ANDROID DEVELOPMENT

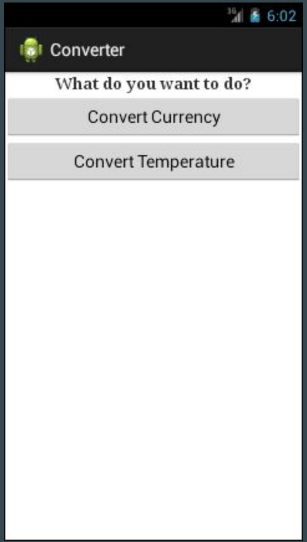
•••

Advanced Terminologies in Android Programming

- Activities

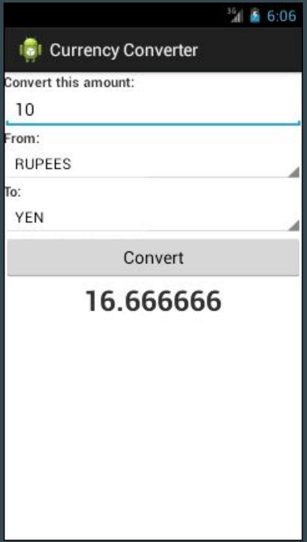
Activities provide a distinct visual UI designed for a single, well-defined purpose.

It is something visible to user. Each Activity has one UI. In Short, Android calls activity and it in turns calls an UI.


6:02

Converter

What do you want to do?


6:06

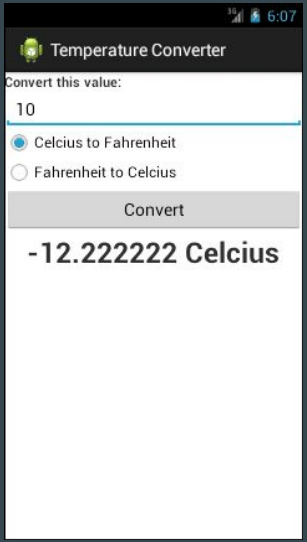
Currency Converter

Convert this amount:
10

From:
RUPEES

To:
YEN

16.666666


6:07

Temperature Converter

Convert this value:
10

(● Celcius to Fahrenheit) ○ Fahrenheit to Celcius

-12.222222 Celcius

Advanced Terminologies in Android Programming

- Services

Not visible to user. Runs on background. A service is an application component like an activity but without a user interface. They have a simpler life cycle than activities.

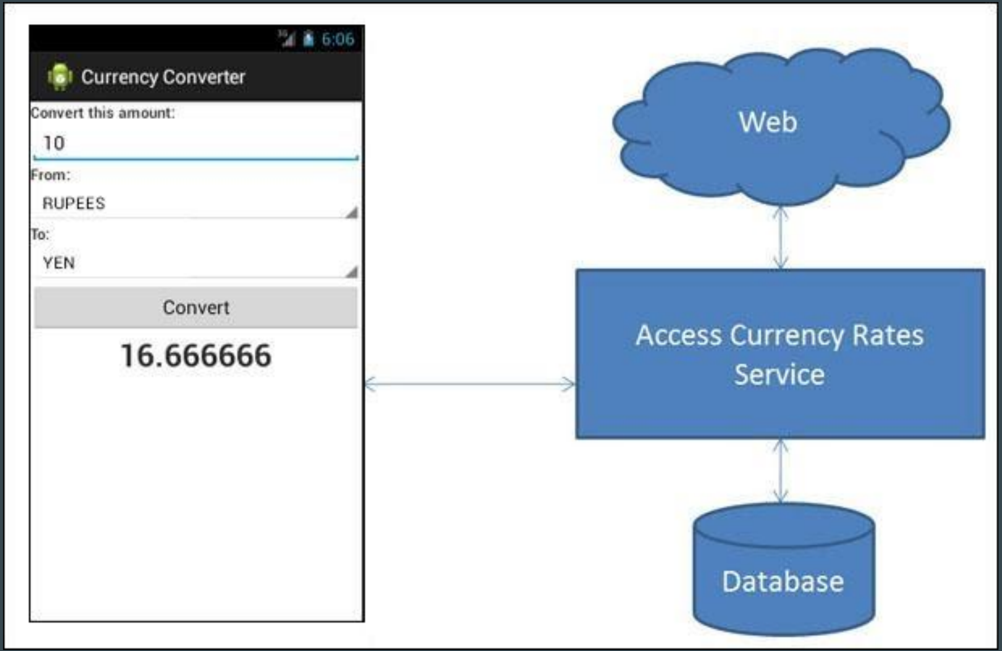
The bulk of your app’s code is there to interact with the user, but sometimes you need to do things in the background, such as download a large file, stream a piece of music, or listen for a message from the server.

In addition to writing your own services, you can use Android’s built-in ones.

Built-in services include the notification service, location service, alarm service, and download service.

Advanced Terminologies in Android Programming

- Services



Advanced Terminologies in Android Programming

- Content Providers

Helps data of other Applications to interact with our own application’s data.

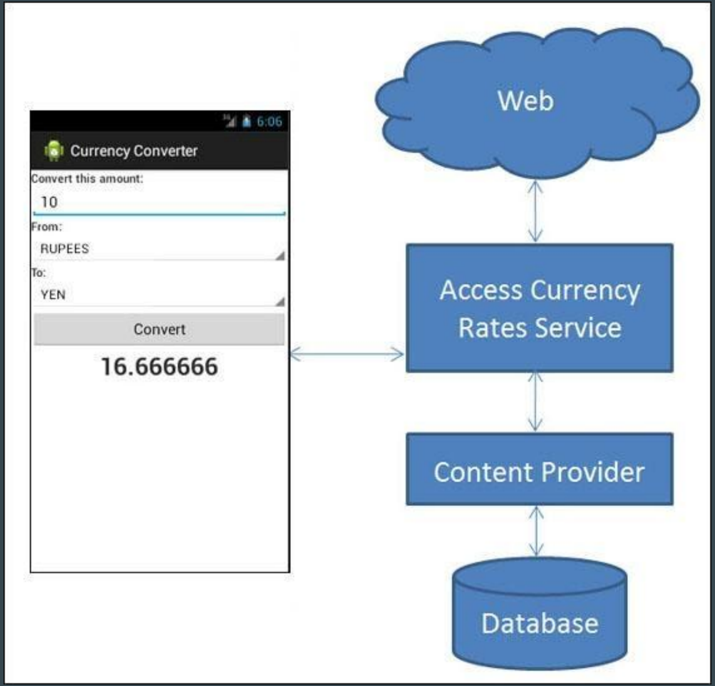
Content providers help you to store and retrieve data, and make the data accessible to all apps.

But what if you want to use another app’s data in your own app?

You can’t access another app’s data by interrogating its database, Instead, you use a content provider, which is an interface that allows apps to share data in a controlled way. It allows you to perform queries to read the data, insert new records, and update or delete existing records.

Advanced Terminologies in Android Programming

- Content Providers



Advanced Terminologies in Android Programming

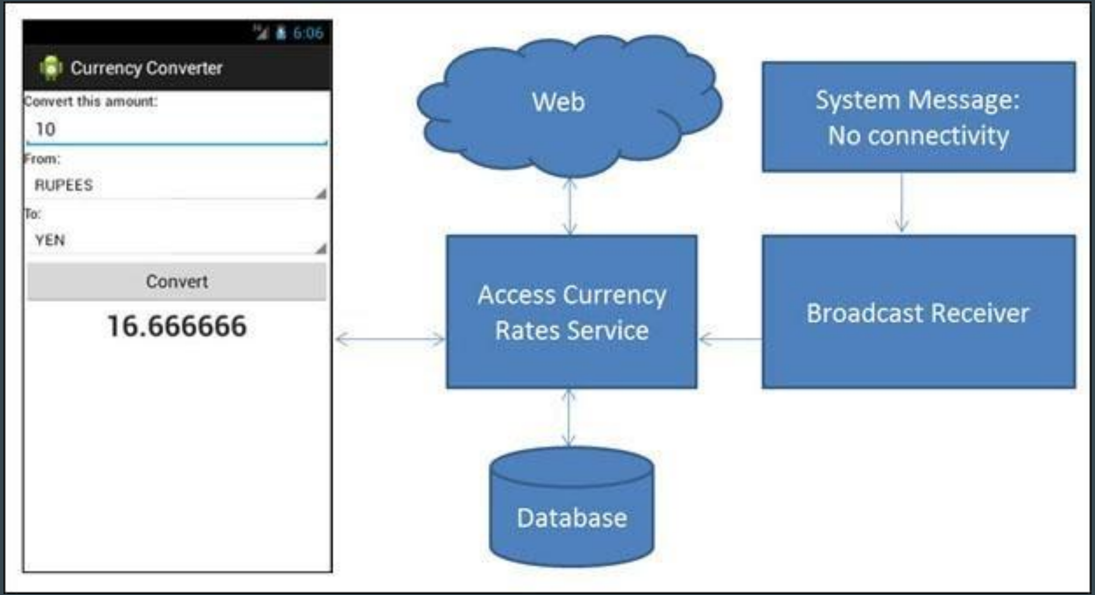
- Broadcast Receiver

Is a component of an app that responds to system-wide broadcast announcements. Does not provide a UI. However, it can create a status bar notification to alert the user when a broadcast event occurs.

Suppose you want your app to react in some way when a system event occurs. You may, for example, have built a music app, and you want it to stop playing music if the headphones are removed. How can your app tell when these events occur?

Advanced Terminologies in Android Programming

- Broadcast Receiver



Advanced Terminologies in Android Programming

- Quick Revision:

Which of the following building blocks of an Android app represents an application component that keeps running in the background without requiring any user intervention?

  - Activity
  - Service
  - Content provider
  - Broadcast receiver

Advanced Terminologies in Android Programming

- View

Anything that you see is a view. A view is the UI element such as button, label, text field etc.

Advanced Terminologies in Android Programming

- Manifest File

Applications can request services from the device’s built in components such as camera and networking components , but the request to this services are added to manifest file at application design and development time.

This file can be viewed as the “instruction book” the target device uses to run the application.

It contains things like permissions to use features on the device such as the GPS system, references to the files that should be included when the application is bundled up for deployment (hence the inclusion of the word manifest in the filename), version and revision numbers, API information, and so on.

Advanced Terminologies in Android Programming

- Manifest File

All activities need to be declared in AndroidManifest.xml. If an activity isn’t declared in the file, the system won’t know it exists.

And if the system doesn’t know it exists, the activity will never run.

The following line is mandatory and is used to specify the class name of the activity, in this example "MyActivityClassName":

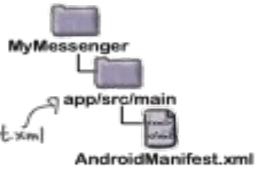
android:name=".MyActivityClassName"

☑ Create 1st activity
☑ Create 2nd activity
☐ Call 2nd activity
☐ Pass data

Welcome to the Android manifest file

Every Android app must include a file called *AndroidManifest.xml*. You can find it in the *app/src/main* folder of your project. The *AndroidManifest.xml* file contains essential information about your app, such as what activities it contains, required libraries, and other declarations. Android creates the file for you when you create the app. If you think back to the settings you chose when you created the project, some of the file contents should look familiar.

Here's what our copy of *AndroidManifest.xml* looks like:



You can find AndroidManifest.xml in this folder.

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

This is the package name we specified.

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

Android Studio gave our app default icons.

The theme affects the appearance of the app. We'll look at this later.

Watch it!

If you develop Android apps without an IDE, you'll need to create this file manually.

This is the first activity, Create Message Activity.

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

This bit specifies that it's the main activity of the app.

This says the activity can be used to launch the app.

```
        <activity android:name=".ReceiveMessageActivity"></activity>

    </application>

</manifest>
```

This is the second activity, ReceiveMessageActivity. Android Studio added this code when we added the second activity.

Advanced Terminologies in Android Programming

- Manifest File:

• Package is the Unique Identifier for any Android Application.

• xlmns stands for XML Namespace

• <application>: It's the starting tag for any application.

• There is only one launcher in manifest file under activity tag and that launcher looks for the main activity and starts the Android Application with the main activity. Main activity in turns calls it's UI.

• Mipmap folder is a resource and it's under a res folder. ic_launcher is the icon.

Intents

Intent is used to invoke components. An Android Intent is an abstract description of an operation to be performed. It is mainly used to:

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

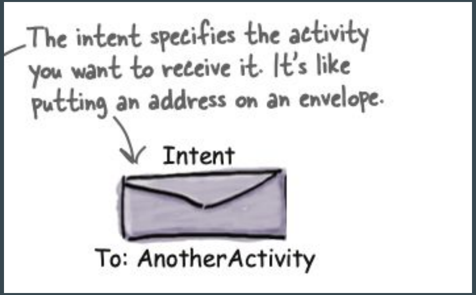
Intents

Whenever you want an activity to start a second activity, you use an intent.

