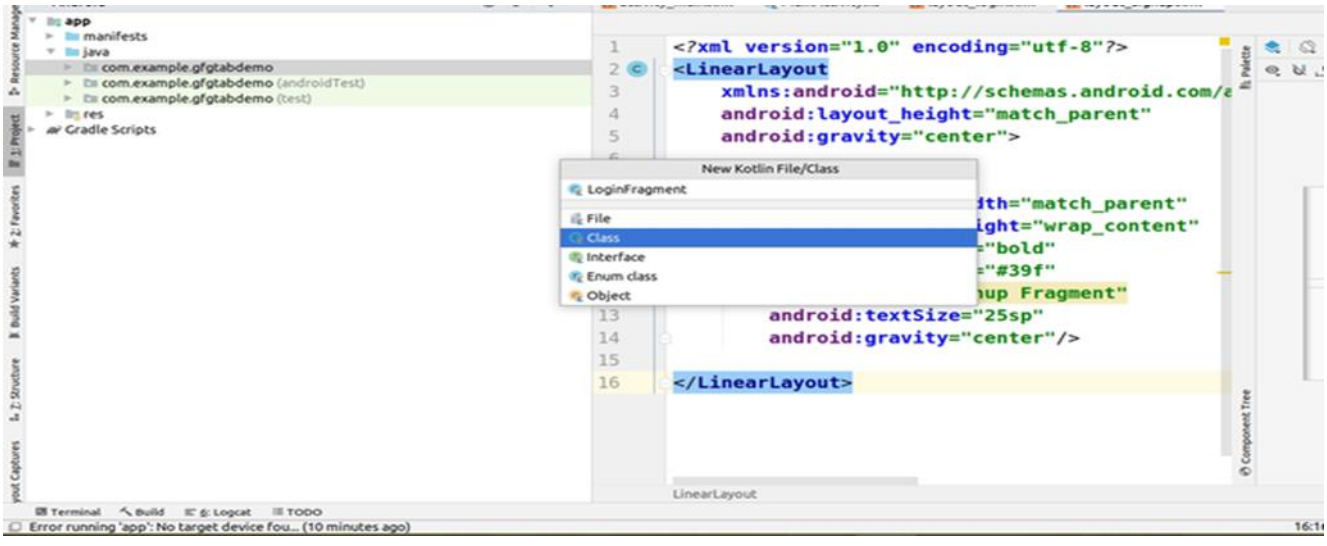
```
<?xml version="1.0" encoding="utf-8"?>
<!-- This linear layout is used to show elements
    in vertical or in horizontal linear manner -->
<LinearLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:gravity="center">

   <!-- This TextView indicates new fragment is open -->
   <TextView
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:gravity="center"
      android:text="Login Fragment"
      android:textColor="#0F9D58"
```

```
    android:textSize="25sp"
    android:textStyle="bold" />
</LinearLayout>
```

To create Fragment class, right-click on the first package of java directory which is located at **app > java > "com.example.gfgtabdemo"**, where "gfgtabdemo" is the project name in a small case. Move cursor on "**New**" and select "**Kotlin file/class**".



- Give "LoginFragment" as a name to that file and select the "class" option as shown in the below screenshot.



- To create a **Fragment**, it is necessary to make this class as a child of the Fragment class by using the ":" symbol. And override the "**onCreateView**" method to set the layout resource file to that fragment file as shown in the following code snippet.
- Use the above procedure to create the "**SignupFragment**" file.
- After that, use the following code in the "**LoginFragment.kt**" file.

LoginFragment.kt:

```
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment

// Here ":" symbol is indicate that LoginFragment
// is child class of Fragment Class
class LoginFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(
            R.layout.layout_login, container, false
        )
    }
    // Here "layout_login" is a name of layout file
    // created for LoginFragment
}
```

SignupFragment.kt:

```
import android.os.Bundle
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import androidx.fragment.app.Fragment

// Here ":" symbol is indicate that SignupFragment
```

```
// is child class of Fragment Class
class SignupFragment : Fragment() {
    override fun onCreateView(
        inflater: LayoutInflater, container: ViewGroup?, savedInstanceState: Bundle?
    ): View? {
        return inflater.inflate(
            R.layout.layout_signup, container, false
        )
    }}
```

Step3: Theme Configuration

- Open "**styles.xml**" which is placed inside of folder "**app > res > values > styles.xml**" as shown in the image below.

- Add the following code inside of <resources> tag in **styles.xml**.

```
<resources>
    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.DarkActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
    <style name="AppTheme.NoActionBar">
```

```
    <item name="windowActionBar">false</item>
    <item name="windowNoTitle">true</item>
  </style>

 <style name="AppTheme.AppBarOverlay"
    parent="ThemeOverlay.AppCompat.Dark.ActionBar" />
  <style name="AppTheme.PopupOverlay"
    parent="ThemeOverlay.AppCompat.Light" />
</resources>
```

- After that open, the "AndroidManifest.xml" file placed inside of folders "app > manifest > AndroidManifest.xml". We need to set the theme "@style/AppTheme.NoActionBar" inside of <activity> tag. To do the same type the highlighted line in the following screenshot, inside of that activity tag, in which you want to use tab layout.

```
        android:icon="@mipmap/ic_launcher"
        android:label="GFGTabDemo"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity"
            android:theme="@style/AppTheme.NoActionBar">
            <intent-filter>
                <action android:name="android.intent.action.MAIN

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

Step4: Adding Required Views

For the implementation of this topic, it is important to add some views. To do the same first, open build.gradle (Module: app) file, located at "Gradle Script > build.gradle (Module: app)", and add the following dependency inside of dependencies block. And don't forget to click on "sync now". This dependency is required, to make use of "appbar layout".

Step5: Working with the activity_main.xml file

After that, it is necessary to add the following views in a layout file of activity so open it. Here we use "activity_main.xml".

- AppBarLayout

- ToolBar
- TabLayout
- ViewPager

Add the following code to the “activity_main.xml” file. Comments are added inside the code to understand the code in more detail.

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


<!-- In order to use tabs, coordinator layout is useful-->
<androidx.coordinatorlayout.widget.CoordinatorLayout
   xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:app="http://schemas.android.com/apk/res-auto"
   android:layout_width="match_parent"
   android:layout_height="match_parent">

   <!--This appbarlayout covers the toolbar and tablayout-->
   <com.google.android.material.appbar.AppBarLayout
      android:id="@+id/appbar"
      android:layout_width="match_parent"
      android:layout_height="wrap_content"
      android:background="#0F9D58"
      android:theme="@style/AppTheme.AppBarOverlay">

      <!--toolbar is one component which is necessary
         because if we not use this then title is not shown
         when project executed-->
      <androidx.appcompat.widget.Toolbar
         android:id="@+id/toolbar"
         android:layout_width="match_parent"
         android:layout_height="wrap_content"
         app:layout_scrollFlags="scroll|enterAlways"
         app:popupTheme="@style/AppTheme.PopupOverlay" />

      <!-- tablayout which contains which is
```

```
        important to show tabs -->
    <com.google.android.material.tabs.TabLayout
        android:id="@+id/tab_tablayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:tabIndicatorColor="#FFF"
        app:tabIndicatorHeight="3dp"
        app:tabMode="fixed" />
</com.google.android.material.appbar.AppBarLayout>

<!-- view pager is used to open other fragment by using
     left or right swip-->
<androidx.viewpager.widget.ViewPager
    android:id="@+id/tab_viewpager"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:layout_marginTop="5dp"
    app:layout_behavior="@string/appbar_scrolling_view_behavior" />

</androidx.coordinatorlayout.widget.CoordinatorLayout>
```

Step6: Working with the MainActivity.kt file

```
import android.os.Bundle
import androidx.annotation.Nullable
import androidx.appcompat.app.AppCompatActivity
import androidx.appcompat.widget.Toolbar
import androidx.fragment.app.Fragment
import androidx.fragment.app.FragmentManager
import androidx.fragment.app.FragmentPagerAdapter
import androidx.viewpager.widget.ViewPager
import com.google.android.material.tabs.TabLayout
```

```
class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Create the object of Toolbar, ViewPager and
        // TabLayout and use “findViewById()” method*/
        var tab_toolbar = findViewById<Toolbar>(R.id.toolbar)
        var tab_viewpager = findViewById<ViewPager>(R.id.tab_viewpager)
        var tab_tablayout = findViewById<TabLayout>(R.id.tab_tablayout)

        // As we set NoActionBar as theme to this activity
        // so when we run this project then this activity doesn't
        // show title. And for this reason, we need to run
        // setSupportActionBar method
        setSupportActionBar(tab_toolbar)
        setupViewPager(tab_viewpager)

        // If we dont use setupWithViewPager() method then
        // tabs are not used or shown when activity opened
        tab_tablayout.setupWithViewPager(tab_viewpager)
    }

    // This function is used to add items in arraylist and assign
    // the adapter to view pager
    private fun setupViewPager(viewpager: ViewPager) {
        var adapter: ViewPagerAdapter = ViewPagerAdapter(supportFragmentManager)

        // LoginFragment is the name of Fragment and the Login
        // is a title of tab
        adapter.addFragment(LoginFragment(), "Login")
```

```
    adapter.addFragment(SignupFragment(), "Signup")

    // setting adapter to view pager.
    viewpager.setAdapter(adapter)
}

// This "ViewPagerAdapter" class overrides functions which are
// necessary to get information about which item is selected
// by user, what is title for selected item and so on.*/
class ViewPagerAdapter : FragmentPagerAdapter {

    // objects of arraylist. One is of Fragment type and
    // another one is of String type.*/
    private final var fragmentList1: ArrayList<Fragment> = ArrayList()
    private final var fragmentTitleList1: ArrayList<String> = ArrayList()

    // this is a secondary constructor of ViewPagerAdapter class.
    public constructor(supportFragmentManager: FragmentManager)
            : super(supportFragmentManager)

    // returns which item is selected from arraylist of fragments.
    override fun getItem(position: Int): Fragment {
        return fragmentList1.get(position)
    }

    // returns which item is selected from arraylist of titles.
    @Nullable
    override fun getPageTitle(position: Int): CharSequence {
        return fragmentTitleList1.get(position)
    }

    // returns the number of items present in arraylist.
```

```
    override fun getCount(): Int {
        return fragmentList1.size
    }

    // this function adds the fragment and title in 2 separate  arraylist.
    fun addFragment(fragment: Fragment, title: String) {
        fragmentList1.add(fragment)
        fragmentTitleList1.add(title)
    }
  }
}
```

2.5 Themes

A *theme* is a collection of attributes that's applied to an entire app, activity, or view hierarchy—not just an individual view. When you apply a theme, every view in the app or activity applies each of the theme's attributes that it supports. Themes can also apply styles to non-view elements, such as the status bar and window background.

Styles and themes are declared in a style resource file in res/values/, usually named styles.xml.

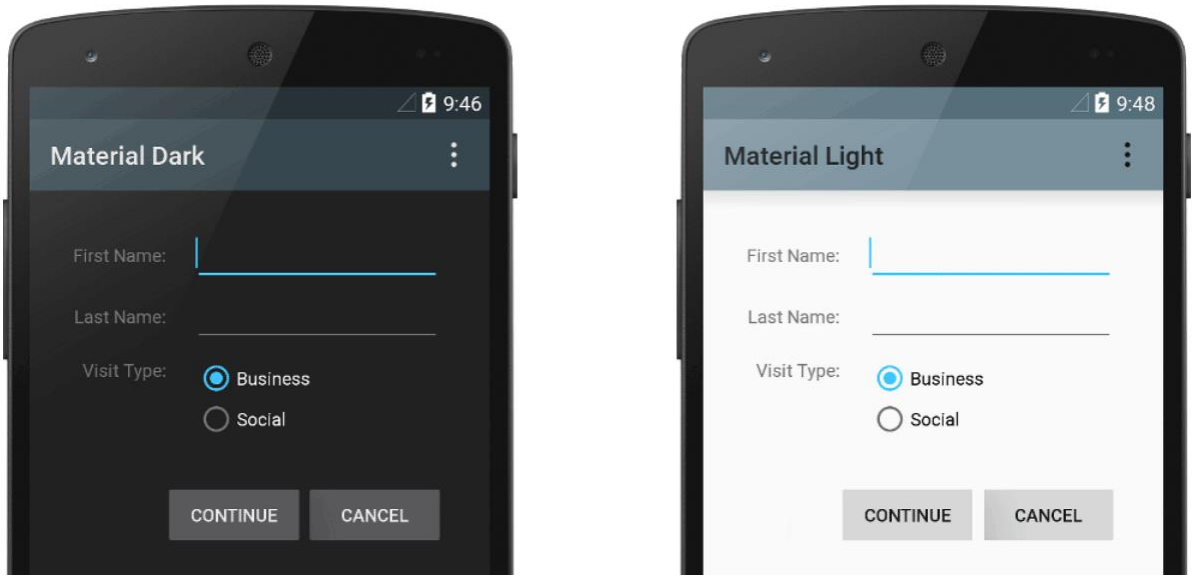


Figure - Two themes applied to the same activity: Theme.AppCompat (left) and Theme.AppCompat.Light (right).

Syntax

```
<manifest ... >
    <application android:theme="@style/ThemeAppCompat" ... >
```

```
    </application>
</manifest>
```

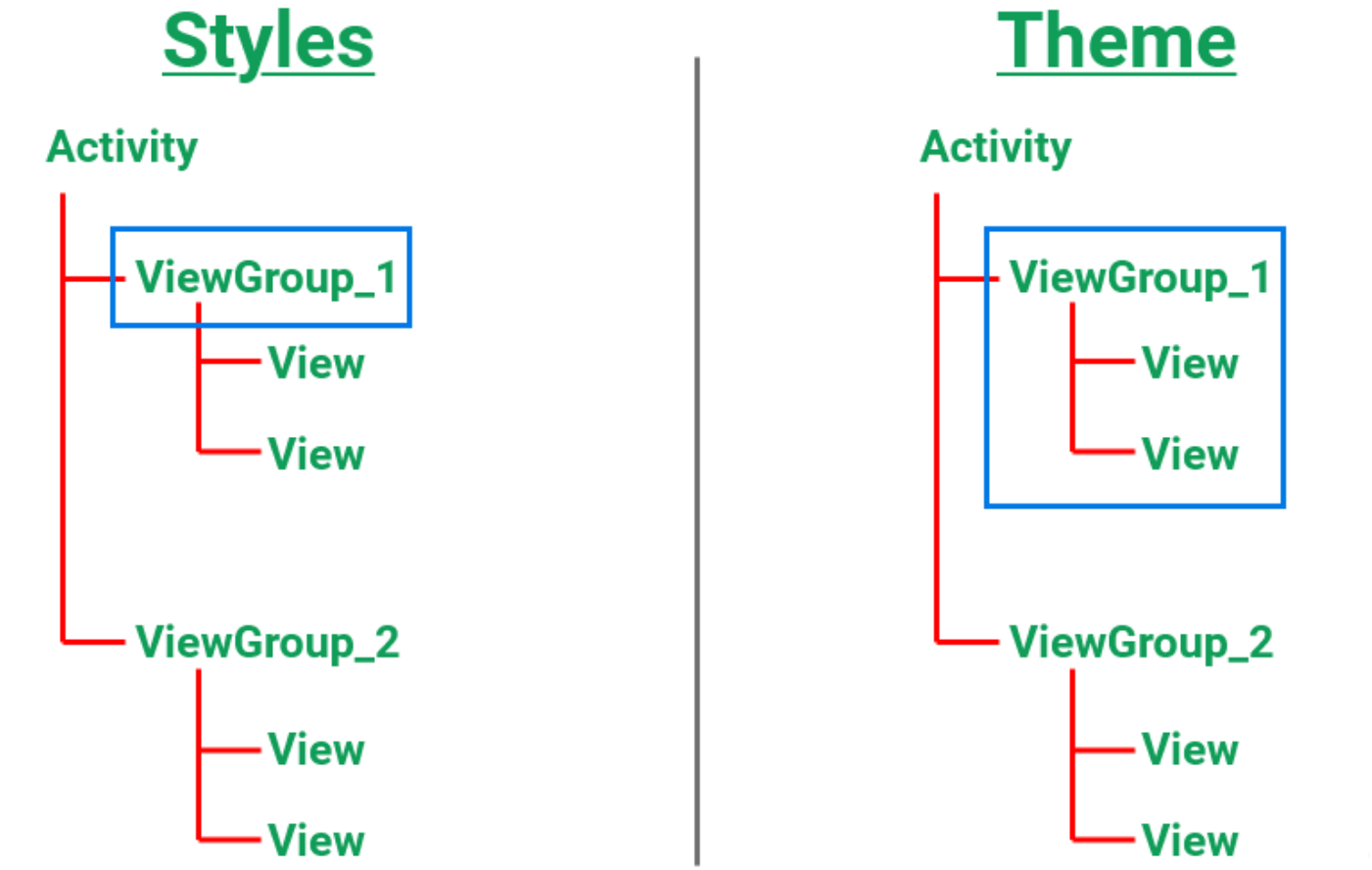
Themes versus styles

Themes and styles have many similarities, but they are used for different purposes. Themes and styles have the same basic structure—a key-value pair that maps *attributes* to *resources*.

