

UNIT -III

HISTORY OF ANDROID



In October 2003, Android Inc was founded in Palo Alto, California and its four founders were Rich Miner, Nick Sears, Chris White, and Andy Rubin. In 2005, Android was acquired by Google. Rubin stayed at Google as head of the Android team until 2013.

The logo for the Android OS was created by Irina Blok while she was employed by Google.

Versions of Android along with their names

ANDROID1.5	CUPCAKE
ANDROID1.6	DONUT
ANDROID2.0-2.1	ÉCLAIR
ANDROID2.2	FROYO
ANDROID2.3	GINGERBREAD
ANDROID3.0	HONEYCOMB
ANDROID4.0	ICECREAMSANDWICH
ANDROID4.1-4.3	JELLYBEAN
ANDROID4.4	KITKAT
ANDROID5.0	LOLLIPOP
ANDROID6.0	MARSHMALLOW
ANDROID7.0	NOUGAT
ANDROID8.0	OREO
ANDROID9.0	PIE
ANDROID10 Q	

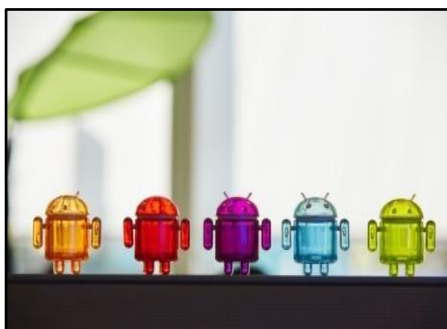
Android Q will allow users to control apps' access to their phone's Photos and Videos or the Audio collections via new runtime permissions.

INTRODUCTION TO ANDROID

In 2007, Apple launched the first iPhone and ushered in a new era in mobile computing. In Sept. 2008, the very [first Android smartphone was announced](#), the T-Mobile G1 went on sale in the U.S. Oct. of that year.

Android 1.0 OS inside integrated a number of the company's other products and services, including Google Maps, YouTube, and an HTML browser (pre-Chrome) that, of course, used Google's search services. It also had the first version of Android Market, the app store with —dozens of unique, first-of-a-kind Android applications.¶

The first version of the OS (1.0) released in Sept. 2008 did not have a code name at all. However, it reportedly used the internal name —Petit four¶ while it was in development at Google. The name refers to a French dessert.



Android has come a long way from its humble beginnings, as the product of a small start up, all the way to becoming the leading mobile operating system worldwide. Google's introduction of [Project Treble](#) in Android Oreo should make it easier for phone makers to update their devices faster.

One challenge for Android device owners that has been an issue for the OS ever since it launched is updating it with the latest security patches, for major feature updates. Google's supported Nexus and Pixel devices consistently receive regular monthly security updates, and the latest version of the OS.

OPERATING SYSTEMS

Different OS run on different types of hardware and are designed for different types of applications. For example, iOS is designed for iPhones and iPad tablets, while Mac desktops and laptops use macOS.

MICROSOFT WINDOWS :

Initial versions of Windows worked with MS-DOS, providing a modern graphical interface on top of DOS's traditional text-based commands. The Windows Start menu helps users find programs and files on their devices.

APPLE IOS

Apple's iOS is one of the most popular smartphone operating systems, second only to Android. It runs on Apple hardware, including iPhones, iPad tablets and iPod Touch media players.

GOOGLE'S ANDROID OS

Android is the most popular operating system in the world judging by the number of devices installed. Users can download custom versions of the operating system.

APPLE MAC OS

Apple's macOS, successor to the popular OS X operating system, runs on Apple laptops and desktops.. MacOS is known for its user-friendly features, which include Siri and FaceTime.

LINUX OPERATING SYSTEM

Linux can be run on a wide variety of hardware and is available free of charge over the internet.

ANDROID DEVELOPMENT TOOLS

- [Editors and IDEs](#)
- [Language Resources](#)
- [Libraries](#)
- [Plug-ins](#)

Android Editors and IDEs

- [Android Studio](#) – The official IDE, based on the community-created IntelliJIDEA (see below).
- [Eclipse](#) – Before Android Studio, this was the official Android development environment. Used to code Java but can be expanded to other languages via plugins, it is still a powerful tool.
- [IntelliJIDEA](#) – Android Studio is based on this, and this IDE is not only extremely useful, but has a massive amount of community-created plugins, making it highly customisable.
- [DroidEdit](#) – An Android text and code editor to use on Android platforms.
- [Android-IDE](#) – A complete web and Android development environment, it also allows you to edit Java and PhoneGap apps.
- [Cordova](#) – Mobile apps with HTML, CSS and JS, its one of the best tools if you want to create hybrid apps. Free and open source.
- [Corona](#) – A 2D-development platform with a specific focus on games but can be used to create other types of mobile apps too. One of the best for cross-platform development and 100% free.
- [Titanium](#) – One of the lesser-known platforms, it allows for the creation of native apps for iOS, Android and Windowsphone and runs off a single JavaScript codebase.
- [Xamarin](#) – Widely featured in the press and a very impressive IDE for native Android, iOS and Windows applications. Open source and free with two further price plans, it uses C# as its language
- [CppDroid](#) – Allows you to code, edit compile and execute C and C++ code. Packed full of features including practice programs and syntax highlighting.

Android Language Resources

- [Java](#) – Straight to the source, if you're developing in Android, Java is probably the language you want to be using. Has its own development kit, but there are plenty of other SDKs out there too.
- [Codecademy](#) – One of the premier code-learning resources online, it has been used by thousands of people to get into Java coding, as well as other languages and frameworks. An interactive, learn-as-you-code format.
- [Team treehouse](#) – Another e-learning website, but well known for the strength of its Java courses.
- [Udemy](#) – Online learning can't go without mentioning Udemy, which features dozens of both highly specific and generic Java learning courses.
- [New Boston](#) – Youtube tutorials to learn how to develop in Android – currently has over 5 million views. Covers everything from setting up the SDK to XML Layouts. 200 videos in total.

- [Ryan Park Apps resource list](#) – Ryan Parks taught himself how to code in Java and published, among others, a personal finance application. This is the list of resources he used.
- [Oracle Java Tutorials](#) – Both general and specialised Java tutorials by IT giants Oracle, starts from the very basic concepts and overview.
- [Cave of Programming](#) – Covers both Java and C++, comes with exercises and tests: also sometimes offers paid-for courses for free, pending approval by the creator of the site, John.

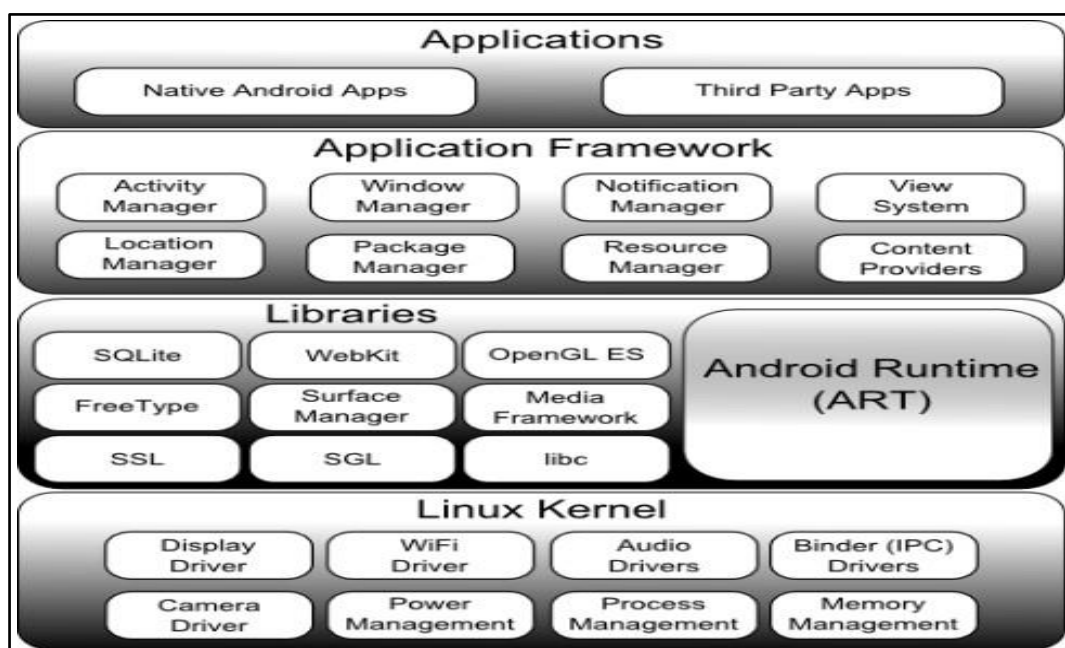
Android Libraries

- [Gson](#) – Serialising and deserialising Java objects in JSON.
- [Retrofit](#) – Described as an —elegant solution for organising API calls.
- [Awesome Java](#) – A list of some of the best Java frameworks and libraries.
- [AndroidView Animations](#) – Library with very simple syntax to get regular View animations working smoothly.
- [EventBus](#) – Aimed at making communication between parts of your application as smooth and easy as possible.
- [ButterKnife](#) – Very lightweight library which streamlines various wordy Android syntax issues by using annotations to create boilerplate template code.
- [Picasso](#) – Specifically useful when download images for apps. Just inputting the image’s URL will download the image, store as bitmap and cache it.

Android Plug-ins

- [Plugin collection for IntelliJ](#) – The main repository for IntelliJ plugins, an absolute treasure-trove of handy tools for the IntelliJ IDE.
- [A curated list of IntelliJ Plugins](#) – The above repository is absolutely huge, so to help you get started and find some gems, here’s a curated list of the best IntelliJ plugins.
- [Import Drawables](#) – For IntelliJ, allows importing of drawables at different resolutions and other image-based functionalities.
- [GenyMotion](#) – One of the biggest and most reliable testing and emulation tools for Android apps – employed by BlaBla Car among other high-profile names.
- [Boilerplate Code Generation](#) – For IntelliJ, generates parcelable boilerplate code.
- [Android Holo Colors](#) – Generates all necessary XML to have edittext and colour spinners in your Android app.

ANDROID ARCHITECTURE



Android is structured in the form of a software stack comprising applications, an operating system, run-time environment, middleware, services and libraries. Each layer of the stack, and the corresponding elements within each layer, are tightly integrated and carefully tuned to provide the optimal application development and execution environment for mobile devices.

THE LINUX KERNEL

Positioned at the bottom of the Android software stack, the Linux Kernel provides a level of abstraction between the device hardware and the upper layers of the Android software stack. Based on Linux version 2.6, the kernel provides pre-emptive multitasking, low-level core system services such as memory, process and power management in addition to providing a network stack and device drivers for hardware such as the device display, Wi-Fi and audio.

ANDROID RUNTIME – ART

When an Android app is built within Android Studio it is compiled into an intermediate byte-code format (DEX format). When the application is subsequently loaded onto the device, the Android Runtime (ART) uses a process referred to as Ahead-of-Time (AOT) compilation to translate the byte-code down to the native instructions required by the device processor. This format is known as Executable and Linkable Format (ELF). Each time the application is subsequently launched, the ELF executable version is run, resulting in faster application performance and improved battery life.

ANDROID LIBRARIES

In addition to a set of standard Java development libraries (providing support for such general purpose tasks as string handling, networking and file manipulation), the Android development environment also includes the Android Libraries. These are a set of Java-based libraries that are specific to Android development.

C/C++ LIBRARIES

The Android runtime core libraries are Java-based and provide the primary APIs for developers writing Android applications. It is important to note, however, that the core libraries do not perform much of the actual work and are, in fact, essentially Java ~~wrappers~~ around a set of C/C++ based libraries.

APPLICATION FRAMEWORK

The Application Framework is a set of services that collectively form the environment in which Android applications run and are managed. This framework implements the concept that Android applications are constructed from reusable, interchangeable and replaceable components. This concept is taken a step further in that an application is also able to publish its capabilities along with any corresponding data so that they can be found and reused by other applications.

APPLICATIONS

Located at the top of the Android software stack are the applications. These comprise both the native applications provided with the particular Android implementation (for example web browser and email applications) and the third party applications installed by the user after purchasing the device.

UNIT – IV

DEVELOPMENT TOOLS

INSTALLING AND USING ECLIPSE WITH ADT PLUG-IN

Installing the Eclipse Plugin

Android offers a custom plugin for the Eclipse IDE, called Android Development Tools (ADT). This plugin provides a powerful, integrated environment in which to develop Android apps. It extends the capabilities of Eclipse to let you quickly set up new Android projects, build an app UI, debug your app, and export signed (or unsigned) app packages (APKs) for distribution.

If you need to install Eclipse, you can download it from eclipse.org/mobile.

Note: If you prefer to work in a different IDE, you do not need to install Eclipse or ADT. Instead, you can directly use the SDK tools to build and debug your application.

Download the ADT Plugin

1. Start Eclipse, then select Help > Install New Software.
2. Click Add, in the top-right corner.
3. In the Add Repository dialog that appears, enter "ADT Plugin" for the Name and the following URL for the Location:
4. <https://dl-ssl.google.com/android/eclipse/>
5. Click OK.
6. If you have trouble acquiring the plugin, try using "http" in the Location URL, instead of "https" (https is preferred for security reasons).
7. In the Available Software dialog, select the checkbox next to Developer Tools and click Next.
8. In the next window, you'll see a list of the tools to be downloaded. Click Next.
9. Read and accept the license agreements, then click Finish.
10. If you get a security warning saying that the authenticity or validity of the software can't be established, click OK.
11. When the installation completes, restart Eclipse.

Configure the ADT Plugin

1. Once Eclipse restarts, you must specify the location of your Android SDK directory:
2. In the "Welcome to Android Development" window that appears, select **Use existing SDKs**.
3. Browse and select the location of the Android SDK directory you recently downloaded and unpacked.
4. Click **Next**.
5. Your Eclipse IDE is now set up to develop Android apps, but you need to add the latest SDK platform tools and an Android platform to your environment. To get these packages for your SDK, continue to [Adding Platforms and Packages](#).

Troubleshooting Installation

1. If you are having trouble downloading the ADT plugin after following the steps above, here are some suggestions:
2. If Eclipse can not find the remote update site containing the ADT plugin, try changing the remote site URL to use http, rather than https. That is, set the Location for the remote site to:
3. <http://dl-ssl.google.com/android/eclipse/>
4. If you are behind a firewall (such as a corporate firewall), make sure that you have properly configured your proxy settings in Eclipse. In Eclipse, you can configure proxy information from the main Eclipse menu in **Window** (on Mac OS X, **Eclipse**) > **Preferences** > **General** > **Network Connections**.