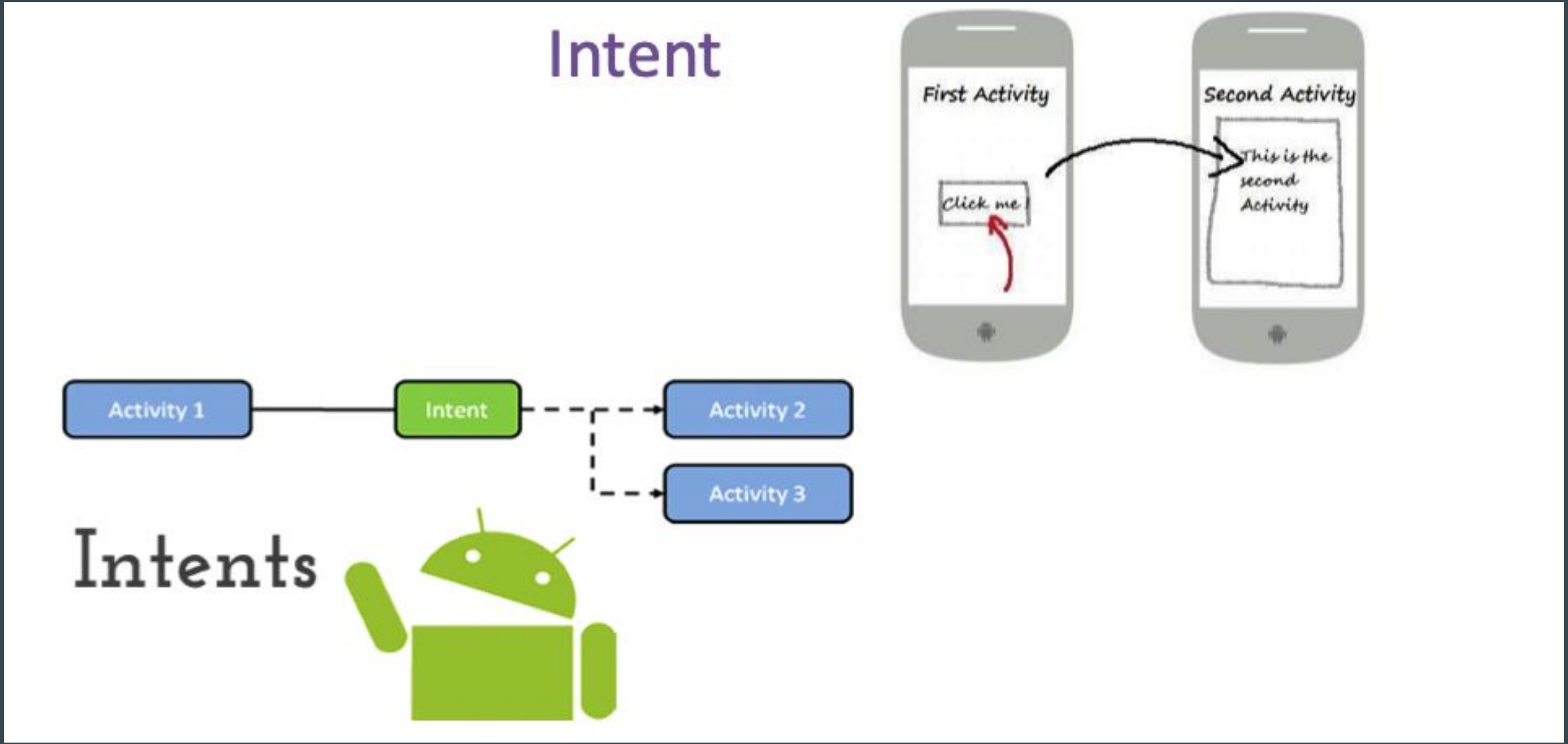
You can think of an intent as an “intent to do something.”

It’s a type of message that allows you to bind separate objects (such as activities) together at runtime.

If one activity wants to start a second activity, it does it by sending an intent to Android. Android will then start the second activity and pass it the intent.


The intent specifies the activity you want to receive it. It’s like putting an address on an envelope.
Intent
To: AnotherActivity

Intent


Intent
First Activity
Click me!
Second Activity
This is the second Activity
Activity 1
Intent
Activity 2
Activity 3
Intents

Intents

In Activity 1:

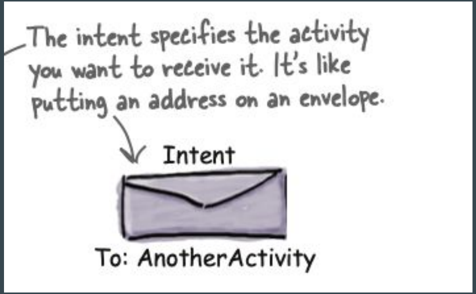
You start by creating the intent like this:

```
Intent intent = new Intent(this, Target.class);
```

The first parameter tells Android which object the intent is from: you can use the word this to refer to the current activity. The second parameter is the class name of the activity that needs to receive the intent.

Once you've created the intent, you pass it to Android like this:

```
startActivity(intent);
```


The intent specifies the activity you want to receive it. It's like putting an address on an envelope.
Intent
To: AnotherActivity

Intents

What all can Intents Launch?

Starting an activity
An Activity represents a single screen in an app. You can start a new instance of an Activity by passing an Intent to startActivity(). The Intent describes the activity to start and carries any necessary data.
If you want to receive a result from the activity when it finishes, call startActivityForResult(). Your activity receives the result as a separate Intent object in your activity's onActivityResult() callback. For more information, see the Activities guide.

Intents

What all can Intents Launch?

Starting a service
A Service is a component that performs operations in the background without a user interface. With Android 5.0 (API level 21) and later, you can start a service with JobScheduler.
You can start a service to perform a one-time operation (such as downloading a file) by passing an Intent to startService(). The Intent describes the service to start and carries any necessary data.
If the service is designed with a client-server interface, you can bind to the service from another component by passing an Intent to bindService().

Intents

What all can Intents Launch?

Delivering a broadcast
A broadcast is a message that any app can receive. The system delivers various broadcasts for system events, such as when the system boots up or the device starts charging. You can deliver a broadcast to other apps by passing an Intent to sendBroadcast() or sendOrderedBroadcast()

Intents

Types of Intents:

1. Explicit Intent
2. Implicit Intent

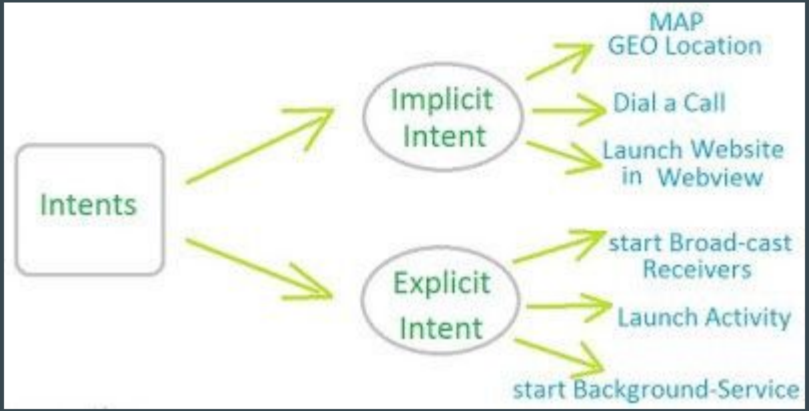
Talking in General Terms:

When talking about writing, **“explicit” means something that is stated plainly,** while **“implicit”** refers to something that is **implied and not stated directly**.

Intents

Types of Intents:

1. Explicit Intent
2. Implicit Intent



Intents

Types of Intents:

1. Explicit Intent
2. Implicit Intent

Explicit Intent: Explicit intents specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name. You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, you might start a new activity within your app in response to a user action, or start a service to download a file in the background.

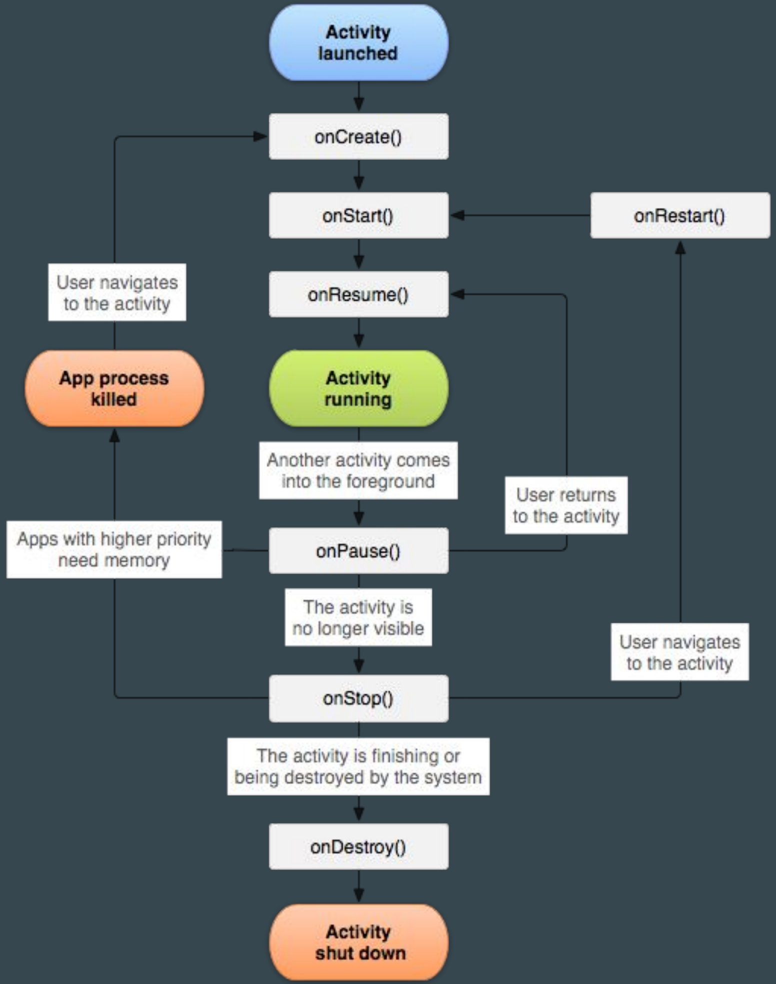
Intents

Types of Intents:

1. Explicit Intent
2. Implicit Intent

Implicit intents do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map

Activity Lifecycle


Activity launched
onCreate()
onStart()
onRestart()
User navigates to the activity
onResume()
App process killed
Activity running
Another activity comes into the foreground
User returns to the activity
Apps with higher priority need memory
onPause()
The activity is no longer visible
User navigates to the activity
onStop()
The activity is finishing or being destroyed by the system
onDestroy()
Activity shut down

Activity Lifecycle

For Details on each of the stage, Refer the below link:

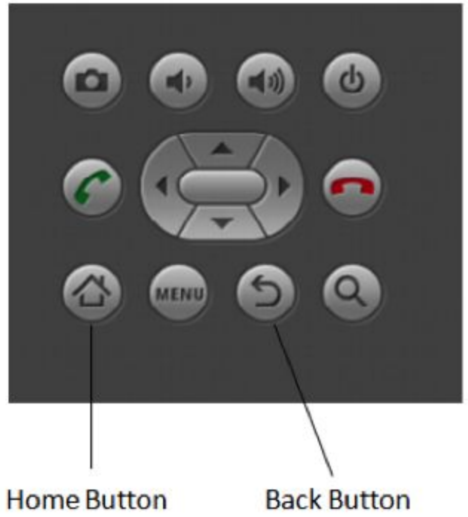
https://stackoverflow.com/questions/8515936/android-activity-life-cycle-what-are-all-these-methods-for

For Official Documentation:

https://developer.android.com/reference/android/app/Activity.html#onStart%28%29

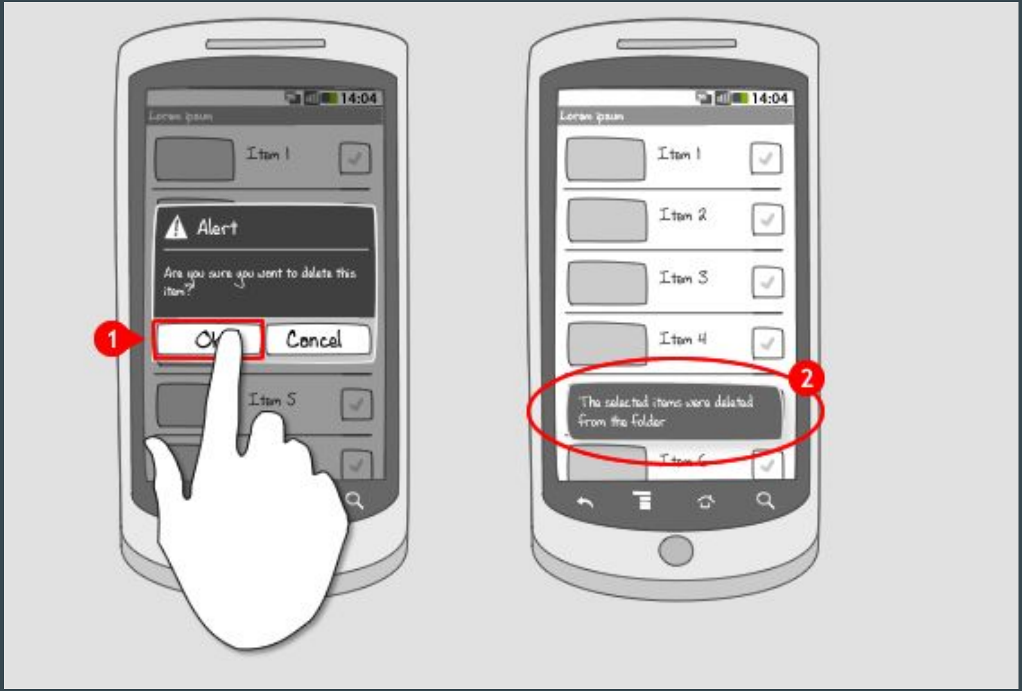
Activity Lifecycle

1. When you run the following code, you see when the application starts there are three toast messages that come after one another. First is **ON CREATE** , second is **ON START** and third is **ON RESUME**.
2. When you press the home button from the emulator, **ON PAUSE** and **ON STOP** methods call.
3. And when you open the application again **ON RESTART**, **ON START** and **ON RESUME** methods call.
4. And at last, when you press the back button from emulator, **ON PAUSE** , **ON STOP** and **ON DESTROY** method calls.