A *style* specifies attributes for a particular type of view. For example, one style might specify a button's attributes. Every attribute you specify in a style is an attribute you can set in the layout file. Extracting all the attributes to a style makes it easy to use and maintain them across multiple widgets.

A *theme* defines a collection of named resources that can be referenced by styles, layouts, widgets, and so on. Themes assign semantic names, like colorPrimary, to Android resources.

Styles and themes are meant to work together. For example, you might have a style that specifies that one part of a button is colorPrimary, and another part is colorSecondary. The actual definitions of those colors are provided in the theme. When the device goes into night mode, your app can switch from its "light" theme to its "dark" theme, changing the values for all those resource names. You don't need to change the styles, since the styles are using the semantic names and not specific color definitions.



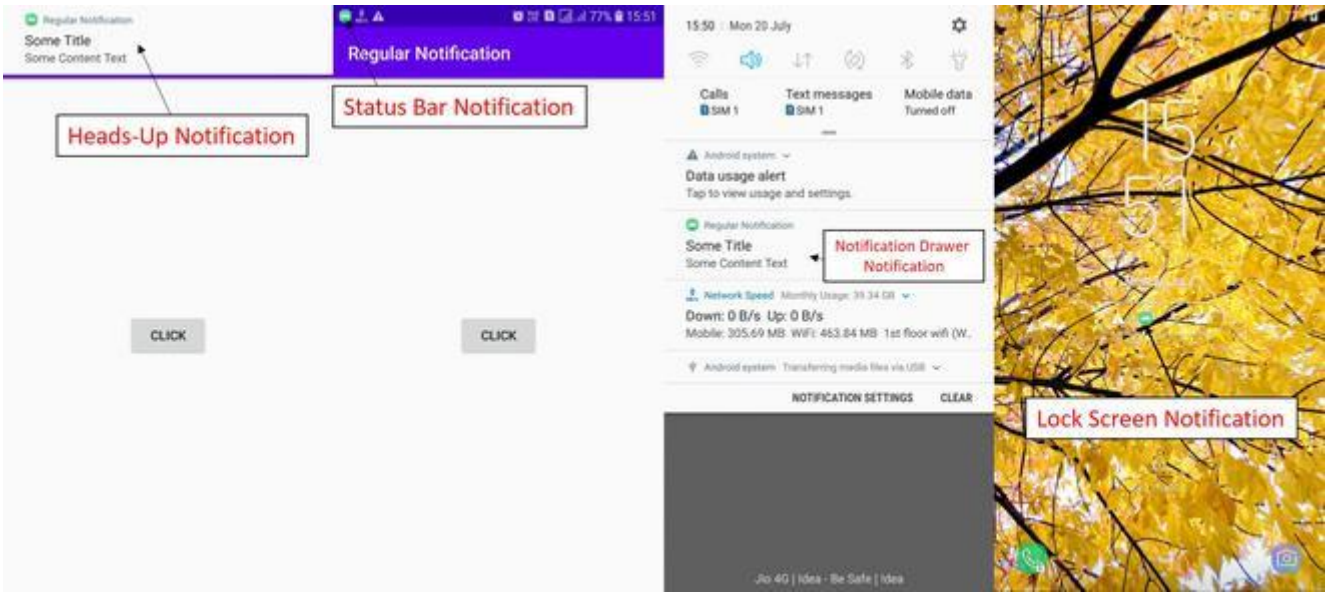
- Android developers witness the ambiguity in styles and themes. The reason is that in Android there are only tag names <style>, there is no <theme> tag available.
- Style is a collection of attributes that define the appearance of a single view. The style attributes are the font, color, size, background color, etc. For example, the font size may be different for heading and body.
- Theme, on the contrary, is applied to the entire app, or activity, or view hierarchy, not just for individual views. Themes can also apply styles to non-view elements such as status bar, window background, etc. For example, the colorPrimary is applied to all the Floating Action Buttons or Normal Buttons of the entire application. One can get the difference in the following image.

2.6 Notifications

Android Notification provides short, timely information about the action happened in the application, even it is not running. The notification displays the icon, title and some amount of the content text.

Notifications could be of various formats and designs depending upon the developer. In General, one must have witnessed these four types of notifications:

1. Status Bar Notification (appears in the same layout as the current time, battery percentage)
2. Notification drawer Notification (appears in the drop-down menu)
3. Heads-Up Notification (appears on the overlay screen, ex: Whatsapp notification, OTP messages)
4. Lock-Screen Notification (I guess you know it)



Set Android Notification Properties:

- **setSmallIcon()**: It sets the icon of notification.
- **setContentTitle()**: It is used to set the title of notification.
- **setContentText()**: It is used to set the text message.
- **setAutoCancel()**: It sets the cancelable property of notification.
- **setPriority()**: It sets the priority of notification.

Example:

Step1: Working with the activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   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"
   tools:context=".MainActivity">

   <Button
      android:id="@+id/btn"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
```

```
    android:layout_centerInParent="true"
    android:text="Send Notification" />

</RelativeLayout>
```

Step2: Create a new empty activity

Name the activity as afterNotification. When someone clicks on the notification, this activity will open up in our app that is the user will be redirected to this page. Below is the code for the activity_after_notification.xml file.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
  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"
  tools:context=".afterNotification">

  <TextView
    android:id="@+id/textView"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerInParent="true"
    android:text="Welcome To userdefine Notification"
    android:textSize="15sp"
    android:textStyle="bold" />

</RelativeLayout>
```

Step3: Working with the MainActivity.kt file

```
import android.app.Notification
import android.app.NotificationChannel
import android.app.NotificationManager
import android.app.PendingIntent
import android.content.Context
import android.content.Intent
```

```
import android.graphics.BitmapFactory
import android.graphics.Color
import android.os.Build
import android.os.Bundle
import android.widget.Button
import android.widget.RemoteViews
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

    // declaring variables
    lateinit var notificationManager: NotificationManager
    lateinit var notificationChannel: NotificationChannel
    lateinit var builder: Notification.Builder
    private val channelId = "i.apps.notifications"
    private val description = "Test notification"

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // accessing button
        val btn = findViewById<Button>(R.id.btn)

        // it is a class to notify the user of events that happen.
        // This is how you tell the user that something has happened in the
        // background.
        notificationManager = getSystemService(Context.NOTIFICATION_SERVICE) as
NotificationManager

        // onClick listener for the button
        btn.setOnClickListener {
```

```
        // pendingIntent is an intent for future use i.e after
        // the notification is clicked, this intent will come into action
        val intent = Intent(this, afterNotification::class.java)

        // FLAG_UPDATE_CURRENT specifies that if a previous
        // PendingIntent already exists, then the current one
        // will update it with the latest intent
        // 0 is the request code, using it later with the
        // same method again will get back the same pending
        // intent for future reference
        // intent passed here is to our afterNotification class
        val pendingIntent = PendingIntent.getActivity(this, 0, intent,
PendingIntent.FLAG_UPDATE_CURRENT)

        // RemoteViews are used to use the content of
        // some different layout apart from the current activity layout
        val contentView = RemoteViews(packageName, R.layout.activity_after_notification)

        // checking if android version is greater than oreo(API 26) or not
        if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.O) {
            notificationChannel = NotificationChannel(channelId, description,
NotificationManager.IMPORTANCE_HIGH)
            notificationChannel.enableLights(true)
            notificationChannel.lightColor = Color.GREEN
            notificationChannel.enableVibration(false)
            notificationManager.createNotificationChannel(notificationChannel)

            builder = Notification.Builder(this, channelId)
                .setContent(contentView)
                .setSmallIcon(R.drawable.ic_launcher_background)
                .setLargeIcon(BitmapFactory.decodeResource(this.resources,
R.drawable.ic_launcher_background))
                .setContentIntent(pendingIntent)
```

```
        } else {
            builder = Notification.Builder(this)
                    .setContent(contentView)
                    .setSmallIcon(R.drawable.ic_launcher_background)
                    .setLargeIcon(BitmapFactory.decodeResource(this.resources,
R.drawable.ic_launcher_background))
                    .setContentIntent(pendingIntent)
        }
        notificationManager.notify(1234, builder.build())
    }
  }
}
```

Step4: NotificationView.java

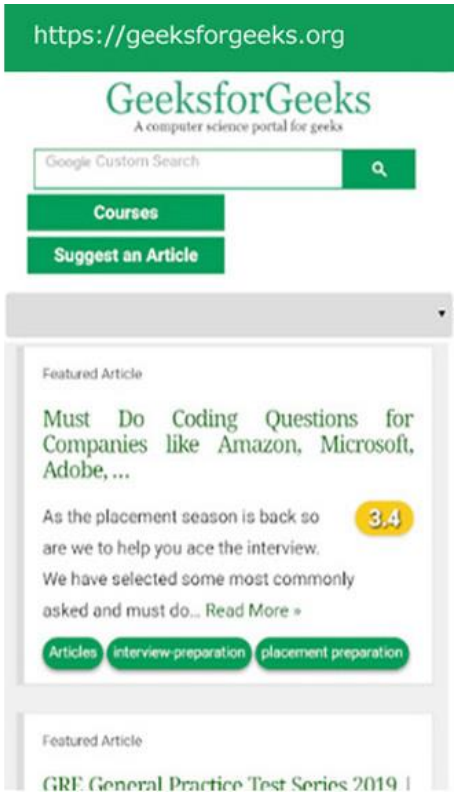
```
package example.javatpoint.com.androidnotification;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;
import android.widget.Toast;

public class NotificationView extends AppCompatActivity {
  TextView textView;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_notification_view);
    textView = findViewById(R.id.textView);
    //getting the notification message
    String message=getIntent().getStringExtra("message");
    textView.setText(message);
  }
}
```

2.7 Invoking Built-in Applications

In Android, you can invoke built-in applications using Intents. Intents are a fundamental part of the Android system that allows components (such as activities, services, and broadcast receivers) to request actions or interactions with other components. There are two types of intents: implicit and explicit.

1.Implicit Intent

Using implicit Intent, components can't be specified. An action to be performed is declared by implicit intent. Then android operating system will filter out components that will respond to the action. **For Example,**



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

Step by Step Implementation

Creating an Android App to Open a Webpage Using Implicit Intent

Step 1: Create a New Project in Android Studio

Step 2: Working with the XML Files

Next, go to the **activity_main.xml file**, which represents the UI of the project. Below is the code for the **activity_main.xml** file. Comments are added inside the code to understand the code in more detail.

Syntax:

android:id="@+id/id_name"

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">

    <EditText
        android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/btn"
        android:text="Search"
```

```
android:onClick="search"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/editText" />


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

Step 3: Working with the MainActivity File

Now, we will create the Backend of the App. For this, Open the MainActivity file and instantiate the component (Button) created in the XML file using the findViewById() method. This method binds the created object to the UI Components with the help of the assigned ID.

Syntax:

```
ComponentType object = (ComponentType) findViewById(R.id.IdOfTheComponent);
```

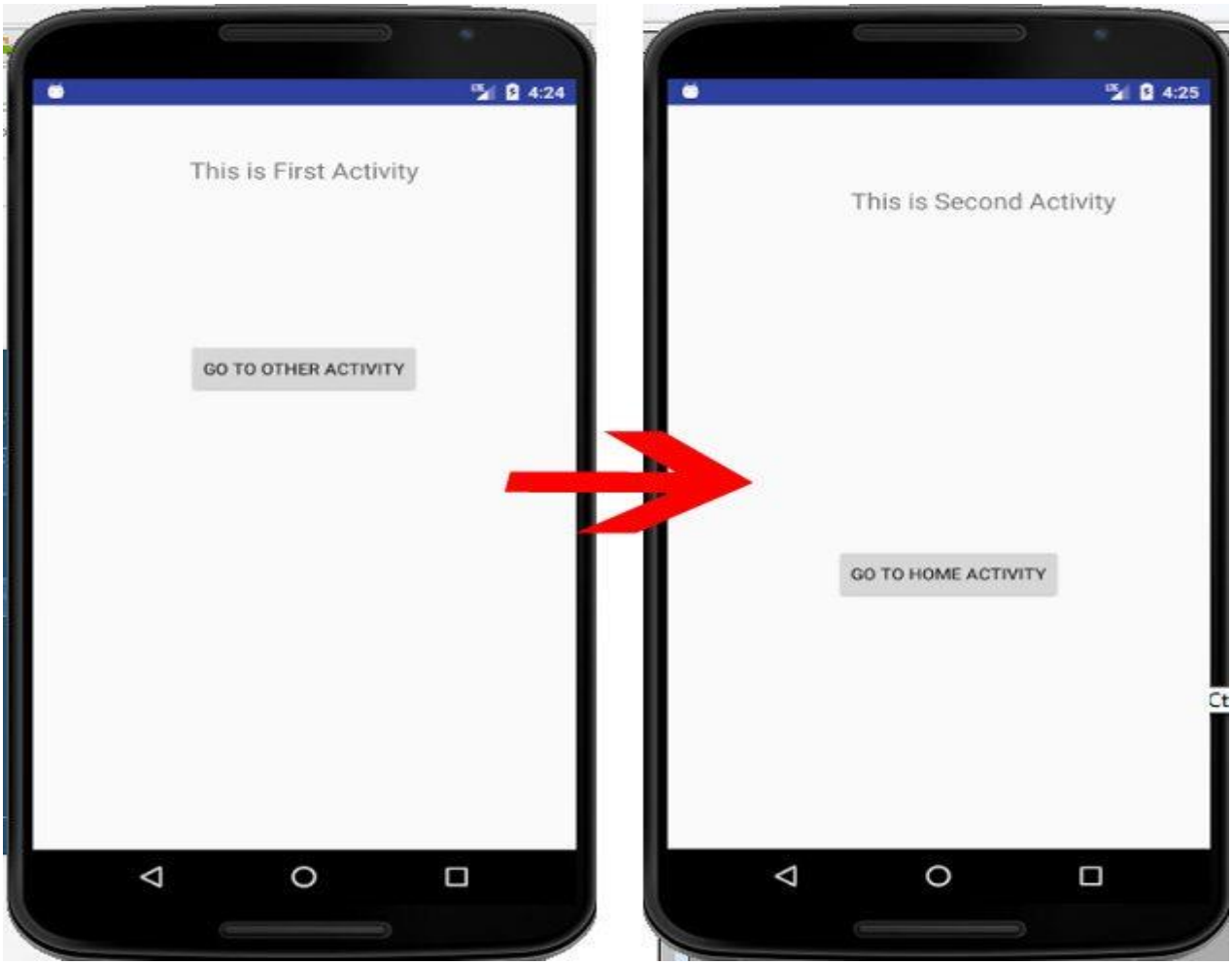
Kotlin

```
import android.content.Intent
import android.net.Uri
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import androidx.appcompat.app.AppCompatActivity


class MainActivity : AppCompatActivity() {
```

```
    lateinit var editText: EditText


    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)


        editText = findViewById(R.id.editText)
    }


    fun search() {
        val url = editText.text.toString()
        val urlIntent = Intent(Intent.ACTION_VIEW, Uri.parse(url))
        startActivity(urlIntent)
    }
}
```

2.Explicit Intent

Using explicit intent any other component can be specified. In other words, the targeted component is specified by explicit intent. So only the specified target component will be invoked. **For Example:**



In the above example, There are two activities (FirstActivity, and SecondActivity). When you click on the 'GO TO OTHER ACTIVITY' button in the first activity, then you move to the second activity. When you click on the 'GO TO HOME ACTIVITY' button in the second activity, then you move to the first activity. This is getting done through Explicit Intent.

