Andrid programming

What is Android

Before learning all topics of android, it is required to know what is android.

Android is a software package and linux based operating system for mobile devices such as tablet computers and smartphones.

It is developed by Google and later the OHA (Open Handset Alliance). Java language is mainly used to write the android code even though other languages can be used.

The goal of android project is to create a successful real-world product that improves the mobile experience for end users.

There are many code names of android such as Lollipop, Kitkat, Jelly Bean, Ice cream Sandwich, Froyo, Ecliar, Donut etc which is covered in next page.

What is Open Handset Alliance (OHA)

It's a consortium of 84 companies such as google, samsung, AKM, synaptics, KDDI, Garmin, Teleca, Ebay, Intel etc.

It was established on 5th November, 2007, led by Google. It is committed to advance open standards, provide services and deploy handsets using the Android Plateform.

Features of Android

After learning what is android, let's see the features of android. The important features of android are given below:

1) It is open-source.

2) Anyone can customize the Android Platform.

3) There are a lot of mobile applications that can be chosen by the consumer.

4) It provides many interesting features like weather details, opening screen, live RSS (Really Simple Syndication) feeds etc.

It provides support for messaging services(SMS and MMS), web browser, storage (SQLite), connectivity (GSM, CDMA, Blue Tooth, Wi-Fi etc.), media, handset layout etc.

Categories of Android applications

There are many android applications in the market. The top categories are:

History of Android

The history and versions of android are interesting to know. The code names of android ranges from A to J currently, such as Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwitch, Jelly Bean, KitKat and Lollipop. Let's understand the android history in a sequence.

1) Initially, Andy Rubin founded Android Incorporation in Palo Alto, California, United States in October, 2003.

2) In 17th August 2005, Google acquired android Incorporation. Since then, it is in the subsidiary of Google Incorporation.

3) The key employees of Android Incorporation are Andy Rubin, Rich Miner, Chris White and Nick Sears.

4) Originally intended for camera but shifted to smart phones later because of law

5) Android is the nick name of Andy Rubin given by coworkers because of his love to robots.

6) In 2007, Google announces the development of android OS.

7) In 2008, HTC launched the first android mobile.

android Versions, Codename and API

In this Android version list, we have collated all Android version names along with the year they were released. If you wish to learn more about Android, you can check out this online Android course.

Android Version	Android Version Names	Release Year
Android Versions 1.0 – 1.1	No codename	2008
Android Version 1.5	Cupcake	2009
Android Version 1.6	Donut	2009
Android Versions 2.0 – 2.1	Eclair	2009
Android Version 2.2	Froyo	2010
Android Version 2.3	Gingerbread	2010
Android Versions 3.0 – 3.2	Honeycomb	2011
Android Version 4.0	Ice Cream Sandwich	2011
Android Versions 4.1 – 4.3	Jelly Bean	2012
Android Version 4.4	KitKat	2013
Android Versions 5.0 – 5.1	Lollipop	2014
Android Version 6.0	Marshmallow	2015

Android Versions 7.0 – 7.1	Nougat	2016
Android Versions 8.0 – 8.1	Oreo	2017
Android Version 9	Pie	2018

1. Android Version 1.0 – 1.1: (No Code name)

- Google released its first commercial Android version 1.0 in 2008.
- This version of Android was very basic but had Google apps like Gmail, YouTube, Calendar, Maps, Search, Instant Messaging, and many more. All of these applications were integrated into the Android OS directly.
- It also supported HTML and XHTML web pages, a camera, Wi-Fi, and came with Bluetooth support.

2. Android Version 1.5: Cupcake

Cupcake
Andriod 1.5

- In 2009, Android released its second major release, Android version 1.5 Cupcake.
- With this release, the tradition of naming Android versions after confectionaries started.
- With Cupcake, Android introduced the first on-screen keyboard as people moved to touchscreen smartphones from keypad-style handsets.
- They also introduced a framework for third-party app widgets, which was a significant step.
- Cupcake also introduced the platform's first-ever video recording option.

3. Android Version 1.6: Donut

E.

Image Source: cables.ph

- In the fall of 2009, Android dropped its next major update, Android 1.6 Donut.
- It came out at the right time, as the world slowly transitioned towards bigger screens and resolution phones.
- Donut came with built-in support for CDMA networks, which helped Android grow quickly.
- In addition, it introduced multiple new features like voice and text entry search, bookmark history, and WVGA resolution.
- It also allowed the users to select multiple photos for deletion at a time.

Also Read: ***How To Become An Android Developer***

4. Android Versions 2.0 – 2.1: Eclair

Image Source: android.fandom.com

- Android 2.0 Eclair was released just six weeks after Donut.
- Android. with this version, became popular among the masses, due to the hype created around Motorola's DROID phone and the marketing campaign led by Verizon.
- It introduced SMS, MMS, voice-guided turn-by-turn navigation and real-time traffic information, pinch-to-zoom capability (which only Apple had at that time), Bluetooth 2.1, fixed minor API, and a few bug fixes.

5. Android Version 2.2: Froyo

Image Source: twitter.com

- Froyo was released four months after Eclair's introduction.
- This version of Android largely focused on back-end performance, speed, and memory optimization.
- In addition, it introduced voice actions, which allowed users to perform basic functions like speaking a command, making notes, and getting directions.
- It also supported Adobe Flash, which Apple never offered to users.

6. Android Version 2.3: Gingerbread

Image Source: pinterest.com

- With Gingerbread, Android started foraying distinctive visual design.
- For example, the Android mascot is green in color and this version's prominent colors were Green and Black, which was visible throughout their UI.
- In addition, Gingerbread supported an extra-large screen, a simplified interface, enhanced copy/paste functions, NFC (Near Field Communication), and a host of improvements.

7. Android Versions 3.0 – 3.2: Honeycomb

- Android released the Honeycomb version in 2011 for the first Android-based tablet, the Motorola Xoom.
- With subsequent updates, 3.1 and 3.2, the Honeycomb remained a tablet-exclusive entity.
- This was a detour for Android's visual appeal, as this time it had blue colored holographic design instead of their usual black and green combo.
- In addition, it was different, designed to make the most of the tablet's widescreen space.

8. Android Version 4.0: Ice Cream Sandwich

Image Source: teknofilo.com

- Ice Cream Sandwich marked the entry of Android into the modern design language.
- While the Honeycomb version is considered the connection between old and new design, Ice Cream Sandwich refined all the visual elements with a single, unified UI vision that reunited phone and tablet design.
- It carried over the card-like appearance from Honeycomb and also introduced swiping, common for navigating across the OS.
- It also brought a framework or design standardization known as ‘Holo’ across OS and Android’s app ecosystem.

9. Android Versions 4.1 – 4.3: Jelly Bean

Image Source: blog.phonehouse.es

- Android Jelly Bean was introduced in 2012 and made the best impressions among new users.
- Jelly Bean was built on the foundation built by Android version 4.0.
- It polished many rough edges, making the OS more attractive and appealing.
- In addition, it improved accessibility and offered multiple features like screen lock, bug removal, 4K support, and Google Now.

10. Android Version 4.4: KitKat

Image Source: softonic.com

- Late in 2013, Android released KitKat.
- This Android version introduced “OK Google” support, offline music support, smart caller ID, better application compatibility, and many other built-in features.

11. Android Versions 5.0 – 5.1: Lollipop

Image Source: gbgmumbai.org

- With the version Lollipop, Android reinvented itself.
- Amongst all these versions in the Android list, it established the material design standard, which stands even today. This gave the OS a fresh and new visual look across all Android apps and even other Google products.
- Furthermore, the team maximized the usage of the card-based concept, which became a core pattern for the Android team.
- It also introduced at-a-glance access for all the notifications from the lock screen itself.
- In addition, Lollipop improved the 'OK Google' command support. This feature's voice activation was extended to work even when the device's screen was off.

Also Read: ***Android Architecture***

12. Android Version 6.0: Marshmallow

Image Source: pcmag.com

- This Android version was slightly updated when compared to the Lollipop version.
- With Marshmallow, Android started the trend of releasing a major update per year.
- In addition, Marshmallow introduced support for fingerprint readers, USB-C, App Standby feature, Doze mode to save battery life, and many more with lasting impressions.

13. Android Version 7.0 – 7.1: Nougat

Image Source: android.com

- Android Nougat is popular for releasing Google Assistant.
- This Android version offered few improvements, but all of them were significant.
- For example, they offered split-screen mode, a Data Saver feature, file-based encryption, battery usage alerts, a zoom-in screen, and many more features.
- Google also released Pixel, its first self-made phone, along the same timeline.

14. Android Version 8.0 – 8.1: Oreo

Image Source: android.com

- Oreo brought in some of the best features, like picture-in-picture support, adaptive icons, 2x booting speed, Google Play Protect, a notification snoozing option, and many other features.
- In addition, this Android version included many elements aligned with Google's goal of aligning Android and Chrome OS and transforming the Chromebook user experience.
- It also helped in Project Treble, which helped device manufacturers offer more timely software updates through a modular base for Android's code.

15. Android Version 9.0: Pie

Image Source: android.com

- Pie breathed in some fresh air to the Android mobile OS. It transformed and gave a new look to Android to make it feel more modern.
- The most popular change was the hybrid gesture/button navigation system, which replaced Android's Black panel for the Back, Home, and Overview keys.
- Pie also introduced many productivity features, which were missing in previous Android versions.
- In addition, it introduced many security and privacy enhancements and intelligent systems to manage power and screen brightness.

What is Open Handset Alliance (OHA)

It's a consortium of 84 companies such as google, samsung, AKM, synaptics, KDDI, Garmin, Teleca, Ebay, Intel etc.

It was established on 5th November, 2007, led by Google. It is committed to advance open standards, provide services and deploy handsets using the Android Plateform.

andoid platform and android sdk

This page provides release information about the SDK packages available for download from the SDK Manager, in the SDK Platforms tab.

Android Platform

The Android platform is a platform for mobile devices that uses a modified Linux kernel. The Android Platform was introduced by the Open Handset Alliance in November of 2007. Most applications that run on the Android platform are written in the Java programming language.

Explains Android Platform

The Android Platform was launched in 2007 by the Open Handset Alliance, an alliance of prominent companies that includes Google, HTC, Motorola, Texas Instruments and others. Although most of the applications that run on the Android Platform are written in Java, there is no Java Virtual Machine. Instead, the Java classes are first compiled into what are known as Dalvik Executables and run on the Dalvik Virtual Machine.

Android is an open development platform. However, it is not open in the sense that everyone can contribute while a version is under development. This is all done behind closed-doors at Google. Rather, the openness of Android starts when its source code is released to the public after it is finalized. This means once it is released anyone interested can take the code and alter it as they see fit.

To create an application for the platform, a developer requires the Android SDK, which includes tools and APIs. To shorten development time, Android developers typically integrate the SDK into graphical user IDEs (Integrated Development Environments). Beginners can also make use of the App Inventor, an application for creating Android apps that can be accessed online.

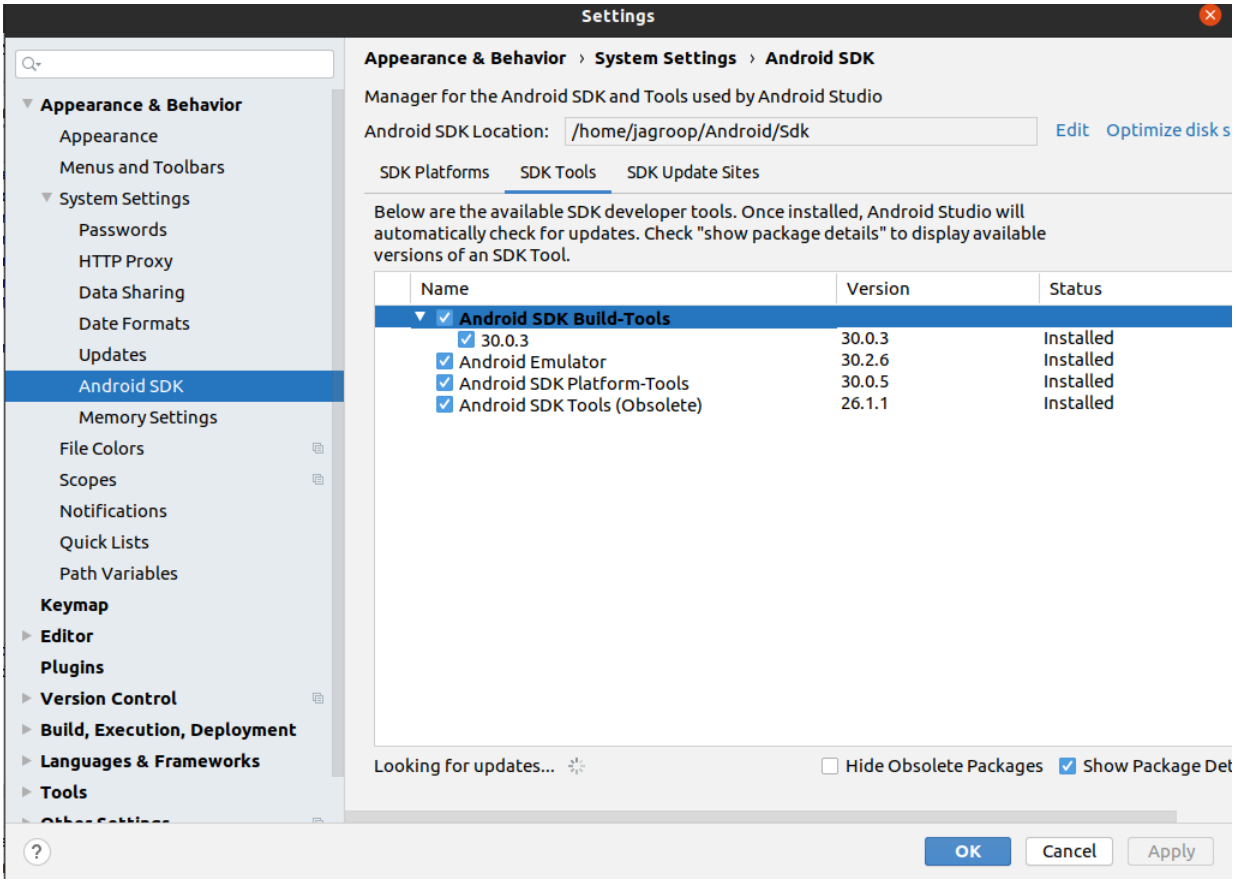
Android SDK and it’s Components

•

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

About Android SDK

Android SDK is a collection of libraries and Software Development tools that are essential for Developing Android Applications. Whenever Google releases a new version or update of Android Software, a corresponding SDK also releases with it. In the updated or new version of SDK, some more features are included which are not present in the previous version. Android SDK consists of some tools which are very essential for the development of Android Application. These tools provide a smooth flow of the development process from developing and debugging. Android SDK is compatible with all operating systems such as Windows, Linux, macOS, etc.



Components of Android SDK

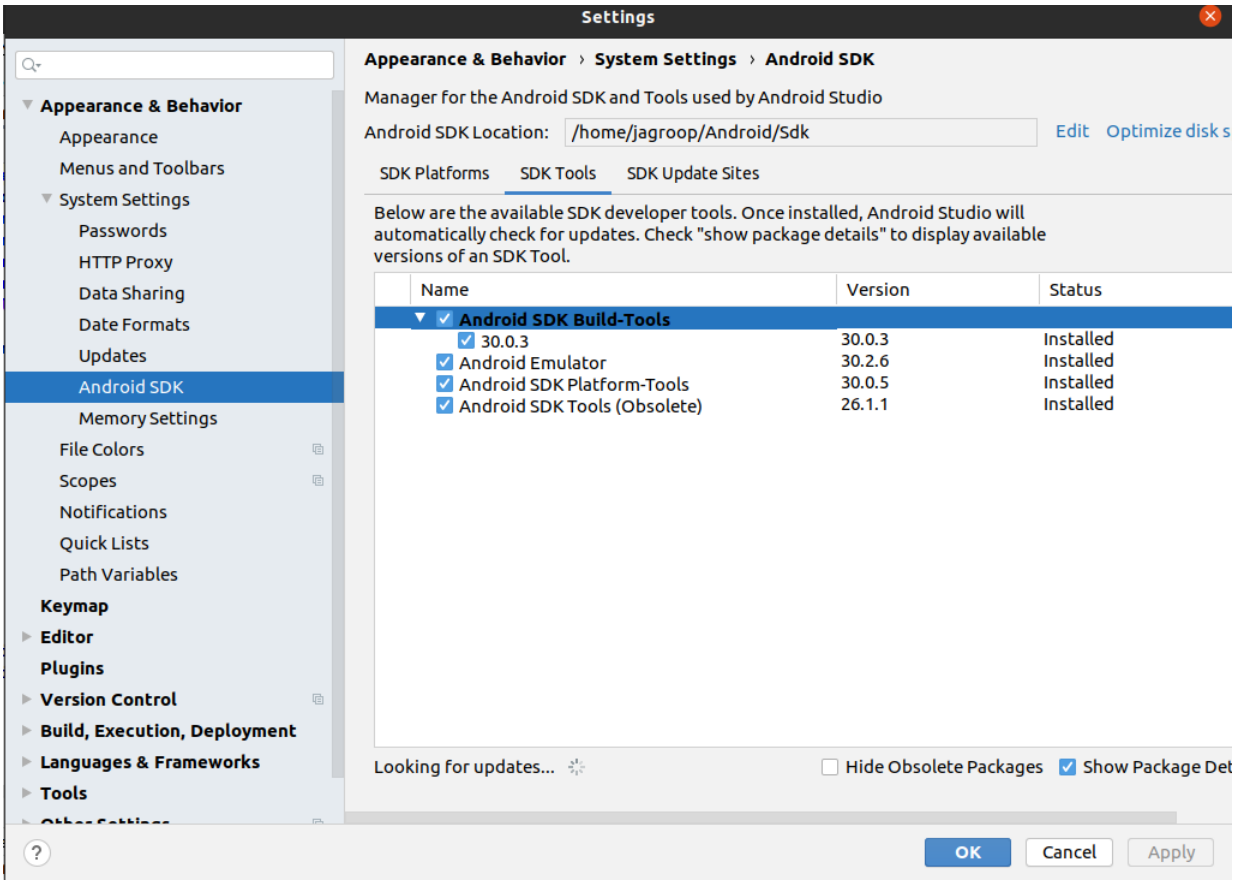
Android SDK Components play a major role in the Development of Android applications. Below are the important components:

1. Android SDK Tools

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

- Android SDK Build tool.
- Android Emulator.
- Android SDK Platform-tools.
- Android SDK Tools.

These are shown below :



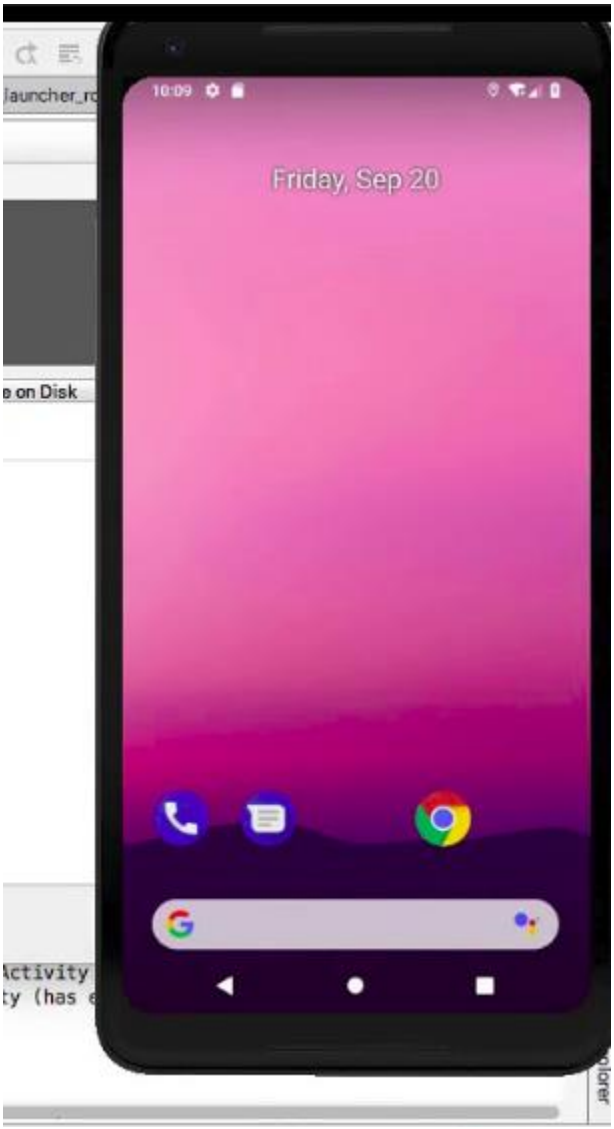
2. Android SDK Build-Tools

Android SDK build tools are used for building actual binaries of Android App. The main functions of Android SDK Build tools are built, debug, run and test Android applications. The latest version of the Android SDK Build tool is 30.0.3. While downloading or updating Android in our System, one must ensure that its latest version is download in SDK Components.

3. Android Emulator

An Android Emulator is a device that simulates an Android device on your system. Suppose we want to run our android application that we code. One option is that we will run this on our Android Mobile by Enabling USB Debugging on our mobile. Another option is using Android Emulator. In Android Emulator the virtual android device is shown on our system on which we run the Android application that we code.

Thus, it simply means that without needing any physical device Android SDK component “Android Emulator” provides a virtual device on the System where we run our Application. The emulator’s come with the configuration for Various android phones, tablets, Wear OS, and Android TV devices.



In Android Virtual Emulator all functions that are feasible on real Android mobile is works on virtual Device like:

- phone calls, text messages.
- stimulate different network speeds.
- specify the location of a device
- access on google play store and lot's more.

But there is one disadvantage of this emulator is that. It is very slow when System's PC has less RAM. It works fine when a maximum GB of RAM is present on our device.

4. Android SDK Platform-tools

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

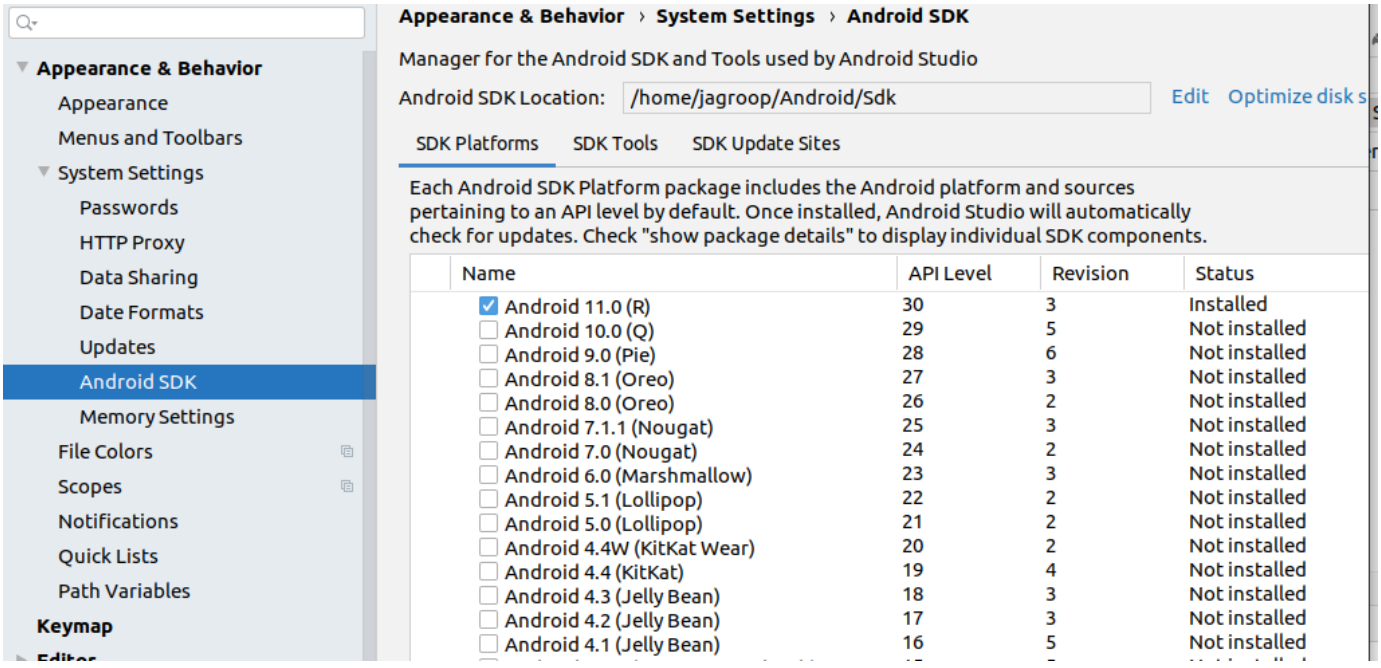
- Android Debug Bridge (ADB), is a command-line tool that helps to communicate with the device. It allows us to perform an action such as Installing App and Debugging App etc.
- Fastboot allows you to flash a device with a new system image.
- Systrace tools help to collect and inspect timing information. It is very crucial for App Debugging.

5. Android SDK Tools

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

6. SDK Platforms

For Each Android Software, one SDK platform is available as shown below:

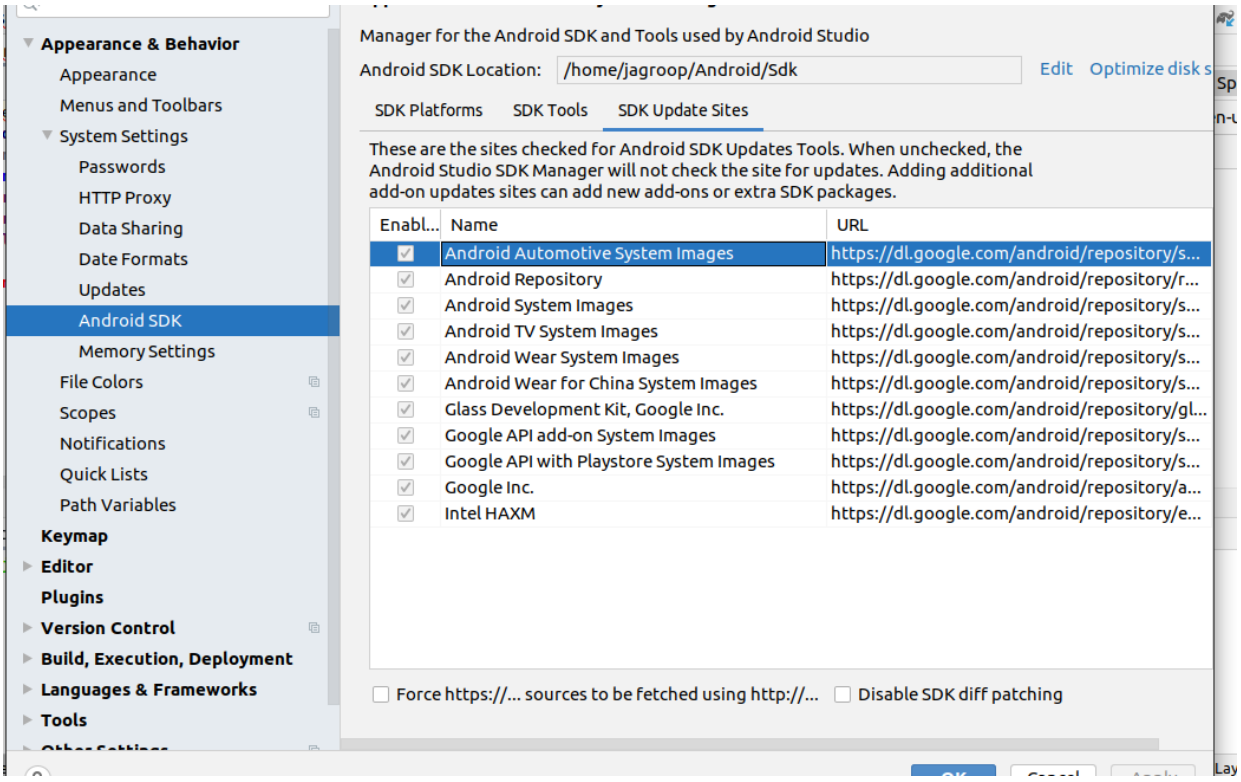


Like in this Android 11.0(R) is installed.

These are numbered according to the android version. The new version of the SDK platform has more features and more compatible but the old version is less compatible with fewer features. Like in Android 11.0(R) have more compatible and have more feature but the below versions like Android 10.0(Q), Android4.4(KitKat) have less feature and is less compatible.

7. SDK Update Sites

In SDK Update Sites, some sites are embedded in it which will check for Android SDK Updates Tools. In this, one must ensure we don't unclick the button below because these are checked by default which will check for updates if we will unclick it then it doesn't check updates for those.



And why go anywhere else when our DSA to Development: Coding Guide will help you master all this in a few months! Apply now to our

Fix "SDK location not found. Define location with sdk.dir in the local.properties file or with an ANDROID_HOME environment variable" in Android Studio

SDK is a Software Development Kit. SDK brings together a group of tools that enable programming for Mobile Applications such as Android, and iOS. You can download SDK from the official Google Developers Website for Android and then Extract/put it in the default folder if there is no such then make it. The Default Path of SDK in PC is: Windows: sdk.

2 min read

Different Ways to Change Android SDK Path in Android Studio

Android SDK is one of the most useful components which is required to develop Android Applications. Android SDK is also referred to as the Android Software Development Kit which provides so many features which are required in Android which are given below: A sample source code.An Emulator.Debugger.Required set of libraries.Required APIs for Android

5 min read

Different Ways to Fix “Select Android SDK” Error in Android Studio

Android SDK is one of the most useful components which is required to develop Android Applications. Android SDK is also referred to as the Android Software Development Kit which provides so many features which are required in Android which are given below: A sample source code.An Emulator.Debugger.Required set of libraries.Required APIs for Android

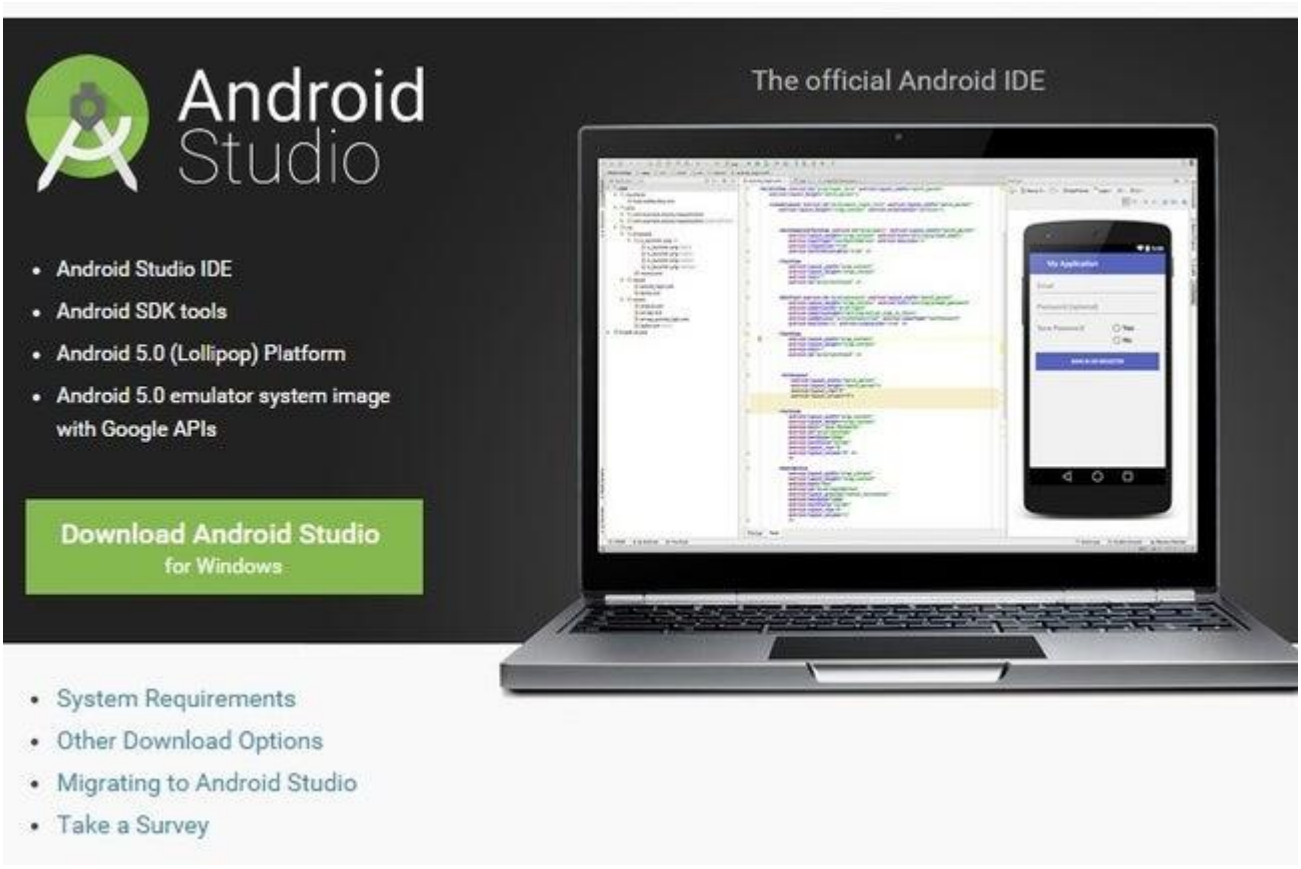
5 min read

Introduction: How to Create an Android App With Android

Stud

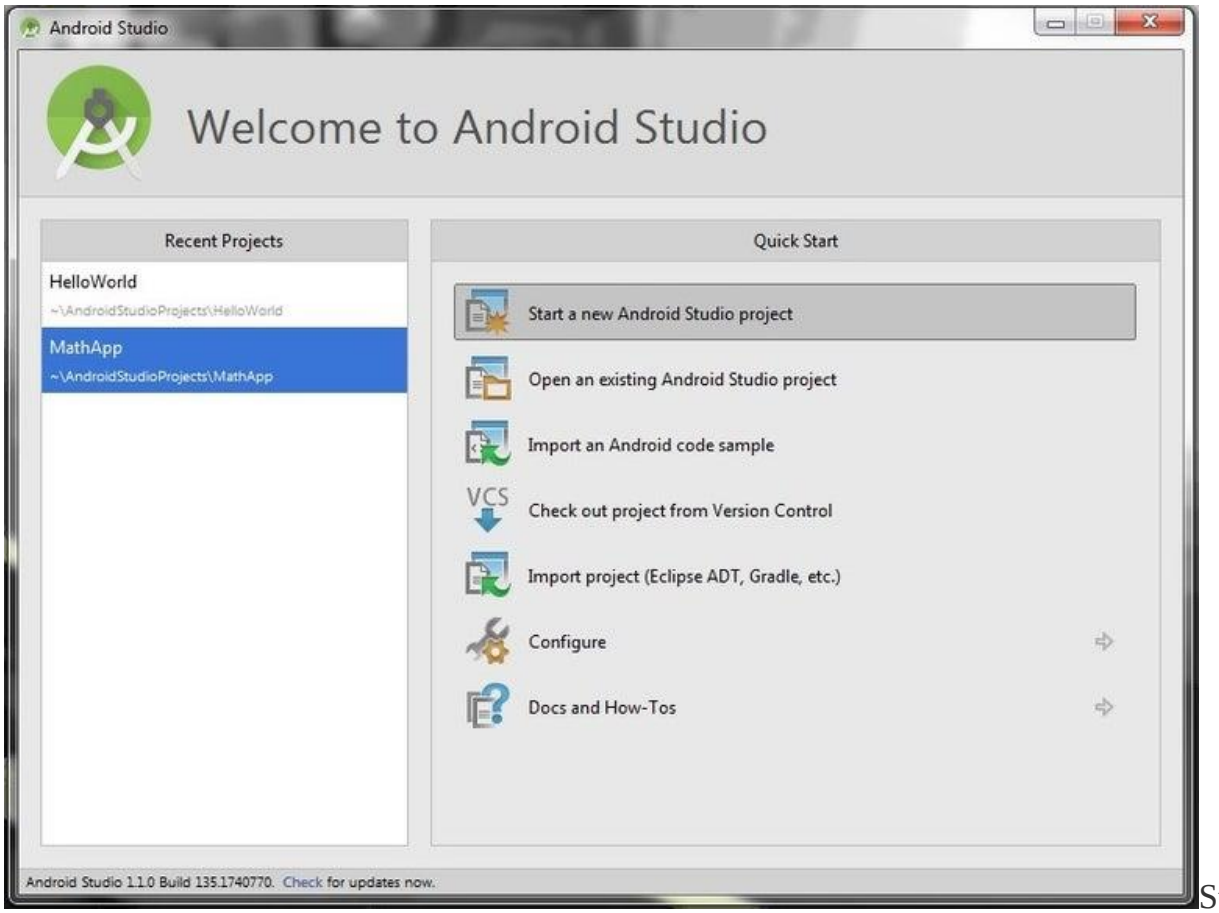
This tutorial will teach you the basics of how to build an Android app using the Android Studio development environment. As Android devices become increasingly more common, demand for new apps will only increase. Android Studio is an easy to use (and free) development environment to learn on. It's best if one has a working knowledge of the Java programming language for this tutorial because it is the language used by Android. There won't be much code used in this tutorial, so I will assume that you know enough Java to understand or are willing to look up what you don't know. This will take roughly 30-60 minutes, depending on how quickly you are able to download and install Android Studio. After using this tutorial to create your first Android app, you'll be well on your way to a fun new hobby or possibly even a promising career in mobile development.