Toasts

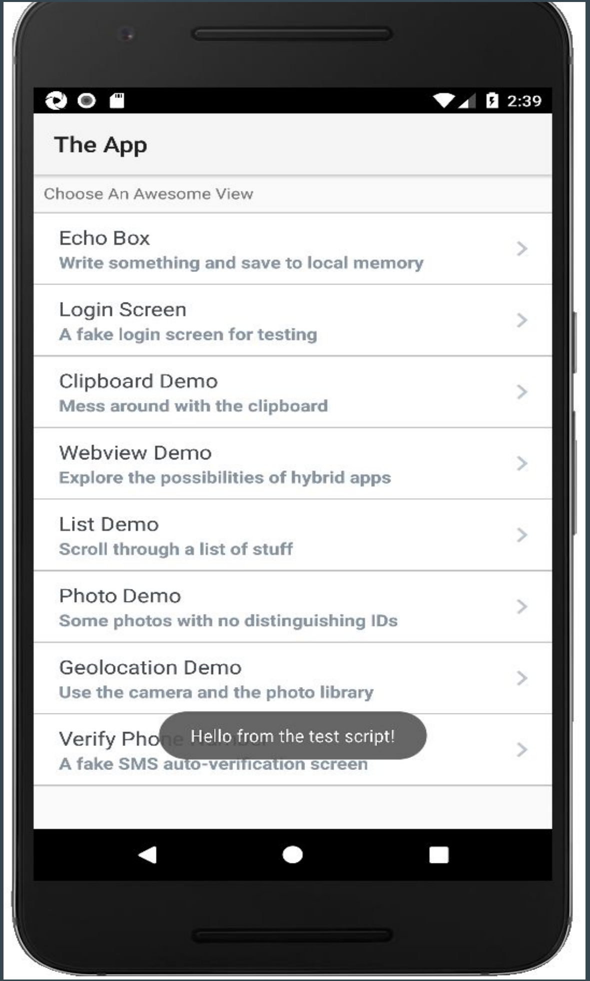
- Android Toast can be used to display information for the short period of time.
- A toast contains message to be displayed quickly and disappears after sometime.
- The android.widget.Toast class is the subclass of java.lang.Object class.


14:04
Lorem ipsum
Item 1
Alert
Are you sure you want to delete this item?
1
OK
Cancel
Item 5
14:04
Lorem ipsum
Item 1
Item 2
Item 3
Item 4
2
The selected items were deleted from the folder

Toasts

Toast toast = Toast.makeText(context, text, duration).show();

Context is Instances of the the class android.content.Context provide the connection to the Android system which executes the application. For example, you can check the size of the current device display via the Context.
Context is an interface to global information about an application environment.



2:39

The App

Choose An Awesome View

Echo Box
Write something and save to local memory

Login Screen
A fake login screen for testing

Clipboard Demo
Mess around with the clipboard

Webview Demo
Explore the possibilities of hybrid apps

List Demo
Scroll through a list of stuff

Photo Demo
Some photos with no distinguishing IDs

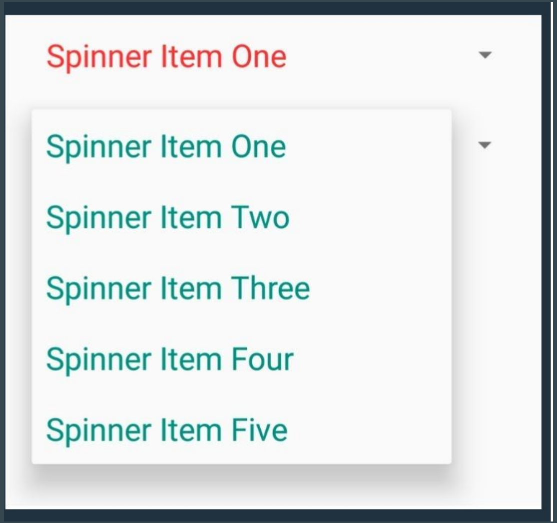
Geolocation Demo
Use the camera and the photo library

Verify Pho
A fake SMS auto-verification screen

Hello from the test script!

Spinner Object

Android Spinner is a view similar to the dropdown list which is used to select one option from the list of options. It provides an easy way to select one item from the list of items and it shows a dropdown list of all values when we click on it. The default value of the android spinner will be the currently selected value and by using Adapter we can easily bind the items to the spinner objects. Generally, we populate our Spinner control with a list of items by using an ArrayAdapter



Event Listeners and Handlers

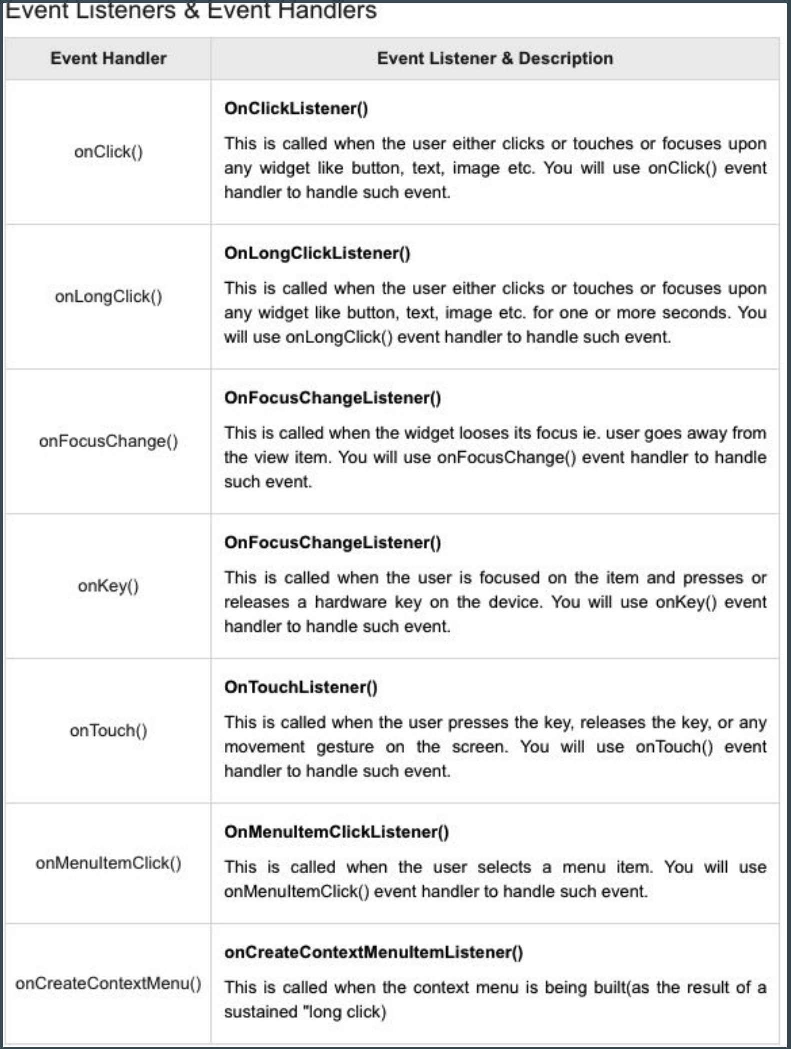
Events are a useful way to collect data about a user's interaction with interactive components of Applications. Like button presses or screen touch etc. The Android framework maintains an event queue as first-in, first-out (FIFO) basis. You can capture these events in your program and take appropriate action as per requirements.

Event Listeners and Handlers

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

Event Listeners Registration – Event Registration is the process by which an Event Handler gets registered with an Event Listener so that the handler is called when the Event Listener fires the event.

Event Handlers – When an event happens and we have registered an event listener for the event, the event listener calls the Event Handlers, which is the method that actually handles the event.



Event Listeners & Event Handlers

Event Handler	Event Listener & Description
onClick()	**OnClickListener()** This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. You will use onClick() event handler to handle such event.
onLongClick()	**OnLongClickListener()** This is called when the user either clicks or touches or focuses upon any widget like button, text, image etc. for one or more seconds. You will use onLongClick() event handler to handle such event.
onFocusChange()	**OnFocusChangeListener()** This is called when the widget looses its focus ie. user goes away from the view item. You will use onFocusChange() event handler to handle such event.
onKey()	**OnFocusChangeListener()** This is called when the user is focused on the item and presses or releases a hardware key on the device. You will use onKey() event handler to handle such event.
onTouch()	**OnTouchListener()** This is called when the user presses the key, releases the key, or any movement gesture on the screen. You will use onTouch() event handler to handle such event.
onMenuItemClick()	**OnMenuItemClickListener()** This is called when the user selects a menu item. You will use onMenuItemClick() event handler to handle such event.
onCreateContextMenu()	**onCreateContextMenuItemListener()** This is called when the context menu is being built(as the result of a sustained "long click)

Event Listeners and Handlers

BONUS QUESTION:

We have used two ways till know to see how we can interact with views:

1. XML File: android:onClick = “Name_of_function”
2. Java File: button.setOnClickListener()

Think about the difference between these two. Why do you think one is inside the onCreate() method and the other is a separate method in itself?

Event Listeners and Handlers

BONUS QUESTION:

We have used if(txt.equals(“India”)) and not txt==”India”

What is the difference between the two in Java?

Try to see if you can replace.equals() with == and it still works or not.

Logging in Android (Additional Information)

Sometimes we need to store/log some information without it being displayed to the user.

Logcat is a command-line tool that dumps a log of system messages, including stack traces when the device throws an error and messages that you have written from your app with the Log class.

Since there is a huge dump, sometimes it might be difficult to track our messages.

Logcat provides the following resources to segregate your logs:

Log.v(String, String) (verbose)

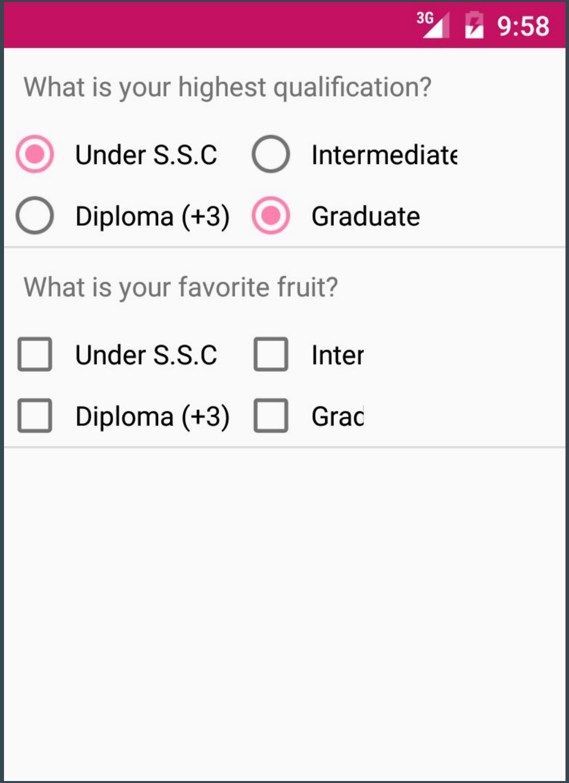
Log.d(String, String) (debug)

Log.i(String, String) (information)

Log.w(String, String) (warning)

Log.e(String, String) (error)

Layouts in Android



What is your highest qualification?