Step by Step Implementation

How to create an Android App to move to the next activity using Explicit Intent(with Example)

Step 1: Create a New Project in Android Studio

Step 2: Working with the activity_main.xml File

Next, go to the **activity_main.xml file**, which represents the UI of the project. Below is the code for the **activity_main.xml** file. Comments are added inside the code to understand the code in more detail.

Syntax:

android:id="@+id/id_name"

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">

    <TextView
        android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Welcome to GFG Home Screen"
        android:textAlignment="center"
        android:textSize="28sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
        android:id="@+id/btn1"
        android:text="Go to News Screen"
```

```
        android:onClick="newsScreen"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editText" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

Step 3: Working with the MainActivity File

Now, we will create the Backend of the App. For this, Open the MainActivity file and instantiate the component (Button, TextView) created in the XML file using the findViewById() method. This method binds the created object to the UI Components with the help of the assigned ID.

Syntax:

```
ComponentType object = (ComponentType) findViewById(R.id.IdOfTheComponent);
Intent i = new Intent(getApplicationContext(), <className>);
startActivity(i);
```

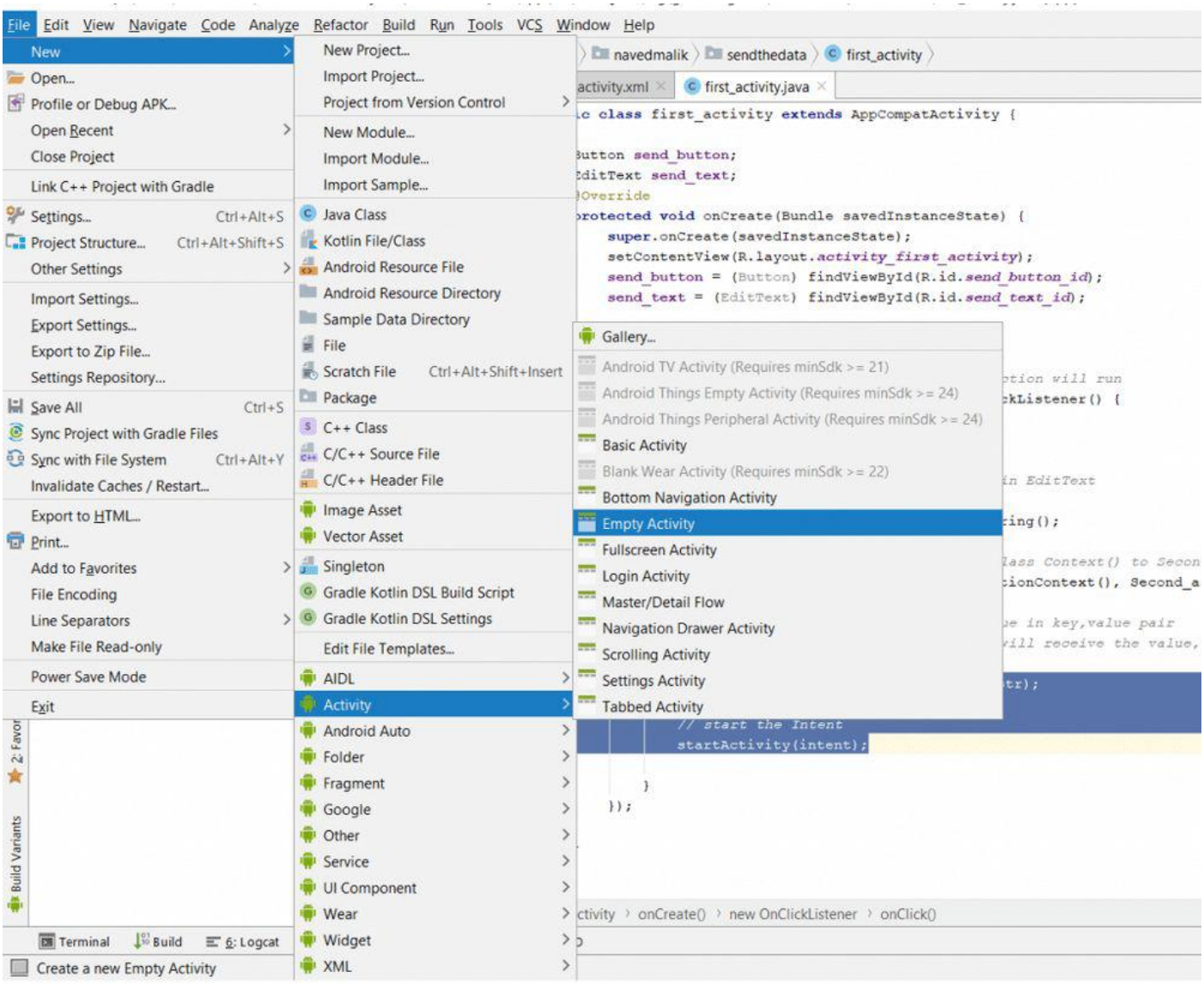
Kotlin

```
import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
    }
```

```
    fun newsScreen() {
        val i = Intent(applicationContext, MainActivity2::class.java)
        startActivity(i)
    }
}
```

Step 4: Working with the activity_main2.xml File

Now we have to create a second activity as a destination activity. The steps to create the second activity are **File > new > Activity > Empty Activity.**



Next, go to the **activity_main2.xml file**, which represents the UI of the project. Below is the code for the **activity_main2.xml** file. Comments are added inside the code to understand the code in more detail.

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">

    <TextView
        android:id="@+id/editText"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Welcome to GFG News Screen"
        android:textAlignment="center"
        android:textSize="28sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.0"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <Button
```

```
        android:id="@+id/btn2"
        android:text="Go to Home Screen"
        android:onClick="homeScreen"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/editText" />
</androidx.constraintlayout.widgext.ConstraintLayout>
```

Step 5: Working with the MainActivity2 File

Now, we will create the Backend of the App. For this, Open the MainActivity file and instantiate the component (Button, TextView) created in the XML file using the findViewById() method. This method binds the created object to the UI Components with the help of the assigned ID.

Syntax:

```
ComponentType object = (ComponentType) findViewById(R.id.IdOfTheComponent);
Intent i = new Intent(getApplicationContext(), <className>);
startActivity(i);
```

Kotlin

```
import android.content.Intent
import android.os.Bundle
import androidx.appcompat.app.AppCompatActivity

class MainActivity2 : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
        setContentView(R.layout.activity_main2)
    }

    fun homeScreen() {
        val i = Intent(applicationContext, MainActivity::class.java)
        startActivity(i)
    }
}
```

UNIT-3

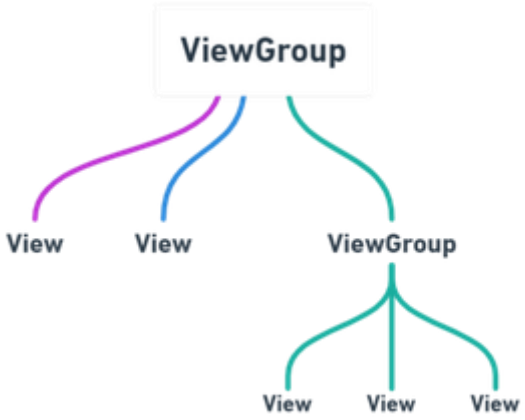
Working with Views

3.1 Working with View Groups

A ViewGroup is a special view that can contain other views. The ViewGroup is the base class for Layouts in android, like LinearLayout , RelativeLayout , FrameLayout etc. In other words, ViewGroup is generally used to define the layout in which views(widgets) will be set/arranged/listed on the android screen.

ViewGroups acts as an invisible container in which other Views and Layouts are placed. Yes, a layout can hold another layout in it, or in other words a **ViewGroup** can have another **ViewGroup** in it.**View:** 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.

ViewGroup: 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

3.2 Designing different types of Views

- **Android WebView:** 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 GridView:** GridView is a ViewGroup that is used to display a scrollable list of items in a grid view of rows and columns.

- **Android RecyclerView:** A RecyclerView is an advanced version of ListView with improved performance.
- **Android CardView:** CardView is a new widget in Android that can be used to display any sort of data by providing a rounded corner layout along with a specific elevation.
- **Android TextView:** Android TextView is simply a view that are used to display the text to the user and optionally allow us to modify or edit it.
- **Android Button:** A Button is a user interface that is used to perform some action when clicked or tapped.
- **Android RadioGroup:** RadioGroup class of Kotlin programming language is used to create a container which holds multiple RadioButtons.
- **Android ToggleButton:** ToggleButton is just like a switch containing two states either ON or OFF which is represented using boolean values true and false respectively.
- **Android CheckBox:** A CheckBox is a special kind of button in Android which has two states either checked or unchecked.

ListView:

Android ListView is a ViewGroup which is used to display the list of items in multiple rows and contains an adapter which automatically inserts the items into the list.

The main purpose of the adapter is to fetch data from an array or database and insert each item that placed into the list for the desired result. So, it is main source to pull data from strings.xml file which contains all the required strings in Kotlin or xml files.

Android Adapter:

Adapter holds the data fetched from an array and iterates through each item in data set and generates the respective views for each item of the list. So, we can say it act as an intermediate between the data sources and adapter views such as ListView, Gridview.

Different Types of Adapter:

- **ArrayAdapter:** It always accepts an Array or List as input. We can store the list items in the strings.xml file also.
- **CursorAdapter:** It always accepts an instance of cursor.
- **SimpleAdapter:** It mainly accepts a static data defined in the resources like array or database.

- **BaseAdapter:** It is a generic implementation for all three adapter types and it can be used in the views according to our requirements.

activity_main.xml file

```
<?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">
    <ListView
        android:id="@+id/userlist"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" >
    </ListView>
</LinearLayout>
```

GridView:

A Grid View is a type of adapter view that is used to display the data in the grid layout format. For setting the data to the grid view adapter is used with the help of the setAdapter() method. This adapter helps to set the data from the database or array list to the items of the grid view. In this article, we will take a look at How to implement Grid View in Android using Kotlin language.

Working with the activity_main.xml file:

```
<?xml version="1.0" encoding="utf-8"?>
<androidx.cardview.widget.CardView
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_gravity="center"
    android:layout_margin="5dp"
    app:cardCornerRadius="5dp"
    app:cardElevation="5dp">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
        android:orientation="vertical">

        <ImageView
            android:id="@+id/idIVCourse"
            android:layout_width="80dp"
            android:layout_height="80dp"
            android:layout_gravity="center"
            android:layout_margin="5dp"
            android:padding="4dp"
            android:src="@mipmap/ic_launcher" />

        <TextView
            android:id="@+id/idTVCourse"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_gravity="center"
            android:layout_margin="5dp"
            android:padding="4dp"
            android:text="@string/app_name"
            android:textAlignment="center"
            android:textColor="@color/black" />

    </LinearLayout>

</androidx.cardview.widget.CardView>
```

RecyclerView:

A RecyclerView is an advanced version of ListView with improved performance. When you have a long list of items to show you can use RecyclerView. It has the ability to reuse its views. In RecyclerView when the View goes out of the screen or not visible to the user it won't destroy it, it will reuse these views. This feature helps in reducing power consumption and providing more responsiveness to the application.

- **onCreateViewHolder():** This function sets the views to display the items.

- **onBindViewHolder():** This function is used to bind the list items to our widgets such as TextView, ImageView, etc.
- **getItemCount():** It returns the count of items present in the list.

Working with 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">

        <androidx.recyclerview.widget.RecyclerView
                android:id="@+id/recyclerview"
                android:layout_width="match_parent"
                android:layout_height="match_parent"
                tools:itemCount="5"
                tools:listitem="@layout/card_view_design" />

</LinearLayout>
```

CardView:

CardView is a new widget in Android that can be used to display any sort of data by providing a rounded corner layout along with a specific elevation. CardView is the view that can display views on top of each other. The main usage of CardView is that it helps to give a rich feel and look to the UI design. This widget can be easily seen in many different Android Apps. CardView can be used for creating items in listview or inside Recycler View. The best part about CardView is that it extends Framelayout and it can be displayed on all platforms of Android.

Some important attributes of Cardview are :

1.cardBackgroundColor : Used for setting up the background-color of the card.

2.cardElevation : Defines the elevation (the process of moving to a higher place or more important position) of the card. It's value should not be quite large else the design may not look good.

3.cardCornerRadius : It sets radius around the corners of the card. More the value of this attribute more circular the edges would appear in the card.

4.cardUseCompactPadding : It has two values true and false. By default, the cardview is set to (0,0) top left corner of the screen. And if this attribute is set to true then the card will set padding for itself so that our UI looks good. This case is helpful in the scenarios when our gravity is not set to center or any other parameters.

Working with XML:

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        tools:context=".MainActivity">

<androidx.cardview.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        app:cardElevation="10dp"
        app:cardCornerRadius="20dp"
        android:layout_margin="10dp"
        app:cardBackgroundColor="@color/white"
        app:cardMaxElevation="12dp"
        app:cardPreventCornerOverlap="true"
        app:cardUseCompatPadding="true"
        >

        <ImageView
                android:layout_width="200dp"
                android:layout_height="200dp"
                android:layout_gravity="center"
                android:src="@drawable/gfgimage"
                android:layout_margin="10dp"
                android:id="@+id/img"
                android:contentDescription="@string/app_name" />
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="@string/app_name"
        android:layout_gravity="bottom|center_horizontal"
        android:layout_marginTop="20dp"
        android:layout_marginBottom="20dp"
        android:textSize="20sp"
        android:textStyle="bold"
        />
</androidx.cardview.widget.CardView>
</RelativeLayout>
```

TextView:

Android TextView is simply a view that are used to display the text to the user and optionally allow us to modify or edit it.

Different attributes of TextView:

Attributes	Description
ndroid:text	Sets text of the Textview
android:id	Gives a unique ID to the Textview
android:cursorVisible	Use this attribute to make cursor visible or invisible. Default value is visible.
android:drawableBottom	Sets images or other graphic assets to below of the Textview.
android:drawableEnd	Sets images or other graphic assets to end of Textview.
android:drawableLeft	Sets images or other graphic assets to left of Textview.
android:drawablePadding	Sets padding to the drawable(images or other graphic assets) in the Textview.
android:autoLink	This attribute is used to automatically detect url or emails and show it as clickable link.
android:autoText	Automatically correct spelling errors in text of the Textview.
android:capitalize	It automatically capitalize whatever the user types in the Textview.
android:drawableRight	Sets drawables to right of text in the Textview.
android:drawableStart	Sets drawables to start of text in the Textview.
android:drawableTop	Sets drawables to top of text in the Textview.

android:ellipsize	Use this attribute when you want text to be ellipsized if it is longer than the Textview width.
android:ems	Sets width of the Textview in ems.
android:gravity	We can align text of the Textview vertically or horizontally or both.
android:height	Use to set height of the Textview.
android:hint	Use to show hint when there is no text.
android:inputType	Use to set input type of the Textview. It can be Number, Password, Phone etc.
android:lines	Use to set height of the Textview by number of lines.
android:maxHeight	Sets maximum height of the Textview.
android:minHeight	Sets minimum height of the Textview.
android:maxLength	Sets maximum character length of the Textview.
android:maxLines	Sets maximum lines Textview can have.
android:minLines	Sets minimum lines Textview can have.
android:maxWidth	Sets maximum width Textview can have.
android:minWidth	Sets minimum lines Textview can have.
android:textAllCaps	Show all texts of the Textview in capital letters.
android:textColor	Sets color of the text.
android:textSize	Sets font size of the text.
android:textStyle	Sets style of the text. For example, bold, italic, bolditalic.
android:typeface	Sets typeface or font of the text. For example, normal, sans, serif etc
android:width	Sets width of the TextView.

activity_main.xml file;

```
<?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:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">

    <TextView
```

```
        android:id="@+id/text_view_id"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content"
        android:text="@string/text_view"
        android:textColor="#008000"
        android:textSize="40dp"
        android:textStyle="bold"/>
</LinearLayout>
```

Button:

A **Button** is a user interface that is used to perform some action when clicked or tapped. It is a very common widget in Android and developers often use it.

XML Attributes of Button:

XML Attributes	Description
android:id	Used to specify the id of the view.
android:text	Used to the display text of the button.
android:textColor	Used to the display color of the text.
android:textSize	Used to the display size of the text.
android:textStyle	Used to the display style of the text like Bold, Italic, etc.
android:textAllCaps	Used to display text in Capital letters.
android:background	Used to set the background of the view.
android:padding	Used to set the padding of the view.
android:visibility	Used to set the visibility of the view.
android:gravity	Used to specify the gravity of the view like center, top, bottom, etc

activity_main.xml file:

```
<?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"
    android:background="#168BC34A"
    tools:context=".MainActivity">
    <Button
        android:id="@+id/button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:background="#4CAF50"
        android:paddingStart="10dp"
        android:paddingEnd="10dp"
        android:text="@string/btn"
        android:textColor="@android:color/background_light"
        android:textSize="24sp"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

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

RadioGroup:

RadioGroup class of Kotlin programming language is used to create a container which holds multiple RadioButtons. The RadioGroup class is beneficial for placing a set of radio buttons inside it because this class adds multiple-exclusion scope feature to the radio buttons. This feature assures that the user will be able to check only one of the radio buttons among all which belongs to a RadioGroup class. If the user checks another radio button, the RadioGroup class unchecks the previously checked radio button.