If you are still unable to use Eclipse to download the ADT plugin as a remote update site, you can download the ADT zip file to your local machine and manually install it:

1. Download the ADT Plugin zip file (do not unpack it):

Package	Size	MD5 Checksum
ADT-21.1.0.zip	13564671 bytes	f1ae183891229784bb9c33bcc9c5ef1e

2. Start Eclipse, then select **Help > Install New Software**.
3. Click **Add**, in the top-right corner.
4. In the Add Repository dialog, click **Archive**.
5. Select the downloaded ADT-21.1.0.zip file and click **OK**.
6. Enter "ADT Plugin" for the name and click **OK**.
7. In the Available Software dialog, select the checkbox next to Developer Tools and click **Next**.
8. In the next window, you'll see a list of the tools to be downloaded. Click **Next**.
9. Read and accept the license agreements, then click **Finish**.
10. If you get a security warning saying that the authenticity or validity of the software can't be established, click **OK**.
11. When the installation completes, restart Eclipse.

To update your plugin once you've installed using the zip file, you will have to follow these steps again instead of the default update instructions.

For Linux users:

If you encounter this error when installing the ADT Plugin for Eclipse:



...then your development machine lacks a suitable Java VM. Installing Sun Java 6 will resolve this issue and you can then reinstall the ADT Plugin.

INSTALLING VIRTUAL MACHINE FOR ANDROID JELLY BEAN (EMULATOR)

Initially, install VirtualBox on your Windows PC.

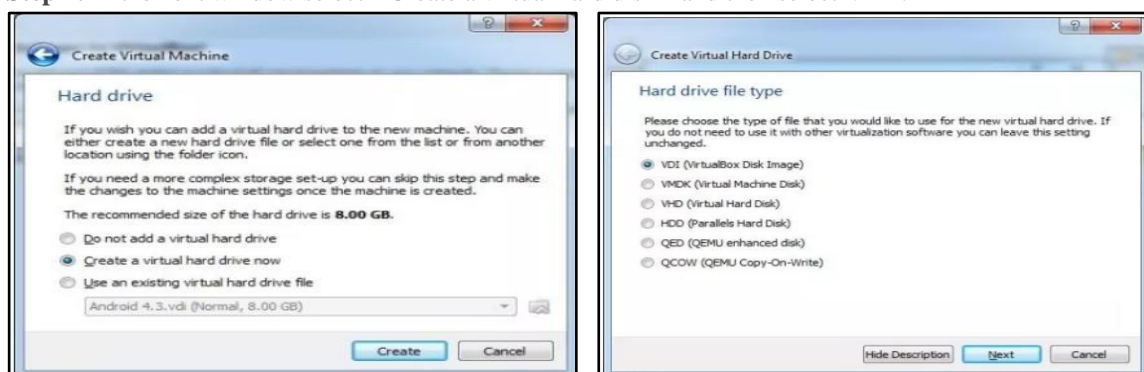
Instructions to Install Android 4.3 on a windows computer

Step 1: Install and open **Virtualbox**.

Step 2: Click on new and enter a name and the operating system details for the virtual machine. Select type as Linux and version as other and click next.

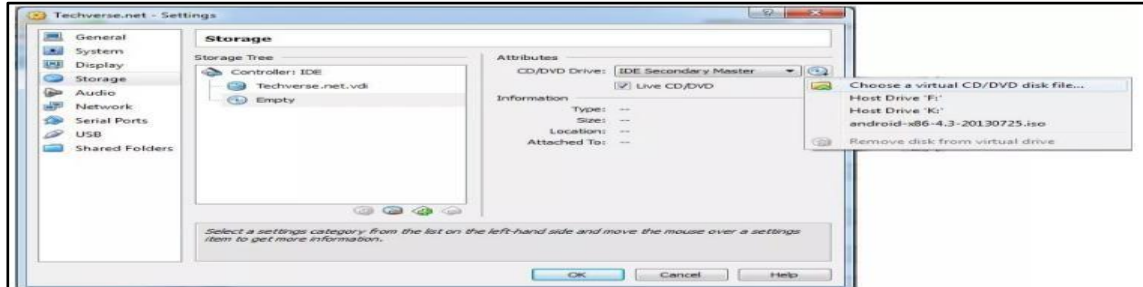
Step 3: Enter the amount of ram you would like to allot for the virtual machine and click next. Android 4.3 requires at least 1Gb of ram but its not necessary.

Step 4: In the next window select —Create a virtual hard disk —and then select VDI .



Step 5: According to your space requirement you can either select dynamically allocated or fixed size for your storage space . i selected fixed size because i want to allocate only 8Gb of storage space to android.

Step 6: Your virtual machine is now set . all you need to do is add the location of the Android 4.3 image file . Click on the settings button in virtualbox . Under the settings navigate to storage , below the storage tree select empty and click on the disk image and select **—choose a virtual CD/DVD disk file —**and select the android 4.3 image . Check the Live CD/DVD box and click ok.



Step 7: Double click on your virtual machine to start it and click OK for all the dialog boxes . Select Install Android-X86 to hard disk and click OK for all the dialog boxes .

Step 8: in the next window you have to create a partition for installing Android . The new partition will not mess up anything with your windows computer . From now onwards you have to use your up , down , left and right keys on your keyboard to Select **—create/modify partitions —**and click OK .

Step 9: In the next windows select new > primary and then specify the size of the new partition

Step 10: Your new partition has been created . Select write and press enter and type **—yes —**and press enter again when prompted . In the next window select quit and press enter.

Step 11 : In the next window select the Sda1 and press enter. select the et3 file system and press enter . When prompted to install grub loader select yes. Select now when prompted to make system directory as read-write .

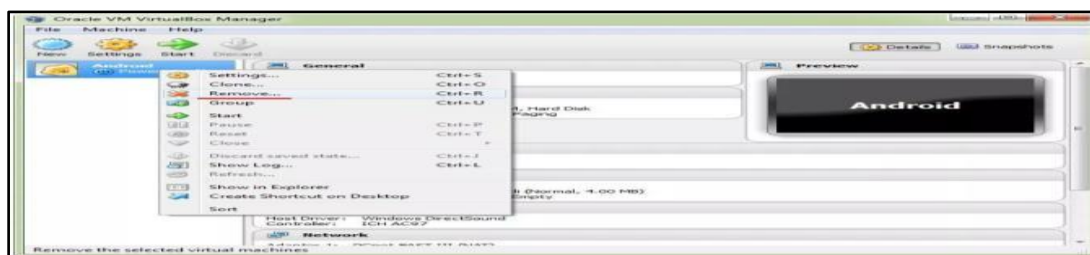
Step 12: Now android 4.3 is successfully installed on your virtual machine . select Run Android 4.3 and press enter . click OK for any other dialog boxes that appear . You will now see the android loading screen .

Step 13: Select your language and enter , now fill in the Gmail details and all the details that are asked .

Step 14: Now We have successfully installed Android 4.3 on windows computer .



In order to uninstall the android 4.3 virtual box, right click on the virtual machine and select remove. Next select **—delete all files—**to remove Android 4.3 completely from PC.

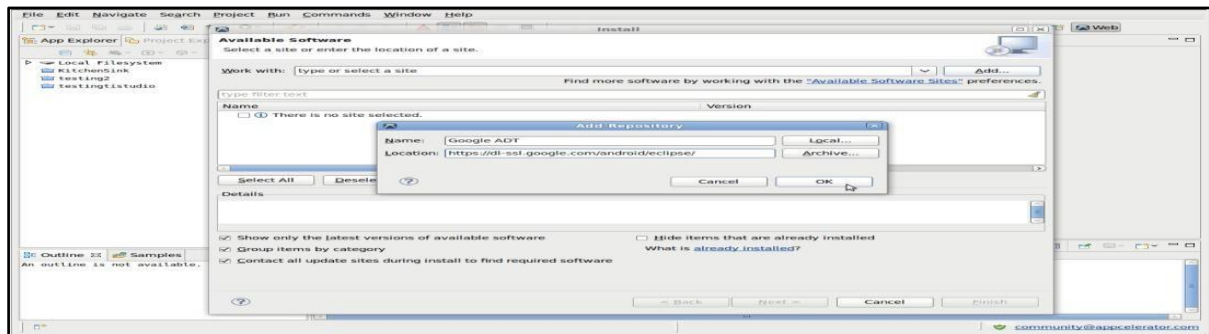


CONFIGURING THE INSTALLED TOOLS

Install the Eclipse Java Development Tools plugin

Install the ADT plugin

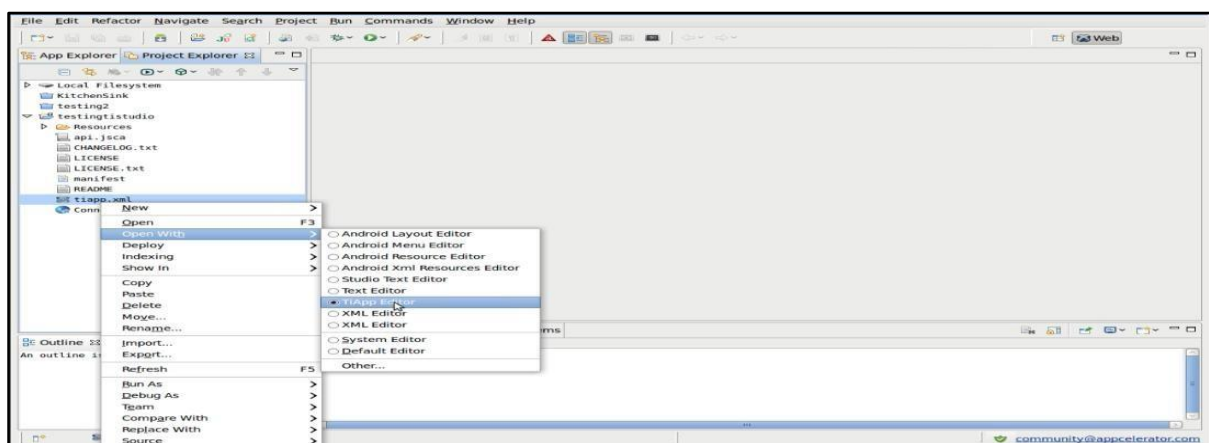
- On Google's official Android Developers website, under the section [Downloading the ADT Plugin](#), copy the update URL to the clipboard
- From the Studio menu bar, select **Help > Install New Software...**
- Click the **Add** button, to add the Google ADT Plugin update site
- In the **Name** field, enter something descriptive, such as **Google ADT**
- Paste the Google ADT update site URL, copied to the clipboard in the previous step, into the **Location** field
- Click the **OK** button



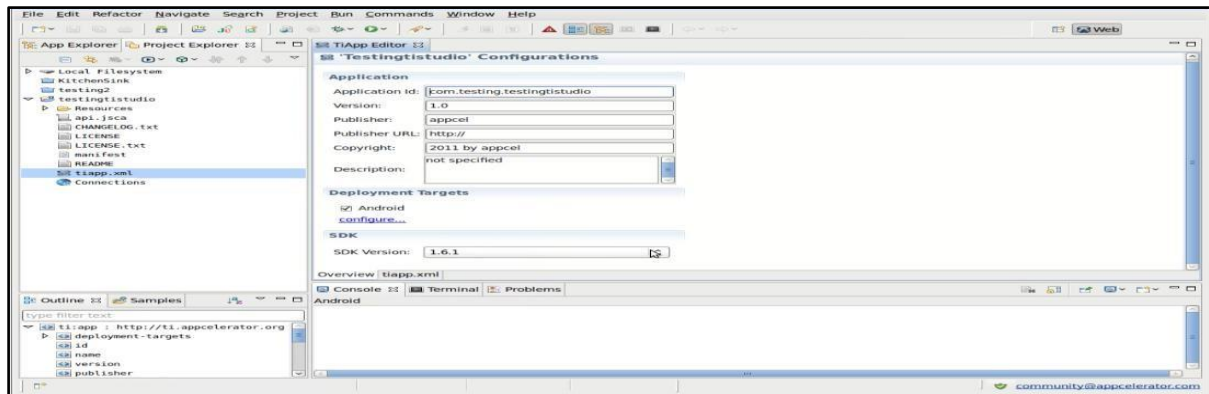
- Using the **Work with** drop-down menu, select the **Google ADT** entry that you added in the previous step
- Wait for the package list to be populated
- Select all the resulting ADT packages
- Click the **Next** button
- Click the **Next** button on the **Install Details** screen that follows
- Select each package in turn from the left-hand pane and accept the respective license agreement
- Click the **Finish** button
- Once the installation is complete, click the **Restart Now** button

Configure Studio to use the SDKs

- Using the perspective icon(s) in the top-right hand corner, select the one titled **Web**
- Using the **Project Explorer** tab in the left-hand pane, right-click an existing project and select **Open Project**
- Browse the resulting project file list, right-click tiapp.xml in root of project, select **Open With > TiApp Editor**



- Choose your preferred Titanium SDK version from the **SDK Version** drop-down list
- Check the [Android SDK / Target Android Platform](#) section of the Titanium Compatibility Matrix, to determine which Android versions are compatible with your chosen Titanium SDK. For example, Titanium SDK 1.6.X is compatible with Android versions 1.6 to 2.3. This information will be needed for the configuration in the following steps
- Close tiapp.xml

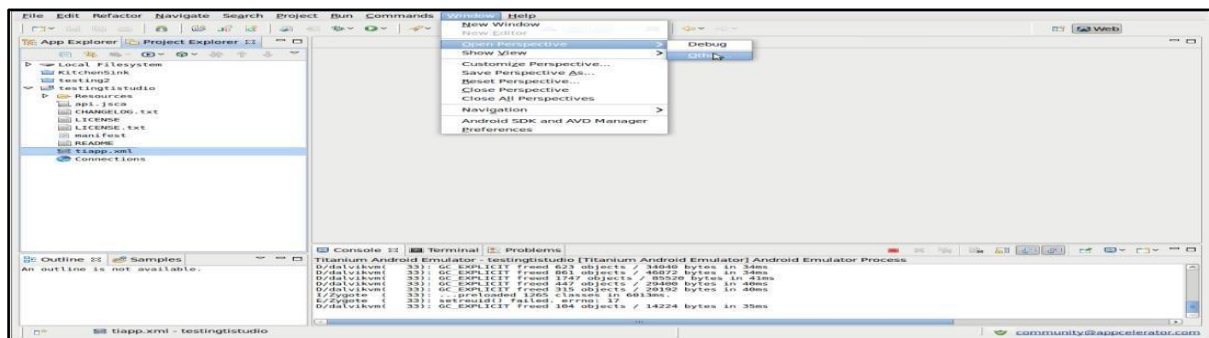


- From the Studio menu, select **Window > Preferences** or **Studio > Preference** for Mac OS X to open the **Preferences** dialog
- Navigate to **Android**
- Click the **Browse...** button to configure the **Android SDK Location**
- Select a target Android SDK from the list, ensuring that its version is within the range compatible with the Titanium SDK you have chosen, as discovered in the previous step
- Navigate to **Studio > Platforms > Android**
- Click the **Browse...** button to configure the **Android SDK Home**
- Select a target Android SDK from the **Default Android SDK** drop-down list, ensuring that its version is within the range compatible with the Titanium SDK you have chosen, as discovered in the previous step
- Click **OK** to save preference changes

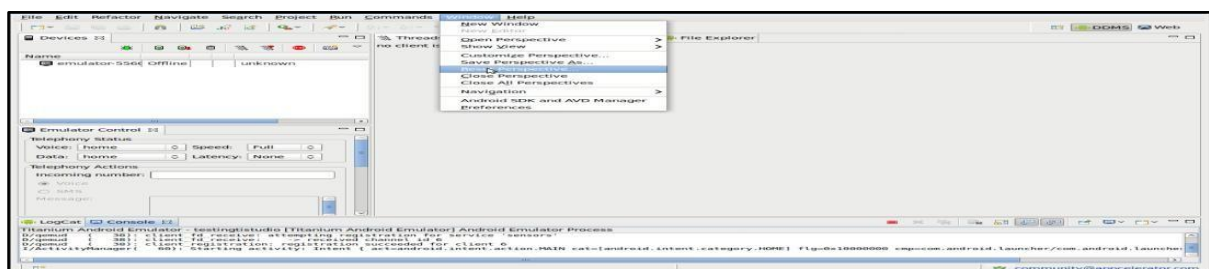
As explained in the [Android SDK / Target Android Platform](#), if you require advanced Android features, such as Maps, remember to choose a target that includes the enhanced Google APIs, listed as **Google APIs** in the **Default Android SDK** list

Launch the emulator and app

- Using the **Project Explorer**, select the project that was opened earlier
- Using the **Launch** toolbar button, located between the **Project Explorer** tab and its file list, select **Android Emulator** to launch the project app
- Add the **DDMS** perspective button
- While the emulator boots, open the ADT perspective. From the Studio menu bar, select the **Window > Open Perspective > Other...**



- Select **DDMS** (which stands for, Dalvik Debug Monitor Server) from the list of available perspectives
- To ensure that the perspective is in its default state, select the **Window > Reset Perspective...** menu
- Click the **OK** button, when the resulting *Do you want to reset the current DDMS perspective to its defaults?* dialog displays



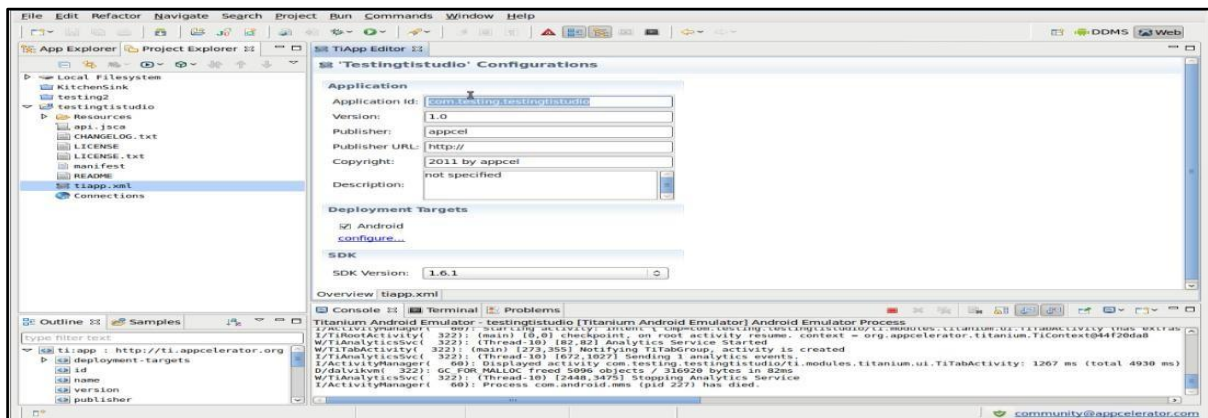
- Click the **LogCat** viewer tab above the bottom pane, to watch the console output while the emulator boots
- If you wish, you may relocate this tab to the main pane, next to the **File Explorer** tab, using a simple drag and drop gesture
- To show only Ti.API log messages, create a logcat filter using the green *plus* icon in the logcat toolbar and set the by Log Tag field to "TiAPI". See [Reading and Writing Logs](#) for more information about logcat filters.
- Once booted, unlock the emulator and wait for the app to launch



- Now that the emulator is running, select it from the list of devices in the left-hand pane, and inspect it using the tools
- For example, select the File Explorer tab, and navigate to the directory `/data/data/yourAppId`

The `/data/data/yourAppId` directory is equivalent to [Titanium.Filesystem.applicationDataDirectory](#)

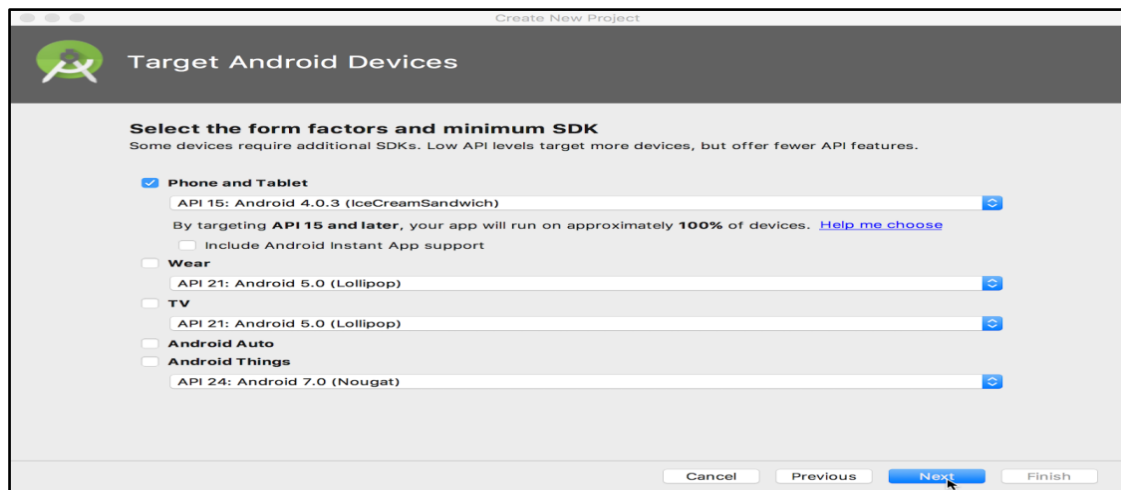
`AppId` was defined when the project was created, as shown in the **TiApp Editor** (see below)



CREATING AN ANDROID PROJECT

CREATE THE APP PROJECT

1. Open Android Studio if it is not already opened.
2. In the main **Welcome to Android Studio** window, click **Start a new Android Studio project**.
3. In the **Create Android Project** window, enter **Hello World** for the **Application name**.
4. Verify that the default **Project location** is where you want to store your Hello World app and other Android Studio projects, or change it to your preferred directory.
5. Accept the default **android.example.com** for **Company Domain**, or create a unique company domain. If you are not planning to publish your app, you can accept the default. Be aware that changing the package name of your app later is extra work.
6. Leave unchecked the options to **Include C++ support** and **Include Kotlin support**, and click **Next**.
7. On the **Target Android Devices** screen, **Phone** and **Tablet** should be selected. Ensure that **API 15: Android 4.0.3 IceCreamSandwich** is set to Minimum SDK; if not, use the popup menu to set it.



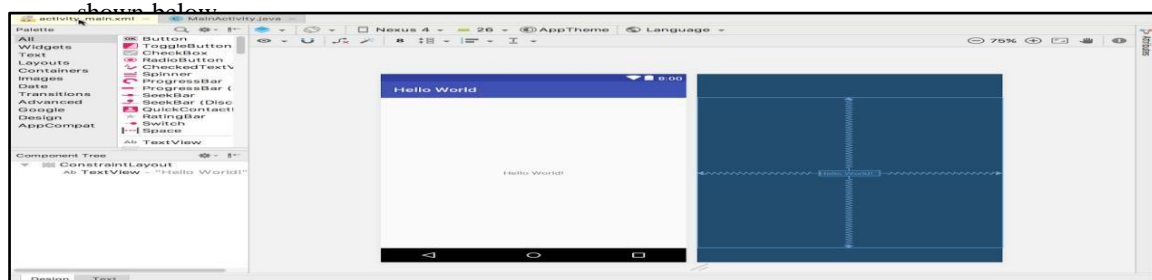
These are the settings. As of this writing, these settings make Hello World app compatible with 97% of Android devices active on the Google Play Store.

8. Leave unchecked the **Include Instant App support** and all other options. Then click **Next**. If your project requires additional components for your chosen target SDK, Android Studio will install them automatically.
9. The **Add an Activity** window appears. An [Activity](#) is a single, focused thing that the user can do. It is a crucial component of any Android app. An Activity typically has a layout associated with it that defines how UI elements appear on a screen. Android Studio provides Activity templates to help you get started. For the Hello World project, choose **Empty Activity** as shown below, and click **Next**.
10. The **Configure Activity** screen appears (which differs depending on which template you chose in the previous step). By default, the empty Activity provided by the template is named MainActivity. You can change this if you want, but this lesson uses MainActivity.
11. Make sure that the **Generate Layout file** option is checked. The layout name by default is activity_main. You can change this if you want, but this lesson uses activity_main.
12. Make sure that the **Backwards Compatibility (App Compat)** option is checked. This ensures that your app will be backwards-compatible with previous versions of Android.
13. Click **Finish**.

Android Studio creates a folder for your projects, and builds the project with [Gradle](#).

The Android Studio editor appears. Follow these steps:

1. Click the **activity_main.xml** tab to see the layout editor.
2. Click the layout editor **Design** tab, if not already selected, to show a graphical rendition of the layout as shown below.



3. Click the **MainActivity.java** tab to see the code editor as shown below.

Explore the Project > Android pane

1. If not already selected, click the **Project** tab in the vertical tab column on the left side of the Android Studio window. The Project pane appears.
2. To view the project in the standard Android project hierarchy, choose **Android** from the popup menu at the top of the Project pane, as shown below.


Explore the manifests folder

The manifests folder contains files that provide essential information about your app to the Android system, which the system must have before it can run any of the app's code.

1. Expand the **manifests** folder.
2. Open the **AndroidManifest.xml** file.

The AndroidManifest.xml file describes all of the components of your Android app. All components for an app, such as each Activity, must be declared in this XML file. In other course lessons you will modify this file to add features and feature permissions. For an introduction, see [App Manifest Overview](#).

RUN ON EMULATOR

1. Lets create an android virtual device (avd). In order to run an emulator on your computer, you have to create a configuration that describes the virtual device. In Android Studio, select **Tools > Android > AVD Manager**, or click the AVD Manager icon  in the toolbar. The **Your Virtual Devices** screen appears. If you've already created virtual devices, the screen shows them; otherwise you see a blank list.

2. Click the **+Create Virtual Device**. The **Select Hardware** window appears showing a list of pre configured hardware devices. For each device, the table provides a column for its diagonal display size (**Size**), screen resolution in pixels (**Resolution**), and pixel density (**Density**).


3. Choose a device such as **Nexus 5x** or **Pixel XL**, and click **Next**. The **System Image** screen appears.

4. Click the **Recommended** tab if it is not already selected, and choose which version of the Android system to run on the virtual device (such as **Oreo**). Click the link to start the download, and click **Finish** when it's done.

5. After choosing a system image, click **Next**. The **Android Virtual Device (AVD)** window appears. You can also change the name of the AVD. Check your configuration and click **Finish**.

Run the app on the virtual device

Let's run your Hello World app.

1. In Android Studio, choose **Run > Run app** or click the **Run** icon  in the toolbar.
2. The **Select Deployment Target** window, under **Available Virtual Devices**, select the virtual device, which you just created, and click **OK**



The emulator starts and boots just like a physical device. Your app builds, and once the emulator is ready, Android Studio will upload the app to the emulator and run it.

DEPLOY IT ON USB-CONNECTED ANDROID DEVICE

Configure the Android device

In order to install an application directly to your device, you need to configure it to use a USB connection. The configuration settings vary by device.

For Android 4.2 and later devices, you need to enable **Developer options** by opening **Settings**, click **About** then click the **Build number** item seven items. If you do not do this, you will not see the **Developer options** item in **Settings**.

1. Open **Settings**.
2. Click **Security**.
3. Enable **Unknown sources**, that is, check this option. This permits the device to install apps that do not originate from Google Play.
4. Back out to **Settings**.
5. Click **Developer options**.
6. If available: Set the switch in the title bar to on.
7. Enable **USB debugging**, that is, check this option. This permits the device to install apps over a USB connection.
8. Optional: Enable **Stay awake**, that is, check this option. This option keeps the screen on and disables the lock screen while the device is connected to USB.
9. Optional: Enable **Allow mock locations**, that is, check this option. This option creates fake GPS locations to test location services.
10. Back out of or close **Settings**.

Install the USB driver (Windows only)

Developers on Windows may need to install a USB driver specific to the manufacturer and model of the device on which they'll be testing. The driver enables your Windows computer to communicate with your Android device. Google provides download links to the drivers at [Android Developer: OEM USB Drivers](#).

Connect the device

Connect the Android device to your computer using an USB cord. Note that some USB cables are only power cables and do not allow communications with the device. Make sure you use a USB cable that allows a data connection.

For 4.2 devices, an "Allow USB debugging?" dialog will appear once connected via USB. Click the **OK** button.

Deploy the application using Axway Appcelerator Studio

Once you have configured your device and connected it to your computer's USB port, you are ready to deploy your app to it.



In Studio, first select the project in the **Project Explorer** view, then in the global tool bar, select **Run** from the **Launch Mode** drop-down list and an Android device from the **Target** drop-down list under the **Android Application Installer** category. If the **Launch Automatically** option is enabled under the **Target** drop-down list, the application will be automatically launched after the device is selected. If not, you need to click the **Run** button to start the build process. Your app will be built, installed to your device and automatically launched.

MULTIPLE SCREEN SIZES

Android devices come in all shapes and sizes, so your app's layout needs to be flexible. That is, instead of defining your layout with rigid dimensions that assume a certain screen size and aspect ratio, your layout should gracefully respond to different screen sizes and orientations.

The best way to create a responsive layout for different screen sizes is to use [Constraint Layout](#) as the base layout in your UI. [Constraint Layout](#) allows you to specify the position and size for each view according to spatial relationships with other views in the layout. This way, all the views can move and stretch together as the screen size changes.

The easiest way to build a layout with [Constraint Layout](#) is to use the Layout Editor in Android Studio. It allows you to drag new views to the layout, attach their constraints to the parent view and other sibling views, and edit the view's properties, all without editing any XML by hand.

USER INTERFACE DESIGN

FORM WIDGETS

Widgets enable users to interact with an Android Studio application page. There are various kinds of widgets, such as Buttons and TextViews.

To see all the widgets at your disposal, create a new application project called `—Widgets` and select "empty activity". Call your activity `—MainActivity`.

There are two components of each Android activity: the XML (Extensible Markup Language) design (the beauty) and the Java text (the brains).

On the `activity_main.xml` page, you can see the full widgets palette underneath the various layout options.

As you can see, there are 20 widgets available for you to use. In this guide, we'll discuss TextViews and Buttons, which are probably the most common widgets in Android development.

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.

You can add a text field to your layout with the [EditText](#) object. You should usually do so in your XML layout with a `<EditText>` element.

Text fields can have different input types, such as number, date, password, or email address.

LAYOUTS

A layout defines the visual structure for a user interface, such as the UI for an [activity](#) or [app widget](#). 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.

BUTTON CONTROL

Button is a user interface control which is used to perform an action whenever the user click or tap on it. Buttons in android will contains a text or an icon or both and perform an action when user touches it. Different types of buttons available are **ImageButton**, **ToggleButton**, **RadioButton**.

TOGGLE BUTTONS

A toggle button allows the user to change a setting between two states.

You can add a basic toggle button to your layout with the [ToggleButton](#) object. Android 4.0 (API level 14) introduces another kind of toggle button called a switch that provides a slider control, which you can add with a [Switch](#) object. [SwitchCompat](#) is a version of the Switch widget which runs on devices back to API 7.

SPINNERS / COMBO BOXES

Spinners provide a quick way to select one value from a set. In the default state, a spinner shows its currently selected value. Touching the spinner displays a dropdown menu with all other available values, from which the user can select a new one.

IMAGES

```
public abstract class Image
extends Object implements AutoCloseable
java.lang.Object
↳ android.media.Image
```

A single complete image buffer to use with a media source such as a [MediaCodec](#) or a [CameraDevice](#).

This class allows for efficient direct application access to the pixel data of the Image through one or more [ByteBuffers](#). Each buffer is encapsulated in a [Plane](#) that describes the layout of the pixel data in that plane. Due to this direct access, and unlike the [Bitmap](#) class, Images are not directly usable as UI resources.

MENU

In android, **Options Menu** is a primary collection of menu items for an [activity](#) and it is useful to implement actions that have a global impact on the app, such as Settings, Search, etc.

In case, if we define items for the options menu in both activity or fragment, then those items will be combine and display in UI.

DIALOG

A dialog is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.

The [Dialog](#) class is the base class for dialogs, but you should avoid instantiating [Dialog](#) directly. Instead, use one of the following subclasses:

- [AlertDialog](#): A dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout.
- [DatePickerDialog](#) or [TimePickerDialog](#): A dialog with a pre-defined UI that allows the user to select a date or time.

AGENDA

- Introduction to Android
- Android Studio
- Hello World Application
- Application Components
- Application Resources
- User Interface
- Code Time
- Good UI
- Play Store
- Learn Android



Introduction to Android

ANDROID JOURNEY

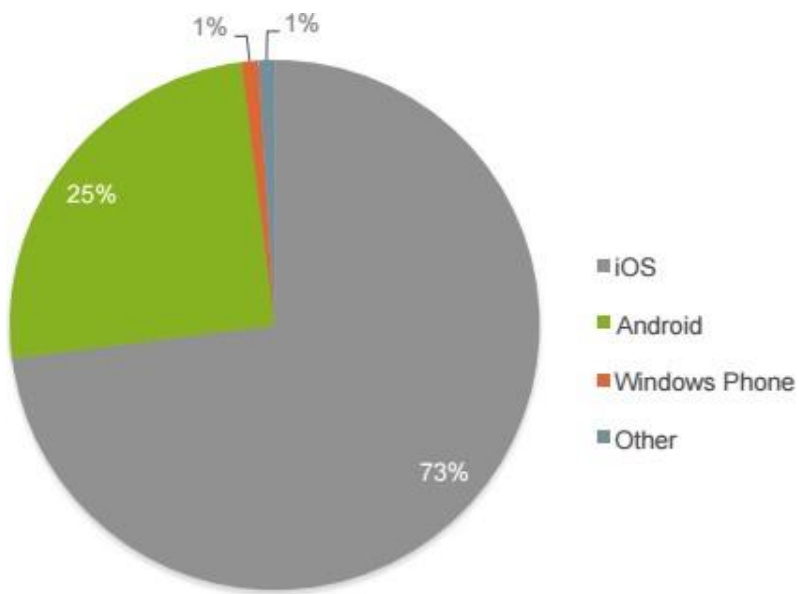
Developed by Google. First
released version 1.5(Cupcake).

Current version - 6.0 (Marshmallow)

Major contributors: Samsung, LG, Vodafone, Acer, Dell, HTC, Sony Ericsson, Intel, Wipro
including others.

Pre-installed Google apps: Gmail, Google Calendar, Google Play, Google Music, Chrome
and others.

Android World



Openness
Customizable
Affordable(low-cost)

WHY NOT ANY OTHER TECHNOLOGY ?

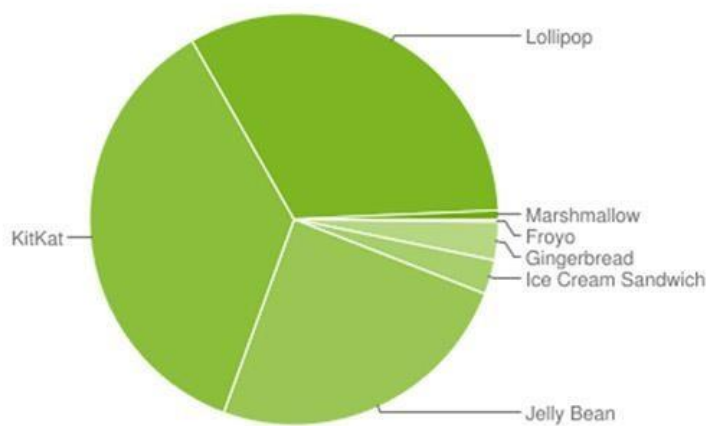
1. Highest paying technology
2. A lot of career opportunities
3. Freshers are in high demand



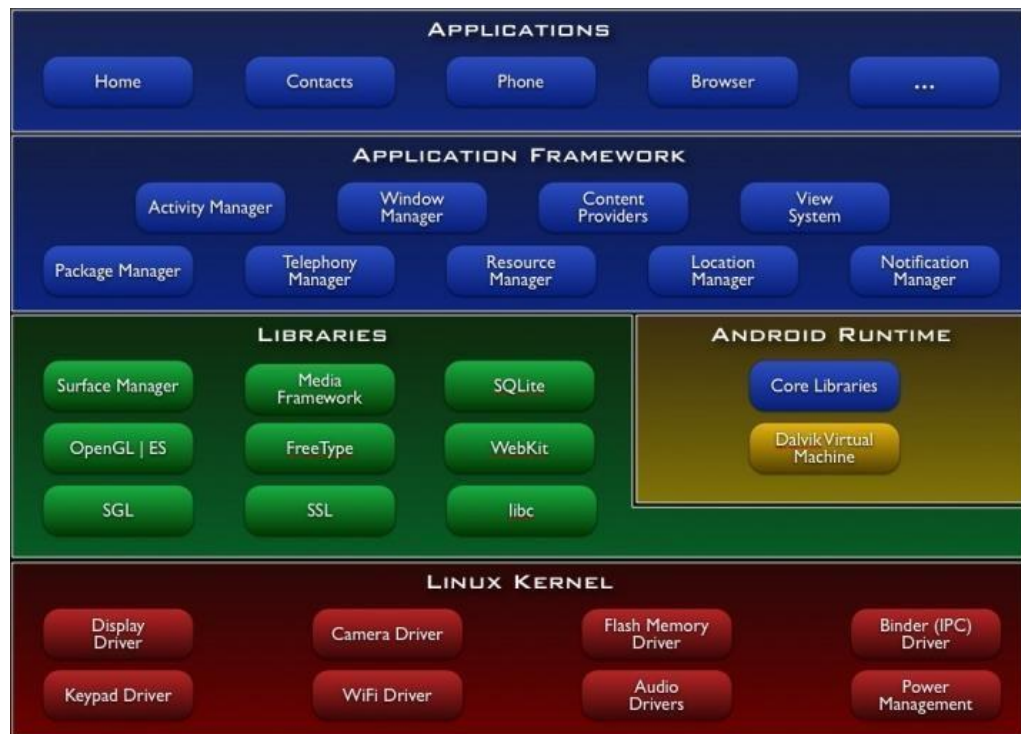
ANDROID VERSIONS



Version	Codename	API	Distribution
2.2	Froyo	8	0.2%
2.3.3 - 2.3.7	Gingerbread	10	3.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	2.7%
4.1.x	Jelly Bean	16	9.0%
4.2.x		17	12.2%
4.3		18	3.5%
4.4	KitKat	19	36.1%
5.0	Lollipop	21	16.9%
5.1		22	15.7%
6.0	Marshmallow	23	0.7%



ANDROID - ARCHITECTURE



ANDROID IS NOT LINUX

- Android is built on the Linux kernel, but Android is not a Linux.
- **Linux kernel** it's a component which is responsible for device drivers, power management, memory management, device management and resource access.
- Native **libraries** such as WebKit, OpenGL, FreeType, SQLite, Media, C runtime library
- (libc) etc.
- **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.
- **App Framework** - Android framework includes Android API's **Applications** - All applications

EACH ANDROID APP LIVES IN ITS OWN SECURITY SANDBOX

- The Android operating system is a multi-user Linux system in which each app is a different user.
- **By default, the system assigns each app a unique Linux user ID.** The system sets permissions for all the files in an app so that only the user ID assigned to that app can access them.
- **Each process has its own virtual machine (VM)**, so an app's code runs in isolation from other apps.
- **By default, every app runs in its own Linux process.** Android starts the process when any of the app's components need to be executed, then shuts down the process when it's no longer needed or when the system must recover memory for other apps.
- **Each app has own lifecycle**

ANDROID STUDIO

STARTING ANDROID DEVELOPMENT



**I DONT KNOW WHO YOU
ARE BUT I LL FIND YOU**

**AND INSTALL ANDROID STUDIO ON
YOUR PC**

memegenerator.net

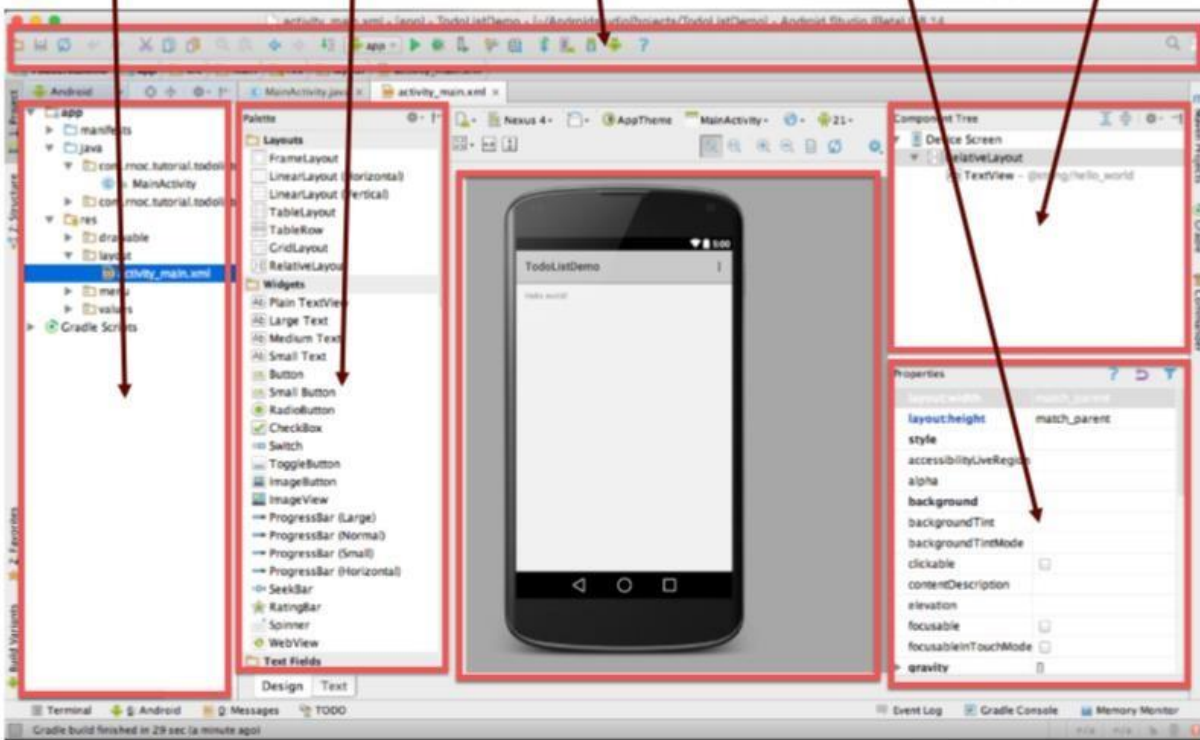
Project View

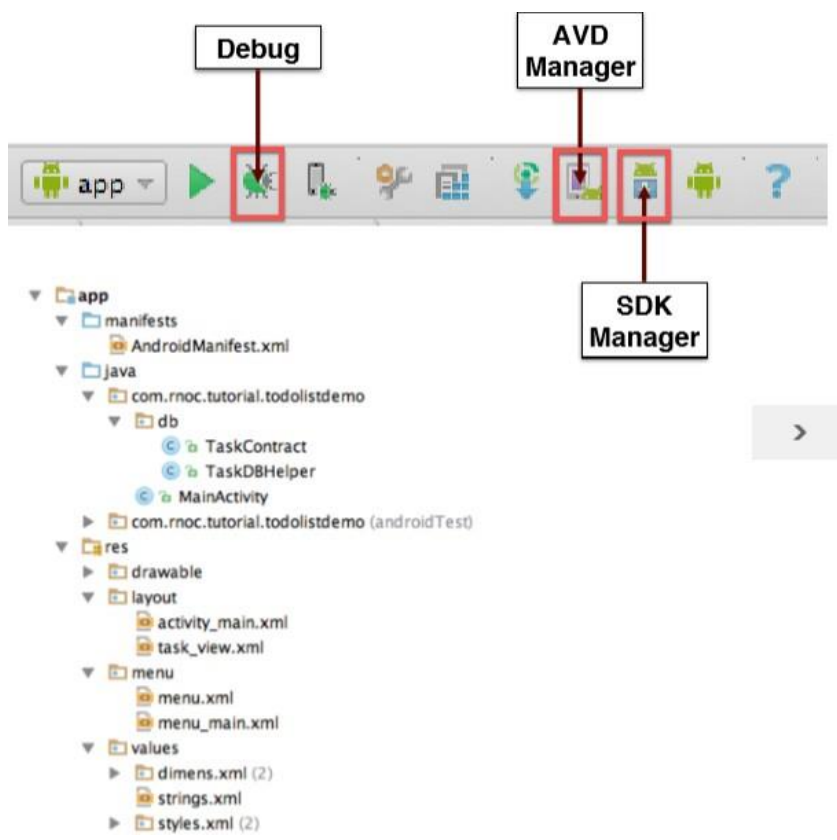
Palette

Toolbar

Properties Editor

Component Tree





Logcat



HELLO WORLD APPLICATION

CREATING A VIRTUAL ANDROID DEVICE



- Open the AVD Manager
 - Click „*Create Virtual Device*“
 - Choose a „Phone“ device from the list
 - Resolution should be lower then your desktop resolution

Nexus 5	4,0"	480x800	hdpi
Nexus One	3,7"	480x800	hdpi
Nexus 6P	5,7"	1440x2560	560dpi
Nexus 6	5,96"	1440x2560	560dpi
Nexus SX	5,2"	1080x1920	420dpi
Nexus 5	4,95"	1080x1920	xxhdpi
Nexus 4	4,7"	768x1280	xhdpi
Galaxy Nexus	4,65"	720x1280	xhdpi
5.4" FWVGA	5,4"	480x854	mdpi
5.1" WVGA	5,1"	480x800	mdpi
4.7" WXGA	4,7"	720x1280	xhdpi

Hello World!

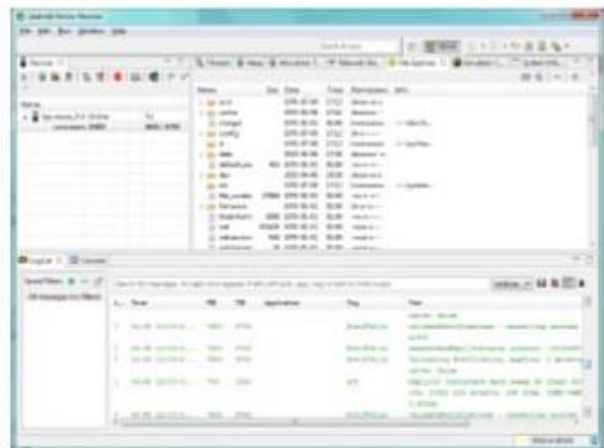
■ Tasks:

- Create a new Android-Project (File > New > New Project)
- Create a run configuration for this project
- Launch your application without any code changes on an emulated or a physical device



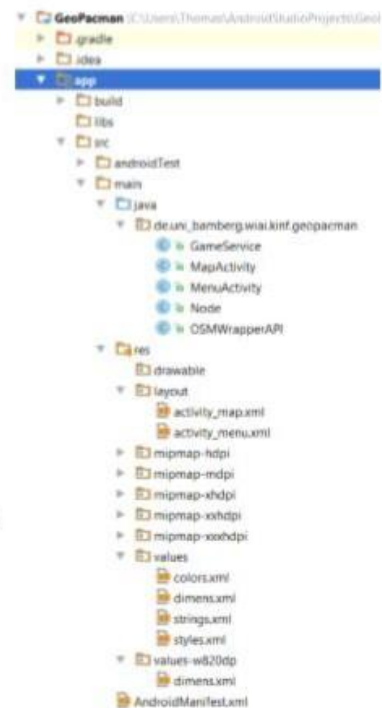
Debugging

- Logging, LogCat
 - Use android.util.Log to
 - LogCat displays Log-Messages
- Android Device Monitor
 - Displays other usefull debugging information like memory, cpu and network usage



Project Structure

- *src Folder*
 - Contains the java source code files
- *res Folder*
 - Mostly XML configuration files
 - E.G.: layout, strings
- *R.java class*
 - Automatically generated
 - All resource IDs are defined in this class
 - Provides access to resources
 - `String helloString = getString(R.string.hello);`
 - `setContentView(R.activity_main);`



Gradle

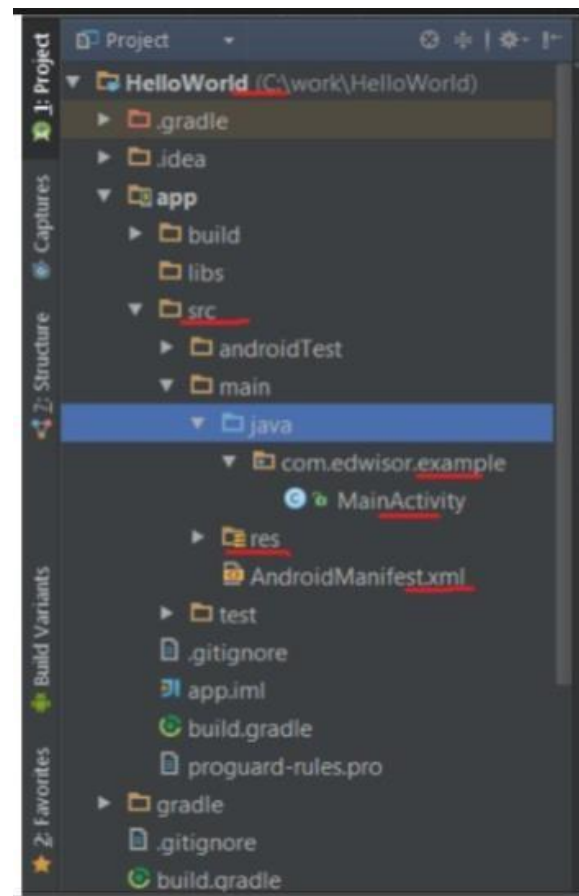
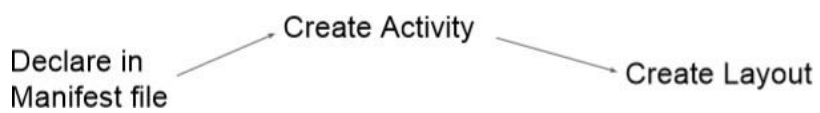
- Open source build automation system
- Uses a Groovy-based domain-specific language
- Designed for multi-project builds which can grow to be quite large
- Android Studio projects contain a top-level build file and a build file for each module, both called *build.gradle*
- Android-specific build options as well as app configurations and dependencies can be adjusted



```
android {  
    compileSdkVersion 23  
    buildToolsVersion "23.0.2"  
  
    defaultConfig {  
        applicationId "de.geopacman"  
        minSdkVersion 15  
        targetSdkVersion 23  
        versionCode 1  
        versionName "1.0"  
    }  
}
```

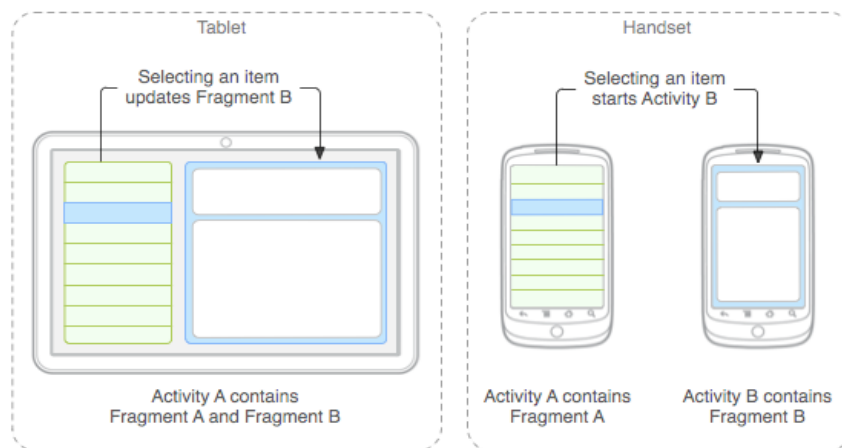
HELLO WORLD APPLICATION

1. Activity
2. Manifest File
3. Layout



Activities

- Represents a single screen in an App.
- Hosts all the view components of the screen like button, textview and popups.
- Hosts all the logic of user interaction.
- Have their own lifecycle.



Activities

Every Activity will inherit an “Activity”.

Extending a class.

Accessing inherited methods and member variables.

Overriding methods.



1 Activity

```
public class MainActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_main);  
    }  
}
```

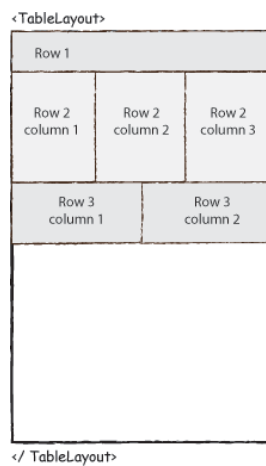
2.LAYOUT

Linear Layout

Relative Layout

Grid Layout

- Layout Items
 - Image View
 - Button
 - Text View
 - List View



3.ANDROID MANIFEST FILE

Declare Application Name

Declare Activates

Declare Application Theme

Declare Application Icon

Declare Other Components

Declare Permissions

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example" 1
    android:versionCode="1"
    android:versionName="1.0">

    <uses-sdk android:minSdkVersion="15"/> 2

    <application android:label="@string/app_name"
        android:debuggable="true" 3
        android:icon="@drawable/ic_launcher">

        <activity android:name="MainActivity"
            android:label="@string/app_name" 4
            <intent-filter>
                <action android:name="android.intent.action.MAIN"/>
                <category android:name="android.intent.category.LAUNCHER"/>
            </intent-filter>
        </activity>

        <service>...</service> 5

        <receiver>...</receiver> 6

        <provider>...</provider> 7

    </application>
</manifest>
```

APPLICATION COMPONENTS

APPLICATION COMPONENTS

- **Activities**
- Intent, Intent Filters
- **Broadcast Receivers**
- **Services**
- **Content Providers**
- Processes and Threads

ACTIVITIES

An **Activity** is an application component that provides a screen with which users can interact. *Activity = Window*

How to create an **Activity**

1. Create a layout for your screen

```
<LinearLayout ... >
    <EditText ... />
    <EditText ... />
    <Button ... />
</LinearLayout>
```


How to create an [Activity](#)

2. Create class which extends class **Activity** and override **onCreate()** method

```
public class LoginActivity extends Activity {  
  
    @Override  
    protected void onCreate(Bundle savedInstanceState) {  
        super.onCreate(savedInstanceState);  
        setContentView(R.layout.activity_login);  
    }  
}
```

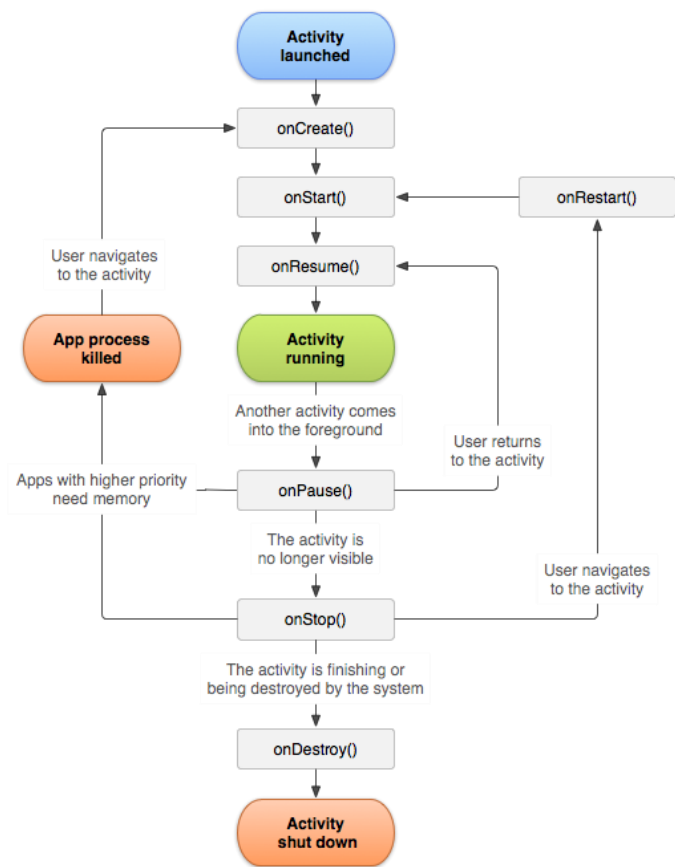
3. Declare your activity in the manifest file

```
<activity  
    android:name=".LoginActivity"  
    android:label="@string/app_name">  
    <intent-filter>  
        <action android:name="android.intent.action.MAIN"/>  
        <category android:name="android.intent.category.LAUNCHER"/>  
    </intent-filter>  
</activity>
```

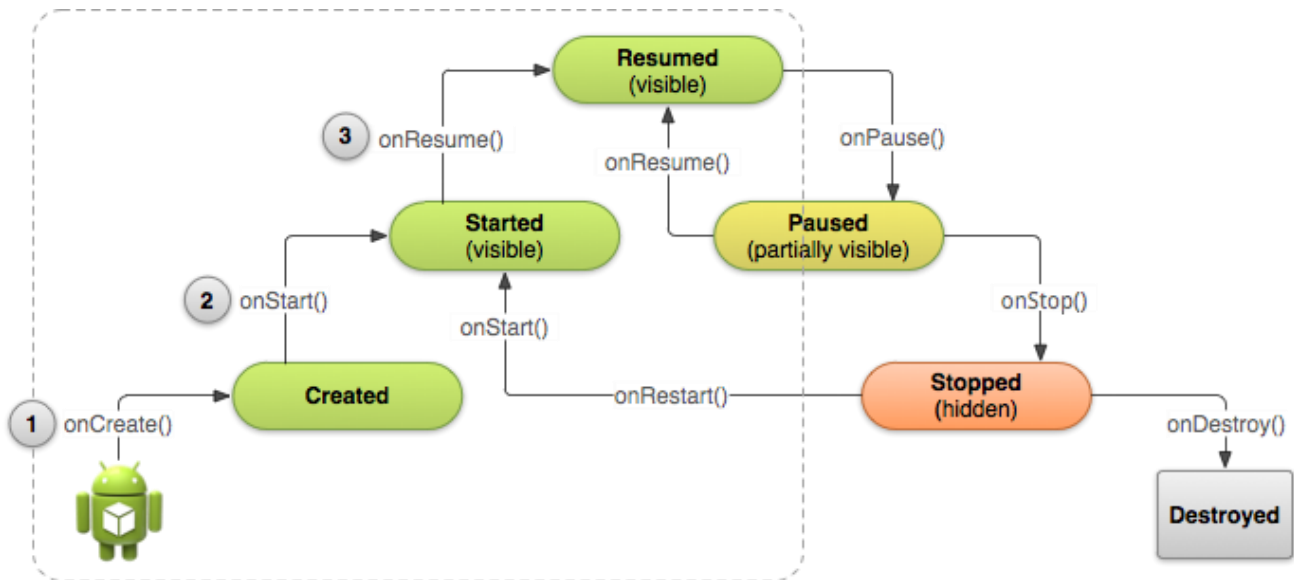
Activity Lifecycle

When an activity transitions into and out of the different states it is notified through various callback methods.

[Complete Android Fragment & Activity Lifecycle](#)



Activity Lifecycle

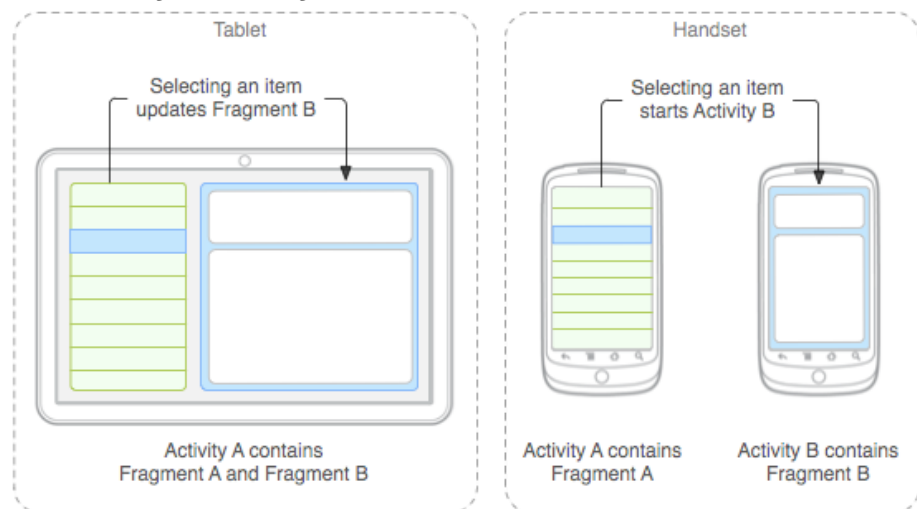


Fragments

A **Fragment** represents a behavior or a portion of user interface in an **Activity**.

A Fragment must always be embedded in an activity and the **fragment's lifecycle is directly affected by the host activity's lifecycle**.

Android introduced fragments in Android 3.0 (API level 11), primarily to support more dynamic and flexible UI designs on large screens, such as tablets.



How to add a Fragment into app

Create a Fragment Class

```
public class OrderSpinnerFragment extends Fragment
{
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState)
    {
        return inflater.inflate(R.layout.fragment_orders_spinner, container, false);
    }
}
```



Adding a fragment to an activity

- Declare the fragment inside the activity's layout file.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout ...>
    <fragment android:name="com.example.news.ArticleListFragment"
        android:id="@+id/list"
        android:layout_weight="1"
        android:layout_width="0dp"
        android:layout_height="match_parent" />
</LinearLayout>
```

- Or, programmatically add the fragment to an existing ViewGroup.

```
private void addUserInfoFragment() {
    FragmentManager fragmentManager = getFragmentManager();
    FragmentTransaction fragmentTransaction =
        fragmentManager.beginTransaction();

    UserInfoFragment fragment = new UserInfoFragment();
    fragmentTransaction.add(R.id.fragment_container, fragment);
    fragmentTransaction.commit();
}
```

INTENT AND INTENT FILTERS

An **Intent** is a messaging object you can use to request an action from another app component. Although intents facilitate communication between components in several ways.

- start **Activity**
- start **Service**
- deliver **Broadcast**

There are two types of intents:

- **Explicit intents** specify the component to start by name
- **Implicit intents** declare a general action to perform, which allows a component from another app to handle it

Building an Intent

An **Intent** object carries information that the Android system uses to determine which component to start (such as the exact component name or component category that should receive the intent), plus information that the recipient component uses in order to properly perform the action (such as the action to take and the data to act upon).

The primary information contained in an Intent is the following:

- Component name - The name of the component to start.
- Action - The general action to be performed, such as ACTION_VIEW, ACTION_EDIT, ACTION_MAIN, etc.
- Data - The data to operate on, such as a person record in the contacts database, expressed as a Uri.
- Category - Gives additional information about the action to execute.
- Type - Specifies an explicit type (a MIME type) of the intent data.
- Extras - This is a Bundle of any additional information.

Example of explicit intent

```
public void startUserDetailsActivity(long userId) {
    Intent userDetailsIntent = new Intent(this, UserDetailsActivity.class);
    userDetailsIntent.putExtra(USER_ID, userId);
    startActivity(userDetailsIntent);
}
```

Example of implicit intent

```
public void sendMessageIntent(String textMessage) {
    Intent sendIntent = new Intent();
    sendIntent.setAction(Intent.ACTION_SEND);
    sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
    sendIntent.setType("text/plain");
    startActivity(sendIntent);
}
```

APPLICATION RESOURCES

APPLICATION RESOURCES

You should always externalize resources such as images and strings from your application code, so that you can maintain them independently. Externalizing your resources also allows you to provide alternative resources that support specific device configurations such as different languages or screen sizes, which becomes increasingly important as more Android-powered devices become available with different configurations.

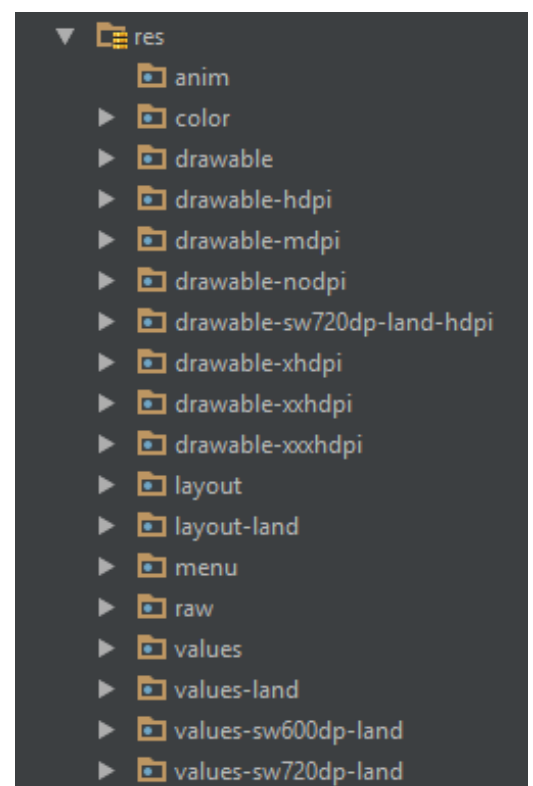
```
MyProject/  
  src/  
    MyActivity.java  
  res/  
    drawable/  
      graphic.png  
    layout/  
      main.xml  
      info.xml  
    mipmap/  
      icon.png  
    values/  
      strings.xml
```

RESOURCE DIRECTORIES SUPPORTED INSIDE PROJECT RES/ DIRECTORY

Directory	Resource Type
ani ma tor/ ani m/	XMLfiles that define property animations and tween animations.
color/	XMLfiles that define a state list of colors.
drawable/	Bitmap files (.png, .9.png, .jpg, .gif) or XMLfiles.
layout/	XMLfiles that define a userinterface layout.
menu/	XMLfiles that define application menus,such as an Options Menu, Context Menu, or Sub Menu.
values/	XMLfiles that contain simple values, such as strings, integers, and colors.

PROVIDING ALTERNATIVE RESOURCES

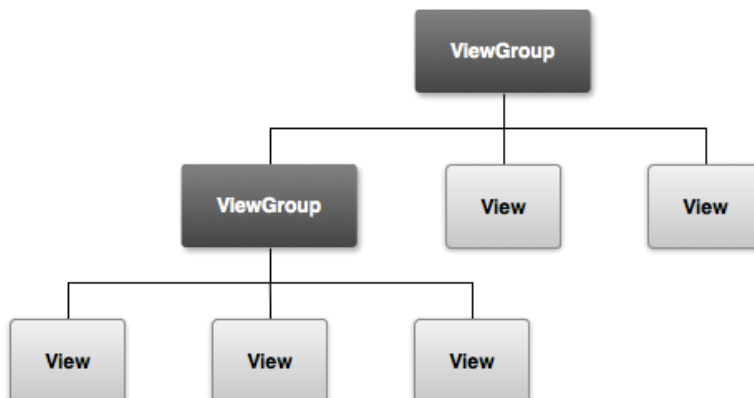
Almost every application should provide alternative resources to support specific device configurations. For instance, you should include alternative drawable resources for different screen densities and alternative string resources for different languages. At runtime, Android detects the current device configuration and loads the appropriate resources for your application.



USER INTERFACE

USER INTERFACE

- All user interface elements in an Android app are built using **View** and **ViewGroup** objects. A **View** is an object that draws something on the screen that the user can interact with. A **ViewGroup** is an object that holds other **View** (and **ViewGroup**) objects in order to define the layout of the interface.



LAYOUTS

A layout defines the visual structure for a user interface, such as the UI for an activity or app widget. You can declare a layout in two ways:

- Declare UI elements in XML.
- Instantiate layout elements at runtime.

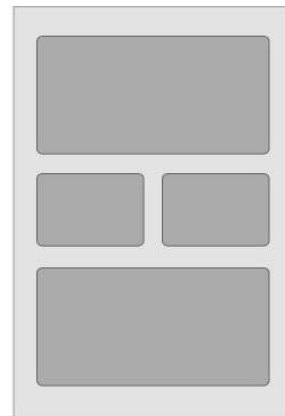
```
<LinearLayout ... >
    <EditText ... />
    <EditText ... />
    <Button ... />
</LinearLayout>
```

```
public class LoginActivity extends Activity{
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login);
    }
}
```


LAYOUTS

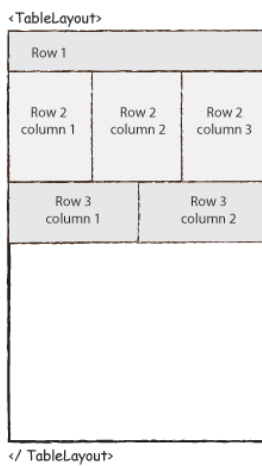


LinearLayout is a view group that aligns all children in a single direction, vertically or horizontally.



RelativeLayout is a view group that displays child views in relative positions.

LAYOUTS



TableLayout is a view that groups views into rows and columns.

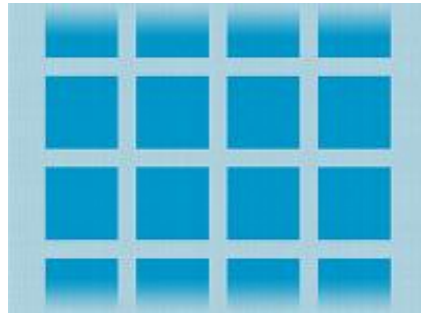


FrameLayout is a placeholder on screen that you can use to display a single view.

LAYOUTS



ListView is a view group that displays a list of scrollable items.



GridView is a ViewGroup that displays items in a two-dimensional, scrollable grid.

Comparing Android UI Elements to Swing UI Elements

Activities

Comparing Android UI Elements to Swing UI Elements

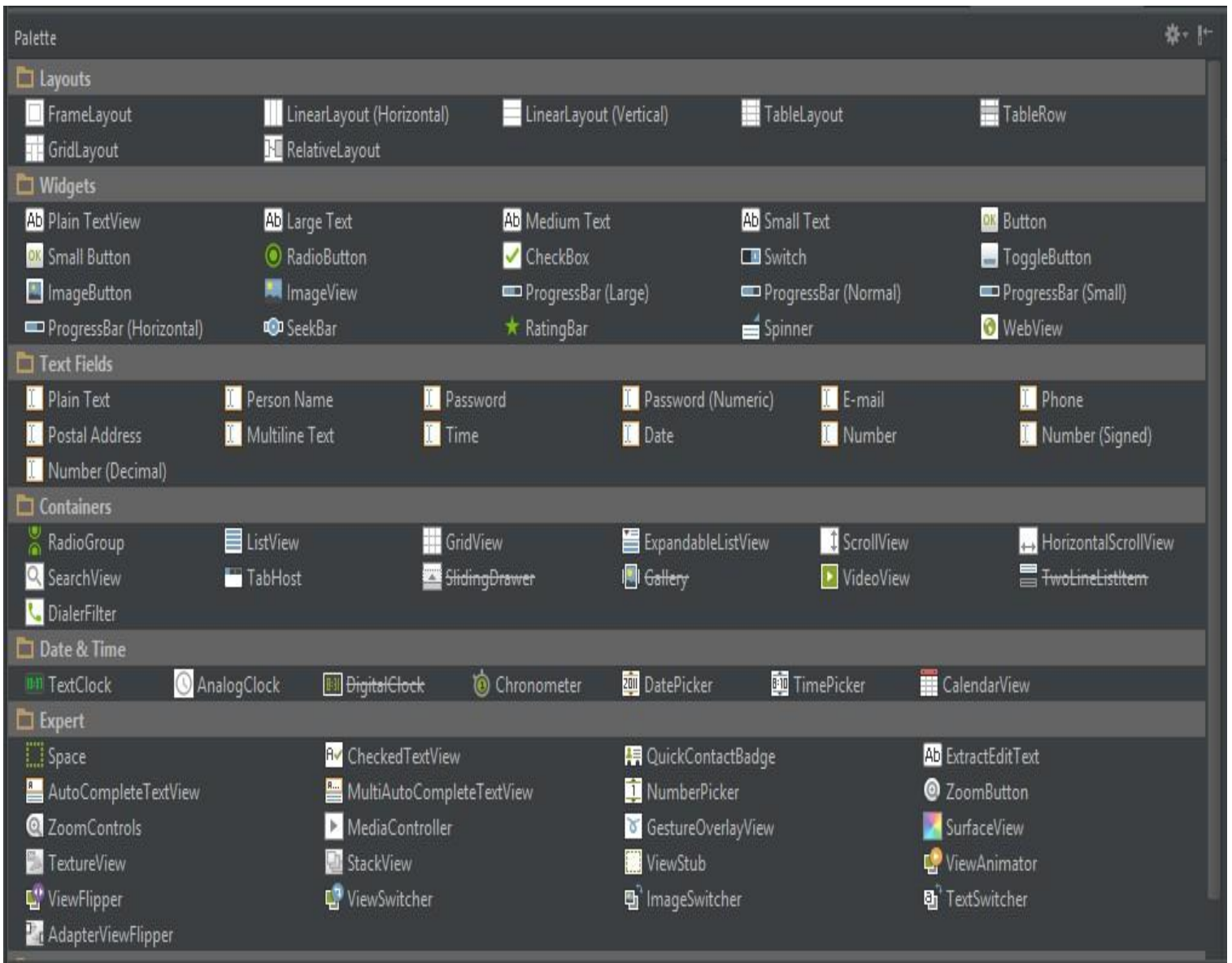
- ❑ **Activities** in Android refers *almost* to a **(J)Frame** in Swing.
- ❑ **Views** in Android refers to **(J)Components** in Swing.
- ❑ **TextViews** in Android refers to a **(J)Labels** in Swing.
- ❑ **EditTexts** in Android refers to a **(J)TextFields** in Swing.
- ❑ **Buttons** in Android refers to a **(J)Buttons** in Swing.

```
// Android
myView.setOnClickListener(new OnClickListener() { ...

// Swing
myButton.addActionListener(new ActionListener() { ...
```

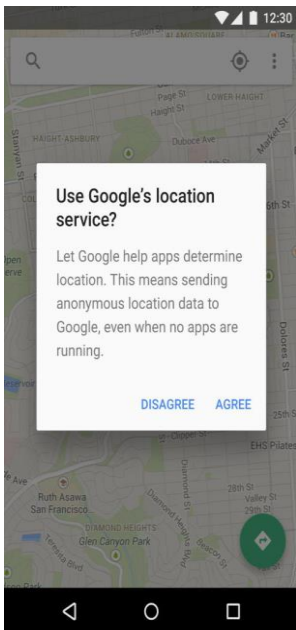
```
<EditText
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:textColor="@color/opaque_red"
    android:text="Hello, World!" />
```

WIDGETS

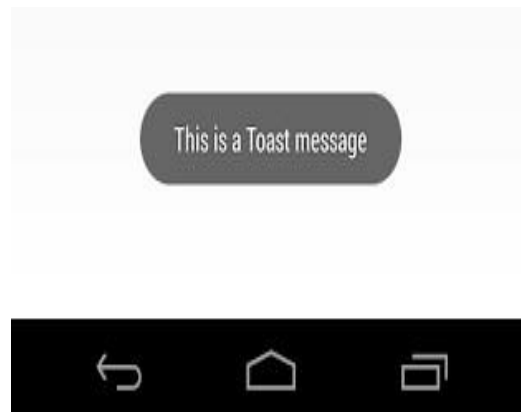


DIALOGS AND TOASTS

A **Dialog** is a small window that prompts the user to make a decision or enter additional information. A dialog does not fill the screen and is normally used for modal events that require users to take an action before they can proceed.



A **Toast** provides simple feedback about an operation in a small popup.



EVENTS WITH UI

BUTTON

OnClickListener

```
mButton.setOnClickListener(new View.OnClickListener() {  
    @Override  
    public void onClick(View v) {  
        //Do the required task  
    }  
});
```




But first,
let's code!

GOOD UI

WHY GOOD UI IS IMPORTANT?

Smooth performance.

Attracts users and keeps them.

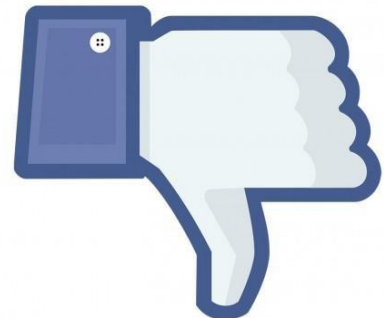
Does not irritate users.

Does what is expected.

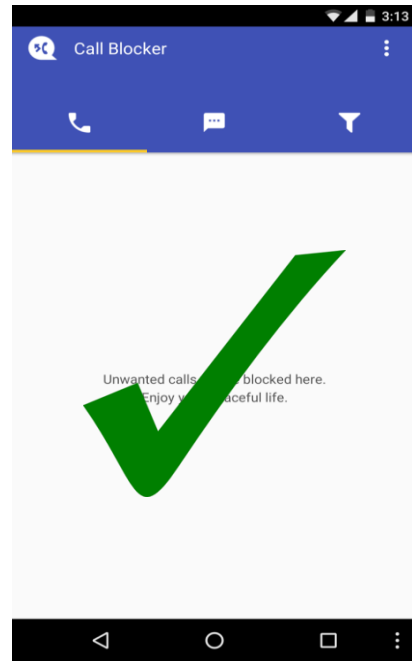
“No matter how useful or how good your app is, if its UI is not awesome it will have very few chances of remaining there in your user’s device.”

Bad UI

=



BAD UI



**TOTAL NUMBER OF ANDROID
APPLICATIONS IN PLAY STORE IS**

> 20,00,000

made on imgur

BUT NOT TO WORRY...

800,000 apps have < 100 downloads.

Another 700,000 have < 1000 downloads.

Another 400,000 have < 10,000 downloads.

Only 35,000 apps are downloaded more than 500,000.

UNIT – V

DATABASE

UNDERSTANDING OF SQLite DATABASE

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

Database -Creation

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	<p>openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler)</p> <p>This method only opens the existing database with the appropriate flag mode. The common flags mode could be OPEN_READWRITE OPEN_READONLY</p>
2	<p>openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)</p> <p>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</p>
3	<p>openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)</p> <p>It not only opens but create the database if it not exists. This method is equivalent to openDatabase method.</p>
4	<p>openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)</p> <p>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()</p>

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 TutorialPoint(Username VARCHAR,Password VARCHAR);");  
mydatabase.execSQL("INSERT INTO TutorialPoint 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	<code>execSQL(String sql, Object[] bindArgs)</code> This method not only insert data , but also used to update or modify already existing data in database using bind arguments

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 = mydatabase.rawQuery("Select * from MRCET",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	<code>getColumnCount()</code> This method return the total number of columns of the table.
2	<code>getColumnIndex(String columnName)</code> This method returns the index number of a column by specifying the name of the column
3	<code>getColumnName(int columnIndex)</code> This method returns the name of the column by specifying the index of the column
4	<code>getColumnNames()</code> 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

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) {}
}
```

Example

Here is an example demonstrating the use of SQLite Database. It creates a basic contacts applications that allows insertion, deletion and modification of contacts.

To experiment with this example, you need to run this on an actual device on which camera is supported.

Steps	Description
1	You will use Android studio to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to get references of all the XML components and populate the contacts on listView.
3	Create new src/DBHelper.java that will manage the database work
4	Create a new Activity as DisplayContact.java that will display the contact on the screen

5	Modify the res/layout/activity_main to add respective XML components
6	Modify the res/layout/activity_display_contact.xml to add respective XML components
7	Modify the res/values/string.xml to add necessary string components
8	Modify the res/menu/display_contact.xml to add necessary menu components
9	Create a new menu as res/menu/mainmenu.xml to add the insert contact option
10	Run the application and choose a running android device and install the application on it and verify the results.

Following is the content of the modified **MainActivity.java**.

```

package com.example.sairamkrishna.myapplication;

import android.content.Context;
import android.content.Intent;
import android.support.v7.app.ActionBarActivity;
import android.os.Bundle;

import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;

import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;

import java.util.ArrayList;
import java.util.List;

```

```

public class MainActivity extends ActionBarActivity {
    public final static String EXTRA_MESSAGE = "MESSAGE";

    private ListView obj;

    DBHelper mydb;

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

        mydb = new DBHelper(this);

        ArrayList array_list = mydb.getAllCotacts();

        ArrayAdapter arrayAdapter=new ArrayAdapter(this,android.R.layout.simple_list_item_1, array_list);

        obj = (ListView)findViewById(R.id.listView1);
        obj.setAdapter(arrayAdapter);
        obj.setOnItemClickListener(new OnItemClickListener(){

            @Override
            public void onItemClick(AdapterView<?> arg0, View arg1, int arg2,long arg3) {

                // TODO Auto-generated method stub
                int id_To_Search = arg2 + 1;

                Bundle dataBundle = new Bundle();
                dataBundle.putInt("id", id_To_Search);

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

                intent.putExtras(dataBundle);
                startActivity(intent);
            }
        });
    }
}

```

```
@Override
```

```
public boolean onCreateOptionsMenu(Menu menu) {  
    // Inflate the menu; this adds items to the action bar if it is present.  
    getMenuInflater().inflate(R.menu.menu_main, menu);  
    return true;  
}
```

```
@Override
```

```
public boolean onOptionsItemSelected(MenuItem item){  
    super.onOptionsItemSelected(item);  
  
    switch(item.getItemId()) {  
        case R.id.item1:Bundle dataBundle = new Bundle();  
        dataBundle.putInt("id", 0);  
  
        Intent intent = new Intent(getApplicationContext(),DisplayContact.class);  
        intent.putExtras(dataBundle);  
  
        startActivity(intent);  
        return true;  
        default:  
        return super.onOptionsItemSelected(item);  
    }  
}  
  
public boolean onKeyDown(int keycode, KeyEvent event) {  
    if (keycode == KeyEvent.KEYCODE_BACK) {  
        moveTaskToBack(true);  
    }  
    return super.onKeyDown(keycode, event);  
}  
}
```

Following is the modified content of display contact activity **DisplayContact.java**

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

import android.os.Bundle;
import android.app.Activity;
import android.app.AlertDialog;

import android.content.DialogInterface;
import android.content.Intent;
import android.database.Cursor;

import android.view.Menu;
import android.view.MenuItem;
import android.view.View;

import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class DisplayContact extends Activity {
    int from_Where_I_Am_Coming = 0;
    private DBHelper mydb ;

    TextView name ;
    TextView phone;
    TextView email;
    TextView street;
    TextView place;

    int id_To_Update = 0;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```

setContentView(R.layout.activity_display_contact);

name = (TextView) findViewById(R.id.editTextName);
phone = (TextView) findViewById(R.id.editTextPhone);
email = (TextView) findViewById(R.id.editTextStreet);
street = (TextView) findViewById(R.id.editTextEmail);
place = (TextView) findViewById(R.id.editTextCity);

mydb = new DBHelper(this);

Bundle extras = getIntent().getExtras();
if(extras !=null) {
    int Value = extras.getInt("id");

    if(Value>0){
        //means this is the view part not the add contact part.
        Cursor rs = mydb.getData(Value);
        id_To_Update = Value;
        rs.moveToFirst();

        String nam = rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_NAME));
        String phon = rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_PHONE));
        String emai = rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_EMAIL));
        String stree = rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_STREET));
        String plac = rs.getString(rs.getColumnIndex(DBHelper.CONTACTS_COLUMN_CITY));

        if (!rs.isClosed()) {
            rs.close();
        }
        Button b = (Button)findViewById(R.id.button1);
        b.setVisibility(View.INVISIBLE);

        name.setText((CharSequence)nam);
        name.setFocusable(false);
    }
}

```

```

name.setClickable(false);

phone.setText((CharSequence)phon);
phone.setFocusable(false);
phone.setClickable(false);

email.setText((CharSequence)emai);
email.setFocusable(false);
email.setClickable(false);

street.setText((CharSequence)stree);
street.setFocusable(false);
street.setClickable(false);

place.setText((CharSequence)plac);
place.setFocusable(false);
place.setClickable(false);
}
}
}

```

@Override

```

public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    Bundle extras = getIntent().getExtras();

    if(extras !=null) {
        int Value = extras.getInt("id");
        if(Value>0){
            getMenuInflater().inflate(R.menu.display_contact, menu);
        } else{
            getMenuInflater().inflate(R.menu.menu_main menu);
        }
    }
}

```

```

    }

    return true;
}

public boolean onOptionsItemSelected(MenuItem item) {
    super.onOptionsItemSelected(item);

    switch(item.getItemId()) {

        case R.id.Edit_Contact:
            Button b = (Button)findViewById(R.id.button1);

            b.setVisibility(View.VISIBLE);

            name.setEnabled(true);

            name.setFocusableInTouchMode(true);

            name.setClickable(true);

            phone.setEnabled(true);

            phone.setFocusableInTouchMode(true);

            phone.setClickable(true);

            email.setEnabled(true);

            email.setFocusableInTouchMode(true);

            email.setClickable(true);

            street.setEnabled(true);

            street.setFocusableInTouchMode(true);

            street.setClickable(true);

            place.setEnabled(true);

            place.setFocusableInTouchMode(true);

            place.setClickable(true);

            return true;

        case R.id.Delete_Contact:

```



```

AlertDialog.Builder builder = new AlertDialog.Builder(this);

builder.setMessage(R.string.deleteContact)
    .setPositiveButton(R.string.yes, new DialogInterface.OnClickListener() {

        public void onClick(DialogInterface dialog, int id) {

            mydb.deleteContact(id_To_Update);

            Toast.makeText(getApplicationContext(), "Deleted Successfully",
                Toast.LENGTH_SHORT).show();

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

            startActivity(intent);

        }

    })
    .setNegativeButton(R.string.no, new DialogInterface.OnClickListener() {

        public void onClick(DialogInterface dialog, int id) {

            // User cancelled the dialog

        }

    });

```

```

AlertDialog d = builder.create();
d.setTitle("Are you sure");
d.show();

```

```

return true;

default:

return super.onOptionsItemSelected(item);

}

}

```

```

public void run(View view) {

    Bundle extras = getIntent().getExtras();

    if(extras !=null) {

        int Value = extras.getInt("id");

        if(Value>0){

```

```
if(mydb.updateContact(id_To_Update,name.getText().toString(),
    phone.getText().toString(), email.getText().toString(),
        street.getText().toString(), place.getText().toString())){
    Toast.makeText(getApplicationContext(), "Updated", Toast.LENGTH_SHORT).show();
    Intent intent = new Intent(getApplicationContext(),MainActivity.class);
    startActivity(intent);
} else{
    Toast.makeText(getApplicationContext(), "not Updated", Toast.LENGTH_SHORT).show();
}
} else{
    if(mydb.insertContact(name.getText().toString(), phone.getText().toString(),
        email.getText().toString(), street.getText().toString(),
            place.getText().toString())){
        Toast.makeText(getApplicationContext(), "done",
            Toast.LENGTH_SHORT).show();
    } else{
        Toast.makeText(getApplicationContext(), "not done",
            Toast.LENGTH_SHORT).show();
    }
    Intent intent = new Intent(getApplicationContext(),MainActivity.class);
    startActivity(intent);
}
}
}
}
```

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;

}

}

```

Following is the content of the **res/layout/activity_main.xml**

```

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

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

    android:layout_height="match_parent"

```

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

```
<TextView
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/textView"
    android:layout_alignParentTop="true"
    android:layout_centerHorizontal="true"
    android:textSize="30dp"
    android:text="Data Base" />
```

```
<TextView
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Tutorials Point"
    android:id="@+id/textView2"
    android:layout_below="@+id/textView"
    android:layout_centerHorizontal="true"
    android:textSize="35dp"
    android:textColor="#ff16ff01" />
```

```
<ImageView
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:id="@+id/imageView"
    android:layout_below="@+id/textView2"
    android:layout_centerHorizontal="true"
    android:src="@drawable/logo" />
```

```
<ScrollView
```

```
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:id="@+id/scrollView"
android:layout_below="@+id/imageView"
android:layout_alignParentLeft="true"
android:layout_alignParentStart="true"
android:layout_alignParentBottom="true"
android:layout_alignParentRight="true"
android:layout_alignParentEnd="true">
```

```
<ListView
```

```
    android:id="@+id/listView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:layout_centerVertical="true" >
```

```
</ListView>
```

```
</ScrollView>
```

```
</RelativeLayout>
```

Following is the content of the **res/layout/activity_display_contact.xml**

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/scrollView1"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    tools:context=".DisplayContact" >

    <RelativeLayout

        android:layout_width="match_parent"

        android:layout_height="370dp"
```

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

```
<EditText
```

```
    android:id="@+id/editTextName"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignParentLeft="true"  
    android:layout_marginTop="5dp"  
    android:layout_marginLeft="82dp"  
    android:ems="10"  
    android:inputType="text" >
```

```
</EditText>
```

```
<EditText
```

```
    android:id="@+id/editTextEmail"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignLeft="@+id/editTextStreet"  
    android:layout_below="@+id/editTextStreet"  
    android:layout_marginTop="22dp"  
    android:ems="10"  
    android:inputType="textEmailAddress" />
```

```
<TextView
```

```
    android:id="@+id/textView1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignBottom="@+id/editTextName"  
    android:layout_alignParentLeft="true"  
    android:text="@string/name"
```



```
android:textAppearance="?android:attr/textAppearanceMedium" />
```

```
<Button
```

```
    android:id="@+id/button1"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignLeft="@+id/editTextCity"  
    android:layout_alignParentBottom="true"  
    android:layout_marginBottom="28dp"  
    android:onClick="run"  
    android:text="@string/save" />
```

```
<TextView
```

```
    android:id="@+id/textView2"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignBottom="@+id/editTextEmail"  
    android:layout_alignLeft="@+id/textView1"  
    android:text="@string/email"  
    android:textAppearance="?android:attr/textAppearanceMedium" />
```

```
<TextView
```

```
    android:id="@+id/textView5"  
    android:layout_width="wrap_content"  
    android:layout_height="wrap_content"  
    android:layout_alignBottom="@+id/editTextPhone"  
    android:layout_alignLeft="@+id/textView1"  
    android:text="@string/phone"  
    android:textAppearance="?android:attr/textAppearanceMedium" />
```

```
<TextView
```

```
    android:id="@+id/textView4"  
    android:layout_width="wrap_content"
```

```
android:layout_height="wrap_content"
android:layout_above="@+id/editTextEmail"
android:layout_alignLeft="@+id/textView5"
android:text="@string/street"
android:textAppearance="?android:attr/textAppearanceMedium" />
```

<EditText

```
android:id="@+id/editTextCity"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignRight="@+id/editTextName"
android:layout_below="@+id/editTextEmail"
android:layout_marginTop="30dp"
android:ems="10"
android:inputType="text" />
```

<TextView

```
android:id="@+id/textView3"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignBaseline="@+id/editTextCity"
android:layout_alignBottom="@+id/editTextCity"
android:layout_alignParentLeft="true"
android:layout_toLeftOf="@+id/editTextEmail"
android:text="@string/country"
android:textAppearance="?android:attr/textAppearanceMedium" />
```

<EditText

```
android:id="@+id/editTextStreet"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:layout_alignLeft="@+id/editTextName"
android:layout_below="@+id/editTextPhone"
```

```

        android:ems="10"
        android:inputType="text" >
        <requestFocus />
    </EditText>

    <EditText
        android:id="@+id/editTextPhone"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignLeft="@+id/editTextStreet"
        android:layout_below="@+id/editTextName"
        android:ems="10"
        android:inputType="phone|text" />

</RelativeLayout>
</ScrollView>

```

Following is the content of the **res/value/string.xml**

```

<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="app_name">Address Book</string>
    <string name="action_settings">Settings</string>
    <string name="hello_world">Hello world!</string>
    <string name="Add_New">Add New</string>
    <string name="edit">Edit Contact</string>
    <string name="delete">Delete Contact</string>
    <string name="title_activity_display_contact">DisplayContact</string>
    <string name="name">Name</string>
    <string name="phone">Phone</string>
    <string name="email">Email</string>
    <string name="street">Street</string>
    <string name="country">City/State/Zip</string>
    <string name="save">Save Contact</string>
    <string name="deleteContact">Are you sure, you want to delete it.</string>

```

```
<string name="yes">Yes</string>
<string name="no">No</string>
</resources>
```

Following is the content of the **res/menu/main_menu.xml**

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

  <item android:id="@+id/item1"
        android:icon="@drawable/add"
        android:title="@string/Add_New" >

  </item>

</menu>
```

Following is the content of the **res/menu/display_contact.xml**

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

  <item
        android:id="@+id/Edit_Contact"
        android:orderInCategory="100"
        android:title="@string/edit"/>

  <item
        android:id="@+id/Delete_Contact"
        android:orderInCategory="100"
        android:title="@string/delete"/>

</menu>
```

This is the default **AndroidManifest.xml** of this project

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

```

<application
    android:allowBackup="true"
    android:icon="@mipmap/ic_launcher"
    android:label="@string/app_name"
    android:theme="@style/AppTheme" >

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

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

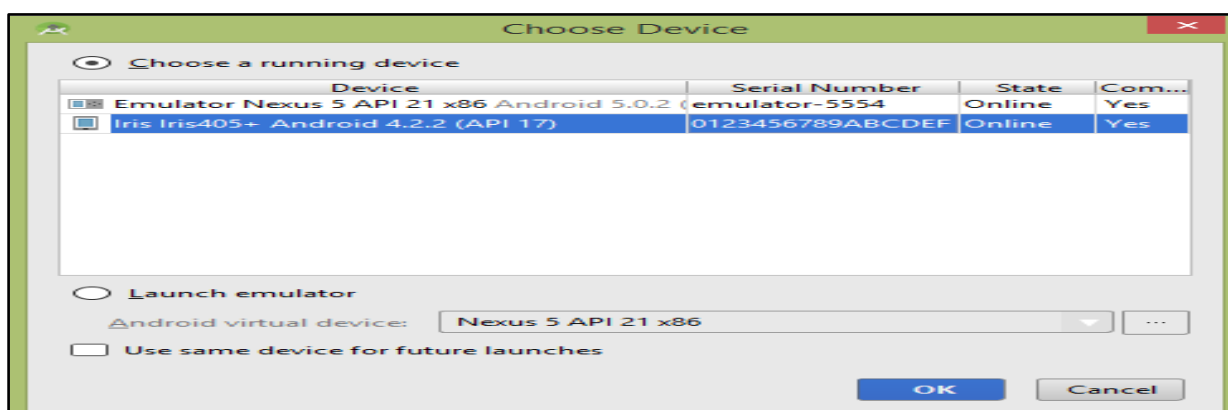
    </activity>

    <activity android:name=".DisplayContact"/>

</application>
</manifest>

```

Let's try to run your application. I assume you have connected your actual Android Mobile device with your computer. To run the app from Android studio , open one of your project's activity files and click Run icon from the tool bar. Before starting your application,Android studio will display following window to select an option where you want to run your Android application.



Select your mobile device as an option and then check your mobile device which will display following screen

Optional menu appears different places on different versions

Click on the add button of the menu screen to add a new contact. It will display the following screen –

It will display the following fields. Please enter the required information and click on save contact. It will bring you back to main screen.

Now our contact has been added. In order to see where the database is created, open your android studio, connect your mobile. Go **tools/android/android device monitor**. Now browse the file explorer tab. Now browse this folder **/data/data/<your.package.name>/databases<database-name>**.

CONNECTING WITH THE DATABASE

In Android, the SQLiteDatabase namespace defines the functionality to connect and manage a database. It provides functionality to create, delete, manage and display database content.

Create a Database

Simple steps to create a database and handle are as following.

- Create "SQLiteDatabase" object.
- Open or Create database and create connection.
- Perform insert, update or delete operation.
- Create Cursor to display data from table of database.
- Close the database connectivity.

Step 1: Instantiate "SQLiteDatabase" object

SQLiteDatabase db;

Before you can use the above object, you must import the android.database.sqlite.SQLiteDatabase namespace in your application.

```
db=openOrCreateDatabase(String path, int mode, SQLiteDatabase.CursorFactory factory)
```

This method is used to create/open database. As the name suggests, it will open a database connection if it is already there, otherwise it will create a new one.

Example,

```
db=openOrCreateDatabase("XYZ_Database",SQLiteDatabase.CREATE_IF_NECESSARY,null);
```

Arguments:

String path	Name of the database
Int mode	operating mode. Use 0 or "MODE_PRIVATE" for the default operation, or "CREATE_IF_NECESSARY" if you like to give option that "if database is not there, create it"
CursorFactory factory	An optional factory class that is called to instantiate a cursor when query is called

Step 2: Execute DDL command

db.execSQL(String sql) throws SQLException

This command is used to execute single SQL statement which doesn't return any data means other than SELECT or any other.

```
db.execSQL("Create Table Temp (id Integer, name Text)");
```

In the above example, it takes "CREATE TABLE" statement of SQL. This will create a table of "Integer" & "Text" fields.

Try and Catch block is require while performing this operation. An exception that indicates there was an error with SQL parsing or execution.

Step 3: Create object of "ContentValues" and Initiate it.

```
ContentValues values=new ContentValues();
```

This class is used to store a set of values. We can also say, it will map ColumnName and relavent ColumnValue.

```
values.put("id", eid.getText().toString());  
values.put("name", ename.getText().toString());
```

String Key	Name of field as in table. Ex. "id", "name"
String Value	Value to be inserted.

Step 4: Perform Insert Statement.

```
insert(String table, String nullColumnHack, ContentValues values)
```

String table	Name of table related to database.
String nullColumnHack	If not set to null, the nullColumnHack parameter provides the name of nullable column name to explicitly insert a NULL into in the case where your <i>values</i> is empty.
ContentValues values	This map contains the initial column values for the row.

This method returns a long. The row ID of the newly inserted row, or -1 if an error occurred.

Example,

```
db.insert("temp", null, values);
```

Step 5: Create Cursor

This interface provides random read-write access to the result set returned by a database query.

```
Cursor c=db.rawQuery(String sql, String[] selectionArgs)
```

Strign sql	The SQL query
String []selectionArgs	You may include ?s in where clause in the query, which will be replaced by the values from selectionArgs. The values will be bound as Strings.

Example,

```
Cursor c=db.rawQuery("SELECT * FROM temp",null);
```

Methods

moveToFirst	Moves cursor pointer at first position of result set
moveToNext	Moves cursor pointer next to current position.
isAfterLast	Returns false, if cursor pointer is not at last position of result set.

Example,

```
c.moveToFirst();
while(!c.isAfterLast())
{
    //statementâ€¦
    c.moveToNext();
}
```

Step 6: Close Cursor and Close Database connectivity

It is very important to release our connections before closing our activity. It is advisable to release the Database connectivity in "onStop" method. And Cursor connectivity after use it.

DatabaseDemoActivity.java

```
package com.DataBaseDemo;
```

```
import android.app.Activity;
import android.content.ContentValues;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
```

```
public class DataBaseDemoActivity extends Activity {
    /** Called when the activity is first created. */
    SQLiteDatabase db;
    Button btnInsert;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        btnInsert=(Button)findViewById(R.id.button1);
        try{
            db=openOrCreateDatabase("StudentDB",SQLiteDatabase.CREATE_IF_NECESSARY,null);
            db.execSQL("Create Table Temp(id integer,name text)");
        }catch(SQLException e)
        {
        }
        btnInsert.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                EditText eid=(EditText) findViewById(R.id.editText1);
                EditText ename=(EditText)findViewById(R.id.editText2);
                ContentValues values=new ContentValues();
                values.put("id", eid.getText().toString());
                values.put("name", ename.getText().toString());
                if((db.insert("temp", null, values))!=-1)
                {
                    Toast.makeText(DataBaseDemoActivity.this, "Record Successfully Inserted", 2000).show();
                }
            }
        });
    }
}
```



```

    {
    Toast.makeText(DataBaseDemoActivity.this, "Insert Error", 2000).show();
    }
    eid.setText("");
    ename.setText("");

    Cursor c=db.rawQuery("SELECT * FROM temp",null);
    c.moveToFirst();
    while(!c.isAfterLast())
    {
    Toast.makeText(DataBaseDemoActivity.this,c.getString(0)+ " "+c.getString(1),1000).show();
    c.moveToNext();
    }
    c.close();
    }
    });
}
@Override
protected void onStop() {
// TODO Auto-generated method stub
db.close();
super.onStop();
}
}

```



To see where your database is stored, (1) *Start Your Emulator* (It is necessary to start Emulator to see File Explorer content and (2) *Open "File Explorer"*

Data -> Data -> find your "package" -> databases -> "database"