[✓ Under S.S.C] Intermediate

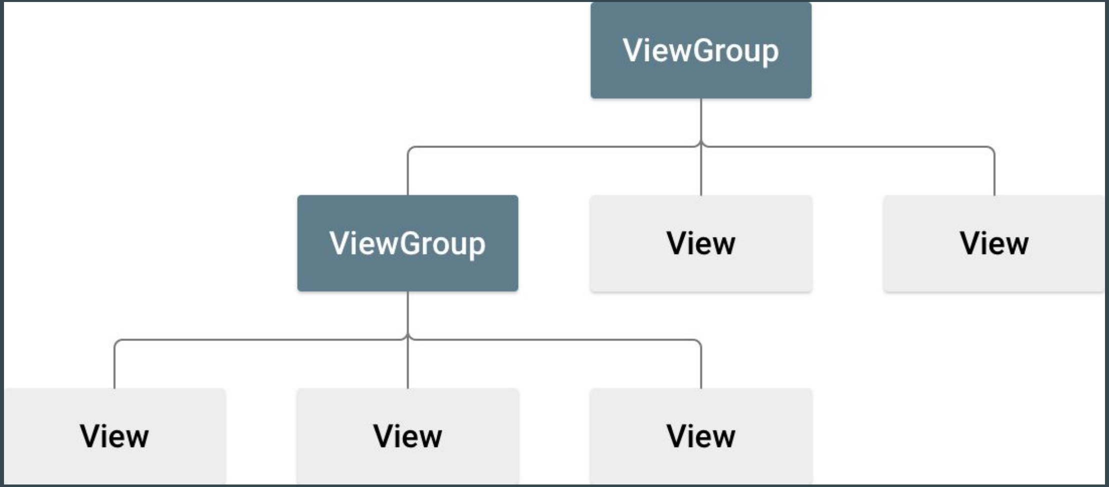
Diploma (+3) [✓ Graduate]

What is your favorite fruit?

Under S.S.C Inter

Diploma (+3) Grad

- Radio Button Vs Checklist?



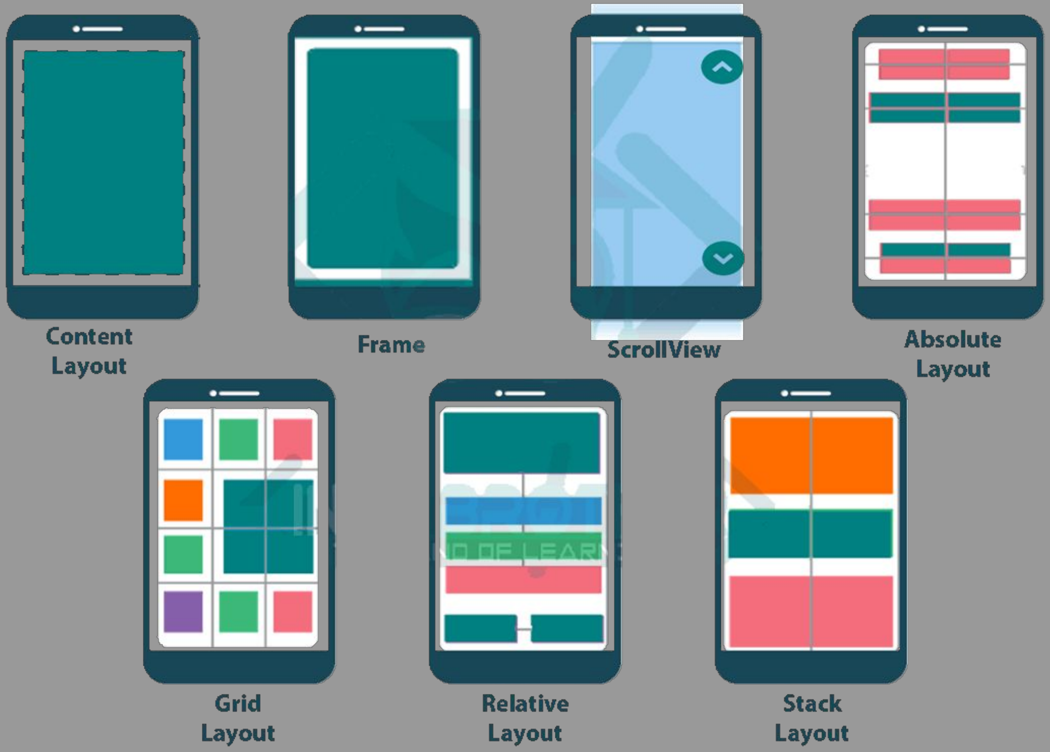
Layouts in Android

You can declare a layout in two ways:

Declare UI elements in XML. Android provides a straightforward XML vocabulary that corresponds to the View classes and subclasses, such as those for widgets and layouts.
You can also use Android Studio's Layout Editor to build your XML layout using a drag-and-drop interface.

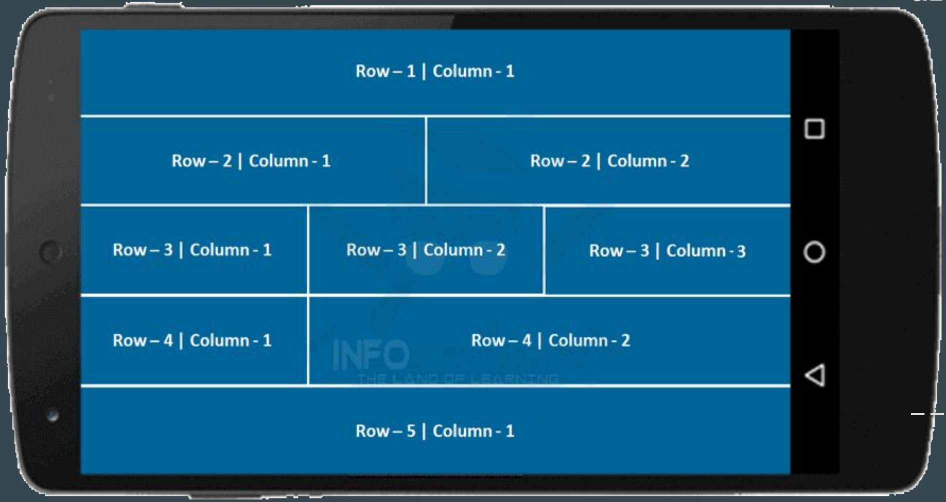
Instantiate layout elements at runtime. Your app can create View and ViewGroup objects (and manipulate their properties) programmatically.

Layouts in Android



Layouts in Android

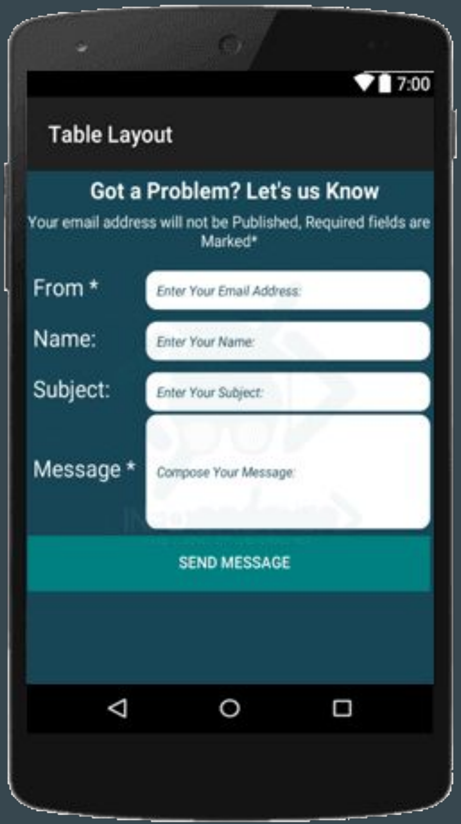
Table Layout



```
<TableLayout
xmlns:android="http://schemas.android.com/apk/res/an
droid"
    android:layout_width="match_parent"
    android:layout_height="match_parent">

    <!-- first row of the table layout-->
    <TableRow
        android:id="@+id/row1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <!-- Add elements/columns in the first row
-->

    </TableRow>


  </TableLayout>
```

Layouts in Android

`Table Layout`



7:00

Table Layout

Got a Problem? Let's us Know

Your email address will not be Published, Required fields are Marked*

From *	Enter Your Email Address:
Name:	Enter Your Name:
Subject:	Enter Your Subject:
Message *	Compose Your Message:

SEND MESSAGE

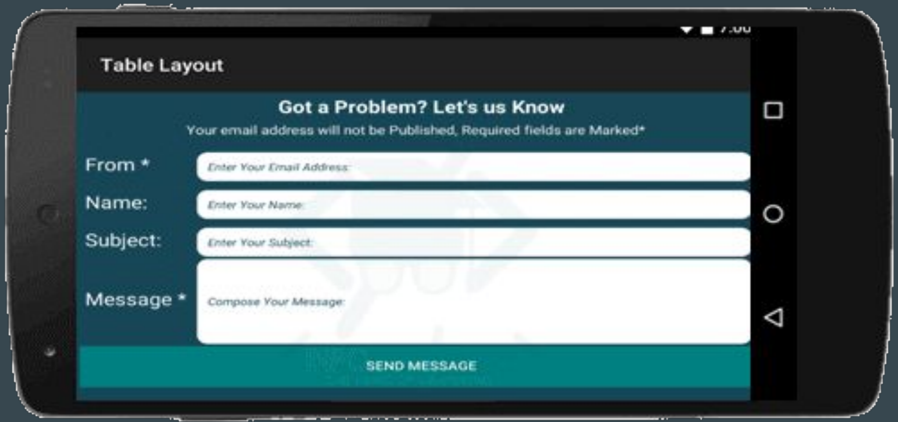


Table Layout

Got a Problem? Let's us Know

Your email address will not be Published, Required fields are Marked*

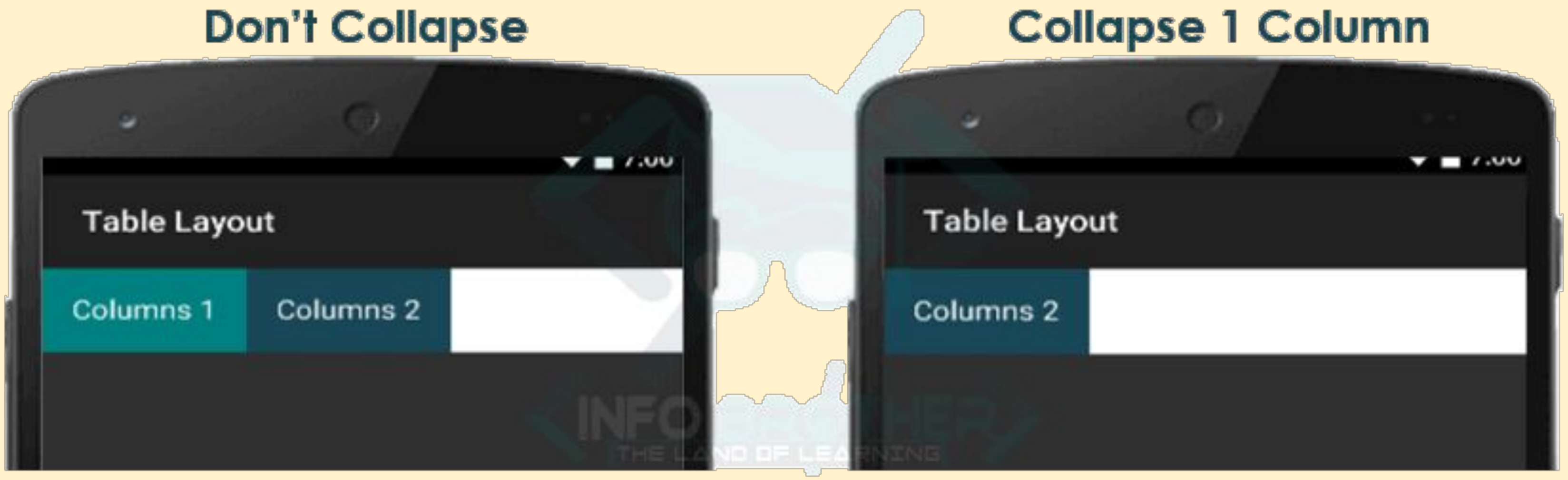
From *	Enter Your Email Address:
Name:	Enter Your Name:
Subject:	Enter Your Subject:
Message *	Compose Your Message:

SEND MESSAGE

Layouts in Android

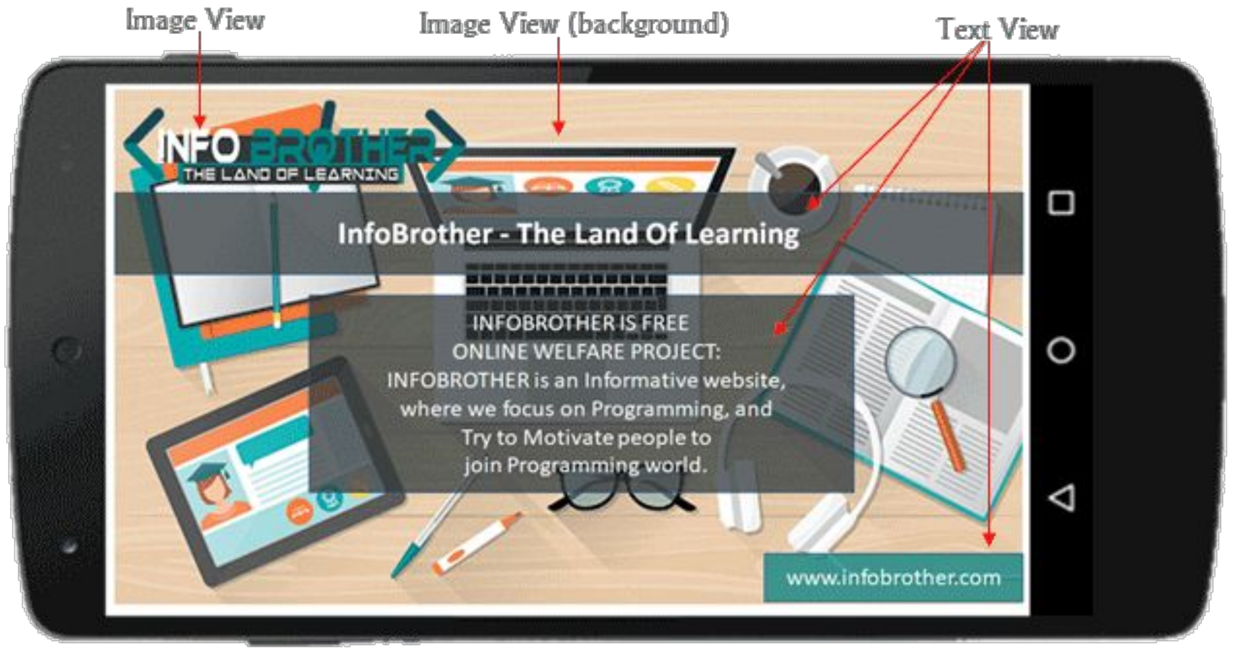
Table Layout

1. Collapse Columns
2. Stretch Columns
3. Shrink Columns



Layouts in Android

`Frame Layout`

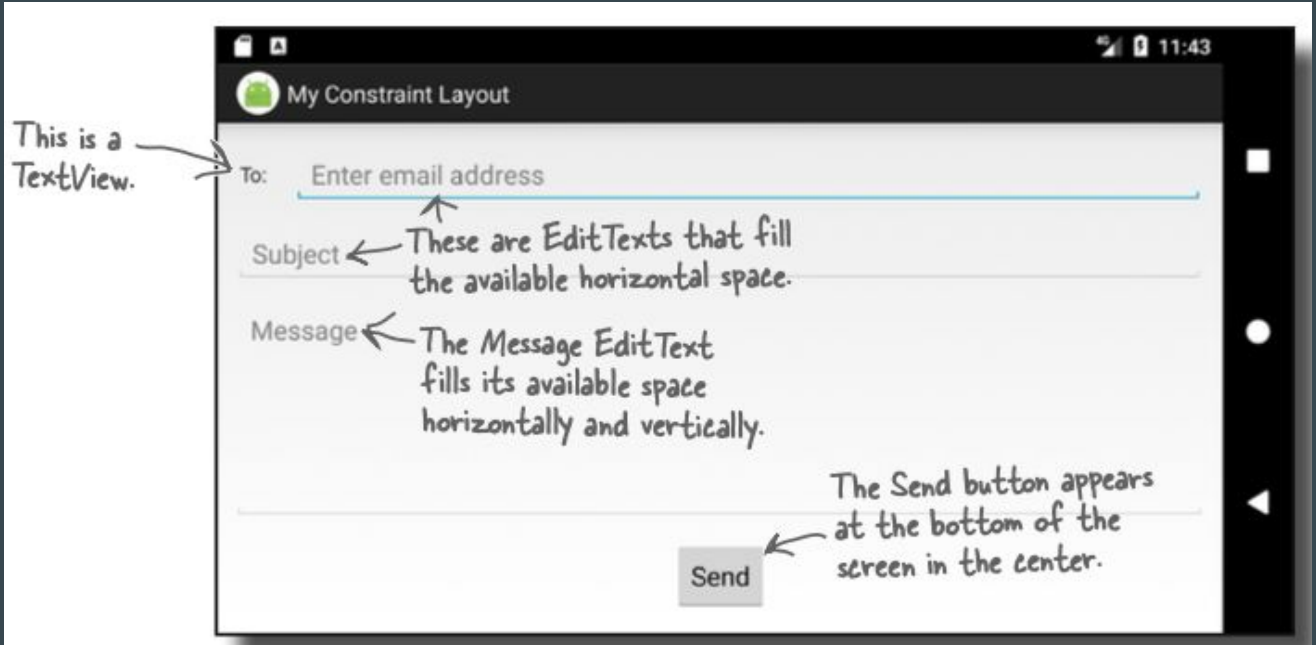


Frame Layout is one of the most efficient and simplest layouts used by Android to organize view controls. We use Frame layout as a container to displays only one view, or views which overlap.
Set Foreground and background

Layouts in Android

`Constraint Layout`

ConstraintLayout allows you to create large and complex layouts with a flat view hierarchy (no nested view groups). It's similar to RelativeLayout in that all views are laid out according to relationships between sibling views and the parent layout, but it's more flexible than RelativeLayout and easier to use with Android Studio's Layout Editor.


11:43
My Constraint Layout
This is a TextView.
To:
Enter email address
These are EditTexts that fill the available horizontal space.
Subject
Message
The Message EditText fills its available space horizontally and vertically.
The Send button appears at the bottom of the screen in the center.
Send

Horizontal Constraints;
Vertical Constraints

How to change a view's size

With a constraint layout, you have several different options for specifying a view's size:

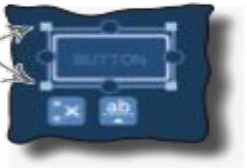
- Make it a fixed size by specifying a specific width and height.
- Use `wrap_content` to make the view just big enough to display its contents.
- Tell it to match the size of its constraints (if you've added constraints to opposite sides of the view).
- Specify a ratio for the width and height so that, for example, the view's width is twice the size of its height.

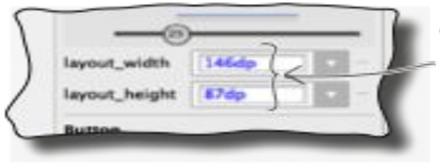
We'll go through these options one-by-one.

1. Make the view a fixed size

There are a couple of ways of using the design editor to make the view a fixed size. One way is to simply resize the view in the blueprint by clicking and dragging the square resizing handles on its corners. The other way is to type values into the `layout_width` and `layout_height` fields in the properties window:

You can resize a view using the square resizing handles on its corners.



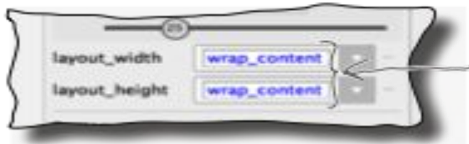

layout_width 146dp
layout_height 87dp
Button

You can also hardcode the width and height in the view's property window.

In general, **making your view a fixed size is a bad idea**, as it means the view can't grow or shrink to fit the size of its contents or the size of the screen.

2. Make it just big enough

To make the view just large enough to display its contents, change the view's `layout_width` and `layout_height` properties to `wrap_content`. You do this in the view's property window as shown here:

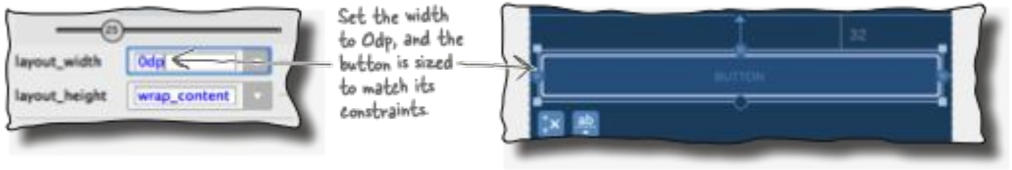

layout_width wrap_content
layout_height wrap_content

Setting the width and height to "wrap_content" makes it just large enough to display its contents, just as it does in other layouts.

3. Match the view's constraints

If you've added constraints to opposite sides of your view, you can make the view as wide as its constraints. You do this by setting its width and/or height to 0dp: set its width to 0dp to get the view to match the size of its horizontal constraints, and set its height to 0dp to get it to match the size of its vertical constraints.

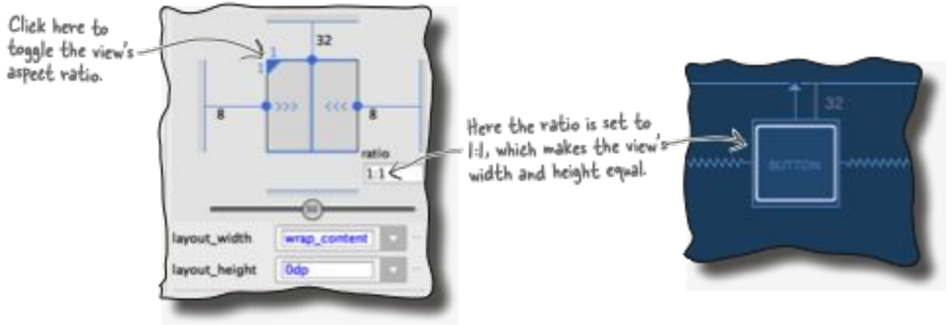
In our case, we've added constraints to the left and right sides of our button, so we can get the button to match the size of these constraints. To do this, go to the view's property window, and change the `layout_width` property to 0dp. In the blueprint, the button should expand to fill the available horizontal space (allowing for any margins):


25
Set the width to 0dp, and the button is sized to match its constraints.
32
BUTTON

layout_width 0dp
layout_height wrap_content

4. Specify the width:height ratio

Finally, you can specify an aspect ratio for the view's width and height. To do this, change the view's `layout_width` *or* `layout_height` to 0dp as you did above, then click in the top-left corner of the view diagram that's displayed in the property window. This should display a ratio field, which you can then update:


Click here to toggle the view's aspect ratio.
32
8
8
Here the ratio is set to 1:1, which makes the view's width and height equal.
32
BUTTON

ratio 1:1
layout_width wrap_content
layout_height 0dp

Now that you've seen how to resize a view, try experimenting with the different techniques before having a go at the exercise on the next page.

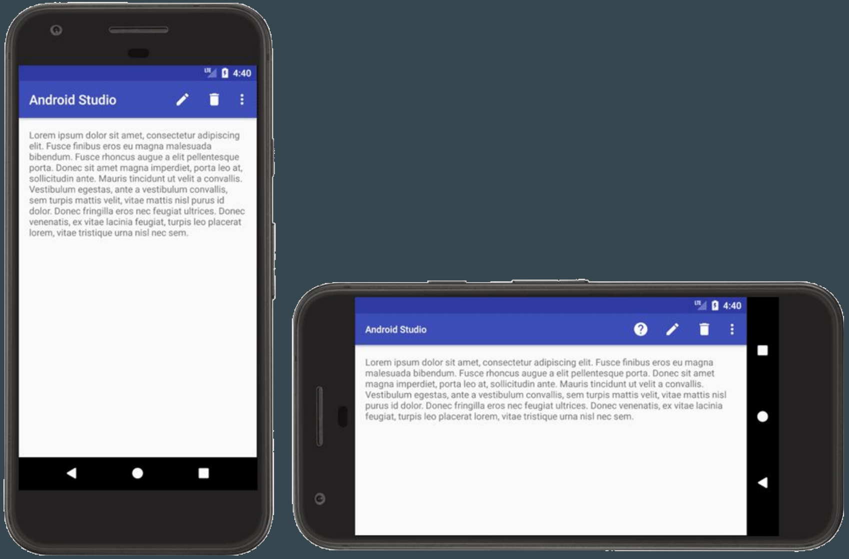
ANDROID MULTI SCREEN SCREEN SIZES

How does Android Deal with screens with varied sizes?

Till now all activities have been performed using different emulators, and all have gotten the same results. This is because of layouts.

- Use view dimensions that allow the layout to resize
- Create alternative UI layouts according to the screen configuration
- Provide bitmaps that can stretch with the views

1. Avoid hard-coded layout sizes
   a. Use - match_parent, wrap_content



Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce finibus eros eu magna malesuada bibendum. Fusce rhoncus augue a elit pellentesque porta. Donec sit amet magna imperdiet, porta leo at, sollicitudin ante. Mauris tincidunt ut velit a convallis. Vestibulum egestas, ante a vestibulum convallis, sem turpis mattis velit, vitae mattis nisl purus id dolor. Donec fringilla eros nec feugiat ultrices. Donec venenatis, ex vitae lacinia feugiat, turpis leo placerat lorem, vitae tristique urna nisl nec sem.