XML attributes of RadioGroup:

XML Attributes	Description
android:id	To uniquely identify the RadioGroup
android:background	To set a background colour
android:onClick	A method to perform certain action when RadioGroup is clicked
android:onClick	It's a name of the method to invoke when the radio button clicked.
android:visibility	Used to control the visibility i.e., visible, invisible or gone
android:layout_width	To set the width
android:layout_height	To set the height
android:contentDescription	To give a brief description of the view
android:checkedButton	Stores id of child radio button that needs to be checked by default within this radio group
android:orientation	To fix the orientation constant of the view

activity_main.xml file:

```
<?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"
        android:background="#168BC34A"
        tools:context=".MainActivity">
        <TextView
                android:id="@+id/textView"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:fontFamily="@font/roboto"
                android:text="@string/Heading"
                android:textAlignment="center"
                android:textColor="@android:color/holo_green_dark"
                android:textSize="36sp"
                android:textStyle="bold"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"
```

```
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="0.19" />
    <RadioGroup
        android:id="@+id/radioGroup1"
        android:layout_width="0dp"
        android:layout_height="wrap_content"
        android:background="#024CAF50"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toBottomOf="@+id/textView"
        app:layout_constraintVertical_bias="0.24000001">
        <RadioButton
            android:id="@+id/radioButton1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="@font/roboto"
            android:text="@string/RadioButton1"
            android:textSize="24sp" />
        <RadioButton
            android:id="@+id/radioButton2"
```

```
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:fontFamily="@font/roboto"
        android:text="@string/radioButton2"
        android:textSize="24sp" />
    <RadioButton
        android:id="@+id/radioButton3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:fontFamily="@font/roboto"
        android:text="@string/radioButton3"
        android:textSize="24sp" />
    <RadioButton
        android:id="@+id/radioButton4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:fontFamily="@font/roboto"
        android:text="@string/radioButton4"
        android:textSize="24sp" />
    <RadioButton
        android:id="@+id/radioButton5"
```

```
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:fontFamily="@font/roboto"
            android:text="@string/radioButton5"
            android:textSize="24sp" />
    </RadioGroup>
</androidx.constraintlayout.widget.ConstraintLayout>
```

ToggleButton:

ToggleButton is just like a switch containing two states either ON or OFF which is represented using boolean values true and false respectively. ToggleButton unlike switch does not have a slider interface i.e we cannot slide to change the states.

Android ToggleButton XML Attributes:

Attribute	Description
android:id	The id assigned to the toggle button
android:textOff	The text shown on the button when it is not checked
android:textOn	The text shown on the button when it is checked
android:disabledAlpha	The alpha (opacity) to apply to the when disabled.

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">

    <ToggleButton
        android:id="@+id/toggleButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

CheckBox:

A CheckBox is a special kind of button in Android which has two states either checked or unchecked. The Checkbox is a very common widget to be used in Android and a very good example is the "Remember me" option in any kind of Login activity of an app which asks the user to activate or deactivate that service. There are many other uses of the CheckBox widget like offering a list of options to the user to choose from and the options are mutually exclusive i.e., the user can select more than one option. This feature of the CheckBox makes it a better option to be used in designing multiple-choice questions application or survey application in android.

XML attributes of CheckBox:

XML Attributes	Description
android:id	Used to uniquely identify a CheckBox
android:checked	To set the default state of a CheckBox as checked or unchechek
android:background	To set the background color of a CheckBox
android:text	Used to store a text inside the CheckBox
android:fontFamily	To set the font of the text of the CheckBox
android:textSize	To set the CheckBox text size
android:layout_width	To set the CheckBox width
android:layout_height	To set the CheckBox height
android:gravity	Used to adjust the CheckBox text alignment
android:padding	Used to adjust the left, right, top and bottom padding of the CheckBox

activity_main.xml file:

```
<?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"
```

```
android:background="#168BC34A"

tools:context=".MainActivity" >

<TextView

        android:id="@+id/textView"

        android:layout_width="wrap_content"

        android:layout_height="wrap_content"

        android:fontFamily="@font/roboto"

        android:text="@string/Heading"

        android:textAlignment="center"

        android:textColor="@android:color/holo_green_dark"

        android:textSize="36sp"

        android:textStyle="bold"

        app:layout_constraintBottom_toBottomOf="parent"

        app:layout_constraintEnd_toEndOf="parent"

        app:layout_constraintStart_toStartOf="parent"

        app:layout_constraintTop_toTopOf="parent"

        app:layout_constraintVertical_bias="0.17000002" />

<LinearLayout

        android:id="@+id/checkBox_container"

        android:layout_width="0dp"

        android:layout_height="wrap_content"
```

```
android:orientation="vertical"
app:layout_constraintBottom_toBottomOf="parent"
app:layout_constraintEnd_toEndOf="parent"
app:layout_constraintStart_toStartOf="parent"
app:layout_constraintTop_toBottomOf="@+id/textView"
app:layout_constraintVertical_bias="0.18">
<CheckBox
        android:id="@+id/checkBox"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:fontFamily="@font/roboto"
        android:text="@string/checkBox1_text"
        android:textSize="18sp"
        android:padding="7dp"/>
<CheckBox
        android:id="@+id/checkBox2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:fontFamily="@font/roboto"
        android:text="@string/checkBox2_text"
        android:textSize="18sp"
```

```
        android:padding="7dp"/>
    <CheckBox
        android:id="@+id/checkBox3"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:fontFamily="@font/roboto"
        android:text="@string/checkBox3_text"
        android:textSize="18sp"
        android:padding="7dp"/>
    <CheckBox
        android:id="@+id/checkBox4"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:fontFamily="@font/roboto"
        android:text="@string/checkBox4_text"
        android:textSize="18sp"
        android:padding="7dp"/>
    <CheckBox
        android:id="@+id/checkBox5"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
```

```
                android:fontFamily="@font/roboto"
                android:text="@string/checkBox5_text"
                android:textSize="18sp"
                android:padding="7dp"/>
        </LinearLayout>

        <Button
                android:id="@+id/submitButton"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:background="#AB4CAF50"
                android:fontFamily="@font/roboto"
                android:text="@string/submitButton"
                android:textSize="18sp"
                android:textStyle="bold"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toBottomOf="@+id/checkBox_container"
                app:layout_constraintVertical_bias="0.23000002" />
</androidx.constraintlayout.widget.ConstraintLayout>
```

3.3 Implementing Screen Orientation

Screen Orientation, also known as **screen rotation**, is the attribute of activity element in android. When screen orientation change from one state to other, it is also known as **configuration change**.

States of Screen Orientation :

There are various possible **screen orientation states** for any android application, such as:

- ActivityInfo.SCREEN_ORIENTATION_LANDSCAPE
- ActivityInfo.SCREEN_ORIENTATION_PORTRAIT
- ActivityInfo.SCREEN_ORIENTATION_UNSPECIFIED
- ActivityInfo.SCREEN_ORIENTATION_USER
- ActivityInfo.SCREEN_ORIENTATION_SENSOR
- ActivityInfo.SCREEN_ORIENTATION_BEHIND
- ActivityInfo.SCREEN_ORIENTATION_NOSENSOR
- ActivityInfo.SCREEN_ORIENTATION_SENSOR_LANDSCAPE
- ActivityInfo.SCREEN_ORIENTATION_SENSOR_PORTRAIT
- ActivityInfo.SCREEN_ORIENTATION_REVERSE_PORTRAIT

The initial orientation of the Screen has to be defined in the AndroidManifest.xml file.

Syntax:

```
<activity android:name="package_name.Your_ActivityName"
        android:screenOrientation="orientation_type">
</activity>
```

Example:

```
  android:screenOrientation="orientation_type">
```

How to change Screen orientation?

We will create **two activities** of different screen orientation.

- The first activity will be as "**portrait**" orientation and
- Second activity as "**landscape**" orientation state.

Step-by-Step demonstration:

- **Creating the activities:** There will be two activities and hence two XML files, one for each activity.
    1. **activity_main.xml**: XML file for first activity consist of constraint layout with Button and Text View in it. This activity is in Landscape state.
    2. **activity_next.xml**: XML file for second activity consist of constraint layout with Text View in it. This activity is in Landscape state.

activity_main.xml:

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

```
<android.support.constraint.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="com.screenlayout.screenorientation.MainActivity">
        <Button
                android:id="@+id/b1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="Next Activity"
                android:layout_marginTop="100dp"
                android:onClick="onClick"
                android:layout_marginBottom="10dp"
                app:layout_constraintBottom_toBottomOf="parent"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintVertical_bias="0.613"
                app:layout_constraintHorizontal_bias="0.612"
                app:layout_constraintStart_toStartOf="parent"
                app:layout_constraintTop_toBottomOf="@+id/tv1"
                />
        <TextView
                android:text="Potrait orientation"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerHorizontal="true"
                android:layout_marginTop="124dp"
                android:textSize="25dp"
                app:layout_constraintEnd_toEndOf="parent"
                app:layout_constraintHorizontal_bias="0.502"
                app:layout_constraintStart_toStartOf="parent"
```

```
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

Activity_next.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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_height="match_parent"
    android:layout_width="match_parent"
    tools:context="com.screenlayout.screenorientation.NextActivity">



    <TextView
        android:id="@+id/tv"
        android:text="Landscape orientation"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="170dp"
        android:textSize="22dp"
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
        app:layout_constraintHorizontal_bias="0.502"
        app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

- **Creating the Java file:** There will be two activities and hence two Java files, one for each activity.
    1. **MainActivity.java**: Java file for Main Activity, in which *setOnClick() listener* is attached to the button to launch next activity with different orientation.
    2. **NextActivity.java**: Java file for Next Activity which is in Landscape mode.

Mainactivity.java

```
package com.geeksforgeeks.screenorientation;

import android.support.v7.app.AppCompatActivity;
import android.content.Intent;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

    // declare button variable
    Button button;

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

        // initialise button with id
        button = findViewById(R.id.b1);
    }

    // onClickListener attached to button
    // to send intent to next activity
    public void onClick(View v)
    {
        // Create instance of intent and
        // startActivity with intent object
        Intent intent
            = new Intent(
                MainActivity.this,
```

```
                        NextActivity.class);
            startActivity(intent);
        }
}
```

Nextactivity.java:

```
package com.geeksforgeeks.screenorientation;

import android.support.v7.app.AppCompatActivity;
import android.content.Intent;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {

        // declare button variable
        Button button;

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

            // initialise button with id
            button = findViewById(R.id.b1);
        }

        // onClickListener attached to button
        // to send intent to next activity
        public void onClick(View v)
        {
            // Create instance of intent and
```

```
        // startActivity with intent object
        Intent intent
                = new Intent(
                        MainActivity.this,
                        NextActivity.class);
        startActivity(intent);
    }
}
```

Updating the AndroidManifest file: In AndroidManifest.xml file, add the **screenOrientation state** in activity along with its orientation. Here, we provide "**portrait**" orientation for MainActivity and "**landscape**" for NextActivity.

AndroidManifest.xml:

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

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <!-Define potrait orientation for Main activity-->
        <activity
            android:name="com.geeksforgeeks.screenorientation.MainActivity"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <!--Define landscape orientation for NextActivity-->
```

```
        <activity android:name=".NextActivity"
                android:screenOrientation="landscape">
        </activity>
    </application>
</manifest>
```

3.4. Designing the Views Programmatically

We can create or instantiate UI elements or widgets during runtime by using the custom View and ViewGroup objects programmatically in the Kotlin file. Below is the example of creating a layout using LinearLayout to hold an EditText and a Button in an activity programmatically.

Kotlin

```
import android.os.Bundle
import android.widget.Button
import android.widget.EditText
import android.widget.LinearLayout
import android.widget.Toast
import android.appcompat.app.AppCompatActivity

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // create the button
        val showButton = Button(this)
        showButton.setText("Submit")

        // create the editText
        val editText = EditText(this)

        val linearLayout = findViewById<LinearLayout>(R.id.l_layout)
        linearLayout.addView(editText)
        linearLayout.addView(showButton)
```

```
        // Setting On Click Listener
        showButton.setOnClickListener
        {
            // Getting the user input
            val text = editText.text

            // Showing the user input
            Toast.makeText(this, text, Toast.LENGTH_SHORT).show()
        }
    }
}
```

Different Attribute of the Layouts

XML attributes	Description
android:id	Used to specify the id of the view.
android:layout_width	Used to declare the width of View and ViewGroup elements in the layout.
android:layout_height	Used to declare the height of View and ViewGroup elements in the layout.
android:layout_marginLeft	Used to declare the extra space used on the left side of View and ViewGroup elements.
android:layout_marginRight	Used to declare the extra space used on the right side of View and ViewGroup elements.
android:layout_marginTop	Used to declare the extra space used in the top side of View and ViewGroup elements.

XML attributes	Description
android:layout_marginBottom	Used to declare the extra space used in the bottom side of View and ViewGroup elements.
android:layout_gravity	Used to define how child Views are positioned in the layout.

UNIT-4

Working with Graphics and Animation

4.1 Working with graphics

The Android framework offers a variety of graphics rendering APIs for 2D and 3D that interact with manufacturer implementations of graphics drivers, so it is important to have a good understanding of how those APIs work at a higher level.

- Android provides a huge set of 2D-drawing APIs that allow you to create graphics.
- Android has visually appealing graphics and mind-blowing animations.
- The Android framework provides a rich set of powerful APIS for applying animation to UI elements and graphics as well as drawing custom 2D and 3D graphics.

Canvas: Android graphics provides low-level graphics tools such as canvases, color, filters, points, and rectangles which handle drawing to the screen directly. The Android framework provides a set of 2D-DRAWING APIs which allows user to provide their custom graphics onto a canvas or to modify existing views to customize their look and feel.

There are two ways to draw 2D graphics,

1. Draw your animation into a View object from your layout.
2. Draw your animation directly on a Canvas.

Some of the important methods of Canvas Class are as follows

1. drawText()
2. drawRoundRect()
3. drawCircle()
4. drawRect()
5. drawBitmap()
6. drawARGB()

You can use these methods in the onDraw() method to create your custom user interface.

Drawing an animation with a View is the best option for drawing simple graphics that do not need to change dynamically and are not part of a performance-intensive game. It is used when the user wants to display a static graphic or predefined animation.

Drawing an animation with Canvas is a better option when your application needs to re-draw itself regularly. For example, video games should be drawn to the Canvas on their own.

Example:

Step1: Create a New Project in Android Studio

Step2: 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. Add an image as shown below.