Step 1: Install Android Studio

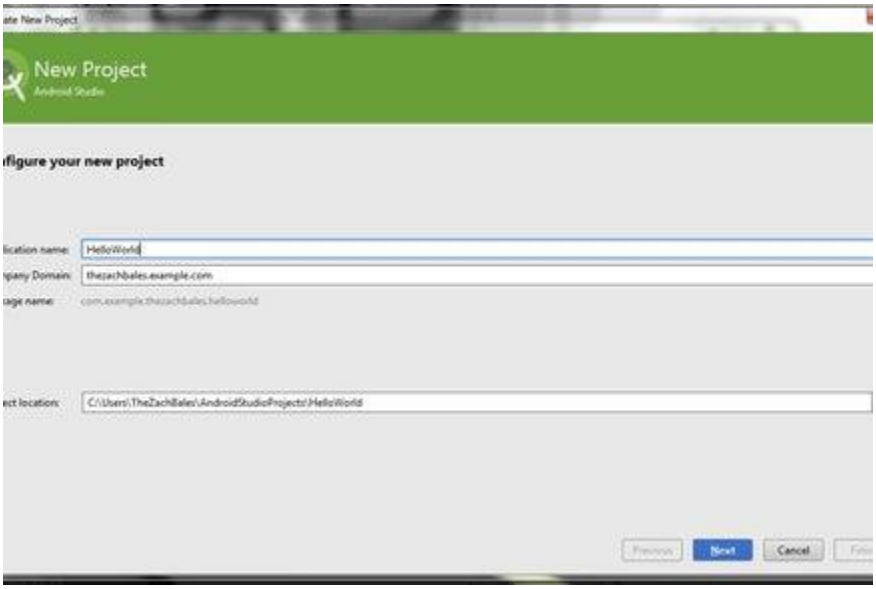


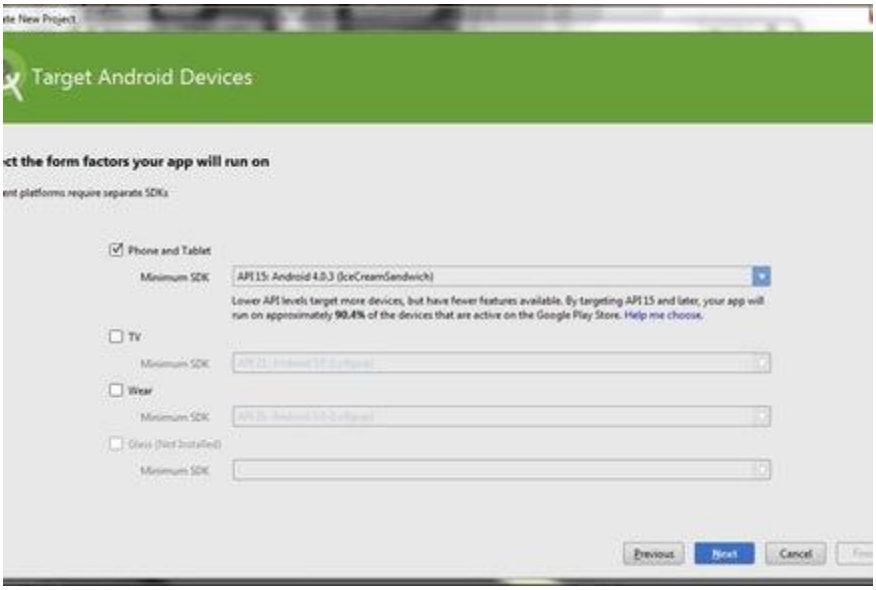
1. Go to http://developer.android.com/sdk/index.html to download Android Studio.

2. Use the installer to install Android Studio following its instructions.



Step 2: Open a New Project





1. Open Android Studio.
2. Under the "Quick Start" menu, select "Start a new Android Studio project."
3. On the "Create New Project" window that opens, name your project "HelloWorld".
4. If you choose to, set the company name as desired*.
5. Note where the project file location is and change it if desired.
6. Click "Next."
7. Make sure on that "Phone and Tablet" is the only box that is checked.
8. If you are planning to test the app on your phone, make sure the minimum SDK is below your phone's operating system level.

9. Click "Next."

10. Select "Blank Activity."

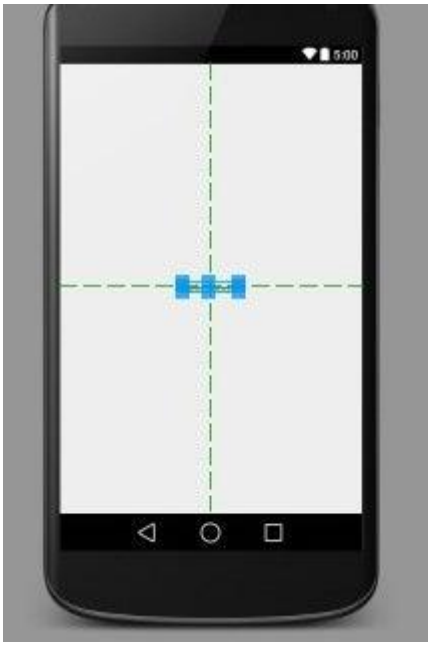
11. Click "Next."

12. Leave all of the Activity name fields as they are.

13. Click "Finish."

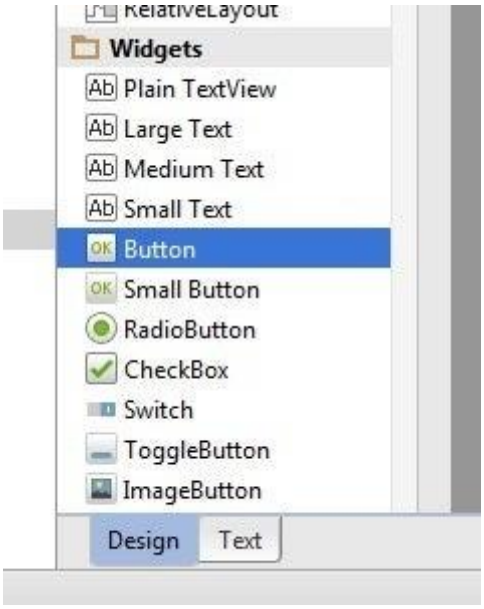
***Note:** It is typical naming convention in Android projects to set the company name as some form of "example.name.here.com".

Step 3: Edit the Welcome Message in the Main Activity



1. Navigate to the activity_main.xml tab if it is not already open.
2. Make sure that the Design tab is open on the activity_main.xml display.
3. Click and drag the "Hello, world!" from the upper left corner of the phone display to the center of the screen.
4. In the project file system on the left side of the window, open the values folder.
5. In the values folder, double-click the strings.xml file.
6. In this file, find the line "Hello world!".
7. After the "Hello world!" message, add "Welcome to my app!"
8. Navigate back to the activity_main.xml tab.
9. Make sure that your centered text now reads "Hello world! Welcome to my app!"

Step 4: Add a Button to the Main Activity



1. Navigate to the Design tab of the activity_main.xml display.

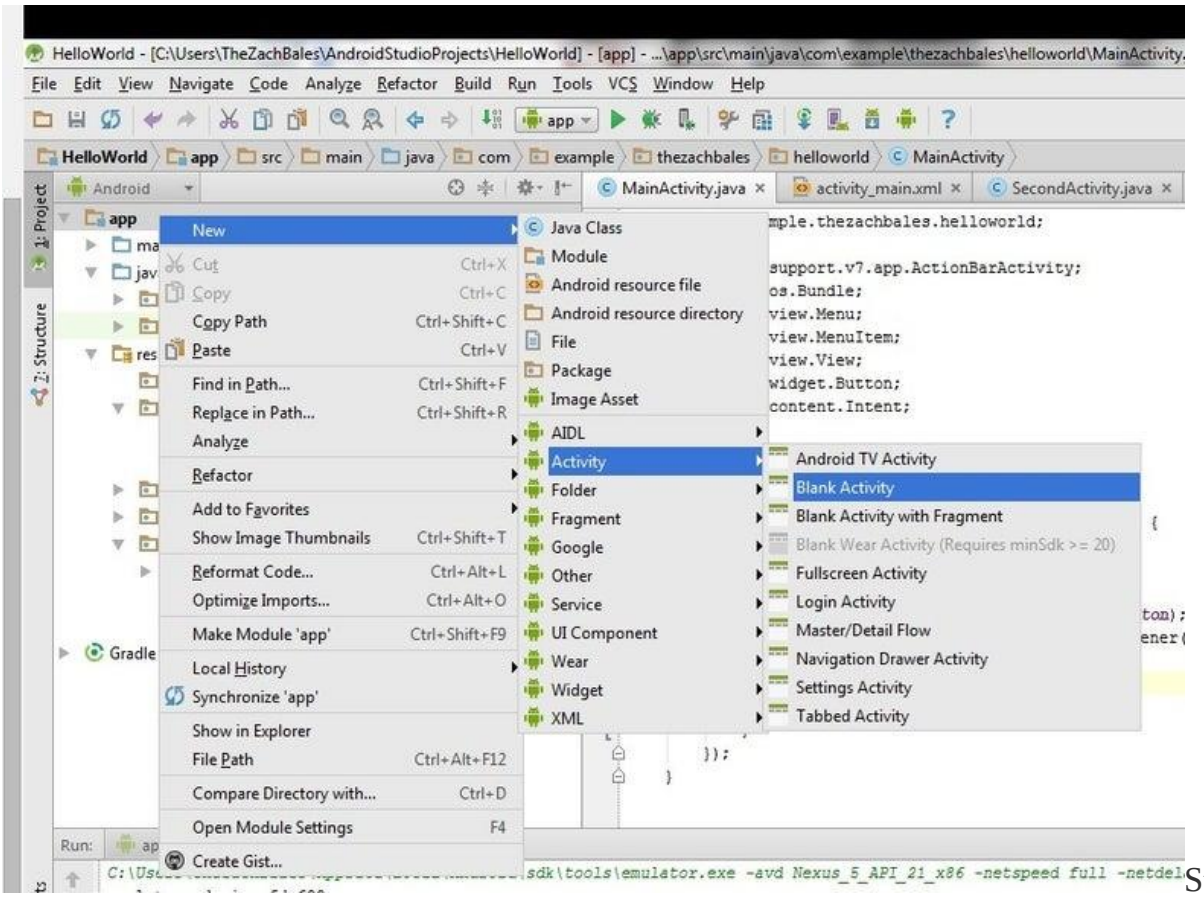
2. In the Palette menu to the left of the phone display, find Button (under the heading Widgets).

3. Click and drag Button to be centered underneath your welcome message.

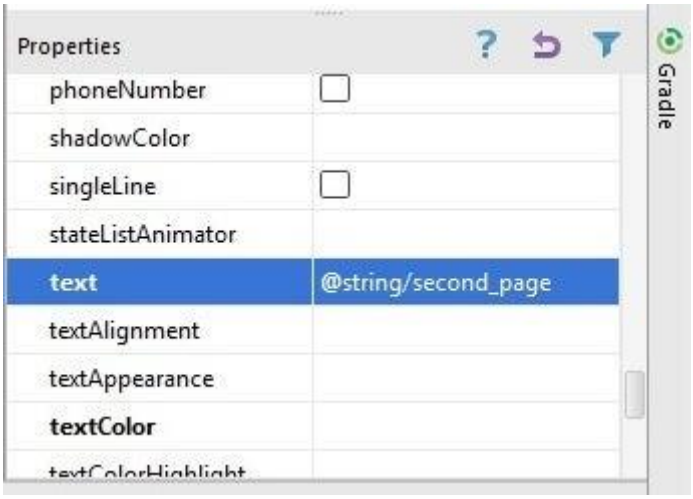
4. Make sure your button is still selected.

5. In the Properties menu (on the right side of the window), scroll down to find the field for "text."

6. Change the text from "New Button" to "Next Page."



Step 5: Create a Second Activity



1. At the top of the project's file system tree, right click on "app."
2. Navigate through to New > Activity > Blank Activity.
3. Change the name of this activity to "SecondActivity".
4. Click "Finish."
5. Make sure you are in the Design view of activity_second.xml.
6. Drag the text box in the upper left of the phone display down to the center as you did on the Main Activity.
7. With the text box still selected, find the "id" field in the Properties menu on the right, and set it to "text2".

8. Open strings.xml again.

9. Add a new line under "Hello world! Welcome to my app!" that reads "Welcome to the second page!".

10. Navigate back to activity_second.xml.

11. Select the text box again.

12. In the Properties pane, set the "text" field to "@string/second_page".

13. Make sure that the text box now reads "Welcome to the second page!" and is in the center of the screen in the phone display.

Step 6: Write the Button's "onClick" Method

```
. = (Button) findViewById(R.ic
'lickListener(new View.OnClicl

id onClick(View v) {
econdActivity();


    ondActivity()

    ew Intent(this, Seco
    ent);


ndroid.support.v7.app.ActionB
ndroid.os.Bundle;
ndroid.view.Menu;
ndroid.view.MenuItem;
ndroid.view.View;
ndroid.widget.Button;
ndroid.content.Intent;
```

1. Select the MainActivity.java tab along the top of the work environment.

2. Add the following lines of code at the end of the onCreate method:

```
Button button = (Button) findViewById(R.id.button);

button.setOnClickListener(new View.onClickListener() {

@Override

public void onClick(View v) {

goToSecondActivity();

}

});
```

3. Add the following method to the bottom of the MainActivity class:

```
private void goToSecondActivity() {

Intent intent = new Intent(this, SecondActivity.class);

startActivity(intent);

}
```

4. Click the + next to import at the third line of MainActivity.java to expand the import statements.

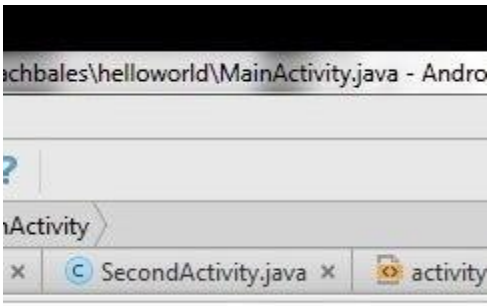
5. Add the following to the end of the import statements if they are not already there:

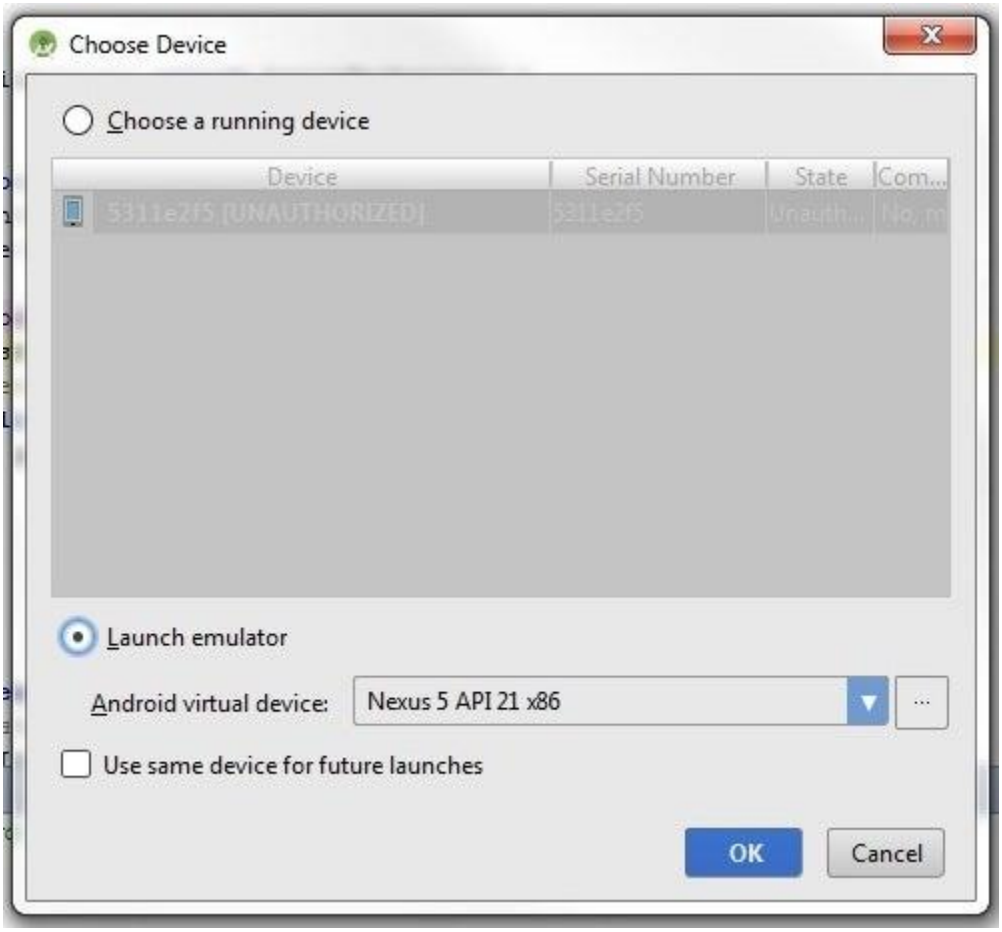
```
import android.content.Intent;
```

import android.view.View;

import android.widget.TextView;

Step 7: Test the Application





1. Click the green play symbol from the toolbar at the top of the Android Studio window.

2. When the "Choose Device" dialog apperas (this may take a few moments), select the "Lauch emulator" option.

3. Click OK.

4. When the emulator opens (this too could take awhile), the app will automatically launch the app upon the virtual phone being unlocked.

5. Make sure that all of your text displays correctly and that the button takes you to the next page.

Step 8: Up, Up, and Away!

Congrats! You've now completed your first Android application with some basic functionality. Your finished app should have a page greeting the user and a button that takes the user to a second page.

From here you have the cursory knowledge you need to go on to learn all there is to know about Android application development.

Anatomy of Android Application

There are four building blocks to an Android application:

- Activity
- Intent Receiver
- Service
- Content Provider

Not every application needs to have all four, but your application will be written with some combination of these.

Once you have decided what components you need for your application, you should list them in a file called AndroidManifest.xml. This is an XML file where you declare the components of your application and what their capabilities and requirements are. We will discuss soon, what the AndroidManifest.xml is responsible for.

Activity :

Activities are the most common of the four Android building blocks. An activity is usually a single screen in your application. Each activity is implemented as a single class that extends the Activity base class. Your class will display a user interface composed of Views and respond to events. Most applications consist of multiple screens. For example, a text messaging application might have one screen that shows a list of contacts to send messages to, a second screen to write the message to the chosen contact, and other screens to review old messages or change settings. Each of these screens would be implemented as an activity.

When a new screen opens, the previous screen is paused and put onto a history stack. The user can navigate backward through previously opened screens in the history. Screens can also choose to be removed from the history stack when it would be inappropriate for them to remain. Android retains history stacks for each application launched from the home screen.

Intent and Intent Filters :

Android uses a special class called Intent to move from screen to screen. Intent describe what an application wants done. The two most important parts of the intent data structure are the action and the data to act upon. Typical values for action are MAIN (the front door of the application), VIEW, PICK, EDIT, etc. The data is expressed as a Uniform Resource Indicator (URI). For example, to view a website in the browser, you would create an Intent with the VIEW action and the data set to a Website-URI.

```
new Intent(android.content.Intent.VIEW_ACTION, ContentURI.create("http
://anddev.org"));
```

There is a related class called an IntentFilter. While an intent is effectively a request to do something, an intent filter is a description of what intents an activity (or intent receiver, see below) is capable of handling. Activities publish their IntentFilters in the AndroidManifest.xml file.

Intent Receiver :
You can use an IntentReceiver when you want code in your application to execute in reaction to an external event, for example, when the phone rings, or when the data network is available, or when it's midnight. Intent receivers do not display a UI, although they may display Notifications to alert the user if something interesting has happened. Intent receivers are also registered in AndroidManifest.xml, but you can also register them from code using Context.registerReceiver().

Service :
A Service is code that is long-lived and runs without a UI. A good example of this is a media player playing songs from a play list. In a media player application, there would probably be one or more activities that allow the user to choose songs and start playing

them. However, the music playback itself should not be handled by an activity because the user will expect the music to keep playing even after navigating to a new screen. In this case, the media player activity could start a service using Context.startService() to run in the background to keep the music going. The system will then keep the music playback service running until it has finished. (You can learn more about the priority given to services in the system by reading Life Cycle of an Android Application.) Note that you can connect to a service (and start it if it's not already running) with the Context.bindService() method. When connected to a service, you can communicate with it through an interface exposed by the service. For the music service, this might allow you to pause, rewind, etc.

Content Provider :
Applications can store their data in files, a SQLite database, preferences or any other mechanism that makes sense. A content provider, however, is useful if you want your application's data to be shared with other applications. A content provider is a class that implements a standard set of methods to let other applications store and retrieve the type of data that is handled by that content provider.

Android Terminology Topics

Android SDK(Software Development Kit)

- Like JDK of Java, The Android SDK (Software Development Kit) is a set of development tools and libraries provided by Google for creating applications for the Android operating system.
- It includes a variety of components that enable developers to build, test, and debug Android apps.
- Developers use the Android SDK to create a wide range of applications, including games, productivity apps, social media apps, and much more.
- The SDK supports various programming languages, but Java and Kotlin are the most commonly used languages for Android app development.
- Before starting Android app development, we need to download and install Android Studio, which comes bundled with the Android SDK and all the necessary tools to get started.
- Some key components of the Android SDK are as follows:-
    -

- **Android Emulator/Android Virtual Device(AVD)**:
  - The Android Emulator is a virtual device that allows us to run and test Android applications on our computer without needing a physical device.
  - The Android Emulator allows developers to run and test their apps on a virtual Android device without the need for a physical device.
  - The Android Emulator emulates the hardware and software of an Android device, making it an essential tool for Android development.
  - It comes with various pre-configured device profiles and allows for the simulation of different Android versions, API levels, and screen sizes.
  - it tests apps on different Android versions and device configurations without needing multiple physical devices.
  - It's an essential tool for app testing across different Android versions and device configurations i.e. it simulates real-world conditions like incoming calls, text messages, location, network conditions, and more.
  - It emulates different device configurations, including various screen sizes, resolutions, and hardware specifications.
  - It supports hardware acceleration for faster performance.
  - It allows snapshot saving and quick boot for faster startup times.
  - It seamlessly integrates with Android Studio for easy deployment and testing of the applications.
  - It provides access to the Google Play Store (in some configurations) for testing app compatibility with Google services.
  - ***To Activate/Optimize Emulator Performance***
    - Enable Hardware Acceleration by ensuring that Intel *HAXM* (Hardware Accelerated Execution Manager) is installed on Intel-based CPUs/systems.

- Enable Hardware Acceleration for AMD processors, by enabling Hyper-V or WHPX (Windows Hypervisor Platform) options.
- Increase the amount of RAM as possible and CPU cores allocated to the emulator for better performance.
- Regularly check for updates to the emulator and system images to benefit from performance improvements and new features.

- ***To Configure or Set up the Android Emulator***
  - Download and install Android Studio from the official website.
  - Open/Launch Android Studio.
  - Go to `Tools` menu > click `AVD Manager`, or click the AVD Manager icon in the toolbar.
  - Click “Create Virtual Device” and choose a device definition that matches the type of device we want to emulate (e.g., Pixel 4, Nexus 5X).
  - Select the system image by choosing an Android version (API level) from the lists for the virtual device. Download it if it’s not already installed.
  - Configure AVD Settings by customizing advanced settings like RAM, internal storage, and more if needed.
  - Review the configuration and click “Finish” to create the AVD (Android Virtual Device).

- ***To Launch/Execute the Emulator***
  - Start/Open the Emulator in the AVD Manager, by clicking the play button next to the virtual device to start the emulator.
  - Deploy the created App by running the app from Android Studio, and choosing the emulator as the target device.

- **Android Debug Bridge (ADB)**:
  -
  - ADB is a key part of the Android SDK.
  - Android Debug Bridge (ADB) is a versatile command-line tool that lets us communicate between a developer's computer and a connected Android device or emulator.
  - It is used to perform various tasks/actions, such as installing and debugging apps, copying files back and forth, transferring files, and accessing the device's shell.
  - It provides a command-line interface to the Android device.
  - It executes commands directly on the device.
  - It is used to Install, uninstall, and update apps.
  - It is helpful to clear app data and cache.
  - It starts and stops services, view system information, and more

Android Terminology

Android SDK(Software Development Kit)

- Like JDK of Java, The Android SDK (Software Development Kit) is a set of development tools and libraries provided by Google for creating applications for the Android operating system.
- It includes a variety of components that enable developers to build, test, and debug Android apps.
- Developers use the Android SDK to create a wide range of applications, including games, productivity apps, social media apps, and much more.
- The SDK supports various programming languages, but Java and Kotlin are the most commonly used languages for Android app development.

- Before starting Android app development, we need to download and install Android Studio, which comes bundled with the Android SDK and all the necessary tools to get started.
- Some key components of the Android SDK are as follows:-
    - **Android Emulator/Android Virtual Device(AVD)**:
        - The Android Emulator is a virtual device that allows us to run and test Android applications on our computer without needing a physical device.
        - The Android Emulator allows developers to run and test their apps on a virtual Android device without the need for a physical device.
        - The Android Emulator emulates the hardware and software of an Android device, making it an essential tool for Android development.
        - It comes with various pre-configured device profiles and allows for the simulation of different Android versions, API levels, and screen sizes.
        - it tests apps on different Android versions and device configurations without needing multiple physical devices.
        - It's an essential tool for app testing across different Android versions and device configurations i.e. it simulates real-world conditions like incoming calls, text messages, location, network conditions, and more.
        - It emulates different device configurations, including various screen sizes, resolutions, and hardware specifications.
        - It supports hardware acceleration for faster performance.
        - It allows snapshot saving and quick boot for faster startup times.
        - It seamlessly integrates with Android Studio for easy deployment and testing of the applications.

- It provides access to the Google Play Store (in some configurations) for testing app compatibility with Google services.
- ***To Activate/Optimize Emulator Performance***
    - Enable Hardware Acceleration by ensuring that Intel *HAXM* (Hardware Accelerated Execution Manager) is installed on Intel-based CPUs/systems.
    - Enable Hardware Acceleration for AMD processors, by enabling Hyper-V or WHPX (Windows Hypervisor Platform) options.
    - Increase the amount of RAM as possible and CPU cores allocated to the emulator for better performance.
    - Regularly check for updates to the emulator and system images to benefit from performance improvements and new features.
- ***To Configure or Set up the Android Emulator***
    - Download and install Android Studio from the official website.
    - Open/Launch Android Studio.
    - Go to `Tools` menu > click `AVD Manager`, or click the AVD Manager icon in the toolbar.
    - Click “Create Virtual Device” and choose a device definition that matches the type of device we want to emulate (e.g., Pixel 4, Nexus 5X).
    - Select the system image by choosing an Android version (API level) from the lists for the virtual device. Download it if it’s not already installed.
    - Configure AVD Settings by customizing advanced settings like RAM, internal storage, and more if needed.
    - Review the configuration and click “Finish” to create the AVD (Android Virtual Device).

- ***To Launch/Execute the Emulator***
    - Start/Open the Emulator in the AVD Manager, by clicking the play button next to the virtual device to start the emulator.
    - Deploy the created App by running the app from Android Studio, and choosing the emulator as the target device.
- **Android Debug Bridge (ADB)**:
    -
    - ADB is a key part of the Android SDK.
    - Android Debug Bridge (ADB) is a versatile command-line tool that lets us communicate between a developer's computer and a connected Android device or emulator.
    - It is used to perform various tasks/actions, such as installing and debugging apps, copying files back and forth, transferring files, and accessing the device's shell.
    - It provides a command-line interface to the Android device.
    - It executes commands directly on the device.
    - It is used to Install, uninstall, and update apps.
    - It is helpful to clear app data and cache.
    - It starts and stops services, view system information, and more.

What is Context in Android?

Last Updated : 15 Jul, 2024

Android Applications are popular for a long time and it is evolving to a greater level as users' expectations are that they need to view the data that they want in an easier

smoother view. Hence, the android developers must know the important terminologies before developing the app. In Android Programming we generally come across the word **Context**. So what exactly is this Context and why is it so important? To answer this question let's first see what the literal meaning of Context is:

The Circumstances that form the setting for an Event, Statement, or Idea, and in terms of which it can be fully understood.

Looking at this definition we come across two things:

- The Context tells us about the surrounding information.
- It is very important to understand the environment which we want to understand.

Similarly when we talk about Android Programming Context can be understood as something which gives us the Context of the current state of our application. **We can break the Context and its use into three major points:**

- It allows us to access resources.
- It allows us to interact with other Android components by sending messages.
- It gives you information about your app environment.

The code has been given in both **Java and Kotlin Programming Language for Android.**

Understanding Context by a Real World Example

Let's a person visit a hotel. He needs breakfast, lunch, and dinner at a suitable time. Except for these things there are also many other things, he wants to do during his time of stay. So how does he get these things? He will ask the room-service person to bring these things for him. Right? So here the **room-service person is the Context** considering **you are the single activity** and **the hotel to be your app**, finally, the **breakfast, lunch & dinner have to be the resources**.

How Does this Work?

1. It is the Context of the current/active state of the application.

Usually, the app got multiple screens like display/inquiry/add/delete screens(A general requirement of a basic app). So when the user is searching for something, the Context is an inquiry screen in this case.

2. It is used to get information about the activity and application.

The inquiry screen's Context specifies that the user is in inquiry activity, and he/she can submit queries related to the app

3. It is used to get access to resources, databases, shared preferences, etc.

Via Rest services, API calls can be consumed in android apps. Rest Services usually hold database data and provide the output in JSON format to the android app. The Context for the respective screen helps to get hold of database data and the shared data across screens

4. Both the Activity and Application classes extend the Context class.

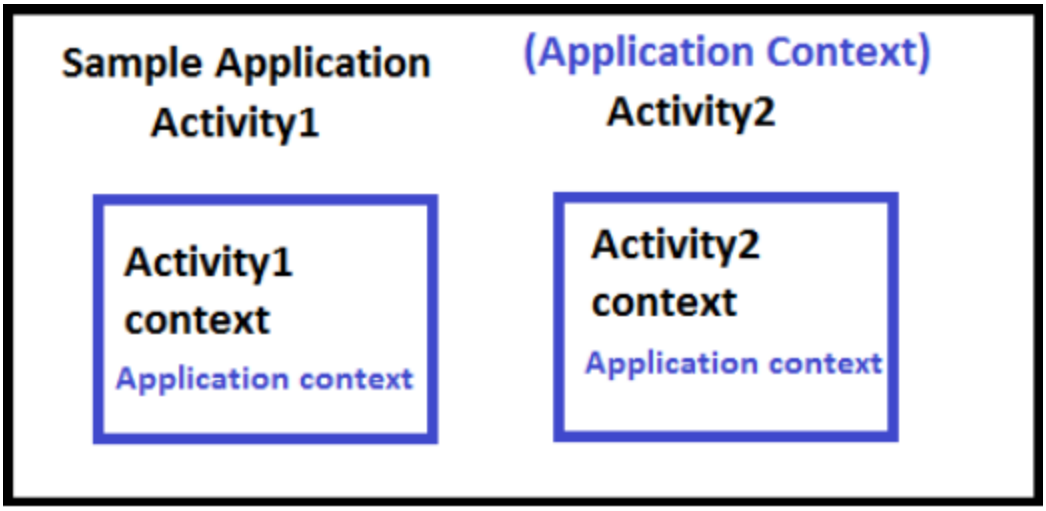
In android, Context is the main important concept and the wrong usage of it leads to memory leakage. Activity refers to an individual screen and Application refers to the whole app and both extend the Context class.