Lorem ipsum dolor sit amet, consectetur adipiscing elit. Fusce finibus eros eu magna malesuada bibendum. Fusce rhoncus augue a elit pellentesque porta. Donec sit amet magna imperdiet, porta leo at, sollicitudin ante. Mauris tincidunt ut velit a convallis. Vestibulum egestas, ante a vestibulum convallis, sem turpis mattis velit, vitae mattis nisl purus id dolor. Donec fringilla eros nec feugiat ultrices. Donec venenatis, ex vitae lacinia feugiat, turpis leo placerat lorem, vitae tristique urna nisl nec sem.

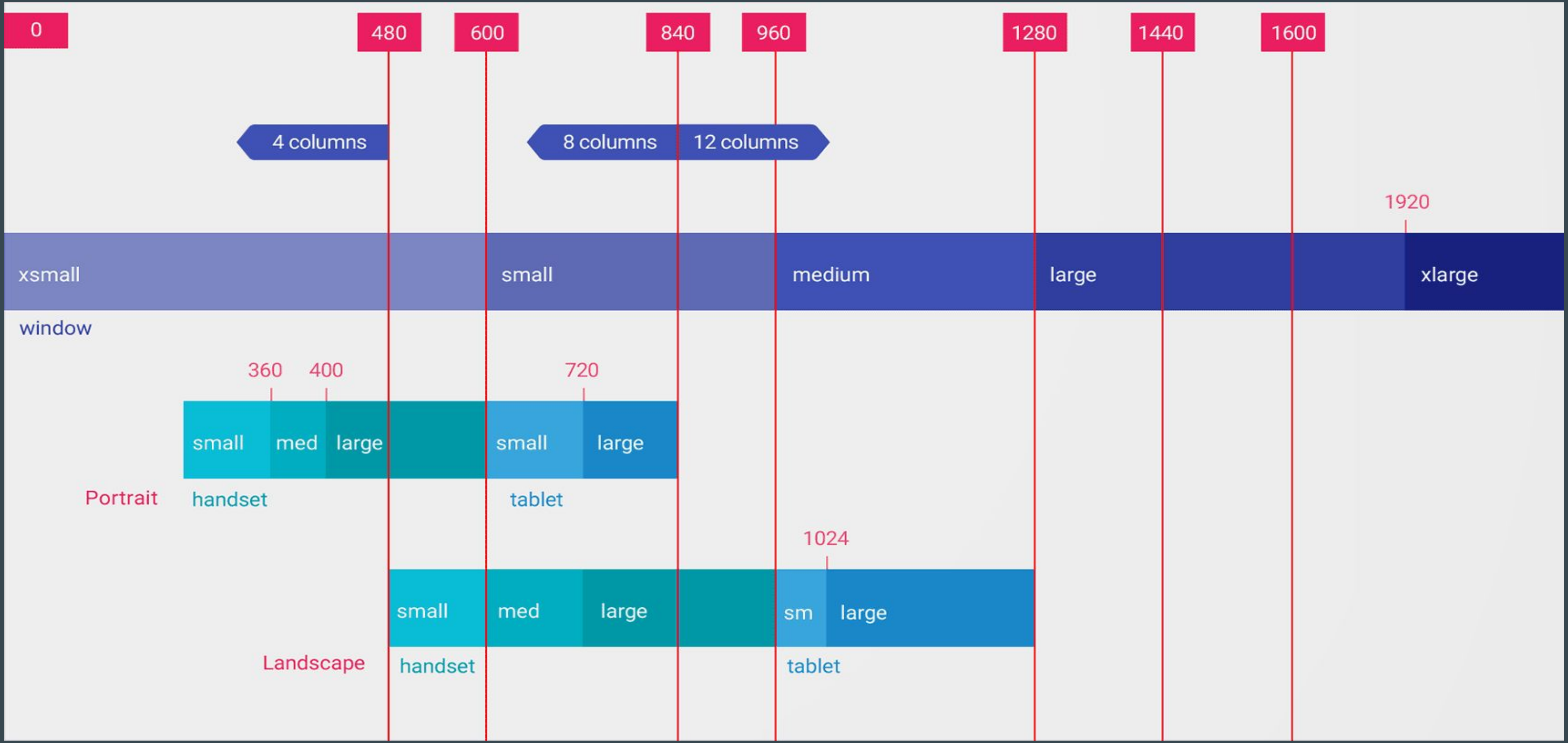
ANDROID MULTI SCREEN SCREEN SIZES

2. Create alternate Layouts: Although your layout should always respond to different screen sizes by stretching the space within and around its views, that might not provide the best user experience for every screen size. For example, the UI you designed for a phone, probably doesn't offer a good experience on a tablet. Therefore, your app should also provide alternative layout resources to optimize the UI design for certain screen sizes.

You can provide screen-specific layouts by creating additional res/layout/ directories—one for each screen configuration that requires a different layout—and then append a screen configuration qualifier to the layout directory name (such as layout-w600dp for screens that have 600dp of available width).

```
res/layout/main_activity.xml           # For handsets (smaller than 600dp available width)
res/layout-sw600dp/main_activity.xml   # For 7” tablets (600dp wide and bigger)
res/layout-sw600dp-land/main_activity.xml   # For 7” tablets in landscape
```

ANDROID MULTI SCREEN SCREEN SIZES



ANDROID MULTI SCREEN SCREEN SIZES

2. Create alternate Layouts: Although your layout should always respond to different screen sizes by stretching the space within and around its views, that might not provide the best user experience for every screen size. For example, the UI you designed for a phone, probably doesn't offer a good experience on a tablet. Therefore, your app should also provide alternative layout resources to optimize the UI design for certain screen sizes.

If your app supports Android 3.1 (API level 12) or lower, you need to use the legacy size qualifiers in addition to the smallest/available width qualifiers from above.

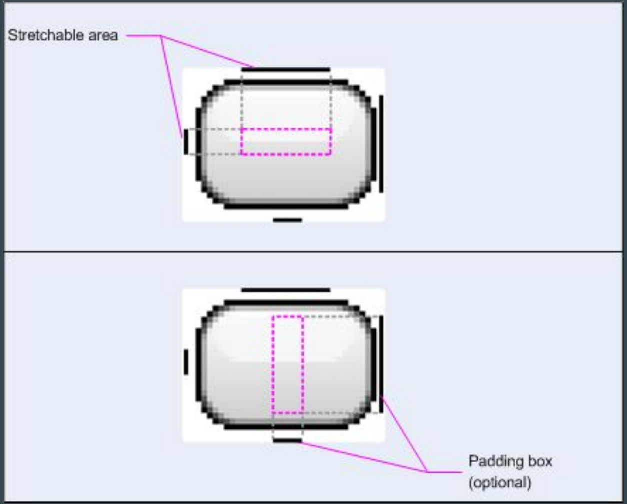
```
rees/layout-large/main_activity.xml     # For small tablets (640dp x 480dp and bigger)
res/layout-xlarge/main_activity.xml    # For large tablets (960dp x 720dp and bigger)
```

ANDROID MULTI SCREEN SCREEN SIZES

Create stretchable nine-patch bitmaps

If you use a bitmap as the background in a view that changes size, you will notice Android scales your images as the view grows or shrinks based on the size of the screen or content in the view. This often leads to visible blurring or other scaling artifacts. The solution is using nine-patch bitmaps, which are specially formatted PNG files that indicate which areas can and cannot be stretched.

A nine-patch bitmap is basically a standard PNG file, but with an extra 1px border that indicates which pixels should be stretched (and with a .9.png extension instead of just .png). As shown in figure 5, the intersection between the black lines on the left and top edge is the area of the bitmap that can be stretched.



ANDROID MULTI SCREEN SCREEN SIZES

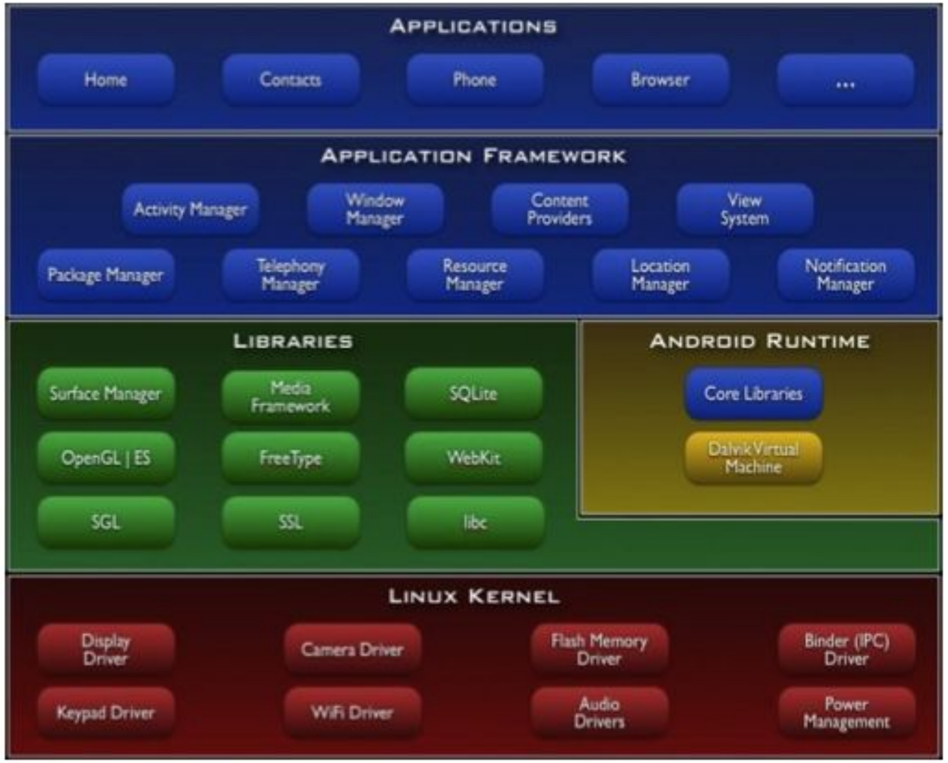
Test on all screen sizes

It's important to test your app on a variety of screen sizes so you can ensure your UI scales correctly. If you don't have access to physical devices for all the different screen sizes, you can use the Android Emulator to emulate any screen size.

Architecture of Android

5 Layer Architecture

- Linux kernel - Android is based on Linux. The main work of Linux kernel is to get the work done from hardware. Hardware could be anything camera, display or Wi-Fi
- Libraries – Are nothing but logical group of instructions that we want to give to the kernel to perform some action.
- Application Framework – Is a group of instances put together for purpose for developer to make things understandable. E.g Activity manager manages application's life cycle.
- Applications – Are built in our phone like browser, phone and camera etc



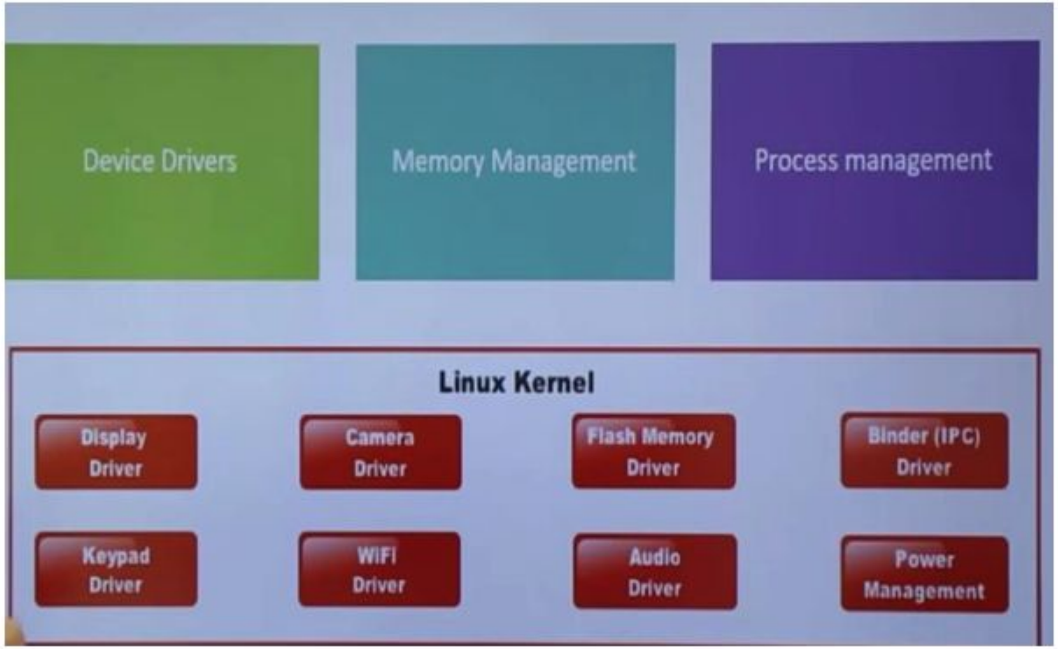
Linux Kernel

Kernel is involved in three main parts:

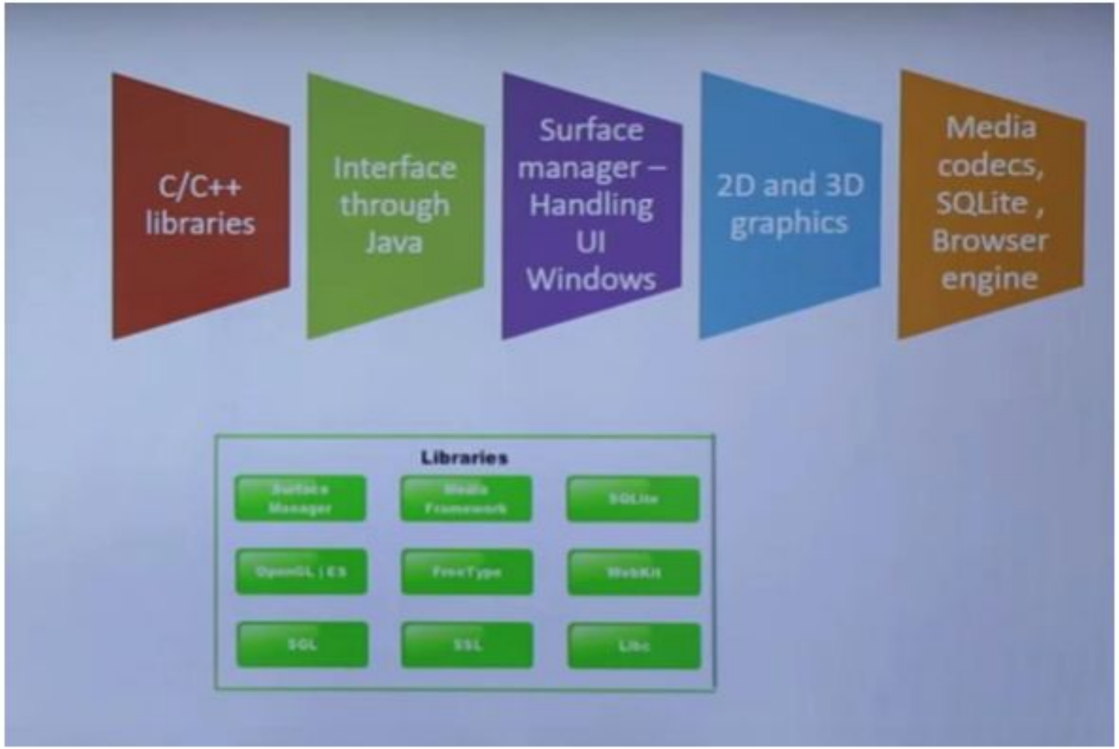
1. Device Drivers – Helps to get the work done from devices

2. Memory Management – How to manage memory

3. Process Management- A program in execution is a process E.g. Playing music, Turning Off/On the Bluetooth.

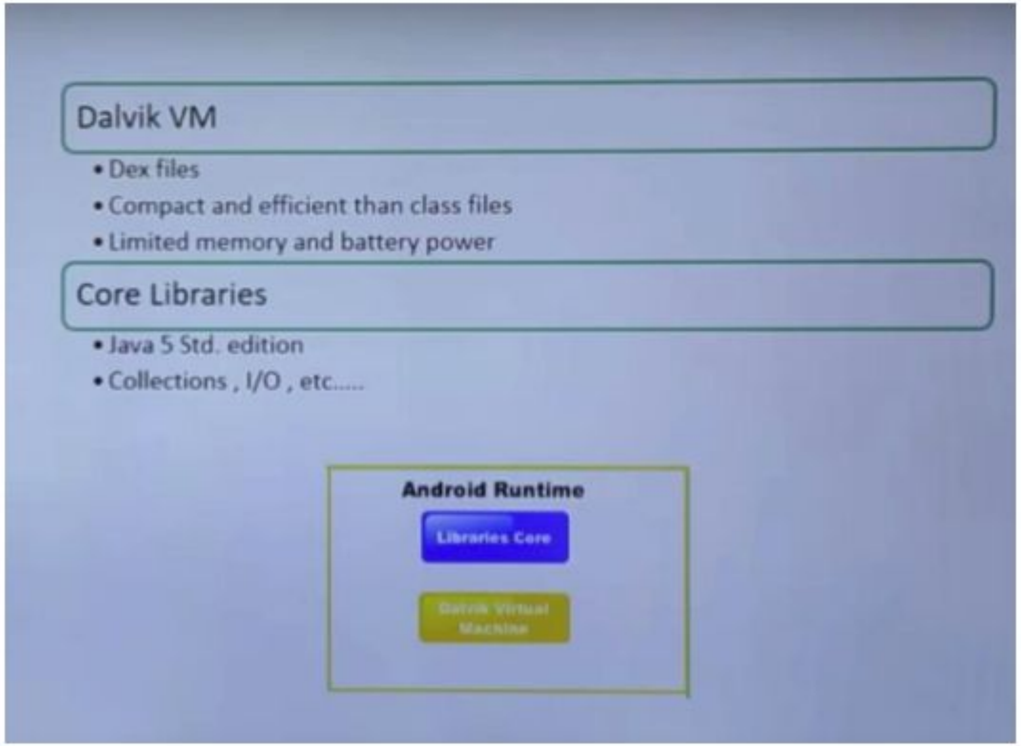


Libraries


C/C++ libraries
Interface through Java
Surface manager – Handling UI Windows
2D and 3D graphics
Media codecs, SQLite , Browser engine
Libraries
Surface Manager
Media Framework
SQLite
OpenGL | ES
FreeType
WebKit
SGL
SSL
Libc

Android Runtime

1. Like for Java we have Java Virtual Machines(JVM), for android we have Dalvik Virtual Machines(DVM). The Dalvik Virtual Machine (DVM) is an android virtual machine optimized for mobile devices. It optimizes the virtual machine for memory, battery life and performance. Dalvik is a name of a town in Iceland. The Dalvik VM was written by Dan Bornstein.
2. Like .jar files we have dex files here. The Dex compiler converts the class files into the .dex file that run on the Dalvik VM. Multiple class files are converted into one dex file.
3. DVM and Core libraries together form Android Run Time.



Dalvik VM

- Dex files
- Compact and efficient than class files
- Limited memory and battery power

Core Libraries

- Java 5 Std. edition
- Collections , I/O , etc.....

Android Runtime

Libraries Core

Dalvik Virtual Machine

ANDROID DATABASE MANAGEMENT SYSTEM

And on Android you usually keep your data safe inside a SQLite database.

Operations that can be performed on the database -

1. Create Table.
2. Insert in Table
3. Update in Table
4. Delete in Table

ANDROID DATABASE MANAGEMENT SYSTEM

1. **It's lightweight:** Most database systems need a special database server process in order to work. SQLite doesn't; a SQLite database is just a file. When you're not using the database, it doesn't use up any processor time. That's important on a mobile device, because we don't want to drain the battery.

2. **It's optimized for a single user:** Our app is the only thing that will talk to the database, so we shouldn't have to identify ourselves with a username and password.

3. **It's stable and fast:.** SQLite databases are amazingly stable. They can handle database transactions, which means if you're updating several pieces of data and mess up, SQLite can roll the data back. Also, the code that reads and writes the data is written in optimized C code. Not only is it fast, but it also reduces the amount of processor power it needs.

ANDROID DATABASE MANAGEMENT SYSTEM

CLASSES IN USE:

1. SQLite Helper: A SQLite helper enables you to create and manage databases. You create one by extending the SQLiteOpenHelper class.
2. The SQLite Database: The SQLiteDatabase class gives you access to the database. It’s like a SQLConnection in JDBC.
3. Cursors: A Cursor lets you read from and write to the database. It’s like a ResultSet in JDBC.

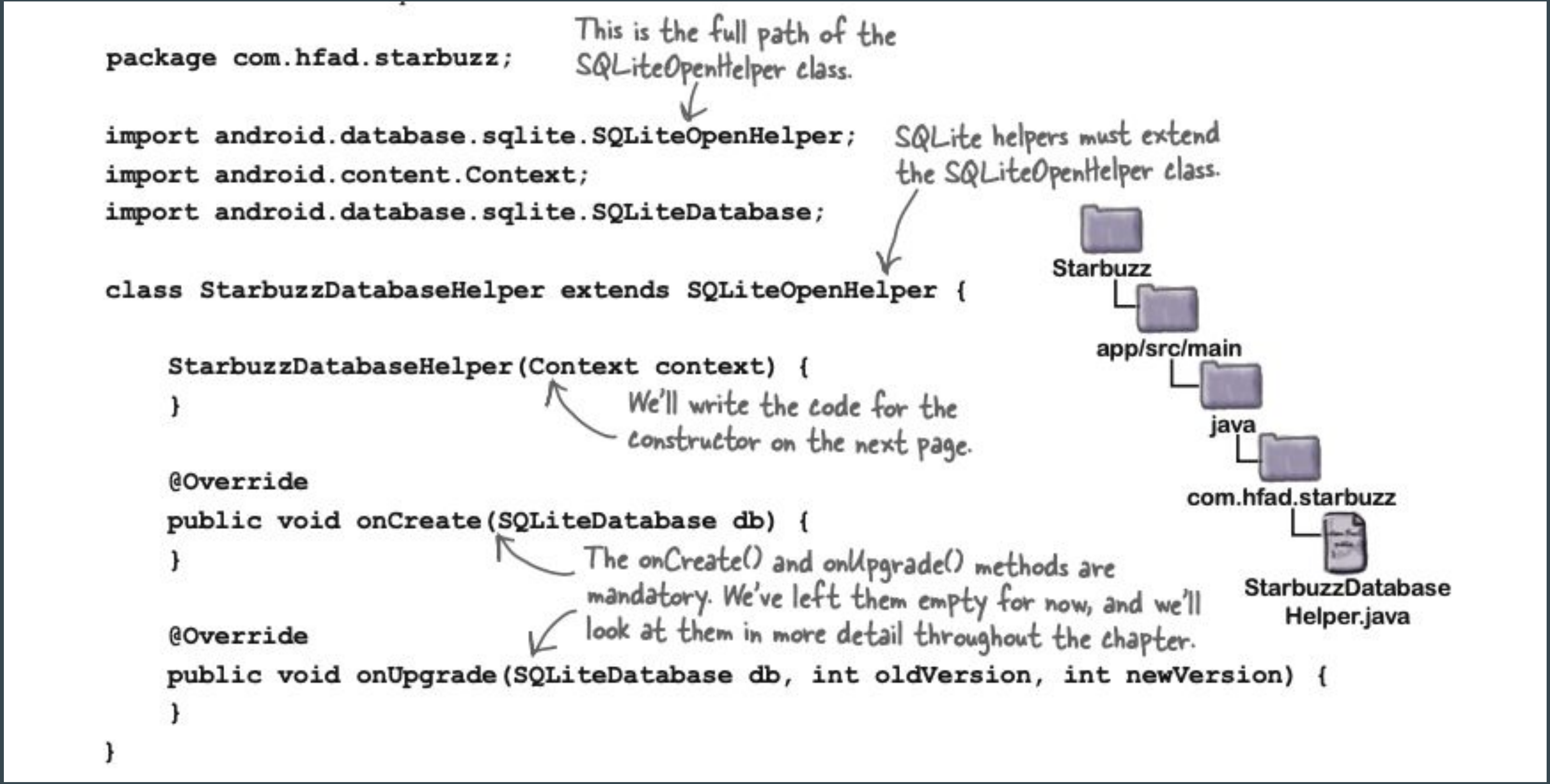
ANDROID DATABASE MANAGEMENT SYSTEM

You create a SQLite helper by writing a class that extends the SQLiteOpenHelper class. When you do this, you must override the onCreate() and onUpgrade() methods. These methods are mandatory.

The onCreate() method gets called when the database first gets created on the device. The method should include all the code needed to create the tables you need for your app.

The onUpgrade() method gets called when the database needs to be upgraded. As an example, if you need to modify the structure of the database after it's been released, this is the method to do it in.

ANDROID DATABASE MANAGEMENT SYSTEM



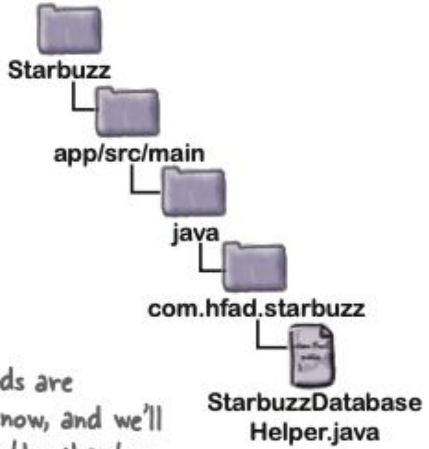
```
package com.hfad.starbuzz;

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

class StarbuzzDatabaseHelper extends SQLiteOpenHelper {

    StarbuzzDatabaseHelper(Context context) {
    }

    @Override
    public void onCreate(SQLiteDatabase db) {
    }

    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
```

This is the full path of the SQLiteOpenHelper class.