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

   <ImageView
      android:id="@+id/image_view_1"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      tools:ignore="ContentDescription"
      android:background="@color/black"/>

</RelativeLayout>
```

Step3: Working with the MainActivity.kt file

```
package org.main_act.drawlines

import android.annotation.SuppressLint
import android.graphics.Bitmap
import android.graphics.Canvas
import android.graphics.Color
import android.graphics.Paint
import android.os.Build
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.MotionEvent
```

```
import android.view.View
import android.widget.ImageView
import androidx.annotation.RequiresApi

class MainActivity : AppCompatActivity(), View.OnTouchListener {

    // Declaring ImageView, Bitmap, Canvas, Paint,
    // Down Coordinates and Up Coordinates
    private lateinit var mImageView: ImageView
    private lateinit var bitmap: Bitmap
    private lateinit var canvas: Canvas
    private lateinit var paint: Paint
    private var downX = 0f
    private var downY = 0f
    private var upX = 0f
    private var upY = 0f

    @RequiresApi(Build.VERSION_CODES.R)
    @SuppressLint("ClickableViewAccessibility")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // Initializing the ImageView
        mImageView = findViewById(R.id.image_view_1)

        // Getting the current window dimensions
        val currentDisplay = windowManager.currentWindowMetrics
        val dw = currentDisplay.bounds.width()
        val dh = currentDisplay.bounds.height()

        // Creating a bitmap with fetched dimensions
```

```
    bitmap = Bitmap.createBitmap(dw, dh, Bitmap.Config.ARGB_8888)

    // Storing the canvas on the bitmap
    canvas = Canvas(bitmap)

    // Initializing Paint to determine
    // stoke attributes like color and size
    paint = Paint()
    paint.color = Color.RED
    paint.strokeWidth = 10F

    // Setting the bitmap on ImageView
    mImageView.setImageBitmap(bitmap)

    // Setting onTouchListener on the ImageView
    mImageView.setOnTouchListener(this)
}

// When Touch is detected on the ImageView,
// Initial and final coordinates are recorded
// and a line is drawn between them.
// ImagView is updated
@SuppressLint("ClickableViewAccessibility")
override fun onTouch(v: View?, event: MotionEvent?): Boolean {
    when (event!!.action) {
        MotionEvent.ACTION_DOWN -> {
            downX = event.x
            downY = event.y
        }

        MotionEvent.ACTION_UP -> {
            upX = event.x
```

```
            upY = event.y
            canvas.drawLine(downX, downY, upX, upY, paint)
            mImageView.invalidate()
        }
    }
    return true
  }
}
```

4.2 Animation

Animation is the process of adding a motion effect to any view, image, or text. With the help of an animation, you can add motion or change the shape of a specific view. Animation in Android is generally used to give your UI a rich look and feel.

Following are the three animation systems used in Android applications:

- Property Animation
- View Animation
- Drawable Animation

Property Animation: Property animation is the preferred method of animation in Android. This animation is a robust framework that lets you animate any properties of any objects, view or non-view objects. The android.animation provides classes that handle property animation.

Property Animation is one of the robust frameworks that allows animating for almost everything. This is one of the powerful and flexible animations which was introduced in Android 3.0. Property animation can be used to add any animation in the CheckBox, RadioButtons, and widgets other than any view.

View Animation: View Animation is also called Tween Animation. The android.view.animation provides classes that handle view animation.. This animation can be used to animate the content of a view. .It is limited to simple transformations such as moving, re-sizing and rotation, but not its background color.

View Animation can be used to add animation to a specific view to perform tweened animation on views. Tweened animation calculates animation information such as size, rotation, start point, and endpoint. These animations are slower and less flexible. An example of View animation can be used if we want to expand a specific layout in that place we can use View Animation. An example of View Animation can be seen in Expandable RecyclerView.

Drawable Animation: Drawable animation is implemented using the AnimationDrawable class.. This animation works by displaying a running sequence of 'Drawable' resources that is images, frame by frame inside a view object.

Drawable Animation is used if you want to animate one image over another. The simple way to understand is to animate drawables is to load the series of drawables one after another to create an animation. A simple example of drawable animation can be seen in many apps Splash screen on apps logo animation.

Example:

Step1: Create New Project

1. Open Android Studio
2. Go to File => New => New Project.
3. Then, select Empty Activity and click on next
   - Write the application name as DynamicEditTextKotlin
   - Select the minimum SDK you need, here we have selected 21 as the minimum SDK
   - Choose the language as Kotlin and click on the finish button.

Step2: Modify activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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"
    tools:context=".MainActivity">
    <TextView
        android:id="@+id/textView"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:layout_above="@+id/linearLayout"
        android:gravity="center"
```

```
    android:text="It’s an animation time"
     android:textSize="32sp"
    android:textColor="@color/colorPrimaryDark"
    android:textStyle="bold" />
<LinearLayout
    android:id="@+id/linearLayout"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:orientation="vertical">
    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:orientation="horizontal"
        android:weightSum="2">
        <Button
            android:id="@+id/fade_in"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="Fade In"
            android:textAllCaps="false" />
        <Button
            android:id="@+id/fade_out"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="Fade Out"
            android:textAllCaps="false" />
    </LinearLayout>
    <LinearLayout
        android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:weightSum="2">
    <Button
        android:id="@+id/zoom_in"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:text="Zoom In"
        android:textAllCaps="false" />
    <Button
        android:id="@+id/zoom_out"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:text="Zoom Out"
        android:textAllCaps="false" />
</LinearLayout>
<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:orientation="horizontal"
    android:weightSum="2">
    <Button
        android:id="@+id/slide_down"
        android:layout_width="0dp"
        android:layout_height="match_parent"
        android:layout_weight="1"
        android:text="Slide Down"
        android:textAllCaps="false" />
    <Button
        android:id="@+id/slide_up"
```

```
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout_weight="1"
            android:text="Slide Up"
            android:textAllCaps="false" />
        </LinearLayout>
        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:orientation="horizontal"
            android:weightSum="2">
            <Button
                android:id="@+id/bounce"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:text="Bounce"
                android:textAllCaps="false" />
            <Button
                android:id="@+id/rotate"
                android:layout_width="0dp"
                android:layout_height="match_parent"
                android:layout_weight="1"
                android:text="Rotate"
                android:textAllCaps="false" />
        </LinearLayout>
    </LinearLayout>
</RelativeLayout>
```

bounce.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<set
    xmlns:android="http://schemas.android.com/apk/res/android"
```

```
    android:interpolator="@android:anim/linear_interpolator"
    android:fillAfter="true">
    <translate
        android:fromYDelta="100%"
        android:toYDelta="-20%"
        android:duration="300" />
    <translate
        android:startOffset="500"
        android:fromYDelta="-20%"
        android:toYDelta="10%"
        android:duration="150" />
    <translate
        android:startOffset="1000"
        android:fromYDelta="10%"
        android:toYDelta="0"
        android:duration="100" />
</set>
```

rotate.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<rotate xmlns:android="http://schemas.android.com/apk/res/android"
    android:duration="1000"
    android:fromDegrees="0"
    android:interpolator="@android:anim/linear_interpolator"
    android:pivotX="50%"
    android:pivotY="50%"
    android:startOffset="0"
    android:toDegrees="360" />
```

Step3: Create MainActivity.kt file

```
package net.animate.animationsinkotlin

import androidx.appcompat.app.AppCompatActivity
```

```
import android.os.Bundle
import android.os.Handler
import android.view.View
import android.view.animation.AnimationUtils
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        bounce.setOnClickListener {
            val animationBounce = AnimationUtils.loadAnimation(this, R.anim.bounce)
            textView.startAnimation(animationBounce)
        }
        rotate.setOnClickListener {
            val animationRotate = AnimationUtils.loadAnimation(this, R.anim.rotate)
            textView.startAnimation(animationRotate)
        }
    }
}
```

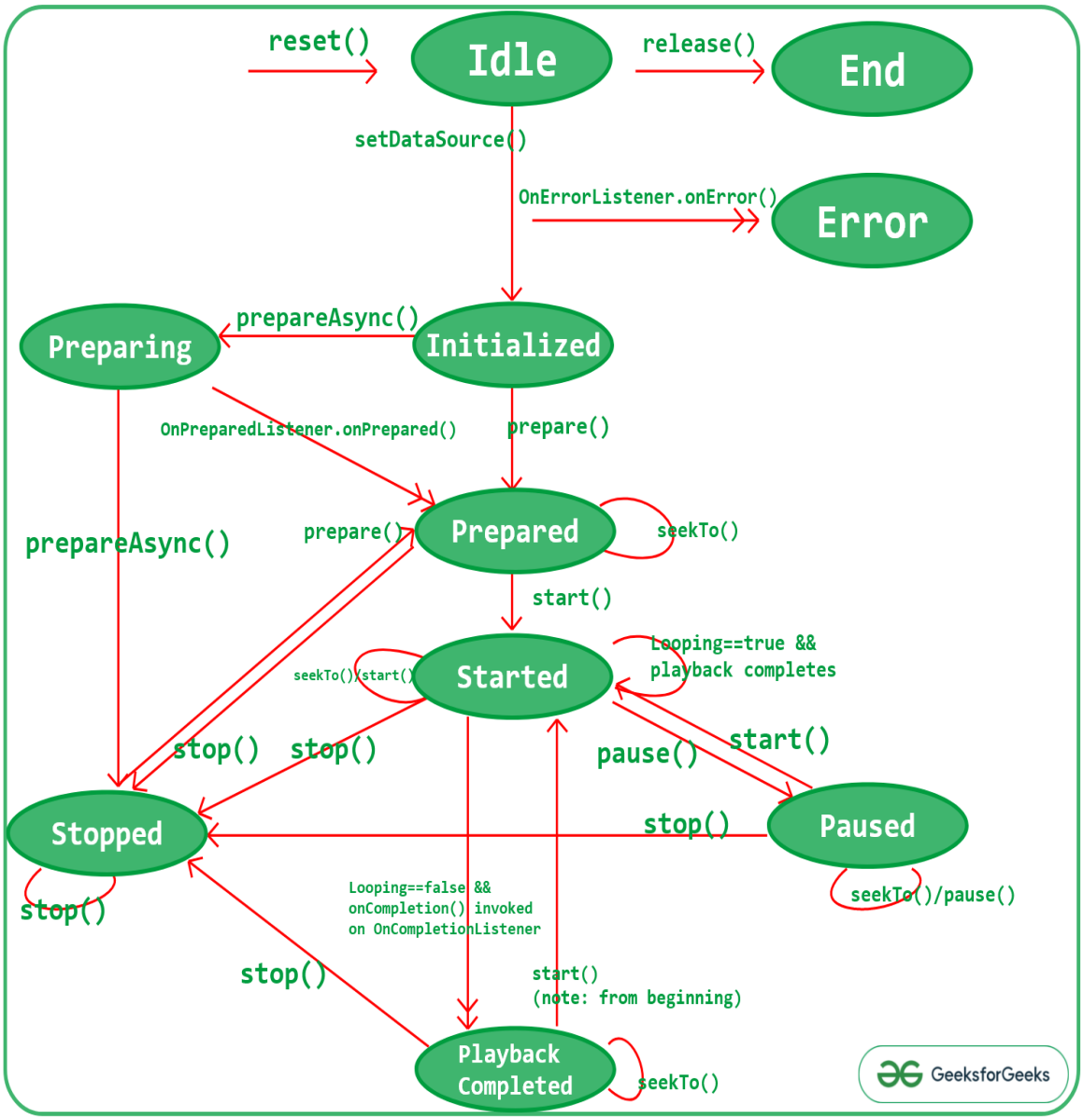
4.3 Media Player

MediaPlayer Class in Android is used to play media files. Those are Audio and Video files. It can also be used to play audio or video streams over the network. So in this article, the things discussed are:

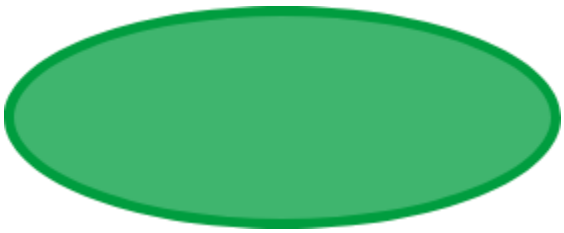
- MediaPlayer State diagram
- Creating a simple audio player using MediaPlayer API. Have a look at the following image. Note that we are going to implement this project using the Kotlin language.

State Diagram of the MediaPlayer Class

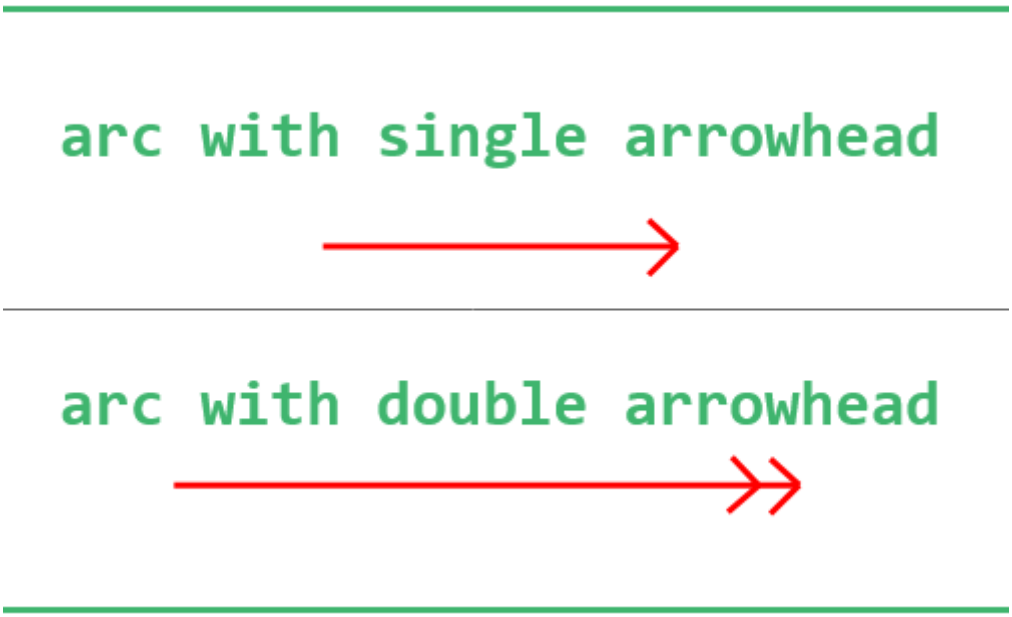
- The playing of the audio or video file using MediaPlayer is done using a state machine.
- The following image is the MediaPlayer state diagram.



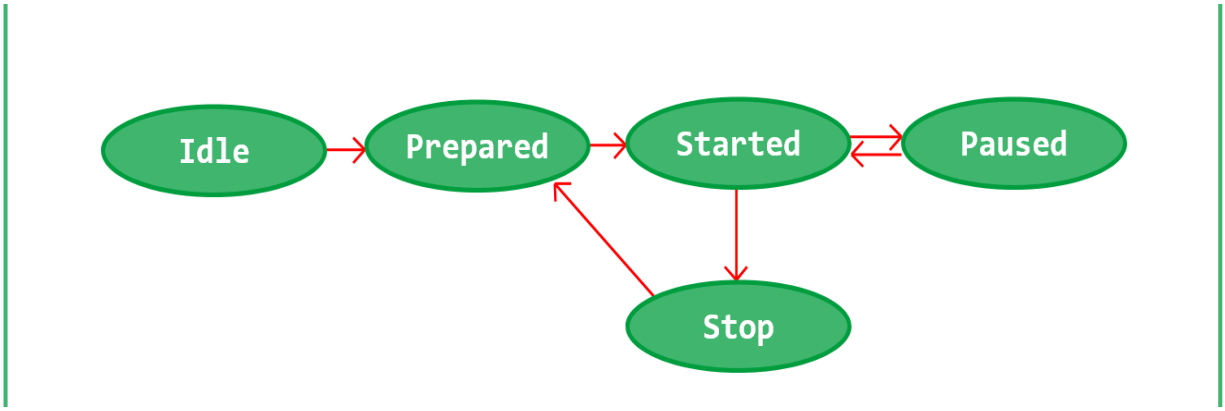
- In the above MediaPlayer state diagram, the oval shape represents the state of the MediaPlayer instance resides in.