Types of Context in Android

There are mainly two types of Context that are available in Android.

1. **Application Context and**
2. **Activity Context**

The Overall view of the App hierarchy looks like the following:



It can be seen in the above image that in “Sample Application”, the nearest Context is Application Context. In “Activity1” and “Activity2”, both Activity Context (Here it is Activity1 Context for Activity1 and Activity2 Context for Activity2) and Application Context.The nearest Context to both is their Activity Context only.

Application Context

This Context is tied to the **Lifecycle of an Application**. Mainly it is an instance that is a singleton and can be accessed via **getApplicationContext()**. Some use cases of Application Context are:

- If it is necessary to create a singleton object
- During the necessity of a library in an activity

getApplicationContext():

It is used to return the Context which is linked to the Application which holds all activities running inside it. When we call a method or a constructor, we often have to pass a Context and often we use **“this”** to pass the activity Context or **“getApplicationContext”** to pass the application Context. This method is generally used for the application level and can be used to refer to all the activities. For example, if we want to access a variable throughout the android app, one has to use it via **getApplicationContext()**.

Example:

JavaKotlin

```
import android.app.Application;

public class GlobalExampleClass extends Application {
    private String globalName;
    private String globalEmail;

    public String getName() {
        return globalName;
    }

    public void setName(String aName) {
        globalName = aName;
    }

    public String getEmail() {
        return globalEmail;
    }

    public void setEmail(String aEmail) {
        globalEmail = aEmail;
    }
}
```

Inside the activity class, set the name and email of GlobalExampleClass, which can be accessed from another activity. Let us see via the below steps.

Syntax:

```
// Activity 1
public class <your activity1> extends Activity {
  ........
  ........
  private <yourapplicationname> globarVar;
  ........
  @Override
  public void onCreate(Bundle savedInstanceState) {
    .......
    final GlobalExampleClass globalExampleVariable =
(GlobalExampleClass) getApplicationContext();

    // In this activity set name and email and can reuse in other activities
    globalExampleVariable.setName("getApplicationContext example");
    globalExampleVariable.setEmail("xxxxxx@gmail.com");

    .......
}
```

```
// Activity 2
public class <your activity2> extends Activity {
  ........
  ........
  private <yourapplicationname> globarVar;
  .......
  @Override
  public void onCreate(Bundle savedInstanceState) {
    .......
    final GlobalExampleClass globalExampleVariable =
(GlobalExampleClass) getApplicationContext();

    // As in activity1, name and email is set, we can retrieve it here
    final String globalName  = globalExampleVariable.getName();
    final String globalEmail = globalExampleVariable.getEmail();


    .......
}
```

So, whenever the variable scope is required throughout the application, we can get it by means of **getApplicationContext()**. Following is a list of functionalities of Application Context.

List of functionalities of Application Context:

- Load Resource Values
- Start a Service
- Bind to a Service
- Send a Broadcast
- Register BroadcastReceiver

Activity Context

It is the activity Context meaning each and every screen got an activity. For example, EnquiryActivity refers to EnquiryActivity only and AddActivity refers to AddActivity only. It is tied to the life cycle of activity. It is used for the current Context. The method of invoking the Activity Context is **getContext()**.

Some use cases of Activity Context are:

- The user is creating an object whose lifecycle is attached to an activity.
- Whenever inside an activity for UI related kind of operations like toast, dialogue, etc.,

getContext():

It returns the Context which is linked to the Activity from which it is called. This is useful when we want to call the Context from only the current running activity.

Example:

JavaKotlin

```
@Override
public void onItemClick(AdapterView<?> parent, View view, int pos, long id) {
```

```
    // view.getContext() refers to the current activity view
    // Here it is used to start the activity
    Intent intent = new Intent(view.getContext(), <your java
classname>::class.java);
    intent.putExtra(pid, ID);
    view.getContext().startActivity(intent);
}
```

List of Functionalities of Activity Context:

- Load Resource Values
- Layout Inflation
- Start an Activity
- Show a Dialog
- Start a Service
- Bind to a Service
- Send a Broadcast
- Register BroadcastReceiver

From the functionalities of both Application and Activity, we can see that the difference is that the **Application Context** is **not related to UI**. It should be used only to start a service or load resource values etc. Apart from **getApplicationContext() and getContext()**, **getBaseContext() or this** are the different terminologies used throughout the app development. Let us see with an example

getBaseContext():

The base Context is set by the constructor or **setBaseContext()**.This method is only valid if we have a **ContextWrapper**. Android provides a ContextWrapper class that is created around an existing Context using:

```
ContextWrapper wrapper = new ContextWrapper(context);
```

The benefit of using a ContextWrapper is that it lets you **"modify behavior without changing the original Context"**.

Syntax:

```
public <YourHandler>(Context ctx) {
    // if the context is instanceof ContextWrapper
    while (ctx instanceof ContextWrapper) {
        // use getBaseContext()
        final Context baseContext =
((ContextWrapper)context).getBaseContext();
        if (baseContext == null) {
            break;
        }
        // And then we can assign to context and reuse that
        ctx = baseContext;
    }
}
```

this:

"this" argument is of a type "Context". To explain this Context let's take an example to show a Toast Message using "this".

Example:

Java Kotlin

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

// Show a simple toast message, that can be done after doing some activities
// Toast.makeText(this, "Action got completed", Toast.LENGTH_SHORT).show();

public class ExampleActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main_example);

        // Displaying Toast
        Toast.makeText(this,"Action done",Toast.LENGTH_SHORT).show();
    }
}
```

Another example to start the activity using "this":

Java Kotlin

```java
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;

public class FirstActivity extends AppCompatActivity {

    public static final String newMessage = "Your message to go for next screen";

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

    // There must be some button and on click of
    // that below method can be invoked
    public void sendMessageToNextScreen(View view) {
        // Here it is used with "this"
        Intent intent = new Intent(this, SecondActivity.class);
        EditText editText = (EditText) findViewById(R.id.editText);
```

```
        String message = editText.getText().toString();
        intent.putExtra(newMessage, message);

        // Start the SecondActivity
        startActivity(intent);
    }
}
```


Servlet - Context Event and Context Listener

ServletContextEvent class provides alerts/notifications for changes to a web application's servlet context. ServletContextListener is a class that receives alerts/notifications about changes to the servlet context and acts on them. When the context is initialized and deleted, the ServletContextListener is utilized to conduct crucial tasks. In a nut

3 min read

Spring MVC context:annotation-config VS context:component-scan

In this article, we will learn about Spring MVC <context:annotation-config> vs <context:component-scan>. Activating all annotations existing in Java beans

that have already been registered either by definition in an application context file or by component scanning is the primary function of annotation-config. The requirement for regist

3 min read

Introduction to activities

The Activity class is a crucial component of an Android app, and the way activities are launched and put together is a fundamental part of the platform's application model. Unlike programming paradigms in which apps are launched with a main() method, the Android system initiates code in an Activity instance by invoking specific callback methods that correspond to specific stages of its lifecycle.

This document introduces the concept of activities, and then provides some lightweight guidance about how to work with them. For additional information about best practices in architecting your app, see Guide to App Architecture.

The concept of activities

The mobile-app experience differs from its desktop counterpart in that a user's interaction with the app doesn't always begin in the same place. Instead, the user journey often begins non-deterministically. For instance, if you open an email app from your home screen, you might see a list of emails. By contrast, if you are using a social media app that then launches your email app, you might go directly to the email app's screen for composing an email.

The Activity class is designed to facilitate this paradigm. When one app invokes another, the calling app invokes an activity in the other app, rather than the app as an atomic whole. In this way, the activity serves as the entry point for an app's interaction with the user. You implement an activity as a subclass of the Activity class.

An activity provides the window in which the app draws its UI. This window typically fills the screen, but may be smaller than the screen and float on top of other windows. Generally, one activity implements one screen in an app. For instance, one of an app’s activities may implement a *Preferences* screen, while another activity implements a *Select Photo* screen.

Most apps contain multiple screens, which means they comprise multiple activities. Typically, one activity in an app is specified as the *main activity*, which is the first screen to appear when the user launches the app. Each activity can then start another activity in order to perform different actions. For example, the main activity in a simple e-mail app may provide the screen that shows an e-mail inbox. From there, the main activity might launch other activities that provide screens for tasks like writing e-mails and opening individual e-mails.

Although activities work together to form a cohesive user experience in an app, each activity is only loosely bound to the other activities; there are usually minimal dependencies among the activities in an app. In fact, activities often start up activities belonging to other apps. For example, a browser app might launch the Share activity of a social-media app.

To use activities in your app, you must register information about them in the app's manifest, and you must manage activity lifecycles appropriately. The rest of this document introduces these subjects.

Configuring the manifest

For your app to be able to use activities, you must declare the activities, and certain of their attributes, in the manifest.

Declare activities

To declare your activity, open your manifest file and add an <activity> element as a child of the <application> element. For example:

```
<manifest ... >
  <application ... >
      <activity android:name=".ExampleActivity" />
      ...
  </application ... >
  ...
</manifest >
```

The only required attribute for this element is android:name, which specifies the class name of the activity. You can also add attributes that define activity characteristics such as label, icon, or UI theme. For more information about these and other attributes, see the <activity> element reference documentation.

Note: After you publish your app, you should not change activity names. If you do, you might break some functionality, such as app shortcuts. For more information on changes to avoid after publishing, see Things That Cannot Change.

Declare intent filters

Intent filters are a very powerful feature of the Android platform. They provide the ability to launch an activity based not only on an *explicit* request, but also an *implicit* one. For example, an explicit request might tell the system to “Start the Send Email activity in the Gmail app". By contrast, an implicit request tells the system to “Start a Send Email screen in any activity that can do the job." When the system UI asks a user which app to use in performing a task, that’s an intent filter at work.

You can take advantage of this feature by declaring an <intent-filter> attribute in the <activity> element. The definition of this element includes an <action> element and, optionally, a <category> element and/or a <data> element. These elements combine to specify the type of intent to which your activity can respond. For example, the following code snippet shows how to configure an activity that sends text data, and receives requests from other activities to do so:

```
<activity android:name=".ExampleActivity" android:icon="@drawable/app_icon">
    <intent-filter>
        <action android:name="android.intent.action.SEND" />
        <category android:name="android.intent.category.DEFAULT" />
        <data android:mimeType="text/plain" />
    </intent-filter>
</activity>
```

In this example, the <action> element specifies that this activity sends data. Declaring the <category> element as DEFAULT enables the activity to receive launch requests. The <data> element specifies the type of data that this activity can send. The following code snippet shows how to call the activity described above:

KotlinJava

```
val sendIntent = Intent().apply {
    action = Intent.ACTION_SEND
    type = "text/plain"
    putExtra(Intent.EXTRA_TEXT, textMessage)
}
startActivity(sendIntent)
```

If you intend for your app to be self-contained and not allow other apps to activate its activities, you don't need any other intent filters. Activities that you don't want to make available to other applications should have no intent filters, and you can start them yourself using explicit intents. For more information about how your activities can respond to intents, see Intents and Intent Filters.

Declare permissions

You can use the manifest's <activity> tag to control which apps can start a particular activity. A parent activity cannot launch a child activity unless both activities have the same permissions in their manifest. If you declare a <uses-permission> element for a parent activity, each child activity must have a matching <uses-permission> element.

For example, if your app wants to use a hypothetical app named SocialApp to share a post on social media, SocialApp itself must define the permission that an app calling it must have:

```
<manifest>
<activity android:name="...."
   android:permission=”com.google.socialapp.permission.SHARE_POST”

/>
```

Then, to be allowed to call SocialApp, your app must match the permission set in SocialApp's manifest:

```
<manifest>
   <uses-permission android:name="com.google.socialapp.permission.SHARE_POST" />
</manifest>
```

For more information on permissions and security in general, see Security and Permissions.

Managing the activity lifecycle

Over the course of its lifetime, an activity goes through a number of states. You use a series of callbacks to handle transitions between states. The following sections introduce these callbacks.

onCreate()

You must implement this callback, which fires when the system creates your activity. Your implementation should initialize the essential components of your activity: For example, your app should create views and bind data to lists here. Most importantly, this is where you must call setContentView() to define the layout for the activity's user interface.

When onCreate() finishes, the next callback is always onStart().

onStart()

As onCreate() exits, the activity enters the Started state, and the activity becomes visible to the user. This callback contains what amounts to the activity's final preparations for coming to the foreground and becoming interactive.

onResume()

The system invokes this callback just before the activity starts interacting with the user. At this point, the activity is at the top of the activity stack, and captures all user input. Most of an app's core functionality is implemented in the onResume() method.

The onPause() callback always follows onResume().

onPause()

The system calls onPause() when the activity loses focus and enters a Paused state. This state occurs when, for example, the user taps the Back or Recents button. When the system calls onPause() for your activity, it technically means your activity is still partially visible, but most often is an indication that the user is leaving the activity, and the activity will soon enter the Stopped or Resumed state.

An activity in the Paused state may continue to update the UI if the user is expecting the UI to update. Examples of such an activity include one showing a navigation map screen or a media player playing. Even if such activities lose focus, the user expects their UI to continue updating.

You should **not** use onPause() to save application or user data, make network calls, or execute database transactions. For information about saving data, see Saving and restoring activity state.

Once onPause() finishes executing, the next callback is either onStop() or onResume(), depending on what happens after the activity enters the Paused state.

onStop()

The system calls onStop() when the activity is no longer visible to the user. This may happen because the activity is being destroyed, a new activity is starting, or an existing activity is entering a Resumed state and is covering the stopped activity. In all of these cases, the stopped activity is no longer visible at all.

The next callback that the system calls is either onRestart(), if the activity is coming back to interact with the user, or by onDestroy() if this activity is completely terminating.

onRestart()

The system invokes this callback when an activity in the Stopped state is about to restart. onRestart() restores the state of the activity from the time that it was stopped.

This callback is always followed by onStart().

onDestroy()

The system invokes this callback before an activity is destroyed.

This callback is the final one that the activity receives. onDestroy() is usually implemented to ensure that all of an activity's resources are released when the activity, or the process containing it, is destroyed.

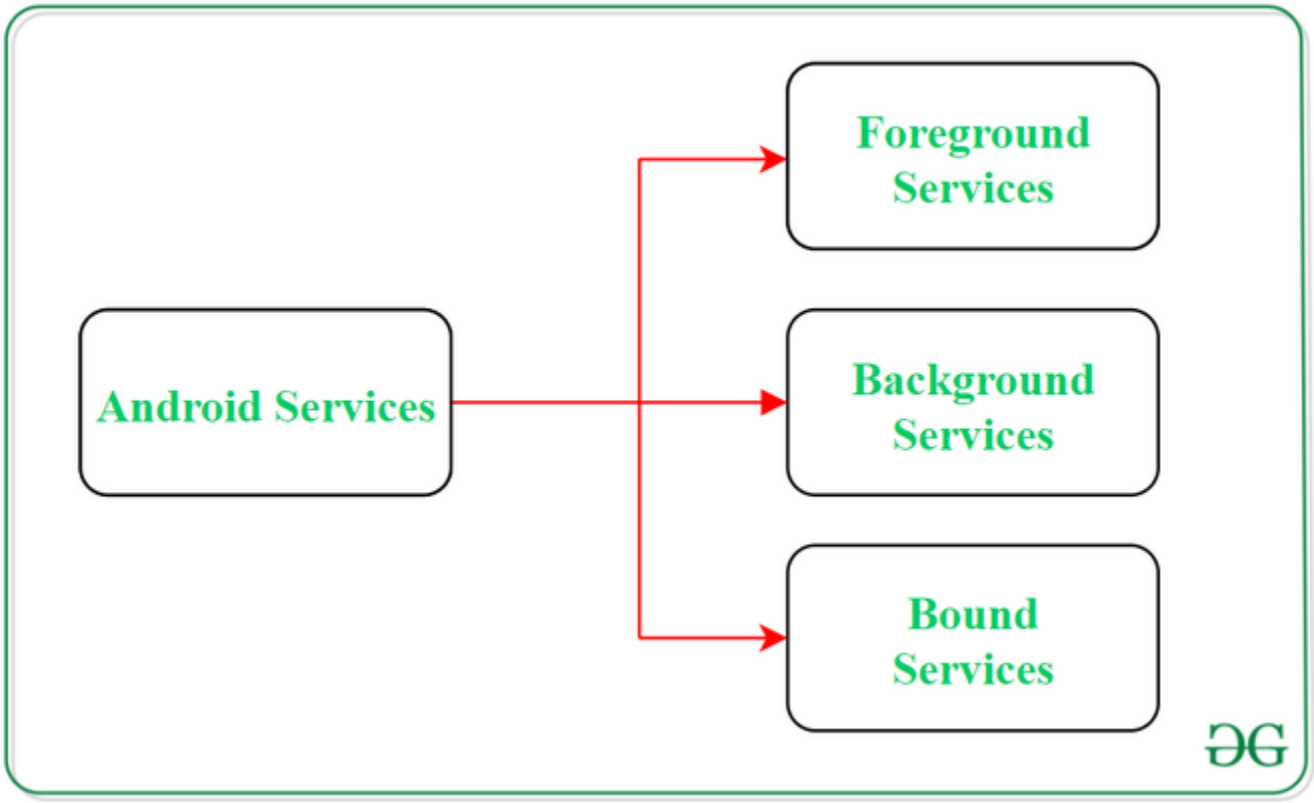
This section provides only an introduction to this topic. For a more detailed treatment of the activity lifecycle and its callbacks, see The Activity Lifecycle.

Services in Android with Example

Last Updated : 08 Apr, 2024

Services in Android are a special component that facilitates an application to run in the background in order to perform long-running operation tasks. The prime aim of a service is to ensure that the application remains active in the background so that the user can operate multiple applications at the same time. A user-interface is not desirable for android services as it is designed to operate long-running processes without any user intervention. A service can run continuously in the background even if the application is closed or the user switches to another application. Further, application components can bind itself to service to carry out inter-process communication(IPC). There is a major difference between android services and threads, one must not be confused between the two. Thread is a feature provided by the Operating system to allow the user to perform operations in the background. While service is an android component that performs a long-running operation about which the user might not be aware of as it does not have UI.

Types of Android Services



1. Foreground Services:
Services that notify the user about its ongoing operations are termed as Foreground Services. Users can interact with the service by the notifications provided about the ongoing task. Such as in downloading a file, the user can keep track of the progress in downloading and can also pause and resume the process.
2. Background Services:
Background services do not require any user intervention. These services do not notify the user about ongoing background tasks and users also cannot access them. The process like schedule syncing of data or storing of data fall under this service.
3. Bound Services:
This type of android service allows the components of the application like activity to bound themselves with it. Bound services perform their task as long as any application component is bound to it. More than one component is allowed to bind themselves with a service at a time. In order to bind an application component with a service bindService() method is used.

The Life Cycle of Android Services

In android, services have 2 possible paths to complete its life cycle namely Started and Bounded.

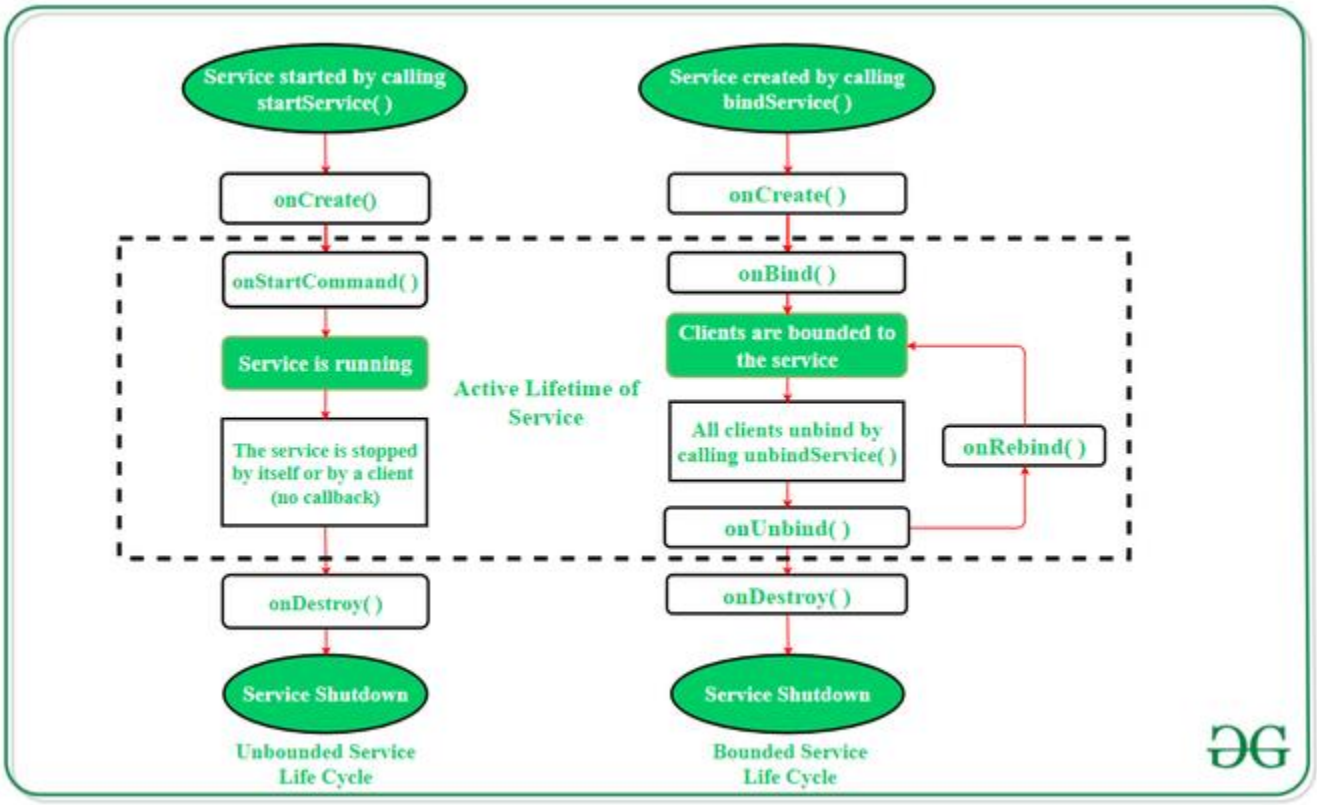
1. Started Service (Unbounded Service):

By following this path, a service will initiate when an application component calls the startService() method. Once initiated, the service can run continuously in the background even if the component is destroyed which was responsible for the start of the service. Two option are available to stop the execution of service:

- By calling stopService() method,
- The service can stop itself by using stopSelf() method.

2. Bounded Service:

It can be treated as a server in a client-server interface. By following this path, android application components can send requests to the service and can fetch results. A service is termed as bounded when an application component binds itself with a service by calling bindService() method. To stop the execution of this service, all the components must unbind themselves from the service by using unbindService() method.



To carry out a downloading task in the background, the startService() method will be called. Whereas to get information regarding the download progress and to pause or resume the process while the application is still in the background, the service must be bounded with a component which can perform these tasks.

Fundamentals of Android Services

A user-defined service can be created through a normal class which is extending the class Service. Further, to carry out the operations of service on applications, there

are certain callback methods which are needed to be overridden. The following are some of the important methods of Android Services:

Methods	Description
onStartCommand()	The Android service calls this method when a component(eg: activity) requests to start a service using startService(). Once the service is started, it can be stopped explicitly using stopService() or stopSelf() methods.
onBind()	This method is mandatory to implement in android service and is invoked whenever an application component calls the bindService() method in order to bind itself with a service. User-interface is also provided to communicate with the service effectively by returning an IBinder object. If the binding of service is not required then the method must return null.
onUnbind()	The Android system invokes this method when all the clients get disconnected from a particular service interface.
onRebind()	Once all clients are disconnected from the particular interface of service and there is a need to connect the service with new clients, the system calls this method.
onCreate()	Whenever a service is created either using onStartCommand() or onBind(), the android system calls this method. This method is necessary to perform a one-time-set-up.
onDestroy()	When a service is no longer in use, the system invokes this method just before the service destroys as a final clean up call. Services must implement this method in order to clean up resources like registered listeners, threads, receivers, etc.

Example of Android Services

Playing music in the background is a very common example of services in android. From the time when a user starts the service, music play continuously in the background even if the user switches to another application. The user has to stop the service explicitly in order to pause the music. Below is the complete step-by-step implementation of this android service using a few callback methods.

Note: Following steps are performed on Android Studio version 4.0

Step 1: Create a new project

1. Click on File, then New => New Project.
2. Choose Empty Views Activity
3. Select language as Java/Kotlin
4. Select the minimum SDK as per your need.

Step 2: Modify strings.xml file

All the strings which are used in the activity are listed in this file.

XML

```
<resources>
    <string name="app_name">Services_In_Android</string>
    <string name="heading">Services In Android</string>
    <string name="startButtonText">Start the Service</string>
    <string name="stopButtonText">Stop the Service</string>
</resources>
```

Step 3: Working with the activity_main.xml file

Open the activity_main.xml file and add 2 Buttons in it which will start and stop the service. Below is the code for designing a proper activity layout.

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"
    android:background="#168BC34A"
    tools:context=".MainActivity">

    <LinearLayout
        android:id="@+id/linearLayout"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_centerVertical="true"
        android:orientation="vertical"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintEnd_toEndOf="parent"
```

```
        app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent"
        app:layout_constraintVertical_bias="1.0"
        tools:ignore="MissingConstraints">

        <TextView
            android:id="@+id/textView1"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginBottom="170dp"
            android:fontFamily="@font/roboto"
            android:text="@string/heading"
            android:textAlignment="center"
            android:textAppearance="@style/TextAppearance.AppCompat.Large"
            android:textColor="@android:color/holo_green_dark"
            android:textSize="36sp"
            android:textStyle="bold" />

        <Button
            android:id="@+id/startButton"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginStart="20dp"
            android:layout_marginTop="10dp"
            android:layout_marginEnd="20dp"
            android:layout_marginBottom="20dp"
            android:background="#4CAF50"
            android:fontFamily="@font/roboto"
            android:text="@string/startButtonText"
            android:textAlignment="center"
            android:textAppearance="@style/TextAppearance.AppCompat.Display1"
            android:textColor="#FFFFFF"
            android:textStyle="bold" />

        <Button
            android:id="@+id/stopButton"
            android:layout_width="match_parent"
            android:layout_height="match_parent"
            android:layout_marginStart="20dp"
            android:layout_marginTop="10dp"
            android:layout_marginEnd="20dp"
            android:layout_marginBottom="20dp"
            android:background="#4CAF50"
            android:fontFamily="@font/roboto"
            android:text="@string/stopButtonText"
            android:textAlignment="center"
            android:textAppearance="@style/TextAppearance.AppCompat.Display1"
            android:textColor="#FFFFFF"
            android:textStyle="bold" />
```

```
        <ImageView
            android:id="@+id/imageView"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_marginTop="80dp"
            app:srcCompat="@drawable/banner" />
    </LinearLayout>

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

Step 4: Creating the custom service class

A custom service class will be created in the same directory where the MainActivity class resides and this class will extend the Service class. The callback methods are used to initiate and destroy the services. To play music, the MediaPlayer object is used. Below is the code to carry out this task.

JavaKotlin

```
import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;
import android.provider.Settings;
import androidx.annotation.Nullable;

public class NewService extends Service {

    // declaring object of MediaPlayer
    private MediaPlayer player;

    @Override

    // execution of service will start
    // on calling this method
    public int onStartCommand(Intent intent, int flags, int startId) {

        // creating a media player which
        // will play the audio of Default
        // ringtone in android device
        player = MediaPlayer.create( this, Settings.System.DEFAULT_RINGTONE_URI );

        // providing the boolean
        // value as true to play
        // the audio on loop
        player.setLooping( true );

        // starting the process
        player.start();
```

```
        // returns the status
        // of the program
        return START_STICKY;
    }

    @Override

    // execution of the service will
    // stop on calling this method
    public void onDestroy() {
        super.onDestroy();

        // stopping the process
        player.stop();
    }

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
    }
}
```

Step 5: Working with the MainActivity file

Now, the button objects will be declared and the process to be performed on clicking these buttons will be defined in the MainActivity class. Below is the code to implement this step.

JavaKotlin

```
import androidx.appcompat.app.AppCompatActivity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity implements View.OnClickListener
{

    // declaring objects of Button class
    private Button start, stop;

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

        // assigning ID of startButton
        // to the object start
```

```
        start = (Button) findViewById( R.id.startButton );

        // assigning ID of stopButton
        // to the object stop
        stop = (Button) findViewById( R.id.stopButton );

        // declaring listeners for the
        // buttons to make them respond
        // correctly according to the process
        start.setOnClickListener( this );
        stop.setOnClickListener( this );
    }

    public void onClick(View view) {

        // process to be performed
        // if start button is clicked
        if(view == start){

            // starting the service
            startService(new Intent( this, NewService.class ) );
        }

        // process to be performed
        // if stop button is clicked
        else if (view == stop){

            // stopping the service
            stopService(new Intent( this, NewService.class ) );

        }
    }
}
```

Step 6: Modify the AndroidManifest.xml file

To implement the services successfully on any android device, it is necessary to mention the created service in the AndroidManifest.xml file. It is not possible for a service to perform its task if it is not mentioned in this file. The service name is mentioned inside the application tag.

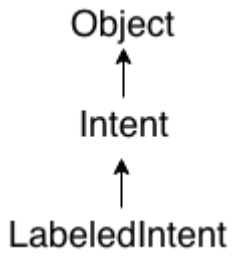
XML

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

    <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>
        <meta-data
            android:name="preloaded_fonts"
            android:resource="@array/preloaded_fonts" />

        <!-- Mention the service name here -->
        <service android:name=".NewService"/>

    </application>

</manifest>
```

Output: Run on Emulator

Android Intent



Android Intent is the *message* that is passed between components such as activities, content providers, broadcast receivers, services etc.

It is generally used with startActivity() method to invoke activity, broadcast receivers etc.

The **dictionary meaning** of intent is *intention or purpose*. So, it can be described as the intention to do action.

The LabeledIntent is the subclass of android.content.Intent class.

Android intents are mainly used to:

ADVERTISEMENT

- Start the service
- Launch an activity
- Display a web page
- Display a list of contacts
- Broadcast a message
- Dial a phone call etc.

Types of Android Intents

There are two types of intents in android: implicit and explicit.

1) Implicit Intent

Implicit Intent doesn't specifiy the component. In such case, intent provides information of available components provided by the system that is to be invoked.

For example, you may write the following code to view the webpage.

```
1.  Intent intent=new Intent(Intent.ACTION_VIEW);
2.  intent.setData(Uri.parse
3.  startActivity(intent);
```

2) Explicit Intent

Explicit Intent specifies the component. In such case, intent provides the external class to be invoked.

```
1.  Intent i = new Intent(getApplicationContext(), ActivityTwo.class);
2.  startActivity(i);
```

To get the full code of explicit intent, visit the next page.

ADVERTISEMENT

Android Implicit Intent Example

Let's see the simple example of implicit intent that displays a web page.

activity_main.xml

File: activity_main.xml

```
1.  <?xml version="1.0" encoding="utf-8"?>
2.  <android.support.constraint.ConstraintLayout xmlns:android="http://schemas.androi
    d.com/apk/res/android"
3.      xmlns:app="http://schemas.android.com/apk/res-auto"
4.      xmlns:tools="http://schemas.android.com/tools"
5.      android:layout_width="match_parent"
6.      android:layout_height="match_parent"
7.      tools:context="example.javatpoint.com.implicitintent.MainActivity">
8.
9.      <EditText
10.         android:id="@+id/editText"
11.         android:layout_width="wrap_content"
12.         android:layout_height="wrap_content"
13.         android:layout_marginEnd="8dp"
14.         android:layout_marginStart="8dp"
15.         android:layout_marginTop="60dp"
16.         android:ems="10"
17.         app:layout_constraintEnd_toEndOf="parent"
18.         app:layout_constraintHorizontal_bias="0.575"
19.         app:layout_constraintStart_toStartOf="parent"
20.         app:layout_constraintTop_toTopOf="parent" />
21.
22.     <Button
23.         android:id="@+id/button"
24.         android:layout_width="wrap_content"
25.         android:layout_height="wrap_content"
26.         android:layout_marginRight="8dp"
27.         android:layout_marginLeft="156dp"
28.         android:layout_marginTop="172dp"
```

```
29.             android:text="Visit"
30.             app:layout_constraintEnd_toEndOf="parent"
31.             app:layout_constraintHorizontal_bias="0.0"
32.             app:layout_constraintStart_toStartOf="parent"
33.             app:layout_constraintTop_toBottomOf="@+id/editText" />
34.         </android.support.constraint.ConstraintLayout>
```

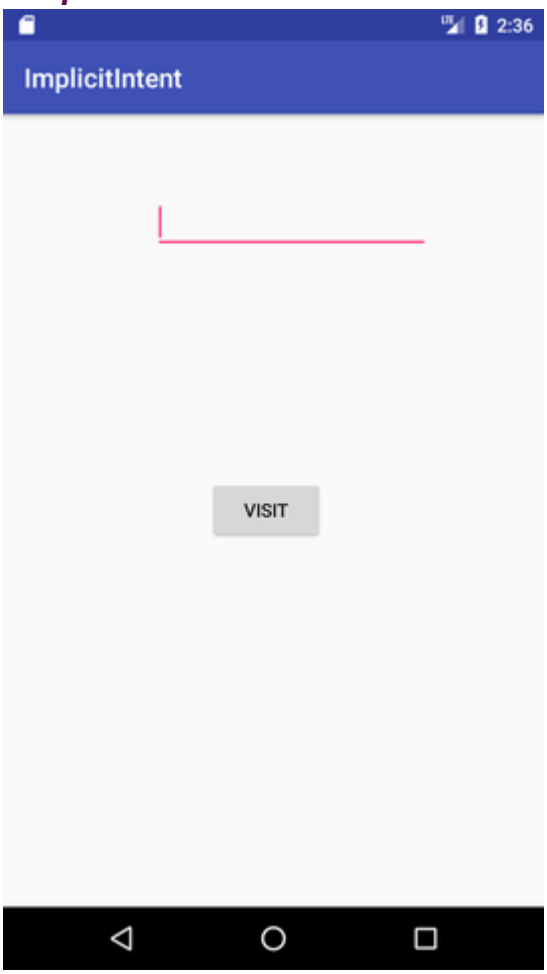
Activity class

File: MainActivity.java

```
1.  package example.javatpoint.com.implicitintent;
2.
3.  import android.content.Intent;
4.  import android.net.Uri;
5.  import android.support.v7.app.AppCompatActivity;
6.  import android.os.Bundle;
7.  import android.view.View;
8.  import android.widget.Button;
9.  import android.widget.EditText;
10.
11. public class MainActivity extends AppCompatActivity {
12.
13.     Button button;
14.     EditText editText;
15.
16.     @Override
17.     protected void onCreate(Bundle savedInstanceState) {
18.         super.onCreate(savedInstanceState);
19.         setContentView(R.layout.activity_main);
20.
21.         button = findViewById(R.id.button);
22.         editText =  findViewById(R.id.editText);
23.
24.         button.setOnClickListener(new View.OnClickListener() {
25.             @Override
```

```
26.                 public void onClick(View view) {
27.                     String url=editText.getText().toString();
28.                     Intent intent=new Intent(Intent.ACTION_VIEW, Uri.parse(url));
29.                     startActivity(intent);
30.                 }
31.             });
32.         }
33.     }
```

Output:



Broadcast Receiver in Android With Example

Broadcast in android is the system-wide events that can occur when the device starts, when a message is received on the device or when incoming calls are received, or when a device goes to airplane mode, etc. Broadcast Receivers are used to respond to these system-wide events. Broadcast Receivers allow us to register for the system and application events, and when that event happens, then the register receivers get notified. There are mainly two types of Broadcast Receivers:

- Static Broadcast Receivers: These types of Receivers are declared in the manifest file and works even if the app is closed.