SQLite helpers must extend the SQLiteOpenHelper class.

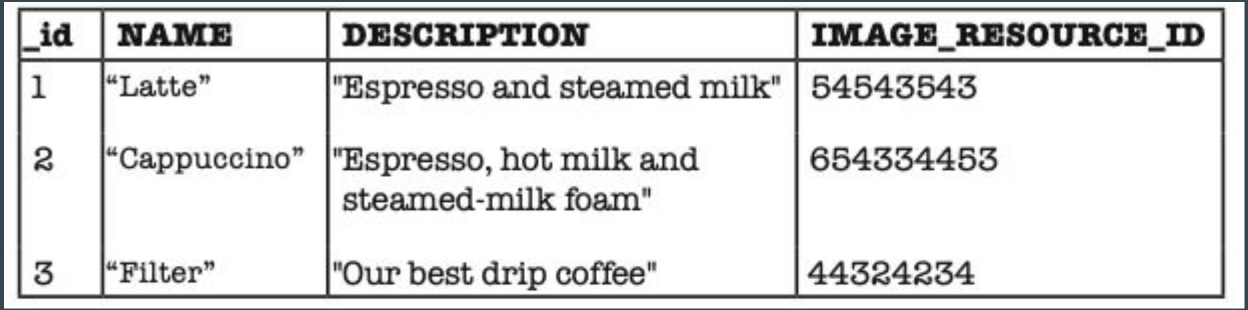
We'll write the code for the constructor on the next page.

The onCreate() and onUpgrade() methods are mandatory. We've left them empty for now, and we'll look at them in more detail throughout the chapter.



ANDROID DATABASE MANAGEMENT SYSTEM

It is an android convention to make sure to call your primary key with columns _id



_id	NAME	DESCRIPTION	IMAGE_RESOURCE_ID
1	“Latte”	"Espresso and steamed milk"	54543543
2	“Cappuccino”	"Espresso, hot milk and steamed-milk foam"	654334453
3	“Filter”	"Our best drip coffee"	44324234

Database Types:
INTEGER Any integer type
TEXT Any character type
REAL Any floating-point number
NUMERIC Booleans, dates, and date-times
BLOB Binary Large Object

ANDROID DATABASE MANAGEMENT SYSTEM

Unlike most database systems, you don’t need to specify the column size in SQLite. Under the hood, the data type is translated into a much broader storage class. This means you can say very generally what kind of data you’re going to store, but you’re not forced to be specific about the size of data.

ANDROID DATABASE MANAGEMENT SYSTEM

The onCreate() method is called when the database is created

```
CREATE TABLE DRINK (_id INTEGER PRIMARY KEY AUTOINCREMENT,
NAME TEXT,
DESCRIPTION TEXT,
 IMAGE_RESOURCE_ID INTEGER)
```

You can use the SQLiteDatabase execSQL() method to execute SQL on the database. This method has one parameter, the SQL you want to execute.

```
public void onCreate(SQLiteDatabase db){
		db.execSQL(String sql);
}
```

ANDROID DATABASE MANAGEMENT SYSTEM

To insert data into a table in a SQLite database, you start by specifying what values you want to insert into the table. To do this, you first create a ContentValues object.

```
ContentValues drinkValues = new ContentValues();
```

Format to enter data:

```
contentValues.put("NAME", "value");
```

```
drinkValues.put("NAME", "Latte");
drinkValues.put("DESCRIPTION", "Espresso and steamed milk");
drinkValues.put("IMAGE_RESOURCE_ID", R.drawable.latte);
```

ANDROID DATABASE MANAGEMENT SYSTEM

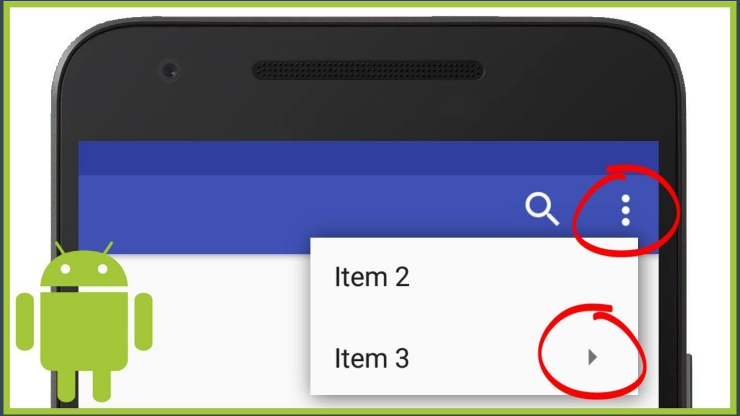
Once you’ve added a row of data to the ContentValues object, you insert it into the table using the SQLiteDatabase insert() method.

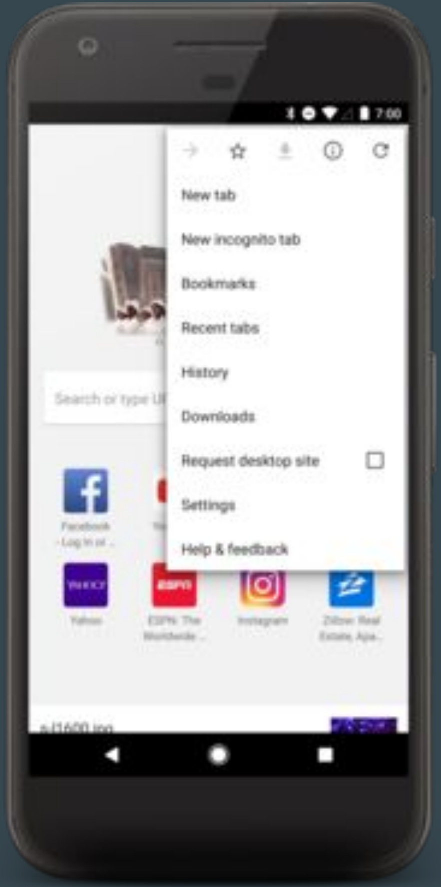
This method inserts data into a table, and returns the ID of the record once it’s been inserted. If the method is unable to insert the record, it returns a value of -1. As an example, here’s how you’d insert the data from drinkValues into the DRINK table:

```
db.insert("DRINK", null, drinkValues);
```

ANDROID MENUS

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

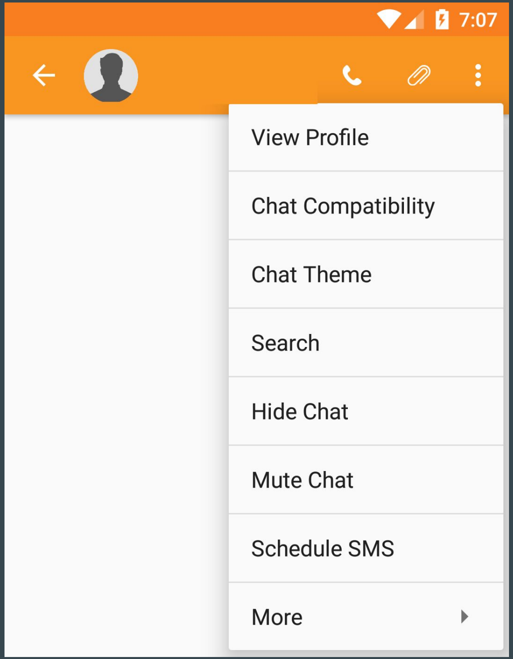




ANDROID MENUS

Types of Menu

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

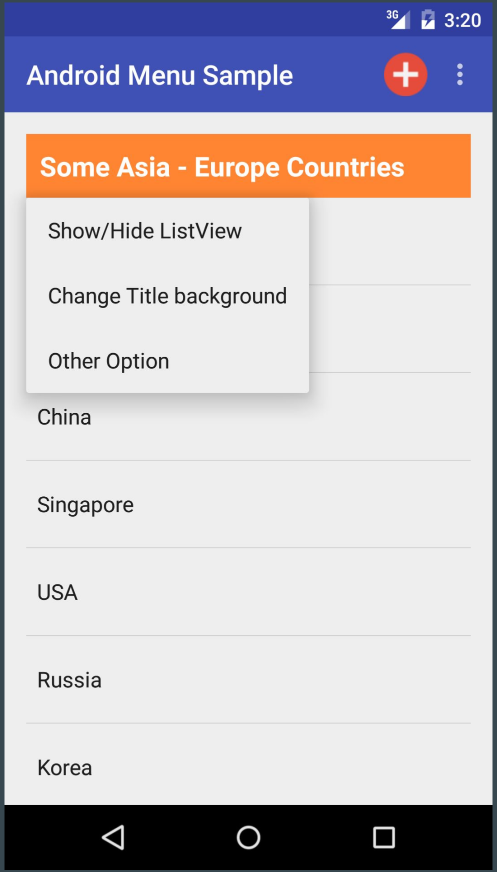


ANDROID MENUS

Types of Menu

2. Context Menu: A context menu is a floating menu that appears when the user performs a long-click on an element. It provides actions that affect the selected content or context frame.

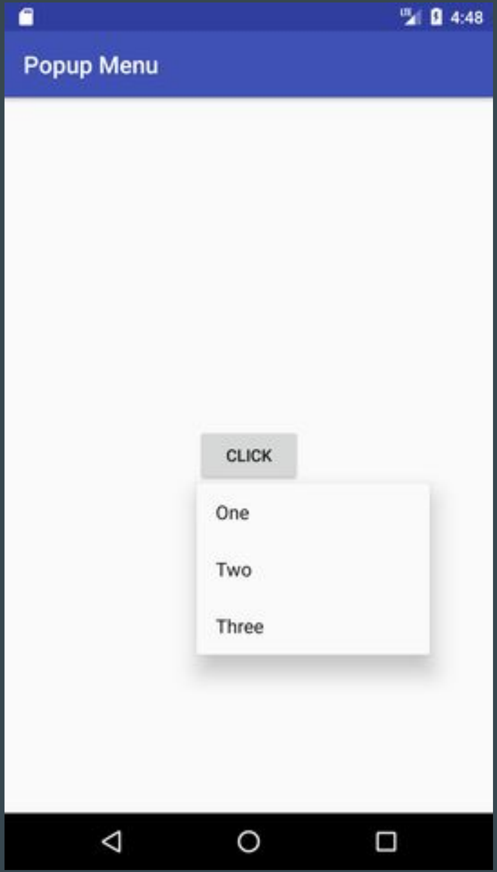
A contextual menu offers actions that affect a specific item or context frame in the UI. You can provide a context menu for any view, but they are most often used for items in a ListView, GridView, or other view collections in which the user can perform direct actions on each item



ANDROID MENUS

Types of Menu

3. Pop Up Menus: A PopupMenu is a modal menu anchored to a View. It appears below the anchor view if there is room, or above the view otherwise. It's useful for:
   a. Providing an overflow-style menu for actions that relate to specific content (such as Gmail's email headers)
   b. Providing a second part of a command sentence (such as a button marked "Add" that produces a popup menu with different "Add" options).
   c.



ANDROID MENUS

Defining a Menu in XML

For all menu types, Android provides a standard XML format to define menu items. Instead of building a menu in your activity's code, you should define a menu and all its items in an XML menu resource. You can then inflate the menu resource (load it as a Menu object) in your activity or fragment.

Using a menu resource is a good practice for a few reasons:

- It's easier to visualize the menu structure in XML.
- It separates the content for the menu from your application's behavioral code.
- It allows you to create alternative menu configurations for different platform versions, screen sizes, and other configurations by leveraging the app resources framework.
-

ANDROID MENUS

To define the menu, create an XML file inside your project's res/menu/ directory and build the menu with the following elements:

<menu>
Defines a Menu, which is a container for menu items. A <menu> element must be the root node for the file and can hold one or more <item> and <group> elements.
<item>
Creates a MenuItem, which represents a single item in a menu. This element may contain a nested <menu> element in order to create a submenu.
<group>
An optional, invisible container for <item> elements. It allows you to categorize menu items so they share properties such as active state and visibility.

ANDROID MENUS

You need to override 2 functions to make your menu work.

1. To specify the options menu for an activity, override onCreateOptionsMenu(). In this method, you can inflate your menu resource (defined in XML) into the Menu provided in the callback.

   ```
   public boolean onCreateOptionsMenu(Menu menu) {
       MenuInflater inflater = getMenuInflater();
       inflater.inflate(R.menu.game_menu, menu);
       return true;
   }
   ```

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

ANDROID MENUS

You need to override 2 functions to make your menu work.

When the user selects an item from the options menu (including action items in the app bar), the system calls your activity's onOptionsItemSelected() method. This method passes the MenuItem selected. You can identify the item by calling getItemId(), which returns the unique ID for the menu item (defined by the android:id attribute in the menu resource or with an integer given to the add() method).

a. You can match this ID against known menu items to perform the appropriate action.:

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case R.id.new_game:
            newGame();
            return true;
        case R.id.help:
            showHelp();
            return true;
        default:
            return super.onOptionsItemSelected(item);
    }   }
```