- There are two types of arcs showing in the state diagram. One with the single arrowhead represents the synchronous method calls of the MediaPlayer instance and one with the double arrowhead represents the asynchronous calls.



- The release method which is one of the important element in the MediaPlayer API. This helps in releasing the Memory resources allocated for the Mediaplayer instance when it is not needed anymore. Refer to How to Clear or Release Audio Resources in Android? to know how the memory allocated by the Mediaplayer can be released. So that the memory management is done accordingly.
- If the stop() method is called using the Mediaplayer instance, then it needs to be prepared for the next playback.
- The MediaPlayer can be moved to the specific time position using the **seekTo()** method so that the MediaPlayer instance can continue playing the Audio or Video playback from that specified position.
- The focus of the audio playback should be managed accordingly using the AudioManager service which is discussed in the article How to Manage Audio Focus in Android?.
- The following image is the summarised version of the MediaPlayer state diagram.



Steps to create a simple MediaPlayer in Android

Step1: Create an empty activity project

- Create an empty activity Android Studio project. And select Kotlin as a programming language.

Step2: Create a raw resource folder

- Create a raw resource folder under the res folder and copy one of the .mp3 file extensions.

Step3: Working with the activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    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"
    tools:context=".MainActivity"
    tools:ignore="HardcodedText">

    <TextView
        android:id="@+id/headingText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="32dp"
        android:text="MEDIA PLAYER"
        android:textSize="18sp"
        android:textStyle="bold" />

    <LinearLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/headingText"
        android:layout_marginTop="16dp"
        android:gravity="center_horizontal">
```

```
    <Button
        android:id="@+id/stopButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:backgroundTint="@color/colorPrimary"
        android:text="STOP"
        android:textColor="@android:color/white" />

    <Button
        android:id="@+id/playButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginEnd="8dp"
        android:backgroundTint="@color/colorPrimary"
        android:text="PLAY"
        android:textColor="@android:color/white" />

    <Button
        android:id="@+id/pauseButton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:backgroundTint="@color/colorPrimary"
        android:text="PAUSE"
        android:textColor="@android:color/white" />

  </LinearLayout>

</RelativeLayout>
```

Step4: Working with the MainActivity.kt file

- The MediaPlayer instance needs the attributes to be set before playing any audio or video file.
- Invoke the following inside the MainActivity.kt file. Comments are added for better understanding.

```
import android.media.MediaPlayer
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Button

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)

        // create an instance of mediplayer for audio playback
        val mediaPlayer: MediaPlayer = MediaPlayer.create(applicationContext, R.raw.music)

        // register all the buttons using their appropriate IDs
        val bPlay: Button = findViewById(R.id.playButton)
        val bPause: Button = findViewById(R.id.pauseButton)
        val bStop: Button = findViewById(R.id.stopButton)

        //Handle the start button to
          //Start the audio playback
        bPlay.setOnClickListener {
            // start method is used to start
              // playing the audio file
            mediaPlayer.start()
        }

        //Handle the pause button to put the
```

```
        // MediaPlayer instance at the Pause state
        bPause.setOnClickListener {
            // pause() method can be used to
            //Pause the media player instance
            mediaPlayer.pause()
        }

        //Handle the stop button to stop playing
        // and prepare the MediaPlayer instance
        //For the next instance of play
        bStop.setOnClickListener {
            // stop() method is used to completely
            //Stop playing the media player instance
            MediaPlayer.stop()

            //After stopping the MediaPlayer instance
            //It again needs to be prepared
            //For the next instance of playback
            MediaPlayer.prepare()
        }
    }
}
```

UNIT-5

Events Database and Connectivity

5.1 Handling UI Events

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.Android provides a huge set of 2D-drawing APIs that allow you to create graphics.

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.

Event Listeners and their respective event handlers

1. **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().
2. **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().
3. **OnMenuItemClickListener():** This method is called when the user selects a menu item. Event handler used for this is onMenuItemClick().
4. **OnMenuItemClickListener():** This method is called when the user selects a menu item. Event handler used for this is onMenuItemClick().

There are various other event listeners available which can be used for different requirements and can be found in the official documentation.

Example:

Step1: Create a New Project in Android Studio

Step2: Working with the activity_main.xml file

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

   <ImageView
      android:id="@+id/image_view_1"
      android:layout_width="match_parent"
      android:layout_height="match_parent"
      tools:ignore="ContentDescription"
      android:background="@color/black"/>

</RelativeLayout>
```

Step3: Working with the MainActivity.kt file

```
package com.latihan.darmanto.radiobutton
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.RadioButton
import android.widget.TextView
class MainActivity : AppCompatActivity() {
        override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
 }
```

```
fun onRadioButtonClicked(view: View) {
        var textView: TextView = findViewById(R.id.textView)
        if (view is RadioButton) {
        // Is the button now checked?
        val checked = view.isChecked
        // Check which radio button was clicked
        when (view.getId()) {
        R.id.radio_pirates ->
        if (checked) {
        // Pirates are the best
        textView.setText("Pirates")
        }
        R.id.radio_ninjas ->
        if (checked) {
        // Ninjas rule
        textView.setText("Ninjas")
            }
        }
    }
  }
}
```

5.2 Building data with adapter view class

In Android, whenever we want to bind some data which we get from any data source (e.g. ArrayList, HashMap, SQLite, etc.) with a UI component(e.g. ListView, GridView, etc.) then **Adapter** comes into the picture. Basically **Adapter** acts as a bridge between the UI component and data sources. Here **Simple Adapter** is one type of **Adapter**. It is basically an easy adapter to map static data to views defined in our XML file(UI component) and is used for customization of List or Grid items. Here we use an ArrayList of Map (e.g. hashmap, mutable map, etc.) for data-backing. Each entry in an ArrayList is corresponding to one row of a list. The Map contains the data for each row. Now to display the row we need a view for which we used to specify a custom list item file (an XML file).

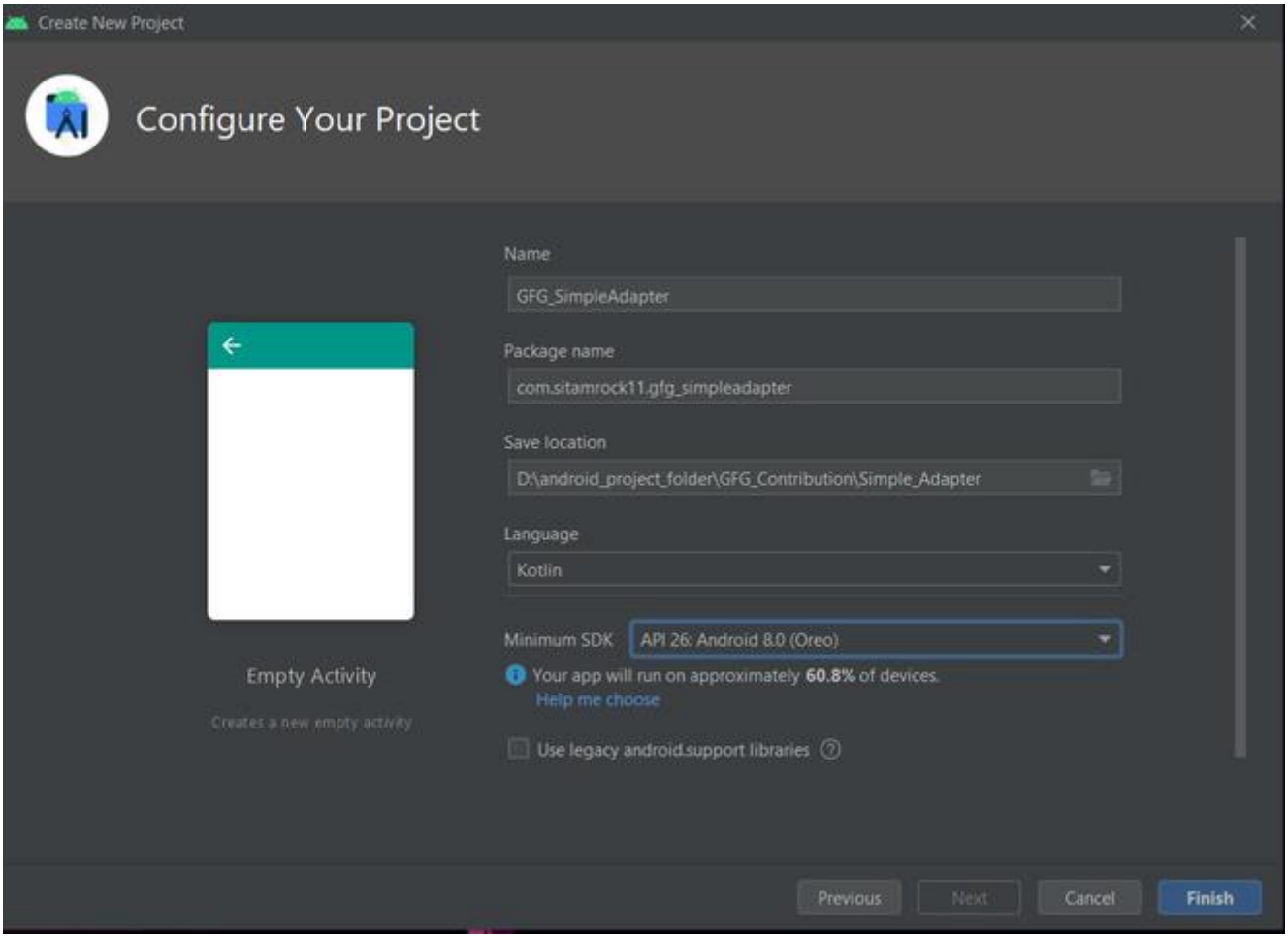
Example:

A sample image is given below to get an idea about what we are going to do in this article. In this project, we are going to make this application which has a list of some fruits and in each row of the list has a fruit

image and name. Note that we are going to implement this same project in both Kotlin and Java languages. Now you choose your preferred language.

Step1: Create New Project

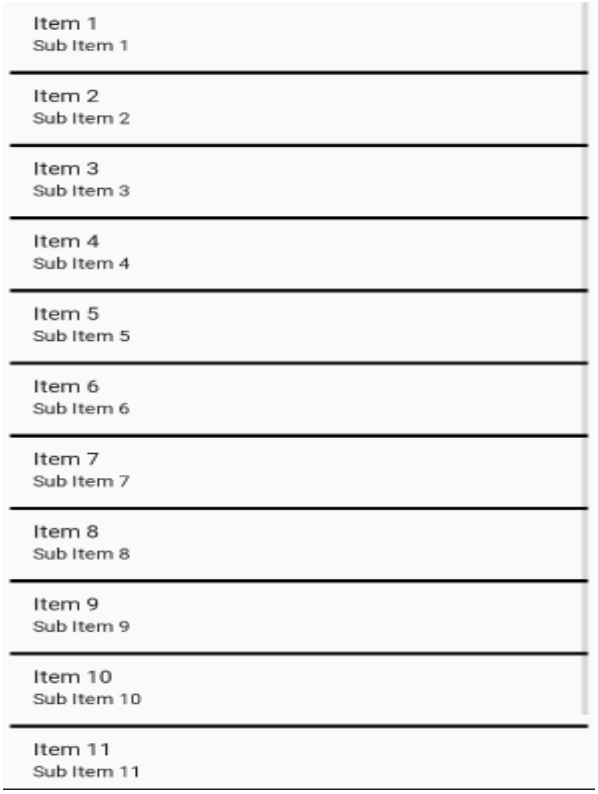
1. Open Android Studio
2. Go to File => New => New Project.
3. Then, select Empty Activity and click on next



Step2: Modify activity_main.xml file

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
    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"
    tools:context=".MainActivity">

    <!--Creating a ListView-->
```

```
    <ListView
        android:id="@+id/listView"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:divider="#000000"
        android:dividerHeight="3dp"
        android:padding="5dp" />


</RelativeLayout>
```

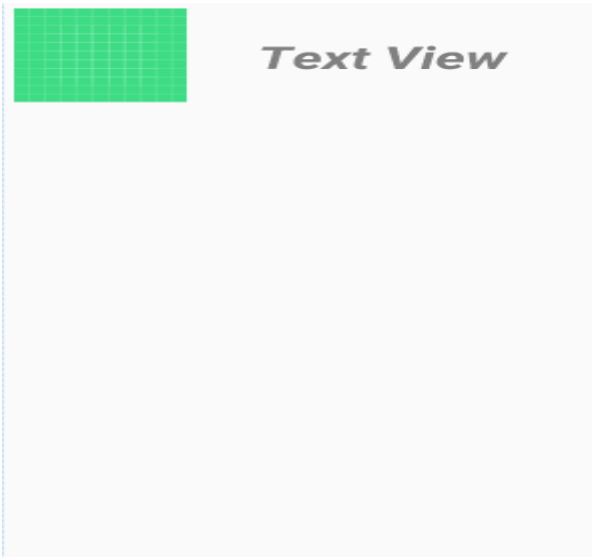


Create another XML file (named list_row_items) and create UI for each row of the ListView:

Below is the code for the list_row_items.xml file.

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


    <!--Creating a ImageView-->
    <ImageView
```

```
    android:id="@+id/imageView"
    android:layout_width="120dp"
    android:layout_height="120dp"
    android:layout_margin="10dp"
    android:scaleType="fitCenter"
    android:src="@drawable/ic_launcher_background"  />

  <!--Creating a TextView-->
  <TextView
    android:id="@+id/textView"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_marginTop="40dp"
    android:layout_marginRight="20dp"
    android:layout_toRightOf="@+id/imageView"
    android:gravity="center"
    android:padding="5dp"
    android:text="Text View"
    android:textColor="#808080"
    android:textSize="40sp"
    android:textStyle="bold|italic" />

</RelativeLayout>
```



Step3: Create MainActivity.kt file

```
import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.ListView
import android.widget.SimpleAdapter
import java.util.ArrayList
import java.util.HashMap

class MainActivity : AppCompatActivity() {

    private lateinit var listView:ListView
    // creating  a String type array
      // (fruitNames) which contains
    // names of different fruits' images
    private val fruitNames=arrayOf("Banana","Grape","Guava","Mango","Orange","Watermelon")

    // creating an Integer type array (fruitImageIds) which
    // contains IDs of different fruits' images
    private val fruitImageIds=arrayOf(R.drawable.banana,
                        R.drawable.grape,
                        R.drawable.guava,
                        R.drawable.mango,
                        R.drawable.orange,
                        R.drawable.watermelon)

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_main)
        // ViewBinding the ListView  of activity_main.xml file
        // with this kotlin code in MainActivity.kt
        listView=findViewById(R.id.listView)
        // creating an ArrayList of HashMap.The kEY of the HashMap is
```

```
    // a String and VALUE is of any datatype(Any)
    val list=ArrayList<HashMap<String,Any>>()

    // By a for loop, entering different types of data in HashMap,
    // and adding this map including it's datas into the ArrayList
    // as list item and this list is the second parameter of the SimpleAdapter
    for(i in fruitNames.indices){
        val map=HashMap<String,Any>()
        // Data entry in HashMap
        map["fruitName"] = fruitNames[i]
        map["fruitImage"]=fruitImageIds[i]
        // adding the HashMap to the ArrayList
        list.add(map)
    }

    // creating A string type array(from) which contains
    // column names for each View in each row of the list
    // and this array(form) is the fourth parameter of the SimpleAdapter
    val from=arrayOf("fruitName","fruitImage")
    // creating an integer type array(to) which contains
    id of each View in each row of the list
    and this array(form) is the fifth parameter of the SimpleAdapter*/
    val to= intArrayOf(R.id.textView,R.id.imageView)
    // creating an Object of SimpleAdapter
      // class and passing
    // all the required parameters
    val simpleAdapter=SimpleAdapter(this,list,R.layout.list_row_items,from,to)
    // now setting the simpleAdapter
      // to the ListView
    listView.adapter = simpleAdapter
  }
}
```

5.3 Introducing the data storage option

We employ some form of storage in Android to retain the data permanently (until destroyed) for future reference. Android Storage System is the name given to these storage systems. Internal storage, external storage, shared preferences, database, and shared storage are some of the storage options offered by Android. However, many users are unsure when to use which storage. So, in this blog, we'll discover when to use which storage unit. Let's begin with the internal storage system. We create several variables for storing various data that are used in an Android application when developing it. For example, we can utilize a variable to save data from a remote database and then use that variable throughout the application. These variables, however, are in-app storage, which means they will be visible to you while the app is operating. When the application is ended, all of the data in the variable is wiped, and you are left with nothing. Those variables will be created again when you start the application, and new values can be saved in those variables.