- Dynamic Broadcast Receivers: These types of receivers work only if the app is active or minimized.

Since from API Level 26, most of the broadcast can only be caught by the dynamic receiver, so we have implemented dynamic receivers in our sample project given below. There are some static fields defined in the Intent class which can be used to broadcast different events. We have taken a change of airplane mode as a broadcast event, but there are many events for which broadcast register can be used. Following are some of the important system-wide generated intents:-

Activity_main.xml

File: activity_main.xml

```
<RelativeLayout xmlns:androclass="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:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context=".MainActivity" >

   <TextView
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/hello_world" />

</RelativeLayout>
```

Activity class

File: MainActivity.java

```
package com.example.callstatebroadcastreceiver;

import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;

```

```
7.  public class MainActivity extends Activity {
8.
9.      @Override
10.     protected void onCreate(Bundle savedInstanceState) {
11.         super.onCreate(savedInstanceState);
12.         setContentView(R.layout.activity_main);
13.     }
14.
15.
16.     @Override
17.     public boolean onCreateOptionsMenu(Menu menu) {
18.         // Inflate the menu; this adds items to the action bar if it is present.
19.         getMenuInflater().inflate(R.menu.main, menu);
20.         return true;
21.     }
22.
23. }
```

IncommingCallReceiver

File: IncommingCallReceiver.java

```
1.  package com.example.callstatebroadcastreceiver;
2.
3.  import android.content.BroadcastReceiver;
4.  import android.content.Context;
5.  import android.content.Intent;
6.  import android.telephony.TelephonyManager;
7.  import android.widget.Toast;
8.
9.
10.  public class IncommingCallReceiver extends BroadcastReceiver{
11.      Context context;
12.
13.      @Override
14.      public void onReceive(Context context, Intent intent){
```

```
15.                 try{
16.                  String state = intent.getStringExtra(TelephonyManager.EXTRA_STATE);
17.
18.                    if(state.equals(TelephonyManager.EXTRA_STATE_RINGING)){
19.                        Toast.makeText(context, "Phone Is Ringing", Toast.LENGTH_LONG).show();
20.                    }
21.
22.                    if(state.equals(TelephonyManager.EXTRA_STATE_OFFHOOK)){
23.                        Toast.makeText(context, "Call Recieved", Toast.LENGTH_LONG).show();
24.                    }
25.
26.                    if (state.equals(TelephonyManager.EXTRA_STATE_IDLE)){
27.                        Toast.makeText(context, "Phone Is Idle", Toast.LENGTH_LONG).show();
28.                    }
29.                }
30.                catch(Exception e){e.printStackTrace();}
31.            }
32.
33.        }
```

AndroidManifest.xml file in android

The AndroidManifest.xml file *contains information of your package*, including components of the application such as activities, services, broadcast receivers, content providers etc.

It performs some other tasks also:

- It is **responsible to protect the application** to access any protected parts by providing the permissions.
- It also **declares the android api** that the application is going to use.

- It **lists the instrumentation classes**. The instrumentation classes provides profiling and other informations. These informations are removed just before the application is published etc.

This is the required xml file for all the android application and located inside the root directory.

A simple AndroidManifest.xml file looks like this:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.javatpoint.hello"
  android:versionCode="1"
  android:versionName="1.0" >

  <uses-sdk
    android:minSdkVersion="8"
    android:targetSdkVersion="15" />

  <application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <activity
      android:name=".MainActivity"
      android:label="@string/title_activity_main" >
      <intent-filter>
        <action android:name="android.intent.action.MAIN" />

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

AndroidManifest.xml file

The elements used in the above xml file are described below.

<manifest>

manifest is the root element of the AndroidManifest.xml file. It has **package** attribute that describes the package name of the activity class.

<application>

application is the subelement of the manifest. It includes the namespace declaration. This element contains several subelements that declares the application component such as activity etc.

The commonly used attributes are of this element are **icon**, **label**, **theme** etc.

android:icon represents the icon for all the android application components.

android:label works as the default label for all the application components.

android:theme represents a common theme for all the android activities.

<activity>

activity is the subelement of application and represents an activity that must be defined in the AndroidManifest.xml file. It has many attributes such as label, name, theme, launchMode etc.

android:label represents a label i.e. displayed on the screen.

android:name represents a name for the activity class. It is required attribute.

<intent-filter>

intent-filter is the sub-element of activity that describes the type of intent to which activity, service or broadcast receiver can respond to.

<action>

It adds an action for the intent-filter. The intent-filter must have at least one action element.

<category>

It adds a category name to an intent-filter.

Android - Intents and Filters

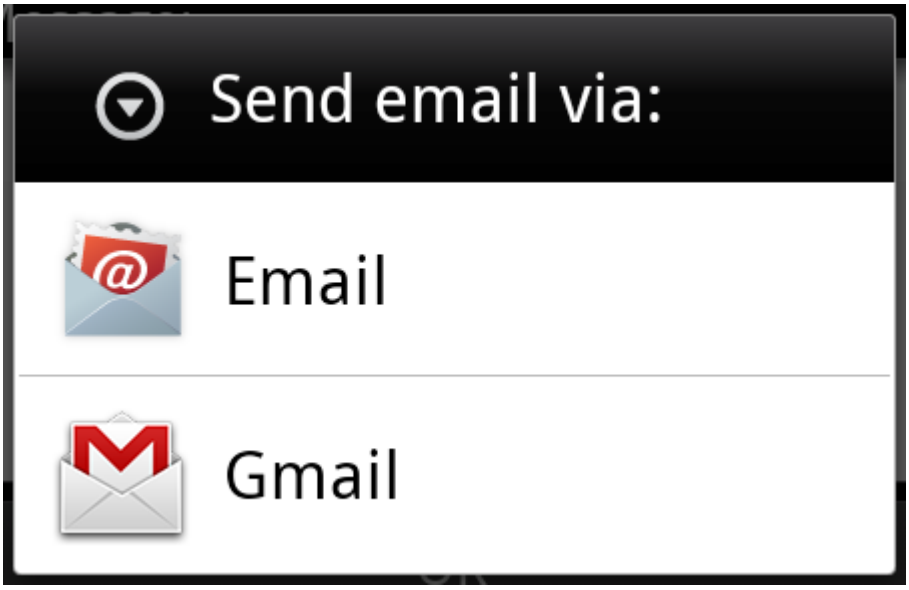
An Android **Intent** is an abstract description of an operation to be performed. It can be used with **startActivity** to launch an Activity, **broadcastIntent** to send it to any interested BroadcastReceiver components, and **startService(Intent)** or **bindService(Intent, ServiceConnection, int)** to communicate with a background Service.

The intent itself, an Intent object, is a passive data structure holding an abstract description of an operation to be performed.

For example, let's assume that you have an Activity that needs to launch an email client and sends an email using your Android device. For this purpose, your Activity would send an ACTION_SEND along with appropriate **chooser**, to the Android Intent Resolver. The specified chooser gives the proper interface for the user to pick how to send your email data.

```
Intent email = new Intent(Intent.ACTION_SEND, Uri.parse("mailto:"));
email.putExtra(Intent.EXTRA_EMAIL, recipients);
email.putExtra(Intent.EXTRA_SUBJECT, subject.getText().toString());
email.putExtra(Intent.EXTRA_TEXT, body.getText().toString());
startActivity(Intent.createChooser(email, "Choose an email client
from..."));
```

Above syntax is calling startActivity method to start an email activity and result should be as shown below −



For example, assume that you have an Activity that needs to open URL in a web browser on your Android device. For this purpose, your Activity will send ACTION_WEB_SEARCH Intent to the Android Intent Resolver to open given URL

in the web browser. The Intent Resolver parses through a list of Activities and chooses the one that would best match your Intent, in this case, the Web Browser Activity. The Intent Resolver then passes your web page to the web browser and starts the Web Browser Activity.

```
String q = "tutorialspoint";
Intent intent = new Intent(Intent.ACTION_WEB_SEARCH );
intent.putExtra(SearchManager.QUERY, q);
startActivity(intent);
```

Above example will search as **tutorialspoint** on android search engine and it gives the result of tutorialspoint in your an activity

There are separate mechanisms for delivering intents to each type of component – activities, services, and broadcast receivers.

1. **Context.startActivity()**
   The Intent object is passed to this method to launch a new activity or get an existing activity to do something new.

2. **Context.startService()**
   The Intent object is passed to this method to initiate a service or deliver new instructions to an ongoing service.

3. **Context.sendBroadcast()**
   The Intent object is passed to this method to deliver the message to all interested broadcast receivers.

Intent Objects

An Intent object is a bundle of information which is used by the component that receives the intent as well as information used by the Android system.

An Intent object can contain the following components based on what it is communicating or going to perform –

Action

This is mandatory part of the Intent object and is a string naming the action to be performed — or, in the case of broadcast intents, the action that took place and is being reported. The action largely determines how the rest of the intent object is structured . The Intent class defines a number of action constants corresponding to different intents. Here is a list of
Android Intent Standard Actions

The action in an Intent object can be set by the setAction() method and read by getAction().

Data

Adds a data specification to an intent filter. The specification can be just a data type (the mimeType attribute), just a URI, or both a data type and a URI. A URI is specified by separate attributes for each of its parts –

These attributes that specify the URL format are optional, but also mutually dependent –

- If a scheme is not specified for the intent filter, all the other URI attributes are ignored.
- If a host is not specified for the filter, the port attribute and all the path attributes are ignored.

The setData() method specifies data only as a URI, setType() specifies it only as a MIME type, and setDataAndType() specifies it as both a URI and a MIME type. The URI is read by getData() and the type by getType().

Some examples of action/data pairs are –

1. **ACTION_VIEW content://contacts/people/1**
   Display information about the person whose identifier is "1".

2. **ACTION_DIAL content://contacts/people/1**
   Display the phone dialer with the person filled in.

3. **ACTION_VIEW tel:123**
   Display the phone dialer with the given number filled in.

4. **ACTION_DIAL tel:123**
   Display the phone dialer with the given number filled in.

5. **ACTION_EDIT content://contacts/people/1**
   Edit information about the person whose identifier is "1".

6. **ACTION_VIEW content://contacts/people/**
   Display a list of people, which the user can browse through.

7. **ACTION_SET_WALLPAPER**
   Show settings for choosing wallpaper

8. **ACTION_SYNC**
   It going to be synchronous the data,Constant Value is **android.intent.action.SYNC**

9. **ACTION_SYSTEM_TUTORIAL**
   It will start the platform-defined tutorial(Default tutorial or start up tutorial)

10 **ACTION_TIMEZONE_CHANGED**
It intimates when time zone has changed

11 **ACTION_UNINSTALL_PACKAGE**
It is used to run default uninstaller

Category

The category is an optional part of Intent object and it's a string containing additional information about the kind of component that should handle the intent. The addCategory() method places a category in an Intent object, removeCategory() deletes a category previously added, and getCategories() gets the set of all categories currently in the object. Here is a list of Android Intent Standard Categories.

You can check detail on Intent Filters in below section to understand how do we use categories to choose appropriate activity corresponding to an Intent.

Extras

This will be in key-value pairs for additional information that should be delivered to the component handling the intent. The extras can be set and read using the putExtras() and getExtras() methods respectively. Here is a list of Android Intent Standard Extra Data

Flags

These flags are optional part of Intent object and instruct the Android system how to launch an activity, and how to treat it after it's launched etc.

FLAG_ACTIVITY_CLEAR_TASK

1 If set in an Intent passed to Context.startActivity(), this flag will cause any existing task that would be associated with the activity to be cleared before the activity is started. That is, the activity becomes the new root of an otherwise empty task, and any old activities are finished. This can only be used in conjunction with FLAG_ACTIVITY_NEW_TASK.

FLAG_ACTIVITY_CLEAR_TOP

2 If set, and the activity being launched is already running in the current task, then instead of launching a new instance of that activity, all of the other activities on top of it will be closed and this Intent will be delivered to the (now on top) old activity as a new Intent.

FLAG_ACTIVITY_NEW_TASK

3 This flag is generally used by activities that want to present a "launcher" style behavior: they give the user a list of separate things that can be done, which otherwise run completely independently of the activity launching them.

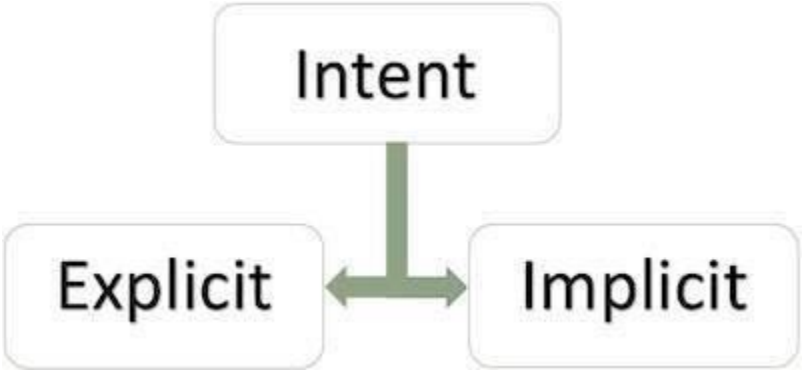
Component Name

This optional field is an android **ComponentName** object representing either Activity, Service or BroadcastReceiver class. If it is set, the Intent object is delivered to an instance of the designated class otherwise Android uses other information in the Intent object to locate a suitable target.

The component name is set by setComponent(), setClass(), or setClassName() and read by getComponent().

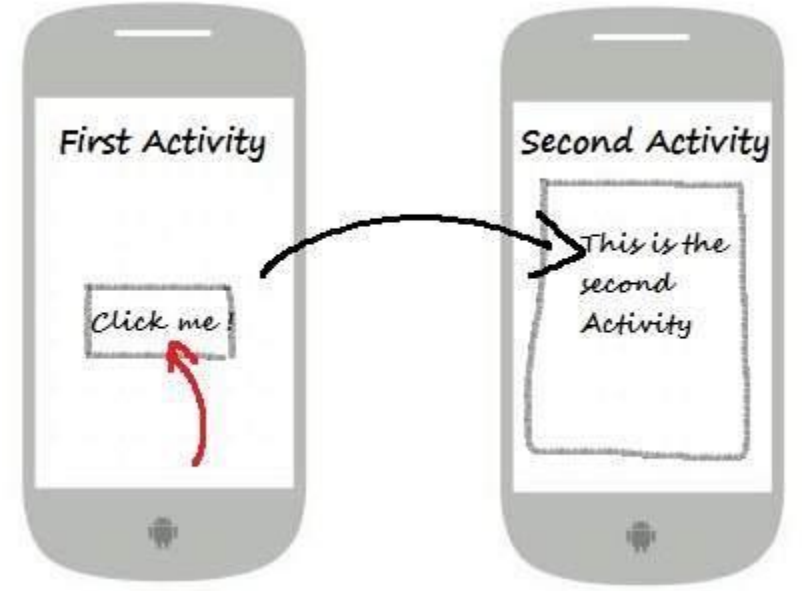
Types of Intents

There are following two types of intents supported by Android



Explicit Intents

Explicit intent going to be connected internal world of application,suppose if you wants to connect one activity to another activity, we can do this quote by explicit intent, below image is connecting first activity to second activity by clicking button.



These intents designate the target component by its name and they are typically used for application-internal messages - such as an activity starting a subordinate service or launching a sister activity. For example −

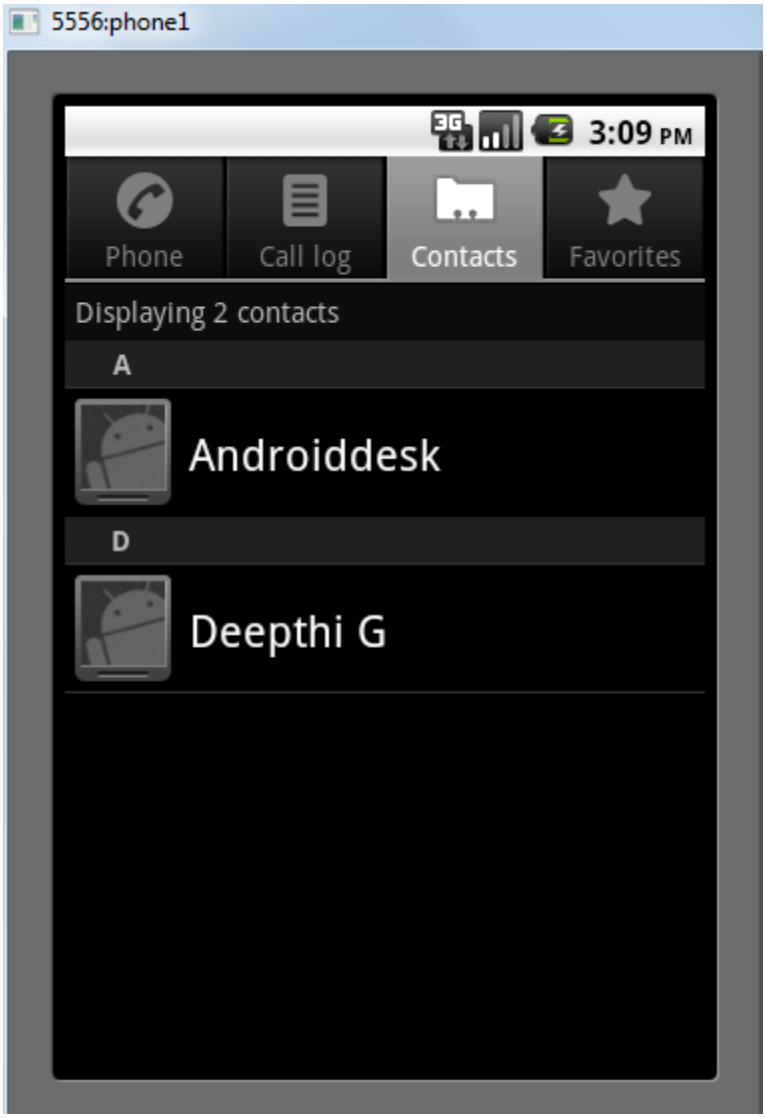
```
// Explicit Intent by specifying its class name
Intent i = new Intent(FirstActivity.this, SecondActivity.class);

// Starts TargetActivity
startActivity(i);
```

Implicit Intents

These intents do not name a target and the field for the component name is left blank. Implicit intents are often used to activate components in other applications. For example −

```
Intent read1=new Intent();
read1.setAction(android.content.Intent.ACTION_VIEW);
read1.setData(ContactsContract.Contacts.CONTENT_URI);
startActivity(read1);
```

Above code will give result as shown below



The target component which receives the intent can use the **getExtras()** method to get the extra data sent by the source component. For example –

```
// Get bundle object at appropriate place in your code
Bundle extras = getIntent().getExtras();

// Extract data using passed keys
String value1 = extras.getString("Key1");
String value2 = extras.getString("Key2");
```

Example

Following example shows the functionality of a Android Intent to launch various Android built-in applications.

Step	Description
1	You will use Android studio IDE to create an Android application and name it as *My Application* under a package *com.example.saira_000.myapplication*.
2	Modify *src/main/java/MainActivity.java* file and add the code to define two listeners corresponding two buttons ie. Start Browser and Start Phone.
3	Modify layout XML file *res/layout/activity_main.xml* to add three buttons in linear layout.
4	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/com.example.My Application/MainActivity.java**.

```
package com.example.saira_000.myapplication;

import android.content.Intent;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {
   Button b1,b2;

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

      b1=(Button)findViewById(R.id.button);
      b1.setOnClickListener(new View.OnClickListener() {

         @Override
         public void onClick(View v) {
            Intent i = new Intent(android.content.Intent.ACTION_VIEW,
               Uri.parse("http://www.example.com"));
            startActivity(i);
         }
```

```
        });

        b2=(Button)findViewById(R.id.button2);
        b2.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                Intent i = new Intent(android.content.Intent.ACTION_VIEW,
                    Uri.parse("tel:9510300000"));
                startActivity(i);
            }
        });
    }
}
```

Following will be the content of **res/layout/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"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   android:paddingBottom="@dimen/activity_vertical_margin"
   tools:context=".MainActivity">

   <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Intent Example"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:textSize="30dp" />

   <TextView
      android:id="@+id/textView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Tutorials point"
      android:textColor="#ff87ff09"
      android:textSize="30dp"
      android:layout_below="@+id/textView1"
      android:layout_centerHorizontal="true" />

   <ImageButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/imageButton"
```

```
      android:src="@drawable/abc"
      android:layout_below="@+id/textView2"
      android:layout_centerHorizontal="true" />

   <EditText
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/editText"
      android:layout_below="@+id/imageButton"
      android:layout_alignRight="@+id/imageButton"
      android:layout_alignEnd="@+id/imageButton" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Start Browser"
      android:id="@+id/button"
      android:layout_alignTop="@+id/editText"
      android:layout_alignRight="@+id/textView1"
      android:layout_alignEnd="@+id/textView1"
      android:layout_alignLeft="@+id/imageButton"
      android:layout_alignStart="@+id/imageButton" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Start Phone"
      android:id="@+id/button2"
      android:layout_below="@+id/button"
      android:layout_alignLeft="@+id/button"
      android:layout_alignStart="@+id/button"
      android:layout_alignRight="@+id/textView2"
      android:layout_alignEnd="@+id/textView2" />
</RelativeLayout>
```

Following will be the content of **res/values/strings.xml** to define two new constants −

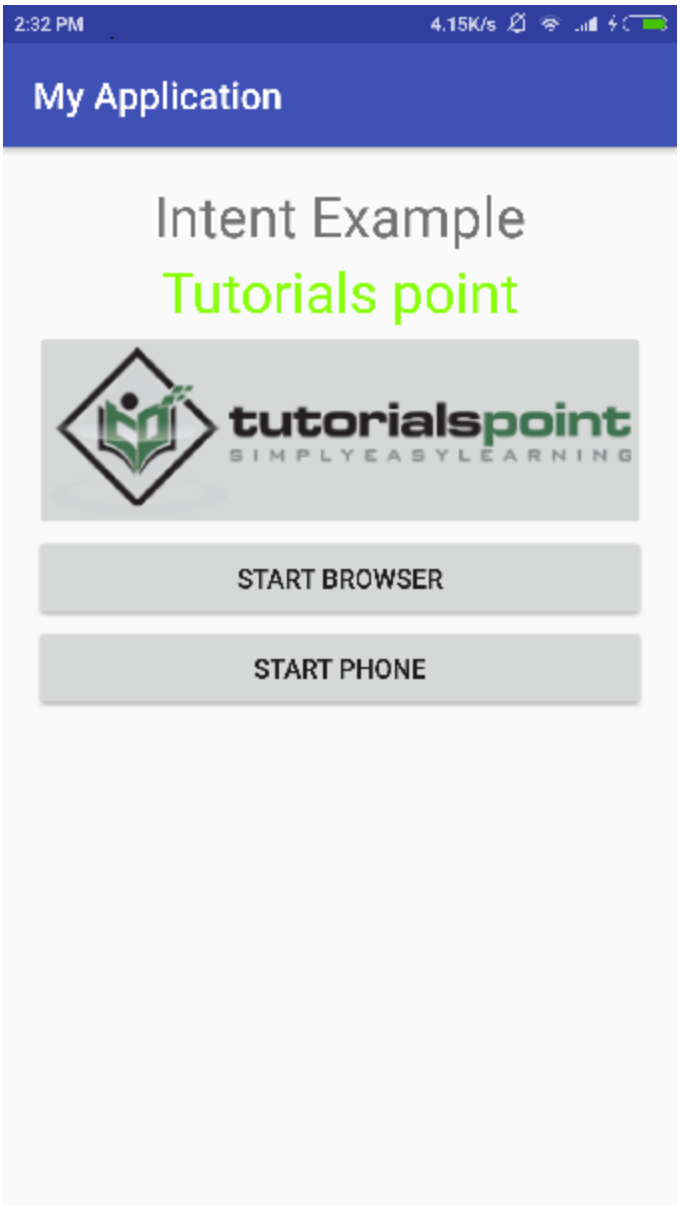
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">My Applicaiton</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

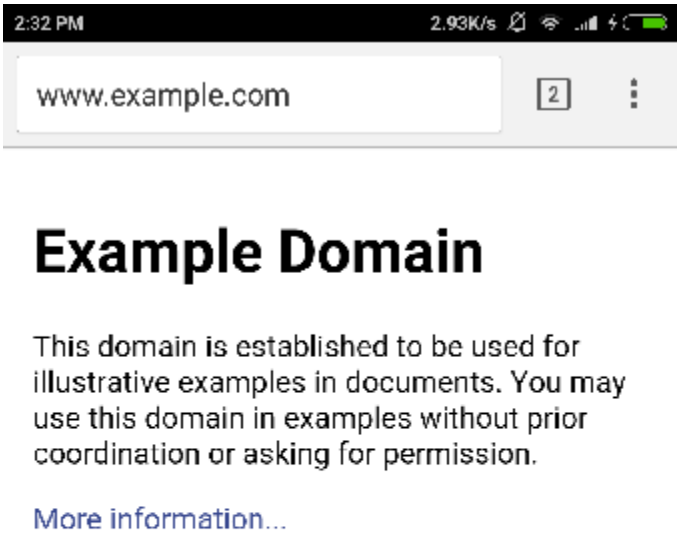
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.saira_000.myapplication">

   <application
      android:allowBackup="true"
      android:icon="@mipmap/ic_launcher"
```

```
      android:label="@string/app_name"
      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>
```

Let's try to run your **My Application** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar.Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window –



Now click on **Start Browser** button, which will start a browser configured and display http://www.example.com as shown below –



Similar way you can launch phone interface using Start Phone button, which will allow you to dial already given phone number.

Intent Filters

You have seen how an Intent has been used to call an another activity. Android OS uses filters to pinpoint the set of Activities, Services, and Broadcast receivers that can handle the Intent with help of specified set of action, categories, data scheme associated with an Intent. You will use **<intent-filter>** element in the manifest file to list down actions, categories and data types associated with any activity, service, or broadcast receiver.

Following is an example of a part of **AndroidManifest.xml** file to specify an activity **com.example.My Application.CustomActivity** which can be invoked by either of the two mentioned actions, one category, and one data –

```
<activity android:name=".CustomActivity"
   android:label="@string/app_name">

   <intent-filter>
      <action android:name="android.intent.action.VIEW" />
      <action android:name="com.example.My Application.LAUNCH" />
      <category android:name="android.intent.category.DEFAULT" />
      <data android:scheme="http" />
   </intent-filter>
```

```
</activity>
```

Once this activity is defined along with above mentioned filters, other activities will be able to invoke this activity using either the **android.intent.action.VIEW**, or using the **com.example.My Application.LAUNCH** action provided their category is **android.intent.category.DEFAULT**.

The <**data**> element specifies the data type expected by the activity to be called and for above example our custom activity expects the data to start with the "http://"

There may be a situation that an intent can pass through the filters of more than one activity or service, the user may be asked which component to activate. An exception is raised if no target can be found.

There are following test Android checks before invoking an activity –

- A filter <intent-filter> may list more than one action as shown above but this list cannot be empty; a filter must contain at least one <action> element, otherwise it will block all intents. If more than one actions are mentioned then Android tries to match one of the mentioned actions before invoking the activity.
- A filter <intent-filter> may list zero, one or more than one categories. if there is no category mentioned then Android always pass this test but if more than one categories are mentioned then for an intent to pass the category test, every category in the Intent object must match a category in the filter.
- Each <data> element can specify a URI and a data type (MIME media type). There are separate attributes like **scheme, host, port**, and **path** for each part of the URI. An Intent object that contains both a URI and a data type passes the data type part of the test only if its type matches a type listed in the filter.

Example

Following example is a modification of the above example. Here we will see how Android resolves conflict if one intent is invoking two activities defined in , next how to invoke a custom activity using a filter and third one is an exception case if Android does not file appropriate activity defined for an intent.

Step	Description
1	You will use android studio to create an Android application and name it as *My Application* under a package *com.example.tutorialspoint7.myapplication;*.
2	Modify *src/Main/Java/MainActivity.java* file and add the code to define three listeners corresponding to three buttons defined in layout file.

3	Add a new *src/Main/Java/CustomActivity.java* file to have one custom activity which will be invoked by different intents.
4	Modify layout XML file *res/layout/activity_main.xml* to add three buttons in linear layout.
5	Add one layout XML file *res/layout/custom_view.xml* to add a simple <TextView> to show the passed data through intent.
6	Modify *AndroidManifest.xml* to add <intent-filter> to define rules for your intent to invoke custom activity.
7	Run the application to launch Android emulator and verify the result of the changes done in the application.

