Dr. MGR GOVT. ARTS AND SCIENCE COLLEGE FOR WOMEN., VILLUPURAM - 605401

STUDY METERIALS

BCS51 - MOBILE APPLICATIONS DEVELOPMENT

B.Sc., Computer Science
(III YEAR – V SEM)

SYLLABUS

BCS51 - MOBILE APPLICATIONS DEVELOPMENT

Objective:

This course aims to provide the students with a detailed knowledge on Mobile Application and Development and covers Android programming from fundamentals to building mobile applications for smart gadgets.

UNIT I Introduction to Mobile Applications:

Native and web applications - Mobile operating systems and applications - Mobile Databases. Android: History of Android - Android Features – OSS – OHA - Android Versions and compatibility - Android devices - Prerequisites to learn Android -– Setting up software – IDE - XML. Android Architecture: Android Stack - Linux Kernel - Android Runtime - Dalvik VM - Application Framework - Android emulator - Android applications.

UNIT II Android development:

Java - Android Studio – Eclipse – Virtualization – APIs and Android tools – Debugging with DDMS – Android File system – Working with emulator and smart devices - A Basic Android Application - Deployment. Android Activities: The Activity Lifecycle – Lifecycle methods – Creating Activity. Intents – Intent Filters – Activity stack.

UNIT III Android Services:

Simple services – Binding and Querying the service – Executing services.- Broadcast Receivers: Creating and managing receivers – Receiver intents – ordered broadcasts. Content Providers: Creating and using content providers – Content resolver. Working with databases: SQLite – coding for SQLite using Android – Sample database applications – Data analysis.

UNIT IV Android User Interface:

Android Layouts – Attributes – Layout styles - Linear – Relative – Table – Grid – Frame. Menus: Option menu – context menu - pop-up menu – Lists and Notifications: creation and display. Input Controls: Buttons-Text Fields-Checkboxes-alert dialogs- Spinners-rating bar-progress bar.

UNIT V Publishing and Internationalizing mobile applications :

Live mobile application development: Game, Clock, Calendar, Convertor, Phone book. App Deployment and Testing: Doodlz app – Tip calculator app – Weather viewer app.

Text Books

Barry Burd, "Android Application Development – All-in-one for Dummies", 2nd Edition, Wiley India, 2016.

Reference:

1. Paul Deitel, Harvey Deitel, Alexander Wald, " Android 6 for Programmers – An App-driven Approach", 3rd edition, Pearson education, 2016.
2. Jerome (J. F) DiMarzio, "Android – A Programmer‟s Guide", McGraw Hill Education, 8th reprint, 2015.
3. http://www.developer.android.com

MOBILE APPLICATIONS DEVELOPMENT

UNIT I

1.1 Native and web applications

1.1.1 Native App

A **Native App** is an app developed essentially for one particular mobile device and is installed directly onto the device itself. Users of native apps usually download them via app stores online or the app marketplace, such as the Apple App Store, the Google Play store and so on. An example of a native app is the Camera+ app for Apple's iOS devices. Native mobile apps provide fast performance and a high degree of reliability. They also have access to a phone's various devices, such as its camera and address book.

Advantages of native applications

- Access to built-in features of the device
- Native UI/UX
- Available from app stores
- SDK for developers

Disadvantages of native app development

- High price and long development time
- Complicated and expensive maintenance and support
- Content not seen by search engines
- Support of multiple versions of the application

Examples of native apps

- Native apps are a popular solution nowadays. They deliver an exceptional user experience and are perfect for solving complicated tasks. Really good examples of native apps include: Google Maps (for iOS and Android), Facebook (for iOS and Android) and LinkedIn (for iOS and Android).

1.1.2 Web App

A **Web App**, on the other hand, is basically Internet-enabled apps that are accessible via the mobile device's web browser. They need not be downloaded onto the user's mobile device in order to be accessed. The Safari browser is a good example of a mobile Web app.

Advantages of web applications

- Build the development team fast
- Support every device, every platform and every version of OS
- Fast deployment of new features

- No need to support multiple versions of the software
- No app store approval
- Visible to search engines

Disadvantages of web apps

- Internet connection required to function properly
- Not available in the app store
- Not appropriate for apps with a complex frontend
- Not native experience
- Limited access to smartphone's features
- Ad blockers

Examples of Web Apps

- Flipkart Lite, Medium, The Washington Post, Gmail, Google Docs.
- Progressive web apps are a breakthrough in modern web development. They allow building cross-platform applications without significant disadvantages for the end user.

1.1.3 The Difference between Native and Web Apps

In order to know which, type of app is better suited to our needs; we need to compare each one of them. Here is a quick comparison between native apps and web apps.

User Interface

From the point of the mobile device user, some native and web apps look and work much the same way, with very little difference between them. The choice between these two types of apps has to be made only when you have to decide whether to develop a user-centric app or an application-centric app. Some companies develop both native and web apps, so as to widen the reach of their apps, while also provide a good overall user experience.

Application Development Process

- The app development process of these two types of apps is what distinguishes them from each other. Each mobile platform that the native app is developed for, stipulates its own unique development process. In the case of web apps running on a mobile device's web browser, the problem that arises is that each of these mobile devices have unique features and come with their unique problems as well.
- Every mobile platform uses a different native programming language. While iOS uses Objective-C, Android uses Java, Windows Mobile uses C++ and so on. Web apps, on the other hand, use languages such as JavaScript, HTML 5, CSS3 or other web application frameworks as per the developer's preferences.

- Each mobile platform offers the developer its own standardized SDK, development tools and other user interface elements, which they can use to develop their native app with relative ease. In the case of web apps, though, there is no such standardization and the developer has no access to SDKs or tools of any sort.

Accessibility

A native app is totally compatible with the device's hardware and native features, such as an accelerometer, camera and so on. Web apps, on the other hand, can access only a limited amount of a device's native features.

While a native app works as a standalone entity, the problem is that the user has to keep downloading updates. A web app, on the other hand, updates itself without the need for user intervention. However, it necessarily needs to be accessed via a mobile device's browser.

Making Money on Apps

App monetization with native apps can be tricky, since certain mobile device manufacturers may lay restrictions on integrating services with certain mobile platforms and networks. Conversely, web apps enable you to monetize apps by way of advertisements, charging membership fees and so on. However, while the app store takes care of your revenue and commissions in the case of native app, you need to setup your own payment system in case of a web app.

Efficiency

Native apps are more expensive to develop. However, they are faster and more efficient, as they work in tandem with the mobile device they are developed for. Also, they are assured of quality, as users can access them only via app stores online.

Web apps may result in higher costs of maintenance across multiple mobile platforms. Also, there is no specific regulatory authority to control quality standards of these apps. The Apple App Store, though, features a listing of Apple's web apps.

1.2 Mobile operating systems and applications

A mobile operating system (Mobile OS) is a software platform on top of which other programs called application programs, can run on mobile devices such as personal digital assistant (PDA), tablets, cellular phones, smartphones and so on. Over the years, Mobile OS design has experienced a three-phase evolution: from the PC-based operating system to an embedded operating system to the

current smartphone-oriented operating system in the past decade. Throughout the process, Mobile OS architecture has gone from complex to simple to something in-between. The evolution process is naturally driven by the technology advancements in hardware, software, and the Internet.

1.2.1 Hardware

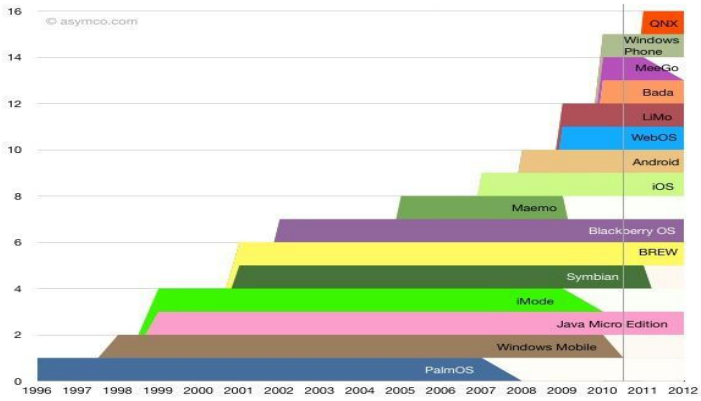
The industry has been reducing the factor size of microprocessors and peripherals to design actual mobile devices. Before the form factor size was reduced enough, the mobile device could not achieve both small size and processing capability at the same time. We had either a PC-sized laptop computer or a much weaker personal data assistant (PDA) in phone size. Mobile operating systems for PDAs usually did not have full multitasking or 3D graphics support. Features like sensors, such as accelerometers, and capacitor-based touch screens were not available in the past mobile operating systems.

1.2.2 Software

With a laptop computer, the software is mainly focused on the user's productivity, where support for keyboard and mouse that have precise inputs are essential. The software for a personal data assistant, as its name implies, helps the user to manage personal data such as contacts information, email, and so on. The mobile operating systems were not designed for good responsiveness or smoothness with a rich user interface (UI) including both touch screen and other sensors.

1.2.3 Internet

Along with Internet development, especially after Web 2.0, there is abundant information in the network waiting to be searched, organized, mined, and brought to users. People are increasingly living with the Internet instead of just browsing the Web. More and more people are involved in the development, including information contribution, application development, and social interactions. The mobile operating systems cannot be self-contained, but have to be open systems.



1.2.4 Mobile OS: Android

Android is an open source operating system for mobile devices developed by Google and the Open Handset Alliance. With 22,7% it is the second most used operating system for mobile devices worldwide behind Symbian. The system architecture consists of

• a modified Linux Kernel

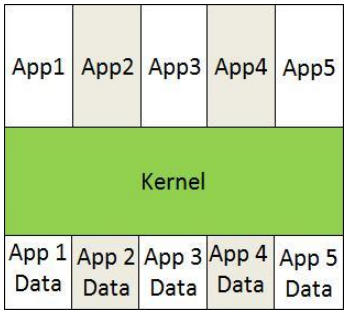
• open source Libraries coded in C and C++

• the Android Runtime, which considers core libraries that disposals the most core functions of Java. As virtual machine it uses Dalvin, which enables to execute Java applications.

• an Application Framework, which disposals services and libraries coded in Java for the application

• and the Applications, which operate on it

Android, iOS and Windows Phone use the same model of application sandboxing, which is shown in below Fig. Each application owns a unique identity and any data, process or permission belongs to it. For example, the data assigned to one application identity has no access to any other data of another applications identity. This sandboxing model will have discussed closer in the following by using the example of Android, to underlay the understatement for it.



Android, iOS, Windows Phone Sandboxing Model

1.2.5 Mobile OS: BlackBerry OS

Blackberry OS is the proprietary mobile operating system, developed by the Canadian company Research in Motion and is used for Blackberry devices only. Instead of all the other regarded mobile operating systems, it is mainly developed for business usage. Gartner says that it is one of the most popular mobile operating system today with 16,0% market share, but they also predict a decreasing relevance in the future.

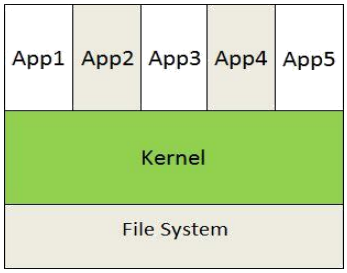


Fig. 4. BlackBerry OS Sandboxing Mode

BlackBerry OS uses an older model for application sandboxing, which can be seen in above Fig. It uses different trust roles for assignments and applications have full access to the complete device and

data. It is also requiring to sign an application via Certificate Authorities (CA) or generated (selfsigned) certificate to run code on the device.

Furthermore, the signature provides information about the privileges for an application, which is necessary because applications have full access on BlackBerry devices, because of its sandboxing model.

1.2.6 Mobile OS: iOS

The proprietary mobile operating system iOS is only used for Apple devices like the iPhone and is a further development of the operating system Mac OSX. Its market share grew continuously over the last year to 15,7

The system architecture is identical to the Mac OSX architecture and consists of the following components:

• Core OS: The kernel of the operating system

• Core Services: Fundamental system-services, which are subdivided in different frameworks and based on C and Objective C. For example, offers the CF Network Frame work the functionality to work with known network protocols.

• Media: Considers the high-level frameworks, which are responsible for using graphic-, audio- and video technologies.

• Coca Touch: Includes the UIKIT, which is an Objective based framework and provides a number of functionalities, which are necessary for the development of an iOS Application like the User Interface Management

Like in the Android section mentioned, iOS uses a similar sandboxing model.

Furthermore, applications must be signed with an issued certificate. This ensures that application have not been manipulated and ensures the runtime to check if an application has not become untrusted since it was last used. Uneven Android applications, iOS applications can only be signed with an official certification

1.2.7 Windows Phone

- Windows Phone is a successor of the operating system Windows Mobile of the software developer Microsoft. By comparison to the other discussed mobile operating systems, the market share is low with only 4,2%.
- Windows Phone uses technologies and tools, which are also used in the station based application development, like the development environment Visual Studio and the Frameworks Silverlight, XNA and .NET Compact. Furthermore, Windows Phone considers a complete integration with the Microsoft Services Windows Live, Zune, Xbox Live and Bing.
- For sandboxing Windows Phone uses the same model like Android and iOS. Furthermore 3rd party applications can only be signed with an official certification, like iOS Application.

1.2.8 Mobile Applications

The diagram shows four basic apps (App 1, App 2, App 3 and App 4), just to give the idea that there can be multiple apps sitting on top of Android.

- Mobile applications (also known as mobile apps) are software programs developed for mobile devices such as smartphones and tablets. They turn mobile devices into miniature powerhouses of function and fun.

- Some devices come preloaded with some mobile apps courtesy of their manufacturers or the mobile service providers with which they're associated (for example, Verizon, AT&T, T-Mobile, etc.), but many more apps are available through device-specific app stores.

- These apps are like any user interface you use on Android; for example, when you use a music player, the GUI on which there are buttons to play, pause, seek, etc is an application.
- Similarly, is an app for making calls, a camera app, and so on. All these apps are not necessarily from Google.

- Anyone can develop an app and make it available to everyone through Google Play Store. These apps are developed in Java, and are installed directly, without the need to integrate with Android OS.

1.3 Mobile Databases

1.3.1 What is Mobile Database?

A mobile database is a database that can be connected to by a mobile computing device over a mobile network. And it is portable and physically separate from the corporate database server.

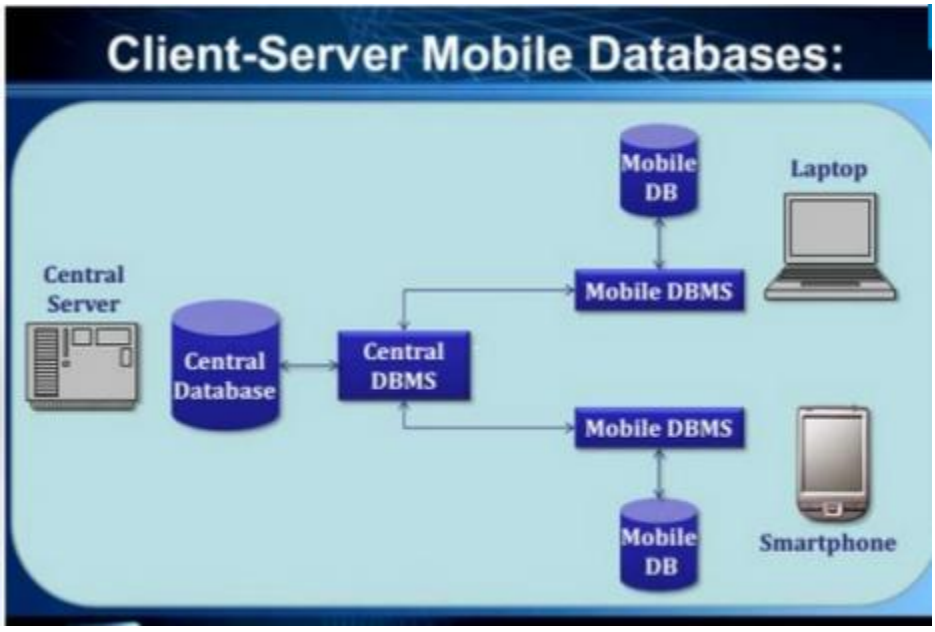
But mobile database is capable of communicating with that corporate database server from remote sites allowing the sharing of corporate database.

With mobile databases, users have access to corporate data on their laptop, PDA, or other Internet access device that is required for applications at remote sites.

1.3.1 The components of a mobile database environment

- **Corporate database server** and DBMS that deals with and stores the corporate data and provides corporate applications
- **Remote database** and DBMS usually manages and stores the mobile data and provides mobile applications
- **mobile database platform** that includes a laptop, PDA, or other Internet access devices
- Two-way communication links between the corporate and mobile DBMS.

Based on the particular necessities of mobile applications, in many of the cases, the user might use a mobile device may and log on to any corporate database server and work with data there, while in others the user may download data and work with it on a mobile device or upload data captured at the remote site to the corporate database.



The communication between the corporate and mobile databases is usually discontinuous and is typically established or gets its connection for a short duration of time at irregular intervals. Although unusual, some applications require direct communication between the mobile databases. The two main issues associated with mobile databases are the management of the mobile database and the communication between the mobile and corporate databases.

1.3.2 Features of mobile database

- Communicate with centralized database server through modes such as wireless or internet access
- Replicate data on centralized database server and mobile device
- Synchronize data on centralized database server and mobile device
- Capture data from various sources such as internet
- Manage/analyze data on the mobile device
- Create customized mobile applications

1.4 Android: History of Android

1.4.1 What is Android?

Android is a Linux based operating system it is designed primarily for touch screen mobile devices such as smart phones and tablet computers. Android is an operating system and programming platform developed by Google for Smartphone and other mobile devices (such as tablets). It can run on many different devices from many different manufacturers. Android includes a software development kit for writing original code and assembling software modules to create apps for Android users. It also provides a marketplace to distribute apps. Altogether, Android represents an ecosystem for mobile apps.

These applications are more comfortable and advanced for the users. The hardware that supports android software is based on ARM architecture platform. The android is an open source operating system means that it's free and any one can use it.

The android development supports with the full java programming language. Even other packages that are API and JSE are not supported. The first version 1.0 of android development kit (SDK) was released in 2008 and latest updated version is jelly bean.

1.4.2 History of Android

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

Initially developed by Android Inc., which Google bought in 2005, Android was unveiled in 2007, with the first commercial Android device launched in September 2008. The current stable version is Android 9 "Pie", released in August 2018. Google released the first beta of the next release, Android Q, on Pixel phones in March 2019. The core Android source code is known as Android Open Source Project (AOSP), which is primarily licensed under the Apache License.

Android is also associated with a suite of proprietary software developed by Google, called Google Mobile Services (GMS),[10] that frequently comes pre-installed on devices. This includes core apps such as Gmail, the application store/digital distribution platform Google Play and associated Google Play Services development platform, and usually includes the Google Chrome web browser and Google Search app. These apps are licensed by manufacturers of Android devices certified under standards imposed by Google, but AOSP has been used as the basis of competing Android ecosystems such as Amazon.com's Fire OS, which use their own equivalents to Google Mobile Services.

Android has been the best-selling OS worldwide on smartphones since 2011 and on tablets since 2013. As of May 2017, it has over two billion monthly active users, the largest installed base of any operating system, and as of December 2018, the Google Play store features over 2.6 million apps.

1.5 Features of Android

Android is a powerful operating system competing with Apple 4GS and supports great features. Few of them are listed below −

Sr.No.	Feature & Description
1	**Beautiful UI** Android OS basic screen provides a beautiful and intuitive user interface.
2	**Connectivity** GSM/EDGE, IDEN, CDMA, EV-DO, UMTS, Bluetooth, Wi-Fi, LTE, NFC and WiMAX.
3	**Storage** SQLite, a lightweight relational database, is used for data storage purposes.
4	**Media support** H.263, H.264, MPEG-4 SP, AMR, AMR-WB, AAC, HE-AAC, AAC 5.1, MP3, MIDI, Ogg Vorbis, WAV, JPEG, PNG, GIF, and BMP.
5	**Messaging** SMS and MMS
6	**Web browser** Based on the open-source WebKit layout engine, coupled with Chrome's V8 JavaScript engine supporting HTML5 and CSS3.
7	**Multi-touch** Android has native support for multi-touch which was initially made available in handsets such as the HTC Hero.
8	**Multi-tasking** User can jump from one task to another and same time various application can run simultaneously.
9	**Resizable widgets** Widgets are resizable, so users can expand them to show more content or shrink them to save space.
10	**Multi-Language** Supports single direction and bi-directional text.

11	**GCM** Google Cloud Messaging (GCM) is a service that lets developers send short message data to their users on Android devices, without needing a proprietary sync solution.
12	**Wi-Fi Direct** A technology that lets apps discover and pair directly, over a high-bandwidth peer-to-peer connection.
13	**Android Beam** A popular NFC-based technology that lets users instantly share, just by touching two NFC-enabled phones together.

Some other features includes

- Video calling
- Screen capture
- External storage
- Streaming media support
- Optimized graphics

1.6 OSS

OSS is IT for running a communication network. OSS is either Operational Support Systems or Operations Support Systems.

It is a set of programs that help a communications service provider for monitor, control, analyze, and manage a telephone or computer network.

OSS is software (Occasionally hardware) applications that support back-office activities which operate a telecommunication network, provision and maintain customer services

It is traditionally used by network planners, service designers, operations, architects, support, and engineering teams in the service provider. Increasingly product managers and senior staff under the CTO or COO may also use or rely on OSS to some extent.

OSS break down:

Operational/operations:

Relating to the day-to-day tasks of supplying and supporting communication services. Getting technical and infrastructure jobs done.

Running the network and services. As opposed to the business of selling, marketing or billing.

Support:

Enabling and improving the service provider's operational activities:

Automating operational tasks; executing them faster; making them consistent; and tracking progress/results.

Systems

One or more distinct software applications, that is responsible for doing specific OSS jobs, running on servers, or on devices installed in the network, or executed in the Cloud.

1.6.1 Functions of OSS

- Network managemet systems
- Service delivery
- Service fulfillment, including the network inventory, activation and provisioning
- Service assurance
- Customer Care

1.6.2 Advantages of OSS

- Support communications
- Support collaborations
- Control industrial process
- Savings through processes redesign
- Savings through processes integration
- Efficiently process business transactions
- Update corporate data base
- Centralized data repository for strategic decisions
- Competitive advantages through the ability of adapt business easier and faster.

1.7 OHA

The **Open Handset Alliance** (OHA) is a business alliance that was created for the purpose of developing open mobile device standards. The OHA has approximately 80 member companies, including HTC, Dell, Intel, Motorola, Qualcomm and Google. The OHA's main product is the Android platform - the world's most popular smartphone platform.

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 Platform.

OHA members are primarily mobile operators, handset manufacturers, software development firms, semiconductor companies and commercialization companies. Members share a commitment to expanding the commercial viability of open platform development.

OHA develop technologies that will significantly lower the cost of developing and distributing mobile devices and services.

OHA devoted to advancing open standards for mobile devices.

Each Android Device Manufacturer of OHA partners can customize Android to suit their needs.

OHA member companies back the open platform concept for a number of reasons, as follows:

- Lower overall handset costs: Opens up resources, which facilitates the focus on creating innovative applications, solutions and services.
- Developer-friendly environment: In the open-source community, developers share notes to expedite application development.
- Post-development: Provides an ideal channel for application marketing and distribution.

Following are some Open Handset Alliance Firms.

1.8 Android Versions

1- Android 1.0 and 1.1(Unnamed)

Both versions are first commercial versions. They are officially released publicly in 2008 and 2009. The first android commercial version was placed on HTC dream device. These versions were released without codename. The features in

Android 1.0 :

- Google Maps
- Camera
- Gmail, Contacts and Google Synchronization
- Web Browser
- Wireless supports – WiFi and Bluetooth

Android 1.1 :

- Add Save attachment in message
- Provides reviews and details when user search business on maps

2- Android 1.5 (CupCake)

Android 1.5 was released in April 2009. This is first released codename with official name **"CupCake"**. It brought features in UI design and update several new features are.

- New upload service on YouTube and Picasa like Uploading Videos and Photos.
- Supporting in MPEG-4, Video recording
- Improving Web Browser-Copy and Paste facility

3- Android 1.6(Donut)

The Android Version 1.6, Codename is **Donut**. It was released in Sept 2009. It including various features

- It supports large screen size
- Providing Gallery and Camera features.
- Improve speed in system apps

4- Android 2.0 and 2.1(Eclair)

The codename for Android 2.0 is **Eclair**. It was brought in Oct 2009 and 2.1 version released in Dec 2009. There features:

- Update UI
- Support Live Wallpaper
- Support Bluetooth 2.1
- Improve Google map
- Minor API Changes

5- Android 2.2(Froyo)

It was released in May 2010 with the codename is **Froyo**. There features :

- Support Animated GIF
- WiFi Support Hotspot functionality
- Speed improvements
- Upload file support in browser
- Support numeric and alphanumeric password

6- Android 2.3 and 2.4 (Gingerbread)

Gingerbread came out on the market in December 2010. It was officially announced in Nexus S android phone which is Google co-developed with Samsung.

- Improve Copy-Paste Facility
- Updated UI design
- Social Networking Supports
- Easy use of keyboard

7- Android 3.0, 3.1 and 3.2 (Honeycomb)

Android 3.0 was released in February 2011 after that quickly followed by 3.1 and 3.2 in July and August of 2011.

- Gmail App improvements
- Updated 3D UI
- Media Sync from SD Card
- Google eBooks
- Google Talk Video Chat
- Support Adobe Flash in Browser
- High-performance WiFi Connections and Lock
- Chinese handwriting

8-Android 4.0 (Ice-Cream Sandwich)

Ice-Cream Sandwich was released in October 2011. It was Google's attempt synthesize Honeycomb. There are some features

Improved text input and spelling check

- WiFi direct
- Photo Decor facility
- Improve in keyboard correction
- Face Lock
- Improve in video recording resolution
- Camera performance
- Up to 16 tabs in web browser

9- Android 4.1, 4.2 and 4.3 (Jelly Bean)

Android 4.1 came out in July 2012. Its codename is **Jelly Bean**. '**Google now**' is the main feature of Jelly Bean. It is used to whenever we want to search data from your google account and location data from your Android device to compile the information you need it. Many others features are

- Voice search
- Smooth UI
- Improve camera application
- Security enhancement
- Voice typing
- Multiple user accounts on tablet only
- 4k resolution support
- Supporting Bluetooth Low Energy
- Bi-directional text and other language support
- Support USB audio

- Lock screen improvement
- Set the volume of incoming calls ad showing message alert
- Native emoji support

10- Android 4.4 (KitKat)

Android 4.4 as a name of **KitKat** announced by Google in September 2013. Features are:

- Screen Recording
- KitKat adds a feature in 'Google now'. Its name is '**OK Google**'. "OK Google" allows access google now to the user without touching your mobile phone.
- GPS Support
- Offline music support
- UI updates for google map navigation and alarm.
- Also, introduce Emoji' to google keyboard.

11- Android 5.0 and 5.1 (Lollipop)

Android 5.0 is called **Lollipop**. It was released in November 2014. Support ART (Android RunTime)

- Save battery on some device
- Improvement in UI
- New material design
- Bug fixes
- Multiple sim card support
- High definition voice call

12- Android 6.0 (Marshmallow)

Marshmallow came out in May 2015. It's new features are

- Fingerprint authentication
- USB Type C support
- Save battery-'Sleep Mode'
- App permission model-OPT(send request for permission)
- New Emoji's

13- Android 7.0 (Nougat)

Android **Nougat** was released in August 2016. It was announced with native split-screen mode and data saver feature.

- Provide multitasking
- Providing multi-window mode
- Improve in storage manager
- Display touch improvement

14- Android 8.0 (Oreo)

After some of the version released in the market recently released Android 8.0 named is **Oreo** in August 2017. There some updated new features

- Support PIP(Picture-in-Picture)
- Multi-display support
- Google Play support
- Adaptive icons
- Improve notification system

1.8.1 Android compatibility

Android is designed to run on many different types of devices, from phones to tablets and televisions. As a developer, the range of devices provides a huge potential audience for your app. In order for your app to be successful on all these devices, it should tolerate some feature variability and provide a flexible user interface that adapts to different screen configurations.

Because Android is an open source project, any hardware manufacturer can build a device that runs the Android operating system. Yet, a **device is "Android compatible"** only if it can correctly run apps written for the *Android execution environment*. The exact details of the Android execution environment are defined by the Android compatibility program and each device must pass the Compatibility Test Suite (CTS) in order to be considered compatible.

A app developer, need not worry, whether a device is Android compatible, because only devices that are Android compatible include Google Play Store. So we can rest assured that users who install your app from Google Play Store are using an Android compatible device.

However, we do need to consider whether our app is compatible with each potential device configuration. Because Android runs on a wide range of device configurations, some features are not available on all devices. For example, some devices may not include a compass sensor. If your app's core functionality requires the use of a compass sensor, then our app is compatible only with devices that include a compass sensor.

Android's purpose is to establish an open platform for developers to build innovative apps.

- The Android Compatibility program defines technical details of the Android platform and provides tools for OEMs to ensure developer applications run on a variety of devices.
- The Android SDK provides built-in tools for developers to clearly state the device features required by their applications.
- Google Play shows applications only to those devices that can properly run those applications.

To build an Android-compatible mobile device, follow this three-step process:

- Obtain the Android software source code. This is the source code for the Android platform that you port to your hardware.
- Comply with the Android Compatibility Definition Document (CDD) (PDF, HTML). The CDD enumerates the software and hardware requirements of a compatible Android device.

- Pass the Compatibility Test Suite (CTS). Use the CTS as an ongoing aid to evaluate compatibility during the development process.

After complying with the CDD and passing the CTS, your device is Android compatible, meaning Android apps in the ecosystem provide a consistent experience when running on your device.

Just as each version of the Android platform exists in a separate branch in the source code tree, there is a separate CTS and CDD for each version as well. The CDD, CTS, and source code are — along with your hardware and your software customizations — everything you need to create a compatible device.

1.9 Android devices

An Android device is a device that runs on the Android operating system. Android is an array of software intended for mobile devices that features an operating system, core applications and middleware.

An Android device may be a smartphone, tablet PC, e-book reader or any type of mobile device that requires an OS.

Within a short period, the Android platform became so popular that it surpassed Windows Mobile and Symbian for a number of applications. Various mobile device manufacturers embraced the Android platform due to its overwhelming popularity. The reasons behind this success are as follows:

- Cutting-edge technology offered by Google
- Extremely user friendly platform
- Can be used in smartphones as well as tablets
- Any user can do modifications to the platform as the Android SDK is open to users
- Availability of huge volume of applications

1.10 Setting up software

1.10.1 Set-up Java Development Kit (JDK)

You can download the latest version of Java JDK from Oracle's Java site − Java SE Downloads. You will find instructions for installing JDK in downloaded files, follow the given instructions to install and configure the setup. Finally set PATH and JAVA_HOME environment variables to refer to the directory that contains **java** and **javac**, typically java_install_dir/bin and java_install_dir respectively.

If you are running Windows and installed the JDK in C:\jdk1.8.0_102, you would have to put the following line in your C:\autoexec.bat file.

```
set PATH=C:\jdk1.8.0_102\bin;%PATH%
set JAVA_HOME=C:\jdk1.8.0_102
```

Alternatively, you could also right-click on *My Computer*, select *Properties*, then *Advanced*, then *Environment Variables*. Then, you would update the PATH value and press the OK button.

On Linux, if the SDK is installed in /usr/local/jdk1.8.0_102 and you use the C shell, you would put the following code into your **.cshrc** file.

```
setenv PATH /usr/local/jdk1.8.0_102/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.8.0_102
```

Alternatively, if you use Android studio, then it will know automatically where you have installed your Java.

1.10.2 Android IDEs

There are so many sophisticated Technologies are available to develop android applications, the familiar technologies, which are predominantly using tools as follows

- Android Studio
- Eclipse IDE(Deprecated)

Android Studio Installation:

1) First of all, Download android studio from this link: https://developer.android.com/studio/index.html

2) JDK 8 is required when developing for Android 5.0 and higher (JRE is not enough). To check if you have JDK installed (and which version), open a terminal and type javac -version. If the JDK is not available or the version is lower than 6, download it from this link.

1.10.3 To set up Android Studio on Windows

1. Launch the .exe file you just downloaded.
2. Follow the setup wizard to install Android Studio and any necessary SDK tools.

On some Windows systems, the launcher script does not find where Java is installed. If you encounter this problem, you need to set an environment variable indicating the correct location.Select Start menu > Computer > System Properties > Advanced System Properties. Then open Advanced tab >

Environment Variables and add a new system variable JAVA_HOME that points to your JDK folder, for example C:\Program Files\Java\jdk1.8.x.(where x is version number).

1.10.4 To set up Android Studio on Mac OSX

1. Launch the .dmg file you just downloaded.
2. Drag and drop Android Studio into the Applications folder.
3. Open Android Studio and follow the setup wizard to install any necessary SDK tools.

Depending on your security settings, when you attempt to open Android Studio, you might see a warning that says the package is damaged and should be moved to the trash. If this happens, go to System Preferences > Security & Privacy and under Allow applications downloaded from, select Anywhere. Then open Android Studio again.

If you need use the Android SDK tools from a command line, you can access them at:

/Users/<user>/Library/Android/sdk/

1.10.5 To set up Android Studio on Linux

1. Unpack the downloaded the Android SDK Manager in one of the following ways:
   - In Android Studio, click SDK Manager in the toolbar.
   - If you're not using Android Studio:
2. Windows: Double-click ZIP file into an appropriate location for your applications.
3. To launch Android Studio, navigate to the android-studio/bin/ directory in a terminal and execute studio.sh. You may want to add android-studio/bin/ to your PATH environmental variable so that you can start Android Studio from any directory.
4. Follow the setup wizard to install any necessary SDK tools.

Android Studio is now ready and loaded with the Android developer tools, but there are still a couple packages you should add to make your Android SDK complete.

3) The SDK separates tools, platforms, and other components into packages you can download as needed using the Android SDK Manager. Make sure that you have downloaded all these packages.

To start adding packages, launch the SDK Manager.exe file at the root of the Android SDK directory.

1.10.6 XML

XML stands for Extensible Markup Language. XML is a markup language much like HTML used to describe data. XML tags are not predefined in XML. We must define our own Tags. Xml as itself is

well readable both by human and machine. Also, it is scalable and simple to develop. In Android we use xml for designing our layouts because xml is lightweight language so it doesn't make our layout heavy.

In this article we will go through the basic concepts of xml in Android and different XML files used for different purpose in Android. This will help you in writing a UI code to design your desired user interface.

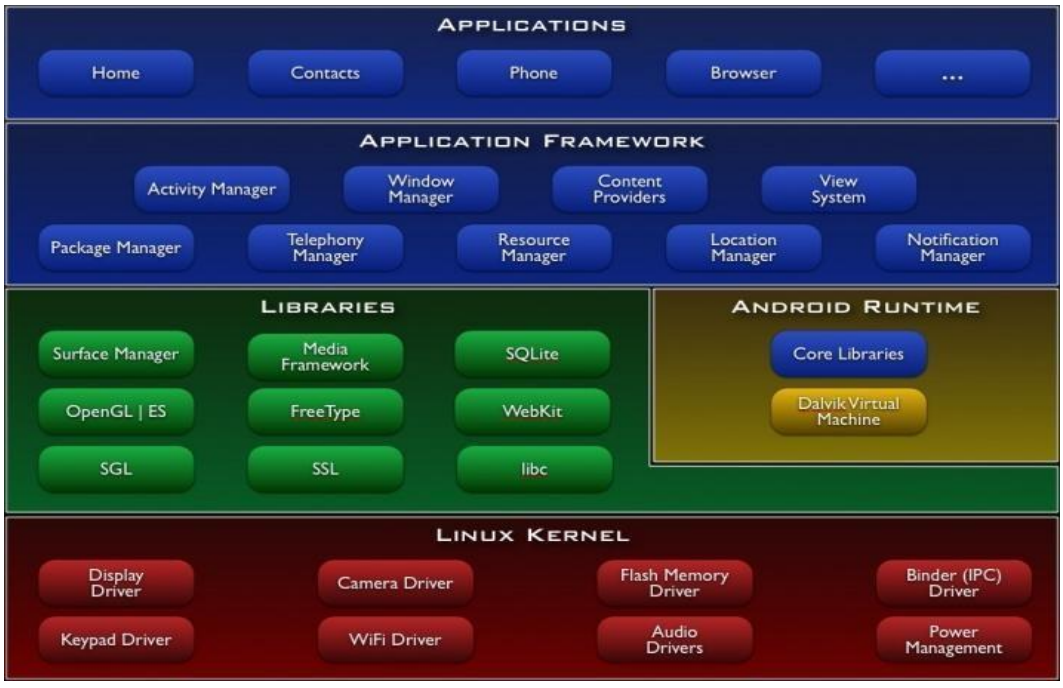
The whole concept of Android User Interface is defined using the hierarchy of View and ViewGroup objects. A ViewGroup is an invisible container that organizes child views. These child views are other widgets which are used to make the different parts of UI. One ViewGroup can have another ViewGroup as an child element as shown in the figure given below:

1.11 Android Architecture:

The Android operating system is built on top of a modified Linux kernel. The software stack contains Java applications running on top of a virtual machine. Components of the system are written in Java, C, C++, and XML. Android operating system is a stack of software components which is roughly divided into five sections

1) Linux kernel
2) Native libraries (middleware),
3) Android Runtime
4) Application Framework
5) Applications

Linux kernel



1.11.1 Linux kernel

It is the heart of android architecture that exists at the root of android architecture. The Linux Kernel is the bottom most layer in the Android architecture.

The Android platform is built on top of the Linux 2.6 Kernel with a few architectural changes.

There are several reasons for choosing the Linux kernel. Most importantly, Linux is a portable platform that can be compiled easily on different hardware. The kernel acts as an abstraction layer between the software and hardware present on the device.

Note that the term kernel refers to the core of any operating system.

The Linux Kernel provides support for

- memory management,
- security management,
- network stack,
- threading,
- process management, and
- Device management.

It has approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

The Linux Kernel contains a list of device drivers that facilitate the communication of an Android device with other peripheral devices.

A device driver is software that provides a software interface to the hardware devices. In doing so, these hardware devices can be accessed by the operating system and other programs.

1.11.2 Libraries (Middleware)

On top of Linux kernel there is a set of libraries or we may say Native libraries including

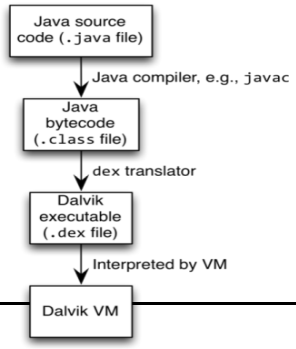
This category contains those Java-based libraries which are specific to Android development. A summarize study of some key core Android libraries available to the Android are discussed below−

- **android.app** − Provides access to the application model and is the cornerstone of all Android applications.
- **android.content** − Facilitates content access, publishing and messaging between applications and application components.

- **android.database** − Used to access data published by content providers and includes SQLite database management classes.
- **android.opengl** − A Java interface to the OpenGL ES 3D graphics rendering API.
- **android.os** − Provides applications with access to standard operating system services including messages, system services and inter-process communication.
- **android.text** − Used to render and manipulate text on a device display.
- **android.view** − The fundamental building blocks of application user interfaces.
- **android.widget** − A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
- **android.webkit** − A set of classes intended to allow web-browsing capabilities to be built into applications.

1.11.3 Android Runtime

- In android runtime, there are core libraries and DVM (Dalvik Virtual Machine) which is responsible to run android application. DVM is like JVM but it is optimized for mobile devices. It consumes less memory and provides fast performance.
- Moreover, the DVM is provides support for platform neutrality (the .dex files are platform neutral) and you can have multiple virtual machine instances execute at the same time efficiently. It should be noted that each Android application executes in its own process—inside its own instance of the Dalvik Virtual Machine
- The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine. DVM provides fast performance and consumes less memory.
- Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU.
- Android apps execute on Dalvik VM, a “clean-room” implementation of JVM
- Dalvik optimized for efficient execution
- Dalvik: register-based VM, unlike Oracle’s stack-based JVM
- Java .class bytecode translated to Dalvik EXecutable (DEX) bytecode, which Dalvik interprets



1.11.4 Application Framework

- On the top of Native libraries and android runtime, there is android framework. Android framework includes Android API's such as UI (User Interface), telephony, resources, locations, Content Providers (data) and package managers. It provides a lot of classes and interfaces for android application development.
- Android framework provides a lot of classes and interfaces for Android application development and higher level services to the applications in the form of Java classes.
  - **Activity Manager**: manages the life cycle of an applications and maintains the back stack as well so that the applications running on different processes has smooth navigations.
  - **Package Manager**: keeps track of which applications are installed in your device.
  - **Window Manager** : Manages windows which are java programming abstractions on top of lower level surfaces provided by surface manager.
  - **Telephony Managers**: manages the API which is use to build the phone applications.
  - **Content Providers:** Provide feature where one application can share the data with another application. like phone number , address, etc
  - **View Manager** : Buttons , Edit text , all the building blocks of UI, event dispatching etc.

1.12 Android emulator

- The Android SDK includes a virtual mobile device emulator that runs on your computer. The emulator lets you prototype, develop and test Android applications without using a physical device.
- Android Emulator: The Emulator is a new application in android operating system. The emulator is a new prototype that is used to develop and test android applications without using any physical device.
- The android emulator has all of the hardware and software features like mobile device except phone calls. It provides a variety of navigation and control keys. It also provides a screen to display your application. The emulators utilize the android virtual device configurations. Once your application is running on it, it can use services of the android platform to help other applications, access the network, play audio, video, store and retrieve the data.

1.13 Android Application

You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games etc.

An Android app is a software application running on the Android platform. Because the Android platform is built for mobile devices, a typical Android app is designed for a smartphone or a tablet PC running on the Android OS.

UNIT II

2.1 Java

21.1 Java knowledge

The Java programming language is one of the glorious tools that make programming Android a breeze compared with programming for other mobile platforms. Whereas other languages insist that you manage memory, deallocate and allocate bytes, and then shift bits around like a game of dominoes, Java's little buddy, the Java Virtual Machine (JVM), helps take care of that for you. The JVM allows you to focus on writing code to solve a business problem by using a clean, understandable programming language (or to build that next cool first-person shooter game you've been dreaming of) instead of focusing on the "plumbing" just to get the screens to show up. You're expected to understand the basics of the Java programming language before you write your first Android application.

2.1.2 Java: Your Android programming language

Android applications are written in Java — not the full-blown version of Java that's familiar to developers using Java Platform, Enterprise Edition (J2EE), but a subset of the Java libraries that are

specific to Android. This smaller subset of Java excludes classes that aren't suitable for mobile devices. If you have experience in Java, you should feel right at home developing apps in Android.

Not every class that's available to Java programmers is available also on Android. Verify that it's available to you before you start trying to use it. If it's not, an alternative is probably bundled with Android that can work for your needs.

2.2 Android Studio

Android Studio is the official IDE (integrated development environment) for developing Android Apps by Google. It is based on JetBrains' IntelliJ IDEA software and has lots of amazing features which helps developer in creating Android App. Android Studio is available for free download on Windows, Mac OS X and Linux.

2.2.1 System Requirement – First your system OS must be either Windows, Max OS X or Linux with below requirement:

- Microsoft Windows 10/8.1/8/7/Vista/2003/XP (32 or 64 bit)
- Mac OS X 10.8.5 or higher, up to 10.10 to up 10.10.2 up 10.10.3 on 10.10.5 (Yosemite)
- GNOME or KDE or Unity desktop on Ubuntu or Fedora or GNU/Linux Debian
- Minimum RAM: 2GB
- Recommended RAM: 4GB
- Disk Space: 500 MB disk space
- Android SDK Space Reqirement: At least 1 GB for Android SDK, emulator system images, and caches
- JDK: Java Development Kit (JDK) 7 or higher
- Screen Resolution: 1280×800 minimum screen resolution
- Prefer faster processor according to your budget

2.2.2 Android Studio installation

The second thing you need is to download Android Studio on your system and install it. It is available for free download on Windows, Mac OS X and Linux OS.

Android Studio – Step by Step procedure:

- Start New Project
- Open Project
- Reopen, Close & Save Project
- Create New Activity
- Create New Java Class
- Create Virtual Device
- Run App In AVD
- Run/Test App in Real Device

- Create Drawable Resource XML File
- Add/Create Landscape Layout
- Create Local HTML File
- Create Raw Folder
- Add/Create Assets Folder
- Install Genymotion Emulator
- Import/Add External JAR File
- Change API SDK Level
- Create/Add New Package Inside Src Folder
- Creating Folders for Adding Different Resolution Images
- Create An Interface
- Add Image to Drawable Folder in Android Studio
- Change Icon Of Your Android App
- Add Audio To Android App
- Application Launcher Icon Size
- Basic Activity In Android Studio
- Implement Abstract Method
- Change Package Name In Android Studio
- Generate Signed Apk In Android Studio For Publishing & Updating App – If you have completed developing your Android App and now wants to publish it on Playstore. Then the first step you need to take is generate signed apk in Android Studio for publishing your App

2.3 Eclipse

2.3.1 What is Eclipse?

■ Eclipse is a universal platform for integrating development tools for integrating development tools.

■ Open, extensible architecture based on plug-ins

Provide open platform for application development tools

– Run on a wide range of operating systems

– GUI and non GUI

■ Language –neutral

– Permit unrestricted content types

– HTML, Java, C, JSP, EJB, XML, GIF, …

■ Facilitate seamless tool integration

– At UI and deeper

– Add new tools to existing installed products

■ Attract community of tool developers

– Including independent software vendors (ISVs) g independent software vendors (ISVs)

–Capitalize on popularity of Java for writing tools Eclipse created by OT Eclipse created by OTI and IBM teams responsible for IDE products

– IBM VisualAge/Smalltalk (Smalltalk IDE)

– IBM VisualAge/Java (Java IDE)

– IBM VisualAge/Micro Edition (J Edition (Java IDE)

■ Initially staffed with 40 full Initially staffed with 40 full-time developers time developers

■ Geographically dispersed development teams Geographically dispersed development teams

– OTI Ottawa, OTI Minneapolis, OTI Zurich, IBM Toronto, OTI Ralei Toronto, OTI Raleigh, IBM RTP, IBM St. h, IBM RTP, IBM St. Nazaire Nazaire (France) (France)

■ Effort transitioned into open source project

–IBM donated initial Eclipse code

• Platform, JDT, PDE

2.3.2 Brief History of Eclipse

1999:April - Work begins on Eclipse inside OTI/IBM

2000:June - Eclipse Tech Preview ships

2001:March - http://www.eclipsecorner.org/ /www.eclipsecorner.org/ opens

June - Eclipse 0.9 ships Eclipse 0.9 ships

October - Eclipse 1.0 ships Eclipse 1.0 ships

November - IBM donates Eclipse source base - eclipse.org board announced http://www.eclipse.org/ /www.eclipse.org/

2002: June - Eclipse 2.0 ships

September - Eclipse 2.0.1 ships

November - Eclipse 2.0.2 ships

2003:March - Eclipse 2.1 ships

2.3.3 How to setup Android for Eclipse IDE

Software's are required for running an android application on eclipse IDE. Here, you will be able to learn how to install the android SDK and ADT plug-in for Eclipse IDE. Let's see the list of software required to setup android for eclipse IDE manually.

1. Install the JDK
2. Download and install the Eclipse for developing android application
3. Download and Install the android SDK
4. Install the ADT plug-in for eclipse
5. Configure the ADT plug-in
6. Create the AVD

7.Create the hello android application

1) Install the Java Development Kit (JDK)

For creating android application, JDK must be installed if you are developing the android application with Java language.

2) Download and install the Eclipse IDE

For developing the android application using eclipse IDE, you need to install the Eclipse.

3) Download and install the android SDK

First of all, download the android SDK.

4) Download the ADT plugin for eclipse

ADT (Android Development Tools) is required for developing the android application in the eclipse IDE. It is the plugin for Eclipse IDE that is designed to provide the integrated environment.

For downloading the ADT, you need to follow these steps:

1) Start the eclipse IDE, then select **Help** > **Install new software...**

2) In the **work with** combo box, write https://dl-ssl.google.com/android/eclipse/

3) **select the checkbox** next to Developer Tools and **click next**

4) You will see, a list of tools to be downloaded here, **click next**

5) **click finish**

6) After completing the installation, restart the eclipse IDE

5) Configuring the ADT plugin

After the installing ADT plugin, now tell the eclipse IDE for your android SDK location. To do so:

1. Select the **Window menu > preferences**
2. Now select the android from the left panel. Here you may see a dialog box asking if you want to send the statistics to the google. Click **proceed**.
3. Click on the browse button and locate your SDK directory e.g. my SDK location is C:\Program Files\Android\android-sdk .
4. Click the apply button then OK.

6) Create an Android Virtual Device (AVD)

For running the android application in the Android Emulator, you need to create and AVD. For creating the AVD:

1. Select the **Window menu > AVD Manager**
2. Click on the **new** button, to create the AVD
3. Now a dialog appears, write the AVD name e.g. myavd. Now choose the target android version e.g. android2.2.
4. click the **create AVD**

7) Create and run the simple android example

How to make android apps

To create the simple hello android application. We are creating the simple example of android using the Eclipse IDE. For creating the simple example:

1. Create the new android project
2. Write the message (optional)
3. Run the android application

2.4 Virtualization

This includes making a single physical resource (such as a server, an operating system, an application, or storage device) appear to function as multiple virtual resources; it can also include making multiple physical resources (such as storage devices or servers) appear as a single virtual resource..."

2.4.1 Types of Virtualization

Today the term virtualization is widely applied to a number of concepts including:

- Server Virtualization
- Client / Desktop / Application Virtualization
- Network Virtualization
- Storage Virtualization
- Service / Application Infrastructure Virtualization

In most of these cases, either virtualizing one physical resource into many virtual resources or turning many physical resources into one virtual resource is occurring.

2.4.2 Defining API Virtualization

API virtualization is the process of using a tool that creates a virtual copy of your API, which mirrors all of the specifications of your production API, and using this virtual copy in place of your production API for testing.

Instead of setting up a whole separate server stack to mimic production, API virtualization aims to simulate the minimum behaviors of one or more API endpoints.

To illustrate, API virtualization is the equivalent of allowing you (or, in this case, your testing team) to taste a cake – its flavor, texture, and all – before it has finished baking.

With API virtualization, your development teams can create virtual APIs instead of production APIs, enabling frequent and comprehensive testing even when the API is still in the midst of being developed.

By emulating behaviors and specifications that will be present in the final production API, virtualization allows for testing much earlier in the development process, removing key bottlenecks that would otherwise delay production and time-to-market. More and more companies are using virtualization to improve productivity, reduce testing costs, and deploy higher-quality APIs in a shorter timeframe.

By quickly and easily removing dependency constraints across your organization through virtualization, you can gain a competitive advantage over other companies still waiting in the linear-development limbo.

The Android SDK includes a virtual mobile device emulator that runs on your computer. The emulator lets you prototype, develop and test Android applications without using a physical device.

In this we are going to explore different functionalities in the emulator that are present in the real android device.

2.4.3 Creating AVD

If you want to emulate a real device, first crate an AVD with the same device configurations as real device, then launch this AVD from AVD manager.

Changing Orientation

Usually by default when you launch the emulator, its orientation is vertical, but you can change it orientation by pressing Ctrl+F11 key from keyboard.

First launch the emulator. It is shown in the picture below −

Once it is launched, press **Ctrl+F11** key to change its orientation. It is shown below −

2.4.4 Emulator Commands.

Apart from just orientation commands, there are other very useful commands of emulator that you should keep in mind while using emulator. They are listed below −

Sr.No	Command & description
1	**Home** Shifts to main screen
2	**F2** Toggles context sensitive menu
3	**F3** Bring out call log
4	**F4** End call

5	**F5** Search
6	**F6** Toggle trackball mode
7	**F7** Power button
8	**F8** Toggle data network
9	**Ctrl+F5** Ring Volume up
10	**Ctrl+F6** Ring Volume down

2.4.5 Emulator - Sending SMS

You can emulate sending SMS to your emulator. There are two ways to do that. You can do that from DDMS which can be found in Android studio, or from Telnet.(Network utility found in windows).

Sending SMS through Telnet.

Telnet is not enabled by default in windows. You have to enable it to use it. Once enabled you can go to command prompt and start telnet by typing telnet.

In order to send SMS , note down the AVD number which can be found on the title bar of the emulator. It could be like this 5554 e.t.c. Once noted , type this command in command prompt.

```
telnet localhost 5554
```

Press enter when you type the command. It is shown below in the figure.

You will see that you are now connected to your emulator. Now type this command to send message.

```
sms send 1234 "hello"
```

Once you type this command , hit enter. Now look at the AVD. You will receive a notification displaying that you got a new text message. It is shown below −

Emulator - Making Call

You can easily make phone calls to your emulator using telent client. You need to connect to your emulator from telnet. It is discussed in the sending sms topic above.

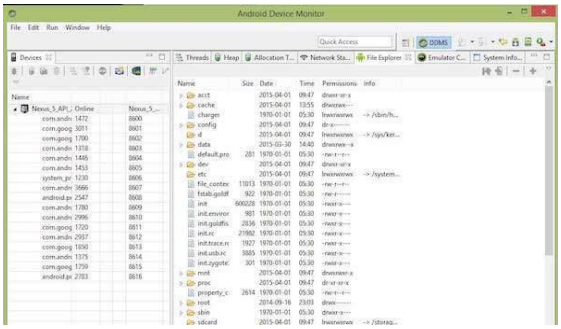
After that you will type this command in the telent window to make a call. Its syntax is given below −

```
gsm call 1234
```

Once you type this command , hit enter. Now look at the AVD. You will receive a call from the number your put in the command. It is shown below −

Emulator - Transferring files

You can easily transfer files into the emulator and vice versa. In order to do that, you need to select the DDMS utility in Android studio. After that select the file explorer tab. It is shown below −



Browse through the explorer and make new folder , view existing contents e.t.c.

2.5 APIs and Android tools

Use the Android Management API to integrate support for Android device and app management into your EMM console. The API and its companion DPC app, Android Device Policy, work together as a self-contained solution. For more information, see Development options.

2.5.1 Google Play EMM API

You can use the Google Play EMM API to integrate support for the following tasks into your EMM console:

- Specify apps that users are allowed to download onto managed devices.
- Host app APKs outside of Google Play. (Google Play hosts only the metadata for these APKs.)
- Manage user licensees in bulk for paid apps.
- Manage app installation.

The Play EMM API doesn't include device management features. To enforce management policies on devices, you need to develop your own DPC. For more information, see Develop a solution.

2.5.2 DPC development

For guidance on how to create a device policy controller (DPC) app, see Build a DPC.

A **sample DPC app** called **Test DPC** is available on Google Play. The code for Test DPC is available as an open source project on GitHub.

You can use Test DPC as a sample DPC or as a testing tool. As a testing tool, Test DPC provides an effective way to test applications and platforms in a managed context. For details, see the Test DPC readme file. To report an issue, use the Test DPC issue tracker.

When using Test DPC, keep the following in mind:

- Building Test DPC requires Android SDK v23, Android Build Tools v23.0.1, and the Android support repository.
- Test DPC supports devices running Android 5.0 Lollipop or later.
- Test DPC uses the Gradle build system.

2.5.3 Installing and Configuring Your Support Tools

It's time to put these exciting Android concepts into action, but before you can do so, you need to install and configure a few tools, including the software development kits (SDKs):

✓ Java JDK: Lays the foundation for the Android SDK.

✓ Android SDK: Provides access to Android libraries and allows you to develop for Android. ✓ Eclipse IDE (integrated development environment): Brings together Java, the Android SDK, and the Android Android Development Tools (ADT) and provides tools for you to write Android programs.

✓ Android ADT: Does a lot of the grunt work for you, such as creating the files and structure required for an Android app.

2.6 Debugging Applications with DDMS

The Dalvik Debug Monitor Service (DDMS) is a debugging utility that is integrated into Eclipse through a special Eclipse perspective. The DDMS perspective provides a number of useful features for interacting with emulators and handsets and debugging applications.

The DDMS perspective, with one emulator and two Android devices connected (the Nexus S running 2.3.1 and the Samsung Galaxy Tablet running

2.6.1 The features of DDMS are roughly divided into five functional areas:

- Task management
- File management
- Emulator interaction
- Logging
- Screen captures

DDMS and the DDMS perspective are essential debugging tools. Now let's take a look at how to use these features in a bit more detail.

2.6.2 Built-in debugging tools

- Logcat
- Debug
- Lint

2.6.3 Plugins for developer productivity

- ADB Idea
- Codota
- Lombok Plugin

Debugging Android: Debug

Using Logcat to log and correct code is okay for very simple apps. For more complicated apps, this form of debugging can be tedious. Instead you'll want something that lets you debug the app's executable code. Android Studio's built-in Debug tool offers many capabilities, including the following:

- Select a device on which to debug your app.
- Set breakpoints in your application code (Java, Kotlin, or C/C++).
- Examine variables and evaluate expressions at runtime.

2.6.4 Debugging tool

1. If your app includes C/C++ source code, you'll need to install LLDB from the SDK Manager (see Figure 3). Fortunately, the example app for this series (**W2A**) doesn't contain C/C++ code, so we can ignore this prerequisite.
2. You must enable debugging on a connected device. However, if you're using an emulator (which we are for this example), you can ignore this prerequisite. Debugging is enabled by default on emulated devices.
3. You must run a debug gable build variant. By default, this is created for you, so in many cases (including this example) you don't have to worry about it.

2.7 Android File system

2.7.1 Flash Memory Android File System

1. exFAT

Originally created by Microsoft for flash memory, the exFAT file system is not a part of the standard Linux kernel. However, it still provides support for Android devices in some cases. It stands for Extended File Allocation Table.

2.F2FS

Users of Samsung smartphones are bound to have come across this type of file system if they have been using the smartphone for a while. F2FS stands for Flash-Friendly File System, which is an Open Source Linux file system. This was introduced by Samsung 4 years ago, in 2012.

3. JFFS2

It stands for the Journal Flash File System version 2. This is the default flash file system for the Android Open Source Project kernels. This version of Android File System has been around since the Android Ice Cream Sandwich OS was released. JFFS2 has since replaced the JFFS.

4. YAFFS2

It stands for Yet Another Flash File System version 2. Funny as the name might sound like, it is actually a serious business! It has not been a part of the AOSP for a while now and is rarely found in Android smartphones. However, it does tend to make a few appearances every now and then.

The Android OS is a popular and universally used operating system for smart phones. The Android File Systems tend to be rather complicated and have a number of users scratching their head in amusement.

2.7.2 Media-based Android File System

1. EXT2/EXT3/EXT4

Ext, which stands for the EXTended file systems, are the standards for the Linux file system. The latest out of these is the EXT4, which has now been replacing the YAFFS2 and the JFFS2 file systems on Android smartphones.

2. MSDOS

Microsoft Disk Operating System is known to be one of the oldest names in the world of Operating Systems, and it helps FAT 12, FAT 16 and FAT 32 file systems to run.

3. VFAT

An extension to the aforementioned FAT 12, FAT 16 and FAT 32 file systems, the VFAT is a kernel module seen alongside the MSDOS module. External SD cards that help expand the storage space are formatted using VFAT.

2.7.3 Pseudo File Systems

1. CGroup

Cgroup stands for Control Group. It is a pseudo file system which allows access and meaning to various kernel parameters. Cgroups are very important for the Android File System as the Android OS makes use of these control groups for user accounting and CPU Control.

2. Rootfs

Rootfs acts as the mount point, and it is a minimal file system. It is located at the mount point "/".

3. Procfs

Usually found mounted at the /proc directory. The procfs file system has files which showcase the live kernel data. Sometimes this file system also reflects a number of kernel data structures. These number directories are reflective of the process IDs for all the currently running tasks.

4. Sysfs

Usually mounted on the /sys directory. The sysfs file system helps the kernel identify the devices. Upon identifying a new device, the kernel builds an object in sys/module/ directory. There are various other elements stored inside the /sys/ folder which helps the kernel communicate with various Android File Systems.

5. Tmpfs

A temporary file system, tmpfs is usually mounted on /dev directory. Data on this is lost when the device is rebooted.

2.8 Working with emulator and smart devices

To perform mobile testing, you need a mobile device. This is to access that how our product will work and look like on a given mobile set.

Suppose we are developing an application for flight ticket booking system. Once the product is entirely developed, as a part of mobile testing, we need to check if the application is working as expected with all the majorly used devices like Android phones, iOS, Blackberry phones, and other different types of tablets and iPads.

To do this kind of check, we need to acquire each such device and then we can check if the application behaves as per expectation. Yes you thought right, as a product owner one will defiantly find this very expensive to procure such a large number of mobile devices and carry out testing. So is there any smart alternate available?

The solution to this problem is to use Mobile Simulators and Mobile Emulators. These are primarily software programs designed to provide simulation for important features of a smartphone. They are very similar in nature, so sometimes, they are used interchangeably.

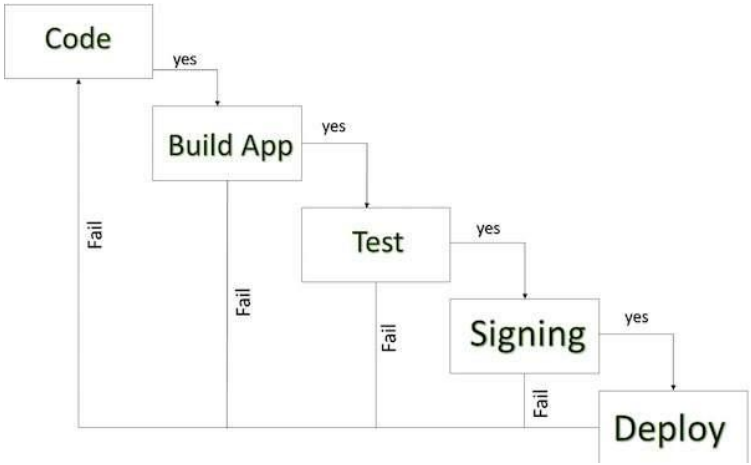
2.8.1 Testing on an Emulator/Simulator is different from testing on a real device

	Real Device	Emulator / Simulator
Price	Getting real devices will cost you a lot.	It is almost free, we just need to download and install them
Processing Speed	It has faster processing; however network latency may be normal.	It is slower as compared to actual devices. It has observed less latency than real devices connected to the local network or in the cloud.
Debugging	Debugging is not that easy.	It provides step-by-step debugging of an application. Also, it provides an efficient way for capturing screenshots.
Web-app Testing	Web applications can be tested in a normal way.	Testing a web application is much easier.
Reliability	Testing on a real device has a major advantage that it always gives accurate results.	It cannot simulate all types of user interactions; hence it may lead to false results sometimes. So it scores low when it comes to reliability.

A simulator/emulator cannot mimic the following features −

- Mobile device battery
- Mobile device's camera
- Difficult to mimic interruptions like incoming calls and SMS.
- Not so much realistic simulation for mobile device memory usage.

Android application publishing is a process that makes your Android applications available to users. Infect, publishing is the last phase of the Android application development process.



ANDROID DEVELOPMENT LIFE CYCLE

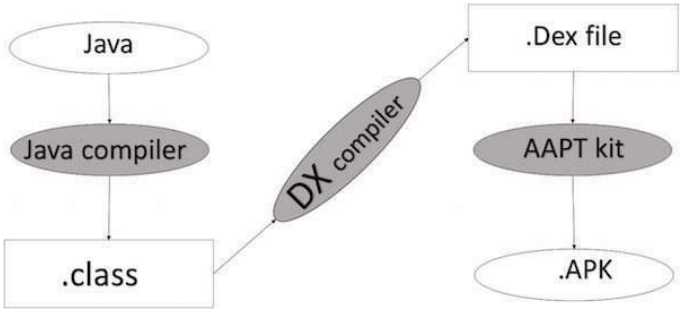
Once you developed and fully tested your Android Application, you can start selling or distributing free using Google Play (A famous Android marketplace). You can also release your applications by sending them directly to users or by letting users download them from your own website.

You can check a detailed publishing process at Android official website, but this tutorial will take you through simple steps to launch your application on Google Play. Here is a simplified check list which will help you in launching your Android application −

Step	Activity
1	**Regression Testing** Before you publish your application, you need to make sure that its meeting the basic quality expectations for all Android apps, on all of the devices that you are targeting. So perform all the required testing on different devices including phone and tablets.
2	**Application Rating** When you will publish your application at Google Play, you will have to specify a content rating for your app, which informs Google Play users of its maturity level. Currently available ratings are (a) Everyone (b) Low maturity (c) Medium maturity (d) High maturity.
3	**Targeted Regions** Google Play lets you control what countries and territories where your application will be sold. Accordingly you must take care of setting up time zone, localization or any other specific requirement as per the targeted region.
4	**Application Size** Currently, the maximum size for an APK published on Google Play is 50 MB. If your app exceeds that size, or if you want to offer a secondary download, you can use APK Expansion Files, which Google Play will host for free on its server infrastructure and automatically handle the download to devices.
5	**SDK and Screen Compatibility** It is important to make sure that your app is designed to run properly on the Android platform versions and device screen sizes that you want to

	target.
6	**Application Pricing** Deciding whether you app will be free or paid is important because, on Google Play, free app's must remain free. If you want to sell your application then you will have to specify its price in different currencies.
7	**Promotional Content** It is a good marketing practice to supply a variety of high-quality graphic assets to showcase your app or brand. After you publish, these appear on your product details page, in store listings and search results, and elsewhere.
8	**Build and Upload release-ready APK** The release-ready APK is what you you will upload to the Developer Console and distribute to users. You can check complete detail on how to create a release-ready version of your app: **Preparing for Release**.
9	**Finalize Application Detail** Google Play gives you a variety of ways to promote your app and engage with users on your product details page, from colourful graphics, screen shots, and videos to localized descriptions, release details, and links to your other apps. So you can decorate your application page and provide as much as clear crisp detail you can provide.

2.8.2 Export Android Application Process



APK DEVELOPMENT PROCESS

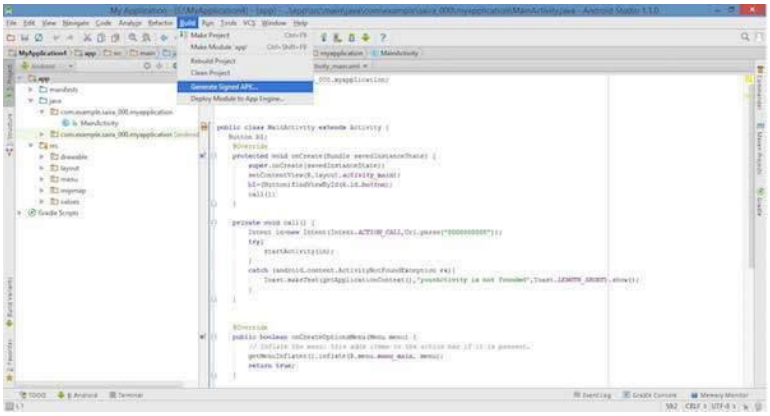
Before exporting the apps, you must some of tools

- **Dx tools**(Dalvik executable tools): It going to convert **.class file** to **.dex file**. it has useful for memory optimization and reduce the boot-up speed time

- **AAPT**(Android assistance packaging tool):it has useful to convert **.Dex file** to**.Apk**
- **APK**(Android packaging kit): The final stage of deployment process is called as .apk.

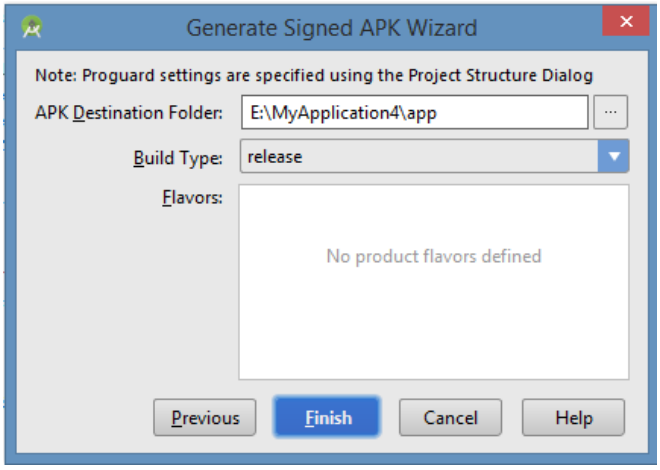
You will need to export your application as an APK (Android Package) file before you upload it Google Play marketplace.

To export an application, just open that application project in Android studio and select **Build → Generate Signed APK** from your Android studio and follow the simple steps to export your application –

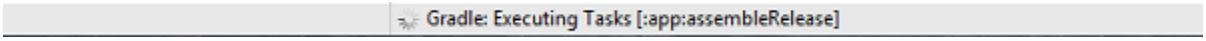


Next select, **Generate Signed APK** option as shown in the above screen shot and then click it so that you get following screen where you will choose **Create new keystore** to store your application.

Enter your key store path,key store password,key alias and key password to protect your application and click on **Next** button once again. It will display following screen to let you create an application –



Once you filled up all the information,like app destination,build type and flavours click **finish** button While creating an application it will show as below



Finally, it will generate your Android Application as APK formate File which will be uploaded at Google Play marketplace.

2.9 A Basic Android Application

To develop Android applications and, in the process, install the Eclipse Android Development Tools (ADT) plug-in. It gives you the power to generate new Android applications directly from within the Eclipse File menu.

Follow these steps to create your first Android application project:

1. In Eclipse, choose File➪New➪Other. Select Android Application Project.

2. Enter Hello Android as the application name. The application name is the name of the application as it pertains to Android. When the application is installed on the emulator or physical device, this name appears in the application launcher.

The Project and Package names should auto complete for you.

The Project Name field is important. The descriptive name you provide identifies your project in the Eclipse workspace. After your project is created, a folder in the workspace is named with the project name you define.

3. In the Package Name box, type com.dummies.android.helloandroid. This is the name of the Java package. (See the nearby sidebar “Java package nomenclature.”)

4. Select Android 4.1 from the Build SDK drop-down list and API 8: Android 2.2 from the Minimum Required SDK drop-down list, and then click Next.

5. (Optional) Create an application icon for your project and click Next. 6. In the Create Activity box, choose BlankActivity and click Next. The New Blank Activity screen appears.

7. Enter MainActivity in the Activity Name box. The New Blank Activity screen defines what the initial activity is called — the entry point to your application. When Android runs your application, this

file is the first one to be accessed. A common naming pattern for the first activity in your application is MainActivity.java. (How creative.)

8. Click the Finish button. You're done! You should see Eclipse with a single project in the Package Explorer. Different names; same function.

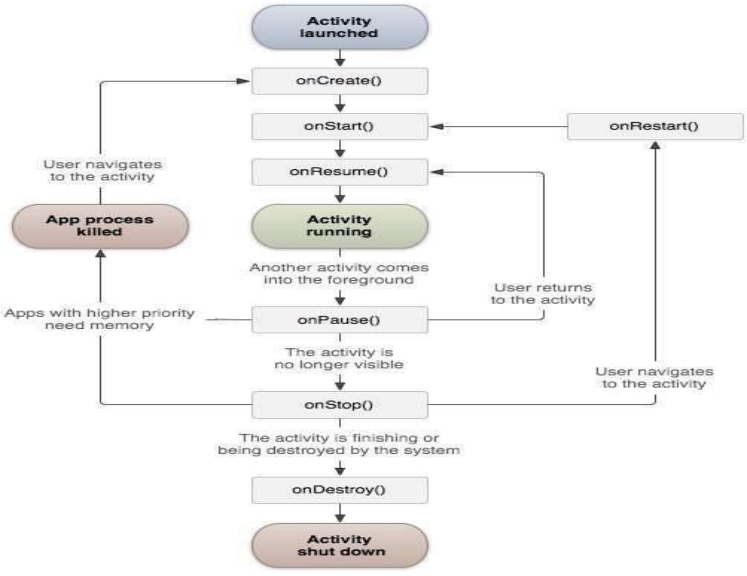
2.10 Activity

Android system initiates its program within an **Activity** starting with a call on *onCreate()* callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity. as shown in the below Activity life cycle diagram: (*image courtesy : android.com*)

2.10.1 Activity life cycle

The rectangles represent callback methods you can implement to respond to events in the activity. The shaded ovals represent the major states of the activity.

The Activity class defines the following call backs i.e. events. You don't need to implement all the callbacks methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.



2.10.2 The movement of an activity life cycle:

Sr.No	Callback & Description
1	**onCreate()** This is the first callback and called when the activity is first created.
2	**onStart()** This callback is called when the activity becomes visible to the user.
3	**onResume()** This is called when the user starts interacting with the application.
4	**onPause()** The paused activity does not receive user input and cannot execute any code and called when the current activity is being paused and the previous activity is being resumed.
5	**onStop()** This callback is called when the activity is no longer visible.
6	**onDestroy()** This callback is called before the activity is destroyed by the system.
7	**onRestart()** This callback is called when the activity restarts after stopping it.

2.10.3 Lifecycle methods

You may be interested in monitoring these three loops in your activity:

✓ The **entire lifetime** takes place between the first call to onCreate() and the final call to onDestroy(). The activity performs all global setupin onCreate() and releases all remaining resources in onDestroy(). For example, if you create a thread to download a file from the Internet in the background, it may be initialized in the onCreate() method. That thread can be stopped in the onDestroy() method.

✓ The **visible lifetime** of the activity takes place between the onStart() and onStop() methods. During this time, the user can see the activity onscreen (though it may not be in the foreground interacting with the user, which can happen when the user is interacting with a dialog box). Between these two methods, you can maintain the resources that are needed to show and run your activity.

✓ The **foreground lifetime** of the activity begins at the call to onResume() and ends at the call to onPause(). During this time, the activity is in front of all other activities and is interacting with the user. An activity normally toggles between onResume() and onPause() multiple times, for example, when the device goes to sleep or when a new activity handles a particular event — therefore, the code in these methods must be fairly lightweight.

Viewing activity methods

The entire activity life cycle boils down to these methods:

```
public class Activity extends ApplicationContext {
protected void onCreate(Bundle savedInstanceState);
protected void onStart();
protected void onRestart();
protected void onResume();
protected void onPause();
protected void onStop();
protected void onDestroy();
}
```

All methods can be overridden, and custom code can be placed in all of them. All activities implement onCreate() for initialization and may also implement onPause() for clean-up. You should always call the superclass (base class) when implementing these methods.

2.10.4 Creating Your First Activity

You may have already created your first activity if you created a project using the New Android roject Wizard in Chapter 3: the MainActivity activity. Open the MainActivity.java file in your project to enhance it in the following sections.

Starting with onCreate

The entry point into your application is the onCreate() method. The code for the MainActivity.java file already contains an implementation of the onCreate() method. It's where you start writing code! For now, your code should look like this:

```
public class MainActivity extends Activity {
/** Called when the activity is first created. */
@Override
public void onCreate(Bundle savedInstanceState) {
```

```
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
}
}
```

You write the initialization code directly below the setContentView() method.

Be sure to always include this method call to your onCreate() method:

super.onCreate(savedInstanceState);

It's required for the application to run. This line directs the base Activity class to perform setup work for the MainActivity class. If you omit this line of code, you receive a runtime exception.

2.11 Intents

All Android activities are started or activated with an intent. Intents are message objects that make a request to the Android runtime to start an activity or other app component in your app or in some other app. You don't start those activities yourself.

When your app is first started from the device home screen, the Android runtime sends an intent to your app to start your app's main activity (the one defined with the MAIN action and the LAUNCHER category in the Android Manifest). In addition to starting activities, intents are also used to pass data between activities. When you create an intent to start a new activity, you can include information about the data you want that new activity to operate on.

2.11.1 Intent types

There are two types of intents in Android

1. Explicit intents specify the receiving activity (or other component) by that activity's fully-qualified class name. Use an explicit intent to start a component in your own app (for example, to move between screens in the user interface), because you already know the package and class name of that component.

2. Implicit intents do not specify a specific activity or other component to receive the intent. Instead you declare a general action to perform in the intent. The Android system matches your request to an activity or other component that can handle your requested action.

2.11.2 Intent objects and fields

An Intent object is an instance of the Intent class. For explicit intents, the key fields of intent include the following:

1. The activity class (for explicit intents). This is the class name of the activity or other component that should receive the intent, for example, com.example.SampleActivity.class. Use the intent constructor or the intent's setComponent(), setComponentName() or setClassName() methods to specify the class.

2. The intent data. The intent data field contains a reference to the data you want the receiving activity to operate on, as a Uri object.

3. Intent extras. These are key-value pairs that carry information the receiving activity requires to accomplish the requested action.

4. Intent flags. These are additional bits of metadata, defined by the Intent class. The flags may instruct the Android system how to launch an activity or how to treat it after it's launched.

2.11.3 Starting an activity with an explicit intent

To start a specific activity from another activity, use an explicit intent and the startActivity() method. Explicit intents include the fully-qualified class name for the activity or other component in the Intent object. All the other intent fields are optional, and null by default.

Ex:

intent message intent=new intent(this, showmessageAvtivity.class);

startActivity(messageIntent);

The Intent constructor takes two arguments for an explicit intent.

- An application context. In this example, the activity class provides the content (here, this).
- The specific component to start (ShowMessageActivity.class).

Passing data between activities with intents

In addition to simply starting one activity from another, you also use intents to pass information between activities. The intent object you use to start an activity can include intent data (the URI of an object to act on), or intent extras, which are bits of additional data the activity might need. In the first (sending) activity, you

1. Create the Intent object.
2. Put data or extras into that intent.

3. Start the new activity with startActivity(). In the second (receiving) activity, you: 1. Get the intent object the activity was started with. 2. Retrieve the data or extras from the Intent object.

2.12 Intent Filters

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 action is 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.

2.13 Activity stack

Activities in the system are managed as an activity stack. When a new activity is created, it's placed on top of the stack and becomes the running activity. The previous running activity always remains below it in the stack and returns to the foreground only when the new activity exits.

2.13.1 An activity has essentially four states

Activity State	*Description*
Active/running	The activity is in the foreground of the screen (at the top of the stack).
Paused	The activity has lost focus but is still visible. (A new, non-full-size or transparent activity has the focus on top of your activity.) Because a paused activity is completely alive, it can maintain state and member information and remains attached to the window manager in Android. However, up through Gingerbread (3.0) the activity can be killed by the Android system in extreme low-memory conditions.
Stopped	If an activity becomes obscured by another activity, it's stopped. It retains all state and member information, but isn't visible to the user. Therefore, the window is hidden and will often be killed by the Android system when memory is needed elsewhere.
Created and resumed	The system has either paused or stopped the activity. The system can reclaim the memory by asking it to finish, or it can kill the process. When it displays the activity to the user, it must resume by restarting and restoring to its previous state.

2.1 Simple Service

- A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.
- Services are sort of like Activities that don't have a user interface or user interaction directly tied to them. Services can be launched by Activities or Applications, but then do their own thing in the background even after the starting component is closed.
- Once a Service is started, it keeps running until it is explicitly stopped

A service is implemented as a subclass of Service class as follows:

```
public class MyService extends Service
{


}
```

Some common uses for a Service include:

- Downloading or uploading data from/to a network in the background even if the app is closed
- Saving data to a database without crashing if the user leaves the app
- Running some other kind of long-running, "background" task even after the app is closed, such as playing music.
- A service can essentially take two states –

Sr.No.	State & Description
1	**Started** A service is **started** when an application component, such as an activity, starts it by calling *startService()*. Once started, a service can run in the background indefinitely, even if the component that started it is destroyed.
2	**Bound** A service is **bound** when an application component binds to it by calling *bindService()*. A bound service offers a client-server interface that allows components to interact with the service, send requests, get results, and even do so across processes with interprocess communication (IPC).

Foreground Services

A foreground service performs some operation that is noticeable to the user. For example, an audio app would use a foreground service to play an audio track. Foreground services must display a Notification. Foreground services continue running even when the user isn't interacting with the app.

Background Services

A background service performs an operation that isn't directly noticed by the user. For example, if an app used a service to compact its storage, that would usually be a background service.

2.1.1 Creating a Service

To create a service, you create a Java class that extends the Service base class or one of its existing subclasses. The Service base class defines various callback methods and the most important are given below.

Sr.No.	Callback & Description
1	**onStartCommand()** The system calls this method when another component, such as an activity, requests that the service be started, by calling startService(). If you implement this method, it is your responsibility to stop the service when its work is done, by calling stopSelf() or stopService() methods.
2	**onBind()** The system calls this method when another component wants to bind with the service by calling bindService(). If you implement this method, you must provide an interface that clients use to communicate with the service, by returning an IBinder object. You must always implement this method, but if you don't want to allow binding, then you should return null.
3	**onUnbind()** The system calls this method when all clients have disconnected from a particular interface published by the service.
4	**onRebind()** The system calls this method when new clients have connected to the service, after it had previously been notified that all had disconnected in its onUnbind(Intent).

5	**onCreate()** The system calls this method when the service is first created using onStartCommand() or onBind(). This call is required to perform one-time set-up.
6	**onDestroy()** The system calls this method when the service is no longer used and is being destroyed. Your service should implement this to clean up any resources such as threads, registered listeners, receivers, etc.

The following skeleton service demonstrates each of the life cycle methods −

```
package com.tutorialspoint;
import android.app.Service;
import android.os.IBinder;
import android.content.Intent;
import android.os.Bundle;
public class HelloService extends Service {
  /** indicates how to behave if the service is killed */
  int mStartMode;
  /** interface for clients that bind */
  IBinder mBinder;
  /** indicates whether onRebind should be used */
  boolean mAllowRebind;
 /** Called when the service is being created. */
  @Override
  public void onCreate() {
  }
```

```
    /** The service is starting, due to a call to startService() */
    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {
        return mStartMode;
    }
    /** A client is binding to the service with bindService() */
    @Override
    public IBinder onBind(Intent intent) {
        return mBinder;
    }
    /** Called when all clients have unbound with unbindService() */
    @Override
    public boolean onUnbind(Intent intent) {
        return mAllowRebind;
    }
    /** Called when a client is binding to the service with bindService()*/
    @Override
    public void onRebind(Intent intent) {
    }
    /** Called when The service is no longer used and is being destroyed */
    @Override
    public void onDestroy() {
    }
}
```

To implement any kind of service in your app:

1. Declare the service in the manifest.
2. Create implementation code, as described in Started services and Bound services, below.
3. Manage the service lifecycle.

2.1.2 Declaring services in the manifest

As with activities and other components, you must declare all services in your application's manifest file. To declare a service, add a <service> element as a child of the <application> element. For example:

```
<manifest ... >
  ...
  <application ... >
      <service android:name="ExampleService"
               android:exported="false" />
      ...
  </application>
</manifest>
```

To block access to a service from other applications, declare the service as private. To do this, set the android:exported attribute to false. This stops other apps from starting your service, even when they use an explicit intent.

2.1.3 Started service

- When a service is started, it has a lifecycle that's independent of the component that started it and the service can run in the background indefinitely, even if the component that started it is destroyed. As such, the service should stop itself when its job is done by calling **stopSelf(),** or another component can stop it by calling **stopService().**
- An application component such as an activity can start the service by calling **startService()** and passing an Intent that specifies the service and includes any data for the service to use. The service receives this Intent in the **onStartCommand()** method.
- For instance, suppose an activity needs to save some data to an online database. The activity can start a companion service and deliver it the data to save by passing an intent to **startService**(). The service receives the intent in **onStartCommand()**, connects to the Internet and performs the database transaction. When the transaction is done, the service stops itself and it is destroyed.
- Traditionally, there are two classes you can extend to create a started service:

Service

This is the base class for all services. When you extend this class, it's important that you create a new thread in which to do all the service's work, because the service uses your application's main thread, by default, which could slow the performance of any activity your application is running.

IntentService

This is a subclass of Service that uses a worker thread to handle all start requests, one at a time. This is the best option if you don't require that your service handle multiple requests simultaneously. All you need to do is implement **onHandleIntent(),** which receives the intent for each start request so you can do the background work.

2.1.3.1 IntentService

- Because most started services don't need to handle multiple requests simultaneously (which can actually be a dangerous multi-threading scenario), it's probably best if you implement your service using the IntentServiceclass.
- The IntentService does the following:
    1. Creates a default worker thread that executes all intents delivered to **onStartCommand()** separate from your application's main thread.
    2. Creates a work queue that passes one intent at a time to your **onHandleIntent()** implementation, so you never have to worry about multi-threading.
    3. Stops the service after all start requests have been handled, so you never have to call **stopSelf()**.
    4. Provides default implementation of **onBind()** that returns null.
    5. Provides a default implementation of **onStartCommand()** that sends the intent to the work queue and then to your **onHandleIntent()** implementation.

Here's an example implementation of **IntentService**:

```
public class HelloIntentService extends IntentService {

  /**
   * A constructor is required, and must call the super IntentService(String)
   * constructor with a name for the worker thread.
   */
  public HelloIntentService() {
      super("HelloIntentService");
  }

  /**
   * The IntentService calls this method from the default worker thread with
   * the intent that started the service. When this method returns, IntentService
   * stops the service, as appropriate.
   */

  @Override
  protected void onHandleIntent(Intent intent) {
```

```
    // Normally we would do some work here, like download a file.
    // For our sample, we just sleep for 5 seconds.
    long endTime = System.currentTimeMillis() + 5*1000;
    while (System.currentTimeMillis() < endTime) {
      synchronized (this) {
        try {
          wait(endTime - System.currentTimeMillis());
        } catch (Exception e) {
        }
      }
    }
  }
}
```

2.1.3.2 Extending Service Class

If, however, you require your service to perform multi-threading (instead of processing start requests through a work queue), then you can extend the Service class to handle each intent.

However, because you handle each call to **onStartCommand()** yourself, you can perform multiple requests simultaneously.

Notice that the **onStartCommand()** method must return an integer. The integer is a value that describes how the system should continue the service in the event that the system kills it (as discussed above, the default implementation for **IntentService** handles this for you, though you are able to modify it). The return value from **onStartCommand()** must be one of the following constants:

START_NOT_STICKY - If the system kills the service after **onStartCommand()** returns, do not recreate the service, unless there are pending intents to deliver. This is the safest option to avoid running your service when not necessary and when your application can simply restart any unfinished jobs.

START_STICKY - If the system kills the service after **onStartCommand()** returns, recreate the service and call **onStartCommand()**, but *do not* redeliver the last intent. Instead, the system calls **onStartCommand()**with a null intent, unless there were pending intents to start the service, in which case, those intents are delivered. This is suitable for media players (or similar services) that are not executing commands, but running indefinitely and waiting for a job.

START_REDELIVER_INTENT-If the system kills the service after **onStartCommand()** returns, recreate the service and - all **onStartCommand()** with the last intent that was delivered to the service. Any pending intents are delivered in turn. This is suitable for services that are actively performing a job that should be immediately resumed, such as downloading a file.

2.1.4 Bounding and Querying the Service

- A client can bind to the service by calling **bindService()**. When it does, it must provide an implementation of **ServiceConnection**, which monitors the connection with the service.

The **bindService()** method returns immediately without a value, but when the Android system creates the connection between the client and service, it calls **onServiceConnected()** on the **ServiceConnection,** to deliver the **IBinder** that the client can use to communicate with the service.

- Multiple clients can connect to the service at once. However, the system calls your service's **onBind()** method to retrieve the **IBinder** only when the first client binds. The system then delivers the same **IBinder** to any additional clients that bind, without calling **onBind()** again.
- When the last client unbinds from the service, the system destroys the service (unless the service was also started by **startService()**).
- When you implement your bound service, the most important part is defining the interface that your **onBind()** callback method returns. There are a few different ways you can define your service's **IBinder** interface and the following section discusses each technique.

Implementing a bound service

- To implement a bound service, define the interface that specifies how a client can communicate with the service. This interface, which your service returns from the onBind() callback method, must be an implementation of **IBinder**.
- To retrieve the IBinder interface, a client application component calls bindService(). Once the client receives the IBinder, the client interacts with the service through that interface.

Binding to a service

To bind to a service that is declared in the manifest and implemented by an app component, use **bindService()** with an **explicit** Intent.

```
public class LocalService extends Service {
    // Binder given to clients
    private final IBinder mBinder = new LocalBinder();
    // Random number generator
    private final Random mGenerator = new Random();

    /**
     * Class used for the client Binder.  Because we know this service always
     * runs in the same process as its clients, we don't need to deal with IPC.
     */
    public class LocalBinder extends Binder {
        LocalService getService() {
            // Return this instance of LocalService so clients can call public methods
            return LocalService.this;
        }
    }

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

```
    /** method for clients */
    public int getRandomNumber() {
      return mGenerator.nextInt(100);
    }
}
```

The LocalBinder provides the getService() method for clients to retrieve the current instance of LocalService. This allows clients to call public methods in the service. For example, clients can call getRandomNumber() from the service.

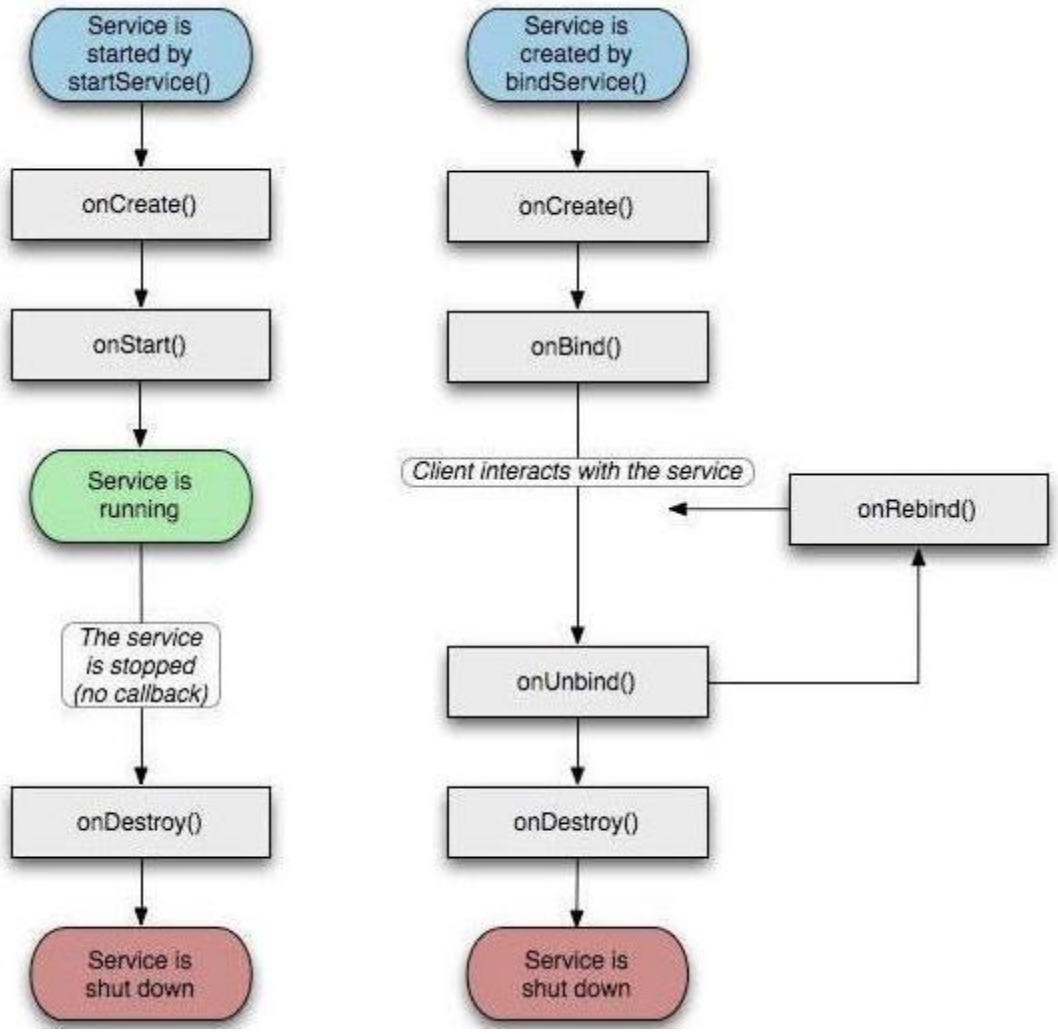
2.1.5 Managing the life cycle

- When a service is unbound from all clients, the Android system destroys it (unless it was also started with **onStartCommand**()). As such, you don't have to manage the lifecycle of your service if it's purely a bound service—the Android system manages it for you based on whether it is bound to any clients.
- However, if you choose to implement the **onStartCommand**() callback method, then you must explicitly stop the service, because the service is now considered to be started. In this case, the service runs until the service stops itself with **stopSelf**() or another component calls stopService(), regardless of whether it is bound to any clients.
- Additionally, if your service is started and accepts binding, then when the system calls your onUnbind()method, you can optionally return true if you would like to receive a call to **onRebind**() the next time a client binds to the service (instead of receiving a call to **onBind()**). onRebind() returns void, but the client still receives the **IBinder** in its **onServiceConnected()** callback.

2.2 Service Life Cycle

- The lifecycle of a service is simpler than that of an activity. However, it's even more important that you pay close attention to how your service is created and destroyed. Because a service has no UI, services can continue to run in the background with no way for the user to know, even if the user switches to another application. This consumes resources and drains battery.

- Like an activity, a service has lifecycle callback methods that you can implement to monitor changes in the service's state and perform work at the appropriate times.



2.3. Executing the services

Starting a service

- You can start a service from an activity or other application component by passing an Intent to **startService()** or **startForegroundService()**. The Android system calls the service's **onStartCommand()** method and passes it the **Intent**, which specifies which service to start.
- The **startService()** method returns immediately, and the Android system calls the service's **onStartCommand()** method. If the service isn't already running, the system first calls **onCreate(),** and then it calls **onStartCommand()**.
- If the service doesn't also provide binding, the intent that is delivered with **startService()** is the only mode of communication between the application component and the service. However, if you want the service to send a result back, the client that starts the service can create a **PendingIntent** for a broadcast (with **getBroadcast()**) and deliver it to the service in the Intent that starts the service. The service can then use the broadcast to deliver a result.

- Multiple requests to start the service result in multiple corresponding calls to the service's **onStartCommand()**. However, only one request to stop the service (with **stopSelf() or stopService()**) is required to stop it.

Stopping a service

- A started service must manage its own lifecycle. That is, the system doesn't stop or destroy the service unless it must recover system memory and the service continues to run after **onStartCommand()** returns. The service must stop itself by calling **stopSelf(),** or another component can stop it by calling **stopService().**
- Once requested to stop with **stopSelf()** or **stopService()**, the system destroys the service as soon as possible.
- If your service handles multiple requests to **onStartCommand()** concurrently, you shouldn't stop the service when you're done processing a start request, as you might have received a new start request (stopping at the end of the first request would terminate the second one). To avoid this problem, you can use **stopSelf(int)**to ensure that your request to stop the service is always based on the most recent start request. That is, when you call **stopSelf(int)**, you pass the ID of the start request (the startId delivered to **onStartCommand())** to which your stop request corresponds. Then, if the service receives a new start request before you are able to call **stopSelf(int)**, the ID doesn't match and the service doesn't stop.

2.4 Broadcast Receivers

- A broadcast receiver is a dormant component of the Android system. Only an Intent (for which it is registered) can bring it into action. The Broadcast Receiver's job is to pass a notification to the user, in case a specific event occurs.
- Using a Broadcast Receiver, applications can register for a particular event. Once the event occurs, the system will notify all the registered applications.
- There are following two important steps to make BroadcastReceiver works for the system broadcasted intents −
    - Creating the Broadcast Receiver.
    - Registering Broadcast Receiver

2.4.1 Creating Broadcast Receiver

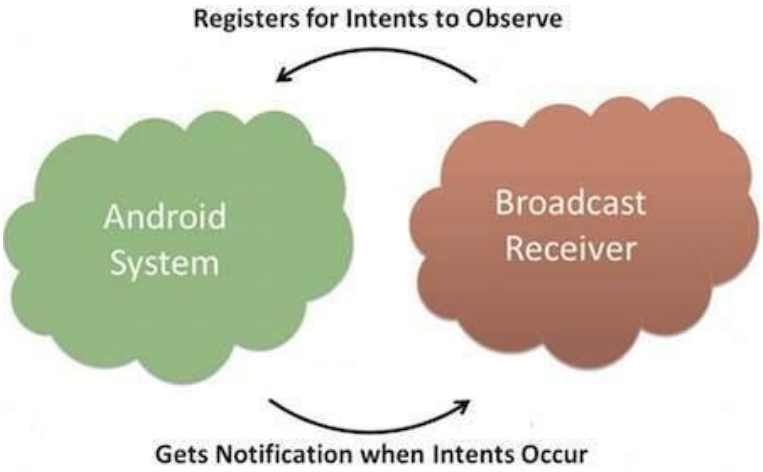
A broadcast receiver is implemented as a subclass of BroadcastReceiverclass and overriding the onReceive() method where each message is received as a Intent object parameter.

public class MyReceiver extends BroadcastReceiver {

```
    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "Intent Detected.", Toast.LENGTH_LONG).show();
    }
}
```

2.4.2 Registering Broadcast Receiver

An application listens for specific broadcast intents by registering a broadcast receiver in AndroidManifest.xml file. Consider we are going to register MyReceiver for system generated event ACTION_BOOT_COMPLETED which is fired by the system once the Android system has completed the boot process.




```
<application
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >
    <receiver android:name="MyReceiver">
        <intent-filter>
        <action android:name="android.intent.action.BOOT_COMPLETED">
        </action>
      </intent-filter>
      </receiver>
</application>
```

- Now whenever your Android device gets booted, it will be intercepted by BroadcastReceiver *MyReceiver* and implemented logic inside *onReceive()* will be executed.
- For instance, a Broadcast receiver triggers battery Low notification that you see on your mobile screen.
- Other instances caused by a Broadcast Receiver are new friend notifications, new friend feeds, new message etc. on your Facebook app.
- In fact, you see broadcast receivers at work all the time. Notifications like incoming messages, WiFi Activated/Deactivated message etc. are all real-time announcements of what is happening in the Android system and the applications.
- There are several system generated events defined as final static fields in the **Intent** class. The following table lists a few important system events.

Sr.No	Event Constant & Description
1	android.intent.action.BATTERY_CHANGED Sticky broadcast containing the charging state, level, and other information about the battery.
2	android.intent.action.BATTERY_LOW Indicates low battery condition on the device.
3	android.intent.action.BATTERY_OKAY Indicates the battery is now okay after being low.
4	android.intent.action.BOOT_COMPLETED This is broadcast once, after the system has finished booting.
5	android.intent.action.BUG_REPORT Show activity for reporting a bug.
6	android.intent.action.CALL Perform a call to someone specified by the data.

7	android.intent.action.CALL_BUTTON The user pressed the "call" button to go to the dialer or other appropriate UI for placing a call.
8	android.intent.action.DATE_CHANGED The date has changed.
9	android.intent.action.REBOOT Have the device reboot.

2.4.3 Implementing the Broadcast Receiver

You need to follow these steps to implement a broadcast receiver:

1) Create a subclass of Android's BroadcastReceiver

2) Implement the onReceive() method: In order for the notification to be sent, an onReceive() method has to be implemented. Whenever the event for which the receiver is registered occurs, onReceive() is called. For instance, in case of battery low notification, the receiver is registered to Intent.ACTION_BATTERY_LOW event. As soon as the battery level falls below the defined level, this onReceive() method is called.

Following are the two arguments of the onReceive() method:

- Context: This is used to access additional information, or to start services or activities.
- Intent: The Intent object is used to register the receiver.

Security

As the broadcast receivers have a global work-space, security is very important concern here. If you do not define the limitations and filters for the registered receivers, other applications can abuse them.
Here are a few limitations that might help:

- Whenever you publish a receiver in your application's manifest, make it unavailable to external applications by using android: exported="false". You might think that specifying Intent filters while publishing the receiver would do the task for you, when in reality they are not enough.
- When you send a broadcast, it is possible for the external applications too to receive them. This can be prevented by specifying a few limitations.

- Similarly, when you register your receiver using registerReceiver, any application may send it broadcasts. This can be prevented using permissions as well.

2.4.4 Broadcasting Custom Intents

- If you want your application itself should generate and send custom intents then you will have to create and send those intents by using the sendBroadcast() method inside your activity class. If you use the sendStickyBroadcast(Intent) method, the Intent is sticky, meaning the Intent you are sending stays around after the broadcast is complete.

```
public void broadcastIntent(View view) {
   Intent intent = new Intent();
   intent.setAction("com.tutorialspoint.CUSTOM_INTENT");
   sendBroadcast(intent);
}
```

- This intent com.tutorialspoint.CUSTOM_INTENT can also be registered in similar way as we have regsitered system generated intent.

```
<application
   android:icon="@drawable/ic_launcher"
   android:label="@string/app_name"
   android:theme="@style/AppTheme" >
   <receiver android:name="MyReceiver">

      <intent-filter>
         <action android:name="com.tutorialspoint.CUSTOM_INTENT">
         </action>
      </intent-filter>
      </receiver>
</application>
```

2.4.5 Classes of Broadcasts

The two major classes of broadcasts are:

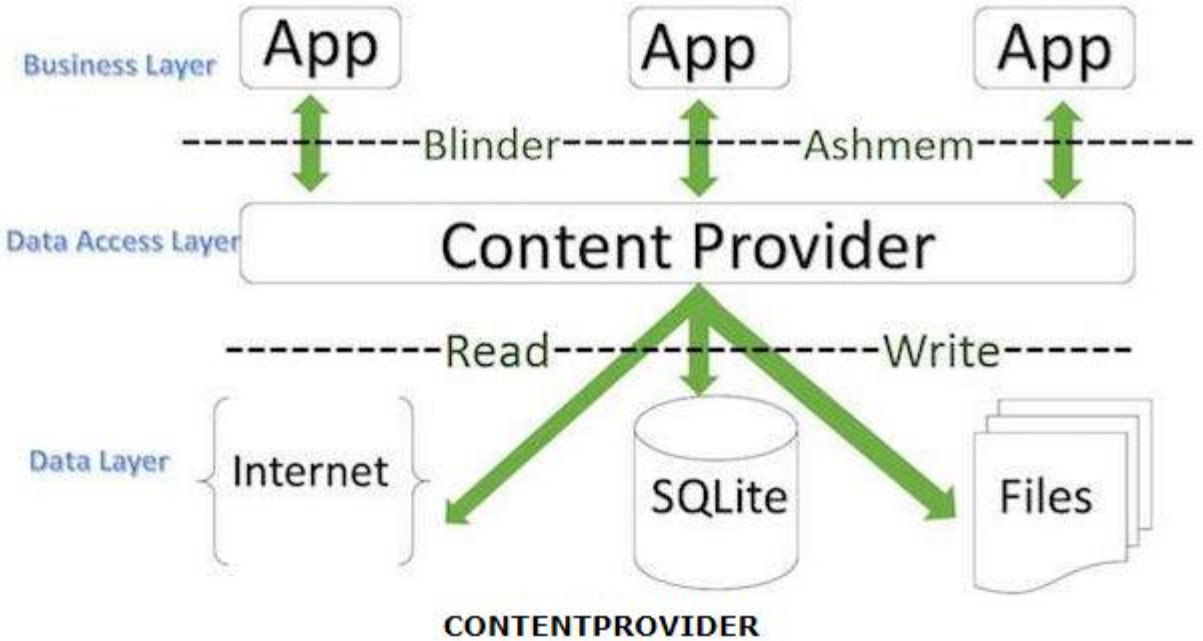
1) Ordered Broadcasts: These broadcasts are synchronous, and therefore follow a specific order. The order is defined using **android: priority attribute**. The **receivers with greater priority would receive the broadcast first**. In case there are receivers with same priority levels, the broadcast would not follow an order. Each receiver (when it receives the broadcast) can either pass on the notification to the next one, or abort the broadcast completely. On abort, the notification would not be passed on to the receivers next in line.

2) Normal Broadcasts: Normal broadcasts are **not orderly**. Therefore, the registered receivers often **run all at the same time**. This is very efficient, but the **Receivers are unable to utilize the results**.

Sometimes to avoid system overload, the system delivers the broadcasts one at a time, even in case of normal broadcasts. However, the receivers still cannot use the results.

2.5 Content Provider

- A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the **ContentResolver** class.
- A content provider manages a shared set of app data that you can store in the file system, in a SQLite database, on the web, or on any other persistent storage location that your app can access. Through the content provider, other apps can query or modify the data if the content provider allows it.



CONTENTPROVIDER

- For example, the Android system provides a content provider that manages the user's contact information. As such, any app with the proper permissions can query the content provider, such as Contacts.
- 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 a 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 providers are also useful for reading and writing data that is private to your app and not shared.
- A content provider is implemented as a subclass of Content Provider and must implement a standard set of APIs that enable other apps to perform transactions.

Sr.No	Part & Description
1	Prefix This is always set to content://
2	Authority This specifies the name of the content provider, for example contacts, browser etc. For third-party content providers, this could be the fully qualified name, such as com.tutorialspoint.statusprovider
3	data_type This indicates the type of data that this particular provider provides. For example, if you are getting all the contacts from the Contactscontent provider, then the data path would be people and URI would look like thiscontent://contacts/people
4	Id This specifies the specific record requested. For example, if you are looking for contact number 5 in the Contacts content provider then URI would look like this content://contacts/people/5.

2.5.1 Content URIs

- To query a content provider, you specify the query string in the form of a URI which has following format −
- <prefix>://<authority>/<data_type>/<id>

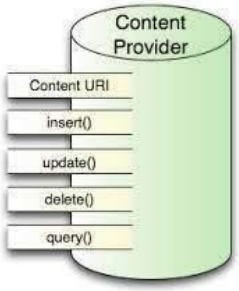
- Here is the detail of various parts of the URI −

2.5.2 Create Content Provider

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

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

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



CONTENTPROVIDER

- **onCreate()** This method is called when the provider is started.
- **query()** This method receives a request from a client. The result is returned as a Cursor object.

- **insert()**This method inserts a new record into the content provider.
- **delete()** This method deletes an existing record from the content provider.
- **update()** This method updates an existing record from the content provider.
- **getType()** This method returns the MIME type of the data at the given URI.

Example

Step	Description
1	You will use Android StudioIDE to create an Android application and name it as My Application under a package com.example.MyApplication, with blank Activity.
2	Modify main activity file MainActivity.java to add two new methods onClickAddName() and onClickRetrieveStudents().
3	Create a new java file called StudentsProvider.java under the package com.example.MyApplication to define your actual provider and associated methods.
4	Register your content provider in your AndroidManifest.xml file using <provider.../> tag
5	Modify the default content of res/layout/activity_main.xml file to include a small GUI to add students records.
6	No need to change string.xml.Android studio take care of string.xml file.
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/com.example.MyApplication/MainActivity.java. This file can include each of the fundamental life cycle methods. We have added two new methods onClickAddName() and onClickRetrieveStudents() to handle user interaction with the application.

```
package com.example.MyApplication;

import android.net.Uri;

import android.os.Bundle;

import android.app.Activity;

import android.content.ContentValues;

import android.content.CursorLoader;
```

```
import android.database.Cursor;
import android.view.Menu;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends Activity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
  }
  public void onClickAddName(View view) {
    // Add a new student record
    ContentValues values = new ContentValues();
    values.put(StudentsProvider.NAME,
      ((EditText)findViewById(R.id.editText2)).getText().toString());
    values.put(StudentsProvider.GRADE,
      ((EditText)findViewById(R.id.editText3)).getText().toString());
    Uri uri = getContentResolver().insert(
      StudentsProvider.CONTENT_URI, values);
    Toast.makeText(getBaseContext(),
      uri.toString(), Toast.LENGTH_LONG).show();
  }
  public void onClickRetrieveStudents(View view) {
    // Retrieve student records
    String URL = "content://com.example.MyApplication.StudentsProvider";
    Uri students = Uri.parse(URL);
    Cursor c = managedQuery(students, null, null, null, "name");
```

```
    if (c.moveToFirst()) {
      do{
        Toast.makeText(this,
          c.getString(c.getColumnIndex(StudentsProvider._ID)) +
            ", " +  c.getString(c.getColumnIndex( StudentsProvider.NAME)) +
              ", " + c.getString(c.getColumnIndex( StudentsProvider.GRADE)),
        Toast.LENGTH_SHORT).show();
      } while (c.moveToNext());
    }
  }
}
```

Create new file StudentsProvider.java under com.example.MyApplicationpackage.

You can write activities against update and delete operations by providing callback functions in **MainActivity.java** file and then modify user interface to have buttons for update and deleted operations in the same way as we have done for add and read operations.

This way you can use existing Content Provider like Address Book or you can use Content Provider concept in developing nice database oriented applications where you can perform all sort of database operations like read, write, update and delete as explained above in the example

2.5.3 Content Resolver

- The Content Resolver is the single, global instance in your application that provides access to your (and other applications') content providers.
- The Content Resolver behaves exactly as its name implies: it accepts requests from clients, and resolves these requests by directing them to the content provider with a distinct authority. To do this, the Content Resolver stores a mapping from authorities to Content Providers. This design is important, as it allows a simple and secure means of accessing other applications' Content Providers.
- The Content Resolver includes the CRUD (create, read, update, delete) methods corresponding to the abstract methods (insert, query, update, delete) in the Content Provider class.
- The Content Resolver does not know the implementation of the Content Providers it is interacting with (nor does it need to know); each method is passed an URI that specifies the Content Provider to interact with.
- Each android application can be a content provider. For example, android phone contacts, short message system and android media library.

- To get data from a content provider, you need to use a **ContentResolver** instance in your app.

Generally the **ContentResolver** instance can be obtained by **Activity**'s **getContentResolver()** method.

```
ContentResolver contentResolver = getContentResolver();
```

- Then you can invoke **ContentResolver**'s method to insert, delete, update and query data that another content provider shared. This is something like SQLite database operation.

2.5.3.1 ContentResolver Methods

Before process data operaion, you should first get the content provider URI instance using below method.

```
Uri contentUri = Uri.parse("content://com.dev2qa.example.provider/userinfo");
```

Insert Data To Content Provider

insert(Uri providerUri, ContentValues values)

```
Uri contentUri = Uri.parse("content://....");

ContentValues contentValues = new ContentValues();

contentValues.put("column1", value1);

contentValues.put("column2", value2);

getContentResolver().insert(contentUri, contentValues);
```

Update Content Provider Data

update(Uri providerUri, ContentValues values, String whereClause, String conditionValueArr[])

```
Uri contentUri = Uri.parse("content://....");

ContentValues contentValues = new ContentValues();

contentValues.put("userName", userName);

contentValues.put("password", passwrod);

String whereClause = "id = ?";

String placeHolderValueArr[] = {"1"}

getContentResolver().update(contentUri, contentValues, whereClause , placeHolderValueArr);
```

Delete Content Provider Data

delete(Uri providerUri, String whereClause, String conditionValueArr[])

```
Uri contentUri = Uri.parse("content://....");

String whereClause = "id = ?";
```

```
String placeHolderValueArr[] = {"1"}

getContentResolver().delete(contentUri, whereClause , placeHolderValueArr);
```

Query Content Provider Data

query(Uri uri, String columnArray[], String whereClause, String wherePlaceHolderValue[], String orderByClause)

The query method return a android.database.Cursor object, if it is not null, then use it's moveToFirst() method to move to the first row.

Then loop in the cursor to get each row use it's moveToNext()method.

```
Uri contentUri = Uri.parse("content://....");

Cursor cursor = getContentResolver().query(contentUri , null, null, null, null);

if(cursor!=null)
{
   cursor.moveToFirst();

   // Loop in the cursor to get each row.
   do{
      // Get column 1 value.
      int column1Index= cursor.getColumnIndex("column1");
      String column1Value= cursor.getString(column1Index);

      // Get column 2 value.
      int column2Index= cursor.getColumnIndex("column2");
      String column2Value= cursor.getString(column2Index);
   }while(cursor.moveToNext());
```

2.6 SQLite

- SQLite is a opensource SQL database that stores data to a text file on a device. Android comes in with built in SQLite database implementation.
- SQLite supports all the relational database features. In order to access this database, you don't need to establish any kind of connections for it like JDBC,ODBC e.t.c

Database - Package

- The main package is android.database.sqlite that contains the classes to manage your own databases

2.6.1 Database – Creation

SQLite database storage classes

Storage classes refer to how stuff is stored within the database. SQLite databases store values in one of five possible storage classes:

- NULL – For null value.
- INTEGER – For integers containing as much as 8 bytes (thats from byte to long).

- REAL – Numbers with floating point.
- TEXT – Text strings, stored using the database encoding (UTF-8 or UTF-16).
- BLOB – Binary data, stored exactly as input.

- In order to create a database you just need to call this method openOrCreateDatabase with your database name and mode as a parameter. It returns an instance of SQLite database which you have to receive in your own object.Its syntax is given below

```
SQLiteDatabase mydatabase = openOrCreateDatabase("your database name",MODE_PRIVATE,null);
```

Apart from this , there are other functions available in the database package , that does this job. They are listed below

Sr.No	Method & Description
1	**openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler)** This method only opens the existing database with the appropriate flag mode. The common flags mode could be OPEN_READWRITE OPEN_READONLY
2	**openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)** It is similar to the above method as it also opens the existing database but it does not define any handler to handle the errors of databases
3	**openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)** It not only opens but create the database if it not exists. This method is equivalent to openDatabase method.
4	**openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)** This method is similar to above method but it takes the File object as a path rather then a string. It is equivalent to file.getPath()

2.6.2 Database – Insertion

We can create table or insert data into table using execSQL method defined in SQLiteDatabase class. Its syntax is given below

mydatabase.execSQL("CREATE TABLE IF NOT EXISTS TutorialsPoint(Username VARCHAR,Password VARCHAR);");

mydatabase.execSQL("INSERT INTO TutorialsPoint VALUES('admin','admin');");

This will insert some values into our table in our database. Another method that also does the same job but take some additional parameter is given below

Sr.No	Method & Description
1	**execSQL(String sql, Object[] bindArgs)** This method not only insert data , but also used to update or modify already existing data in database using bind arguments

2.6.3 Database – Fetching

We can retrieve anything from database using an object of the Cursor class. We will call a method of this class called rawQuery and it will return a resultset with the cursor pointing to the table. We can move the cursor forward and retrieve the data.

Cursor resultSet = mydatbase.rawQuery("Select * from TutorialsPoint",null);

resultSet.moveToFirst();

String username = resultSet.getString(0);

String password = resultSet.getString(1);

There are other functions available in the Cursor class that allows us to effectively retrieve the data. That includes

Sr.No	Method & Description
1	**getColumnCount()** This method return the total number of columns of the table.
2	**getColumnIndex(String columnName)** This method returns the index number of a column by specifying the name of the column

3	**getColumnName(int columnIndex)** This method returns the name of the column by specifying the index of the column
4	**getColumnNames()** This method returns the array of all the column names of the table.
5	**getCount()** This method returns the total number of rows in the cursor
6	**getPosition()** This method returns the current position of the cursor in the table
7	**isClosed()** This method returns true if the cursor is closed and return false otherwise

2.6.4 Database - Helper class

For managing all the operations related to the database , an helper class has been given and is called SQLiteOpenHelper. It automatically manages the creation and update of the database. Its syntax is given below

```
  public class DBHelper extends SQLiteOpenHelper {

  public DBHelper(){

    super(context,DATABASE_NAME,null,1);

  }

  public void onCreate(SQLiteDatabase db) {}

  public void onUpgrade(SQLiteDatabase database, int oldVersion, int newVersion) {}

}
```

2.6.5 Sample database application

Following is the content of Database class DBHelper.java

```
package com.example.sairamkrishna.myapplication;


import java.util.ArrayList;
```

```
import java.util.HashMap;
import java.util.Hashtable;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.DatabaseUtils;
import android.database.sqlite.SQLiteOpenHelper;
import android.database.sqlite.SQLiteDatabase;

public class DBHelper extends SQLiteOpenHelper {

   public static final String DATABASE_NAME = "MyDBName.db";
   public static final String CONTACTS_TABLE_NAME = "contacts";
   public static final String CONTACTS_COLUMN_ID = "id";
   public static final String CONTACTS_COLUMN_NAME = "name";
   public static final String CONTACTS_COLUMN_EMAIL = "email";
   public static final String CONTACTS_COLUMN_STREET = "street";
   public static final String CONTACTS_COLUMN_CITY = "place";
   public static final String CONTACTS_COLUMN_PHONE = "phone";
   private HashMap hp;

   public DBHelper(Context context) {
      super(context, DATABASE_NAME , null, 1);
   }

   @Override
   public void onCreate(SQLiteDatabase db) {
      // TODO Auto-generated method stub
```

```
    db.execSQL(
      "create table contacts " +
      "(id integer primary key, name text,phone text,email text, street text,place text)"
    );
  }

  @Override
  public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    // TODO Auto-generated method stub
    db.execSQL("DROP TABLE IF EXISTS contacts");
    onCreate(db);
  }

  public boolean insertContact (String name, String phone, String email, String street,String place) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put("name", name);
    contentValues.put("phone", phone);
    contentValues.put("email", email);
    contentValues.put("street", street);
    contentValues.put("place", place);
    db.insert("contacts", null, contentValues);
    return true;
  }

  public Cursor getData(int id) {
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor res =  db.rawQuery( "select * from contacts where id="+id+"", null );
```

```
    return res;
  }


  public int numberOfRows(){
    SQLiteDatabase db = this.getReadableDatabase();
    int numRows = (int) DatabaseUtils.queryNumEntries(db, CONTACTS_TABLE_NAME);
    return numRows;
  }


  public boolean updateContact (Integer id, String name, String phone, String email, String street,String
place) {
    SQLiteDatabase db = this.getWritableDatabase();
    ContentValues contentValues = new ContentValues();
    contentValues.put("name", name);
    contentValues.put("phone", phone);
    contentValues.put("email", email);
    contentValues.put("street", street);
    contentValues.put("place", place);
    db.update("contacts", contentValues, "id = ? ", new String[] { Integer.toString(id) } );
    return true;
  }


  public Integer deleteContact (Integer id) {
    SQLiteDatabase db = this.getWritableDatabase();
    return db.delete("contacts",
    "id = ? ",
    new String[] { Integer.toString(id) });
  }
```

```
  public ArrayList<String> getAllCotacts() {
    ArrayList<String> array_list = new ArrayList<String>();


    //hp = new HashMap();
    SQLiteDatabase db = this.getReadableDatabase();
    Cursor res =  db.rawQuery( "select * from contacts", null );
    res.moveToFirst();


    while(res.isAfterLast() == false){
      array_list.add(res.getString(res.getColumnIndex(CONTACTS_COLUMN_NAME)));
      res.moveToNext();
    }
    return array_list;
  }
}
```

2.6.6 Situations Where SQLite Works Well

- **Embedded devices and the internet of things**

Because an SQLite database requires no administration, it works well in devices that must operate without expert human support. SQLite is a good fit for use in cellphones, set-top boxes, televisions, game consoles, cameras, watches, kitchen appliances, thermostats, automobiles, machine tools, airplanes, remote sensors, drones, medical devices, and robots: the "internet of things".

- **Application file format**

SQLite is often used as the on-disk file format for desktop applications such as version control systems, financial analysis tools, media cataloging and editing suites, CAD packages, record keeping programs, and so forth. The traditional File/Open operation calls sqlite3_open() to attach to the database file. Updates happen automatically as application content is revised so the File/Save menu option becomes superfluous. The File/Save_As menu option can be implemented using the backup API.

- **Websites**

SQLite works great as the database engine for most low to medium traffic websites (which is to say, most websites). The amount of web traffic that SQLite can handle depends on how heavily the website uses its database. Generally speaking, any site that gets fewer than 100K hits/day should work fine with SQLite. The 100K hits/day figure is a conservative estimate, not a hard upper bound. SQLite has been demonstrated to work with 10 times that amount of traffic.

- **Data analysis**

People who understand SQL can employ the sqlite3 command-line shell (or various third-party SQLite access programs) to analyze large datasets. Raw data can be imported from CSV files, then that data can be sliced and diced to generate a myriad of summary reports. More complex analysis can be done using simple scripts written in Tcl or Python (both of which come with SQLite built-in) or in R or other languages using readily available adaptors. Possible uses include website log analysis, sports statistics analysis, compilation of programming metrics, and analysis of experimental results. Many bioinformatics researchers use SQLite in this way.

- **Cache for enterprise data**

Many applications use SQLite as a cache of relevant content from an enterprise RDBMS. This reduces latency, since most queries now occur against the local cache and avoid a network round-trip. It also reduces the load on the network and on the central database server. And in many cases, it means that the client-side application can continue operating during network outages.

- **Server-side database**

Systems designers report success using SQLite as a data store on server applications running in the datacenter, or in other words, using SQLite as the underlying storage engine for an application-specific database server.

- **Data transfer format**

Because an SQLite database is a single compact file in a well-defined cross-platform format, it is often used as a container for transferring content from one system to another. The sender gathers content into an SQLite database file, transfers that one file to the receiver, then the receiver uses SQL to extract the content as needed.

- **File archive and/or data container**

The SQLite Archive idea shows how SQLite can be used as a substitute for ZIP archives or Tarballs. An archive of files stored in SQLite is only very slightly larger, and in some cases actually smaller, than the equivalent ZIP archive. And an SQLite archive features incremental and atomic updating and the ability to store much richer metadata.

SQLite is a good solution for any situation that requires bundling diverse content into a self-contained and self-describing package for shipment across a network. Content is encoding in a well-defined, cross-platform, and stable file format. The encoding is efficient, and receivers can extract small subsets of the content without having to read and parse the entire file.

- **Replacement for ad hoc disk files**

Many programs use fopen(), fread(), and fwrite() to create and manage files of data in home-grown formats. SQLite works particularly well as a replacement for these *ad hoc* data files. Contrary to intuition, SQLite can be faster than the filesystem for reading and writing content to disk.

- **Internal or temporary databases**

For programs that have a lot of data that must be sifted and sorted in diverse ways, it is often easier and quicker to load the data into an in-memory SQLite database and use queries with joins and ORDER BY clauses to extract the data in the form and order needed rather than to try to code the same operations manually.

- **Stand-in for an enterprise database during demos or testing**

Client applications typically use a generic database interface that allows connections to various SQL database engines. It makes good sense to include SQLite in the mix of supported databases and to statically link the SQLite engine in with the client. That way the client program can be used standalone with an SQLite data file for testing or for demonstrations.

- **Education and Training**

Because it is simple to setup and use (installation is trivial: just copy the **sqlite3** or **sqlite3.exe** executable to the target machine and run it) SQLite makes a good database engine for use in teaching SQL. Students

can easily create as many databases as they like and can email databases to the instructor for comments or grading.

- **Experimental SQL language extensions**

The simple, modular design of SQLite makes it a good platform for prototyping new, experimental database language features or ideas.

2.6.7 Data Analysis

Data Analysis is a process of collecting, transforming, cleaning, and modeling data with the goal of discovering the required information. The results so obtained are communicated, suggesting conclusions, and supporting decision-making. Data visualization is at times used to portray the data for the ease of discovering the useful patterns in the data. Data Analysis Process consists of the following phases that are iterative in nature −

- Data Requirements Specification
- Data Collection
- Data Processing
- Data Cleaning
- Data Analysis
- Communication

UNIT IV

4.1 ANDROID LAYOUTS

4.1.1 What is a Layout?

Layout defines a visual structure of an Activity (or app widget). It may be considered as a set of rules according to which controls (buttons, text fields, input fields etc.) are placed on the *View*.

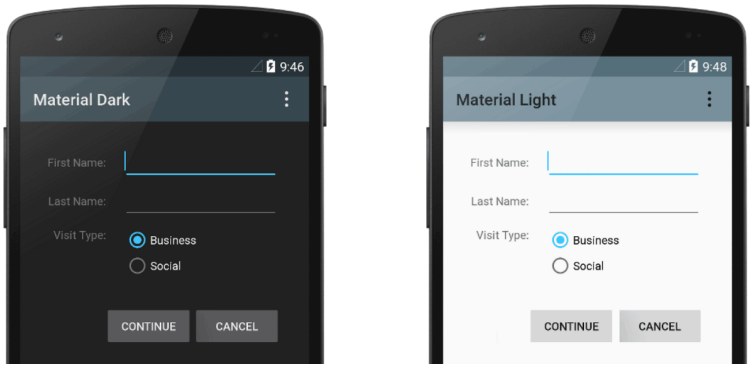
4.1.2 Layouts structure

Basically, user interface in Android apps is built using *Layouts*. Each *Layout* is a subclass of ViewGroup class, which derives from View class, which is the basic UI building block. View is the base class for buttons, text fields etc.,

A ***style*** is a collection of attributes that specify the appearance for a single View. A style can specify attributes such as font color, font size, background color, and much more.

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

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



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

4.1.3 Create and apply a style

To create a new style or theme, open your project's res/values/styles.xml file. For each style you want to create, follow these steps:

- Add a <style> element with a name that uniquely identifies the style.
- Add an <item> element for each style attribute you want to define.

The name in each item specifies an attribute you would otherwise use as an XML attribute in your layout. The value in the <item> element is the value for that attribute.

For example, if you define the following style:

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <stylename="GreenText"parent="TextAppearance.AppCompat">
        <item name="android:textColor">#00FF00</item>
    </style>
</resources>
```

You can apply the style to a view as follows:

```
<TextView
    style="@style/GreenText"
    ... />
```

Each attribute specified in the style is applied to that view if the view accepts it. The view simply ignores any attributes that it does not accept.

4.2 Layout Attributes

Each layout has a set of attributes which define the visual properties of that layout. There are few common attributes among all the layouts and there are other attributes which are specific to that layout. Following are common attributes and will be applied to all the layouts:

Sr.No	Attribute & Description
1	android:id This is the ID which uniquely identifies the view.
2	android:layout_width This is the width of the layout.
3	android:layout_height This is the height of the layout
4	android:layout_marginTop This is the extra space on the top side of the layout.
5	android:layout_marginBottom This is the extra space on the bottom side of the layout.
6	android:layout_marginLeft This is the extra space on the left side of the layout.
7	android:layout_marginRight This is the extra space on the right side of the layout.
8	android:layout_gravity This specifies how child Views are positioned.
9	android:layout_weight This specifies how much of the extra space in the layout should be allocated to the View.
10	android:layout_x This specifies the x-coordinate of the layout.
11	android:layout_y This specifies the y-coordinate of the layout.
12	android:layout_width This is the width of the layout.
13	android:layout_width This is the width of the layout.
14	android:paddingLeft This is the left padding filled for the layout.
15	android:paddingRight This is the right padding filled for the layout.

16	android:paddingTop This is the top padding filled for the layout.
17	android:paddingBottom This is the bottom padding filled for the layout.

Here width and height are the dimension of the layout/view which can be specified in terms of dp (Density-independent Pixels), sp(Scale-independent Pixels), pt (Points which is 1/72 of an inch), px(Pixels), mm (Millimeters) and finally in (inches).

android:layout_width=wrap_content tells your view to size itself to the dimensions required by its content.

android:layout_width=fill_parent tells your view to become as big as its parent view.

Gravity attribute plays important role in positioning the view object and it can take one or more (separated by '|') of the following constant values.

4.3 Layout styles

There are number of Layouts provided by Android which you will use in almost all the Android applications to provide different view, look and feel.

Sr.No	Layout & Description
1	Linear Layout LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.
2	Relative Layout RelativeLayout is a view group that displays child views in relative positions.
3	Table Layout TableLayout is a view that groups views into rows and columns.
4	Absolute Layout AbsoluteLayout enables you to specify the exact location of its children.
5	Frame Layout The FrameLayout is a placeholder on screen that you can use to display a single view.

6	List View ListView is a view group that displays a list of scrollable items.
7	Grid View GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

4.4 Linear Layout

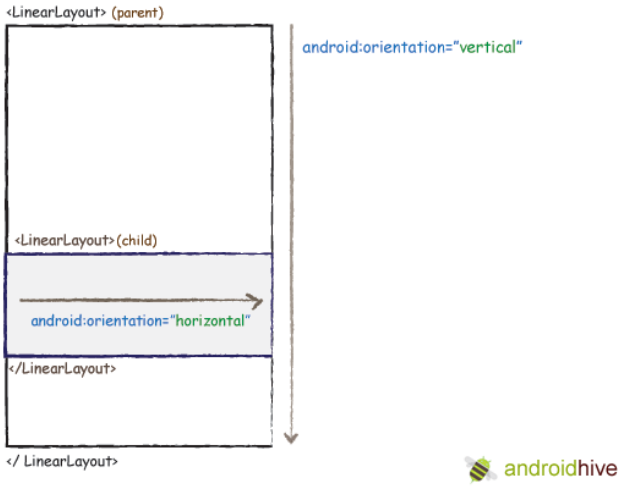
In a linear layout, like the name suggests, all the elements are displayed in a linear fashion(below is an example of the linear layouts), either **Horizontally** or **Vertically** and this behavior is set in **android:orientation** which is an attribute of the node LinearLayout.

Example of Vertical layout snippet

<LinearLayoutandroid:orientation="vertical"> .... </LinearLayout>

Example of Horizontal layout snippet

<LinearLayoutandroid:orientation="horizontal"> .... </LinearLayout>



Here are the steps you need to follow to create them

1. Create a new project **File -> New -> Android Project**

2. In Package Explorer right click on **res/layout** folder and create a new Android XML File and name it as you wish. I am naming it as "**linear_layout.xml**"

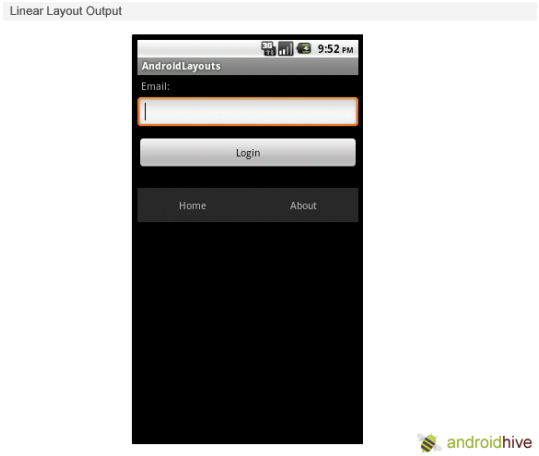
res/layout -> Right Click -> New -> Android XML File

3. Now open newly created xml file (in my case "**linear_layout.xml**") and type the code.

To set this newly created view as the initial view of your app, Open your MainActivity.java file. You would see the following line inside the **onCreate** function **setContentView(R.layout.main)**. Change **R.layout.main** to **R.layout.yourlinearviewname**. In my case its **R.layout.linear_layout**

```
package com.example.androidlayouts;

import android.app.Activity;

import android.os.Bundle;

public class AndroidLayoutsActivity extends Activity {

    @Override

    public void onCreate(Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);

        setContentView(R.layout.linear_layout);

    }

}
```

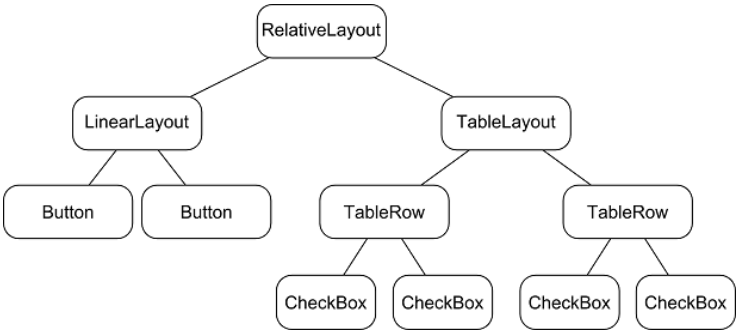
5. To run the application, **right click on the project -> Run As -> 1. Android Application**. You should see your newly created linear layout in the emulator.



4.5 Relative Layout

In a relative layout every element arranges itself relative to other elements or a parent element. As an example, lets consider the layout defined below. The "**Cancel**" button is placed relatively, to the **right of** the "**Login**" button **parallely**. Here is the code snippet that achieves the mentioned alignment (Right of Login button parallely)

The following diagram presents this hierarchy based on *RelativeLayout*:

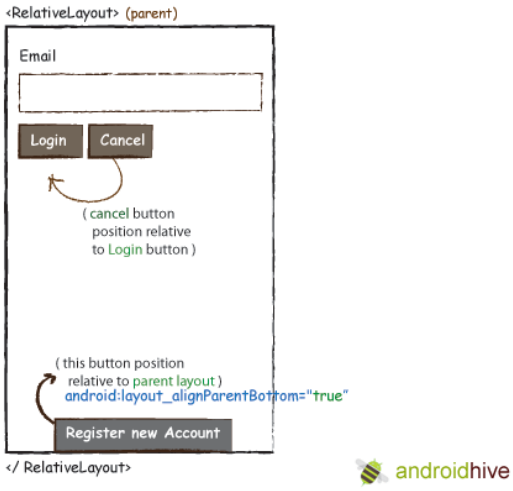


Example code snippet

<Button android:id="@+id/btnLogin" ..></Button>

<Button android:layout_toRightOf="@id/btnLogin"

android:layout_alignTop="@id/btnLogin" ..></Button>



Here are the steps to create a relative layout

1. Create a new project File -> New -> Android Project

2. In Package Explorer right click on **res/layout** folder and create a new Android XML File and name it as you wish. I am naming it as "relative_layout.xml"

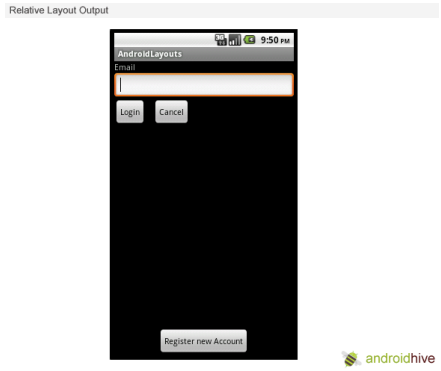
res/layout -> Right Click -> New -> Android XML File

3. Now open newly created xml file (in my case "**relative_layout.xml**") and type the following code.

4. Same like before open your MainActivity.java file and set the layout to your newly created relative layout file. In my case its **R.layout.relative_layout**

setContentView(R.layout.relative_layout);

5. To run the application, **right click on the project -> Run As -> 1. Android Application**. You should see your newly created relative layout in the emulator.



4.6 Table Layout

Table layouts in Android works in the same way HTML table layouts work. You can divide your layouts into **rows** and **columns**. Its very easy to understand.

1.Create a new project **File -> New -> Android Project**

2. In Package Explorer right click on res/layout folder and create a new Android XML File and name it as you wish. I am naming it as "table_layout.xml"

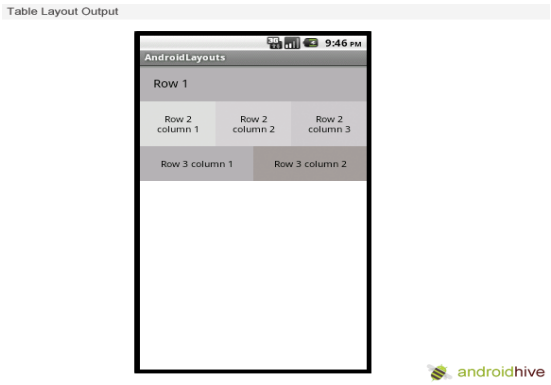
res/layout -> Right Click -> New -> Android XML File

3. Now open newly created xml file (in my case "***table_layout.xml***") and type the code.

4. Same like before open your MainActivity.java file and set the layout to your newly created table layout file. In my case its **R.layout.table_layout**

setContentView(R.layout.table_layout);

5. To run the application, **right click on the project -> Run As -> 1. Android Application**. You should see your newly created table layout in the emulator.



Following are the important attributes specific to TableLayout −

Sr.No.	Attribute & Description
1	android:id This is the ID which uniquely identifies the layout.
2	android:collapseColumns This specifies the zero-based index of the columns to collapse. The column indices must be separated by a comma: 1, 2, 5.
3	android:shrinkColumns The zero-based index of the columns to shrink. The column indices must be separated by a comma: 1, 2, 5.
4	android:stretchColumns The zero-based index of the columns to stretch. The column indices must be separated by a comma: 1, 2, 5.

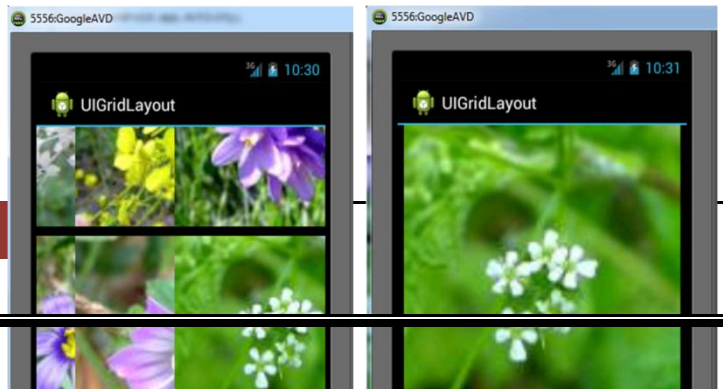
4.7 Grid View

It is a ViewGroup that displays items in a two-dimensional, scrollable grid. The grid items are automatically inserted to the layout using a ListAdapter.

An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter can be used to supply the data to like spinner, list view, grid view etc.

The ListView and GridView are subclasses of AdapterView and they can be populated by binding them to an Adapter, which retrieves data from an external source and creates a View that represents each data entry.

Demonstration of Grid View .



4.7.1 View Identification

A view object may have a unique ID assigned to it which will identify the View uniquely within the tree. The syntax for an ID, inside an XML tag is −

android:id="@+id/my_button"

Following is a brief description of @ and + signs −

The at-symbol (@) at the beginning of the string indicates that the XML parser should parse and expand the rest of the ID string and identify it as an ID resource.

The plus-symbol (+) means that this is a new resource name that must be created and added to our resources. To create an instance of the view object and capture it from the layout, use the following −

ButtonmyButton=(Button)findViewById(R.id.my_button);

4.8 Frame Layout

Frame Layout is designed to block out an area on the screen to display a single item. Generally, Frame Layout should be used to hold a single child view, because it can be difficult to organize child views in a way that's scalable to different screen sizes without the children overlapping each other.

You can, however, add multiple children to a Frame Layout and control their position within the Frame Layout by assigning gravity to each child, using the android:layout_gravity attribute.

4.8.1 Frame Layout Attributes

Following are the important attributes specific to Frame Layout −

Sr.No	Attribute & Description
1	android:id This is the ID which uniquely identifies the layout.
2	android:foreground This defines the drawable to draw over the content and possible values may be a color value, in the form of "#rgb", "#argb", "#rrggbb", or "#aarrggbb".
3	android:foregroundGravity Defines the gravity to apply to the foreground drawable. The gravity defaults to fill. Possible values are top, bottom, left, right, center, center_vertical, center_horizontal etc.
4	android:measureAllChildren Determines whether to measure all children or just those in the VISIBLE or INVISIBLE state when measuring. Defaults to false.

4.9 Menus

Menus are a common user interface component in many types of applications. To provide a familiar and consistent user experience, you should use the Menu APIs to present user actions and other options in your activities.

Beginning with Android 3.0 (API level 11), Android-powered devices are no longer required to provide a dedicated *Menu* button. With this change, Android apps should migrate away from a dependence on the traditional 6-item menu panel and instead provide an app bar to present common user actions.

Although the design and user experience for some menu items have changed, the semantics to define a set of actions and options is still based on the Menu APIs. This guide shows how to create the three fundamental types of menus or action presentations on all versions of Android:

4.10 Options menu

The options menu is the primary collection of menu items for an activity. It's where you should place actions that have a global impact on the app, such as "Search," "Compose email," and "Settings."

To specify the options menu for an activity, override onCreateOptionsMenu()(fragments provide their own onCreateOptionsMenu() callback). In this method, you can inflate your menu resource (defined in XML) into the Menu provided in the callback. For example:

JAVA

```
overridefunonCreateOptionsMenu(menu:Menu):Boolean{
    valinflater:MenuInflater=menuInflater
    inflater.inflate(R.menu.game_menu, menu)
    returntrue
}
```

You can also add menu items using add() and retrieve items with findItem() to revise their properties with MenuItem APIs.

If you've developed your application for Android 2.3.x and lower, the system calls onCreateOptionsMenu() to create the options menu when the user opens the menu for the first time. If you've developed for Android 3.0 and higher, the system calls onCreateOptionsMenu() when starting the activity, in order to show items to the app bar.

4.11 Contextual Menus

A contextual menu offers actions that affect a specific item or context frame in the UI. You can provide a context menu for any view, but they are most often used for items in a ListView, GridView, or other view collections in which the user can perform direct actions on each item.

There are two ways to provide contextual actions: In a floating context menu. A menu appears as a floating list of menu items (similar to a dialog) when the user performs a long-click (press and hold) on a view that declares support for a context menu. Users can perform a contextual action on one item at a time.

1. **Creating a floating context menu**

To provide a floating context menu:

Register the View to which the context menu should be associated by calling registerForContextMenu() and pass it the View.

If your activity uses a ListView or GridView and you want each item to provide the same context menu, register all items for a context menu by passing the ListView or GridView to registerForContextMenu().

Implement the onCreateContextMenu() method in your Activity or Fragment.

When the registered view receives a long-click event, the system calls your onCreateContextMenu() method. This is where you define the menu items, usually by inflating a menu resource. For example:

JAVA

```
overridefunonCreateContextMenu(menu:ContextMenu, v:View,
                menuInfo:ContextMenu.ContextMenuInfo){
    super.onCreateContextMenu(menu, v,menuInfo)
    valinflater:MenuInflater=menuInflater
    inflater.inflate(R.menu.context_menu, menu)
}
```

Implement onContextItemSelected().

When the user selects a menu item, the system calls this method so you can perform the appropriate action. For example:

JAVA

```
overridefunonContextItemSelected(item:MenuItem):Boolean{
    val info =item.menuInfoasAdapterView.AdapterContextMenuInfo
    returnwhen(item.itemId){
        R.id.edit->{
            editNote(info.id)
            true
        }
        R.id.delete->{
            deleteNote(info.id)
            true
        }
        else->super.onContextItemSelected(item)
    }
}
```

The getItemId() method queries the ID for the selected menu item, which you should assign to each menu item in XML using the android:id attribute, as shown in the section about Defining a Menu in XML.

In the contextual action mode. This mode is a system implementation of ActionMode that displays a *contextual action bar* at the top of the screen with action items that affect the selected item(s). When this mode is active, users can perform an action on multiple items at once (if your app allows it).

2. Using the contextual action mode

The contextual action mode is a system implementation of ActionMode that focuses user interaction toward performing contextual actions. When a user enables this mode by selecting an item, a *contextual action bar* appears at the top of the screen to present actions the user can perform on the currently selected item(s). While this mode is enabled, the user can select multiple items (if you allow it), deselect items, and continue to navigate within the activity (as much as you're willing to allow).

The action mode is disabled and the contextual action bar disappears when the user deselects all items, presses the BACK button, or selects the *Done* action on the left side of the bar.

For views that provide contextual actions, you should usually invoke the contextual action mode upon one of two events (or both):

- The user performs a long-click on the view.
- The user selects a checkbox or similar UI component within the view.

How your application invokes the contextual action mode and defines the behavior for each action depends on your design. There are basically two designs:

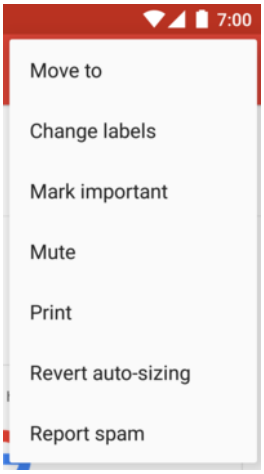
For contextual actions on individual, arbitrary views.

For batch contextual actions on groups of items in a ListView or GridView(allowing the user to select multiple items and perform an action on them all).

4.12 Popup Menu

A Popup Menu is a modal menu anchored to a View. It appears below the anchor view if there is room, or above the view otherwise. It's useful for:

Providing an overflow-style menu for actions that *relate to* specific content (such as Gmail's email headers, below shown in figure).



Providing a second part of a command sentence (such as a button marked "Add" that produces a popup menu with different "Add" options).

Providing a drop-down similar to Spinnerthat does not retain a persistent selection.

PopupMenu is available with API level 11 and higher.

If you define your menu in XML, here's how you can show the popup menu:

Instantiate a PopupMenu with its constructor, which takes the current application Context and the View to which the menu should be anchored.

Use MenuInflater to inflate your menu resource into the Menu object returned by PopupMenu.getMenu().

Call PopupMenu.show().

For example, here's a button with the android:onClick attribute that shows a popup menu:

```
<ImageButton
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:src="@drawable/ic_overflow_holo_dark"
    android:contentDescription="@string/descr_overflow_button"
    android:onClick="showPopup"/>
```

The activity can then show the popup menu like this:

JAVA

```
funshowPopup(v:View){
    val popup =PopupMenu(this, v)
    valinflater:MenuInflater=popup.menuInflater
```

```
    inflater.inflate(R.menu.actions,popup.menu)
    popup.show()
}
```

In API level 14 and higher, you can combine the two lines that inflate the menu with PopupMenu.inflate().

The menu is dismissed when the user selects an item or touches outside the menu area. You can listen for the dismiss event using PopupMenu.OnDismissListener.

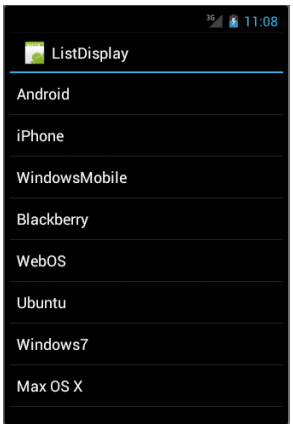
Android ListView is a view which groups several items and display them in vertical scrollable list. The list items are automatically inserted to the list using an Adapter that pulls content from a source such as an array or database.

4.13 List View

An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter holds the data and send the data to adapter view, the view can takes the data from adapter view and shows the data on different views like as spinner, list view, grid view etc.

The ListView and GridView are subclasses of AdapterView and they can be populated by binding them to an Adapter, which retrieves data from an external source and creates a View that represents each data entry.

Android provides several subclasses of Adapter that are useful for retrieving different kinds of data and building views for an AdapterView(i.e.ListView or GridView). The common adapters are ArrayAdapter,Base
Adapter, CursorAdapter,SimpleCursorAdapter,SpinnerAdapter and WrapperListAdapter. We will see separate examples for both the adapters.



4.13.1 List View Attributes

Following are the important attributes specific to GridView −

Sr.No	Attribute & Description
1	android:id This is the ID which uniquely identifies the layout.
2	android:divider This is drawable or color to draw between list items.
3	android:dividerHeight This specifies height of the divider. This could be in px, dp, sp, in, or mm.
4	android:entries Specifies the reference to an array resource that will populate the ListView.
5	android:footerDividersEnabled When set to false, the ListView will not draw the divider before each footer view. The default value is true.
6	android:headerDividersEnabled When set to false, the ListView will not draw the divider after each header view. The default value is true.

4.14 Notification

A **notification** is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the notification area. To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

Android **Toast** class provides a handy way to show users alerts but problem is that these alerts are not persistent which means alert flashes on the screen for a few seconds and then disappears.

4.15 Create and Display Notifications

You have simple way to create a notification. Follow the following steps in your application to create a notification −

Step 1 - Create Notification Builder

As a first step is to create a notification builder using *NotificationCompat.Builder.build()*. You will use Notification Builder to set various Notification properties like its small and large icons, title, priority etc.

NotificationCompat.BuildermBuilder=newNotificationCompat.Builder(this)

Step 2 - Setting Notification Properties

Once you have Builder object, you can set its Notification properties using Builder object as per your requirement. But this is mandatory to set at least following −

A small icon, set by setSmallIcon()
A title, set by setContentTitle()
Detail text, set by setContentText()
mBuilder.setSmallIcon(R.drawable.notification_icon);
mBuilder.setContentTitle("Notification Alert, Click Me!");
mBuilder.setContentText("Hi, This is Android Notification Detail!");
You have plenty of optional properties which you can set for your notification. To learn more about them, see the reference documentation for NotificationCompat.Builder.

Step 3 - Attach Actions

This is an optional part and required if you want to attach an action with the notification. An action allows users to go directly from the notification to an Activity in your application, where they can look at one or more events or do further work.

The action is defined by a Pending Intent containing an Intent that starts an Activity in your application. To associate the Pending Intent with a gesture, call the appropriate method of *NotificationCompat.Builder*. For example, if you want to start Activity when the user clicks the notification text in the notification drawer, you add the Pending Intent by calling setContentIntent().

A Pending Intent object helps you to perform an action on your applications behalf, often at a later time, without caring of whether or not your application is running.
Intent resultIntent = new Intent(this, ResultActivity.class);
TaskStackBuilderstackBuilder = TaskStackBuilder.create(this);
stackBuilder.addParentStack(ResultActivity.class);
// Adds the Intent that starts the Activity to the top of the stack
stackBuilder.addNextIntent(resultIntent);
PendingIntentresultPendingIntent =
stackBuilder.getPendingIntent(0,PendingIntent.FLAG_UPDATE_CURRENT);
mBuilder.setContentIntent(resultPendingIntent);

Step 4 - Issue the notification

Finally, you pass the Notification object to the system by calling NotificationManager.notify() to send your notification. Make sure you call NotificationCompat.Builder.build() method on builder object before notifying it. This method combines all of the options that have been set and return a new Notificationobject.

```
NotificationManagermNotificationManager=(NotificationManager)getSystemService(Context.NOTIFICATION_SERVICE);
// notificationID allows you to update the notification later on.
mNotificationManager.notify(notificationID,mBuilder.build());
```

The NotificationCompat.Builder Class

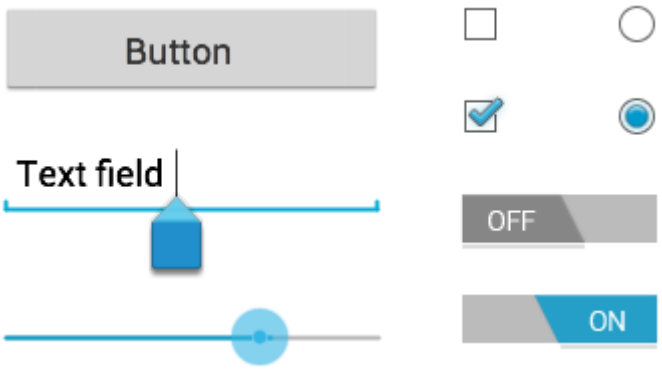
The NotificationCompat.Builder class allows easier control over all the flags, as well as help constructing the typical notification layouts. Following are few important and most frequently used methods available as a part of NotificationCompat.Builder class.

Sr.No.	Constants & Description
1	Notification build() Combine all of the options that have been set and return a new Notification object.
2	NotificationCompat.BuildersetAutoCancel (booleanautoCancel) Setting this flag will make it so the notification is automatically canceled when the user clicks it in the panel.
3	NotificationCompat.BuildersetContent (RemoteViews views) Supply a custom RemoteViews to use instead of the standard one.
4	NotificationCompat.BuildersetContentInfo (CharSequence info) Set the large text at the right-hand side of the notification.
5	NotificationCompat.BuildersetContentIntent (PendingIntent intent) Supply a PendingIntent to send when the notification is clicked.
6	NotificationCompat.BuildersetContentText (CharSequence text) Set the text (second row) of the notification, in a standard notification.
7	NotificationCompat.BuildersetContentTitle (CharSequence title) Set the text (first row) of the notification, in a standard notification.

8	NotificationCompat.BuildersetDefaults (int defaults) Set the default notification options that will be used.
9	NotificationCompat.BuildersetLargeIcon (Bitmap icon) Set the large icon that is shown in the ticker and notification.
10	NotificationCompat.BuildersetNumber (int number) Set the large number at the right-hand side of the notification.
11	NotificationCompat.BuildersetOngoing (boolean ongoing) Set whether this is an ongoing notification.
12	NotificationCompat.BuildersetSmallIcon (int icon) Set the small icon to use in the notification layouts.
13	NotificationCompat.BuildersetStyle (NotificationCompat.Style style) Add a rich notification style to be applied at build time.
14	NotificationCompat.BuildersetTicker (CharSequencetickerText) Set the text that is displayed in the status bar when the notification first arrives.
15	NotificationCompat.BuildersetVibrate (long[] pattern) Set the vibration pattern to use.
16	NotificationCompat.BuildersetWhen (long when) Set the time that the event occurred. Notifications in the panel are sorted by this time.

4.16 Input Controls

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



4.16.1 Common Controls

Android provides several more controls than are listed here. Browse the **android.widget** package to discover more. If your app requires a specific kind of input control, you can build your own custom components.

Control Type	Description	Related Classes
Button	A push-button that can be pressed, or clicked, by the user to perform an action.	**Button**
Text field	An editable text field. You can use the **AutoCompleteTextView** widget to create a text entry widget that provides auto-complete suggestions	**EditText**, **AutoCompleteTextView**
Checkbox	An on/off switch that can be toggled by the user. You should use checkboxes when presenting users with a group of selectable options that are not mutually exclusive.	**CheckBox**
Radio button	Similar to checkboxes, except that only one option can be selected in the group.	**RadioGroup** **RadioButton**
Toggle button	An on/off button with a light indicator.	**ToggleButton**
Spinner	A drop-down list that allows users to select one value from a set.	**Spinner**

Pickers	A dialog for users to select a single value for a set by using up/down buttons or via a swipe gesture. Use a **DatePicker**code> widget to enter the values for the date (month, day, year) or a **TimePicker** widget to enter the values for a time (hour, minute, AM/PM), which will be formatted automatically for the user's locale.	**DatePicker**, **TimePicker**

4.17 Buttons

A button consists of text or an icon (or both text and an icon) that communicates what action occurs when the user touches it.

Depending on whether you want a button with text, an icon, or both, you can create the button in your layout in three ways:

With text, using the **Button** class:

```
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/button_text"
    ... />
```

With an icon, using the **ImageButton** class:

```
<ImageButton
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:src="@drawable/button_icon"
    ... />
```

With text and an icon, using the **Button** class with the **android:drawableLeft** attribute:

```
<Button
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/button_text"
android:drawableLeft="@drawable/button_icon"
```

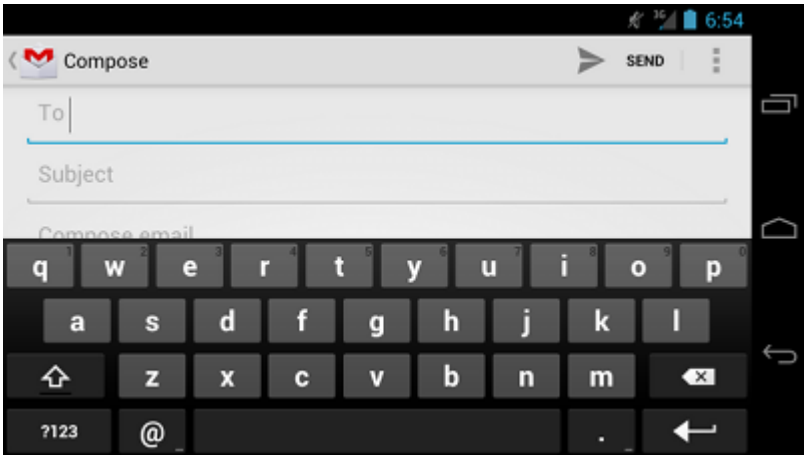
... />

4.18 Text Fields

A text field allows the user to type text into your app. It can be either single line or multi-line. Touching a text field places the cursor and automatically displays the keyboard.

In addition to typing, text fields allow for a variety of other activities, such as text selection (cut, copy, paste) and data look-up via auto-completion.

add a text field to you layout with the EditText object. You should usually do so in your XML layout with a <EditText> element.



Specifying the Keyboard Type

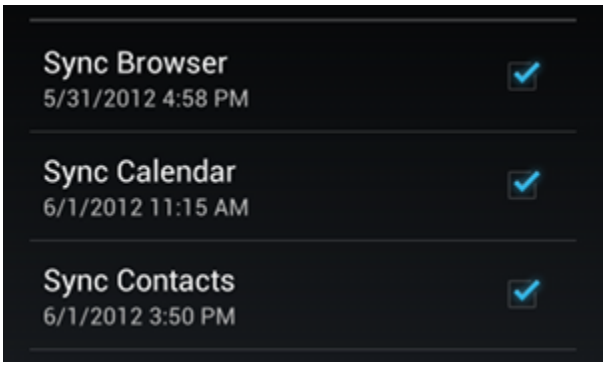
Text fields can have different input types, such as number, date, password, or email address. The type determines what kind of characters are allowed inside the field, and may prompt the virtual keyboard to optimize its layout for frequently used characters.

You can specify the type of keyboard you want for your EditText object with the android:inputType attribute. For example, if you want the user to input an email address, you should use the textEmailAddress input type:

```
<EditText

android:id="@+id/email_address"

android:layout_width="fill_parent"

android:layout_height="wrap_content"

android:hint="@string/email_hint"

android:inputType="textEmailAddress"/>
```

4.19 Checkboxes

Checkboxes allow the user to select one or more options from a set. Typically, you should present each checkbox option in a vertical list.



To create each checkbox option, create a CheckBox in your layout. Because a set of checkbox options allows the user to select multiple items, each checkbox is managed separately and you must register a click listener for each one.

Responding to Click Events

When the user selects a checkbox, the CheckBox object receives an on-click event.

To define the click event handler for a checkbox, add the android:onClick attribute to the <CheckBox>element in your XML layout. The value for this attribute must be the name of the method you want to call in response to a click event. The Activity hosting the layout must then implement the corresponding method.

For example, here are a couple CheckBox objects in a list:

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayoutxmlns:android="http://schemas.android.com/apk/res/android"
android:orientation="vertical"
android:layout_width="fill_parent"
android:layout_height="fill_parent">
<CheckBoxandroid:id="@+id/checkbox_meat"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="@string/meat"
android:onClick="onCheckboxClicked"/>
<CheckBoxandroid:id="@+id/checkbox_cheese"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
```

```
android:text="@string/cheese"
android:onClick="onCheckboxClicked"/>
</LinearLayout>
```

Within the Activity that hosts this layout, the following method handles the click event for both checkboxes:

```
publicvoidonCheckboxClicked(View view){
// Is the view now checked?
booleanchecked=(CheckBox) view).isChecked();

// Check which checkbox was clicked
switch(view.getId()){
caseR.id.checkbox_meat:
if(checked)
// Put some meat on the sandwich
else
// Remove the meat
break;
caseR.id.checkbox_cheese:
if(checked)
// Cheese me
else
// I'm lactose intolerant
break;
// TODO: Veggie sandwich
}
}
```

The method you declare in the android:onClick attribute must have a signature exactly as shown above. Specifically, the method must: Be public Return void

Define a **View** as its only parameter (this will be the **View** that was clicked)

If you need to change the radio button state yourself (such as when loading a saved **CheckBoxPreference**), use the **setChecked(boolean)** or **toggle()** method.

4.20 Alert Dialog

In android, AlertDialog is used to prompt a dialog to the user with message and buttonsto perform an action to proceed further.

The AlertDialog in android application will contain a three regions like as shown below.

In android **Alert Dialogs**, we can show a **title**, up to three buttons, a list of selectable items or a custom layout based on our requirements.

Region	Description
Title	It's an optional and it can be used to show the detailed messages based on our requirements.
Content Area	It is used to display a message, list or other custom layouts based on our requirements.
Action Buttons	It is used to display an action buttons to interact with the users. We can use upto 3 different action buttons in alert dialog, such as positive, negative and neutral.

Generally, in android we can build AlertDialog in our activity file using different dialog methods.

4.20.1 Android Alert Dialog Methods

Following are the some of commonly used methods related to **AlertDialog** control to built alert prompt in android applications.

Method	Description
setTitle()	It is used to set the title of alertdialog and its an optional component.
setIcon()	It is used to set the icon before the title
setMessage()	It is used to set the message required message to display in alertdialog.
setCancelable()	It is used to allow users to cancel alertdialog by clicking on outside of dialog area by setting true / false.

Method	Description
setPositiveButton()	It is used to set the positive button for alertdialog and we can implement click event of positive button.
setNegativeButton()	It is used to set the negative button for alertdialog and we can implement click event of negative button.
setNeutralButton()	It is used to set the neutral button for alertdialog and we can implement click event of neutral button.

4.20.2 Android Alert Dialog Example

Following is the example of defining a one Button control in RelativeLayout to show the AlertDialog and get the action which was performed by user on Button click in android application.

Create a new android application using android studio and give names as **AlertDialogExample**. Now open an **activity_main.xml** file from **\res\layout** path and write the code like as shown below

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">
      <Button
        android:id="@+id/getBtn"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="150dp"
        android:layout_marginTop="200dp"
        android:text="Show Alert" />
</RelativeLayout>
```

If you observe above code we defined a one Button control in RelativeLayout to show the alert dialog on Button click in XML layout file.

Once we are done with creation of layout with required controls, we need to load the XML layout resource from our activity **onCreate()** callback method, for that open main activity file **MainActivity.java** from **\java\com.tutlane.alertdialogexample** path and write the code like as shown below.

MainActivity.java

```
package com.tutlane.alertdialogexample;
import android.content.DialogInterface;
import android.support.v7.app.AlertDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity {
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        Button btn = (Button)findViewById(R.id.getBtn);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
                builder.setTitle("Login Alert")
                        .setMessage("Are you sure, you want to continue ?")
                        .setCancelable(false)
                        .setPositiveButton("Yes", new DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog, int which) {
                                Toast.makeText(MainActivity.this,"Selected Option:
YES",Toast.LENGTH_SHORT).show();
                            }
                        })
                        .setNegativeButton("No", new DialogInterface.OnClickListener() {
                            @Override
                            public void onClick(DialogInterface dialog, int which) {
```

```
                    Toast.makeText(MainActivity.this,"Selected Option:
No",Toast.LENGTH_SHORT).show();
                    }
                });
            //Creating dialog box
            AlertDialog dialog  = builder.create();
            dialog.show();
        }
    });
  }
}
```

If you observe above code we are calling our layout using **setContentView** method in the form of **R.layout.layout_file_name** in our activity file. Here our xml file name is **activity_main.xml** so we used file name **activity_main** and we are trying to show the AlertDialog on Button click.

Generally, during the launch of our activity, **onCreate()** callback method will be called by android framework to get the required layout for an activity.

There are three different kind of lists available with **AlertDialogs** in android, those are

- Single Choice List
- Single Choice List with Radio Buttons
- Single Choice List with Checkboxes

4.21 Spinner

Spinner is a view which allow a user to select one value from the list of values. The spinner in android will behave same like dropdown list in other programming languages.

Generally, the android spinners will provide a quick way to select one item from the list of values and it will show a dropdown menu with a list of all values when we click or tap on it.

By default, the android spinner will show its currently selected value and by using **Adapter** we can bind the items to spinner object.

We can populate our **Spinner** control with list of choices by defining an **ArrayAdapter** in our Activity file.

Generally, the **Adapter** pulls data from a sources such as an array or database and converts each item into a result view and that's placed into the list.

4.21.1 Android Adapter

In android, **Adapter** will act as an intermediate between the data sources and adapter views such as **ListView**, **Gridview** to fill the data into adapter views. The adapter will hold the data and iterates through an items in data set and generate the views for each item in the list.

Generally, in android we have a different types of adapters available to fetch the data from different data sources to fill the data into adapter views, those are

Adapter	Description
ArrayAdapter	It will expects an Array or List as input.
CurosrAdapter	It will accepts an instance of cursor as an input.
SimpleAdapter	It will accepts a static data defined in the resources.
BaseAdapter	It is a generic implementation for all three adapter types and it can be used for List View, Grid view or Spinners based on our requirements

Now we will see how to create spinner or dropdownlist in android applications.

4.21.2 Create Android Spinner in XML Layout File

In android, we can create **Spinner** in XML layout file using **<Spinner>** element with different attributes like as shown below.

```
<Spinner android:id="@+id/spinner1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"/>
```

4.22 Rating Bar

Rating Bar is a UI control which is used to get the rating from the user. The **Rating Bar** is an extension of Seek Bar and Progress Bar that shows a rating in stars and it allow users to set the rating value by touch or click on the stars.

The android **Rating Bar** will always return a rating value as floating point number such as 1.0, 2.0, 2.5, 3.0, 3.5, etc.

In android, by using **android:numStars** attribute we can define the number of stars to display in **Rating Bar**. An example of using Rating Bar is in movie sites or product sites to collect the user rating about the movies or products, etc.

In android, by using **android.widget.RatingBar** component we can display the rating bar with star icons.

4.22.1 Create Android Rating Bar in XML Layout File

In android, we can create Rating Bar in XML layout file using **<RatingBar>** element with different attributes like as shown below.

```
<RatingBar
   android:id="@+id/ratingBar1"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:numStars="5"
   android:rating="3.5"/>
```

If you observe above code snippet, we defined a rating bar (**<RatingBar>**) with different attributes, those are

Attribute	Description
android:id	It is used to uniquely identify the control
android:numStars	It is used to define number of stars to display.
android:rating	It is used to set the default rating value for ratingbar.

Now we will see how to get the rating value from RatingBar control in android applications.

4.22.2 Get Android RatingBar Value

In android, by using **RatingBar** methods (**getNumStars()**, **getRating()**) we can get the number of stars and the rating value which was selected.

Following is the code snippet to get the rating details from RatingBar in android applications.

```
int noofstars = rBar.getNumStars();
float getrating = rBar.getRating();
tView.setText("Rating: "+getrating+"/"+noofstars);
```

This is how we can get the number of stars in RatingBar control and the selected rating value from RatingBar control in android applications.

4.22.3 Android Rating Bar Control Attributes

Following are the some of commonly used attributes related to **RatingBar** control in android applications.

Attribute	Description
android:id	It is used to uniquely identify the control
android:numStars	It is used to define number of stars to display.
android:rating	It is used to set the default rating value for ratingbar.
android:background	It is used to set the background color for progress bar.
android:padding	It is used to set the padding for left, right, top or bottom of progress bar.

4.22.4 Android Rating Bar Control Example

Following is the example of defining a **RatingBar** control, Button control and TextViewcontrol in RelativeLayout to get the selected rating value from RatingBar on Button click.

Create a new android application using android studio and give names as **RatingBarExample**. In case if you are not aware of creating an app in android studio check this article Android Hello World App.

Now open an **activity_main.xml** file from **\res\layout** path and write the code like as shown below

activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent" android:layout_height="match_parent">
    <RatingBar
        android:id="@+id/ratingBar1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="80dp"
        android:layout_marginTop="200dp"
```

```
    android:numStars="5"
    android:rating="3.5"/>
  <Button
    android:id="@+id/btnGet"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/ratingBar1"
    android:layout_below="@+id/ratingBar1"
    android:layout_marginTop="30dp"
    android:layout_marginLeft="60dp"
    android:text="Get Rating"/>
  <TextView
    android:id="@+id/textview1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/btnGet"
    android:layout_below="@+id/btnGet"
    android:layout_marginTop="20dp"
    android:textSize="20dp"
    android:textStyle="bold"/>
</RelativeLayout>
```

If you observe above code we created a one **RatingBar** control, one Button and one TextView control in XML Layout file.

Once we are done with creation of layout with required controls, we need to load the XML layout resource from our activity **onCreate()** callback method, for that open main activity file **MainActivity.java** from **\java\com.tutlane.ratingbarexample** path and write the code like as shown below.

MainActivity.java

```
package com.tutlane.ratingbarexample;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;
import android.view.View;
import android.widget.Button;
```

```
import android.widget.RatingBar;
import android.widget.TextView;
public class MainActivity extends AppCompatActivity {
    private RatingBar rBar;
    private TextView tView;
    private Button btn;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        rBar = (RatingBar) findViewById(R.id.ratingBar1);
        tView = (TextView) findViewById(R.id.textview1);
        btn = (Button)findViewById(R.id.btnGet);
        btn.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                int noofstars = rBar.getNumStars();
                float getrating = rBar.getRating();
                tView.setText("Rating: "+getrating+"/"+noofstars);
            }
        });
    }
}
```

If you observe above code we are calling our layout using **setContentView** method in the form of **R.layout.layout_file_name** in our activity file. Here our xml file name is **activity_main.xml** so we used file name **activity_main** and we are trying to get the number of stars in RatingBar and the selected rating value from RatingBar control.

Generally, during the launch of our activity, **onCreate()** callback method will be called by android framework to get the required layout for an activity.

Output of Android RatingBar Example

When we run above example using android virtual device (AVD) we will get a result.

If you observe above result, we are able to get the rating value from the **RatingBar**control when we click on Button in android application.

This is how we can use **RatingBar** control in android applications to show the ratings based on our requirements.

4.23 Progress Bar

Progress Bar is a user interface control which is used to indicate the progress of an operation. For example, downloading a file, uploading a file.

Following is the pictorial representation of using a different type of progress bars in android applications.

By default the Progress Bar will be displayed as a spinning wheel, in case if we want to show it like horizontal bar then we need to change the style property to horizontal like style="?android:attr/progressBarStyleHorizontal".

4.23.1 Create Android Progress Bar in XML Layout File

In android, we can create Progress Bar in XML layout file using <**ProgressBar**> element with different attributes like as shown below

```
<ProgressBar
    android:id="@+id/pBar3"
    style="?android:attr/progressBarStyleHorizontal"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:minHeight="50dp"
    android:minWidth="250dp"
    android:max="100"
    android:indeterminate="true"
    android:progress="1" />
```

If you observe above code snippet, we defined a progress bar (<ProgressBar>) with different attributes, those are

4.23.2 Android Progress Bar Control Attributes

Following are the some of commonly used attributes related to **Progress Bar** control in android applications.

Attribute	Description

Attribute	Description
android:id	It is used to uniquely identify the control
android:max	It is used to specify the maximum value of the progress can take
android:progress	It is used to specify default progress value.
android:background	It is used to set the background color for progress bar.
android:indeterminate	It is used to enable indeterminate progress mode.
android:padding	It is used to set the padding for left, right, top or bottom of progress bar.

In android, the Progress Bar supports two types of modes to show the progress, those are **Determinate** and **Indeterminate**.

4.23.3 Android Progress Bar with Determinate Mode

Generally, we use the **Determinate** progress mode in progress bar when we want to show the quantity of progress has occurred. For example, the percentage of file downloaded, number of records inserted into database, etc.

To use Determinate progress, we need to set the style of progress bar to **Widget_ProgressBar_Horizontal** or **progressBarStyleHorizontal** and set the amount of progress using **android:progress** attribute.

Following is the example which shows a **Determinate** progress bar that is **50%** completes.

```
<ProgressBar
   android:id="@+id/pBar"
   style="?android:attr/progressBarStyleHorizontal"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:max="100"
   android:progress="50" />
```

By using **setProgress(int)** method, we can update the percentage of progress displayed in app or by calling **incrementProgressBy(int)** method, we can increase the value of current progress completed based on our requirements.

Generally, when the progress value reaches **100** then the progress bar is full. By using **android:max** attribute we can adjust this default value.

4.23.4 Android Progress Bar with Indeterminate Mode

Generally, we use the **Indeterminate** progress mode in progress bar when we don't know how long an operation will take or how much work has done. In indeterminate mode the actual progress will not be shown, only the cyclic animation will be shown to indicate that some progress is happing like as shown in above progress bar loading images.

By using progressBar.**setIndeterminate(true)** in activity file programmatically or using **android:indeterminate = "true"** attribute in XML layout file, we can enable **Indeterminate** progress mode.

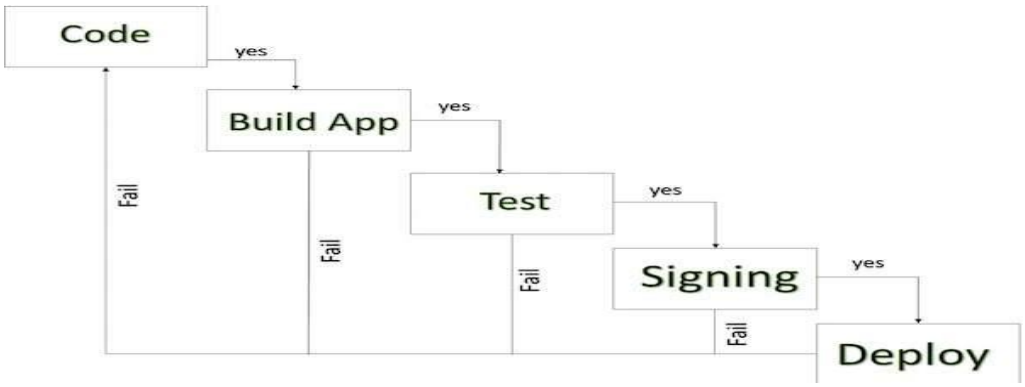
Following is the example to set **Indeterminate** progress mode in XML layout file.

```
<ProgressBar
   android:id="@+id/progressBar1"
   style="?android:attr/progressBarStyleHorizontal"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:indeterminate="true"/>
```

This is how we can define the Progress modes in **ProgressBar** based on our requirements in android applications.

5.0 Live Mobile Application Development

- Android application publishing is a process that makes your Android applications available to users. Infect, publishing is the last phase of the Android application development process.



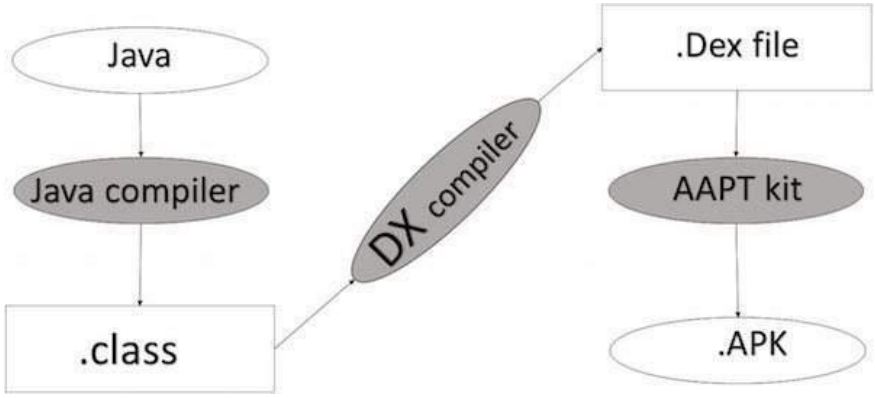
ANDROID DEVELOPMENT LIFE CYCLE

- Once you developed and fully tested your Android Application, you can start selling or distributing free using Google Play (A famous Android marketplace). You can also release your applications by sending them directly to users or by letting users download them from your own website.

Step	Activity
1	**Regression Testing** Before you publish your application, you need to make sure that its meeting the basic quality expectations for all Android apps, on all of the devices that you are targeting. So perform all the required testing on different devices including phone and tablets.
2	**Application Rating** When you will publish your application at Google Play, you will have to specify a content rating for your app, which informs Google Play users of its maturity level. Currently available ratings are (a) Everyone (b) Low maturity (c) Medium maturity (d) High maturity.
3	**Targeted Regions** Google Play lets you control what countries and territories where your application will be sold. Accordingly you must take care of setting up time zone, localization or any other specific requirement as per the targeted region.
4	**Application Size** Currently, the maximum size for an APK published on Google Play is 50 MB. If your app exceeds that size, or if you want to offer a secondary download, you can use APK Expansion Files, which Google Play will host for free on its server infrastructure and automatically handle the download to devices.

5	**SDK and Screen Compatibility** It is important to make sure that your app is designed to run properly on the Android platform versions and device screen sizes that you want to target.
6	**Application Pricing** Deciding whether you app will be free or paid is important because, on Google Play, free app's must remain free. If you want to sell your application then you will have to specify its price in different currencies.
7	**Promotional Content** It is a good marketing practice to supply a variety of high-quality graphic assets to showcase your app or brand. After you publish, these appear on your product details page, in store listings and search results, and elsewhere.
8	**Build and Upload release-ready APK** The release-ready APK is what you you will upload to the Developer Console and distribute to users.
9	**Finalize Application Detail** Google Play gives you a variety of ways to promote your app and engage with users on your product details page, from colourful graphics, screen shots, and videos to localized descriptions, release details, and links to your other apps. So you can decorate your application page and provide as much as clear crisp detail you can provide.

Export Android Application Process



APK DEVELOPMENT PROCESS

Before exporting the apps, you must some of tools

- **Dx tools**(Dalvik executable tools): It going to convert **.class file** to **.dex file**. it has useful for memory optimization and reduce the boot-up speed time
- **AAPT**(Android assistance packaging tool):it has useful to convert **.Dex file** to**.Apk**
- **APK**(Android packaging kit): The final stage of deployment process is called as .apk.

You will need to export your application as an APK (Android Package) file before you upload it Google Play marketplace.

STEPS TO CREATE A SUCCESSFUL MOBILE APP:

Step 1: A great imagination leads to a great app

To create a successful mobile application, the first thing you need to keep in mind is:

- Identify a problem which can be resolved by your app
- Decide the features of your app

Step 2: Identify

To create a successful mobile app, you need to identify or be clear about:

- **Application target users** - An app should always be developed keeping in mind the target users of an application. Having a clear vision regarding the target group, enhance the success ratio of an app.
- **Mobile platforms and devices to be supported** - Mobile platforms and devices should be selected keeping in mind hardware performance, battery life, ruggedness and required peripherals. Certain factors that needs to be considered while selecting mobile platforms and devices includes coverage, device support, performance and other features.
- **Revenue model** - The app market is booming like never before. To ensure this resource and generate revenue, app developer need to select appropriate approach in accordance with the app.

Step 3: Design your app

- Designing your app is yet another significant factor responsible for success of an app in the market.
- An app developer should concentrate on the UI design, multi-touch gestures for touch-enabled devices and consider platform design standards as well.

Step 4: Identify approach to develop the app - native, web or hybrid

Selecting the right approach for developing an app is highly important. Ideally, app development approach must be in accordance with the time and budget constraints of a client.

- **Native:** Native apps enable in delivering the best user experience but require significant time and skill to be developed. These apps are basically platform specific and require expertise along with knowledge.

- **Web:** Web apps are quick and cheap ones to develop and can run on multiple platforms. These are developed using HTML5, CSS and JavaScript code. These web apps are less powerful than native apps.
- **Hybrid:** Hybrid approach is the latest approach to develop any app. This approach combines prebuilt native containers with on-the-fly web coding in order to achieve the best of both worlds.

Step 5: Develop a prototype

- Next stage, after identifying the approach is developing a prototype. It is actually the process of taking your idea and turning it into an application with some basic functionality.
- A prototype makes it quite easier to sell your idea to potential buyers who can now actually view the tangible benefits instead of just visualizing or reading product description.

Step 6: Integrate an appropriate analytics tool

There is also a need to incorporate appropriate analytics which gives you a detailed picture of how many visitors uses your webs, how they arrived on your site and how can they keep coming back.

Some of the mobile analytics tool which helps in this process:

- **Google Analytics**
- **Flurry**
- **Localytics**
- **Mixpanel**
- **Preemptive**

Step 7: Identify beta-testers. Listen to their feedback and integrate relevant ones

- Beta testing is the first opportunity to get feedback from your target customers. It is especially important as it enhances your visibility in the app store. It not only reduces product risk but get you that initial push in the app store.

Step 8: Release / deploy the app

- Deploying an app requires plan, schedule and control of the movement of releases to test and live environments. The major objective of Deployment Management is to ensure the integrity of the live environment is protected and that the correct components are released.

Step 9: Capture the metrics

- There has been significant rise in the mobile app users in the present decade. As a result, the need to collect accurate metrics is highly important. As the number of consumers using mobile applications steadily rises, the need to collect accurate metrics from them is increasingly important.

Step 10: Upgrade your app with improvements and new features

- After capturing the metrics it becomes important to upgrade your app with improvements and innovative features. A mobile app without innovative features loses its usability in long run. Upgrading your app with innovative features enhances its visibility along with downloads of an app.

Game Development Process:

The quality of every Android game depends on the tools and techniques used to create it and, most importantly, the development process used to develop it.

Tools of the trade

Before you start developing an app you need to pick some tools. The "standard" tool for Android app development is Android Studio. The programming language at the heart of Android Studio is Java. We have also many game tools software's

- Corona SDK,
- GameMaker: Studio,
- Unity for 2D
- Stencyl
- Construct 3
- Godot Engine
- Unreal Engine 4

Our typical game development process includes the following phases:

Project Estimation

Our game development process starts with the project's estimation. The estimate is prepared based on the game specification received by the client.

Requirement Analysis

The project leader conducts a detailed analysis of the features to be incorporated into the game as per client's requirement and allocates resources to accomplish specific tasks.

Game Design

The game design phase starts in parallel with the project's concept art. The game's UI flow development and game play mechanic development are done in this phase as well. Architectural design documents, database design documents, and class design documents are prepared by their respective team members.

Concept Art

Our internal team of artists creates concept art by incorporating rough sketches of game characters and other key elements of the project. The art designers create 2D images of characters based on the client's requirements.

Art Development

This phase involves the development of different game assets like a 3D character or 3D models based on the concept art. We use different 3D software such as Blender, Maya, ZBrush, Realflow, Adobe Photoshop, and Adobe After Effects to accomplish art development for our games. The art development process involves:

- Modeling
- Texturing
- UV
- Unwrapping
- Rigging

Development

Here the developers implement the game using game engines. Throughout active development, builds of the game are shared with the customers to solicit feedback and fine tune the look and feel of the game. Code reviews, developer testing, and bug fixes are carrying out simultaneously with coding.

Quality Assurance Testing

Since Android is one of the most diversified platforms, games are required to work well across different screen sizes and other variable device constraints. Our internal team carefully tests our games to

optimize all our games to ensure their performance across multiple devices and to check all functionalities are working properly in the software.

Launch

We conduct user acceptance testing before releasing the games to their respective app stores. Changes in the app's features and functionalities are incorporated into the game according to client feedback.

Game Maintenance and support

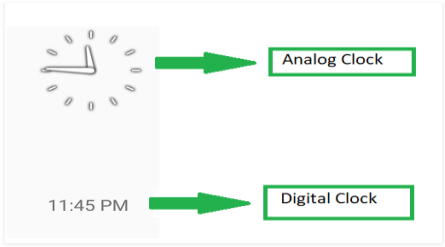
For many projects, our game development process does not end with the product's launch. Similar to other apps, games must also sometimes be updated, and sometimes this regular support may last for years after a game's launch.

Clock

1. **Analog clock**: Analog clock is a subclass of View class. It represents a circular clock. Around the circle, numbers 1 to 12 appear to represent the hour and two hands are used to show instant of the time- shorter one for the hour and longer is for minutes.

2. **Digital clock:** Digital clock is subclass of TextView Class and uses numbers to display the time in "HH:MM" format.

For Example



Steps to create a clock

Step1: Firstly create a new Android Application. This will create an XML file "activity_main.xml" and a Java File "MainActivity.Java".

Step2: Open "activity_main.xml" file and add following widgets in a Relative Layout:
- An Analog clock
- A Digital clock

Step3: Leave the Java file as it is.
Step4: Now Run the app. Both clocks are displayed on the screen.

Activity_main.xml

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

```
    xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">


   <AnalogClock

      android:layout_marginTop="20dp"
      android:layout_marginLeft="120dp"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content" />
    <DigitalClock

       android:layout_marginLeft="140dp"
       android:textSize="25dp"
       android:layout_marginTop="300dp"
       android:layout_width="wrap_content"
       android:layout_height="wrap_content" />

</RelativeLayout>
```

MainActivity.java

```
package org.geeksforgeeks.navedmalik.analogdigital;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class MainActivity extends AppCompatActivity {

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

Calendar

We are going to display the Calendar using CalendarView. It also provides the selection of the current date and displaying the date. The setOnDateChangeListener Interface is used which provide onSelectedDayChange method.

1. onSelectedDayChange: In this method, we get the values of days, months and years that is selected by the user.

Below are the steps for creating Android Application of the Calendar,

Step 1: Create a new project and you will have a layout XML file and java file. Your screen will look like the image below.

Step 2: Open your xml file and add CalendarView and TextView. And assign id to TextView and CalendarView.

Step 3: Now, open up the activity java file and define the CalendarView and TextView type variable, and also use findViewById() to get the Calendarview and textview.

Step 4: Now, add setOnDateChangeListener interface in object of CalendarView which provides setOnDateChangeListener method. In this method we get the Dates(days, months, years) and set the dates in TextView for Display.

Step 5: Now run the app and set the current date which will be shown on the top of the screen.

Activity_main.xml

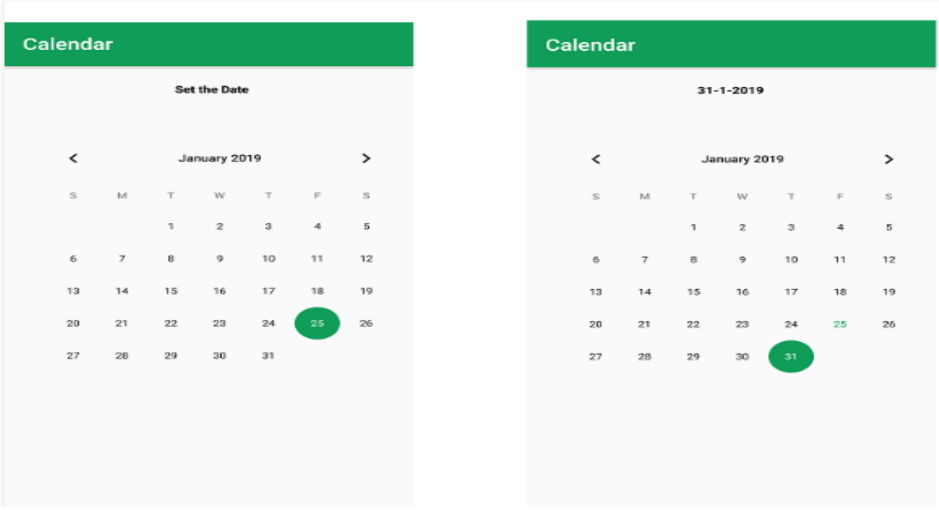
```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
        xmlns:android="http://schemas.android.com/apk/res/android"
        xmlns:app="http://schemas.android.com/apk/res-auto"
        xmlns:tools="http://schemas.android.com/tools"
        android:layout_width="match_parent"
        android:layout_height="match_parent"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

        <!-- Add TextView to display the date -->
        <TextView
                android:id="@+id/date_view"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="150dp"
                android:layout_marginTop="20dp"
                android:text="Set the Date"
                android:textColor="@android:color/background_dark"
                android:textStyle="bold" />

        <!-- Add CalenderView to display the Calender -->
        <CalendarView
                android:id="@+id/calender"
                android:layout_marginTop="80dp"
                android:layout_marginLeft="19dp"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content">
        </CalendarView>

</RelativeLayout>
```

MainActivity.java

```
package org.geeksforgeeks.navedmalik.calendar;
import android.support.annotation.NonNull;
import android.support.v7.app.AppCompatActivity;
```

```
import android.os.Bundle;
import android.widget.Button;
import android.widget.CalendarView;
import android.widget.TextView;

public class MainActivity extends AppCompatActivity {

// Define the variable of CalendarView type and TextView type;
        CalendarView calender;
        TextView date_view;
        @Override
        protected void onCreate(Bundle savedInstanceState)
        {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);

// By ID we can use each component  which id is assign in xml file use findViewById() to get the
CalendarView and TextView
                calender = (CalendarView) findViewById(R.id.calender);
                date_view = (TextView) findViewById(R.id.date_view);

// Add Listener in calendar
                calender.setOnDateChangeListener(  new CalendarView
        .OnDateChangeListener() {
@Override

// In this Listener have one method  and in this method we will  get the value of DAYS, MONTH, YEARS
public void onSelectedDayChange(
        @NonNull CalendarView view,
        int year,
        int month,
        int dayOfMonth)
        {
// Store the value of date with format in String type Variable Add 1 in month because month index is start
with 0
        String Date = dayOfMonth + "-" + (month + 1) + "-" + year;

// set this date in TextView for Display
        date_view.setText(Date);
        }
        });
        }
}
```

Output:



Converter

Converter is used to transform or change the unit of measure to another unit. Some of the conversions provided by Android are Currency, Length, Area, Volume, Temperature, Speed, Time, and Mass.

Simple Currency Converter Android App Example

Open Android Studio from **Start Menu** > **All Programs** or simply tap the icon on the desktop to get started.

Once Android Studio has been fully launched, go to **File** > **New** and **Create a new Project** and name it Currency Converter App or anything you want.

Go to **res** folder > **Layout** and select **activity_main.xml**. Click Text to add the following piece of XML code.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
    <TextView
        android:id="@+id/textview"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Enter Currency in dollars"
        android:textSize="20sp"/>
    <EditText
        android:id="@+id/edtText"
        android:layout_below="@+id/textview"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:inputType="number"
        android:hint="Enter dollars"/>
    <Button
        android:id="@+id/button"
```

```
        android:layout_below="@+id/edtText"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:layout_centerHorizontal="true"
        android:text="CONVERT"
        android:textSize="20sp"
        android:onClick="convertToEuro"/>
    <ImageView
        android:id="@+id/image"
        android:layout_below="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:src="@drawable/dollars"/>
    <TextView
        android:layout_below="@+id/image"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Developed By Martin Tumusiime"
        android:layout_centerHorizontal="true"
        android:textSize="20sp"/>
</RelativeLayout>
```

Head back to **Java** > **com.example.currencyconverterApp** and select **MainActivity**.

```
public class MainActivity extends AppCompatActivity {
    public void convertToEuro(View view){
        EditText editText = (EditText) findViewById(R.id.edtText);
        int dollars = Integer.parseInt(editText.getText().toString());
        int euro = 2000;
        double result = dollars * euro ;
        Toast.makeText(MainActivity.this, Double.toString(result),
Toast.LENGTH_LONG).show();
    }
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

Phone Book

Main_Activity.xml

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/fragmentContainer"
    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" >

</FrameLayout>
```

ADD-EDIT.xml

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

    <GridLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:columnCount="1"
        android:orientation="vertical"
        android:useDefaultMargins="true" >

        <EditText
            android:id="@+id/nameEditText"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="@string/hint_name"
            android:imeOptions="actionNext"
```

```
            android:inputType="textPersonName|textCapWords" >
        </EditText>

        <EditText
            android:id="@+id/phoneEditText"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="@string/hint_phone"
            android:imeOptions="actionNext"
            android:inputType="phone" >
        </EditText>

        <EditText
            android:id="@+id/emailEditText"
            android:layout_width="match_parent"
            android:layout height="wrap content"
            android:hint="@string/hint_email"
            android:imeOptions="actionNext"
            android:inputType="textEmailAddress" >
        </EditText>

        <EditText
            android:id="@+id/streetEditText"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:hint="@string/hint_street"
            android:imeOptions="actionNext"
            android:inputType="textPostalAddress|textCapWords" >
        </EditText>

        <EditText
            android:id="@+id/cityEditText"
            android:layout_width="match_parent"
            android:layout height="wrap content"
            android:hint="@string/hint_city"
            android:imeOptions="actionNext"
            android:inputType="textPostalAddress|textCapWords" >
        </EditText>

        <EditText
            android:id="@+id/stateEditText"
            android:layout width="match parent"
            android:layout_height="wrap_content"
            android:hint="@string/hint_state"
            android:imeOptions="actionNext"
            android:inputType="textPostalAddress|textCapCharacters" >
        </EditText>

        <EditText
            android:id="@+id/zipEditText"
            android:layout width="match parent"
            android:layout_height="wrap_content"
            android:hint="@string/hint zip"
            android:imeOptions="actionDone"
            android:inputType="number" >
        </EditText>

        <Button
            android:id="@+id/saveContactButton"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout gravity="center horizontal"
            android:text="@string/button save contact" >
        </Button>
    </GridLayout>

</ScrollView>
```

MainActivity.java

```
// Hosts Address Book app's fragments
package com.deitel.addressbook;

Import android.app.Activity;
import android.app.FragmentTransaction;
import android.os.Bundle;

public class MainActivity extends Activity
   implements ContactListFragment.ContactListFragmentListener,
      DetailsFragment.DetailsFragmentListener,
      AddEditFragment.AddEditFragmentListener
{
   // keys for storing row ID in Bundle passed to a fragment
   public static final String ROW_ID = "row_id";

   ContactListFragment contactListFragment; // displays contact list

   // display ContactListFragment when MainActivity first loads
   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      // return if Activity is being restored, no need to recreate GUI
      if (savedInstanceState != null)
         return;

      // check whether layout contains fragmentContainer (phone layout);
      // ContactListFragment is always displayed
      if (findViewById(R.id.fragmentContainer) != null)
      {
         // create ContactListFragment
         contactListFragment = new ContactListFragment();

         // add the fragment to the FrameLayout
         FragmentTransaction transaction =
            getFragmentManager().beginTransaction();
         transaction.add(R.id.fragmentContainer, contactListFragment);
         transaction.commit(); // causes ContactListFragment to display
      }
   }

   // called when MainActivity resumes
   @Override
   protected void onResume()
   {
      super.onResume();

      // if contactListFragment is null, activity running on tablet,
      // so get reference from FragmentManager
      if (contactListFragment == null)
      {
         contactListFragment =
            (ContactListFragment) getFragmentManager().findFragmentById(
               R.id.contactListFragment);
      }
   }

   // display DetailsFragment for selected contact
   @Override
   public void onContactSelected(long rowID)
   {
      if (findViewById(R.id.fragmentContainer) != null) // phone
         displayContact(rowID, R.id.fragmentContainer);
      else // tablet
      {
```

```
      getFragmentManager().popBackStack(); // removes top of back stack
      displayContact(rowID, R.id.rightPaneContainer);
   }
}

// display a contact
private void displayContact(long rowID, int viewID)
{
   DetailsFragment detailsFragment = new DetailsFragment();

   // specify rowID as an argument to the DetailsFragment
   Bundle arguments = new Bundle();
   arguments.putLong(ROW_ID, rowID);
   detailsFragment.setArguments(arguments);

   // use a FragmentTransaction to display the DetailsFragment
   FragmentTransaction transaction =
      getFragmentManager().beginTransaction();
   transaction.replace(viewID, detailsFragment);
   transaction.addToBackStack(null);
   transaction.commit(); // causes DetailsFragment to display
}

// display the AddEditFragment to add a new contact
@Override
public void onAddContact()
{
   if (findViewById(R.id.fragmentContainer) != null)
      displayAddEditFragment(R.id.fragmentContainer, null);
   else
      displayAddEditFragment(R.id.rightPaneContainer, null);
}

// display fragment for adding a new or editing an existing contact
private void displayAddEditFragment(int viewID, Bundle arguments)
{
   AddEditFragment addEditFragment = new AddEditFragment();

   if (arguments != null) // editing existing contact
      addEditFragment.setArguments(arguments);

   // use a FragmentTransaction to display the AddEditFragment
   FragmentTransaction transaction =
      getFragmentManager().beginTransaction();
   transaction.replace(viewID, addEditFragment);
   transaction.addToBackStack(null);
   transaction.commit(); // causes AddEditFragment to display
}

// return to contact list when displayed contact deleted
@Override
public void onContactDeleted()
{
   getFragmentManager().popBackStack(); // removes top of back stack

   if (findViewById(R.id.fragmentContainer) == null) // tablet
      contactListFragment.updateContactList();
}

// display the AddEditFragment to edit an existing contact
@Override
public void onEditContact(Bundle arguments)
{
   if (findViewById(R.id.fragmentContainer) != null) // phone
      displayAddEditFragment(R.id.fragmentContainer, arguments);
   else // tablet
      displayAddEditFragment(R.id.rightPaneContainer, arguments);
}
```

```
    // update GUI after new contact or updated contact saved
    @Override
    public void onAddEditCompleted(long rowID)
    {
       getFragmentManager().popBackStack(); // removes top of back stack

       if (findViewById(R.id.fragmentContainer) == null) // tablet
       {
          getFragmentManager().popBackStack(); // removes top of back stack
          contactListFragment.updateContactList(); // refresh contacts

          // on tablet, display contact that was just added or edited
          displayContact(rowID, R.id.rightPaneContainer);
       }
    }
}
```

APP DEPLOYMENT AND TESTING

Mobile Application Testing

- **Mobile application testing** is a process by which application software developed for handheld mobile devices is tested for its functionality, usability and consistency.
- Mobile application testing can be an automated or manual type of testing.
- Mobile applications either come pre-installed or can be installed from mobile software distribution platforms.

Key challenges for mobile application testing

- **Must be downloadable**: The application must be obtainable for the particular platform, generally from an app store.
- **Diversity in mobile platforms/OSes**:There are different mobile operating systems in the market. The major ones are Android, iOS, and Windows Phone. Each operating system has its own limitations.
- **Device availability**: Access to the right set of devices when there is an ever-growing list of devices and operating system versions is a constant mobile application testing challenge. Access to devices can become even more challenging if testers are spread across different locations.
- **Mobile network operators**: There are over 400 mobile network operators in the world; some are CDMA, some GSM, and others use less common network standards like FOMA, and TD-SCDMA.
- **Scripting**: The variety of devices makes executing a test script (scripting) a key challenge. As devices differ in keystrokes, input methods, menu structure and display properties single script does not function on every device.
- **Test method**: There are two main ways of testing mobile applications: testing on real devices or testing on emulators. Emulators often miss issues that can only be caught by testing on real

devices, but because of the multitude of different devices in the market, real devices can be expensive to purchase and time-consuming to use for testing.

- **Compatibility**: It is necessary to test the compatibility; suppose an application can work on the high resolution and it doesn't work on the lower resolution.
- **Should be able to pick up the phone**: While executing the app application should be able to pick up a call.
- **Variety of mobile devices**: Mobile devices differ in screen input methods (QWERTY, touch, normal) with different hardware capabilities.

Types of mobile application testing

- **Functional testing** ensures that the application is working as per the requirements. Most of the tests conducted for this is driven by the user interface and call flow.
- **Laboratory testing**, usually carried out by network carriers, is done by simulating the complete wireless network. This test is performed to find out any glitches when a mobile application uses voice and/or data connection to perform some functions.
- **Performance testing** is undertaken to check the performance and behavior of the application under certain conditions such as low battery, bad network coverage, low available memory, simultaneous access to the application's server by several users and other conditions. Performance of an application can be affected from two sides: the application's server side and client's side
- **Memory leakage testing**: Memory leakage happens when a computer program or application is unable to manage the memory it is allocated resulting in poor performance of the application and the overall slowdown of the system. As mobile devices have significant constraints of available memory, memory leakage testing is crucial for the proper functioning of an application
- **Interrupt testing**: An application while functioning may face several interruptions like incoming calls or network coverage outage and recovery. The different types of interruptions are:
    - Incoming and outgoing SMS and MMS
    - Incoming and outgoing calls
    - Incoming notifications
    - Battery removal
    - Cable insertion and removal for data transfer
    - Network outage and recovery
    - Media player on/off
    - Device power cycle

- **Usability testing** is carried out to verify if the application is achieving its goals and getting a favorable response from users. This is important as the usability of an application is its key to commercial success (it is nothing but user friendliness). Another important part of usability testing is to make sure that the user experience is uniform across all devices.
- **Installation testing**: Certain mobile applications come pre-installed on the device whereas others have to be installed by the store. Installation testing verifies that the installation process goes smoothly without the user having to face any difficulty. This testing process covers installation, updating and uninstalling of an application
- **Certification testing**: To get a certificate of compliance, each mobile device needs to be tested against the guidelines set by different mobile platforms.
- **Security testing** checks for vulnerabilities to hacking, authentication and authorization policies, data security, session management and other security standards.
- **Location testing**: Connectivity changes with network and location, but you can't mimic those fluctuating conditions in a lab. Only in Country[*clarification needed*] non-automated testers can perform comprehensive usability and functionality testing.
- **Outdated software testing**: Not everyone regularly updates their operating system. Some Android users might not even have access to the newest version. Professional testers can test outdated software.
- **Load testing**: When many users all attempt to download, load, and use an app or game simultaneously, slow load times or crashes can occur causing many customers to abandon your app, game, or website. In-country human testing done manually is the most effective way to test load.
- **Black box testing** doesn't include the internally coding logic of the application. The tester tests the application with functionality without peering with internally structure of the application.

DOODLZ APP

Doodlz app is a drawing app enables us to paint by dragging one or more fingers across the screen. The app provides options for setting the drawing color.

Objectives:

- Detect when the user touches the screen, moves a finger across the screen and removes a finger from the screen.
- Process multiple screen touches so the user can draw with multiple fingers at once.

- Use a SensorManager to detect accelerometer motion events to clear the screen when the user shakes the device.
- Use an AtomicBoolean object to allow multiple threads to access a Boolean value in a thread safe manner
- Use a Path objects to store each line's data as the user draws the lines and to draw those lines with a Canvas.
- Use a Toast to briefly display a message on the screen.

Program

DoodlView.xml

```
<?xml version="1.0" encoding="utf-8"?>
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/colorDialogGridLayout"
    android:layout width="match parent"
    android:layout_height="match_parent"
    android:columnCount="2"
    android:orientation="vertical"
    android:useDefaultMargins="true" >

    <TextView
        android:id="@+id/alphaTextView"
        android:layout_column="0"
        android:layout gravity="right|center vertical"
        android:layout_row="0"
        android:text="@string/label alpha" />

    <SeekBar
        android:id="@+id/alphaSeekBar"
        android:layout column="1"
        android:layout_gravity="fill_horizontal"
        android:layout row="0"
        android:max="255" />

    <TextView
        android:id="@+id/redTextView"
        android:layout column="0"
        android:layout_gravity="right|center_vertical"
        android:layout row="1"
        android:text="@string/label_red" />

    <SeekBar
        android:id="@+id/redSeekBar"
        android:layout_column="1"
        android:layout gravity="fill horizontal"
        android:layout_row="1"
        android:max="255" />



    <TextView
        android:id="@+id/greenTextView"
        android:layout_column="0"
        android:layout gravity="right|center vertical"
        android:layout row="2"
        android:text="@string/label_green" />

    <SeekBar
        android:id="@+id/greenSeekBar"
```

```
        android:layout_column="1"
        android:layout_gravity="fill_horizontal"
        android:layout_row="2"
        android:max="255" />

    <TextView
        android:id="@+id/blueTextView"
        android:layout_column="0"
        android:layout_gravity="right|center_vertical"
        android:layout_row="3"
        android:text="@string/label blue" />

    <SeekBar
        android:id="@+id/blueSeekBar"
        android:layout_column="1"
        android:layout gravity="fill horizontal"
        android:layout_row="3"
        android:max="255" />

    <View
        android:id="@+id/colorView"
        android:layout_height="@dimen/color_view_height"
        android:layout column="0"
        android:layout_columnSpan="2"
        android:layout_gravity="fill_horizontal" />

</GridLayout>
```

Activity_Main.xml

```
<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" >

    <fragment
        android:id="@+id/doodleFragment"
        android:name="com.deitel.doodlz.DoodleFragment"
        android:layout width="match parent"
        android:layout_height="match_parent" />

</RelativeLayout>
```

DoodleView.java

```
// Main View for the Doodlz app.
package com.deitel.doodlz;

import java.util.HashMap;
import java.util.Map;

import android.content.Context;
import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Path;
import android.graphics.Point;
import android.os.Build;
import android.provider.MediaStore;
import android.support.v4.print.PrintHelper;
import android.util.AttributeSet;
import android.view.GestureDetector;
import android.view.GestureDetector.SimpleOnGestureListener;
import android.view.Gravity;
```

```
import android.view.MotionEvent;
import android.view.View;
import android.widget.Toast;

// the main screen that is painted
public class DoodleView extends View
{
   // used to determine whether user moved a finger enough to draw again
   private static final float TOUCH_TOLERANCE = 10;

   private Bitmap bitmap; // drawing area for display or saving
   private Canvas bitmapCanvas; // used to draw on bitmap
   private final Paint paintScreen; // used to draw bitmap onto screen
   private final Paint paintLine; // used to draw lines onto bitmap

   // Maps of current Paths being drawn and Points in those Paths
   private final Map<Integer, Path> pathMap = new HashMap<Integer, Path>();
   private final Map<Integer, Point> previousPointMap =
      new HashMap<Integer, Point>();

   // used to hide/show system bars
   private GestureDetector singleTapDetector;

   // DoodleView constructor initializes the DoodleView
   public DoodleView(Context context, AttributeSet attrs)
   {
      super(context, attrs); // pass context to View's constructor
      paintScreen = new Paint(); // used to display bitmap onto screen

      // set the initial display settings for the painted line
      paintLine = new Paint();
      paintLine.setAntiAlias(true); // smooth edges of drawn line
      paintLine.setColor(Color.BLACK); // default color is black
      paintLine.setStyle(Paint.Style.STROKE); // solid line
      paintLine.setStrokeWidth(5); // set the default line width
      paintLine.setStrokeCap(Paint.Cap.ROUND); // rounded line ends

      // GestureDetector for single taps
      singleTapDetector =
         new GestureDetector(getContext(), singleTapListener);
   }

   // Method onSizeChanged creates Bitmap and Canvas after app displays
   @Override
   public void onSizeChanged(int w, int h, int oldW, int oldH)
   {
      bitmap = Bitmap.createBitmap(getWidth(), getHeight(),
         Bitmap.Config.ARGB_8888);
      bitmapCanvas = new Canvas(bitmap);
      bitmap.eraseColor(Color.WHITE); // erase the Bitmap with white
   }

   // clear the painting
   public void clear()
   {
      pathMap.clear(); // remove all paths
      previousPointMap.clear(); // remove all previous points
      bitmap.eraseColor(Color.WHITE); // clear the bitmap
      invalidate(); // refresh the screen
   }

   // set the painted line's color
   public void setDrawingColor(int color)
   {
      paintLine.setColor(color);
   }

   // return the painted line's color
```

```
public int getDrawingColor()
{
   return paintLine.getColor();
}

// set the painted line's width
public void setLineWidth(int width)
{
   paintLine.setStrokeWidth(width);
}

// return the painted line's width
public int getLineWidth()
{
   return (int) paintLine.getStrokeWidth();
}

// called each time this View is drawn
@Override
protected void onDraw(Canvas canvas)
{
   // draw the background screen
   canvas.drawBitmap(bitmap, 0, 0, paintScreen);

   // for each path currently being drawn
   for (Integer key : pathMap.keySet())
      canvas.drawPath(pathMap.get(key), paintLine); // draw line
}

// hide system bars and action bar
public void hideSystemBars()
{
   if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT)
      setSystemUiVisibility(
         View.SYSTEM_UI_FLAG_LAYOUT_STABLE |
         View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION |
         View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN |
         View.SYSTEM_UI_FLAG_HIDE_NAVIGATION |
         View.SYSTEM_UI_FLAG_FULLSCREEN |
         View.SYSTEM_UI_FLAG_IMMERSIVE);
}

// show system bars and action bar
public void showSystemBars()
{
   if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.KITKAT)
      setSystemUiVisibility(
         View.SYSTEM_UI_FLAG_LAYOUT_STABLE |
         View.SYSTEM_UI_FLAG_LAYOUT_HIDE_NAVIGATION |
         View.SYSTEM_UI_FLAG_LAYOUT_FULLSCREEN);
}

// create SimpleOnGestureListener for single tap events
private SimpleOnGestureListener singleTapListener =
   new SimpleOnGestureListener()
   {
      @Override
      public boolean onSingleTapUp(MotionEvent e)
      {
         if ((getSystemUiVisibility() &
            View.SYSTEM_UI_FLAG_HIDE_NAVIGATION) == 0)
            hideSystemBars();
         else
            showSystemBars();
         return true;
      }
   };
```

```
// handle touch event
@Override
public boolean onTouchEvent(MotionEvent event)
{
   // if a single tap event occurred on KitKat or higher device
   if (singleTapDetector.onTouchEvent(event))
      return true;

   // get the event type and the ID of the pointer that caused the event
   int action = event.getActionMasked(); // event type
   int actionIndex = event.getActionIndex(); // pointer (i.e., finger)

   // determine whether touch started, ended or is moving
   if (action == MotionEvent.ACTION_DOWN ||
      action == MotionEvent.ACTION_POINTER_DOWN)
   {
      touchStarted(event.getX(actionIndex), event.getY(actionIndex),
         event.getPointerId(actionIndex));
   }
   else if (action == MotionEvent.ACTION_UP ||
      action == MotionEvent.ACTION POINTER UP)
   {
      touchEnded(event.getPointerId(actionIndex));
   }
   else
   {
      touchMoved(event);
   }

   invalidate(); // redraw
   return true;
} // end method onTouchEvent

// called when the user touches the screen
private void touchStarted(float x, float y, int lineID)
{
   Path path; // used to store the path for the given touch id
   Point point; // used to store the last point in path

   // if there is already a path for lineID
   if (pathMap.containsKey(lineID))
   {
      path = pathMap.get(lineID); // get the Path
      path.reset(); // reset the Path because a new touch has started
      point = previousPointMap.get(lineID); // get Path's last point
   }
   else
   {
      path = new Path();
      pathMap.put(lineID, path); // add the Path to Map
      point = new Point(); // create a new Point
      previousPointMap.put(lineID, point); // add the Point to the Map
   }

   // move to the coordinates of the touch
   path.moveTo(x, y);
   point.x = (int) x;
   point.y = (int) y;
} // end method touchStarted

// called when the user drags along the screen
private void touchMoved(MotionEvent event)
{
   // for each of the pointers in the given MotionEvent
   for (int i = 0; i < event.getPointerCount(); i++)
   {
      // get the pointer ID and pointer index
      int pointerID = event.getPointerId(i);
```

```
         int pointerIndex = event.findPointerIndex(pointerID);

         // if there is a path associated with the pointer
         if (pathMap.containsKey(pointerID))
         {
            // get the new coordinates for the pointer
            float newX = event.getX(pointerIndex);
            float newY = event.getY(pointerIndex);

            // get the Path and previous Point associated with
            // this pointer
            Path path = pathMap.get(pointerID);
            Point point = previousPointMap.get(pointerID);

            // calculate how far the user moved from the last update
            float deltaX = Math.abs(newX - point.x);
            float deltaY = Math.abs(newY - point.y);

            // if the distance is significant enough to matter
            if (deltaX >= TOUCH_TOLERANCE || deltaY >= TOUCH_TOLERANCE)
            {
               // move the path to the new location
               path.quadTo(point.x, point.y, (newX + point.x) / 2,
                  (newY + point.y) / 2);

               // store the new coordinates
               point.x = (int) newX;
               point.y = (int) newY;
            }
         }
      }
   } // end method touchMoved

   // called when the user finishes a touch
   private void touchEnded(int lineID)
   {
      Path path = pathMap.get(lineID); // get the corresponding Path
      bitmapCanvas.drawPath(path, paintLine); // draw to bitmapCanvas
      path.reset(); // reset the Path
   }

   // save the current image to the Gallery
   public void saveImage()
   {
      // use "Doodlz" followed by current time as the image name
      String name = "Doodlz" + System.currentTimeMillis() + ".jpg";

      // insert the image in the device's gallery
      String location = MediaStore.Images.Media.insertImage(
         getContext().getContentResolver(), bitmap, name,
         "Doodlz Drawing");

      if (location != null) // image was saved
      {
         // display a message indicating that the image was saved
         Toast message = Toast.makeText(getContext(),
            R.string.message_saved, Toast.LENGTH_SHORT);
         message.setGravity(Gravity.CENTER, message.getXOffset() / 2,
            message.getYOffset() / 2);
         message.show();
      }
      else
      {
         // display a message indicating that the image was saved
         Toast message = Toast.makeText(getContext(),
            R.string.message_error_saving, Toast.LENGTH_SHORT);
         message.setGravity(Gravity.CENTER, message.getXOffset() / 2,
            message.getYOffset() / 2);
```

```
            message.show();
        }
    } // end method saveImage

    // print the current image
    public void printImage()
    {
        if (PrintHelper.systemSupportsPrint())
        {
            // use Android Support Library's PrintHelper to print image
            PrintHelper printHelper = new PrintHelper(getContext());

            // fit image in page bounds and print the image
            printHelper.setScaleMode(PrintHelper.SCALE_MODE_FIT);
            printHelper.printBitmap("Doodlz Image", bitmap);
        }
        else
        {
            // display message indicating that system does not allow printing
            Toast message = Toast.makeText(getContext(),
               R.string.message error printing, Toast.LENGTH SHORT);
            message.setGravity(Gravity.CENTER, message.getXOffset() / 2,
               message.getYOffset() / 2);
            message.show();
        }
    }
}
```

TIP CALCULATOR APP

- Tip calculator calculates tip amount for various percentages of the cost of the service, and also provides a total amount that includes the tip.
- A top or gratuity is an extra sum of money paid to certain service workers for a provided service. Tip amounts as well as acceptance; vary in different parts of the world.

Activity_main.xml

```
<GridLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/gridLayout"
    android:layout width="match parent"
    android:layout_height="match_parent"
    android:columnCount="2"
    android:orientation="horizontal"
    android:paddingBottom="@dimen/activity_vertical_margin"
    android:paddingLeft="@dimen/activity horizontal margin"
    android:paddingRight="@dimen/activity_horizontal_margin"
    android:paddingTop="@dimen/activity vertical margin"
    android:useDefaultMargins="true"
    tools:context=".MainActivity" >

    <TextView
        android:id="@+id/amountTextView"
        android:layout width="wrap content"
        android:layout height="wrap content"
        android:layout_gravity="right|center_vertical"
        android:labelFor="@+id/amountEditText"
        android:text="@string/amount"
        android:textAppearance="?android:attr/textAppearanceMedium" />

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

```
        android:layout_column="1"
        android:layout_row="0"
        android:digits="0123456789"
        android:inputType="number"
        android:maxLength="6" >

        <requestFocus />
    </EditText>

    <TextView
        android:id="@+id/amountDisplayTextView"
        android:layout width="wrap content"
        android:layout_height="wrap_content"
        android:layout_column="1"
        android:layout_gravity="fill_horizontal"
        android:layout row="0"
        android:background="@android:color/holo_blue_bright"
        android:padding="@dimen/textview padding"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <TextView
        android:id="@+id/customPercentTextView"
        android:layout width="wrap content"
        android:layout_gravity="right|center_vertical"
        android:text="@string/custom_tip_percentage"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <SeekBar
        android:id="@+id/customTipSeekBar"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_gravity="fill_horizontal"
        android:max="30"
        android:progress="18" />

    <LinearLayout
        android:id="@+id/percentLinearLayout"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout column="1"
        android:layout_gravity="fill_horizontal" >

        <TextView
            android:id="@+id/percent15TextView"
            android:layout width="wrap content"
            android:layout_height="match_parent"
            android:layout weight="1"
            android:gravity="center"
            android:text="@string/fifteen_percent"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <TextView
            android:id="@+id/percentCustomTextView"
            android:layout_width="wrap_content"
            android:layout_height="match_parent"
            android:layout weight="1"
            android:gravity="center"
            android:text="@string/eighteen percent"
            android:textAppearance="?android:attr/textAppearanceMedium" />

    </LinearLayout>

    <TextView
        android:id="@+id/tipTextView"
        android:layout gravity="right|center vertical"
        android:text="@string/tip"
        android:textAppearance="?android:attr/textAppearanceMedium"  />
```

```
    <LinearLayout
        android:id="@+id/tipLinearLayout"
        android:layout_height="wrap_content"
        android:layout_column="1"
        android:layout gravity="fill horizontal" >

        <TextView
            android:id="@+id/tip15TextView"
            android:layout_width="0dp"
            android:layout_height="match_parent"
            android:layout gravity="center"
            android:layout marginRight="@dimen/textview margin"
            android:layout_weight="1"
            android:background="@android:color/holo_orange_light"
            android:gravity="center"
            android:padding="@dimen/textview padding"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <TextView
            android:id="@+id/tipCustomTextView"
            android:layout width="0dp"
            android:layout_height="match_parent"
            android:layout gravity="center"
            android:layout_weight="1"
            android:background="@android:color/holo_orange_light"
            android:gravity="center"
            android:padding="@dimen/textview padding"
            android:textAppearance="?android:attr/textAppearanceMedium" />

    </LinearLayout>

    <TextView
        android:id="@+id/totalTextView"
        android:layout gravity="right|center vertical"
        android:text="@string/total"
        android:textAppearance="?android:attr/textAppearanceMedium" />

    <LinearLayout
        android:id="@+id/totalLinearLayout"
        android:layout height="wrap content"
        android:layout_column="1"
        android:layout gravity="fill horizontal" >

        <TextView
            android:id="@+id/total15TextView"
            android:layout_width="0dp"
            android:layout height="match parent"
            android:layout_gravity="center"
            android:layout_marginRight="@dimen/textview_margin"
            android:layout_weight="1"
            android:background="@android:color/holo orange light"
            android:gravity="center"
            android:padding="@dimen/textview padding"
            android:textAppearance="?android:attr/textAppearanceMedium" />

        <TextView
            android:id="@+id/totalCustomTextView"
            android:layout width="0dp"
            android:layout_height="match_parent"
            android:layout_gravity="center"
            android:layout_weight="1"
            android:background="@android:color/holo orange light"
            android:gravity="center"
            android:padding="@dimen/textview_padding"
            android:textAppearance="?android:attr/textAppearanceMedium" />

    </LinearLayout>
```

```
    <Space />

</GridLayout>
```

Activity_main.java

```
// MainActivity.java
// Calculates bills using 15% and custom percentage tips.
package com.deitel.tipcalculator;

import java.text.NumberFormat; // for currency formatting

import android.app.Activity; // base class for activities
import android.os.Bundle; // for saving state information
import android.text.Editable; // for EditText event handling
import android.text.TextWatcher; // EditText listener
import android.widget.EditText; // for bill amount input
import android.widget.SeekBar; // for changing custom tip percentage
import android.widget.SeekBar.OnSeekBarChangeListener; // SeekBar listener
import android.widget.TextView; // for displaying text

// MainActivity class for the Tip Calculator app
public class MainActivity extends Activity
{
   // currency and percent formatters
   private static final NumberFormat currencyFormat =
      NumberFormat.getCurrencyInstance();
   private static final NumberFormat percentFormat =
      NumberFormat.getPercentInstance();

   private double billAmount = 0.0; // bill amount entered by the user
   private double customPercent = 0.18; // initial custom tip percentage
   private TextView amountDisplayTextView; // shows formatted bill amount
   private TextView percentCustomTextView; // shows custom tip percentage
   private TextView tip15TextView; // shows 15% tip
   private TextView total15TextView; // shows total with 15% tip
   private TextView tipCustomTextView; // shows custom tip amount
   private TextView totalCustomTextView; // shows total with custom tip

   // called when the activity is first created
   @Override
   protected void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState); // call superclass's version
      setContentView(R.layout.activity main); // inflate the GUI

      // get references to the TextViews
      // that MainActivity interacts with programmatically
      amountDisplayTextView =
         (TextView) findViewById(R.id.amountDisplayTextView);
      percentCustomTextView =
         (TextView) findViewById(R.id.percentCustomTextView);
      tip15TextView = (TextView) findViewById(R.id.tip15TextView);
      total15TextView = (TextView) findViewById(R.id.total15TextView);
      tipCustomTextView = (TextView) findViewById(R.id.tipCustomTextView);
      totalCustomTextView =
         (TextView) findViewById(R.id.totalCustomTextView);

      // update GUI based on billAmount and customPercent
      amountDisplayTextView.setText(
         currencyFormat.format(billAmount));
      updateStandard(); // update the 15% tip TextViews
      updateCustom(); // update the custom tip TextViews

      // set amountEditText's TextWatcher
      EditText amountEditText =
```

```
      (EditText) findViewById(R.id.amountEditText);
   amountEditText.addTextChangedListener(amountEditTextWatcher);

   // set customTipSeekBar's OnSeekBarChangeListener
   SeekBar customTipSeekBar =
      (SeekBar) findViewById(R.id.customTipSeekBar);
   customTipSeekBar.setOnSeekBarChangeListener(customSeekBarListener);
} // end method onCreate

// updates 15% tip TextViews
private void updateStandard()
{
   // calculate 15% tip and total
   double fifteenPercentTip = billAmount * 0.15;
   double fifteenPercentTotal = billAmount + fifteenPercentTip;

   // display 15% tip and total formatted as currency
   tip15TextView.setText(currencyFormat.format(fifteenPercentTip));
   total15TextView.setText(currencyFormat.format(fifteenPercentTotal));
} // end method updateStandard

// updates the custom tip and total TextViews
private void updateCustom()
{
   // show customPercent in percentCustomTextView formatted as %
   percentCustomTextView.setText(percentFormat.format(customPercent));

   // calculate the custom tip and total
   double customTip = billAmount * customPercent;
   double customTotal = billAmount + customTip;

   // display custom tip and total formatted as currency
   tipCustomTextView.setText(currencyFormat.format(customTip));
   totalCustomTextView.setText(currencyFormat.format(customTotal));
} // end method updateCustom

// called when the user changes the position of SeekBar
private OnSeekBarChangeListener customSeekBarListener =
   new OnSeekBarChangeListener()
{
   // update customPercent, then call updateCustom
   @Override
   public void onProgressChanged(SeekBar seekBar, int progress,
      boolean fromUser)
   {
      // sets customPercent to position of the SeekBar's thumb
      customPercent = progress / 100.0;
      updateCustom(); // update the custom tip TextViews
   } // end method onProgressChanged

   @Override
   public void onStartTrackingTouch(SeekBar seekBar)
   {
   } // end method onStartTrackingTouch

   @Override
   public void onStopTrackingTouch(SeekBar seekBar)
   {
   } // end method onStopTrackingTouch
}; // end OnSeekBarChangeListener

// event-handling object that responds to amountEditText's events
private TextWatcher amountEditTextWatcher = new TextWatcher()
{
   // called when the user enters a number
   @Override
   public void onTextChanged(CharSequence s, int start,
      int before, int count)
```

```
        {
            // convert amountEditText's text to a double
            try
            {
                billAmount = Double.parseDouble(s.toString()) / 100.0;
            } // end try
            catch (NumberFormatException e)
            {
                billAmount = 0.0; // default if an exception occurs
            } // end catch

            // display currency formatted bill amount
            amountDisplayTextView.setText(currencyFormat.format(billAmount));
            updateStandard(); // update the 15% tip TextViews
            updateCustom(); // update the custom tip TextViews
        } // end method onTextChanged

        @Override
        public void afterTextChanged(Editable s)
        {
        } // end method afterTextChanged

        @Override
        public void beforeTextChanged(CharSequence s, int start, int count,
            int after)
        {
        } // end method beforeTextChanged
    }; // end amountEditTextWatcher
}
```

WEATHER VIEWER APP

- It provides the weather status for the user. So this app needs some permission to access location and internet connection.
- You can request for internet permission in your AndroidManifest.xml using

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

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

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

    <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"
            android:theme="@style/CustomTheme"
            android:screenOrientation="portrait">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

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

```
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Main_Activity.java

```
package com.androdocs.weatherapp;

import android.os.AsyncTask;
import android.os.Bundle;
import android.view.View;
import android.widget.TextView;

import androidx.appcompat.app.AppCompatActivity;

import com.androdocs.httprequest.HttpRequest;

import org.json.JSONException;
import org.json.JSONObject;

import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Locale;

public class MainActivity extends AppCompatActivity {

    String CITY = "dhaka,bd";
    String API = "8118ed6ee68db2debfaaa5a44c832918";

    TextView addressTxt, updated_atTxt, statusTxt, tempTxt, temp_minTxt, temp_maxTxt, sunriseTxt,
            sunsetTxt, windTxt, pressureTxt, humidityTxt;

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

        addressTxt = findViewById(R.id.address);
        updated_atTxt = findViewById(R.id.updated_at);
        statusTxt = findViewById(R.id.status);
        tempTxt = findViewById(R.id.temp);
        temp_minTxt = findViewById(R.id.temp_min);
        temp_maxTxt = findViewById(R.id.temp_max);
        sunriseTxt = findViewById(R.id.sunrise);
        sunsetTxt = findViewById(R.id.sunset);
        windTxt = findViewById(R.id.wind);
        pressureTxt = findViewById(R.id.pressure);
        humidityTxt = findViewById(R.id.humidity);

        new weatherTask().execute();
```

```
    }

    class weatherTask extends AsyncTask<String, Void, String> {
        @Override
        protected void onPreExecute() {
            super.onPreExecute();

            /* Showing the ProgressBar, Making the main design GONE */
            findViewById(R.id.loader).setVisibility(View.VISIBLE);
            findViewById(R.id.mainContainer).setVisibility(View.GONE);
            findViewById(R.id.errorText).setVisibility(View.GONE);
        }

        protected String doInBackground(String... args) {
            String response = HttpRequest.excuteGet("https://api.openweathermap.org/data/2.5/weather?q=" + CITY +
"&units=metric&appid=" + API);
            return response;
        }

        @Override
        protected void onPostExecute(String result) {


            try {
                JSONObject jsonObj = new JSONObject(result);
                JSONObject main = jsonObj.getJSONObject("main");
                JSONObject sys = jsonObj.getJSONObject("sys");
                JSONObject wind = jsonObj.getJSONObject("wind");
                JSONObject weather = jsonObj.getJSONArray("weather").getJSONObject(0);

                Long updatedAt = jsonObj.getLong("dt");
                String updatedAtText = "Updated at: " + new SimpleDateFormat("dd/MM/yyyy hh:mm a",
Locale.ENGLISH).format(new Date(updatedAt * 1000));
                String temp = main.getString("temp") + "°C";
                String tempMin = "Min Temp: " + main.getString("temp_min") + "°C";
                String tempMax = "Max Temp: " + main.getString("temp_max") + "°C";
                String pressure = main.getString("pressure");
                String humidity = main.getString("humidity");

                Long sunrise = sys.getLong("sunrise");
                Long sunset = sys.getLong("sunset");
                String windSpeed = wind.getString("speed");
                String weatherDescription = weather.getString("description");

                String address = jsonObj.getString("name") + ", " + sys.getString("country");


                /* Populating extracted data into our views */
                addressTxt.setText(address);
                updated_atTxt.setText(updatedAtText);
                statusTxt.setText(weatherDescription.toUpperCase());
                tempTxt.setText(temp);
                temp_minTxt.setText(tempMin);
```

```
                temp_maxTxt.setText(tempMax);
                sunriseTxt.setText(new SimpleDateFormat("hh:mm a", Locale.ENGLISH).format(new Date(sunrise *
1000)));
                sunsetTxt.setText(new SimpleDateFormat("hh:mm a", Locale.ENGLISH).format(new Date(sunset * 1000)));
                windTxt.setText(windSpeed);
                pressureTxt.setText(pressure);
                humidityTxt.setText(humidity);

                /* Views populated, Hiding the loader, Showing the main design */
                findViewById(R.id.loader).setVisibility(View.GONE);
                findViewById(R.id.mainContainer).setVisibility(View.VISIBLE);


            } catch (JSONException e) {
                findViewById(R.id.loader).setVisibility(View.GONE);
                findViewById(R.id.errorText).setVisibility(View.VISIBLE);
            }

        }
    }
}
```