1. **Storage on the Inside:** When you install an app on your phone, the Android operating system will give you some form of secret internal storage where the app can store its private data. No other application has access to this information. When you uninstall an application, all of the data associated with it is also removed.To save a file to the internal storage, you must first obtain it from the internal directory. You can do this by calling the getFilesDir() or getCacheDir() methods. The getFilesDir() method returns the absolute path to the directory where files are created on the filesystem. getCacheDir() returns the absolute path to the filesystem's application-specific cache directory.

   When Should Internal Storage Be Used?

   The internal storage can be used when you need some confidential data for your application. Another thing to keep in mind is that if your app is storing data that may be utilized by other apps, you should avoid using internal storage since when you remove the app, all of your data will be gone, and other apps will never have access to that data. For instance, if your app is downloading a pdf or storing an image or video that might be used by other apps, you shouldn't use internal storage.

2. **External Hard Drives:** Most Android devices have relatively low internal storage. As a result, we keep our data on an external storage device. These storage units are accessible to everyone, which means they can be accessed by all of your device's applications. You can also access the storage by connecting your mobile device to a computer. You must obtain the READ EXTERNAL STORAGE permission from the user in order to gain access to the external storage. As a result, any application with this permission has access to your app's data.

   When is it appropriate to use external storage?

   You can use external storage if the data that your application stores can be used by other applications. Additionally, if the file stored by your application is huge, such as a video, you can save it to external storage. You can use external storage to keep the data even after uninstalling the application.

3. **Using the Shared Preferences:** You can use the shared preferences if you only have a little amount of data to keep and don't want to use the internal storage. Shared Preferences are used to store data in a key-value format, which means you'll have one key and the associated data or value will be stored depending on that key. The data saved in the shared preferences will remain with the application until you delete it from your phone. All shared preferences will be deleted from the device if you uninstall the application.

   When Should Shared Preferences Be Used?

You can utilize the shared preference in your application when the data you want to store is relatively little. It's not a good idea to save more than 100 kilobytes of data in shared preferences. In addition, if you wish to keep tiny and private data, you can use Android's shared preferences.

4. **Using Android Database:** Databases are collections of data that are organized and saved for future use. Using a Database Management System, you can store any type of data in your database. All you have to do is establish the database and use one query to perform all of the operations, such as insertion, deletion, and searching. The query will be passed to the database, which will return the desired output. In Android, an SQLite database is an example of a database.

   When Should you utilize a database?

   A database is useful when you need to keep track of structured data. A database can hold any type of information. So, if your data is large and you want to retrieve it quickly, you can use a database and store it in a structured style.

5.4 Internal and External Storage

External Storage: Android External Storage is the memory space in which we perform read and write operation. Files in the external storage are stored in /sdcard or /storage folder etc. The files which are saved in the external storage is readable and can be modified by the user.

Before accessing the file in external storage in our application, we should check the availability of the external storage in our device.

Example: In this example, we will write the data to file inside the external storage and read the same file content from same external storage.

Step1: Create activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout 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="example.javatpoint.com.kotlinexternalstoragereadwrite.MainActivity">
    <TextView
        android:id="@+id/textView"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/textView2"
        android:layout_alignParentTop="true"
        android:layout_alignStart="@+id/textView2"
        android:layout_marginTop="68dp"
```

```
    android:text="File Name"
    android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.027"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toTopOf="parent"
    app:layout_constraintVertical_bias="0.065" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/editTextData"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_marginBottom="36dp"
    android:layout_marginLeft="50dp"
    android:layout_marginStart="50dp"
    android:text="File Data"
    android:textAppearance="@style/Base.TextAppearance.AppCompat.Medium"
    app:layout_constraintBottom_toBottomOf="parent"
    app:layout_constraintEnd_toEndOf="parent"
    app:layout_constraintHorizontal_bias="0.027"
    app:layout_constraintStart_toStartOf="parent"
    app:layout_constraintTop_toBottomOf="@+id/textView"
    app:layout_constraintVertical_bias="0.167" />

<EditText
    android:id="@+id/editTextFile"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
```

```
    android:layout_alignLeft="@+id/editTextData"
    android:layout_alignStart="@+id/editTextData"
    android:layout_alignTop="@+id/textView"
    android:ems="10"
    android:inputType="none" />

<EditText
    android:id="@+id/editTextData"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentEnd="true"
    android:layout_alignParentRight="true"
    android:layout_below="@+id/editTextFile"
    android:layout_marginEnd="37dp"
    android:layout_marginRight="37dp"
    android:layout_marginTop="30dp"
    android:ems="10"
    android:inputType="none"
    android:lines="5" />

<Button
    android:id="@+id/button_save"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_marginBottom="68dp"
    android:layout_toLeftOf="@+id/editTextData"
    android:layout_toStartOf="@+id/editTextData"
    android:text="Save" />

<Button
    android:id="@+id/button_view"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBottom="@+id/button_save"
    android:layout_alignEnd="@+id/editTextData"
    android:layout_alignRight="@+id/editTextData"
    android:layout_marginEnd="43dp"
    android:layout_marginRight="43dp"
    android:text="View" />

</RelativeLayout>
```

Step2: Create MainActivity.kt

```
package example.javatpoint.com.kotlinexternalstoragereadwrite

import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.Button
import android.widget.EditText
import android.widget.Toast
import android.os.Environment
import java.io.*

class MainActivity : AppCompatActivity() {
   private val filepath = "MyFileStorage"
   internal var myExternalFile: File?=null
   private val isExternalStorageReadOnly: Boolean get() {
      val extStorageState = Environment.getExternalStorageState()
      return if (Environment.MEDIA_MOUNTED_READ_ONLY.equals(extStorageState)) {
            true
      } else {
            false
      }
```

```
}
private val isExternalStorageAvailable: Boolean get() {
    val extStorageState = Environment.getExternalStorageState()
    return if (Environment.MEDIA_MOUNTED.equals(extStorageState))  {
        true
    } else{
        false
    }
}

override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    val fileName = findViewById(R.id.editTextFile)  as EditText
    val fileData = findViewById(R.id.editTextData)  as EditText
    val saveButton = findViewById<Button>(R.id.button_save)  as Button
    val viewButton  = findViewById(R.id.button_view)  as Button

    saveButton.setOnClickListener(View.OnClickListener  {
       myExternalFile = File(getExternalFilesDir(filepath), fileName.text.toString())
        try {
            val fileOutPutStream = FileOutputStream(myExternalFile)
            fileOutPutStream.write(fileData.text.toString().toByteArray())
            fileOutPutStream.close()
        } catch (e: IOException)  {
            e.printStackTrace()
        }
        Toast.makeText(applicationContext,"data save",Toast.LENGTH_SHORT).show()
    })
    viewButton.setOnClickListener(View.OnClickListener  {
       myExternalFile = File(getExternalFilesDir(filepath), fileName.text.toString())
```

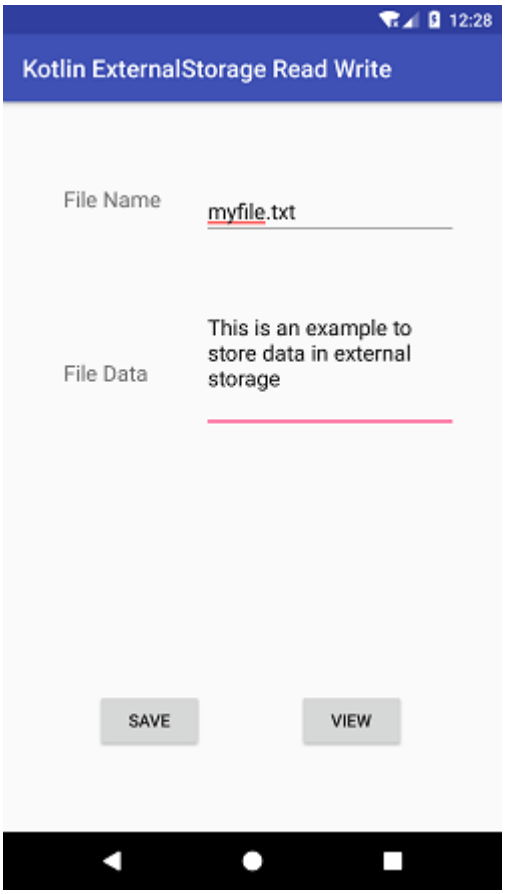
```
        val filename = fileName.text.toString()
        myExternalFile = File(getExternalFilesDir(filepath),filename)
        if(filename.toString()!=null && filename.toString().trim()!=""){
            var fileInputStream =FileInputStream(myExternalFile)
            var inputStreamReader: InputStreamReader = InputStreamReader(fileInputStream)
            val bufferedReader: BufferedReader = BufferedReader(inputStreamReader)
            val stringBuilder: StringBuilder = StringBuilder()
            var text: String? = null
            while ({ text = bufferedReader.readLine(); text }() != null) {
                stringBuilder.append(text)
            }
            fileInputStream.close()
            //Displaying data on EditText
            Toast.makeText(applicationContext,stringBuilder.toString(),Toast.LENGTH_SHORT).show()
        }
    })

    if (!isExternalStorageAvailable || isExternalStorageReadOnly) {
        saveButton.isEnabled = false
    }
  }
}
```

Step3: Create AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="example.javatpoint.com.kotlinexternalstoragereadwrite">
  <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"/>
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
  <application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
```

```
    android:roundIcon="@mipmap/ic_launcher_round"
    android:supportsRtl="true"
    android:theme="@style/AppTheme">
    <activity android:name=".MainActivity">
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

</manifest>
```

OutPut:



5.5 Using SQLite Database

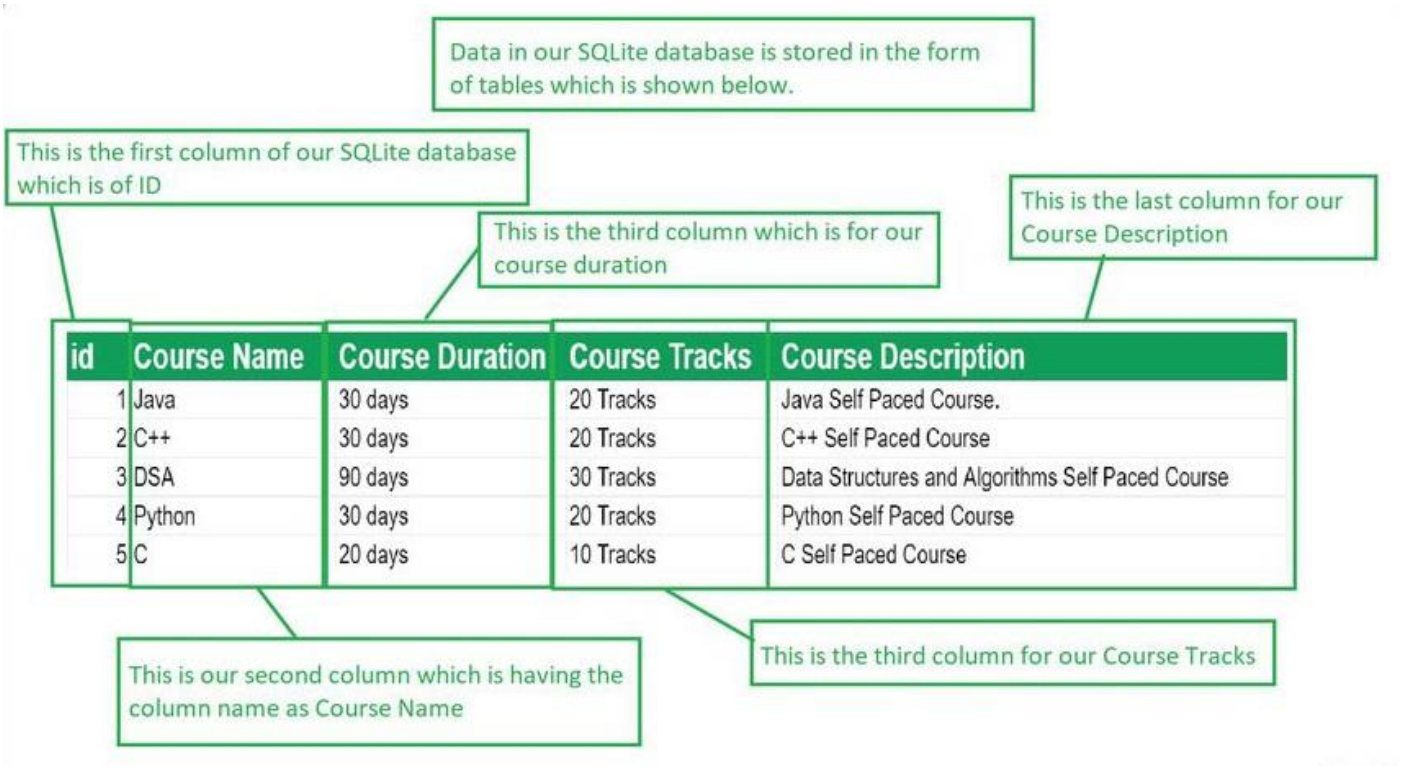
SQLite is another data storage available in Android where we can store data in the user's device and can use it any time when required. In this article, we will take a look at creating an SQLite database in the Android app and adding data to that database in the Android app. This is a series of 4 articles 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.

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.



id	Course Name	Course Duration	Course Tracks	Course Description
1	Java	30 days	20 Tracks	Java Self Paced Course.
2	C++	30 days	20 Tracks	C++ Self Paced Course
3	DSA	90 days	30 Tracks	Data Structures and Algorithms Self Paced Course
4	Python	30 days	20 Tracks	Python Self Paced Course
5	C	20 days	10 Tracks	C Self Paced Course

Step1: Giving permission to access the storage in the AndroidManifest.xml file

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

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

Step2: Working with the activity_main.xml file

Navigate to app > res > layout > activity_main.xml. Add the below code to your file. Below is the code for activity_main.xml.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    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"
    android:orientation="vertical"
    tools:context=".MainActivity">

    <!-- Edit text to enter name -->
    <EditText
        android:id="@+id/enterName"
        android:layout_width="match_parent"
```

```
    android:layout_height="wrap_content"
    android:hint="Enter Name"
    android:textSize="22sp"
    android:layout_margin="20sp"/>

<!-- Edit text to enter age -->
<EditText
    android:id="@+id/enterAge"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_margin="20sp"
    android:textSize="22sp"
    android:hint="Enter Age"/>

<!-- Button to add Name -->
<Button
    android:id="@+id/addName"
    android:layout_width="150sp"
    android:layout_gravity="center"
    android:background="@color/colorPrimary"
    android:text="Add Name"
    android:textColor="#ffffff"
    android:textSize="20sp"
    android:layout_height="wrap_content"
    android:layout_marginTop="20sp"/>

<!-- Button to print Name -->
<Button
    android:id="@+id/printName"
    android:layout_width="150sp"
    android:layout_gravity="center"
    android:background="@color/colorPrimary"
```

```
    android:text="Print Name"
    android:textColor="#ffffff"
    android:textSize="20sp"
    android:layout_height="wrap_content"
    android:layout_marginTop="20sp"/>

<LinearLayout
    android:layout_width="match_parent"
    android:layout_height="wrap_content">

    <!-- Text view to get all name -->
    <TextView
        android:id="@+id/Name"
        android:textAlignment="center"
        android:layout_weight="1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="20sp"
        android:text="Name\n\n"
        android:textSize="22sp"
        android:padding="10sp"
        android:textColor="#000000"/>

    <!-- Text view to get all ages -->
    <TextView
        android:layout_weight="1"
        android:id="@+id/Age"
        android:textAlignment="center"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="20sp"
        android:text="Age\n\n"
```

```
        android:textSize="22sp"
        android:padding="10sp"
        android:textColor="#000000"/>