Following is the content of the modified main activity file **src/MainActivity.java.**

```
package com.example.tutorialspoint7.myapplication;

import android.content.Intent;
import android.net.Uri;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class MainActivity extends AppCompatActivity {
   Button b1,b2,b3;
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      b1=(Button)findViewById(R.id.button);
      b1.setOnClickListener(new View.OnClickListener() {

         @Override
         public void onClick(View v) {
            Intent i = new Intent(android.content.Intent.ACTION_VIEW,
               Uri.parse("http://www.example.com"));
            startActivity(i);
         }
      });

      b2 = (Button)findViewById(R.id.button2);
      b2.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
```

```
            Intent i = new Intent("com.example.
               tutorialspoint7.myapplication.
                  LAUNCH",Uri.parse("http://www.example.com"));
            startActivity(i);
         }
      });

      b3 = (Button)findViewById(R.id.button3);
      b3.setOnClickListener(new View.OnClickListener() {
         @Override
         public void onClick(View v) {
            Intent i = new Intent("com.example.
               My Application.LAUNCH",
                  Uri.parse("https://www.example.com"));
            startActivity(i);
         }
      });
   }
}
```

Following is the content of the modified main activity file **src/com.example.My Application/CustomActivity.java.**

```
package com.example.tutorialspoint7.myapplication;

import android.app.Activity;
import android.net.Uri;
import android.os.Bundle;
import android.widget.TextView;

/**
 * Created by TutorialsPoint7 on 8/23/2016.
 */
public class CustomActivity extends Activity {
   @Override
   public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.custom_view);
      TextView label = (TextView) findViewById(R.id.show_data);
      Uri url = getIntent().getData();
      label.setText(url.toString());
   }
}
```

Following will be the content of **res/layout/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"
```

```
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"

tools:context="com.example.tutorialspoint7.myapplication.MainActivi
ty">

   <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Intent Example"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"
      android:textSize="30dp" />

   <TextView
      android:id="@+id/textView2"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Tutorials point"
      android:textColor="#ff87ff09"
      android:textSize="30dp"
      android:layout_below="@+id/textView1"
      android:layout_centerHorizontal="true" />

   <ImageButton
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/imageButton"
      android:src="@drawable/abc"
      android:layout_below="@+id/textView2"
      android:layout_centerHorizontal="true" />

   <EditText
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:id="@+id/editText"
      android:layout_below="@+id/imageButton"
      android:layout_alignRight="@+id/imageButton"
      android:layout_alignEnd="@+id/imageButton" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Start Browser"
      android:id="@+id/button"
      android:layout_alignTop="@+id/editText"
```

```
      android:layout_alignLeft="@+id/imageButton"
      android:layout_alignStart="@+id/imageButton"
      android:layout_alignEnd="@+id/imageButton" />

   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Start browsing with launch action"
      android:id="@+id/button2"
      android:layout_below="@+id/button"
      android:layout_alignLeft="@+id/button"
      android:layout_alignStart="@+id/button"
      android:layout_alignEnd="@+id/button" />
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Exceptional condition"
      android:id="@+id/button3"
      android:layout_below="@+id/button2"
      android:layout_alignLeft="@+id/button2"
      android:layout_alignStart="@+id/button2"
      android:layout_toStartOf="@+id/editText"
      android:layout_alignParentEnd="true" />
</RelativeLayout>
```

Following will be the content of **res/layout/custom_view.xml** file −

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
xmlns:android="http://schemas.android.com/apk/res/android"
   android:orientation="vertical"
android:layout_width="match_parent"
   android:layout_height="match_parent">
   <TextView android:id="@+id/show_data"
      android:layout_width="fill_parent"
      android:layout_height="400dp"/>
</LinearLayout>
```

Following will be the content of **res/values/strings.xml** to define two new constants −

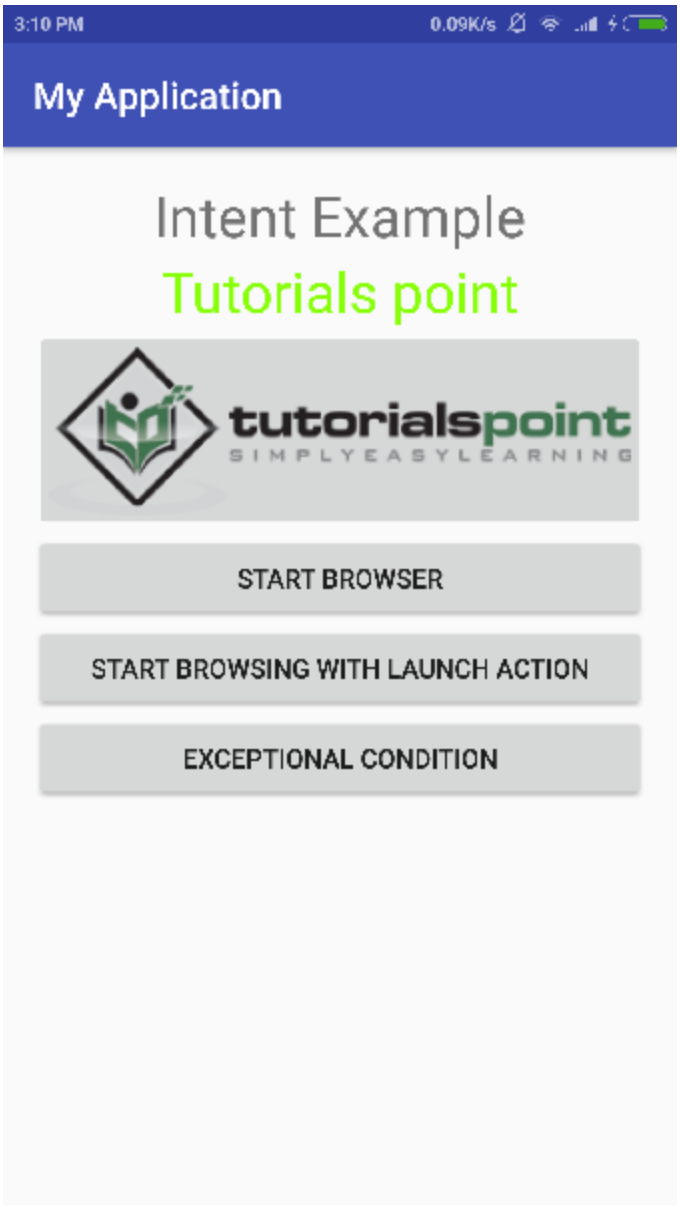
```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <string name="app_name">My Application</string>
</resources>
```

Following is the default content of **AndroidManifest.xml** −

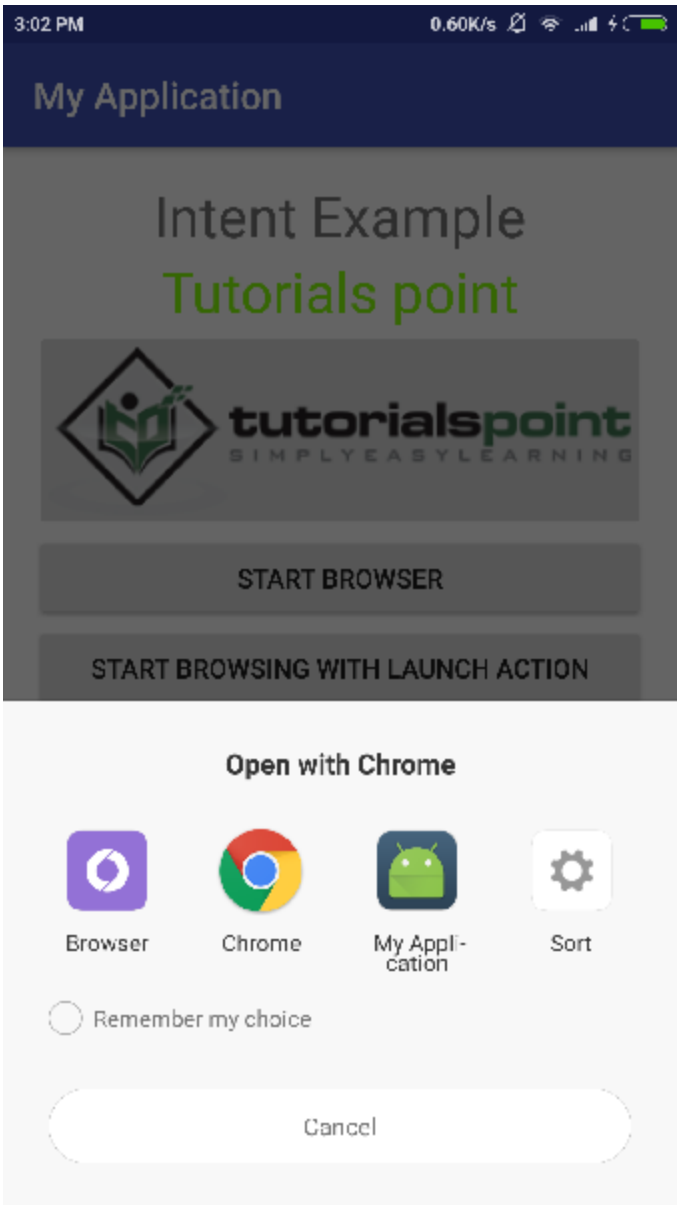
```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
   package="com.example.tutorialspoint7.myapplication">

   <application
```

```
      android:allowBackup = "true"
      android:icon = "@mipmap/ic_launcher"
      android:label = "@string/app_name"
      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>

      <activity
android:name="com.example.tutorialspoint7.myapplication.CustomActiv
ity">

         <intent-filter>
            <action android:name = "android.intent.action.VIEW" />
            <action android:name =
"com.example.tutorialspoint7.myapplication.LAUNCH" />
            <category android:name =
"android.intent.category.DEFAULT" />
            <data android:scheme = "http" />
         </intent-filter>

      </activity>
   </application>

</manifest>
```

Let's try to run your **My Application** application. I assume you had created your **AVD** while doing environment setup. To run the app from Android Studio, open one of your project's activity files and click Run icon from the toolbar. Android Studio installs the app on your AVD and starts it and if everything is fine with your setup and application, it will display following Emulator window –



Now let's start with first button "Start Browser with VIEW Action". Here we have defined our custom activity with a filter "android.intent.action.VIEW", and there is already one default activity against VIEW action defined by Android which is launching web browser, So android displays following two options to select the activity you want to launch.



Permissions on Android

bookmark_border

- **On this page**
- Workflow for using permissions
- Types of permissions
  - Install-time permissions
  - Runtime permissions
  - Special permissions
  - Permission groups

- Best practices
  - Request a minimal number of permissions

App permissions help support user privacy by protecting access to the following:

- Restricted data, such as system state and users' contact information
- Restricted actions, such as connecting to a paired device and recording audio

This page provides an overview to how Android permissions work, including a high-level workflow for using permissions, descriptions of different types of permissions, and some best practices for using permissions in your app. Other pages explain how to minimize your app's requests for permissions, declare permissions, request runtime permissions, and restrict how other apps can interact with your app's components.

To view a complete list of Android app permissions, visit the permissions API reference page.

To view some sample apps that demonstrate the permissions workflow, visit the Android permissions samples repository on GitHub.

Workflow for using permissions

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.

Figure 1 illustrates the workflow for using app permissions:

Types of permissions

Android categorizes permissions into different types, including install-time permissions, runtime permissions, and special permissions. Each permission's type indicates the scope of restricted data that your app can access, and the scope of restricted actions that your app can perform, when the system grants your app that permission. The protection level for each permission is based on its type and is shown on the permissions API reference page.

Install-time permissions

Figure 2. The list of an app'install-time permissions, which appears in an app store.

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.

Android includes several sub-types of install-time permissions, including normal permissions and signature permissions.

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.

The system assigns the normal protection level to normal permissions.

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.

Note: Some signature permissions aren't for use by third-party apps.

The system assigns the signature protection level to signature permissions.

Runtime permissions 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, as shown in figure 3.

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.

The microphone and camera provide access to particularly sensitive information. Therefore, the system helps you explain why your app accesses this information.

The system assigns the dangerous protection level to runtime permissions.

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.

Learn more about how to request special permissions.

The system assigns the appop protection level to special permissions.

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.

Best practices

App permissions build on system security features and help Android support the following goals related to user privacy:

- Control: The user has control over the data that they share with apps.
- Transparency: The user understands what data an app uses and why the app accesses this data.
- Data minimization: An app accesses and uses only the data that's required for a specific task or action that the user invokes.

This section presents a set of core best practices for using permissions effectively in your app. For more details on how you can work with permissions on Android, visit the app permissions best practices page.

Request a minimal number of permissions

When the user requests a particular action in your app, your app should request only the permissions that it needs to complete that action. Depending on how you are using the permissions, there might be an alternative way to fulfill your app's use case without relying on access to sensitive information.

Associate runtime permissions with specific actions

Request permissions as late into the flow of your app's use cases as possible. For example, if your app lets users send audio messages to others, wait until the user has navigated to the messaging screen and has pressed the Send audio message button. After the user presses the button, your app can then request access to the microphone.

Consider your app's dependencies

When you include a library, you also inherit its permission requirements. Be aware of the permissions that each dependency requires and what those permissions are used for.

Be transparent

When you make a permissions request, be clear about what you're accessing, why, and what functionalities are affected if permissions are denied, so users can make informed decisions.

Android Resources

There are many more items which you use to build a good Android application. Apart from coding for the application, you take care of various other **resources** like static content that your code uses, such as bitmaps, colors, layout definitions, user interface strings, animation instructions, and more. These resources are always maintained separately in various sub-directories under **res/** directory of the project.

This tutorial will explain you how you can organize your application resources, specify alternative resources and access them in your applications.

Organize resource in Android Studio

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

Directory & Resource Type

1. **anim/**
   XML files that define property animations. They are saved in res/anim/ folder and accessed from the **R.anim** class.

2. **color/**
   XML files that define a state list of colors. They are saved in res/color/ and accessed from the **R.color** class.

3. **drawable/**
   Image files like .png, .jpg, .gif or XML files that are compiled into bitmaps, state lists, shapes, animation drawable. They are saved in res/drawable/ and accessed from the **R.drawable** class.

4 **layout/**
XML files that define a user interface layout. They are saved in res/layout/ and accessed from the **R.layout** class.

5 **menu/**
XML files that define application menus, such as an Options Menu, Context Menu, or Sub Menu. They are saved in res/menu/ and accessed from the **R.menu** class.

6 **raw/**
Arbitrary files to save in their raw form. You need to call *Resources.openRawResource()* with the resource ID, which is *R.raw.filename* to open such raw files.

7 **values/**
XML files that contain simple values, such as strings, integers, and colors. For example, here are some filename conventions for resources you can create in this directory −

- arrays.xml for resource arrays, and accessed from the **R.array** class.
- integers.xml for resource integers, and accessed from the **R.integer** class.
- bools.xml for resource boolean, and accessed from the **R.bool** class.
- colors.xml for color values, and accessed from the **R.color** class.
- dimens.xml for dimension values, and accessed from the **R.dimen** class.
- strings.xml for string values, and accessed from the **R.string** class.
- styles.xml for styles, and accessed from the **R.style** class.

8 **xml/**
Arbitrary XML files that can be read at runtime by calling *Resources.getXML()*. You can save various configuration files here which will be used at run time.

Alternative Resources

Your application should provide alternative resources to support specific device configurations. For example, you should include alternative drawable resources (i.e.images) for different screen resolution and alternative string resources for different languages. At runtime, Android detects the current device configuration and loads the appropriate resources for your application.

To specify configuration-specific alternatives for a set of resources, follow the following steps −

- Create a new directory in res/ named in the form **<resources_name>-<config_qualifier>**. Here **resources_name** will be any of the resources mentioned in the above table, like layout, drawable etc. The **qualifier** will specify an individual configuration for which these resources are to be used. You can check official documentation for a complete list of qualifiers for different type of resources.
- Save the respective alternative resources in this new directory. The resource files must be named exactly the same as the default resource files as shown in the below example, but these files will have content specific to the alternative. For example though image file name will be same but for high resolution screen, its resolution will be high.

Below is an example which specifies images for a default screen and alternative images for high resolution screen.

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

Below is another example which specifies layout for a default language and alternative layout for Arabic language.

```
MyProject/
    app/
        manifest/
            AndroidManifest.xml
    java/
        MyActivity.java
        res/
            drawable/
                icon.png
                background.png
            drawable-hdpi/
                icon.png
                background.png
            layout/
                activity_main.xml
```

info.xml

layout-ar/

main.xml

values/

strings.xml

Accessing Resources

During your application development you will need to access defined resources either in your code, or in your layout XML files. Following section explains how to access your resources in both the scenarios −

Accessing Resources in Code

When your Android application is compiled, a **R** class gets generated, which contains resource IDs for all the resources available in your **res/** directory. You can use R class to access that resource using sub-directory and resource name or directly resource ID.

Example

To access *res/drawable/myimage.png* and set an ImageView you will use following code −

```
ImageView imageView = (ImageView) findViewById(R.id.myimageview);
imageView.setImageResource(R.drawable.myimage);
```

Here first line of the code make use of *R.id.myimageview* to get ImageView defined with id *myimageview* in a Layout file. Second line of code makes use of *R.drawable.myimage* to get an image with name **myimage** available in drawable sub-directory under **/res**.

Example

Consider next example where *res/values/strings.xml* has following definition −

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

Now you can set the text on a TextView object with ID msg using a resource ID as follows −

```
TextView msgTextView = (TextView) findViewById(R.id.msg);
msgTextView.setText(R.string.hello);
```

Example

Consider a layout *res/layout/activity_main.xml* with the following definition −

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

```
   <TextView android:id="@+id/text"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello, I am a TextView" />

   <Button android:id="@+id/button"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Hello, I am a Button" />

</LinearLayout>
```

This application code will load this layout for an Activity, in the onCreate() method as follows −

```
public void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
}
```

Accessing Resources in XML

Consider the following resource XML *res/values/strings.xml* file that includes a color resource and a string resource −

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
   <color name="opaque_red">#f00</color>
   <string name="hello">Hello!</string>
</resources>
```

Now you can use these resources in the following layout file to set the text color and text string as follows −

```
<?xml version="1.0" encoding="utf-8"?>
<EditText xmlns:android="http://schemas.android.com/apk/res/android"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:textColor="@color/opaque_red"
   android:text="@string/hello" />
```

Now if you will go through previous chapter once again where I have explained **Hello World!** example, and I'm sure you will have better understanding on all the concepts explained in this chapter. So I highly recommend to check previous chapter for working example and check how I have used various resources at very basic level.

Android Resource type

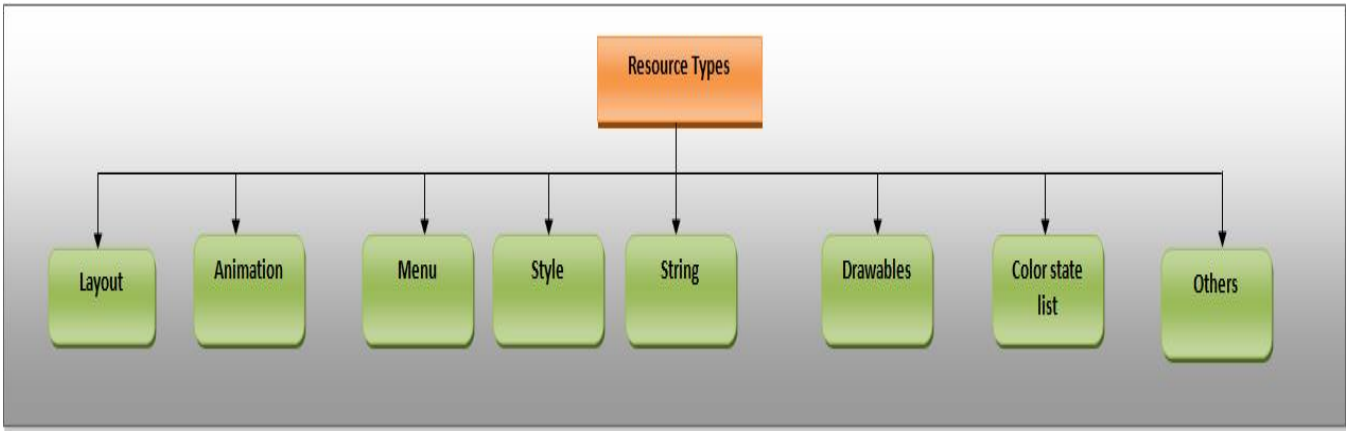
Android Resource types

Today we shall study types of Android resources. We shall have a brief introduction and a glance over the types. It is theoretical so please be patient.

Introduction

Android resource types are as follows:

- Drawable resources
- Color state list resources
- Animation resources
- Layout resources
- Menu resources
- Style resource
- String resources
- Others



Android Drawable Resources

These resources define the graphics of application with xml or bitmaps. They are accessed from R.drawable class and saved in res/drawable/ folder. It is a general concept for a graphic that can be drawn to the screen which can be retrieved with APIs. The different types of drawables are as follows:

- Nine-Patch file: This is a PNG file which has stretchable regions. With this image can be resized according to content.
- Layer List: This is a drawable which manages an array of other Drawables They are drawn in array order. Element with largest index will be at top.
- Level list: This is an XML file. It defines one drawable which manages number of alternate drawables. These alternatives are assigned with a maximum number.
- Bitmap file: This is a simple bitmap graphic file.

- Clip drawable: This XML file defines a drawable that clips another drawable.
- Shape drawable: This XML file defines geometric shape.
- Transition drawable: This Xml file deals with the transition. It cross-fades between two drawable resources.

Android Color State List Resources

A ColorStateList is an object which can be defined in XML. This can be applied as a color. Depending on the state of view object to which it is applied the color actually changes. Each color can be defined in a XML file under <item /> tag. So the state list in an XML file can be described. When state changes, state list is traversed from top to bottom and the most suitable match is picked.

Android Animation Resource

There are two types of animations which an animation resource can refer to and they are:

1 Property Animation: An animator is used to set an object's property over a period of time. In short we modify the properties of object.

2 View Animation: There are two types of animations which can be viewed:

3 Frame animation: A sequence of images is displayed in order

4 Tween animation: An animation is created by performing a series of transformations on a single image.

Android Layout Resource

Android layout resource is used to define the user interface of application. We already know the usage of layouts so there is no point in redundancy.

5.4.6 Android Menu

Android Menu resource is used to design and define the menu of application. We have options menu, context menu and submenu. This can be inflated by MenuInflater.

5.4.7 Android String Resource

Android String resource provides text strings for application. We have an option to format text and style it as well. We have three types of resources:

- String Array: It is an xml resource which provides an array of strings
- String: This is an xml resource which provides a single string
- Quantity Strings: It is an xml resource. It carries the strings for pluralization.

Android Style Resources

Style resource as the name suggests it is going to define the format and look of user interface. An individual view can have a specific style. An entire activity or an application can be stylized by manifest file. It is nothing but a resource which has to be referenced properly.

Other

Other resources are listed below:

- Color: It is an xml resource which has a hexadecimal number. This number corresponds to a particular color.
- Bool: This resource carries a color value.
- Dimension: It contains the value of dimension with the specific unit of measure.
- ID: This is also an xml resource. This is an unique identifier which identifies application resources and components.
- Integer: It is an xml resource which carries an integer value.
- Typed Array: We can use this as array of drawables.
- Integer Array: It is an xml resource and it is an array of integers.

So congratulations guys. we are done with our short tutorial over resources. Stay tuned. As I say, keep practicing. Happy App Developing!!!

Unit -2

Android User Interface Design Essentials:

User Interface:

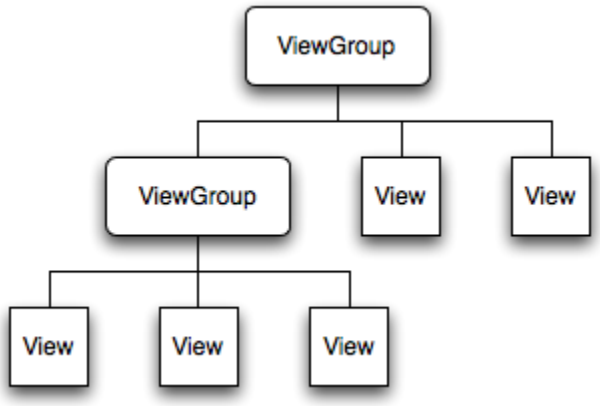
In an Android application, the user interface is built using `View` and `ViewGroup` objects. There are many types of views and view groups, each of which is a descendant of the `View` class.

View objects are the basic units of user interface expression on the Android platform. The View class serves as the base for subclasses called "widgets," which offer fully implemented UI objects, like text fields and buttons. The ViewGroup class serves as the base for subclasses called "layouts," which offer different kinds of layout architecture, like linear, tabular and relative.

A View object is a data structure whose properties store the layout parameters and content for a specific rectangular area of the screen. A View object handles its own measurement, layout, drawing, focus change, scrolling, and key/gesture interactions for the rectangular area of the screen in which it resides. As an object in the user interface, a View is also a point of interaction for the user and the receiver of the interaction events.

The user interface (UI) of your Android app is more than just a visible illustration. It plays a critical function in attracting users, providing a coherent and intuitive experience. A properly-designed UI enhances usability, ensures user interaction, and leaves a long-lasting experience. In this comprehensive guide, we will take you through the method of designing a user interface for your Android app, covering vital ideas, realistic suggestions, and in-depth insights to help you create an excellent user interface that sets your app apart from the competition.

On the Android platform, you define an Activity's UI using a hierarchy of View and ViewGroup nodes, as shown in the diagram below. This hierarchy tree can be as simple or complex as you need it to be, and



you can build it up using Android's set of predefined widgets and layouts, or with custom Views that you

Android - UI Design

In this chapter we will look at the different UI components of android screen. This chapter also covers the tips to make a better UI design and also explains how to design a UI.

User Interface

The user interface is a **visible part of the software**, hardware, or other computer-related systems, which is used for interaction. Actually, it is the means of user-program communication.

UI design defines what exactly **the interaction will look like**. In fact, it consists of different elements, types, and approaches. Every application or website, each computer program, or even code-based device uses it. All of the visual components are actually elements of the UI design.

As a matter of fact, UI impacts not only small visual details but also **defines the whole communicational structure** of the product both visible and hidden. For example, each application has a common view, in other words, the way it is supposed to look for the end-user. Also, they have a command-line user interface, visible only on the coding level, which is used mostly by the dedicated developers. Both of them are UI-based. Also, it is important to understand that a graphic user interface is not the only possible type of UI. Frankly speaking, the multiplicity of types of UI is huge. This amount of variation is caused by the different approaches to using the applications or devices and the constant improvement of their technical capabilities. It complicates making the correct choice because the specific product requires a reference UI design. This is why it is better to get acquainted with all the types of UI, before starting the development of your product. This will help you not only to save time and effort but to, possibly, choose a better method of implementation. So what are the other types of UI?

Types of UI Design Patterns

At the moment there are **5 major types of UI designs**, that differ in the working principles of interaction: **Command-line user interface** (CLI) **Graphical UI** (GUI), **Touchscreen UI** (TUI), and **Voice-controlled UI** (VUI), and **Gesture-based UI** (GbUI). Some of them have their own subtypes. So, let's take a closer look at each of them and find out in which case these types of UI could be used.

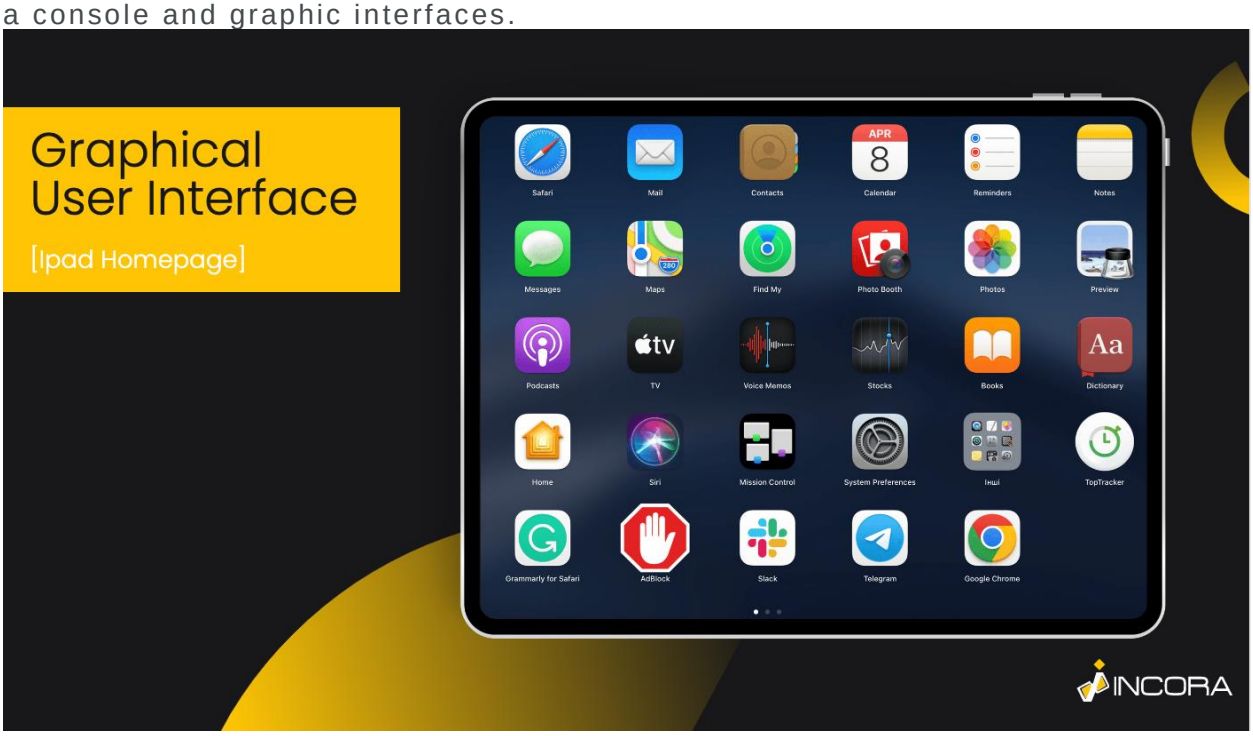
Command-line User Interface

It is one of the first interfaces ever and maybe **the most widespread one**. It is still used almost everywhere as an interaction tool for developers.

It is a **text-only interface**, where all the actions are performed by textual commands. The command-line user interface is often used as a basic platform for most code-based projects.

Moreover, most programming apps or programs are created in the command-line user interface as well. For example, GitHub provides codespaces, as an opportunity to **write and check the code online** and to present their code as a CV or portfolio for employers or other programmers.

The GitHub website is a command-line user interface, but it is more a hybrid of a console and graphic interfaces.



Graphical User Interface

It is the **most common and advanced** type of UI nowadays.

According to the name, the main communication method of this type is visual elements. In this case, graphic components are triggers of the actual digital processes, so the process starts after activating a button or other UI elements. So, unlike the CLI, GUI is **more user-friendly** and allows using all the features without any additional background thanks to the fact, that all the commands are automatized.

Also, most of the graphic components are created according to psychological research in order to **simplify the understanding of these icons or widgets** and to make their functionality as obvious as possible. It is sometimes called the “natural user interface”, meaning that using it feels as if it is natural.

Actually, the natural user interface is a very **subjective thing**, that varies depending on the context of a specific user or group of users. To create such an

interface you will definitely need a lot of UX as well, in order to analyze the specifics of user interactions.

Also, talking about GUI we must say that the variety of UI design approaches, in this case, is the biggest. For example, when creating a website you have at least 4 different ways to embody the GUI: to create a **landing page interface**, a **menu-driven interface**, a Dashboard application, or use the **scroll telling**.

Even if all of these interfaces can exist on their own, it is **better to mix them** and use them rather as elements of the system, instead of as independent approaches.

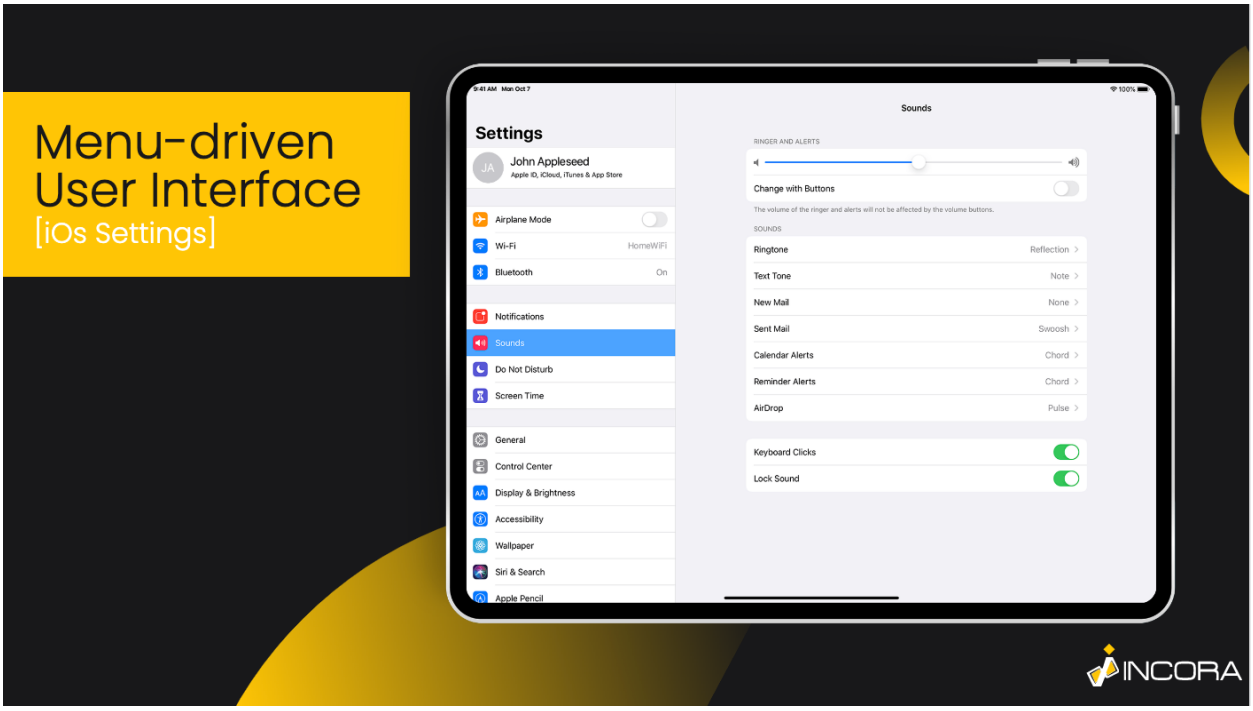
Landing Page Interface:

It is when instead of creating a combination of different pages, linked with each other, the software development team simply creates a single-page interface. In this case, all the information is presented **on the same page** and could be, for example, grouped by categories, etc.

The main aspect is that the interaction with such a page is limited to one main orientation (vertical or sometimes horizontal) Shadda, one of our recent projects, is a great example of a landing page interface.

Mostly, the landing page interface is **used for advertisement or promotion**, because it is easier for the user to perceive the information. This is why it is highly important to understand the **key information** you want to convey and create a correct order of its essential aspects.

The landing page interface also can exist independently, it can be implemented as a part of a full-fledged website.



Menu-driven Interface

It is the **most common GUI** design approach. Actually, it seems that there is no need for a further detailed explanation. In order to simplify the user's orientation over the website or an app, you create a menu with different rooms of information, actions or multimedia groped by different common features or topics.

Also, there is another possible definition of the **menu-driven interface**. For instance, it is an interface, that offers the opportunity to use limited functionality operating the pre-designed list of actions to be done. The best example of such a case is ATM. When you use ATM - you definitely use the menu-driven interface, that allows you to withdraw money or top up your account, check your card balance, etc.

Nevertheless, both cases provide you with a predesigned menu or a list of possible features you can choose.

Admin Dashboard Application:

Another example of GUI design is a **dashboard**.

One of the key features of dashboard apps - is its user-friendly look. It is commonly used to represent certain metrics and aspects, related to the working

processes. It is a **centralized panel**, where all crucial business and performance metrics are gathered and visualized.

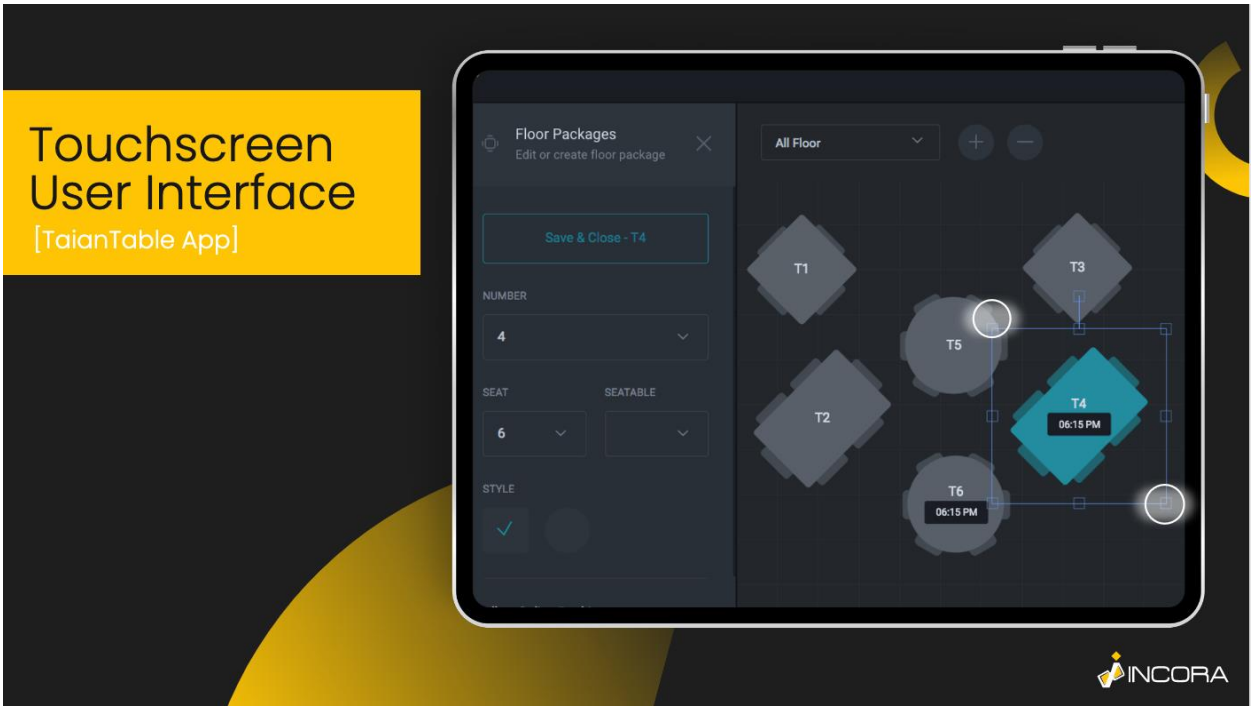
It is a complex solution, which requires setting countless backend and frontend processes. Fortunately, it can still be developed even by a single team extension specialist, who is experienced enough.

Scrollytelling and Feed

It is **mostly used in media or blogs**. Scrollytelling may be also confused as “feed”, but they are different. The principles of its work are easy, as you can understand from its name it is about “scrolling” and “storytelling”.

Usually, it is a single endless page, where all the information is provided. It is a popular type of UI among media because it is untrivial at the moment and easily of interest to the users. It allows making the long-read format interactive.

On the other hand, feed is not actually the type of user interface, but a UI concept. The similarity between feed and scrollytelling is that they present a lot of information on a single page with almost endless scrolling chances. The difference is that while scrollytelling provides a logically connected story, feed gives a bunch of different informational content, which can have no actual linkage at all.

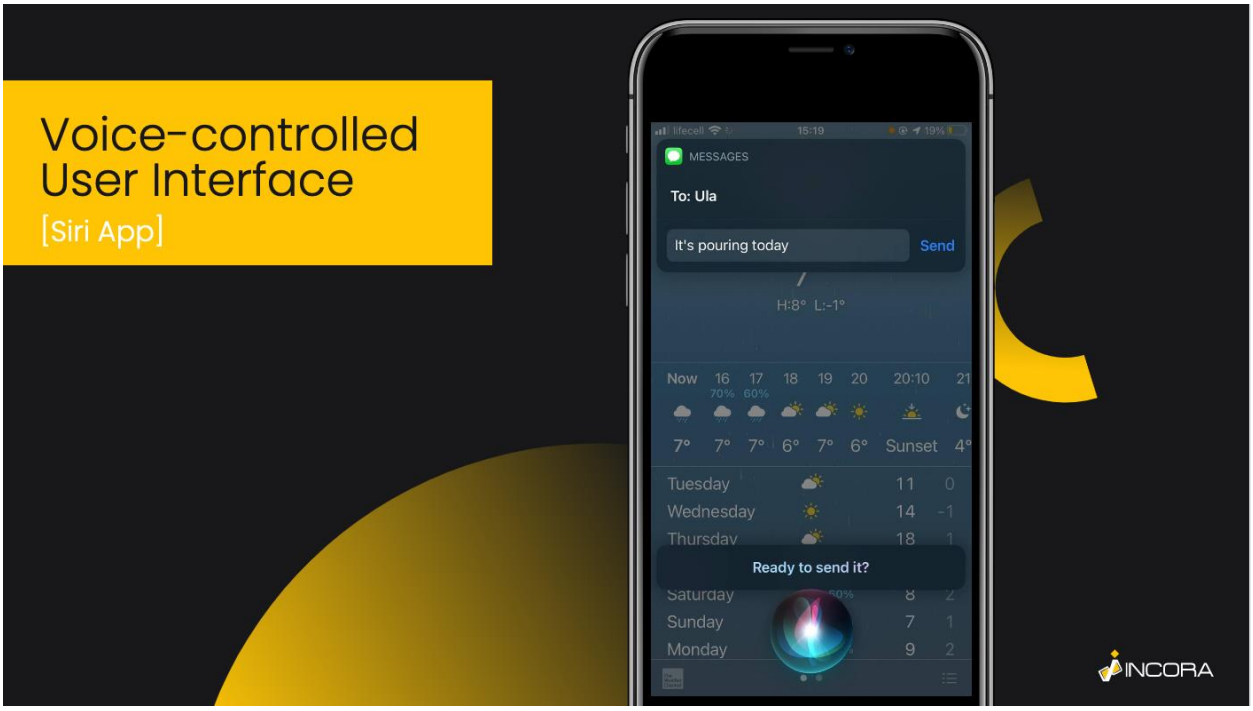


Touchscreen UI

As an **alternative to the usual desktop UI**, the Touchscreen one appeared. Thanks to the growth and rapid development of the mobile sector, as well as the invention of smartphones the need for new UI design patterns arose.

Actually, the main difference between touchscreen UI and, for example, graphical UI consists of the **interaction methods**. It is obvious, that Touchscreen UI's main interaction tools are **toches and scrolling**, while desktop GUI uses a keyboard, mouse, or other controllers, as well as a virtual cursor.

The main additional benefit of using the touchscreen UI is that it is a more natural user interface. It is more obvious for most people how to use a touch interaction system, than different combinations of various icons and signs and diverse controllers.



Voice-controlled UI

In contradistinction to previous ones, voice-controlled UI **has no visual content**. Almost all the operations are done using human voice commands. The system architecture in this case is absolutely different.

Obviously, it would be a crazy idea to choose a voice-controlled UI over the usual graphical one for a website. Nevertheless, it is how technical revolutions happen. For example, it is a great idea for a website, whose target audience is blind people.

Even though this type of UI does not care about the look or visualization, **there are numerous issues** to deal with. For example, one of them is the ability to **distinguish voice and commands**. In addition, the problem with language implementation appears, because different languages obviously have different pronunciations, etc. The list of possible issues related to language distinction is much bigger.

Also, don’t forget about the natural user interface, meaning that commands must be obvious, and easy. It is a great idea to **add synonym keywords**, so there would be no misunderstandings.

At the moment, this UI is **growing its potential**. Mostly, it is used in virtual assistants such as **Siri** for iPhone or **Alexa** from Amazon. Most tech giants use this method to create new devices or to improve already existing ones. As a great bonus, some of such bots are offered as a development platform for third-party companies. For example, you can upgrade your application using the custom voice technology assistant, based on the Alexa platform. Or, you can also use ChatGPT integration to build a virtual assistant.

UI screen components

A typical user interface of an android application consists of action bar and the application content area.

- Main Action Bar
- View Control
- Content Area
- Split Action Bar

These components have also been shown in the image below –

Understanding Screen Components

The basic unit of android application is the activity. A UI is defined in an xml file. During compilation, each element in the XML is compiled into equivalent Android GUI class with attributes represented by methods.

What are some examples of UI elements?

Common UI elements include:

Breadcrumbs
Checkboxes
Dropdowns
Forms
Icons
Input fields
Notifications

And many more.

3. What are the key elements of UI design?

UI elements usually fall into one of the following four categories:

Input controls allow users to input information into the system. If you need your users to tell you what country they are in, for example, you'll use an input control to let them do so.

Navigational components help users move around a product or website. Common navigational components include tab bars on an iOS device and a hamburger menu on an Android.

Informational components share information with users.

Containers hold related content together.

UI Elements:

UI elements usually fall into one of the following four categories:

- **Input controls** allow users to input information into the system. If you need your users to tell you what country they are in, for example, you'll use an input control to let them do so.
- **Navigational components** help users move around a product or website. Common navigational components include tab bars on an iOS device and a hamburger menu on an Android.
- **Informational components** share information with users.
- **Containers** hold related content together.

Here is our list of the most common and important UI elements:

1 Bento Menu

A bento menu, named after bento boxes, represents a menu with grid items. As you read along, you'll begin to notice UI designer is just another word for a foodie—we love to name our UI elements after food.

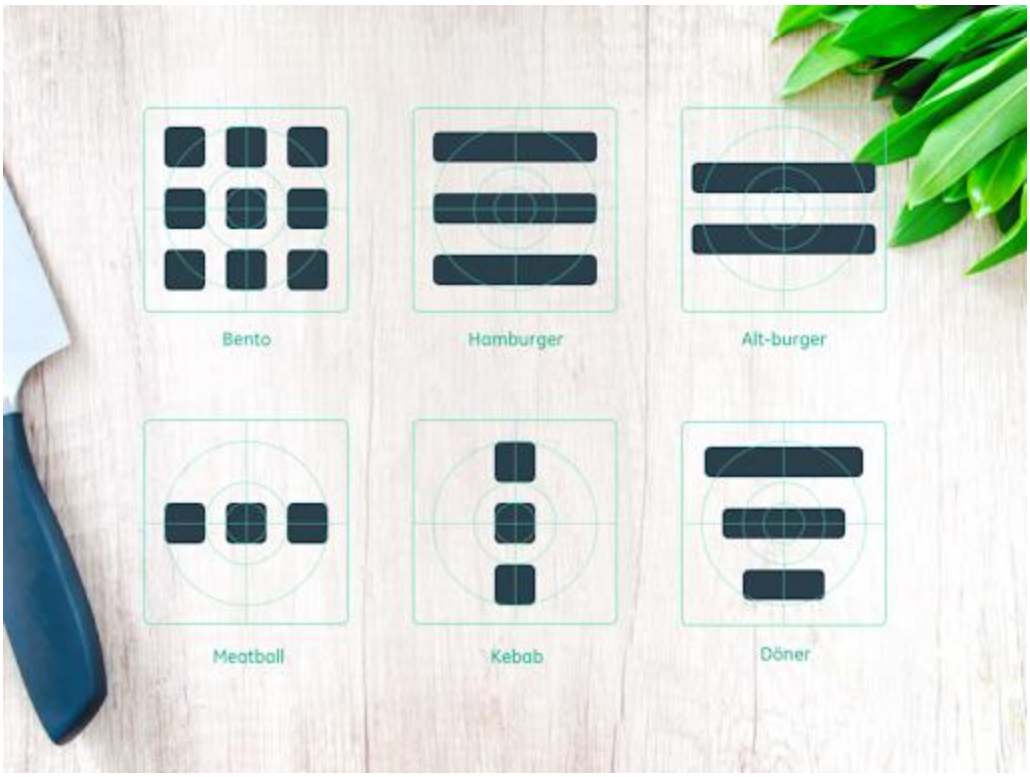


*Image by **Alex Lockwood***

2. Breadcrumb

These little trails of links help users figure out where they are within a website. Often located at the top of a site, breadcrumbs let users see their current location and the proceeding pages. Users are also able to click on them to move between steps.

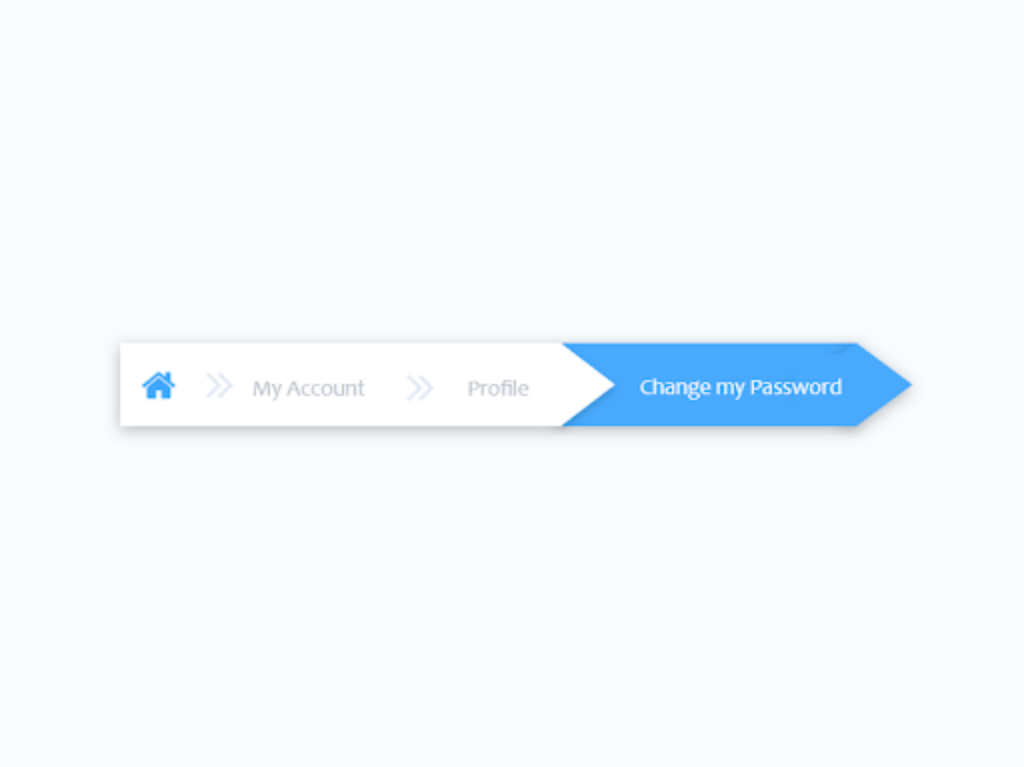


*Image by **Sharon Olorunniwo***

4. Button

Traditionally displayed as shapes with a label, buttons are a vital user element that tells users they can perform a particular action, like submitting.

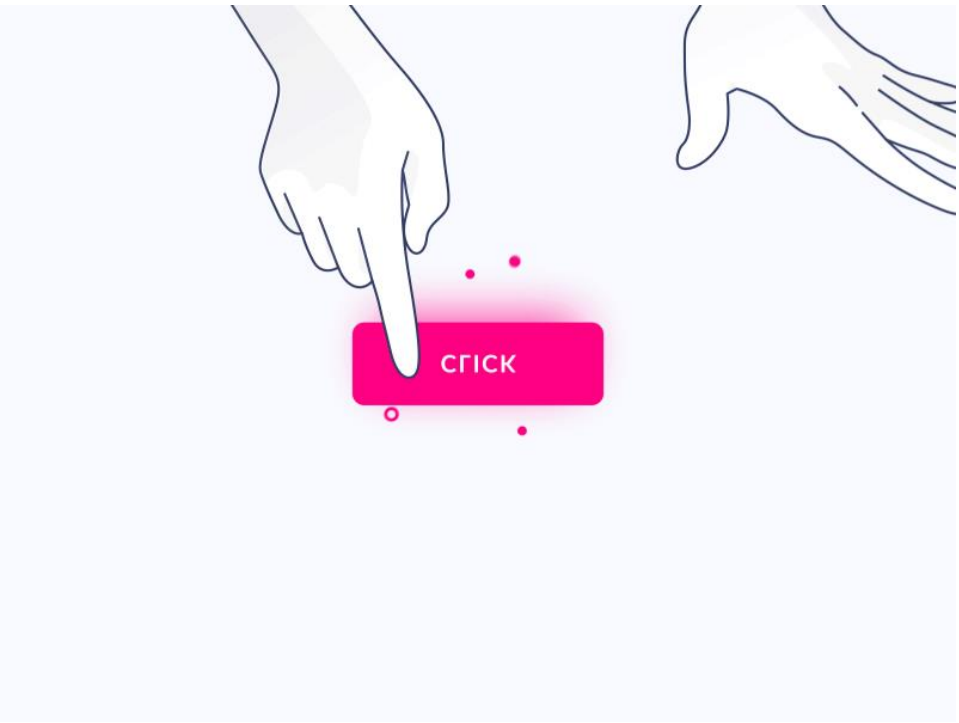


*Image by **Gal Shir***

5. Card

Super popular these days, cards are small rectangular or square modules that contain different kinds of information—in the form of buttons, text, rich media, and so on. These user elements act as an entry point for the user, displaying different kinds of content side by side which the user can then click on.

Cards are a great UI design choice if you want to make smart use of the space available and present the user with multiple content options, without making them scroll through a traditional list.

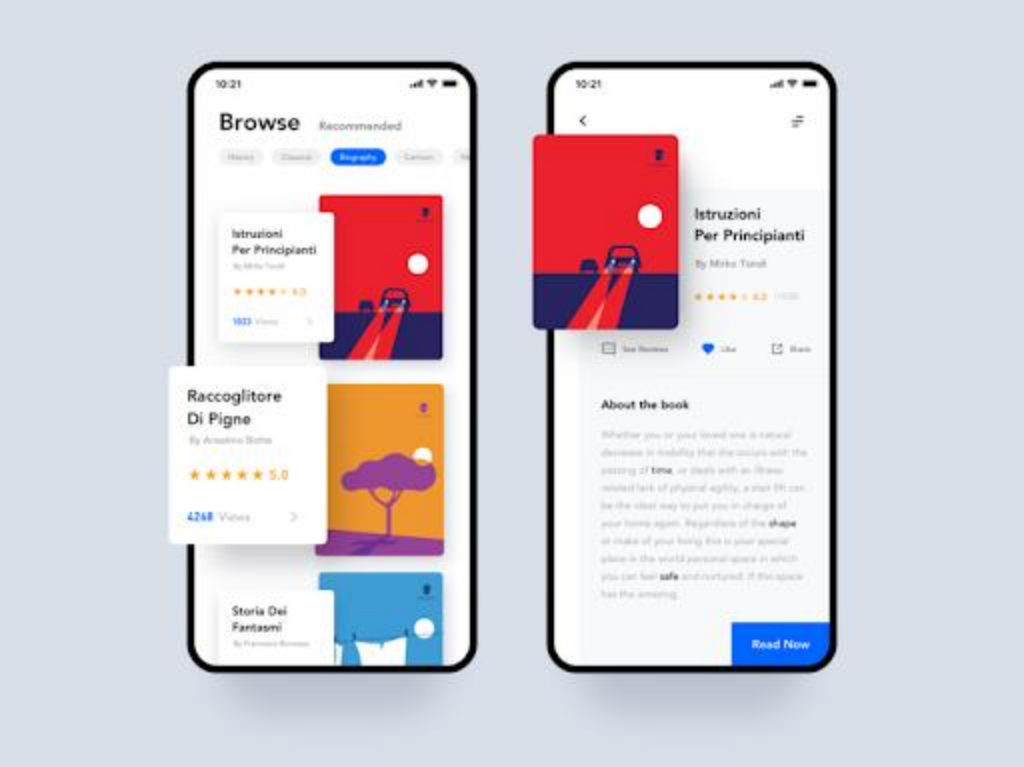
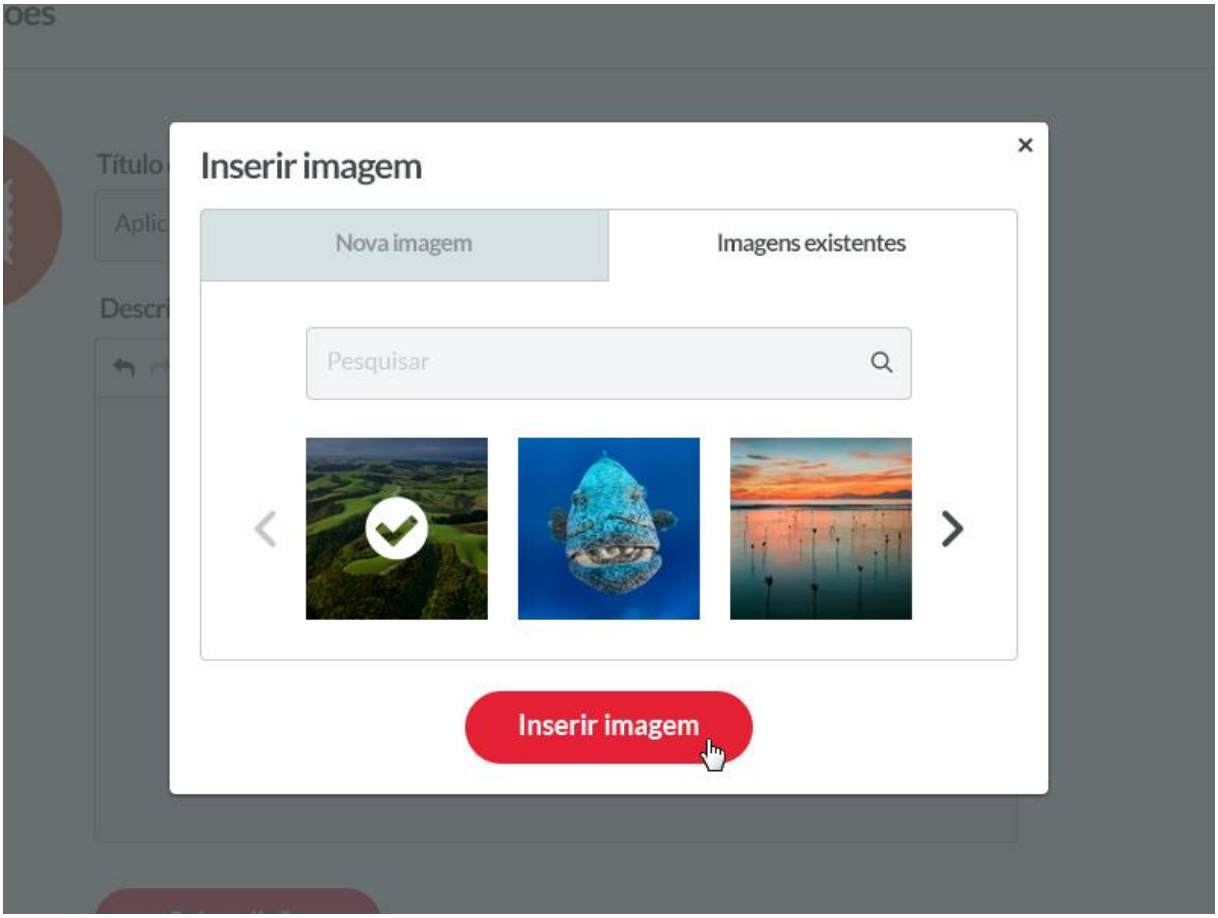


*Image by **Crank***

6. Carousel

Carousels allow users to browse through sets of content, like images or cards, often hyperlinked to more content or sources. The biggest advantage of using carousels in UI design is that they enable more than one piece of content to occupy the same area of space on a page or screen. If you decide to use carousels, be sure to follow **these guidelines set out by the Nielsen Norman Group**.



7. Checkbox

In UI design, a checkbox appears exactly as the name suggests: a little square box on the screen that the user can check or uncheck. A checkbox allows users to select one of multiple options from a list, with each checkbox operating as an individual. Checking the checkbox marks it with a little tick! Some common use cases for this element include forms and databases.

*Image by **Shakuro***

8. Comment

Pretty common around interfaces today, comments display content users input into the system in chronological order. You've seen them around social media engines and on blog posts.

9. Döner Menu

A döner menu is a variation of the more well-known hamburger menu. While a hamburger menu consists of three lines of equal length stacked one on top of the other, a döner menu consists of a vertical stack of three lines of different

lengths: a long line, a shorter line below it, and an even shorter line underneath that! This UI element represents a group of filters.

10. Dropdown

This controversial UI element allows users to select an item from a list that “drops down” once they click on it. To learn more about why this element has a lousy rep, check out this

12. Form

Forms help users input sets of related information into the system and submit them. Think of all the boxes asking for shipping details when you order something online.

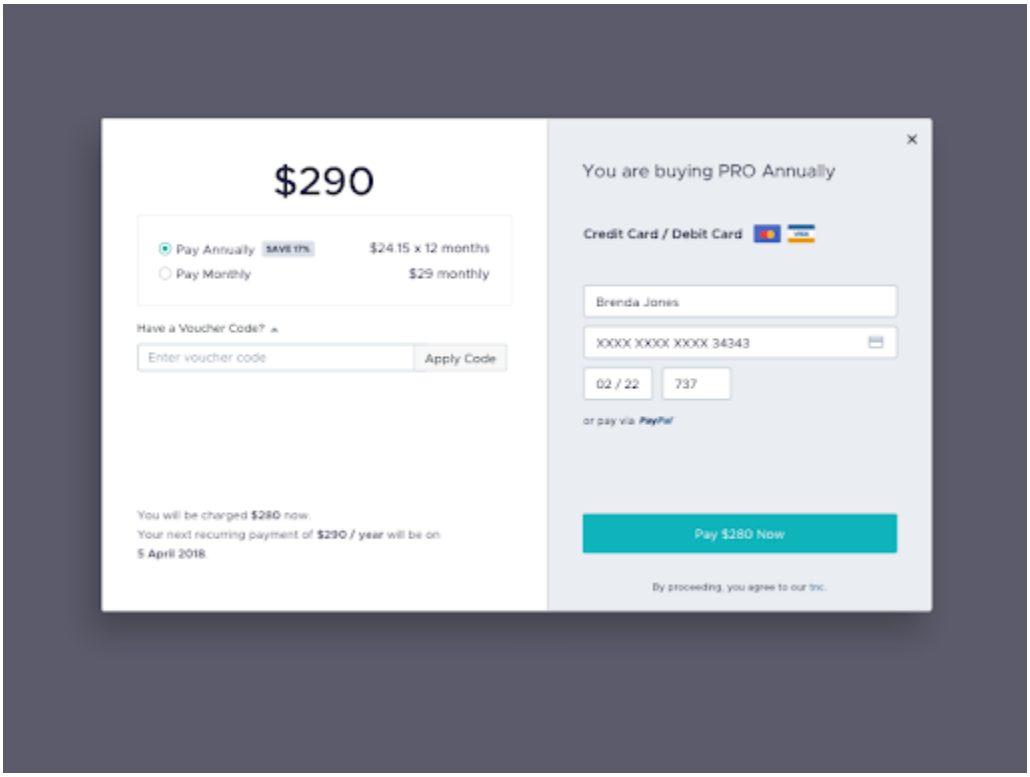


*Image by **girithaara***

13. Hamburger Menu

These three little horizontal lines look slightly like America's quintessential meal and represent a list menu. You'll commonly find the hamburger menu on the top left-hand corner of apps and most likely containing a group of navigation links.

14. Icon

Icons are images used to communicate a variety of things to users. They can help to better communicate content or can communicate and trigger a specific action. You can learn how to design icons from scratch in this **step-by-step guide to the icon design process**.

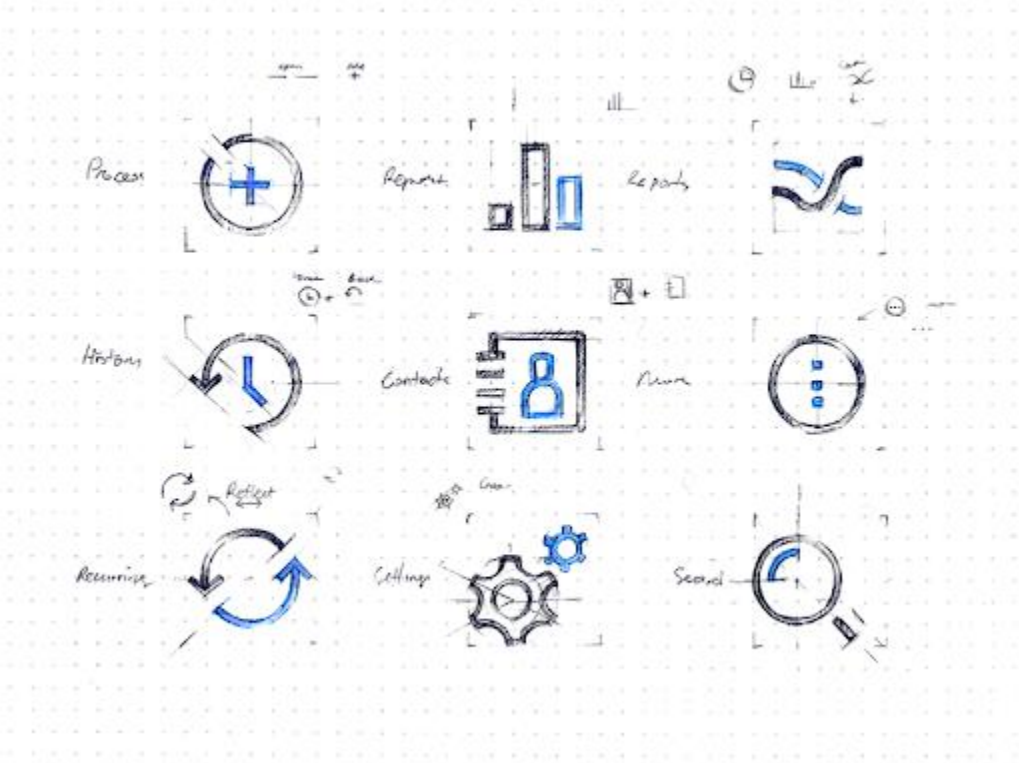


*Image by **Eddie Lobanovskiy***

15. Input Field

Input fields are, quite simply, a place for users to enter content into the system. They aren't just limited to forms, either—**search bars** are input fields as well.

.

Image by Kickass

19. Modal

A modal window is a small box containing content or a message that requires you to interact with it before you can close it and return to your flow.

Think of the last time you deleted an item on your phone. The little message that popped up asking you to confirm that you wanted to remove it is a modal.

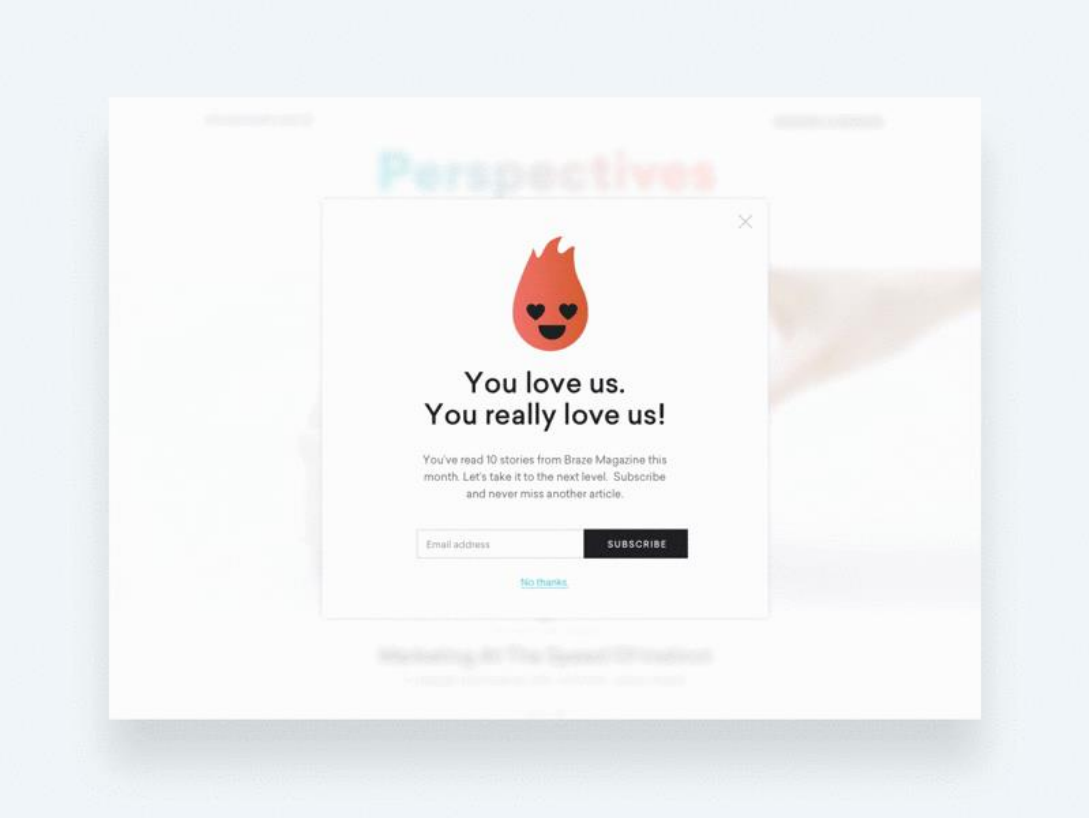


*Image by **Joshua Krohn***

20. Notification

You'll find these little red dots everywhere on interfaces today. They let us know there is something new, like a message, for us to go check out.

However, notifications don't just tell us someone has liked one of our posts—they can let us know an error occurred, or a process was successful.

21. Pagination

Typically found near the bottom of a page, pagination organizes content into pages. Pagination lets users know where they are within a page and click to move into other sections.

22. Picker

Date and time pickers let users pick dates and times. The advantage of using pickers over input fields is the ability to keep all the data users enter tidy and consistently formatted in a database, making information manageable and easy to access.

*Image by **Edoardo Mercati***

23. Progress Bar

Progress bars help users visualize where they are in a series of steps. You'll commonly find them on checkouts, marking the different stages a user needs to complete to finalize a purchase, like billing and shipping.

24. Radio Buttons

Often confused with checkboxes, radio buttons are small circular elements that let users select one option among a list. The key here is noting that users can only choose one option and not multiple options like they can with checkboxes.

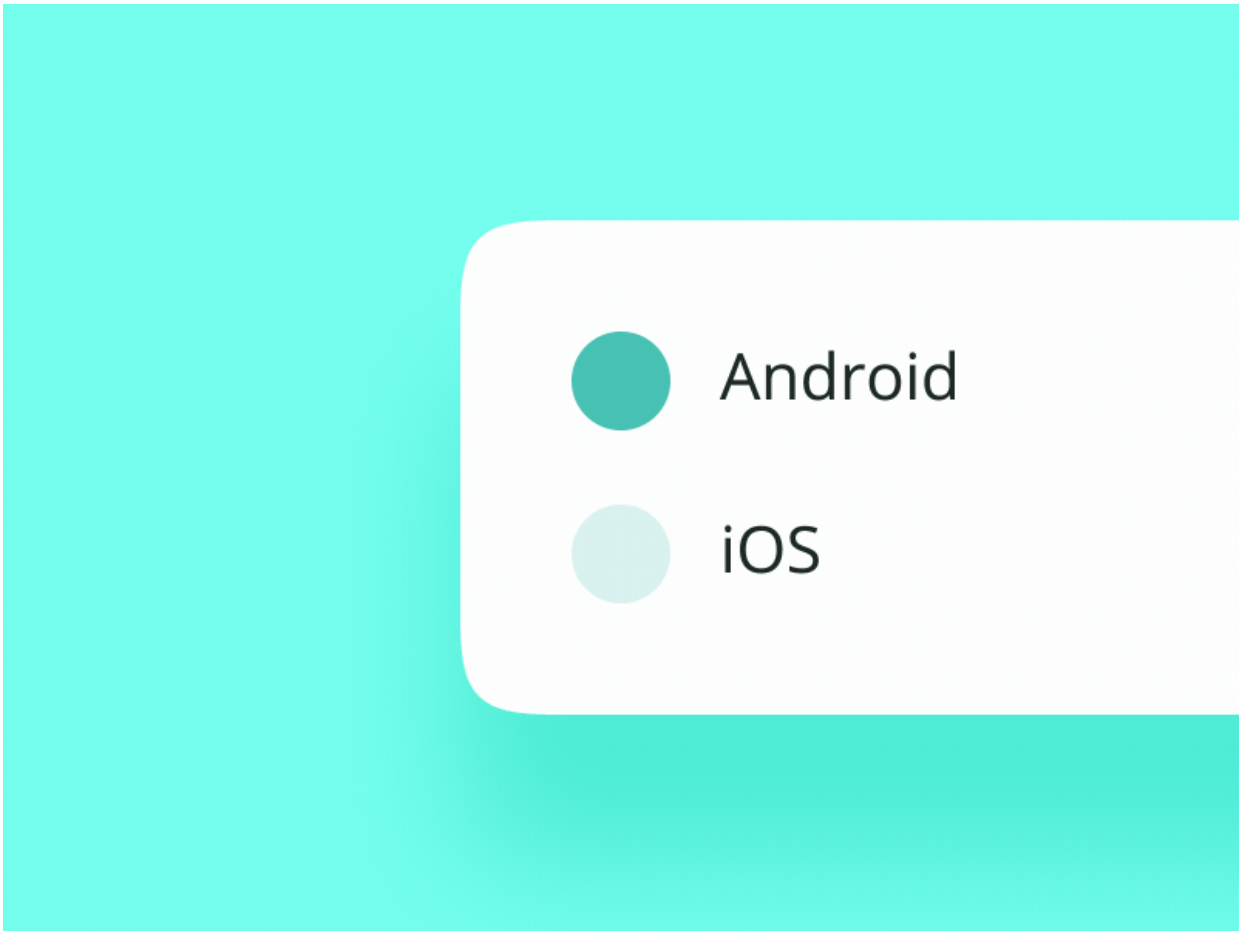


*Image by **Oleg Frolov***

25. Search Field

Commonly represented as an input field with a little magnifying glass inside of it, search fields allow users to input information to find within the system.

26. Sidebar

A sidebar displays a group of navigational actions or content literally on the side of a page. It can be visible or tucked away.

27. Slider Controls

Sliders are a common UI element used for selecting a value or a range of values. By dragging the slider with their finger or mouse, the user can gradually and finely adjust a value—like volume, brightness, or desired price range when shopping.

28. Stepper

Steppers are two-segment controls that also let users adjust a value. However, unlike sliders, they allow users to alter the value in predefined increments only.

29. Tag

In UI design, tags are essentially labels which help to mark and categorize content. They usually consist of relevant keywords which make it easier to find and browse the corresponding piece of content. Tags are often used on social websites and blogs.

30. Tab Bar

Tab bars appear at the bottom of a **mobile app** and allow users to quickly move back and forth between the primary sections of an app.

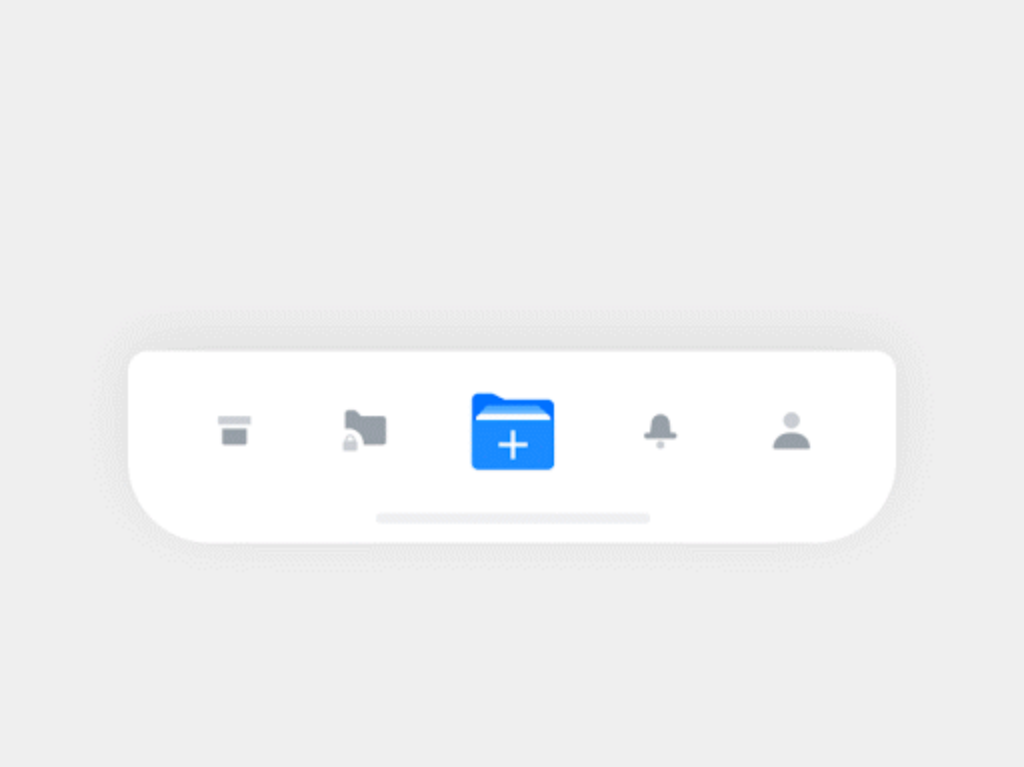


*Image by **[Hoang Nguyen]***

31. Tooltip

Tooltips provide little hints that help users understand a part or process in an interface.

32. Toggle

Think of toggles as on and off switches. They let us do just that: turn something on or off.

Designig UI with layouts:

An activity is consist of views. A view is just a widget that appears on the screen. It could be button e.t.c. One or more views can be grouped together into one GroupView. Example of ViewGroup includes layouts.

Types of layout

There are many types of layout. Some of which are listed below –

- Linear Layout
- Absolute Layout
- Table Layout
- Frame Layout
- Relative Layout

Linear Layout

Linear layout is further divided into horizontal and vertical layout. It means it can arrange views in a single column or in a single row. Here is the code of linear layout(vertical) that includes a text view.

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

   <TextView
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:text="@string/hello" />
</LinearLayout>
```

AbsoluteLayout

The AbsoluteLayout enables you to specify the exact location of its children. It can be declared like this.

```
<AbsoluteLayout
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   xmlns:android="http://schemas.android.com/apk/res/android" >

   <Button
      android:layout_width="188dp"
      android:layout_height="wrap_content"
      android:text="Button"
      android:layout_x="126px"
      android:layout_y="361px" />
</AbsoluteLayout>
```

TableLayout

The TableLayout groups views into rows and columns. It can be declared like this.

```
<TableLayout
   xmlns:android=”http://schemas.android.com/apk/res/android”
   android:layout_height=”fill_parent”
   android:layout_width=”fill_parent” >

   <TableRow>
      <TextView
         android:text=”User Name:”
         android:width =”120dp”
         />

      <EditText
         android:id=”@+id/txtUserName”
         android:width=”200dp” />
   </TableRow>

</TableLayout>
```

RelativeLayout

The RelativeLayout enables you to specify how child views are positioned relative to each other.It can be declared like this.

```
<RelativeLayout
   android:id=”@+id/RLayout”
   android:layout_width=”fill_parent”
   android:layout_height=”fill_parent”
   xmlns:android=”http://schemas.android.com/apk/res/android” >
</RelativeLayout>
```

FrameLayout

The FrameLayout is a placeholder on screen that you can use to display a single view. It can be declared like this.

```
<?xml version=”1.0” encoding=”utf-8”?>
<FrameLayout
   android:layout_width=”wrap_content”
   android:layout_height=”wrap_content”
   android:layout_alignLeft=”@+id/lblComments”
   android:layout_below=”@+id/lblComments”
   android:layout_centerHorizontal=”true” >
```

```
   <ImageView
      android:src = "@drawable/droid"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />
</FrameLayout>
```

Apart form these attributes, there are other attributes that are common in all views and ViewGroups. They are listed below –

Sr.No	View & description
1	**layout_width** Specifies the width of the View or ViewGroup
2	**layout_height** Specifies the height of the View or ViewGroup
3	**layout_marginTop** Specifies extra space on the top side of the View or ViewGroup
4	**layout_marginBottom** Specifies extra space on the bottom side of the View or ViewGroup
5	**layout_marginLeft** Specifies extra space on the left side of the View or ViewGroup
6	**layout_marginRight** Specifies extra space on the right side of the View or ViewGroup
7	**layout_gravity** Specifies how child Views are positioned
8	**layout_weight** Specifies how much of the extra space in the layout should be allocated to the View

Animation in Android with Example

Last Updated : 06 Jun, 2024

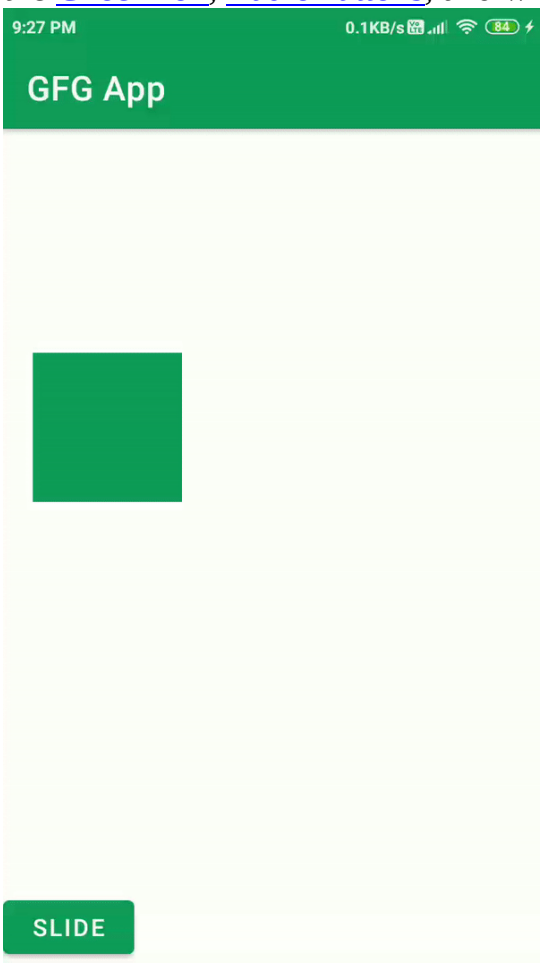
Animation is the process of adding a motion effect to any view, image, or text. With the help of an animation, you can add motion or can change the shape of a specific

view. Animation in Android is generally used to give your UI a rich look and feel. The animations are basically of three types as follows:

1. **Property Animation**
2. **View Animation**
3. **Drawable Animation**

1. Property Animation

Property Animation is one of the robust frameworks which allows animation 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.



2. View Animation

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. The example of View Animation can be seen in Expandable RecyclerView.

3. Drawable Animation

Drawable Animation is used if you want to animate one image over another. The simple way to understand is to animate drawable is to load the series of drawable 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.

Important Methods of Animation

Methods	Description
startAnimation()	This method will start the animation.
clearAnimation()	This method will clear the animation running on a specific view.

Example of Implementation Android Animation

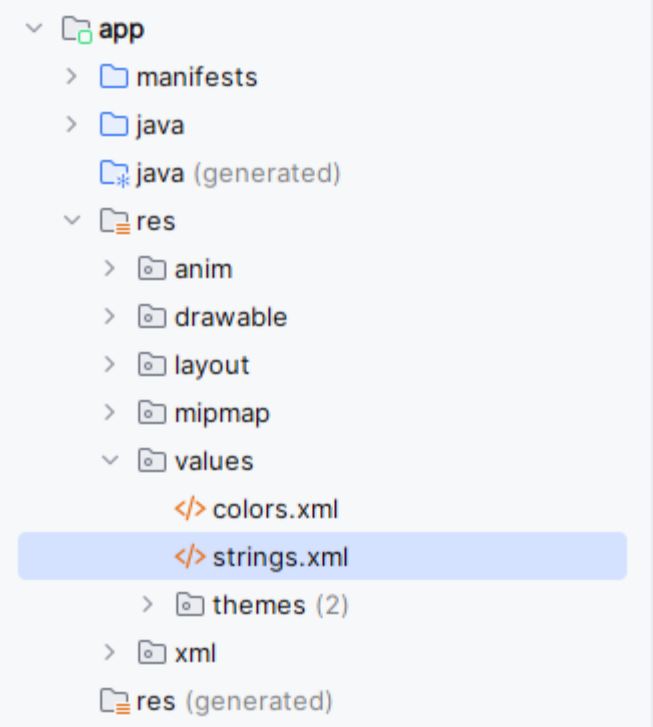
Now we will see the Simple Example to add animations to ImageView. Note that we are going to implement this project using the **Java** language.

Step 1: Create a New Project

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

Step 2: Working with the strings.xml file

Strings.xml can be found from the **app > res > values > strings.xml**.



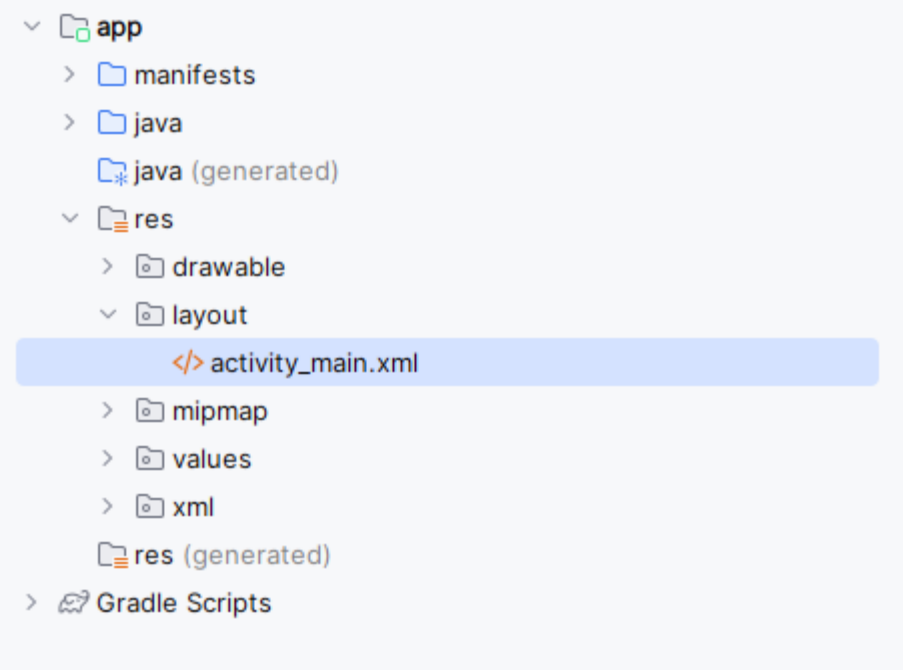
Below is the snippet for the strings.xml file.

strings.xml

```
<resources>
    <string name="app_name">GFG App</string>
    <string name="blink">BLINK</string>
    <string name="clockwise">ROTATE</string>
    <string name="fade">FADE</string>
    <string name="move">MOVE</string>
    <string name="slide">SLIDE</string>
    <string name="zoom">ZOOM</string>
    <string name="stop_animation">STOP ANIMATION</string>
    <string name="course_rating">Course Rating</string>
    <string name="course_name">Course Name</string>
</resources>
```

Step 3: Working with the activity_main.xml file

Create ImageView in the **activity_main.xml** along with buttons that will add animation to the view. Navigate to the **app** > **res** > **layout** > **activity_main.xml**.



Below is the code for the **activity_main.xml** file.

activity_main.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=".MainActivity">

    <ImageView
        android:id="@+id/imageview"
        android:layout_width="200dp"
        android:layout_height="200dp"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="40dp"
        android:contentDescription="@string/app_name"
        android:src="@drawable/gfgimage" />

    <LinearLayout
        android:id="@+id/linear1"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/imageview"
        android:layout_marginTop="30dp"
        android:orientation="horizontal"
        android:weightSum="3">
```

```
        <!--To start the blink animation of the image-->
        <Button
            android:id="@+id/BTNblink"
            style="@style/TextAppearance.AppCompat.Widget.Button"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:layout_weight="1"
            android:padding="3dp"
            android:text="@string/blink"
            android:textColor="@color/white" />

        <!--To start the rotate animation of the image-->
        <Button
            android:id="@+id/BTNrotate"
            style="@style/TextAppearance.AppCompat.Widget.Button"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:layout_weight="1"
            android:padding="3dp"
            android:text="@string/clockwise"
            android:textColor="@color/white" />

        <!--To start the fading animation of the image-->
        <Button
            android:id="@+id/BTNfade"
            style="@style/TextAppearance.AppCompat.Widget.Button"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:layout_weight="1"
            android:padding="3dp"
            android:text="@string/fade"
            android:textColor="@color/white" />

    </LinearLayout>
    <LinearLayout
        android:id="@+id/linear2"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/linear1"
        android:layout_marginTop="30dp"
        android:orientation="horizontal"
        android:weightSum="3">

        <!--To start the move animation of the image-->
        <Button
            android:id="@+id/BTNmove"
```

```
            style="@style/TextAppearance.AppCompat.Widget.Button"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:layout_weight="1"
            android:padding="3dp"
            android:text="@string/move"
            android:textColor="@color/white" />

        <!--To start the slide animation of the image-->
        <Button
            android:id="@+id/BTNslide"
            style="@style/TextAppearance.AppCompat.Widget.Button"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:layout_weight="1"
            android:padding="3dp"
            android:text="@string/slide"
            android:textColor="@color/white" />

        <!--To start the zoom animation of the image-->
        <Button
            android:id="@+id/BTNzoom"
            style="@style/TextAppearance.AppCompat.Widget.Button"
            android:layout_width="0dp"
            android:layout_height="wrap_content"
            android:layout_margin="10dp"
            android:layout_weight="1"
            android:padding="3dp"
            android:text="@string/zoom"
            android:textColor="@color/white" />

    </LinearLayout>

    <!--To stop the animation of the image-->
    <Button
        android:id="@+id/BTNstop"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/linear2"
        android:layout_marginLeft="30dp"
        android:layout_marginTop="30dp"
        android:layout_marginRight="30dp"
        android:text="@string/stop_animation" />

</RelativeLayout>
```

Step 4: Create 6 different types of animation for ImageView

To create new animations we have to create a new directory for storing all our animations. Navigate to the **app > res > Right-Click on res >> New >> Directory >> Name your directory as "anim"**. Inside this directory, we will create our animations. For creating a new anim right click on the anim directory >> Animation Resource file and give the name to your file. Below is the code snippet for 6 different animations.

1. Blink Animation

blink_animation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
    <alpha android:fromAlpha="0.0"
        android:toAlpha="1.0"
        android:interpolator="@android:anim/accelerate_interpolator"
        android:duration="500"
        android:repeatMode="reverse"
        android:repeatCount="infinite"/>
</set>
```

2. Fade Animation

fade_animation.xml

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

    <!-- duration is the time for which animation will work-->
    <alpha
        android:duration="1000"
        android:fromAlpha="0"
        android:toAlpha="1" />

    <alpha
        android:duration="1000"
        android:fromAlpha="1"
        android:startOffset="2000"
        android:toAlpha="0" />

</set>
```

3. Move Animation

move_animation.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:fromXDelta="0%p"
        android:toXDelta="75%p"
        android:duration="700" />
</set>
```

4. Rotate Animation

rotate_animation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set
     xmlns:android="http://schemas.android.com/apk/res/android">
    <rotate
        android:duration="6000"
        android:fromDegrees="0"
        android:pivotX="50%"
        android:pivotY="50%"
        android:toDegrees="360" />

    <rotate
        android:duration="6000"
        android:fromDegrees="360"
        android:pivotX="50%"
        android:pivotY="50%"
        android:startOffset="5000"
        android:toDegrees="0" />

</set>
```

5. Slide Animation

slide_animation.xml

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android"
     android:fillAfter="true" >
    <scale
        android:duration="500"
        android:fromXScale="1.0"
        android:fromYScale="1.0"
        android:interpolator="@android:anim/linear_interpolator"
        android:toXScale="1.0"
        android:toYScale="0.0" />
</set>
```

6. Zoom Animation

zoom_animation.xml

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

    <scale
        android:interpolator="@android:anim/linear_interpolator"
        android:duration = "1000"
        android:fromXScal = "1"
        android:fromYScale = "1"
        android:pivotX = "50%"
        android:pivotY = "50%"
        android:toXScale = "2"
        android:toYScale = "2"/>
</set>
```

Step 5: Working with the MainActivity.java file

Add animation to the ImageView by clicking a specific Button. Navigate to the **app > java > your apps package name >> MainActivity.java**.

Below is the implementation of MainActivity file:

MainActivity.javaMainActivity.kt

```
import androidx.appcompat.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
```

```
import android.widget.Button;
import android.widget.ImageView;

public class MainActivity extends AppCompatActivity {

    // Declare ImageView and Buttons
    ImageView imageView;
    Button blinkBTN, rotateBTN, fadeBTN, moveBTN, slideBTN, zoomBTN, stopBTN;

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

        // Initialize ImageView and Buttons using their IDs
        imageView = findViewById(R.id.imageview);
        blinkBTN = findViewById(R.id.BTNblink);
        rotateBTN = findViewById(R.id.BTNrotate);
        fadeBTN = findViewById(R.id.BTNfade);
        moveBTN = findViewById(R.id.BTNmove);
        slideBTN = findViewById(R.id.BTNslide);
        zoomBTN = findViewById(R.id.BTNzoom);
        stopBTN = findViewById(R.id.BTNstop);

        // Set up click listener for blink button to start blink animation
        blinkBTN.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Load blink animation and start it on the ImageView
                Animation animation =
AnimationUtils.loadAnimation(getApplicationContext()
                                                                    ,
R.anim.blink_animation);
                imageView.startAnimation(animation);
            }
        });

        // Set up click listener for rotate button to start rotate animation
        rotateBTN.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Load rotate animation and start it on the ImageView
                Animation animation =
AnimationUtils.loadAnimation(getApplicationContext()
                                                                    ,
R.anim.rotate_animation);
                imageView.startAnimation(animation);
            }
```

```
        });

        // Set up click listener for fade button to start fade animation
        fadeBTN.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Load fade animation and start it on the ImageView
                Animation animation =
AnimationUtils.loadAnimation(getApplicationContext()
                                                                    ,
R.anim.fade_animation);
                imageView.startAnimation(animation);
            }
        });

        // Set up click listener for move button to start move animation
        moveBTN.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Load move animation and start it on the ImageView
                Animation animation =
AnimationUtils.loadAnimation(getApplicationContext()
                                                                    ,
R.anim.move_animation);
                imageView.startAnimation(animation);
            }
        });

        // Set up click listener for slide button to start slide animation
        slideBTN.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Load slide animation and start it on the ImageView
                Animation animation =
AnimationUtils.loadAnimation(getApplicationContext()
                                                                    ,
R.anim.slide_animation);
                imageView.startAnimation(animation);
            }
        });

        // Set up click listener for zoom button to start zoom animation
        zoomBTN.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Load zoom animation and start it on the ImageView
                Animation animation =
AnimationUtils.loadAnimation(getApplicationContext()
```

```
                                                                    ,
R.anim.zoom_animation);
                imageView.startAnimation(animation);
            }
        });

        // Set up click listener for stop button to clear any ongoing animation
        stopBTN.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // Clear the animation on the ImageView
                imageView.clearAnimation();
            }
        });
    }
}
```

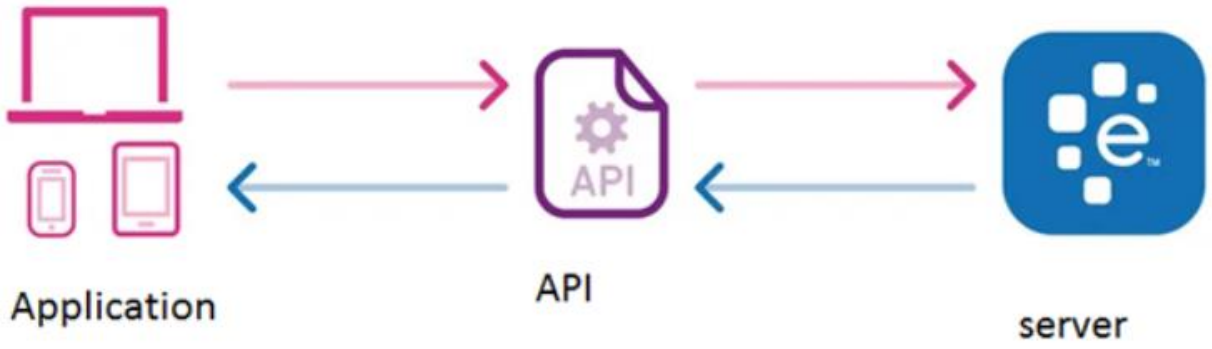
Note: *Drawables and strings can be found in the drawable folder and strings.xml file. Drawables can be found from the* ***app*** *>* ***res*** *>* ***drawable****.*

UNIT-3

Using Android APIs-

In android, API can be used to retrieve the data from the database. Data is received in the format of JSON and XML mainly. It can also be in the plain text file format. We use libraries which contain the predefined code and we have to just import the libraries to perform some tasks.

Android APIs of useful techniques or functions. API consists of libraries.API can be made of several libraries to complete an action. A single library in itself is not an
API.



What is Android API?

Application program interface (API) is a code for a programmer that they use in their applications. This code or (API) allows you to add specific functionalities to your application. In other words, we can say that APIs are the set of protocols and tools used for building software applications. Here is the list of various Android APIs level:

The codenames from Android 10 and above are the internal codenames and starting with Android 10, Google removed the usage of codenames publicly.

Android Version	API Level / SDK	Version Name	Version Code	Year
Android 13 (Beta)	Level 33	Tiramisu	TIRAMISU	
Android 12	Level 32 (Android 12L)	Snow Cone	S_V2	
	Level 31 (Android 12)		S	2021
Android 11	Level 30 (Android 11)	Red Velvet Cake	R	2020
Android 10	Level 29 (Android 10)	Quince Tart	Q	2019
Android 9	Level 28 (Android 9)	Pie	P	2018
Android 8	Level 27 (Android 8.1)	Oreo	O_MR1	2017
	Level 26 (Android 8.0)		O	
Android 7	Level 25 (Android 7.1)	Nougat	N_MR1	2016
	Level 24 (Android 7.0)		N	
Android 6	Level 23 (Android 6)	Marshmallow	M	2015
Android 5	Level 22 (Android 5.1)	Lollipop	LOLLIPOP_MR1	
	Level 21 (Android 5.0)		LOLLIPOP, L	2014
Android 4	Level 20 (Android 4.4w)	KitKat	KITKAT_WATCH	
	Level 19 (Android 4.4)		KITKAT	2013
	Level 18 (Android 4.3)	Jelly Bean	JELLYBEAN_MR2	
	Level 17 (Android 4.2)		JELLYBEAN_MR1	2012
	Level 16 (Android 4.1)		JELLYBEAN	
	Level 15 (Android 4.0.3 - 4.0.4)	Ice Cream Sandwich	ICE_CREAM_SANDWICH_MR1	2011
	Level 14 (Android 4.0.1 - 4.0.2)		ICE_CREAM_SANDWICH	
Android 3	Level 13 (Android 3.2)	Honeycomb	HONEYCOMB_MR2	

Use of Android APIs:

The android platform offers an API that Android applications use to communicate with the underlying Android system. Android framework API consists of the followings points: It is a core set of packages and classes. A collection of XML elements and attributes is declared in a manifest file.

Using Android Data and Storage APIs:

Data Storage in Android-

• SharedPreferences

• Internal storage (flash memory)

• External storage

• SQLite relational DB

• ContentProvider

• Network

1 SharedPreferences • Persistent way to store key/value pairs – Primitive types and Strings • Saved as XML in your application's folder in /data/data • Removed when the app is uninstalled • Can be used for general settings of the application – See PreferenceActivity.

SharedPreferences Privacy

• Can be obtained with different modes

– MODE_PRIVATE

– MODE_WORLD_READABLE

– MODE_WORLD_WRITABLE

• android:sharedUserId + MODE_PRIVATE

Working with SharedPreferences

• getSharedPreferences(String, int) or

getPreferences(int)

• To write values:

– obtain SharedPreferences.Editor by calling edit()

– write stuff with the editor using methods such as

putBoolean() and putString()

– apply the changes by calling commit() to your editor

• To read values:

– SharedPreferences.getBoolean(), getString(), etc.

2 Internal Storage-

• Data saved on the internal storage of the device is located in your application's folder in /data/data

• Like SharedPreferences, these files are removed,when the app is uninstalled

• YAFFS (Yet Another Flash File System)

– read (very fast)

– write (not very fast)

– erase (very slow)

3 External Storage-

• First, external storage is not always SD Card

– Samsung Galaxy Tab has both internal_sd and external_sd

• If you rely on external storage:

– always start with a check to getExternalStorageState()

– listen for broadcasts, regarding the state of the external storage (ACTION_MEDIA_EJECT, ACTION_MEDIA_REMOVED, ACTION_MEDIA_UNMOUNTED, ACTION_MEDIA_BAD_REMOVAL,etc.)

• Who can access the files on the external storage?

• For Android 2.1-update1 (API Level 7) or below, getExternalStorageDirectory() and the standard Java approach for creating and managing files

• Media scanner still recognizes specific folder names,but you have to create them manually

• ".nomedia" empty file - include in your folder, if you want the scanner to skip it

– if you have the Android source, look at /external/opencore/mediascanner.cpp

4 SQLite-

• What is SQLite?

– Embedded RDBMS in 275 KB

– public domain (whether this classifies as opensource is still an open debate)

• Why SQLite?

– Android has full support for SQLite databases

– Lightweight, no separate process

– Very popular (iPhone, Skype, etc.)

• **To create and use SQLite database, use SQLiteOpenHelper**

– use getReadableDatabase() or getWritableDatabase()

– onCreate() of the helper is called (provide SQL CREATE statement here)

– use some of the query() methods of SQLiteDatabase

– Cursor is returned as a result of the query

• Databases are saved in your application's folder in /data/data

• The sqlite3 tool is available for examining the contents of a table (.dump),

the initial SQL CREATE statement (.schema) or executing queries

dynamically (directly)

5 ContentProvider • One of the fundamental components of Android applications

• Encapsulates data and provides common interface for it, independent from the implementation details

• The primary use of most ContentProviders is sharing data between multiple applications

• Generally, its interface is used via ContentResolver objects – Usually you access the same ContentProvider via different ContentResolvers from the different apps

ContentProvider and REST-

• Relation between REST HTTP methods and methods for using ContentProvider data

– query() == GET

– insert() == POST

– update() == PUT

– delete() == DELETE

Managing data Using SQLite-

•SQL- Structured Query Language (SQL): a language for searching and updating a database – a standard syntax that is used by all database software (with minor incompatibilities) – generally case-insensitive • a declarative language: describes what data you are seeking, not exactly how to find it

Basic SQL operations

• SELECT

• INSERT

• UPDATE

• DELETE

1 SELECT •

SELECT FROM

WHERE [ORDER BY [ASC or DESC]] [LIMIT];

• SELECT _id, _data FROM files • SELECT * FROM files; (* means all columns)

• ORDER BY: sort the result by a column

• LIMIT: only get the first n rows in the result

2 INSERT •

INSERT INTO

() VALUES ();

• e.g., INSERT INTO files (data, size, title) VALUES

Insert data into the database by passing a ContentValues object to the insert() method:

KotlinJava

```
// Gets the data repository in write mode
val db = dbHelper.writableDatabase

// Create a new map of values, where column names are the keys
val values = ContentValues().apply {
    put(FeedEntry.COLUMN_NAME_TITLE, title)
    put(FeedEntry.COLUMN_NAME_SUBTITLE, subtitle)
}

// Insert the new row, returning the primary key value of the new row
val newRowId = db?.insert(FeedEntry.TABLE_NAME, null, values)
```

The first argument for insert() is simply the table name.

The second argument tells the framework what to do in the event that the ContentValues is empty (i.e., you did not put any values). If you specify the name of a column, the framework inserts a row and sets the value of that column to null. If you specify null, like in this code sample, the framework does not insert a row when there are no values.

The insert() methods returns the ID for the newly created row, or it will return -1 if there was an error inserting the data. This can happen if you have a conflict with pre-existing data in the database.

3 UPDATE

- **e.g., UPDATE files SET title="profile" WHERE _id=5;**

When you need to modify a subset of your database values, use the update() method.

Updating the table combines the ContentValues syntax of insert() with the WHERE syntax of delete().

```
val db = dbHelper.writableDatabase

// New value for one column
val title = "MyNewTitle"
val values = ContentValues().apply {
    put(FeedEntry.COLUMN_NAME_TITLE, title)
}

// Which row to update, based on the title
val selection = "${FeedEntry.COLUMN_NAME_TITLE} LIKE ?"
val selectionArgs = arrayOf("MyOldTitle")
val count = db.update(
        FeedEntry.TABLE_NAME,
        values,
        selection
```

selectionArgs)

The return value of the update() method is the number of rows affected in the database

4 Delete information from a database

To delete rows from a table, you need to provide selection criteria that identify the rows to the delete() method. The mechanism works the same as the selection arguments to the query() method. It divides the selection specification into a selection clause and selection arguments. The clause defines the columns to look at, and also allows you to combine column tests. The arguments are values to test against that are bound into the clause. Because the result isn't handled the same as a regular SQL statement, it is immune to SQL injection.

Kotlin Java

```
// Define 'where' part of query.
val selection = "${FeedEntry.COLUMN_NAME_TITLE} LIKE ?"
// Specify arguments in placeholder order.
val selectionArgs = arrayOf("MyTitle")
// Issue SQL statement.
val deletedRows = db.delete(FeedEntry.TABLE_NAME, selection, selectionArgs)
```

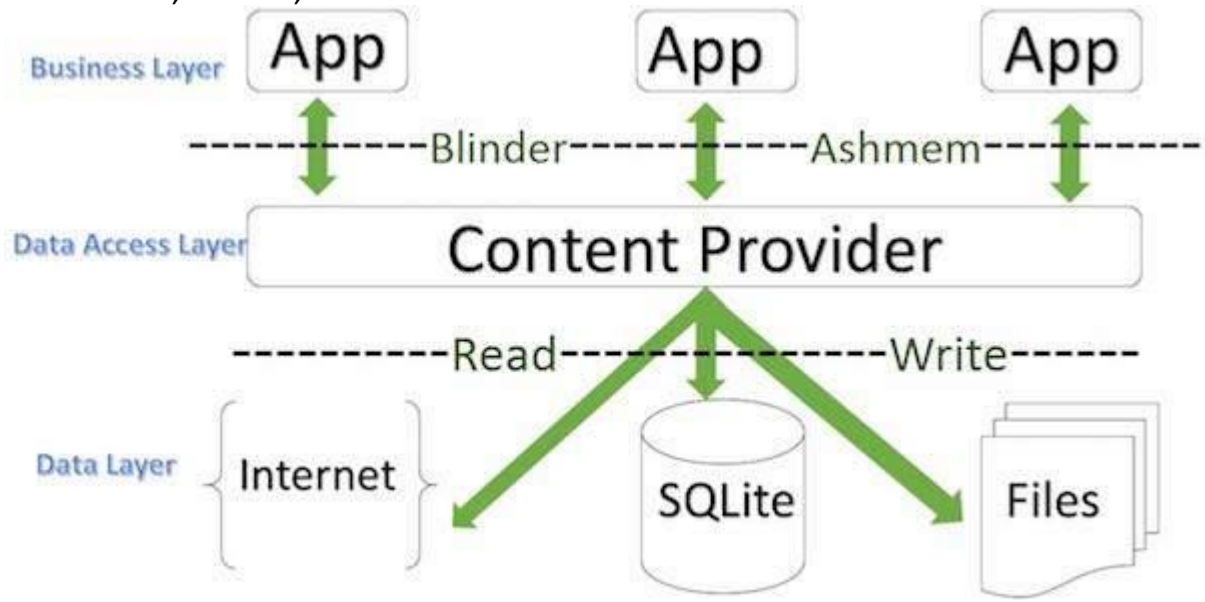
The return value for the delete() method indicates the number of rows that were deleted from the database.

.Share data between applications using Content Providers:

Defination of sharing data: Sharing data between Android apps is essential for enhancing functionality. Content Providers offer a secure way to achieve this. They act as intermediaries, letting one app share data with others through defined URIs. Data can be stored in databases or files, and apps can access, query, and modify it using Content Resolvers. Security is ensured through permissions and URIs, allowing authorized access. Content Providers enable data synchronization and collaboration among apps, fostering integrated and secure app ecosystems.

Content provider:Implementing Content Providers can be difficult because of very less information is available. *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.



ContentProvider

sometimes it is required to share data across applications. This is where content providers become very useful.

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.

```
public class My Application extends  ContentProvider {
}
```

Content Provider:

Content provider is a set of data wrapped up in a custom API to read and write. It acts as an interface that allows you to store and retrieve data from a data source. And also allows you to share an app's data

with other apps. Content providers decouple the app layer from the data layer by abstracting the underlying data source, thereby making apps data-source independent. They allow full permission control by monitoring which app components or users should have access to the data making data sharing easy. As a result, any app with appropriate permissions can add, remove, update, and retrieve data of another app including the data in some native Android databases.

There are 2 types of Content Providers :

Custom content providers: These are created by the developer to suit the requirements of an app.

Native content providers: These provide access to built-in databases, such as contact manager, media player, and other native databases. You need to grant the required permissions to your app before using native content providers.

Eg : The WhatsApp app sharing mobile contacts data.

Media store-Allows other applications to access, store media files.

Above diagram shows how content provider works. Application 2 stores its data in its own database and provides a provider. Application 1 communicates with the provider to access Application 2's data.

Content Provider uses a special URI starting with ***content://*** will be assigned to each content providers and that will be recognized across applications.

Creating a Content Provider :

To create a content provider we need toCreate sub class for ContentProvider.

1. Define content URI
2. Implement all the unimplemented methods. *insert()*, *update()*, *query()*, *delete()*, *getType()*.
3. Declare the content provider in AndroidManifest.xml

Specifying the URI of a Content Provide

Content provider URI consists of four parts.
content://authority/path/id

Each content provider exposes a URI that uniquely identifies the content provider's data set. If a content provider controls multiple

datasets, a separate URI needs to be specified for each dataset. The URI of a content provider begins with the string, **content://**.

What is a Content Resolver?

To get data and interact with a content provider, an app uses a ContentResolver to send requests to the content provider.

The ContentResolver object provides query(), insert(), update(), and delete() methods for accessing data from a content provider.

Each request consists of a URI and a SQL-like query, and the response is a Cursor object.

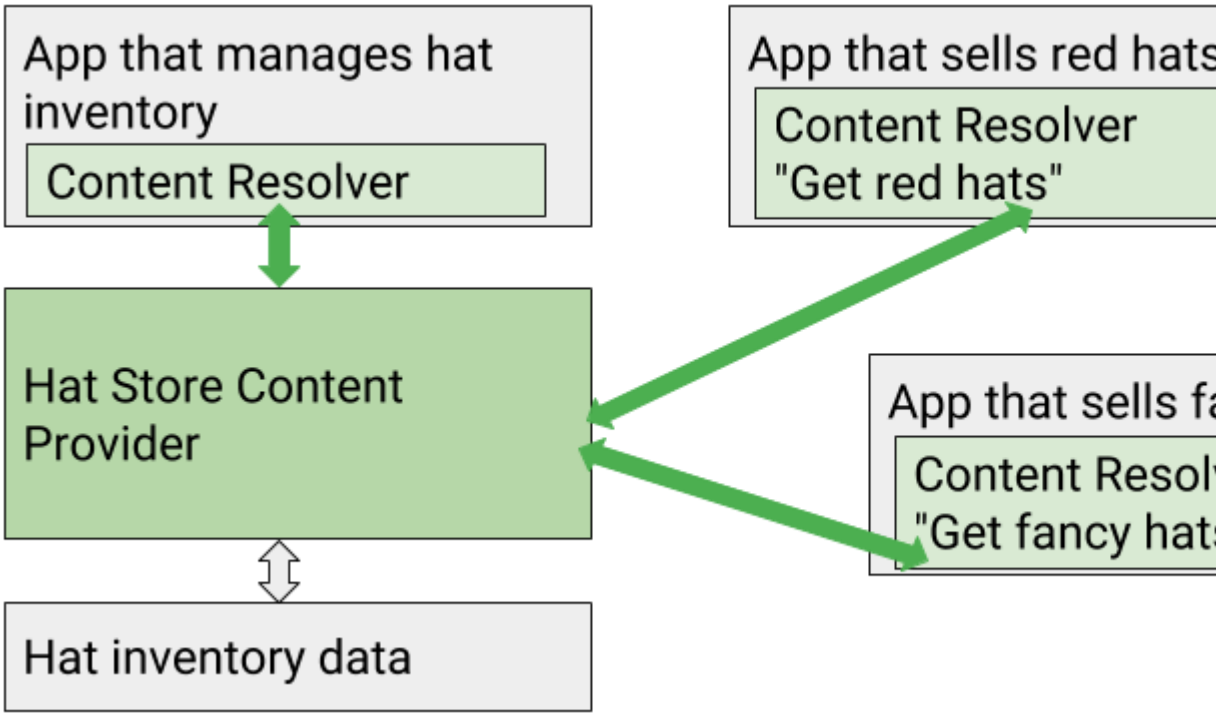
Note: You learned about cursors in the Data Storage chapters, and there is a recap later in this chapter.

The following diagram shows the query flow from an activity using a content resolver to the content provider to data in a SQL database, and back. Note that storing the data in a SQLite database is common, but not required.

Example of an app sharing data using a Content Provider

Consider an app that keeps an inventory of hats and makes it available to other apps that want to sell hats. The app that owns the data manages the inventory, but does not have a customer-facing interface. Two apps, one that sells red hats and one that sells fancy hats, access the inventory repository, and each fetch data

relevant for their shopping apps.



What Content Providers are good for

Content providers are useful for apps that want to make data available to other apps.

With a content provider, you can allow multiple other apps to securely access, use, and modify a single data source that your app provides. Examples: Warehouse inventory for retail stores, game scores, or a collection of physics problems for colleges.
For access control, you can specify levels of permissions for your content provider, specifying how other apps can access the data. For example, stores may not be allowed to change the warehouse inventory data.
You can store data independently from the app, because the content provider sits between your user interface and your stored data. You can change how the data is stored without needing to change the user-facing code. For example, you can build a prototype of your shopping app using mock inventory data, then later replace it with an SQL database for the real data. You could even store some of your data in the cloud and some locally, and it would be all the same to your users.
Another benefit of separating data from the user interface with a content provider is that development teams can work independently on the user interface and data repository of your app. For larger, complex apps it is very common that the user interface and the data backend are developed by different teams, and they can even be separate apps; that is, it is not required that the app with the content provider have a user interface. For example, your inventory app could consist only of the data and the content provider.

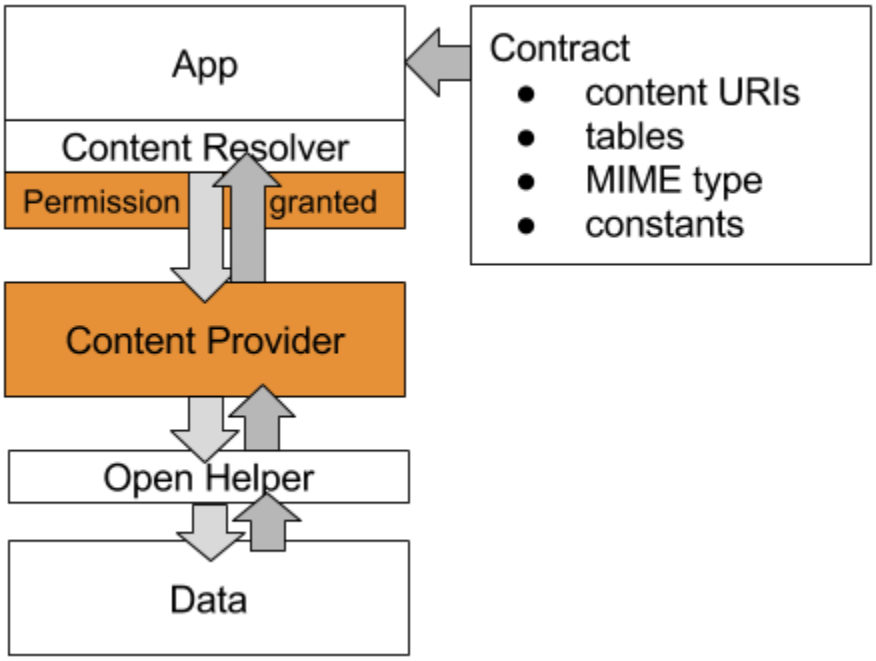
There are other classes that expect to interact with a content provider. For example, you must have a content provider to use a loader, such as CursorLoader, to load data in the background. You will learn about loaders in the next chapter.

Note: If your app is the only one using the data, and you are developing all of it by yourself, you probably don't need a content provider.

App Architecture with a Content Provider

Architecturally, the content provider is a layer between the content-providing app's data storage backend and the rest of the app, separating the data and the interface.

To give you a picture of the whole content provider architecture, this section shows and summarizes all the parts of the implemented content provider architecture, as shown in the following diagram. Each part will be discussed in detail next.



Data and Open Helper: The data repository. The data could be in a database, a file, on the internet, generated dynamically, or even a mix of these. For example, if you had a dictionary app, the base dictionary could be stored in a SQLite database on the user's device. If a definition is not in the database, it could get fetched from the internet, and if that fails, too, the app could ask the user to provide a definition or check their spelling.

Data used with content providers is commonly stored in SQLite databases, and the content provider API mirrors this assumption.

Contract: The contract is a public class that exposes important information about the content provider to other apps. This usually includes the URI schemes,

important constants, and the structure of the data that will be returned. For example, for the hat inventory app, the contract could expose the names of the columns that contain the price and name of a product, and the URI for retrieving an inventory item by part number.

Content Provider: The content provider extends the ContentProvider class and provides query(), insert(), update(), and delete() methods for accessing the data. In addition, it provides a public and secure interface to the data, so that other apps can access the data with the appropriate permissions. For example, to get inventory information from your app's database, the retail hat app would connect to the content provider, not to the database directly, as that is not permitted.

The app that owns the data specifies what permissions other apps need to have to work with the content provider. For example, if you have an app that provides inventory to retail stores, your app owns the data and determines the access permissions of other apps to the data. Permissions are specified in the Android Manifest.

Content Resolver: Content providers are always accessed through a content resolver. Think of the content resolver as a helper class that manages all the details of connecting to a content provider for you. Mirroring the content provider's API, the ContentResolver object provides you with query(), insert(), update(), and delete() methods for accessing data of a content provider. For example, to get get all the inventory items that are red hats, a hat store app would build a query for red hats, and use a content resolver to send that query to the content provider.

Implementing a Content Provider

Referring to the previous diagram, to implement a content provider you need:

Data, for example, in a database.
A way for accessing the data storage, for example, through an open helper for a database.
A declaration of your content provider in the Android Manifest to make it available to your own and other apps.
The subclass of ContentProvider that implements the query(), insert(), delete(), update(), count(), and getType() methods.
Public contract class that exposes the URI scheme, table names, MIME type, and important constants to other classes and apps. While this is not mandatory, without it, other apps cannot know how to access your content provider.
A content resolver to access the content provider using the appropriate methods and queries.

Let's take a look at each of these components.

Data

The data is often stored in a SQLite database, but this is not mandatory. Data could be stored in a file or file system, on the internet, or created dynamically. Or even a mix of these options. To the app, the content resolver the fetched data always as a Cursor object, as if it all came from the same source and in the same format.

The content provider might access the data directly in the case of files, or it might do so through a helper class. For example, apps typically use an open helper to interact with a SQLite database, and the content provider interacts with the open helper to get the data.

Typically, the data is presented to the content provider by the data store as tables, similar to database tables, where each row represents one entry, and each column represents an attribute for that entry. For example, each row contains one contact, and may have columns for email addresses and phone numbers. The structure of the tables is exposed in the contract.

Note: If you are just working with files, you can use the predefined FileProvider class.

Contract

The contract is a public class that exposes important information about an app's content provider so that other apps know how to access and use the content provider.

Using a contract separates public from private app information, design from implementation, and gives other apps one place to get all the information they need to work with a content provider. While the underlying app may change, a contract defines an API that ideally does not change once the app is published.

The contract for a content provider typically includes:

Content URI and URI scheme. The URI scheme shows how to build URIs to access the content provider's data. It's the API for the data.
Table constants. Makes table and column names available as constants, because they are needed to extract data from the returned cursor object.
MIME types, which have information on the data format, so that the app can appropriately process returned data. For example, that data could be encoded in JSON or HTML, or use a custom format.
Other shared constants that make it more convenient for an app to use the content provider.

Note: Contracts are not limited to content providers. You can use a contract anytime you want to share constants across classes of your app or make information about your app available to other apps.

URI Scheme & Content URI

Apps send requests to the content provider using Uniform Resource Identifiers or URIs.

A content URI for content providers has this general form:

```
scheme://authority/path/ID
```

scheme is always **content://** for content URIs.
authority represents the domain, and for content providers customarily ends in `.provider`
path is the path to the data.
ID uniquely identifies the data set to search.

For example, the following URI could be used to request all the entries in the "words" table:

```
content://com.android.example.wordcontentprovider.provider/words
```

The URI scheme for a content provider is defined in the Contract so that it is available to any app that wants to query this content provider. Customarily, this is done defining constants for AUTHORITY, CONTENT_PATH, and CONTENT_URI.

AUTHORITY. Represents the domain. For content providers this includes the unique package name and ends in `.provider`
CONTENT_PATH. The content path is an abstract semantic identifier of the data you are interested in. It does not predict or presume in what form the data is stored or organized in the background. As such, the path could resolve in the name of a table, the name of a file, or the name of the list.
CONTENT_URI. This is a content:// style URI to one set of data. If you have multiple "data containers" in the backend, you would create a content URI for each. For example, you would create a content URI for each table that can be queried. Use the Uri helper class for building and manipulating URIs.

In code, this might look as follows:

```
public static final String AUTHORITY =
"com.android.example.minimalistcontentprovider.provider";

public static final String CONTENT_PATH =  "words";
public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY +
"/" + CONTENT_PATH);
```

Tables in the Contract

A common way of organizing a contract class is to put definitions that are global to your database into the root level of the class. Usually, this is the name of the database.

Create a static abstract inner class for each table with the column names. This inner class commonly implements the BaseColumns interface. By implementing the BaseColumns interface, your class can inherit a primary key field called _ID that some Android classes, such as cursor adapters, expect to exist. This is not required, but can help your database work harmoniously with the Android framework.

Your code in the contract might look like this:

```
public static final String DATABASE_NAME = "wordlist";

public static abstract class WordList implements BaseColumns {
    public static final String WORD_LIST_TABLE = "word_entries";
    // Column names...
    public static final String KEY_ID = "_id";
    public static final String KEY_WORD = "word"
}
```

Methods: insert, delete, update, query

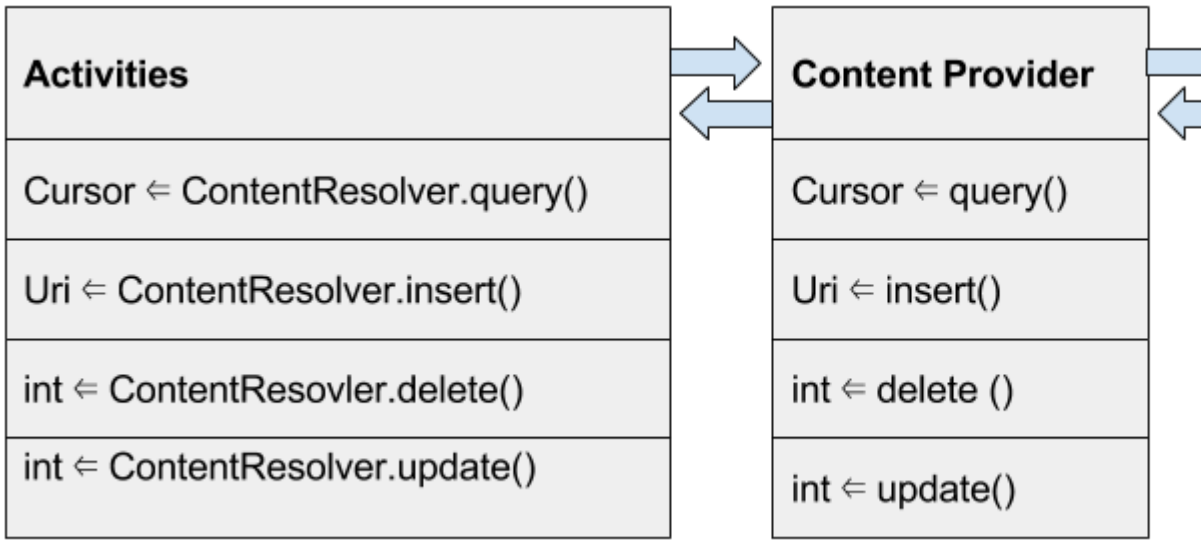
The content resolver makes available query, insert, delete, and update methods, and the content provider implements those same methods to access the data.

As such, it is a good practice to keep the names, method arguments, and return values consistent between all components. This makes implementation and maintenance a lot easier.

The following diagram shows the APIs between the conceptual building blocks of an app that uses a content provider to access data. Note in particular:

The methods are named the same and return the same data type throughout the stack (except for insert()).
Because it is common for a content provider to connect to a database, the query method returns a cursor. If your backend is not a database, the content provider must do the work of converting the returned data into the cursor format.

This diagram does not show additional helper classes, such as open helpers for databases, that may also use the same API convention.



When you create a content provider by extending the ContentProvider class, you need to implement the insert, delete, update, and query methods. If you follow the principle of making the method signatures the same across components, passing data back and forth does not require a lot of code.

Here is are sample methods in a content provider. Note that the content provider receives the values to insert in the correct type, as ContentValues, calls the database, and builds and returns the required Uri for the content resolver.

Insert method

```
/**
* Inserts one row.
*
* @param uri Uri for insertion.
* @param values Container for Column/Row key/value pairs.
* @return URI for the newly created entry.
*/
@Override
public Uri insert(Uri uri, ContentValues values) {
   long id = mDB.insert(values);
   return Uri.parse(CONTENT_URI + "/" + id);
}
```

Delete method

```
/**
* Deletes records(s) specified by selectionArgs.
```

```
*
* @param uri URI for deletion.
* @param selection Where clause.
* @param selectionArgs Where clause arguments.
* @return Number of records affected.
*/
@Override
public int delete(Uri uri, String selection, String[] selectionArgs) {
   return mDB.delete(parseInt(selectionArgs[0]));
}
```

Update method

```
/**
* Updates records(s) specified by selection/selectionArgs combo.
*
* @param uri URI for update.
* @param values Container for Column/Row key/value pairs.
* @param selection Where clause.
* @param selectionArgs Where clause arguments.
* @return Number of records affected.
*/
@Override
public int update(Uri uri, ContentValues values, String selection, String[]
selectionArgs) {
   return mDB.update(parseInt(selectionArgs[0]), values.getAsString("word"));
}
```

query() method

The query method in the content provider has the following signature:

```
public Cursor query(Uri uri, String[] projection, String selection,String[]
selectionArgs, String sortOrder){}
```

The arguments represent the parts of an SQL query and are discussed below. The query method of the content provider must parse the URI argument and determine the appropriate action.

URI Matching

It is a good practice to use an instance of the UriMatcher class to match the URIs. UriMatcher is a helper class for matching URIs for content providers.

Create a new UriMatcher in your content provider.

```
private static UriMatcher sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
```

In onCreate() add the URIs to match the the matcher. These are the content URIs defined in the Contract. You may want to do this in a separate method,

initializeUriMatching() that you call in onCreate(). The example code includes URIs requesting all items, one item by ID, and the item count.

```
/**
* Defines the accepted Uri schemes for this content provider.
* Calls addURI() for all of the content URI patterns that the provider should
recognize.
*/
private void initializeUriMatching() {

   // Matches a URI that is just the authority + the path,
   // triggering the return of all words.
   sUriMatcher.addURI(AUTHORITY, CONTENT_PATH, URI_ALL_ITEMS_CODE);

   // Matches a URI that references one word in the list by its index.
   sUriMatcher.addURI(AUTHORITY, CONTENT_PATH + "/#", URI_ONE_ITEM_CODE);

   // Matches a URI that returns the number of rows in the table.
   sUriMatcher.addURI(AUTHORITY, CONTENT_PATH + "/" + COUNT, URI_COUNT_CODE);
}
```

The query method switches on the matching URI to query the database for all items, one item, or an item count, as shown in this example code.

```
@Override
public Cursor query(Uri uri, String[] projection, String selection, String[]
selectionArgs,
                    String sortOrder) {

   Cursor cursor = null;

   // Determine integer code from the URI matcher and switch on it.
   switch (sUriMatcher.match(uri)) {
       case URI_ALL_ITEMS_CODE:
           cursor =  mDB.query(ALL_ITEMS);
           Log.d(TAG, "case all items " + cursor);
           break;
       case URI_ONE_ITEM_CODE:
           cursor =  mDB.query(parseInt(uri.getLastPathSegment()));
           Log.d(TAG, "case one item " + cursor);
           break;
       case URI_COUNT_CODE:
           cursor = mDB.count();
           Log.d(TAG, "case count " + cursor);
           break;
       case UriMatcher.NO_MATCH:
           // You should do some error handling here.
           Log.d(TAG, "NO MATCH FOR THIS URI IN SCHEME: " + uri);
           break;
       default:
           // You should do some error handling here.
           Log.d(TAG, "INVALID URI - URI NOT RECOGNIZED: "  + uri);
   }
   return cursor;
```

```
}
```

Using a Content Resolver

The ContentResolver object provides methods to query(), insert(), delete(), and update() data. Thus, the content resolver mirrors the content providers API and manages all interaction with the content provider for you. In most situations, you can use the default content resolver provided by the Android system.

Cursors

The content provider always presents the query results as a Cursor in a table format that resembles of a SQL database. This is independent of how the data is actually stored.

A cursor is a pointer into a row of structured data. You can think of it as a linked list of rows. The Cursor class provides methods for moving the cursor through that structure, and methods to get the data from the columns of each row.

When a method returns a cursor, you iterate over the result, extract the data, do something with the data, and finally close the cursor to release the memory.

If you use a SQL database, as shown above, you can implement your open helper to return a cursor, and then again, the content provider returns a cursor via the content resolver. If your data storage returns data in a different format, you will have to convert it into a cursor, usually a MatrixCursor.

The query() method

To make a query to the content provider:

Create an SQL-style query.
Use a content resolver to interact with the content provider to execute the query and return a Cursor.
Process the results in the Cursor.

The query method has the following signature:

```
public Cursor query(Uri uri, String[] projection, String selection,String[]
selectionArgs, String sortOrder){}
```

The arguments to this method represent the parts of a SQL query. Even if you are using another kind of backend, you must still accept a query in this style and handle the arguments appropriately.

uri	The complete content URI queried. This cannot be null. You get the information the correct URI from the contract. For example: `String queryUri = Contract.CONTENT_URI.toString();`
projection	A string array with the names of the columns to return for each row. Setting this null returns all columns. For example: `String[] projection = new String[] {Contract.CONTENT_PATH};`
selection	Indicates which rows/records of the objects you want to access. This is a WHEF clause excluding the actual where. For example: `String where = KEY_WORD + " LIKE ?";`
selectionArgs	Argument values for the selection criteria. If you include ?s in selection, they are replaced by values from selectionArgs, in the order that they appear. IMPORTA is a best security practice to always separate selection and selectionArgs. For example: `String[]whereArgs = new String[]{searchString};`
sortOrder	The order in which to sort the results. Formatted as an SQL ORDER BY clause (excluding the ORDER BY keyword). Usually ASC or DESC; null requests the d sort order, which could be unordered.

And you make a query to to the content provider like this:

```
Cursor cursor = getContentResolver().query(Uri.parse(queryUri), projection,
selectionClause, selectionArgs, sortOrder);
```

Note: The insert, delete, and update methods are provided for convenience and clarity. Technically, the query method could handle all requests, including those to insert, delete, and update data.

UNIT 4

Using Android APIs-

1 Using Android Netwoking APIs:

Networking in Android involves connecting apps to the internet or devices, enabling data exchange, file downloading, and server communication. It supports APIs like HTTP, TCP/IP, Bluetooth, and Wi-Fi. Understanding networking fundamentals is crucial, with popular libraries like OkHttp, Retrofit, and Volley. Network connectivity changes can be managed using BroadcastReceiver, and security measures like SSL/TLS protect data. Best practices include caching, offline support, error handling, secure protocols, background threads, and security measures.

1 Networking Fundamentals

Networking fundamentals refer to the basic principles and concepts that underlie computer networking. These include protocols, architectures, devices, and services that enable communication between computers and other devices. Understanding networking fundamentals is essential for designing, implementing, and troubleshooting network systems.

2 Client-Server Architecture

Client-server architecture is a computing model that divides the processing of an application into two parts: client and server. The client is a device or software that requests and receives services from the server, which is a computer or software that provides services to the client. This architecture enables multiple clients to access and share resources provided by a single server, making it a popular model for many types of applications, including web, email, and database server

3 HTTP and HTTPS Protocols

HTTP (Hypertext Transfer Protocol) and HTTPS (Hypertext Transfer Protocol Secure) are two protocols for transmitting data over the internet. HTTP is a standard protocol used for transmitting data between a client (such as a web browser) and a server. HTTPS is a secure version of HTTP that uses encryption to protect data transmitted between a client and a server. HTTPS is commonly

used for transmitting sensitive data, such as passwords, credit card information, and personal information, over the internet.

Making HTTP Requests with HttpURLConnection:

HttpURLConnection is a class in the Java.net package that provides an API for making HTTP requests in Android. You create a URL object with the URL of the resource you want to access, then use methods such as setRequestMethod() and getResponseCode() to configure the request and get the response from the server.

HttpURLConnection for GET and POST Requests:

Here's an example of making a GET request using HttpURLConnection:

```
URL url = new URL("https://www.example.com/api/data");
HttpURLConnection con1 = (HttpURLConnection) url.openConnection();
con1.setRequestMethod("GET");
int responseCode = con1.getResponseCode();
```

Here's an example of making a POST request using HttpURLConnection:

```
URL url = new URL("https://www.example.com/api/data");
HttpURLConnection con1 = (HttpURLConnection) url.openConnection();
con1.setRequestMethod("POST");
con1.setDoOutput(true);
DataOutputStream out = new DataOutputStream(con1.getOutputStream());
out.writeBytes("param1=value1&param2=value2");
out.flush();
out.close();
int responseCode = con1.getResponseCode();
```

The example involves sending a POST request with parameters "param1" and "param2" and their values "value1" and "value2". Enabling output, writing parameters to the DataOutputStream, flushing and closing the stream, and receiving the response code from the server.

Handling Response Codes and Parsing Server Responses

To handle response codes, you can use the getResponseCode() method of the HttpURLConnection class. This method returns the HTTP response code returned by the server. Here's an example:

```
URL url = new URL("https://www.example.com/api/data");

HttpURLConnection con1 = (HttpURLConnection) url.openConnection();

con1.setRequestMethod("GET");

int responseCode = con1.getResponseCode();

if (responseCode == HttpURLConnection.HTTP_OK) {

   // Handle successful response

} else {

   // Handle unsuccessful response

}
```

To parse server responses, you can use the getInputStream() method of the HttpURLConnection class to get an InputStream object for reading the response body. Then, using a BufferedReader, read the answer line by line. Here's an example:

```
URL url = new URL("https://www.example.com/api/data");

HttpURLConnection con1 = (HttpURLConnection) url.openConnection();

con1.setRequestMethod("GET");

int responseCode = con1.getResponseCode();

if (responseCode == HttpURLConnection.HTTP_OK) {
```

```
   BufferedReader in = new BufferedReader(new
InputStreamReader(con1.getInputStream()));
   String inputLine;
   StringBuffer response = new StringBuffer();
   while ((inputLine = in.readLine()) != null) {
      response.append(inputLine);
   }
   in.close();
   // Handle response string
} else {
   // Handle unsuccessful response
}
```

In this example, we're using a StringBuffer to accumulate the response data line by line. Once we've read the entire response, we can use the toString() method of the StringBuffer to get the response as a string, which we can then parse as needed.

Networking Libraries:

There are several networking libraries available for Android developers, including:

1. **OkHttp: A popular HTTP client library that supports HTTP/2 and SPDY protocols.**
2. **Retrofit: A type-safe REST client for Android that uses annotations to define API interfaces.**
3. **Volley: A library for fast, easy, and efficient network operations.**
4. **Android Asynchronous Http Client (AsyncHttpClient): A library for making asynchronous HTTP requests.**

5. **Loopj: A library that provides an easy-to-use interface for making asynchronous HTTP requests.**

Each library has its strengths and weaknesses, and the choice of library will depend on the specific requirements of your app.

Volley Library:

Volley is a library for making fast, efficient, and easy network requests in Android apps. It's developed and maintained by Google, and it's a popular choice for many Android developers Here's an example of how to use Volley to make a simple network request:

Conclusion:

Networking is crucial in Android app development, considering factors like security, performance, and user experience. Understanding protocols and client-server architecture is essential.

HTTP and HTTPS protocols are widely used for secure data transmission, with Java.net's HttpURLConnection class facilitating Android requests and handling response codes.

Android's built-in JSON library enables data processing, while networking libraries like OkHttp, Retrofit, and Volley handle network requests efficiently.

Registering a BroadcastReceiver streamlines network connectivity, protects user data, and optimizes performance and resource usage.

Android app networking efficiency ensured by caching, offline support, error handling, and secure protocols.

2 Using Android Web APIs:

What is Web API? API stands for Application Programming Interface. A Web API is an application programming interface for the Web. A Browser API can extend the functionality of a web browser. A Server API can extend the functionality of a web server.

What is Web API?

API stands for Application Programming Interface.

A Web API is an application programming interface for the Web.

A Browser API can extend the functionality of a web browser.

A Server API can extend the functionality of a web server.

Browser APIs:

All browsers have a set of built-in Web APIs to support complex operations, and to help accessing data.

For example, the Geolocation API can return the coordinates of where the browser is located.

Example

Get the latitude and longitude of the user's position:

```
const myElement = document.getElementById("demo");

function getLocation() {
  if (navigator.geolocation) {
    navigator.geolocation.getCurrentPosition(showPosition);
  } else {
    myElement.innerHTML = "Geolocation is not supported by this browser.";
  }
}

function showPosition(position) {
  myElement.innerHTML = "Latitude: " + position.coords.latitude +
  "<br>Longitude: " + position.coords.longitude;
}
```

What is Web API and why we use it ?

API stands for Application Programming Interface. API is actually some kind of interface which is having a set of functions. These set of functions will allow programmers to acquire some specific features or the data of an application.

- Web API is an API as the name suggests, it can be accessed over the web using the HTTP protocol. It is a framework that helps you to create and develop HTTP based RESTFUL services. The web API can be developed by using different technologies such as java, ASP.NET, etc. Web API is used in either a web server or a web browser. Basically Web API is a web development concept. It is limited to Web Application's client-side and also it does not include a web server or web browser details. If an

application is to be used on a distributed system and to provide services on different devices like laptops, mobiles, etc then web API services are used. Web API is the enhanced form of the web application.

ASP.NET Web API: ASP.NET stands for Active Server Pages.NET. It is mostly used for creating web pages and web technologies. It is considered a very important tool for developers to build dynamic web pages using languages like C# and Visual Basic. ASP.NET Web API is a framework that helps you to build services by making it easy to reach a wide range of clients including browsers, mobiles, tablets, etc. With the help of ASP.NET, you can use the same framework and same patterns for creating web pages and services both.

Where to use Web API?

1. Web APIs are very useful in implementation of RESTFUL web services using .NET framework.
2. Web API helps in enabling the development of HTTP services to reach out to client entities like browser, devices or tablets.
3. ASP.NET Web API can be used with MVC for any type of application.
4. A web API can help you develop ASP.NET application via AJAX.
5. Hence, web API makes it easier for the developers to build an ASP.NET application that is compatible with any browser and almost any device.

ASP.NET Web API: ASP.NET stands for Active Server Pages.NET. It is mostly used for creating web pages and web technologies. It is considered a very important tool for developers to build dynamic web pages using languages like C# and Visual Basic. ASP.NET Web API is a framework that helps you to build services by making it easy to reach a wide range of clients including browsers, mobiles, tablets, etc. With the help of ASP.NET, you can use the same framework and same patterns for creating web pages and services both.

Where to use Web API?

1. Web APIs are very useful in implementation of RESTFUL web services using .NET framework.
2. Web API helps in enabling the development of HTTP services to reach out to client entities like browser, devices or tablets.
3. ASP.NET Web API can be used with MVC for any type of application.
4. A web API can help you develop ASP.NET application via AJAX.
5. Hence, web API makes it easier for the developers to build an ASP.NET application that is compatible with any browser and almost any device.