    </LinearLayout>

</LinearLayout>
```

Step4: Creating a new class for SQLite operations

```
package com.release.database

import android.content.ContentValues
import android.content.Context
import android.database.Cursor
import android.database.sqlite.SQLiteDatabase
import android.database.sqlite.SQLiteOpenHelper

class DBHelper(context: Context, factory: SQLiteDatabase.CursorFactory?) :
    SQLiteOpenHelper(context, DATABASE_NAME,  factory, DATABASE_VERSION)  {

    // below is the method for creating a database by a sqlite query
    override fun onCreate(db: SQLiteDatabase) {
        // below is a sqlite query, where column names
        // along with their data types is given
        val query = ("CREATE  TABLE " + TABLE_NAME  + " ("
                + ID_COL  + " INTEGER  PRIMARY  KEY, " +
                NAME_COl + " TEXT," +
                AGE_COL  + " TEXT" + ")")
        // we are calling sqlite
        // method for executing our query
        db.execSQL(query)
    }
```

```
override fun onUpgrade(db: SQLiteDatabase, p1: Int, p2: Int) {
    // this method is to check if table already exists
    db.execSQL("DROP  TABLE IF EXISTS " + TABLE_NAME)
    onCreate(db)
}
// This method is for adding data in our database
fun addName(name : String, age : String ){
    // below we are creating
    // a content values variable
    val values = ContentValues()
    // we are inserting our values
    // in the form of key-value pair
    values.put(NAME_COl, name)
    values.put(AGE_COL, age)
    // here we are creating a
    // writable variable of
    // our database as we want to
    // insert value in our database
    val db = this.writableDatabase
    // all values are inserted into database
    db.insert(TABLE_NAME, null, values)
    // at last we are
    // closing our database
    db.close()
}
// below method is to get
// all data from our database
fun getName(): Cursor? {
    // here we are creating a readable
    // variable of our database
    // as we want to read value from it
    val db = this.readableDatabase
```

```
        // below code returns a cursor to
        // read data from the database
        return db.rawQuery("SELECT  * FROM " + TABLE_NAME,  null)
    }
    companion object{
        // here we have defined variables for our database
        // below is variable for database name
        private val DATABASE_NAME  = "Practice_Database"
        // below is the variable for database version
        private val DATABASE_VERSION  = 1
        // below is the variable for table name
        val TABLE_NAME  = "Main_table"
        // below is the variable for id column
        val ID_COL  = "id"
        // below is the variable for name column
        val NAME_COl  = "name"
        // below is the variable for age column
        val AGE_COL  = "age"
    }
}
```

Step5: Working with MainActivity.kt file

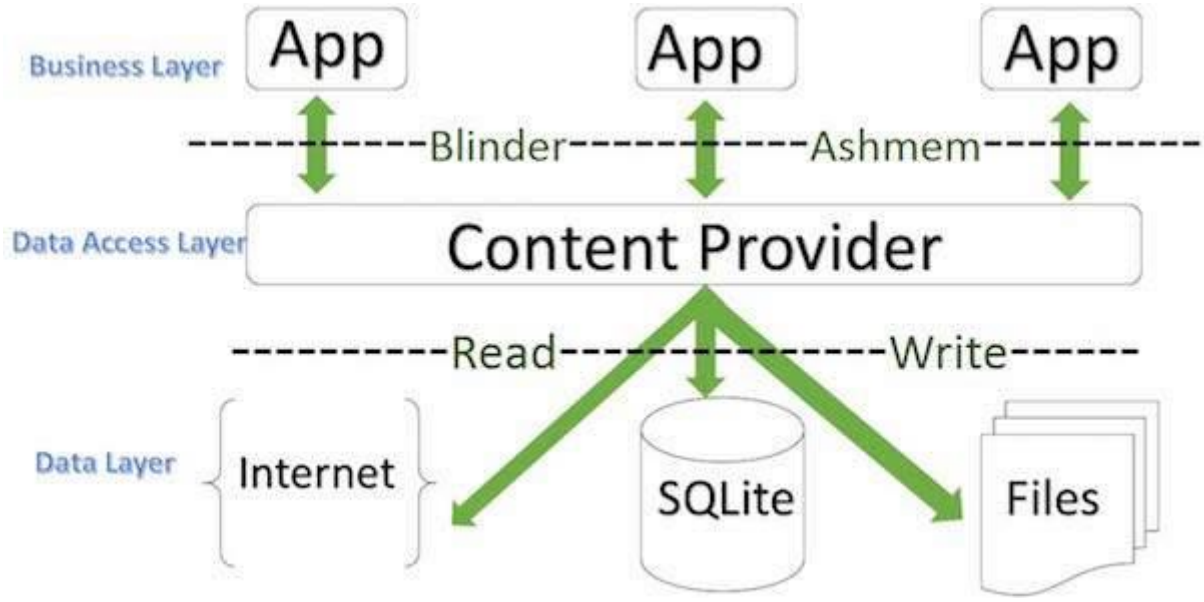
```
package com.release.main

import androidx.appcompat.app.AppCompatActivity
import android.os.Bundle
import android.widget.Toast
import kotlinx.android.synthetic.main.activity_main.*

class MainActivity : AppCompatActivity() {

    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
```

```
setContentView(R.layout.activity_main)
// below code is to add on click
// listener to our add name button
addName.setOnClickListener{
    // below we have created
    // a new DBHelper class,
    // and passed context to it
    val db = DBHelper(this, null)
    // creating variables for values
    // in name and age edit texts
    val name = enterName.text.toString()
    val age = enterAge.text.toString()
    // calling method to add
    // name to our database
    db.addName(name, age)
    // Toast to message on the screen
    Toast.makeText(this, name + " added to database", Toast.LENGTH_LONG).show()
    // at last, clearing edit texts
    enterName.text.clear()
    enterAge.text.clear()
}
// below code is to add on  click
// listener to our print name button
printName.setOnClickListener{
    // creating a DBHelper class
    // and passing context to it
    val db = DBHelper(this, null)
    // below is the variable for cursor
    // we have called method to get
    // all names from our database
    // and add to name text view
    val cursor = db.getName()
```

```
        // moving the cursor to first position and
        // appending value in the text view
        cursor!!.moveToFirst()
        Name.append(cursor.getString(cursor.getColumnIndex(DBHelper.NAME_COl))  + "\n")
        Age.append(cursor.getString(cursor.getColumnIndex(DBHelper.AGE_COL))  + "\n")
        // moving our cursor to next
        // position and appending values
        while(cursor.moveToNext()){
            Name.append(cursor.getString(cursor.getColumnIndex(DBHelper.NAME_COl))  + "\n")
            Age.append(cursor.getString(cursor.getColumnIndex(DBHelper.AGE_COL))  + "\n")
        }
        // at last we close our cursor
        cursor.close()
    }
  }
}
```

5.6 Working with Content Provider

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. A content provider can use different ways to store its data and the data can be stored in a database, in files, or even over a network.



Content providers let you centralize content in one place and have many different applications access it as needed. A content provider behaves very much like a database where you can query it, edit its content, as well as add or delete content using insert(), update(), delete(), and query() methods. In most cases this data is stored in an SQlite database.

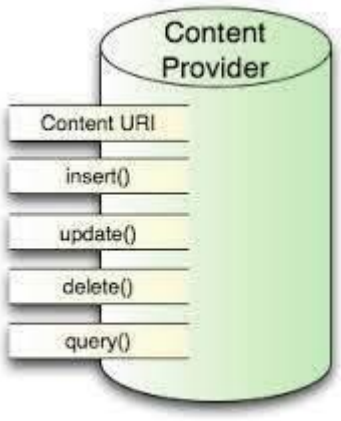
A content provider is implemented as a subclass of ContentProvider class and must implement a standard set of APIs that enable other applications to perform transactions

Create Content Provider

This involves number of simple steps to create your own content provider.

- First of all you need to create a Content Provider class that extends the ContentProviderbaseclass.
- Second, you need to define your content provider URI address which will be used to access the content.
- Next you will need to create your own database to keep the content. Usually, Android uses SQLite database and framework needs to override onCreate() method which will use SQLite Open Helper method to create or open the provider's database. When your application is launched, the onCreate() handler of each of its Content Providers is called on the main application thread.
- Next you will have to implement Content Provider queries to perform different database specific operations.
- Finally register your Content Provider in your activity file using <provider> tag.

Here is the list of methods which you need to override in Content Provider class to have your Content Provider working –



ContentProvider

- **onCreate()** This method is called when the provider is started.
- **query()** This method receives a request from a client. The result is returned as a Cursor object.
- **insert()**This method inserts a new record into the content provider.
- **delete()** This method deletes an existing record from the content provider.

- **update()** This method updates an existing record from the content provider.
- **getType()** This method returns the MIME type of the data at the given URI.

5.7 Web Service and JSON Parsing

JSON Parsing: JSON (Javascript Object Notation) is a programming language . It is minimal, textual, and a subset of JavaScript. It is an alternative to XML.

Android provides support to parse the JSON object and array.

Advantage of JSON over XML:

1. JSON is faster and easier than xml for AJAX applications.
2. Unlike XML, it is shorter and quicker to read and write.
3. It uses array.

JSON Object: A JSON object contains key/value pairs like map. The keys are strings and the values are the JSON types. Keys and values are separated by comma. The { (curly brace) represents the json object.

```
{
   "employee": {
      "name":      "sachin",
      "salary":     56000,
      "married":    true
   }
}
```

JSON array: The [(square bracket) represents the json array.

["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]

Let's take another example of JSON array.

```
{ "Employee" :
   [
    {"id":"101","name":"Sonoo Jaiswal","salary":"50000"},
    {"id":"102","name":"Vimal Jaiswal","salary":"60000"}
   ]
}
```

Example:

Step1: Create a JSON file index.html.

```
{"info":[
```

```
{"name":"Ajay","id":"111","email":"ajay@gmail.com"},
{"name":"John","id":"112","email":"john@gmail.com"},
{"name":"Rahul","id":"113","email":"rahul@gmail.com"},
{"name":"Maich","id":"114","email":"maich@gmail.com"},
{"name":"Vikash","id":"115","email":"vikash@gmail.com"},
{"name":"Mayank","id":"116","email":"mayank@gmail.com"},
{"name":"Prem","id":"117","email":"prem@gmail.com"},
{"name":"Chandan","id":"118","email":"chandan@gmail.com"},
{"name":"Soham","id":"119","email":"soham@gmail.com"},
{"name":"Mukesh","id":"120","email":"mukesh@gmail.com"},
{"name":"Ajad","id":"121","email":"ajad@gmail.com"}
]
}
```


Step2: Add the ListView in the activity_main.xml layout file.

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.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="example.javatpoint.com.kotlinjsonparsing.MainActivity">

   <ListView
      android:id="@+id/listView"
      android:layout_width="match_parent"
      android:layout_height="match_parent">

   </ListView>

</android.support.constraint.ConstraintLayout>
```

Step3: Add the following okhttp dependency in the build.gradle file.

```
compile 'com.squareup.okhttp3:okhttp:3.8.1'
```

Step4: Create a data model class Model.kt which includes the information String "id", String "name" and String "email".

```
package example.javatpoint.com.kotlinjsonparsing


public class Model{
   lateinit var id:String
   lateinit var name:String
   lateinit var email:String


   constructor(id: String,name:String,email:String) {
      this.id = id
      this.name = name
      this.email = email
   }
   constructor()
}
```

Step5: Create an adapter_layout.xml file in the layout directory which contains the row items for ListView.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:id="@+id/linearLayout"
   android:padding="5dp"
   android:orientation="vertical">


   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/tvId"
      android:layout_margin="5dp"
```

```
    android:textSize="16dp"/>
  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/tvName"
    android:textSize="16dp"
    android:layout_margin="5dp"/>

  <TextView
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/tvEmail"
    android:layout_margin="5dp"
    android:textSize="16dp"/>

</LinearLayout>
```

Step6: Create a custom adapter class CustomAdapter.kt and extend BaseAdapter to handle the custom ListView.

```
package example.javatpoint.com.kotlinjsonparsing

import android.content.Context
import android.view.LayoutInflater
import android.view.View
import android.view.ViewGroup
import android.widget.BaseAdapter
import android.widget.LinearLayout
import android.widget.TextView

class CustomAdapter(context: Context,arrayListDetails:ArrayList<Model>) : BaseAdapter(){

  private val layoutInflater: LayoutInflater
  private val arrayListDetails:ArrayList<Model>
```

```
  init {
      this.layoutInflater = LayoutInflater.from(context)
      this.arrayListDetails=arrayListDetails
  }
  override fun getCount(): Int {
      return arrayListDetails.size
  }
  override fun getItem(position: Int): Any {
      return arrayListDetails.get(position)
  }
  override fun getItemId(position: Int): Long {
      return position.toLong()
  }
  override fun getView(position: Int, convertView: View?, parent: ViewGroup): View? {
      val view: View?
      val listRowHolder: ListRowHolder
      if (convertView == null) {
          view = this.layoutInflater.inflate(R.layout.adapter_layout, parent, false)
          listRowHolder = ListRowHolder(view)
          view.tag = listRowHolder
      } else {
          view = convertView
          listRowHolder = view.tag as ListRowHolder
      }
      listRowHolder.tvName.text = arrayListDetails.get(position).name
      listRowHolder.tvEmail.text = arrayListDetails.get(position).email
      listRowHolder.tvId.text = arrayListDetails.get(position).id
      return view
  }
}

private class ListRowHolder(row: View?) {
```

```
    public val tvName: TextView
    public val tvEmail: TextView
    public val tvId: TextView
    public val linearLayout: LinearLayout

    init {
        this.tvId = row?.findViewById<TextView>(R.id.tvId) as TextView
        this.tvName = row?.findViewById<TextView>(R.id.tvName) as TextView
        this.tvEmail = row?.findViewById<TextView>(R.id.tvEmail) as TextView
        this.linearLayout = row?.findViewById<LinearLayout>(R.id.linearLayout) as LinearLayout
    }
}
```

Step7: Add the following code in MainActivity.kt class file. This class read the JSON data in the form of a JSON object. Using the JSON object, we read the JSON array data. The JSON data are bind in ArrayList.

```
package example.javatpoint.com.kotlinjsonparsing

import android.support.v7.app.AppCompatActivity
import android.os.Bundle
import android.view.View
import android.widget.ListView
import android.widget.ProgressBar
import okhttp3.*
import org.json.JSONArray
import org.json.JSONObject
import java.io.IOException
import kotlin.collections.ArrayList

class MainActivity : AppCompatActivity() {
    lateinit var progress:ProgressBar
    lateinit var listView_details: ListView
    var arrayList_details:ArrayList<Model> = ArrayList();
```

```
//OkHttpClient creates connection pool between client and server
val client = OkHttpClient()
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    progress = findViewById(R.id.progressBar)
    progress.visibility = View.VISIBLE
    listView_details = findViewById<ListView>(R.id.listView) as ListView
    run("http://10.0.0.7:8080/jsondata/index.html")
}

fun run(url: String) {
    progress.visibility = View.VISIBLE
    val request = Request.Builder()
            .url(url)
            .build()
    client.newCall(request).enqueue(object : Callback {
        override fun onFailure(call: Call, e: IOException) {
            progress.visibility = View.GONE
        }

        override fun onResponse(call: Call, response: Response) {
            var str_response = response.body()!!.string()
            //creating json object
            val json_contact:JSONObject = JSONObject(str_response)
            //creating json array
            var jsonarray_info:JSONArray= json_contact.getJSONArray("info")
            var i:Int = 0
            var size:Int = jsonarray_info.length()
            arrayList_details= ArrayList();
            for (i in 0.. size-1) {
                var json_objectdetail:JSONObject=jsonarray_info.getJSONObject(i)
```

```
            var model:Model= Model();
            model.id=json_objectdetail.getString("id")
            model.name=json_objectdetail.getString("name")
            model.email=json_objectdetail.getString("email")
            arrayList_details.add(model)
        }

        runOnUiThread {
            //stuff that updates ui
            val obj_adapter : CustomAdapter
            obj_adapter = CustomAdapter(applicationContext,arrayList_details)
            listView_details.adapter=obj_adapter
        }
        progress.visibility = View.GONE
      }
    })
  }
}
```

Step8: Add the Internet permission in AndroidManifest.xml file.

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

OutPut: