VEMU INSTITUTE OF TECHNOLOGY

P.Kothakota, Near Pakala, Chittoor

LECTURE NOTES

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

SUBJECT NAME: MOBILE APPLICATION DEVELOPEMENT

BRANCH: CSE

YEAR AND SEMESTER: IV – I

COURSE: B.TECH

REGULATION: R15

Unit 1
Introduction to Android

Syllabus: The Android 4.1 jelly Bean SDK, Understanding the Android Software Stack, installing the Android SDK, Creating Android Virtual Devices, Creating the First Android Project, Using the Text view Control, Using the Android Emulator, The Android Debug Bridge(ADB), Launching Android Applications on a Handset.

INTRODUCTION

When we talked about operating systems few years ago, the most common answers were Windows, Linux, and mac operating system. However, with the undying competition in the mobile phones market, the next big thing entered was ANDROID, which in no time became the heart of smart phones. Android provides a rich application framework that allows you to build innovative apps and games for mobile devices in a Java language environment.

Android is a mobile operating system developed by Google, based on a modified version of the Linux kernel and other open source software and designed primarily for touch screen mobile devices such as smart phones and tablets. In addition, Google has further developed Android TV for televisions, Android Auto for cars, and Wear OS for wrist watches, each with a specialized user interface. Variants of Android are also used on game consoles, digital cameras, PCs and other electronics.

The Android Operating System is a Linux-based OS developed by the Open Handset Alliance (OHA). The Android OS was originally created by Android, Inc., which was bought by Google in 2005. Google teamed up with other companies to form the Open Handset Alliance (OHA), which has become responsible for the continued development of the Android OS.

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

The android has got millions of apps available that can help you managing your life one or other way and it is available low cost in market at that reasons android is very popular.

Each time the OHA releases an Android version, it names the release after a dessert. Android 1.5 is known as Cupcake, 1.6 as Donut, 2.0/2.1 as Eclair, 2.2 as Froyo and 2.3 is dubbed Gingerbread. Once a version is released, so is its source code.

The Android OS is designed for phones. The important features of android are given below:

1) It is open-source.

2) Anyone can customize the Android Platform.

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

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

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

Software developers who want to create applications for the Android OS can download the Android Software Development Kit (SDK) for a specific version. The SDK includes a debugger, libraries, an emulator, some documentation, sample code and tutorials. For faster development, interested parties

can use graphical integrated development environments (IDEs) such as Eclipse to write applications in Java.

Android Emulator:

The Emulator is a new application in android operating system. The emulator is a new prototype that is used to develop and test android applications without using any physical device.

The android emulator has all of the hardware and software features like mobile device except phone calls. It provides a variety of navigation and control keys. It also provides a screen to display your application. The emulators utilize the android virtual device configurations. Once your application is running on it, it can use services of the android platform to help other applications, access the network, play audio, video, store and retrieve the data.

Android versions: Google did not attach any high-calorie code name to its initial versions 1.0 and 1.1 of the Android Operating System. The code names of android ranges from A to N currently, such as Aestro, Blender, Cupcake, Donut, Eclair, Froyo, Gingerbread, Honeycomb, Ice Cream Sandwitch, Jelly Bean, KitKat, Lollipop and Marshmallow. Let's understand the android history in a sequence.

THE ANDROID 4.1 JELLY BEAN SDK

The Android 4.1 Jelly Bean SDK was released with new features for developers in July 2012. It improves the beauty and simplicity of Android 4.0 and is a major platform release that adds a variety of new features for users and app developers. A few of the big features of this release include the following:

• **Project Butter**—Makes the Jelly Bean UI faster and more responsive. Also CPU Touch Responsiveness is added, which increases CPU performance whenever the screen is touched. It uses the finger's speed and direction to predict where it will be located after some milliseconds, hence making the navigation faster.

• **Faster speech recognition**—Speech recognition is now faster and doesn't require any network to convert voice into text. That is, users can dictate to the device without an Internet connection.

• **Improved notification system**— The notifications include pictures and lists along with text. Notifications can be expanded or collapsed through a variety of gestures, and users can block notifications if desired. The notifications also include action buttons that enable users to call directly from the notification menu rather replying to email.

• **Supports new languages**—Jelly Bean includes support for several languages including Arabic, Hebrew, Hindi, and Thai. It also supports bidirectional text.

• **Predictive keyboard**—On the basis of the current context, the next word of the message is automatically predicted.

• **Auto-arranging Home screen**—Icons and widgets automatically resize and realign as per the existing space.

• **Helpful for visually impaired users**—The Gesture Mode combined with voice helps visually impaired users to easily navigate the user interface.

• **Improved Camera app**—The Jelly Bean Camera app includes a new review mode of the captured photos. Users can swipe in from the right of the screen to quickly view the captured photos. Also, users can pinch to switch to a new film strip view, where they can swipe to delete photos.

• **Better communication in Jelly Bean**—Two devices can communicate with Near Field Communication (NFC); that is, two NFC-enabled Android devices can be tapped to share data. Also, Android devices can be paired to Bluetooth devices that support the Simple Secure Pairing standard by just tapping them together.

• **Improved Google Voice search**—Jelly Bean is equipped with a question and answer search method that helps in solving users' queries similar to Apple's popular Siri.

• **Face Unlock**—Unlocks the device when the user looks at it. It also prevents the screen from blacking out. Optionally "blink" can be used to confirm that a live person is unlocking the device instead of a photo.

• **Google Now**—Provides users "just the right information at just the right time." It displays cards to show desired information automatically. For example, the Places card displays nearby restaurants and shops while moving; the Transit card displays information on the next train or bus when the user is near a bus stop or railway station; the Sports card displays live scores or upcoming game events; the Weather card displays the weather conditions at a user's current location, and so on.

• **Google Play Widgets**—Provides quick and easy access to movies, games, magazines, and other media on the device. It also suggests new purchases on Google Play.

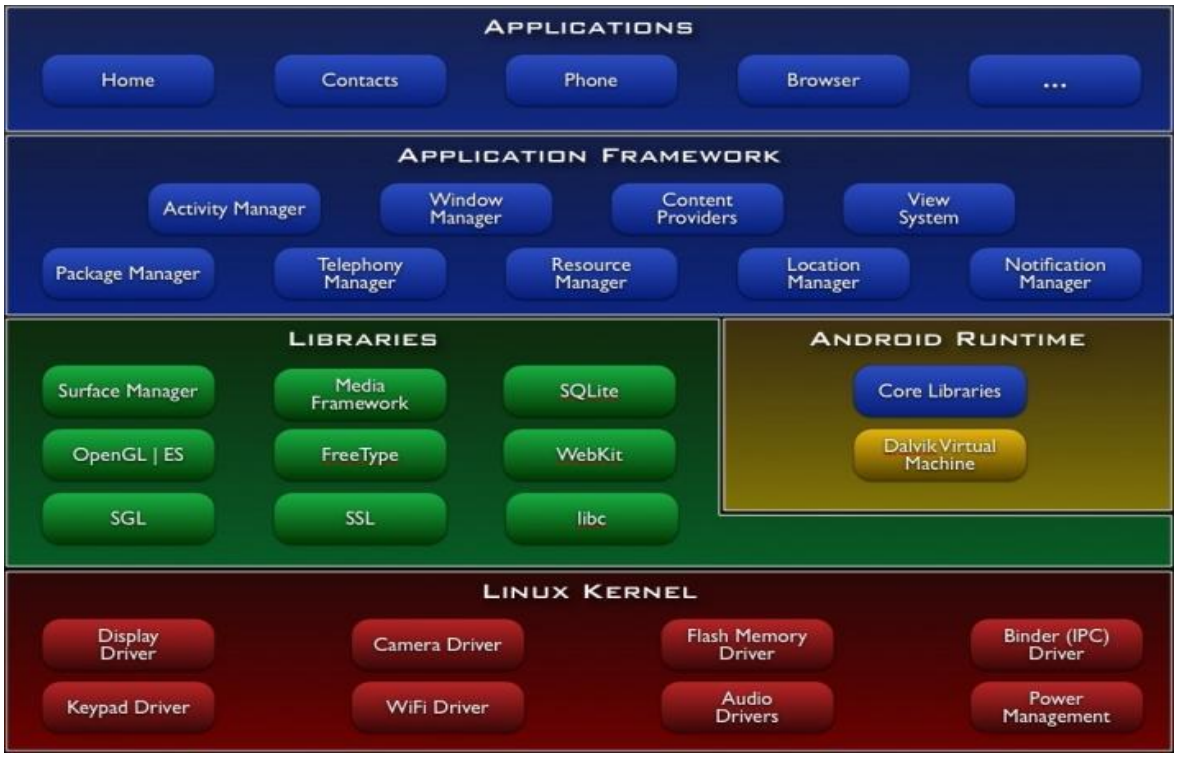
• **Faster Google Search**—Google Search can be opened quickly, from the lock screen and from the system bar by swiping up and also by tapping a hardware search key if it is available on the device.

• **Supports antipiracy**—This feature supports developers in the sense that the applications are encrypted with a device-specific key making it difficult to copy and upload them to the Internet.

UNDERSTANDING THE ANDROID SOFTWARE STACK/ Android Architecture

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

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



1) Linux kernel

It is the heart of android architecture that exists at the root of android architecture. Linux kernel is responsible for device drivers, power management, memory management, device management and resource access. This layer is the foundation of the Android Platform.

- Contains all low level drivers for various hardware components support.
- Android Runtime relies on Linux Kernel for core system services like,
- Memory, process management, threading etc.
- Network stack
- Driver model
- Security and more.

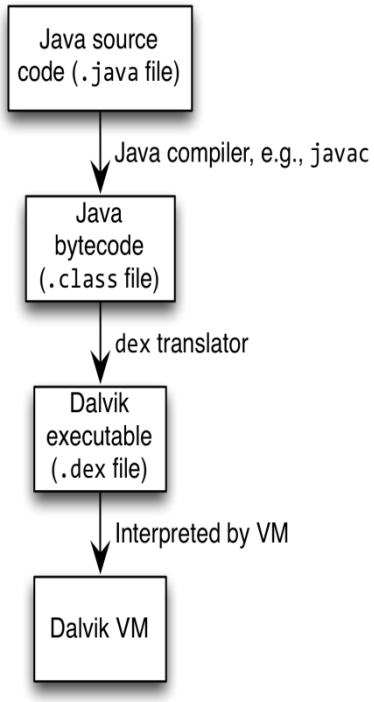
2) Libraries

On top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

- **SQLite** Library used for data storage and light in terms of mobile memory footprints and task execution.
- **WebKit** Library mainly provides Web Browsing engine and a lot more related features.
- The **surface manager** library is responsible for rendering windows and drawing surfaces of various apps on the screen.
- The **media framework** library provides media codecs for audio and video.
- The **OpenGl** (Open Graphics Library) and **SGL**(Scalable Graphics Library) are the graphics libraries for 3D and 2D rendering, respectively.
- The **FreeType** Library is used for rendering fonts.

3) Android Runtime

In android runtime, there are core libraries and DVM (Dalvik Virtual Machine) which is responsible to run android application. DVM is like JVM but it is optimized for mobile devices. It consumes less memory and provides fast performance. The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine.



- Dalvik is a specialized virtual machine designed specifically for Android and optimized for battery-powered mobile devices with limited memory and CPU.
- Android apps execute on Dalvik VM, a "clean-room" implementation of JVM
- Dalvik optimized for efficient execution
- Dalvik: register-based VM, unlike Oracle's stack-based JVM
- Java .class bytecode translated to Dalvik EXecutable (DEX) bytecode, which Dalvik interprets

4) Android Framework

On the top of Native libraries and android runtime, there is android framework. Android framework includes Android API's such as UI (User Interface), telephony, resources, locations, Content Providers (data) and package managers. It provides a lot of classes and interfaces for android application development.

- **Activity Manager**: manages the life cycle of an applications and maintains the back stack as well so that the applications running on different processes has smooth navigations.
- **Package Manager**: keeps track of which applications are installed in your device.
- **Window Manager** : Manages windows which are java programming abstractions on top of lower level surfaces provided by surface manager.
- **Telephony Managers**: manages the API which is use to build the phone applications

- **Content Providers:** Provide feature where one application can share the data with another application. like phone number , address, etc
- **View Manager** : Buttons , Edit text , all the building blocks of UI, event dispatching etc.

5) Applications

- On the top of android framework, there are applications. All applications such as home, contact, settings, games, browsers are using android framework that uses android runtime and libraries. Android runtime and native libraries are using linux kernel. Any applications that you write are located at this layer.

Android Studio Installation:

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

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

http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html

3)To set up Android Studio on **Windows**:

1. Launch the .exe file you just downloaded.

2. Follow the setup wizard to install Android Studio and any necessary SDK tools.

On some Windows systems, the launcher script does not find where Java is installed. If you encounter this problem, you need to set an environment variable indicating the correct location.

Select Start menu > Computer > System Properties > Advanced System Properties. Then open Advanced tab > Environment Variables and add a new system variable JAVA_HOME that points to your JDK folder, for example C:\Program Files\Java\jdk1.8.x.(where x is version number).

To set up Android Studio on Mac OSX:

1. Launch the .dmg file you just downloaded.

2. Drag and drop Android Studio into the Applications folder.

3. Open Android Studio and follow the setup wizard to install any necessary SDK tools.

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

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

/Users/<user>/Library/Android/sdk/ To set up Android Studio on **Linux**:
1. Unpack the downloaded ZIP file into an appropriate location for your applications.

2. To launch Android Studio, navigate to the android-studio/bin/ directory in a terminal and execute studio.sh. You may want to add android-studio/bin/ to your PATH environmental variable so that you can start Android Studio from any directory.

3. Follow the setup wizard to install any necessary SDK tools.

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

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

To start adding packages, launch the Android SDK Manager in one of the following ways:

- In Android Studio, click SDK Manager in the toolbar.
- If you're not using Android Studio:

Windows: Double-click the SDK Manager.exe file at the root of the Android SDK directory.

Mac/Linux: Open a terminal and navigate to the tools/ directory in the Android SDK, then execute android sdk.

5)Now get all the SDK tools, support libraries for additional APIs, Google Play services (if you need to use them).
6)Once you've selected all the desired packages, continue to install:

Click Install X packages.

In the next window, double-click each package name on the left to accept the license agreement for each.

Click Install.

The download progress is shown at the bottom of the SDK Manager window. Do not exit the SDK Manager or it will cancel the download.

7)Now that we have downloaded and installed everything we need. Enjoy your experience with Android. Best of luck from Internshala.com

INSTALLING THE ANDROID SDK

For developing native Android applications that you can publish on the Google Play marketplace, you need to install the following four applications:

• **The Java Development Kit (JDK)** can be downloaded from http://oracle.com/technetwork/java/javase/downloads/index.html.

• **The Eclipse IDE** can be downloaded from http://www.eclipse.org/downloads/.

• **The Android Platform SDK Starter Package** can be download from http://developer.android.com/sdk/index.html.

• **The Android Development Tools (ADT) Plug-in** can be downloaded from http://developer.android.com/sdk/eclipse-adt.html. The plug-in contains project templates and Eclipse tools that help in creating and managing Android projects.

The Android SDK is not a full development environment and includes only the core SDK Tools, which are used to download the rest of the SDK components. This means that after installing the Android SDK Tools, you need to install Android platform tools and the other components required for developing Android applications. Go

to http://developer.android.com/sdk/index.html and download the package by selecting the link for your operating system.

The first screen is a Welcome screen. Select the Next button to move to the next screen. Because the Android SDK requires the Java SE Development Kit for its operation, it checks for the presence of JDK on your computer.

If Java is already installed on your computer before beginning with Android SDK installation, the wizard detects its presence and displays the version number of the JDK found on the machine, as shown in Figure.

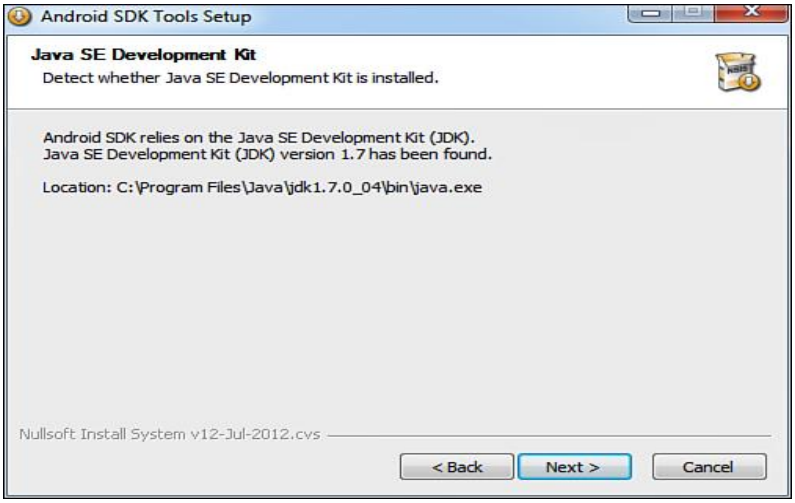
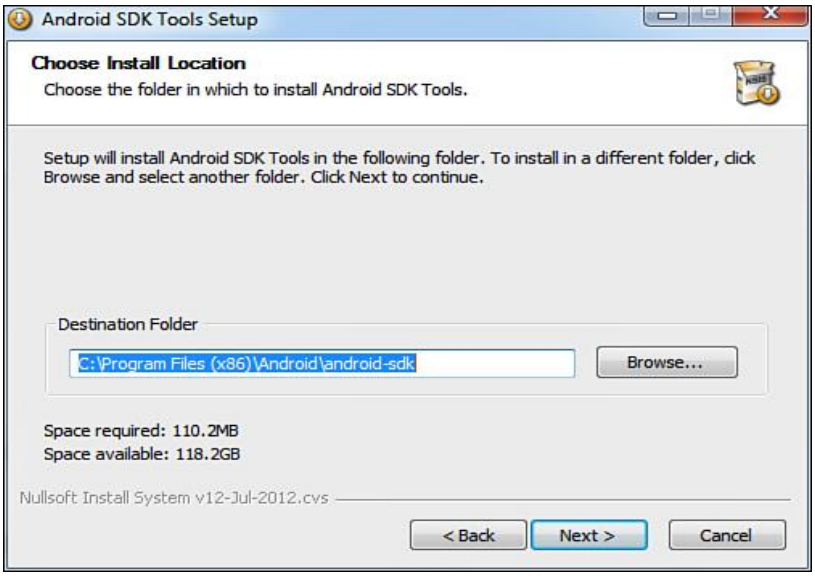


Figure Dialog box informing you that the JDK is already installed on the computer

Select the Next button. You get a dialog box asking you to choose the users for which Android SDK is being installed. The following two options are displayed in the dialog box:

- Install for anyone using this computer
- Install just for me

Select the Install for anyone using this computer option and click Next. The next dialog prompts you for the location to install the Android SDK Tools, as shown in below Figure . The dialog also displays the default directory location for installing Android SDK Tools as C:\Program Files (x86)\Android\android-sdk, which you can change by selecting the Browse button. Keep the default directory for installing Android SDK Tools unchanged; then select the Next button to continue.



The next dialog box asks you to specify the Start Menu folder where you want the program's shortcuts to appear, as shown.

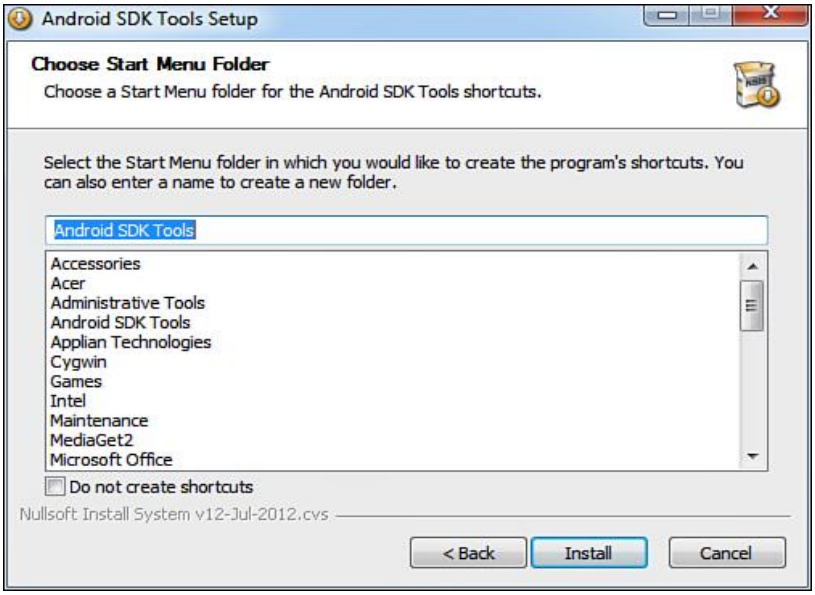


Figure: Dialog box to select the Start menu shortcut folder

A default folder name appears called Android SDK Tools. If you do not want to make a Start Menu folder, select the Do not create shortcuts check box. Let's create the Start Menu folder by keeping the default folder name and selecting the Install button to begin the installation of the Android SDK Tools. After all the files have been downloaded and installed on the computer, select the Next button. The next dialog box tells you that the Android SDK Tools Setup Wizard is complete and the Android SDK Tools have successfully installed on the computer. Select Finish to exit the wizard, as shown in Figure

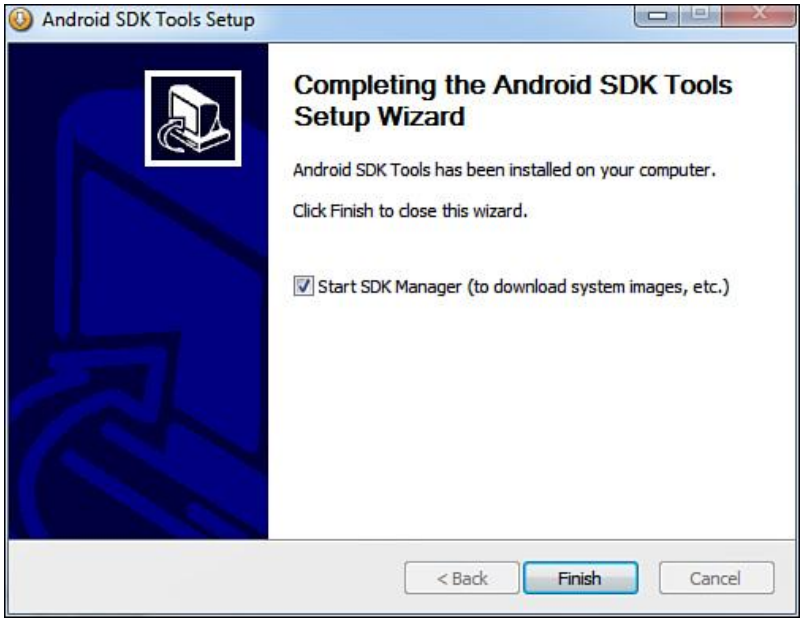


Figure: Successful installation of the Android SDK Tools dialog box

Note that the check box Start SDK Manager (to download system images) is checked by default. It means that after the Finish button is clicked, the Android SDK Manager, one of the tools in the Android SDK Tools package, will be launched. Android SDK is installed in two phases. The first phase is the installation of the SDK, which installs the Android SDK Tools, and the second phase is installation of the Android platforms and other components.

An Android application is a combination of several small components that include Java files, XML resource and layout files, manifest files, and much more. It would be very time-consuming to create all these components manually. So, you can use the following applications to help you:

• **Eclipse IDE**—An IDE that makes the task of creating Java applications easy. It provides a complete platform for developing Java applications with compiling, debugging, and testing support.

• **Android Development Tools (ADT) plug-in**—A plug-in that's added to the Eclipse IDE and automatically creates the necessary Android files so you can concentrate on the process of application development.

Before you begin the installation of Eclipse IDE, first set the path of the JDK that you installed, as it will be required for compiling the applications. To set the JDK path on Windows, right-click on My Computer and select the Properties option. From the System Properties dialog box that appears, select the Advanced tab, followed by the Environment Variables button. A dialog box, Environment Variables, pops up. In the System variables section, double-click on the Path variable. Add the full path of the JDK (C:\Program Files\Java\jdk1.7.0_04\bin\java.exe) to the path variable and select OK to close the windows.

CREATING ANDROID VIRTUAL DEVICES

An Android Virtual Device (AVD) represents a device configuration. There are many Android devices, each with different configuration. To test whether the Android application is compatible with a set of Android devices, you can create AVDs that represent their configuration. For example, you can create an AVD that represents an Android device running version 4.1 of the SDK with a 64MB SD card. After creating AVDs, you point the emulator to each one when developing and testing the application. AVDs are the easiest way of testing the application with various configurations.

To create AVDs in Eclipse, select the Window, AVD Manager option. An Android Virtual Device Manager dialog opens, as shown in Figure. The dialog box displays a list of existing AVDs, letting you create new AVDs and manage existing AVDs. Because you haven't yet defined an AVD, an empty list is displayed.

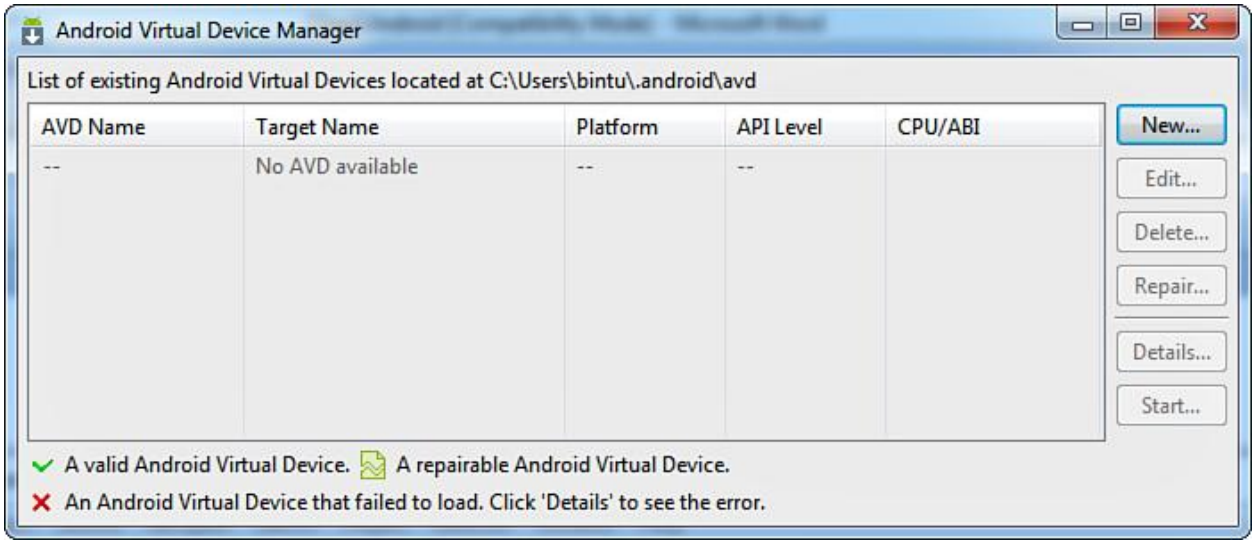


Figure: The AVD Manager dialog

Select the New button to define a new AVD. A Create new Android Virtual Device (AVD)dialog box, appears The fields are as follows:

• Name—Used to specify the name of the AVD.

• Target—Used to specify the target API level. Our application will be tested against the specified API level.

• CPU/ABI—Determines the processor that we want to emulate on our device.

• SD Card—Used for extending the storage capacity of the device. Large data files such as audio and video for which the built-in flash memory is insufficient are stored on the SD card.

• Snapshot—Enable this option to avoid booting of the emulator and start it from the last saved snapshot. Hence, this option is used to start the Android emulator quickly.

• Skin—Used for setting the screen size. Each built-in skin represents a specific screen size. You can try multiple skins to see if your application works across different devices.

• Hardware—Used to set properties representing various optional hardware that may be present in the target device.

In the AVD, set the Name of the AVD to demoAVD, choose Android 4.1—API Level 16 for the Target, set SD Card to 64 MiB, and leave the Default (WVGA800) for Skin.

In the Hardware section, three properties are already set for you depending on the selected target. The Abstracted LCD density is set to 240; the Max VM application heap size is set to 48, and the Device RAM size is set to 512.

You can select these properties and edit their values, delete them, and add new properties by selecting the New button. New properties can include Abstracted LCD density, DPad support, Accelerometer, Maximum horizontal camera pixels, Cache partition size, Audio playback support, and Track-ball support, among others.

Note

Note

The larger the allocated SD Card space, the longer it takes to create the AVD. Unless it is really required, keep the SD Card space as low as possible. I would recommend keeping this small, like 64MiB.

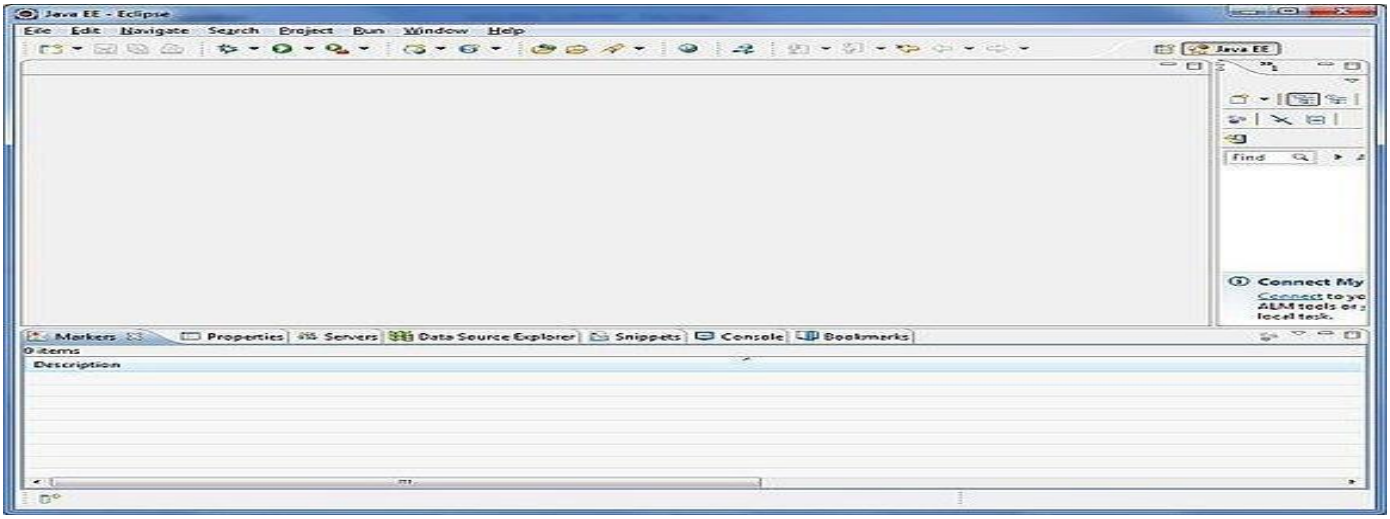
Finally, select the Create AVD button (see Figure 1.20—right) to see how to create the virtual device called demoAVD.

You now have everything ready for developing Android applications—the Android SDK, the Android platform, the Eclipse IDE, the ADT plug-in, and an AVD for testing Android applications. You can now create your first Android application.

Creating the First Android Project

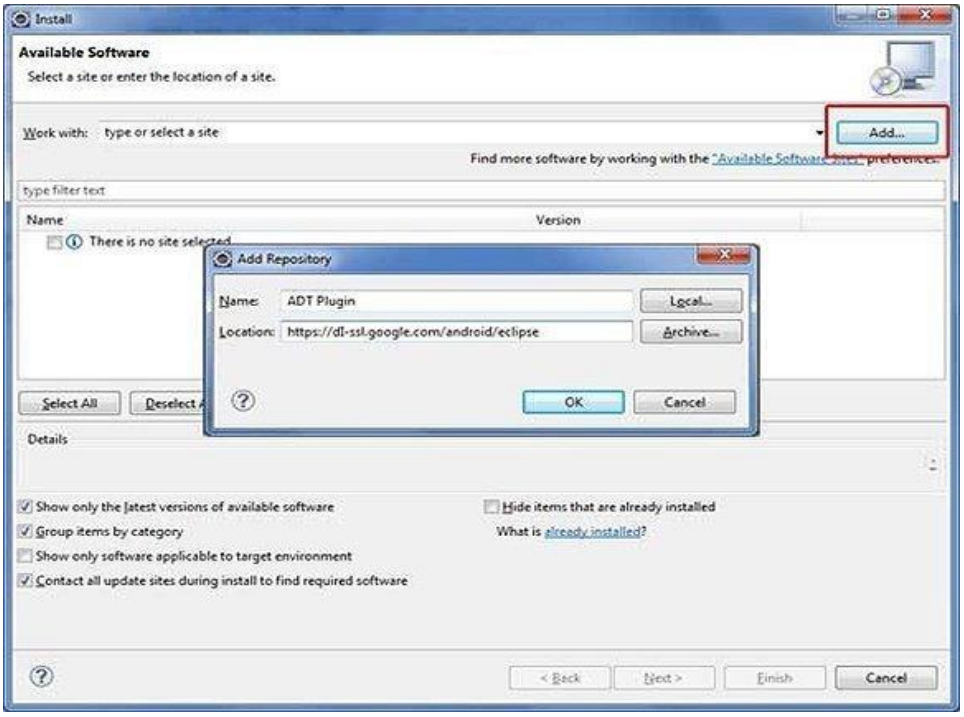
Now let's go over how to set up your first project so all you'll have left to do is write! you'll start a new Android Studio project and get to know the project workspace, including the project editor that you'll use to code the app.

Step 1: Setup Eclipse IDE:Install the latest version of Eclipse. After successful installation, it should display a

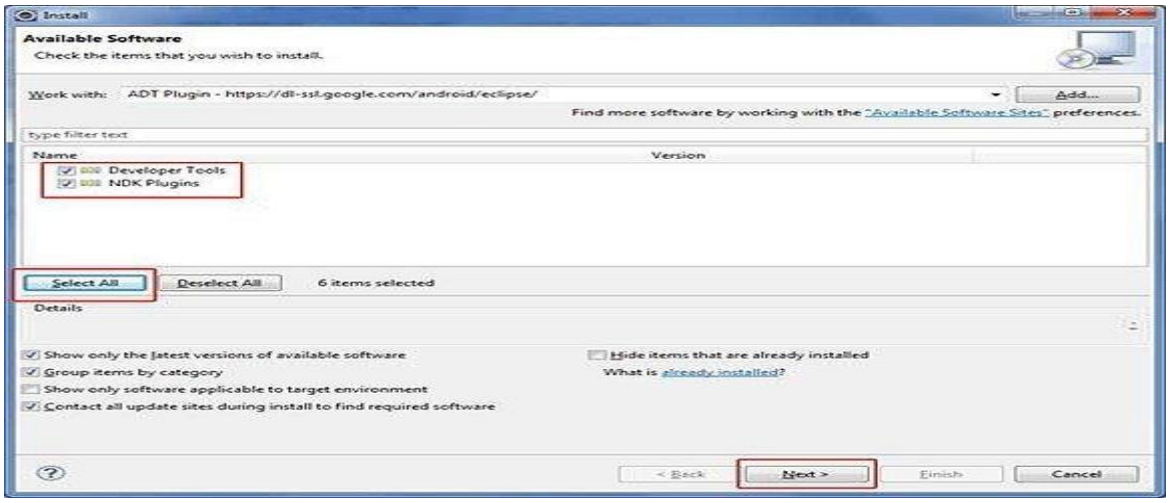


Step 2: Setup Android Development Tools (ADT) Plugin

Here you will learn to install the Android Development Tool plugin for Eclipse. To do this, you have to click on **Help > Software Updates > Install New Software**. This will display the following dialogue box.



Just click on the Add button as shown in the picture and add **https://dl-ssl.google.com/android/eclipse/** as the location. When you press OK, Eclipse will start to search for the required plug-in and finally it will list the found plug-ins.



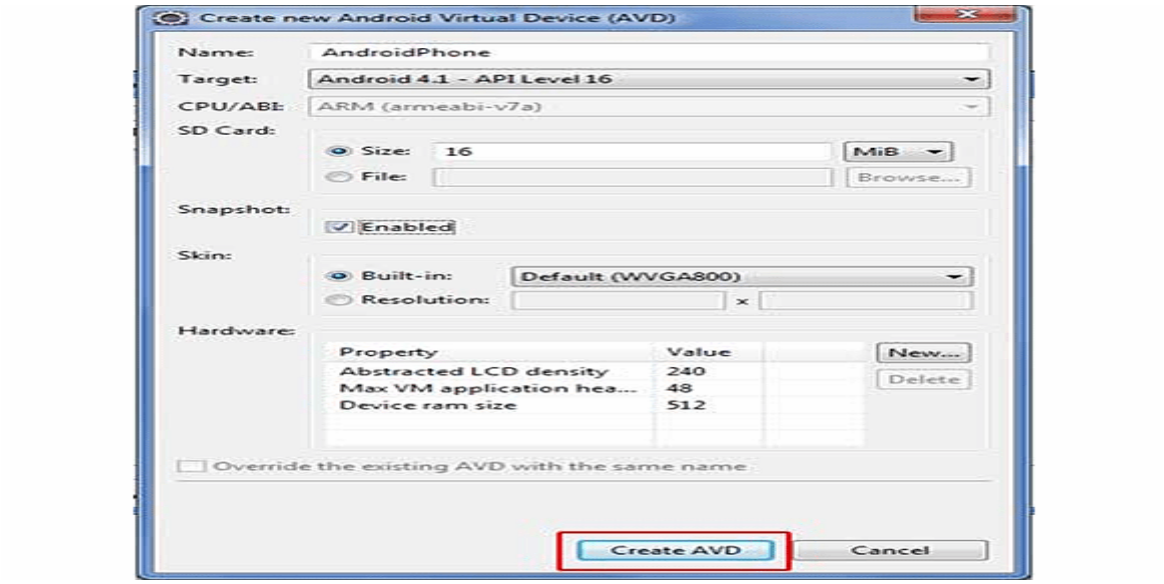
Step 3: Configuring the ADT plugin

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

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

Step 4: Create Android Virtual Device:

The last step is to create Android Virtual Device, which you will use to test your Android applications. To do this, open Eclipse and Launch Android AVD Manager from options **Window > AVD Manager** and click on **New** which will create a successful Android Virtual Device. Use the screenshot below to enter the correct values.



USING THE TEXTVIEW CONTROL

In android **ui** or **input** controls are the interactive or View components which are used to design the user interface of an application. In android we have a wide variety of UI or input controls available, those are TextView, EditText, Buttons, Checkbox, Progressbar, Spinners, etc.

In android, **TextView** is a user interface control which is used to set and display the text to the user based on our requirements. The TextView control will act as like label control and it won't allow users to edit the text. A good example of TextView control usage would be to display textual labels for other controls, like "Enter a Date:", "Enter a Name:" or "Enter a Password:".

In android, we can create a TextView control in two ways either in XML layout file or create it in Activity file programmatically.

Specific attributes of TextView controls you will want to be aware of:

- Give the TextView control a unique name using the id property.
- Set the text displayed within the TextView control using the text property; programmatically set with the setText() method.
- Set the layout height and layout width properties of the control as appropriate.
- Set any other attributes you desire to adjust the control's appearance. For example, adjust the text size, color, font or other style settings.
- By default, text contents of a TextView control are left-aligned. However, you can position the text using the gravity attribute. This setting positions your text relative to the control's overall width and height and only really makes sense to use if there is whitespace within the TextView control.
- In XML, this property would appear within your TextView control as:

  android:gravity="center"

- By default, the background of a TextView control is transparent. That is, whatever is behind the control is shown. However, you can set the background of a control explicitly, to a color resource, or a drawable (picture). In XML, this property would appear within your TextView control as:

  android:background="#0000ff"

- By default, any text contents within a TextView control is displayed as plain text. However, by setting one simple attribute called autoLink, all you can enable automatic detection of web, email, phone and address information within the text. In XML, this property would appear within your TextView control as:

  android:autoLink="all"

- You can control the color of the text within the TextView control by using the textColor attribute. This attribute can be set to a color resource, or a specific color by hex value. In XML, this property would appear within your TextView control as:

  android:textColor="#ff0000"

- You can control the style of the text (bold, italic) and font family (sans, serif, monospace) within the TextView control by using the textStyle and typeface attributes. In XML, these properties would appear within your TextView control as:

  android:textStyle="bold"
  android:typeface="monospace"

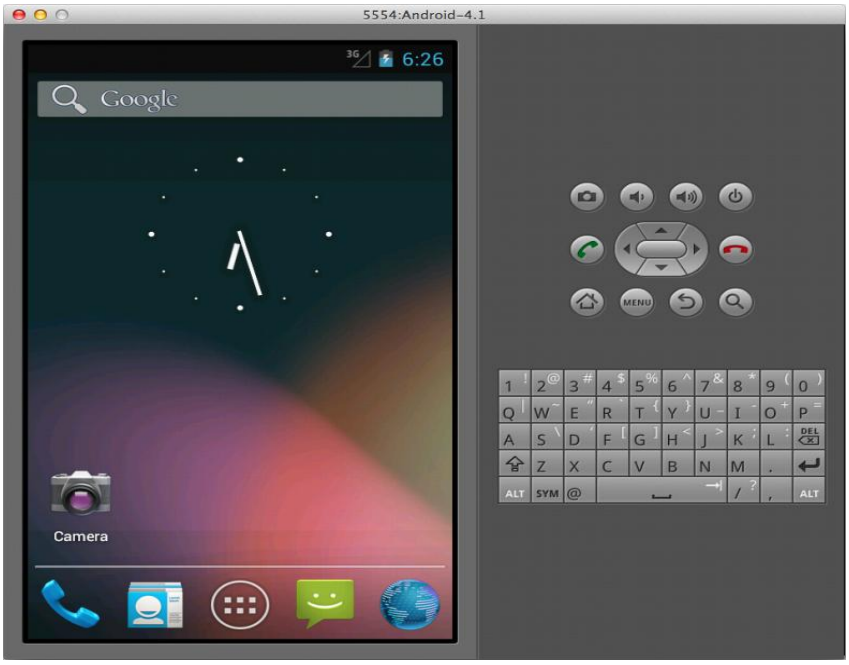
Example:

```
<TextView
   android:id="@+id/message"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   tools:context=".HelloWorldAppActivity"
   android:typeface="serif"
   android:textColor="#0F0"
   android:textSize="25dp"
   android:textStyle="italic"
   android:gravity="center_horizontal" />
```

This code makes the text of the TextView control appear in serif font, green color, 25dp size, italic, and at the horizontal center of the container

USING THE ANDROID EMULATOR

The Android emulator is used for testing and debugging applications before they are loaded onto a real handset. Android emulator is typically used for deploying apps that are developed in your IDE without actually installing it in a device. Android emulators such as Bluestacks can run android apps where in which emulators like AVD and genymotion are used to emulate an entire operating system. The Android emulator is integrated into Eclipse through the ADT plug-in.



Limitations of the Android Emulator

The Android emulator is useful to test Android applications for compatibility with devices of different configurations. But still, it is a piece of software and not an actual device and has several limitations:

- Emulators no doubt help in knowing how an application may operate within a given environment, but they still don't provide the actual environment to an application. For example, an actual device has memory, CPU, or other physical limitations that an emulator doesn't reveal.
- Emulators just simulate certain handset behavior. Features such as GPS, sensors, battery, power settings, and network connectivity can be easily simulated on a computer.
- SMS messages are also simulated and do not use a real network.
- Phone calls cannot be placed or received but are simulated.
- No support for device-attached headphones is available.
- Peripherals such as camera/video capture are not fully functional.
- No USB or Bluetooth support is available.

The emulator provides some facilities too. You can use the mouse and keyboard to interact with the emulator when it is running. For example, you can use your computer mouse to click, scroll, and drag items on the emulator. You can also use it to simulate finger touch on the soft keyboard or a physical emulator keyboard. You can use your computer keyboard to input text into UI controls and to execute specific emulator commands. Some of the most commonly used commands are

- Back [ESC button]
- Call [F3]
- End [F4]
- Volume Up [KEYPAD_PLUS, Ctrl-5]
- Volume down [KEYPAD_MINUS, Ctrl-F6]
- Switching orientations [KEYPAD_7, Ctrl-F11/KEYPAD_9, Ctrl-F12]

You can also interact with an emulator from within the DDMS tool. Eclipse IDE provides three perspectives to work with: *Java perspective, Debug perspective,* and *DDMS perspective.* The Java perspective is the default and the one with which you have been working up to now. You can switch between perspectives by choosing the appropriate icon in the top-right corner of the Eclipse environment. The three perspectives are as follows:

• **The Java perspective**—It's the default perspective in Eclipse where you spend most of the time. It shows the panes where you can write code and navigate around the project.

• **The Debug perspective**—Enables application debugging. You can set breakpoints; step through the code; view LogCat logging information, threads, and so on.

• **The Dalvik Debug Monitor Service (DDMS) perspective**—Enables you to monitor and manipulate emulator and device status. It also provides screen capture and simulates incoming phone calls, SMS sending, and GPS coordinates. To manage content in the device or emulator, you can use the ADB (Android Debug Bridge).

THE ANDROID DEBUG BRIDGE(ADB)

The Android-Debug-Bridge (abbreviated as adb) is a software-interface for the android system, which can be used to connect an android device with a computer using an USB cable or a wireless connection. It can be used to execute commands on the phone or transfer data between the device and the computer.[1]

The tool is part of the Android SDK (Android Software Development Kit) and is located in the subdirectory platform-tools. In previous versions of the SDK it was located in the subdirectory tools.

The Android Debug Bridge is a software interface between the device and the local computer, which allows the direct communication of both components. This includes the possibility to transfer files from one component to the other one, as well as executing commands from the computer on the connected device. The ADB can be used through a command line windows, terminal/shell in Linux-based systems, a command line (cmd) for Windows. The main advantage is to execute commands on the phone directly out of the computer, without any direct user interaction to the phone, which makes especially debugging a lot easier.

It is a client-server program that includes three components:

- **A client**, which sends commands. The client runs on your development machine. You can invoke a client from a command-line terminal by issuing an adb command.
- **A daemon (adbd)**, which runs commands on a device. The daemon runs as a background process on each device.
- **A server**, which manages communication between the client and the daemon. The server runs as a background process on your development machine.

When you start an adb client, the client first checks whether there is an adb server process already running. If there isn't, it starts the server process. When the server starts, it binds to local TCP port 5037 and listens for commands sent from adb clients—all adb clients use port 5037 to communicate with the adb server.

The server then sets up connections to all running devices. It locates emulators by scanning odd-numbered ports in the range 5555 to 5585, the range used by the first 16 emulators. Where the server finds an adb daemon (adbd),

it sets up a connection to that port. Note that each emulator uses a pair of sequential ports — an even-numbered port for console connections and an odd-numbered port for adb connections.

Once the server has set up connections to all devices, you can use adb commands to access those devices. Because the server manages connections to devices and handles commands from multiple adb clients, you can control any device from any client

Launching Android Applications on a Handset

To load an application onto a real handset, you need to plug a handset into your computer, using the USB data cable. You first confirm whether the configurations for debugging your application are correct and then launch the application as described here:

1. In Eclipse, choose the Run, Debug Configurations option.

2. Select the configuration HelloWorldApp_configuration, which you created for the HelloWorldApp application.

3. Select the Target tab, set the Deployment Target Selection Mode to Manual. The Manual option allows us to choose the device or AVD to connect to when using this launch configuration.

4. Apply the changes to the configuration file by clicking the Apply button.

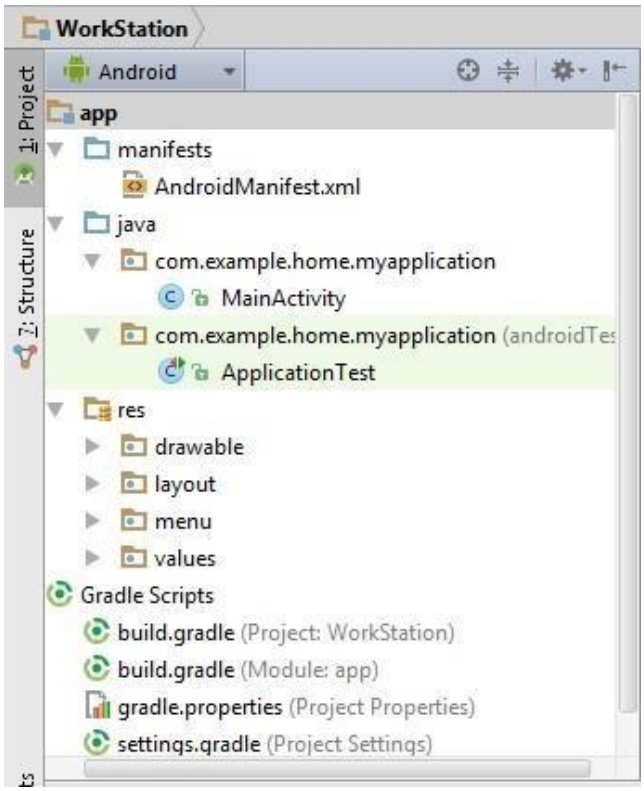
5. Plug an Android device into your computer, using a USB cable.

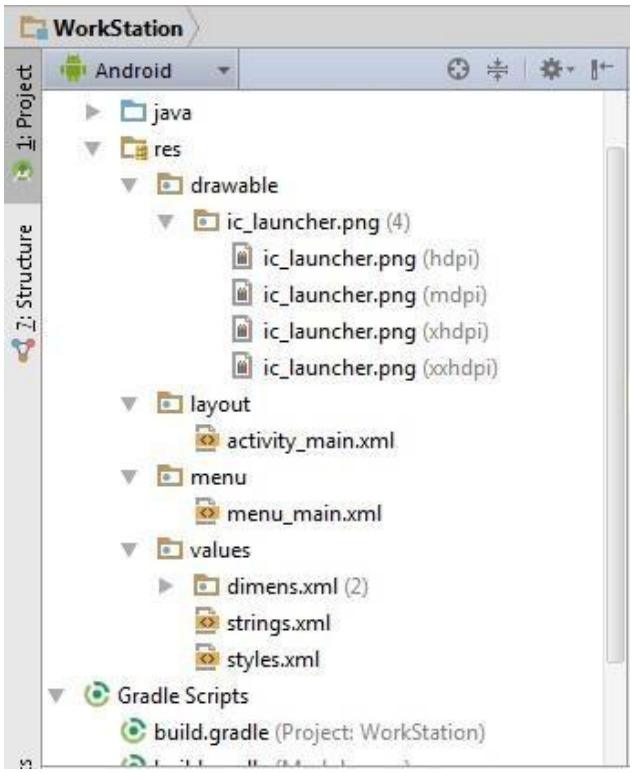
6. Select Run, Debug in Eclipse or press the F11 key. A dialog box appears, showing all available configurations for running and debugging your application. The physical device(s) connected to the computer are also listed. Double-click the running Android device. Eclipse now installs the Android application on the handset, attaches a debugger, and runs the application.

Android Studio Project Structure

Android studio shows a structured view of the project files and also provides a quick access to source files also. Android studio groups all the required source files, build files, resource files of all the modules at the top of project view.

Let`s first understand the anatomy of the Android Studio:





1) **.idea:** This folder contains the directories (subfolders) for IntelliJ IDEA settings.

2) **app:** It contains the actual application code(source files, resource file and manifest files). It further contains the following sub-directories :

a) **bulid:** This folder has sub-folders for the build-variants. The app/build/output/apk directory contains packages named app-<flavor>-<built-type>.apk. So different variant of the single app resides here.

b) **libs:** As the name suggests, this folder has all the .jar files and the library files.

3) **src:** Here you will see two sub-directories androidTest and main.

In **androidTest** the application code for testing purpose is created by android automatically.

This helps in building testing packages without modifying the building files and the application code.

main contains the all the source .java files including the stub mainactivity.java.

The **res** directory is where you will find the resources like drawable files, menu, values (style files, string values, dimension values).

<u>Sub-directories of res folder</u>

- anim/: For XML files that are compiled into animation objects.
- color/: For XML files that describe colors.

- drawable/:For image files (PNG, JPEG, or GIF), 9-Patch image files, and XML files that describe Drawable shapes or Drawable objects that contain multiple states (normal, pressed, or focused).
- mipmap/: For app launcher icons. This folder contains additional sub-folders for different screen resolutions. Once an image is imported into this folder, Studio automatically resizes the image into different resolutions and places them in the appropirate folders. This behavior allows launcher apps to pick the best resolution icon for your app to display on the home screen. To see how to import an image into the mipmap folder, please read about using 'Image asset'.
- layout/: XML files that are compiled into screen layouts (or parts of a parent layout).
- menu/: For XML files that define application menus.
- values/: For XML files that define constants that can be accessed in other XML and Java files.

R.java

R.java is an uneditable file auto-generated by Studio, whenever we modify the XML content. This file links the XML values to Java files. Whenever we need to use XML values in a Java file, we call these values using the R class.

The following images show a sample XML file, where a string variable named 'title' is assigned a value of 'Internshala - Tic Tac Toe'. This variable is then accessed in a java file using the R class.

4) **Gradle scripts:** With Android studio, Google switched to the new advanced building system, Gradle. It is a JVM based build system. If you want to make the package building task automatically, then you can write your own script in java or groovy and distribute it. Gradle allows to create different variants apk files for the same application project. We will learn about Gradles in the later chapters.

5) **External Libraries:** This is the place where all the referenced libraries and the information about the targeted platform SDK are stored.

What is Build system and Gradle

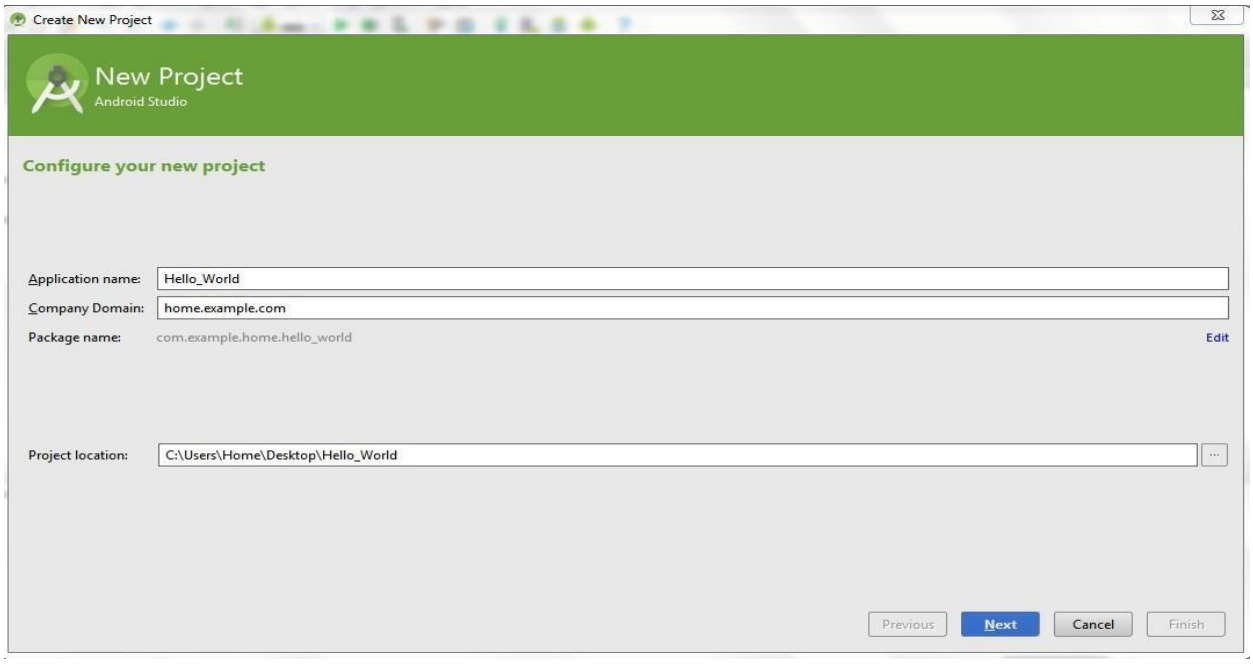
The build system is responsible to build, test an android system and also prepare the deployable files for the specified platform. In simple words, build system generates the .apk files for the application project.

With Android Studio, the advanced build system allows the developers to configure the build systems manually, create different variant APK files from single project (without modifying the execution code) and share the code and resources from other modules. Before Android Studio, Eclipse was the IDE used for android development. Eclipse kept all the .java files (in src directory) and resource files (in res directory) in the same directory. That allows the build system to group all the files into an intermediate code and finally generates .apk file.

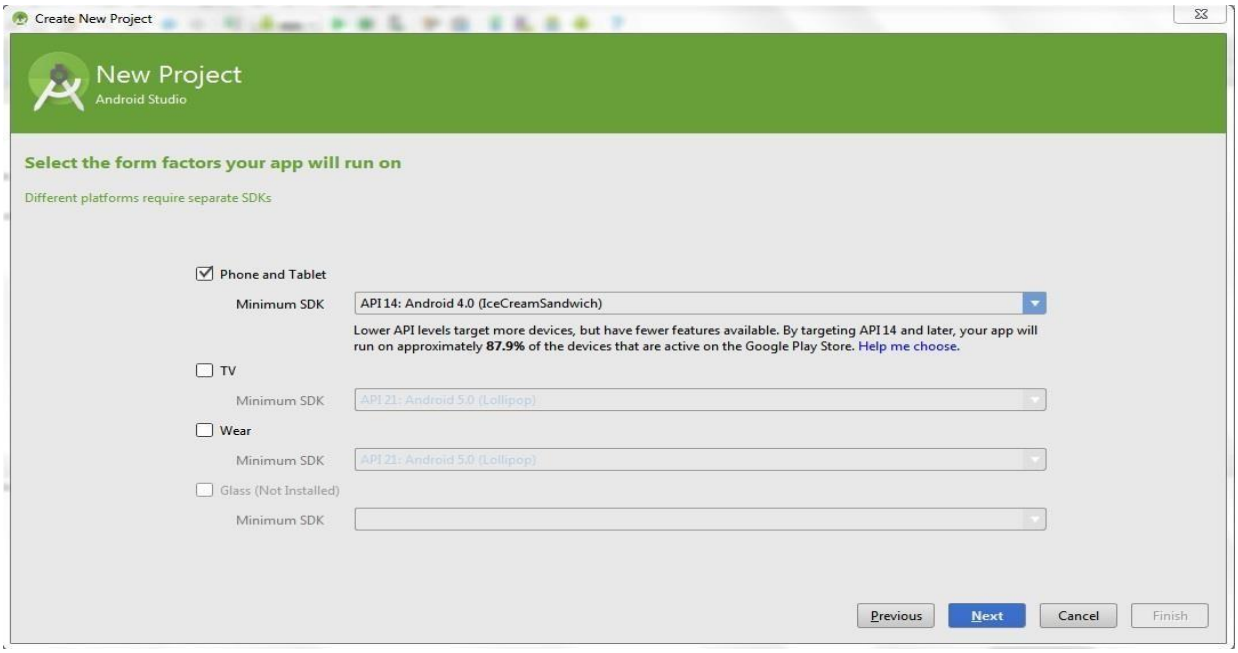
Android Studio uses Gradle as its build system. One intriguing feature that Gradle offers is that it allows you to write your own script to automate the task of building an app. As Gradle is plug-in based system, if you have your own programming language then you can write the plug-in in the script using java or groovy and share it with other developers too. Isn`t that cool?

‘Hello World’ APP

The first you will be creating will be “Hello World”! We will see how to start a project in Android Studio. First of all, launch Android Studio. Goto File-> New Project. Give the name of the application. Here we name it “Hello_World”. Hit next.



Select the category of the target devices and the desire minimum target platform of android:

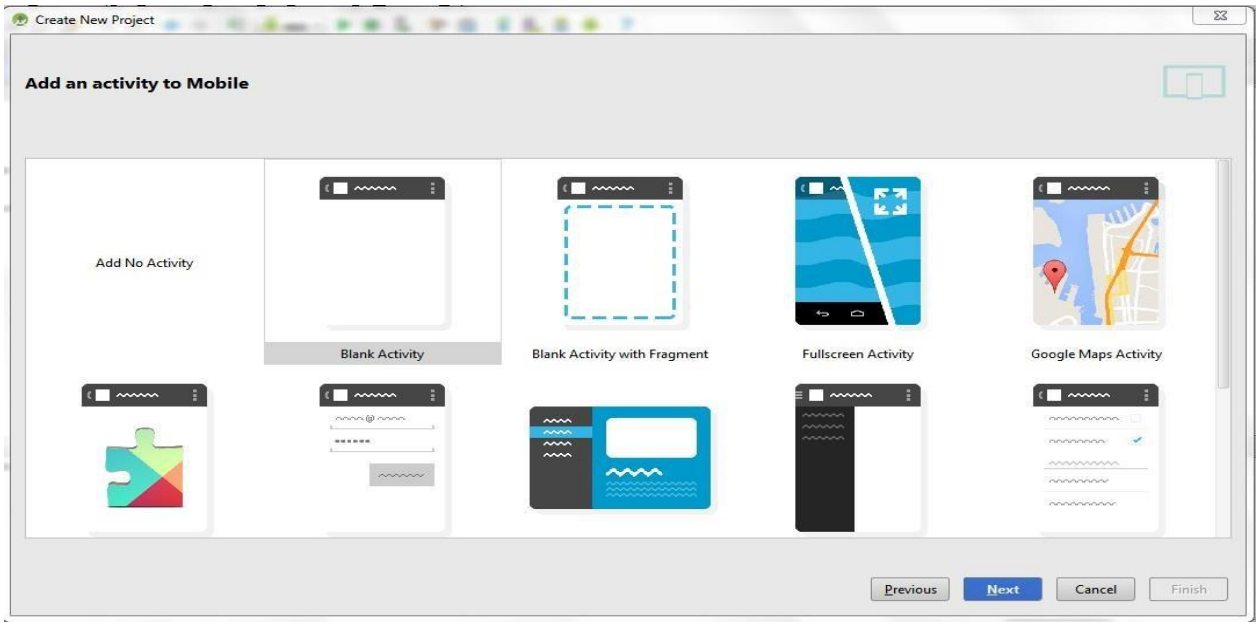


Now, select the main activity style depending upon the needs. Here for “Hello_world”,

select blank activity.

Name the main_activity files, which are the first activity to be displayed in the application. Click ‘finish’. Run the Application.

Building and Running an App



First you need to build your application project before running it on the device. Click on the “build” icon from the toolbar and then select “Make project”.

To Run an application:

Select “Run” from the toolbar and then “run app”.

You can run your app on the emulator or on a real-device from Android Studio. This is done using the debug version of the app. Run configuration defines which module will run, which activity is start, and about all the AVD settings. If you run the android application for the first time, android studio will automatically create a run configuration and choose AVD to run it. Though you can create or modify the run configuration.

Run an app on Emulator:

When you run an app on the emulator, first make sure about the AVD (Android Virtual Device). You can choose from the available AVD or create a new AVD.

Go to Tools >Android > AVD Manager. Click on ‘create virtual device’.

Choose Hardware category and configuration, Click next. Select the desired system version and create the AVD.

To run application on the desired AVD, click on the launch button.

Run an App on the real-device :

First, you have to make sure that the real-device in which you desire to run your app, is debuggable.

1) Check if the android:debuggable attribute is set to true in the build.gradle file. If not then, you cannot debug it. Make it true in case you want to run it on real-device.

2) Please enable the USB Debugging, in the device. In android 4.2 or above, the developer option can be enabled by going to Settings > About Phone and tapping Build Number 7 times. Now you can see the Developer Option in the previous screen.

Now, when you have your AVD set up or the real-devices ready, go to Run > Run (OR Run > Debug). If you don’t see your device in the AVD window, then you need to download appropriate device drivers for your device from the internet.

UNIT-2

Basic Widgets

Syllabus: Understanding the Role of Android Application Components, Understanding the Utility of Android API, Overview of the Android Project Files, Understanding Activities, Role of the Android Manifest File, Creating the User Interface, Commonly Used Layouts and Controls, Event Handling, Displaying Messages Through Toast, Creating and Starting an Activity, Using the Edit Text Control, Choosing Options with Checkbox, Choosing Mutually Exclusive Items Using Radio Buttons

Understanding the Role of Android Application Components

Almost all popular applications are interactive. These applications interact with the user, and, depending on the data supplied by the user, desired actions and/or processing are performed. The user interface controls thus play a major role in getting feedback from the user.

Application components are the ones which when combined together, offers you a brilliant Android application. So, these components exactly act as the building blocks of an Android application. The information regarding all the application components is provided in the manifest file, which is **AndroidManifest.xml**. This file will help you in understanding the use of each and every application component and how do they interact with each other. Android provides four important components to build any android application.

- Activities
- Services
- Intent and broadcast receivers
- Content Providers

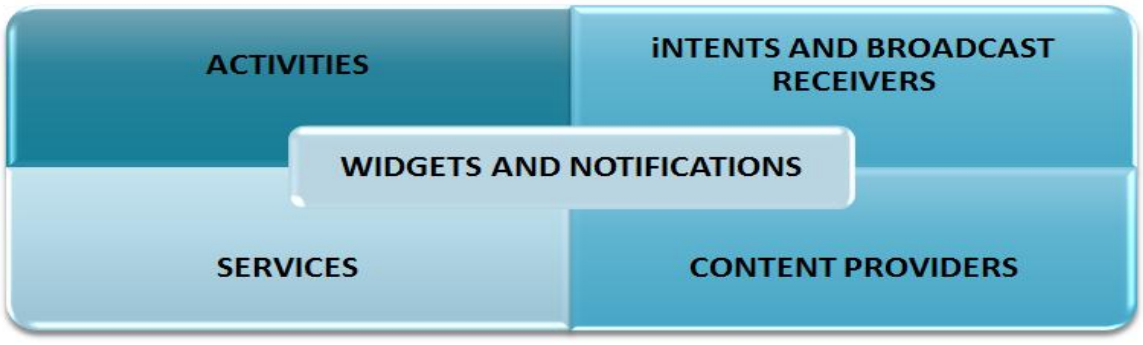


Figure Principal Ingredients of android application

1. Activities-

Activities are said to be the presentation layer of our applications. An activity is the first stepping stone is building an Android user application. The UI of our application is build around one or more extensions of the Activity class.

An activity in android is like your computer welcome screen which presents single user display. In other words, Activity in android represents single screen with a user interface. We can understand Activity in terms of web applications For example: We creates numbers of web pages to build complete web application, similarly on the other hand android application consist of several Activities to run as a complete application. There is one "main" activity. All other activities are child activities. There is a stack called back stack. Whenever, there is a new window is started, previous activity is pushed to the back stack and it is stopped until the new activity is

done. As soon as the back key of your device is pressed, new activity is popped out of stack and destroyed. Now previous activity resumes.

2. Services-

These are like invisible workers of our app. These components run at backend, updating your data sources and Activities, triggering Notification and also broadcast Intents. They also perform some tasks when applications are not active. This component is responsible for handling the time taking operations which generally runs in the background of the operating system (in this case, Android). The simple example of the service component is that when you play music on your mobile phone, you will be able to use other applications too. A service can take two forms:

1. **Started:** After a service starts, it can run indefinitely and usually performs single operation. No result is returned to user. For example, uploading a file. After the task is completed, it should terminate itself.
2. **Bound:** In this case, a component is bound to a service so that a particular task can be completed. This type of service provides a client-server like interface. Requests can be send, receive requests, and return result to the user. Inter process communication is achieved through this service.

3.Intents and Broadcast Receivers-

Android Intent is the *message* that is passed between components such as activities, content providers, broadcast receivers, services etc. It is generally used with startActivity() method to invoke activity, broadcast receivers etc. It binds individual components to each other.

Broadcast Receivers simply respond to broadcast messages from other applications or from the system itself. These messages are sometime called events or intents. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action. another example is, when your device boots up or switched on, the system generates a broadcast to all apps. There should be a procedure or should be something which can receive these broadcasts. These receptors are called broadcast receivers. There are two types of broadcasts:

1. **Normal Broadcasts:** These are asynchronous in nature. Many receivers can be activated at the same time which doesn't have any defined order. But they are very efficient.
2. **Ordered Broadcasts:** They are synchronous in nature. Broadcast received by one receiver passes it to other receivers. Broadcasts are delivered to receiver on one-to-one and sequential basis. Either receiver will pass result to another receiver or it may completely destroy the broadcast.

Content Providers-

It is used to manage and persist the application data also typically interact with SQL database. They are also responsible for sharing the data beyond the application boundaries. The Content Providers of a particular application can be configured to allow access from other applications, and the Content Providers exposed by other applications can also be configured.

With content providers we can save data in SQLite database, on the web or any other persistent storage location, where application can easily access the data. This component is useful in reading and writing private data. For example: we can read and write important reminders or notes in database(within an application).

Android Widgets and Notifications

Android App widgets are the small application views. These views can be embedded into other applications. They can receive updates on periodic basis. A widget is a quick view of your app's functionality and data. This view is accessible from home screen of your device. Now widgets are of following types:

1. Informational Widget: These Android widgets are going to display only that information to user which is important and dynamic in nature. Example the information displayed on your home screen saying time and weather condition is a widget of this type.

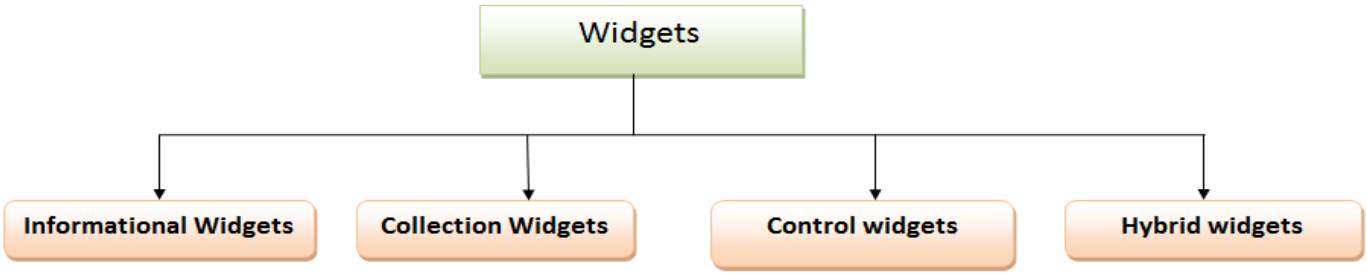


Figure Types of widgets

2.Collection Widgets: These Android widgets scroll in top-to-down direction. Collection of information of same type and then enabling user to open any one of them to full detail. Example is your e-mail app which will display all the mails in your inbox and then allow you to open any one o them.

3.Control Widgets: Displays the most frequently used functionalities which user might want to control from home screen. For example in a video app, you can pause, play, stop, go to previous track, move to next track is an example of control widget.

4.Hybrid Widgets: These Android widgets combine features of all of the above three.

Notification, as the name says keeps the user aware of events going on. User is kept informed like any news channel. For e.g, everyone of us know about facebook or whatsapp, now notification system of app is responsible for informing you about any new friend request, chat request, or a new message from say, dvs or xyz, etc.

There are a few other application components that you should be aware of. These application components include fragments, views, layouts, intents, resources, and manifest. All of these components are used for the creation of above components.

S.No.	Application Components	Description
1	Fragments	* Represents the fragments of a user interface in the Activity component
2	Views	* Includes the user interface elements like buttons, drop-down lists, etc.
3	Layouts	* Controls the screen format based on different hierarchies of the views * Takes care of the appearance of the views on the screen
4	Intents	* Wires the messages of different components together
5	Resources	* Includes external elements like drawable or editable pictures, strings, and constants

6	Manifest	* Carries the information regarding the applications * Configuration file

UNDERSTANDING THE UTILITY OF ANDROID API

API is the acronym for Application Programming Interface, which is a software intermediary that allows two applications to talk to each other. Each time you use an application, you're using an API.

When you use an application on your mobile phone, the application connects to the Internet and sends data to a server. The server then retrieves that data, interprets it, performs the necessary actions and sends it back to your phone. The application then interprets that data and presents you with the information you wanted in a readable way. This is what an API is - all of this happens via API.

In computer programming, an application programming interface (API) is a set of subroutine definitions, protocols, and tools for building an application. An API specifies how software components should interact and APIs are used when programming graphical user interface (GUI) components. In general terms, it is a set of clearly defined methods of communication between various components. A good API makes it easier to develop a computer program by providing all the building blocks, which are then put together by the programmer.

API Level is an integer value that uniquely identifies the framework API revision offered by a version of the Android platform.

The Android platform provides a framework API that applications can use to interact with the underlying Android system. The framework API consists of:

- A core set of packages and classes
- A set of XML elements and attributes for declaring a manifest file
- A set of XML elements and attributes for declaring and accessing resources
- A set of Intents
- A set of permissions that applications can request, as well as permission enforcements included in the system

The following table specifies the API Level supported by each version of the Android platform.

,,

Each successive version of the Android platform can include updates to the Android application framework API that it delivers. The framework API is specified through an integer called API level, and each Android platform version supports exactly one API level, although backward compatibility is there; that is, earlier API levels are supported. The initial release of the Android platform provided API Level 1, and subsequent releases have incremented the API level.

Updates to the framework API are designed so that the new API remains compatible with earlier versions of the API. That is, most changes in the API are additive and introduce new or replacement functionality. As parts of the API are upgraded, the older replaced parts are deprecated but are not removed, so that existing applications can still use them. In a very small number of cases, parts of the API may be modified or removed.

The framework API that an Android platform delivers is specified using an integer identifier called "API Level". Each Android platform version supports exactly one API Level, although support is implicit for all earlier

API Levels (down to API Level 1). The initial release of the Android platform provided API Level 1 and subsequent releases have incremented the API Level.

Uses of API Level in Android

The API Level identifier serves a key role in ensuring the best possible experience for users and application developers:

- It lets the Android platform describe the maximum framework API revision that it supports
- It lets applications describe the framework API revision that they require
- It lets the system negotiate the installation of applications on the user's device, such that version-incompatible applications are not installed.
- Each Android platform version stores its API Level identifier internally, in the Android system itself.

OVERVIEW OF THE ANDROID PROJECT FILES

The following files and directories are created for the Android application. The list below is just an overview of the files and directories

• **/src folder**— The folder that contains the entire Java source file of the application. The folder contains a directory structure corresponding to the package name supplied in the application. The folder contains the project's default package: com.company.basichelloworld. On expanding the package, you find the Activity of the application, the MainActivity.java file, within it.

• **/src/com.company.basichelloworld** — Refers to the package name of the application. To avoid any collision among the class names, variable names, and so on used in the application with those of other Android applications, each application has to be packaged in a unique container.

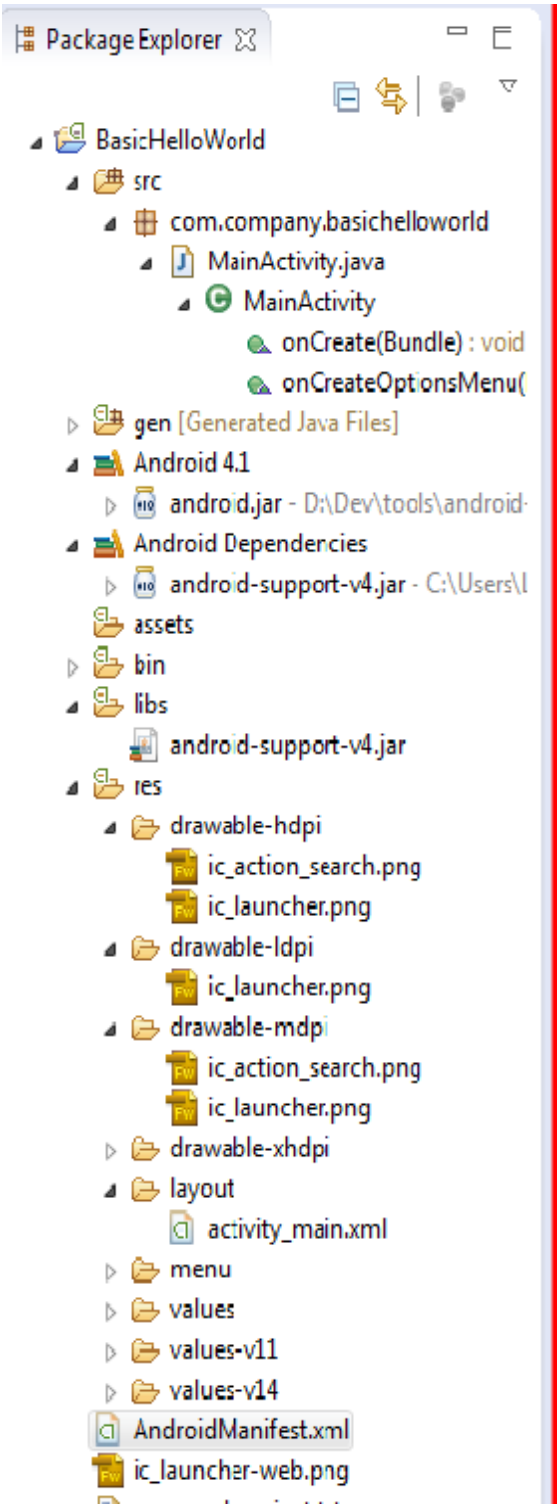
• **/src/com.company.helloworld/MainActivity.java-** The default Activity file of the application. Recall that each application has at least one Activity that acts as the main entry point to the application. The Activity file is automatically defined as the default launch Activity in the Android Manifest file.

• **/gen folder**—Contains Java files generated by ADT on compiling the application. That is, the gen folder will come into existence after compiling the application for the first time. The folder contains two files

1. **R.java**: file that contains references for all the resources defined in the res directory.
2. **BuildConfig.java:** file that is used to run code only in debug mode. It contains a DEBUG constant that helps in running debug-only functions.

• **/gen/com.company.basichelloworld/R.java**—All the layout and other resource information that is coded in the XML files is converted into Java source code and placed in the R.java file. It also means that the file contains the ID of all the resources of the application. The R.java file is compiled into the Java byte code files and then converted into .dex format. You should never edit this file by hand, as it is automatically overwritten when you add or edit resources.

• **/assets folder**—The assets folder is empty by default. It stores raw asset files that may be required in the application. It may contain fonts, external JAR files, and so on to be used in the application. The assets folder is like a resource folder where uncompiled resources of the project are kept and no IDs are generated for the resources kept here.



• **Android SDK jar file**—The jar file for the target platform

• **/bin folder**—The folder that stores the compiled version of the application.

• **/res folder**—The folder where all application resources (images, layout files, and string files) are kept. Instead of hard coding an image or string in an application, a better approach is to create a respective resource in the res folder and include its reference in the application. Each resource is assigned a unique resource ID, which automatically appears in the R.java file and thus in the R class after compilation, enabling us to refer to the resource in the code

• **/res/drawable-xhdpi, /res/drawable-hdpi, /res/drawable-mdpi, /res/drawable-ldpi**—the application's icon and graphic resources are kept in these folders. Because devices have screens of different densities, the graphics of different resolutions are kept in these folders. Usually, graphics of 320dpi, 240dpi, 160dpi, and 120dpi are used in Android applications. The application picks up the graphic from the correct folder after determining the density of the current device.

• **/res/layout**—Stores the layout file(s) in XML format.

• **/res/values**—Stores all the values resources. The values resources include many types such as string resource, dimension resource, and color resource.

• **/res/layout/activity_main.xml**—The layout file used by MainActivity to draw views on the screen. The views or controls are laid in the specified layout.

• **/res/values/strings.xml**—Contains the string resources. String resources contain the text matter to be assigned to different controls of the applications.

• **AndroidManifest.xml**—The central configuration file for the application. It is one of the most important file in the Android project structure. It contains information of the package, including components of the application such as activities, services, broadcast receivers, content providers.

Figure: Package Explorer window showing different directories, subdirectories, and files automatically created by ADT

• **project.properties**—A build file used by Eclipse and the Android ADT plug-in. It contains project settings such as the build target. You can use this file to change various properties of the project. If required, the file should not be edited directly but through editors available in Eclipse.

It's clear that the Java Activity file is the main file that drives the entire Android application.

UNDERSTANDING ACTIVITIES

An Activity is a single screen in Android. It is like a window in a desktop app, or a Frame in a Java program. An Activity allows you place all your UI components or widgets together on the screen. Every unique screen the user interacts with in an application is displayed through an Activity—one Activity for each screen. Users can interact with an application by performing different actions with the visual controls found in the Activity. A simple application may consist of just one Activity, whereas large applications contain several Activities. Each Activity of an application operates independently of the others.

If you have worked with C, C++ or Java programming language then you must have seen that your program starts from **main()** function. Very similar way, Android system initiates its program with in an **Activity** starting with a call on *onCreate()* callback method. There is a sequence of callback methods that start up an activity and a sequence of callback methods that tear down an activity.

A stack of Activities is maintained while running an application, and the Activity at the top of the stack is the one currently being displayed. When the Back button is pressed, the Activity is popped from the stack, making the previous Activity the current Activity, which therefore displays the previous screen. The transition from one Activity to another is accomplished through the use of asynchronous messages called intents. Intents can be used to pass data from one Activity to another. All of the Activities in the application must be defined in the Application's manifest file, an XML file that keeps track of all the components and permissions of the application.

Each Activity in an Android application is either a direct subclass of the Activity base class or a subclass of an Activity subclass.

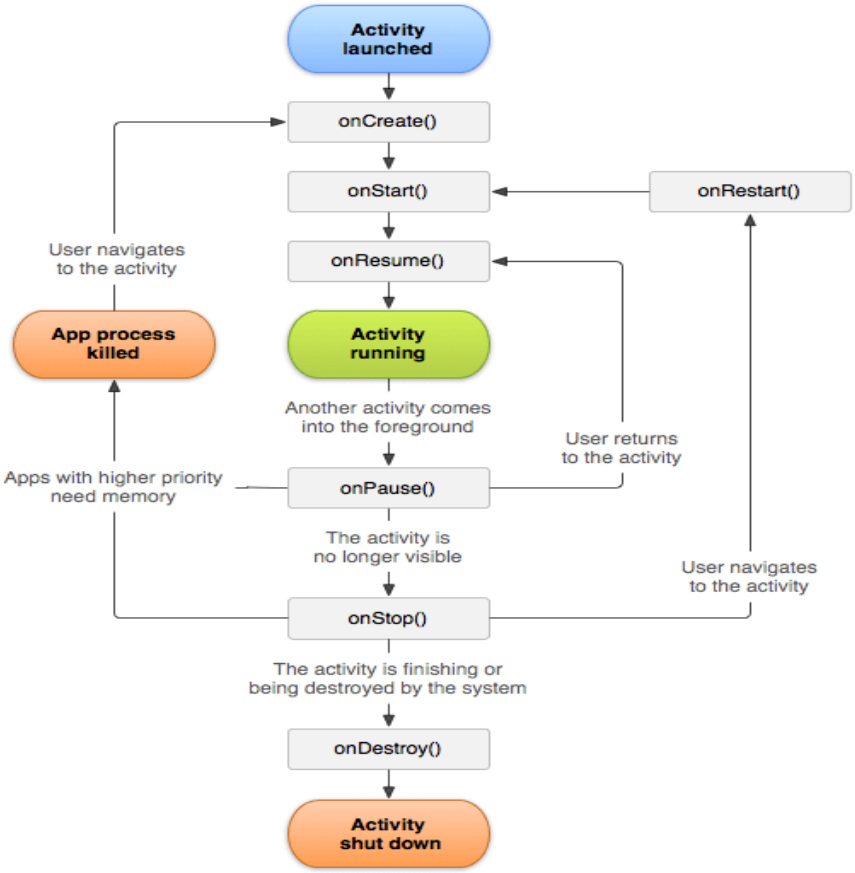
Activity life cycle

Activities have life cycles and inherit a set of life cycle methods from the Activity base class that are executed when the corresponding life cycle state of the Activity is reached.

The Android Activity life cycle defines the states or events that an Activity goes through from the time it is created until it finishes running. The Activity monitors and reacts to these events by executing methods that override the Activity class methods for each event.

Each time the Activity state changes, one of the following lifecycle methods will be called on the Activity class. The lifecycle method of Activity describes how activity will behave at different states

- **onCreate():** This is called when the Activity is first initialized. You need to implement this method in order to do any initialization specific to your Activity. Views are created, bind data, etc. A Bundle object is passed where all the previous activity states are captured.
- **onStart():** This is called the first time that the Activity is about to become visible to the user, as the Activity prepares to come to the foreground become interactive. Once this callback finishes, the onResume() method will be called.
- **onResume():** When the Activity goes into this state, it begins to interacts with the user. The Activity continues in this state till something happen to take focus from the app or Activity (such as an incoming call). When this happens, the onPause() method will be called.



- **onPause():** This method is used to pause operations that should not happen when the Activity is in paused state. A call to this method indicates that the user is leaving the app. For example, in a music player app, an incoming call will cause the app to transition into a paused state. This should mute or pause the currently playing music. When the user returns to the app, the onResume() method will be called.
- **onStop():** This method is called when the Activity is no longer visible in the app. It can happen, for example, when another Activity has been loaded and is taking the full screen of the device. When this method is called, the Activity is said to be in a stopped state. In this state, the system either calls the onRestart() to bring back interactivity with Activity. Or it calls the onDestroy() method to destroy the Activity.
- **onDestroy():** This gets called before the Activity is destroyed. The system calls this method when a user terminates the Activity, or because the system is temporarily destroying the process that contains the Activity to save space. Be sure to free up any resources your Activity has created in this method, or else your app will have a memory leak!
- **onRestart():** This gets called when an Activity restarts after it had been stopped.

All the activities of an application, permissions, and intents are defined through the XML-structured manifest file AndroidManifest.xml. For each Activity, there needs to be an entry in the AndroidManifest.xml file, where you can define labels, permissions, and so on. In fact, the manifest file is the configuration file where all the different components of the application are defined.

Example: program to demonstrate the life cycle methods of an activity

```
import android.app.Activity;
```

```
import android.os.Bundle;
import android.util.Log;

public class MainActivity extends Activity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      Log.d("lifecycle","onCreate invoked");
   }
   @Override
   protected void onStart() {
      super.onStart();
      Log.d("lifecycle","onStart invoked");
   }
   @Override
   protected void onResume() {
      super.onResume();
      Log.d("lifecycle","onResume invoked");
   }
   @Override
   protected void onPause() {
      super.onPause();
      Log.d("lifecycle","onPause invoked");
   }
   @Override
   protected void onStop() {
      super.onStop();
      Log.d("lifecycle","onStop invoked");
   }
   @Override
   protected void onRestart() {
      super.onRestart();
      Log.d("lifecycle","onRestart invoked");
   }
   @Override
   protected void onDestroy() {
      super.onDestroy();
      Log.d("lifecycle","onDestroy invoked");
   }
}
```

ROLE OF THE ANDROID MANIFEST FILE

The AndroidManifest.xml is a configuration file. It is created by ADT when creating a new Android project and is kept in the project's root directory. The manifest file presents essential information about your app to the Android system, information the system must have before it can run any of the app's code.

It's an XML file that defines the overall structure and information about the application for example, the activities, services, and intents used in the application. It also contains the permissions that the application might need to run. The manifest also defines metadata of the application such as its icon and label. Besides this, the file also contains the information required in building and packaging the application for installing and deploying it on an Android device or the emulator. To sum up, the application manifest file contains all the essential information required by the Android system to run the application.

The manifest essentially fulfills the following roles:

- It makes key information about the app available without having to read other files in the package or run the application.
- It defines which components of the application (such as activities and services) will be accessible externally to other apps, and how those components interact with other apps.
- It unambiguously declares which components and resources (icons, text strings, etc.) to use in which cases, in particular with relation to other apps.
- It lists the libraries that the application must be linked against.

For example, if your app can accept files shared from other apps, the manifest says *which* component will receive the files, and *what kind* of files it can receive.

The main advantage of this approach is, it puts all of the information in one place, which reduces the chance of mistakes or human error when a developer is making an app. It is also an extensible system, so other kinds of data can be added later without breaking existing apps. And being able to read one file to get all the app's metadata can help with performance.

Default Code in the Android Manifest File

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.androidunleashed.welcomemsg"
   android:versionCode="1"
   android:versionName="1.0" >
   <uses-sdk
      android:minSdkVersion="8"
      android:targetSdkVersion="15" />
   <application
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
         android:name=".WelcomeMsgActivity"
         android:label="@string/title_activity_welcome_msg" >
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
```

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

The <manifest> tag, is the root element of this XML document and contains several attributes:

• **android**—Identifies the Android namespace used to provide several system attributes used within the application.

• **package**—Its value is set to the application's Java package. The name of the application package acts as the unique identifier for the application in the Android device.

• **versionCode/versionName**—The versionCode attribute is used to define the current application version. The version is used internally to compare application versions. The versionName attribute is used to specify a version number that is displayed to users.

• **<uses-sdk>**—This tag is optional and is used to specify the maximum, minimum, and preferred API level required for the application to operate. Three attributes are used with this tag as follows:

• **android:minSdkVersion**—Used to specify the minimum API level required for this application to run. The default value is "1." For example, the following attribute says that the minimum API level required by the application is 15: android:minSdkVersion="15"

Hence, the application will run on API Level 15 and above, but will not run on API Level 14 or below.

• **android:targetSdkVersion**—Used to specify the preferred API level on which the application is designed to run.

• **android:maxSdkVersion**—Used to specify the maximum API level supportable by the application; that is, the application cannot run on an API level higher than the one specified in this attribute. While installing the application, the Android system checks the <uses-sdk> attributes defined in the application's manifest files and compares it with its own internal API level. The application can be installed only if

- ✓ The value of the android:minSdkVersion attribute of the application must be less than or equal to the system's API level. If the android:minSdkVersion attribute is not declared, it is assumed that the application requires API Level 1.
- ✓ The value of the android:maxSdkVersion attribute of the application (if it is declared) must be equal to, or greater than, the system's API level.

• **<application> tag**—Nested within <manifest> is <application>, which is the parent of application control tags. The icon and label attributes of the application tag refer to icon and label resources that will be displayed in Android devices to represent the application. The term "@drawable/icon" refers to the image file icon.png in the res/drawablefolder, and "@string/app_name" refers to the control resource with ID app_name in the strings.xml file that is stored in the res/values folder of the application.

• **<activity> tag**—Nested within <application> is the <activity> tag, which describes an Activity component. This tag's name attribute identifies a class that is an Activity, WelcomeMsgActivity. This name begins with a period character to show that it's relative to the com.androidunleashed.welcomemsg package. That is, the WelcomeMsgActivity is relative to <manifest>'s package value, com.androidunleashed.welcomemsg. Remember that on creating the new Android project WelcomeMsg, an Activity named WelcomeMsgActivity was automatically created for us in the specified package com.androidunleashed.welcomemsg by the ADT.

• **<intent- filter>**—Nested within <activity> is the <intent-filter>. The intents are used to interact with the applications and services. By default, the intent specifies the action as MAIN and the category as LAUNCHER; that is, it makes this Activity available from the application launcher, allowing it to launch when the application starts. Basically, the <intent-filter> tag declares the capabilities of different components through the nested <action> and <category> tags.

Besides the preceding default tags used in the manifest file, the following are some of the most commonly used tags:

• **<service> tags**—Used for declaring services. Services refer to the processes that run in the background without a user interface. They perform required processing and handle events silently without user interaction.

• **<receiver> tags**—Used for declaring broadcast receivers. Broadcast receivers are used to listen and respond to broadcast announcements. Applications initiate broadcasts for the interested applications when some event occurs that requires attention; for example, essential information or confirmation is required from the user before taking some destructive action. An application can have any number of broadcast receivers. A broadcast receiver responds by taking a specific action.

• **<provider> tags**—Used for declaring content providers. Content providers provide content, that is, data to applications. They help in handling, storing, and sharing data such as audio, images, video, and contact lists with other applications. Android ships with a number of content providers that the applications can use to access and share data with other applications.

• **<uses-permission> tags**—Used for declaring the permissions that the application needs.

CREATING THE USER INTERFACE

There are three approaches to creating user interfaces in Android.

1. You can create user interfaces entirely in Java code
2. Entirely in XML
3. Combining both methods (defining the user interface in XML and then referring and modifying it through Java code).

The third approach, the combined approach, is highly preferred.

The graphical user interface for an Android app is built using a hierarchy of View and ViewGroup objects.

- A View is an interactive UI component (or widget or control), such as button and text field. It controls a *rectangular area* on the screen. It is responsible for drawing itself and handling events (such as clicking, entering texts). Android provides many ready-to-use Views such as TextView, EditText, Button, RadioButton, etc, in package android.widget. You can also create your custom View by extending android.view.View.
- A ViewGroup is an *invisible container* used to *layout* the View components. Android provides many ready-to-use ViewGroups such as LinearLayout, RelativeLayout, TableLayout and GridLayout in package android.widget. You can also create your custom ViewGroup by extending from android.view.ViewGroup.

Views and ViewGroups are organized in a single tree structure called view-tree. You can create a view-tree either using programming codes or describing it in a XML layout file.

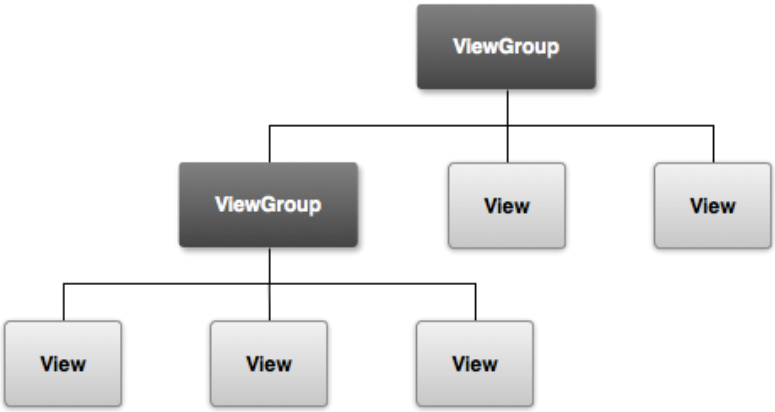


Figure: Illustration of how ViewGroup objects form branches in the layout and contain other View objects.

Example: Program to demonstrate creating a user interface

COMMONLY USED LAYOUTS AND CONTROLS

The views or the controls that we want to display in an application are arranged in an order or sequence by placing them in the desired layout. The layouts also known as Containers or ViewGroups. These are used for organizing the views or controls in the required format. Android provides the following layouts

- **LinearLayout:** In this layout, all elements are arranged in a descending column from top to bottom or left to right. Each element contains properties to specify how much of the screen space it will consume. Depending on the orientation parameter, elements are either arranged in row or column format.
- **RelativeLayout:**In this layout, each child element is laid out in relation to other child elements. That is, a child element appears in relation to the previous child. Also, the elements are laid out in relation to the parent.
- **AbsoluteLayout:** In this layout, each child is given a specific location within the bounds of the parent layout object. This layout is not suitable for devices with different screen sizes and hence is deprecated.
- **FrameLayout:** This is a layout used to display a single view. Views added to this are always placed at the top left of the layout. Any other view that is added to the FrameLayout overlaps the previous view; that is, each view stacks on top of the previous one.
- **TableLayout:** In this layout, the screen is assumed to be divided in table rows, and each of the child elements is arranged in a specific row and column.
- **GridLayout:** n this layout, child views are arranged in a grid format, that is, in the rows and columns pattern. The views can be placed at the specified row and column location. Also, more than one view can be placed at the given row and column position.

The following list highlights some of the controls commonly used in Android applications:

- **TextView:**A read-only text label. It supports multiline display, string formatting, and automatic word wrapping.
- **EditText:** An editable text box that also accepts multiline entry and word-wrapping.
- **ListView:** A ViewGroup that creates and manages a vertical list of views, displaying them as rows within the list.

• **Spinner:** A TextView and an associated list of items that allows us to select an item from the list to display in the text box.
• **Button:** A standard command button.
• **CheckBox:** A button allowing a user to select (check) or unselect (uncheck).
• **RadioButton:** A mutually exclusive button, which, when selected, unselects all other buttons in the group.

EVENT HANDLING

The action of clicking a Button, pressing the Enter key, or performing any action on any control is considered an **event**. The reaction to the event, that is, the action to be taken when the event occurs, is called **event handling**. To handle an event, you use the listeners that wait for an event occurrence. When an event occurs, the listeners detect it and direct the program to the appropriate routine.

An event listener is an interface in the View class that contains a single callback method, called an event occurrence. For example the callback method onClick() is called when the user clicks on a button. For event handling, the event listener is either implemented in the Activity class or is defined as an anonymous class. Thereafter, an instance of the implementation is passed to the respective control through the setOnClickListener() method.

Note: Click is just one type of an event.

There are three ways of event handling:
• Creating an anonymous inner class
• Implementing the OnClickListener interface
• Declaring the event handler in the XML definition of the control

Toast & Custom Toast With Example In Android Studio:

In Android, Toast is used to display information for a period of time. It contains a message to be displayed quickly and disappears after specified period of time. It does not block the user interaction. Toast is a subclass of Object class. In this we use two constants for setting the duration for the Toast. Toast notification in android always appears near the bottom of the screen. We can also create our custom toast by using custom layout(xml file).

Special Note: In Android, Toast is used when we required to notify user about an operation without expecting any user input. It displays a small popup for message and automatically fades out after timeout.

Important Methods Of Toast:

Let's we discuss some important methods of Toast that may be called in order to manage the Toast.

1. makeText(Context context, CharSequence text, int duration): This method is used to initiate the Toast. This method take three parameters:

(A) First is for the application Context,(b) Second is text message and (c)last one is duration for the Toast.

Constants of Toast: Below is the constants of Toast that are used for setting the duration for the Toast.

1. LENGTH_LONG: It is used to display the Toast for a long period of time. When we set this duration the Toast will be displayed for a long duration.

2. LENGTH_SHORT: It is used to display the Toast for short period of time. When we set this duration the Toast will be displayed for short duration.

Below we show the use of makeText() method of Toast in which we set application context, a text message and duration for the Toast.

Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast", Toast.LENGTH_LONG); // initiate the Toast with context, message and duration for the Toast

2. show(): This method is used to display the Toast on the screen. This method is display the text which we create using makeText() method of Toast.

Below we Firstly initiate the Toast and then display it using show() method.

Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android", Toast.LENGTH_LONG); // initiate the Toast with context, message and duration for the Toast

toast.show(); // display the Toast

3. setGravity(int,int,int): This method is used to set the gravity for the Toast. This method accepts three parameters: a Gravity constant, an x-position offset, and a y-position offset.

Below we Firstly initiate the Toast, set top and left gravity and then display it using show() method.

Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android", Toast.LENGTH_LONG); // initiate the Toast with context, message and duration for the Toast

toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0); // set gravity for the Toast.

toast.show(); // display the Toast

4. setText(CharSequence s): This method is used to set the text for the Toast. If we use makeText() method and then we want to change the text value for the Toast then we use this method.

Below we firstly create a new Toast using makeText() method and then set the text for the Toast.

Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android", Toast.LENGTH_LONG); // initiate the Toast with context, message and duration for the Toast

toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0); // set gravity for the Toast.

toast.setText("Changed Toast Text"); // set the text for the Toast

toast.show(); // display the Toast

5. setDuration(int duration): This method is used to set the duration for the Toast. If we use makeText() method and then we want to change the duration for the Toast then we use this method.

Below we firstly create a new Toast using makeText() method and then set the duration for the Toast.

```
Toast toast = Toast.makeText(getApplicationContext(), "Simple Toast In Android", Toast.LENGTH_LONG); // initiate the Toast with context, message and duration for the Toast

toast.setGravity(Gravity.TOP | Gravity.LEFT, 0, 0);     // set gravity for the Toast.

toast.setDuration(Toast.LENGTH_SHORT); // set the duration for the Toast.

toast.show(); // display the Toast
```

6. inflate(int, ViewGroup): This method is used to inflate the layout from the xml. In this method first parameter is the layout resource ID and the second is the root View.

Below we retrieve the Layout Inflater and then inflate the layout from the xml file.

```
// Retrieve the Layout Inflater and inflate the layout from xml

LayoutInflater inflater = getLayoutInflater();

View layout = inflater.inflate(R.layout.custom_toast_layout,

(ViewGroup) findViewById(R.id.toast_layout_root));
```

7. setView(View): This method is used to set the view for the Toast. In this method we pass the inflated layout which we inflate using inflate() method.

Below we firstly retrieve the layout inflater and then inflate the layout and finally create a new Toast and pass the inflated layout in the setView() method.

```
// Retrieve the Layout Inflater and inflate the layout from xml

LayoutInflater inflater = getLayoutInflater();

View layout = inflater.inflate(R.layout.custom_toast_layout,

(ViewGroup) findViewById(R.id.toast_layout_root));

// create a new Toast using context

Toast toast = new Toast(getApplicationContext());

toast.setDuration(Toast.LENGTH_LONG); // set the duration for the Toast

toast.setView(layout); // set the inflated layout

toast.show(); // display the custom Toast
```

Custom Toast in Android:

In Android, Sometimes simple Toast may not be satisfactory, and then we can go for customizing a Toast. For creating a custom layout, define a View layout, in XML and pass the root View object to the setView(View) method.

Steps for Implementation of Custom Toast In Android:

Step 1: Firstly Retrieve the Layout Inflater with getLayoutInflater() (or getSystemService()) and then inflate the layout from XML using inflate(int, ViewGroup). In inflate method first parameter is the layout resource ID and the second is the root View.

Step 2: Create a new Toast with Toast(Context) and set some properties of the Toast, such as the duration and gravity.

Step 3: Call setView(View) and pass the inflated layout in this method.

Step 4: Display the Toast on the screen using show() method of Toast.

In the below example we have shown the functioning of Toast and custom Toast both.

Toast And Custom Toast Example In Android Studio:

Below is the example of Toast and Custom Toast in Android. In this example we display two Button's one for Simple Toast and other for Custom Toast and perform click event on them. Whenever a user click on simple Toast Button a Toast with message "Simple Toast In Android" displayed on the screen and when a user clicks on custom toast Button a message "Custom Toast In Android" with a image displayed on the screen. For Creating a custom toast we firstly retrieve the layout inflater and then inflate the custom toast layout from the xml file. After that we get the reference of TextView and ImageView from the inflated layout and set the text and image in the TextView and ImageView. Finally we create a new Toast and pass the inflated layout in the setView() method and then display the Toast by using show() method of Toast.

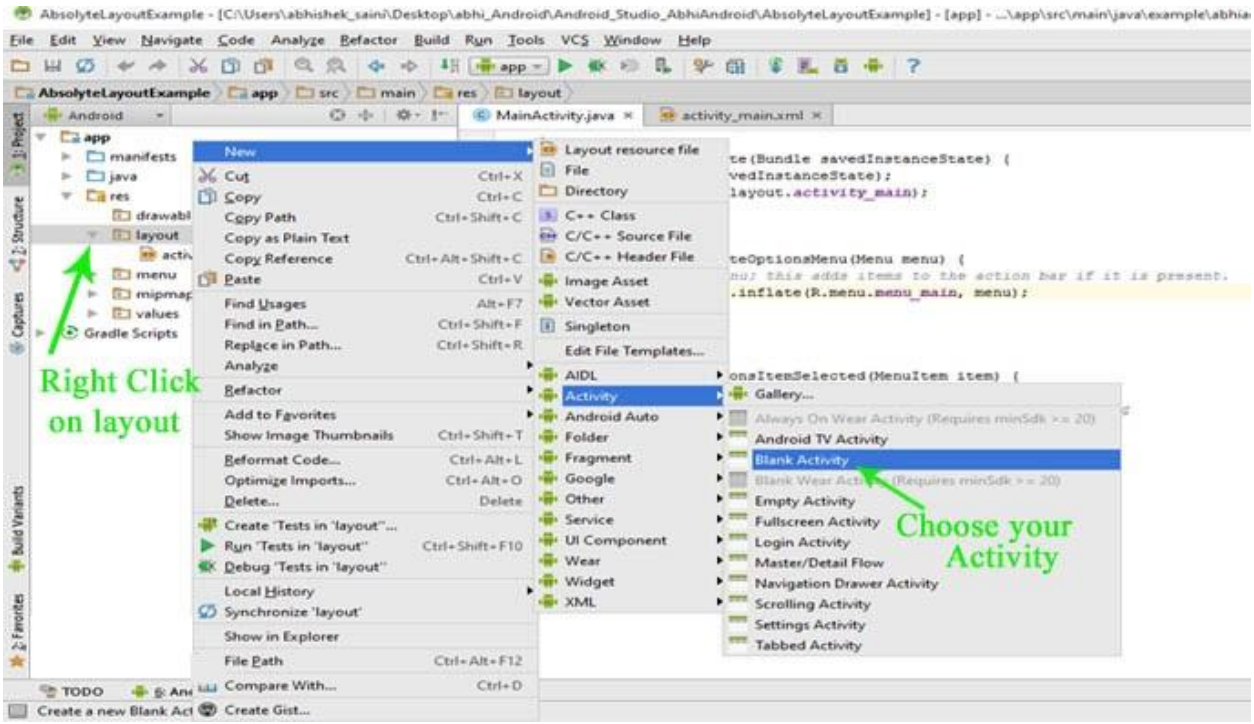
Creating and Starting an Activity:

How To Create New Activity in Android Studio

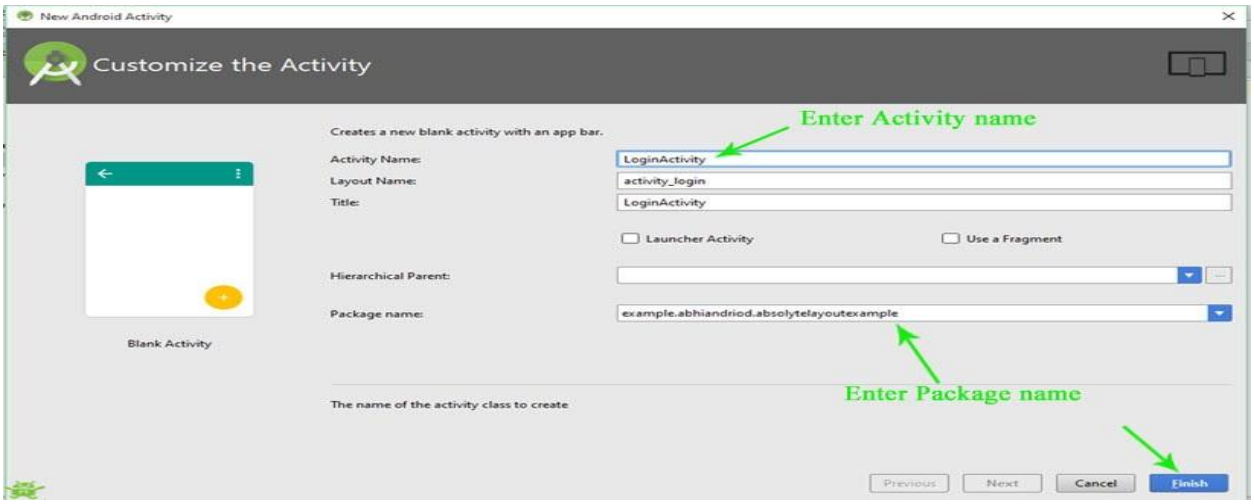
We create New Activity in Android Studio to create XML file for designing UI and java file coding. Below are the steps to create new Activity in Android Studio:

How To Create New Activity in Android Studio:

Step 1: Firstly, click on app > res > layout > Right Click on layout. After that Select New > Activity and choose your Activity as per requirement. Here we choose Blank Activity as shown in figure below.



Step 2: After that Customize the Activity in Android Studio. Enter the "Activity Name" and "Package name" in the Text box and Click on Finish button.



Step 3: After that your new Activity in Layout will be created. Your XML Code is in Text and your Design Output is in Design.

Using the Edit Text Control:

EditText Tutorial With Example In Android Studio: Input Field

In Android, EditText is a standard entry widget in android apps. It is an overlay over TextView that configures itself to be editable. EditText is a subclass of TextView with text editing operations. **We often use EditText in our applications in order to provide an input or text field, especially in forms.** The most simple example of EditText is Login or Sign-in form.

EditText Example For Input Field

Text Fields in Android Studio are basically EditText:

Important Note: An EditText is simply a thin extension of a TextView. An EditText inherits all the properties of a TextView.

EditText Code:

We can create a EditText instance by declaring it inside a layout(XML file) or by instantiating it programmatically (i.e. in Java Class).

EditText code in XML:

```
<EditText

android:id="@+id/simpleEditText"

android:layout_height="wrap_content"

android:layout_width="match_parent"/>
```

Retrieving / Getting the Value From EditText In Java Class:

Below is the example code of EditText in which we retrieve the value from a EditText in Java class. We have used this code in the example you will find at the end of this post.

```
EditText simpleEditText = (EditText) findViewById(R.id.simpleEditText);

String editTextValue = simpleEditText.getText().toString();
```

Attributes of EditText:

Now let's we discuss few attributes that helps us to configure a EditText in your xml.

1. id: id is an attribute used to uniquely identify a text EditText. Below is the example code in which we set the id of a edit text.

```
<EditText

android:id="@+id/simpleEditText"
```

```
android:layout_height="wrap_content"

android:layout_width="match_parent"/>
```

2. gravity: The gravity attribute is an optional attribute which is used to control the alignment of the text like left, right, center, top, bottom, center_vertical, center_horizontal etc.

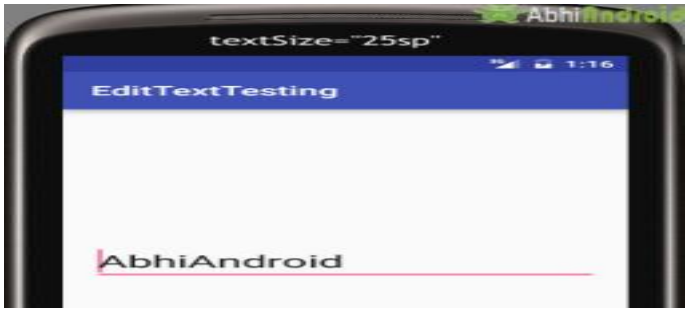
Below is the example code with explanation included in which we set the right gravity for text of a EditText.

```
<EditText

   android:id="@+id/simpleEditText"

   android:layout_width="fill_parent"

   android:layout_height="wrap_content"

   android:text="Enter Email"

   android:gravity="right"/><!--gravity of a edit text-->
```

3. text: text attribute is used to set the text in a EditText. We can set the text in xml as well as in the java class.

Below is the example code in which we set the text "Username" in a edit text.

```
<EditText

   android:id="@+id/simpleEditText"

   android:layout_width="fill_parent"

   android:layout_height="wrap_content"

   android:layout_centerInParent="true"

   android:text="Username"/><!--set text in edit text-->
```



Setting text in EditText In Java class:

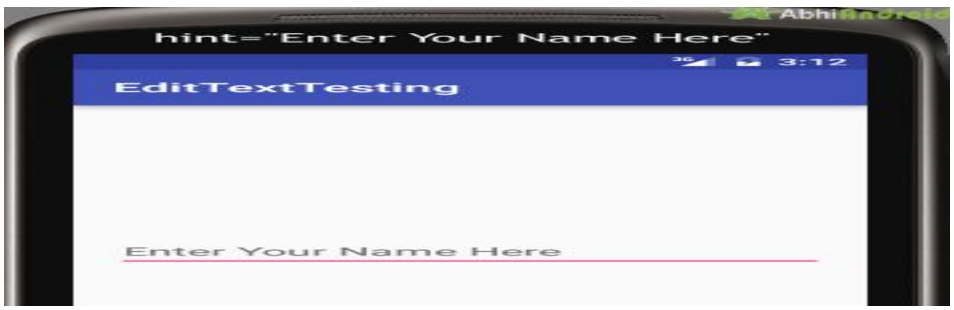
Below is the example code in which we set the text in a text view programmatically means in java class.

```
EditText editText = (EditText)findViewById(R.id.simpleEditText);

editText.setText("Username");//set the text in edit text
```

4. hint: hint is an attribute used to set the hint i.e. what you want user to enter in this edit text. Whenever user start to type in edit text the hint will automatically disappear.

Below is the example code with explanation in which we set the hint of a edit text.

```
<EditText

   android:id="@+id/simpleEditText"

   android:layout_width="fill_parent"

   android:layout_height="wrap_content"

   android:layout_centerInParent="true"

   android:hint="Enter Your Name Here" /><!--display the hint-->
```



Setting hint in EditText In Java class:

Below is the example code in which we set the text in a text view programmatically means in java class.

```
EditText editText = (EditText)findViewById(R.id.simpleEditText);

editText.setHint("Enter Your Name Here");//display the hint
```

5. textColor: textColor attribute is used to set the text color of a text edit text. Color value is in the form of “#argb”, “#rgb”, “#rrggbb”, or “#aarrggbb”.

Below is the example code with explanation included in which we set the red color for the displayed text of a edit text.

```
<EditText

   android:id="@+id/simpleEditText"

   android:layout_width="fill_parent"

   android:layout_height="wrap_content"

   android:layout_centerInParent="true"

   android:text="Password"

   android:textColor="#f00"/><!--set the red text color-->
```

Setting textColor in EditText In Java class:

Below is the example code in which we set the text color of a edit text programmatically means in java class.

```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);

simpleEditText.setTextColor(Color.RED);//set the red text color
```

6. textColorHint: textColorHint is an attribute used to set the color of displayed hint.

Below is the example code with explanation included in which we set the green color for displayed hint of a edit text.

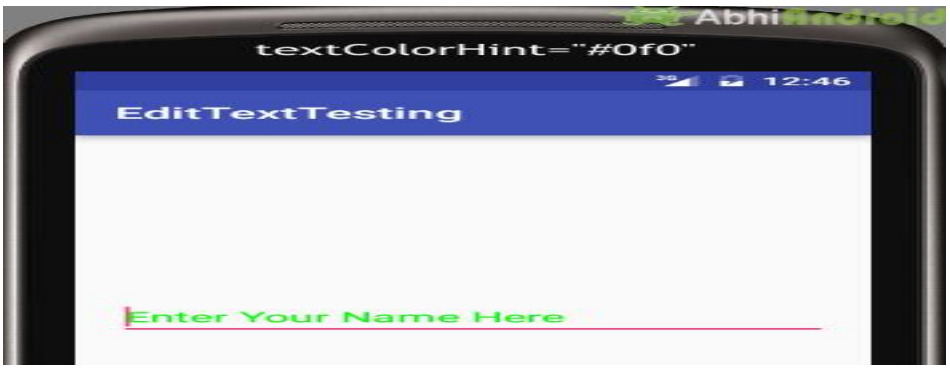
```
<EditText

   android:id="@+id/simpleEditText"
```

```
android:layout_width="fill_parent"

android:layout_height="wrap_content"

android:layout_centerInParent="true"

android:hint="Enter Your Name Here"

android:textColorHint="#0f0"/><!--set the hint color green-->
```

Setting textColorHint in EditText In Java class:

Below is the example code in which we set the hint color of a edit text programmatically means in java class.

```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);

simpleEditText.setHintTextColor(Color.green(0));//set the green hint color
```



7. textSize: textSize attribute is used to set the size of text of a edit text. We can set the text size in sp(scale independent pixel) or dp(density pixel).

Below is the example code in which we set the 25sp size for the text of a edit text.

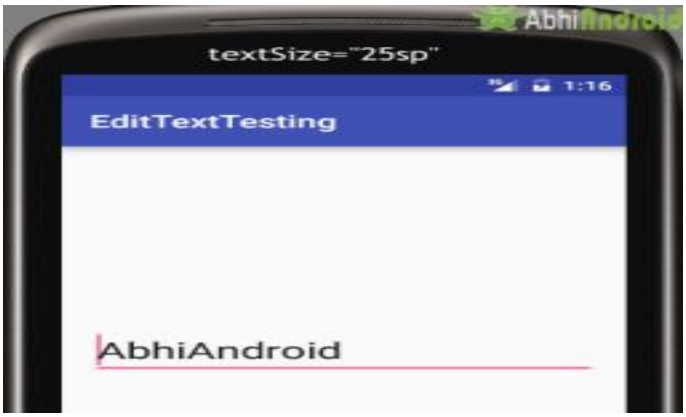
```
<EditText

android:id="@+id/simpleEditText"

android:layout_width="fill_parent"

android:layout_height="wrap_content"

android:layout_centerInParent="true"

android:text="AbhiAndroid"

android:textSize="25sp" /><!--set 25sp text size-->
```

Setting textSize in EditText in Java class:

Below is the example code in which we set the text size of a edit text programmatically means in java class.

```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);
```

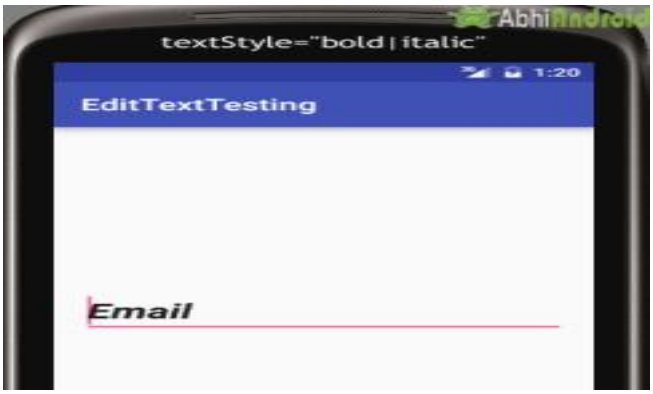
simpleEditText.setTextSize(25);//set size of text



8. textStyle: textStyle attribute is used to set the text style of a edit text. The possible text styles are bold, italic and normal. If we need to use two or more styles for a edit text then "|" operator is used for that.

Below is the example code with explanation included, in which we set the bold and italic text styles for text.

```
<EditText

   android:id="@+id/simpleEditText"

   android:layout_width="fill_parent"

   android:layout_height="wrap_content"

   android:layout_centerInParent="true"

   android:text="Email"

   android:textSize="25sp"

   android:textStyle="bold|italic"/><!--set bold and italic text style-->
```



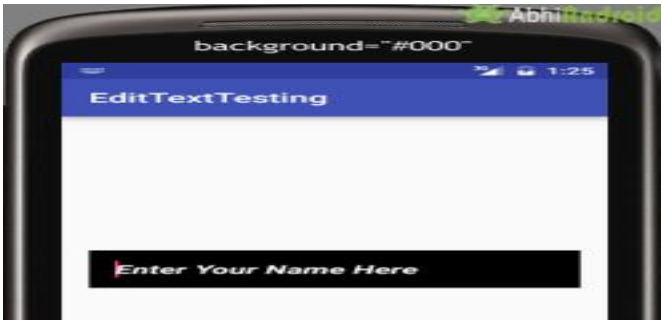
9. background: background attribute is used to set the background of a edit text. We can set a color or a drawable in the background of a edit text.

Below is the example code with explanation included in which we set the black color for the background, white color for the displayed hint and set 10dp padding from all the side's for edit text.

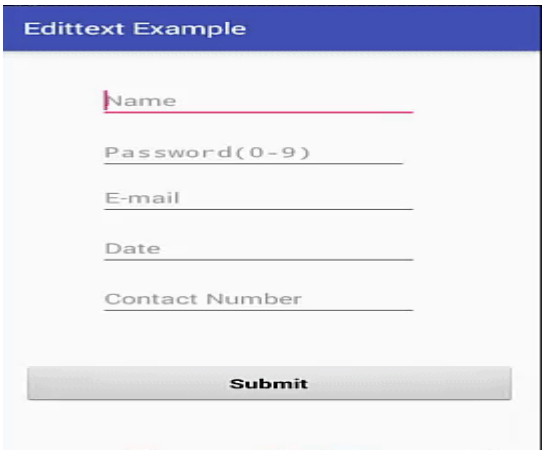
```
<EditText          android:id="@+id/simpleEditText"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:layout_centerInParent="true"
   android:hint="Enter Your Name Here"
   android:padding="15dp"
   android:textColorHint="#fff"
   android:textStyle="bold|italic"
   android:background="#000"/><!--set background color black-->
```

Setting Background in EditText In Java class: Below is the example code in which we set the background color of a edit text programmatically means in java class.

```
EditText simpleEditText=(EditText)findViewById(R.id.simpleEditText);
simpleEditText.setBackgroundColor(Color.BLACK);//set black background color
```



10. padding: padding attribute is used to set the padding from left, right, top or bottom. In above example code of background we also set the 10dp padding from all the side's of edit text.



Example I – EditText in Android Studio

Below is the example of edit text in which we get the value from multiple edittexts and on button click event the Toast will show the data defined in Edittext.

Step 1: Create a new project in Android Studio and name it EditTextExample. **Step 2:** Now Open res -> layout -> xml (or) activity_main.xml and add following code. In this code we have added multiple edittext and a button with onclick functionality.

```
<?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:id="@+id/activity_main"

   android:layout_width="match_parent"

   android:layout_height="match_parent"

   android:paddingBottom="@dimen/activity_vertical_margin"

   android:paddingLeft="@dimen/activity_horizontal_margin"

   android:paddingRight="@dimen/activity_horizontal_margin"

   android:paddingTop="@dimen/activity_vertical_margin"

   tools:context="com.example.edittextexample.MainActivity">

   <EditText

      android:id="@+id/editText1"

      android:layout_width="wrap_content"

      android:layout_height="wrap_content"

      android:layout_alignParentLeft="true"
```

```
    android:layout_alignParentStart="true"
    android:layout_alignParentTop="true"
    android:layout_marginLeft="50dp"
    android:layout_marginStart="50dp"
    android:layout_marginTop="24dp"
    android:ems="10"
    android:hint="@string/name"
    android:inputType="textPersonName"
    android:selectAllOnFocus="true" />
<EditText
    android:id="@+id/editText2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText1"
    android:layout_alignStart="@+id/editText1"
    android:layout_below="@+id/editText1"
    android:layout_marginTop="19dp"
    android:ems="10"
    android:hint="@string/password_0_9"
    android:inputType="numberPassword" />
<EditText
    android:id="@+id/editText3"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText2"
    android:layout_alignStart="@+id/editText2"
    android:layout_below="@+id/editText2"
```

```
    android:layout_marginTop="12dp"
    android:ems="10"
    android:hint="@string/e_mail"
    android:inputType="textEmailAddress" />
<EditText
    android:id="@+id/editText4"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText3"
    android:layout_alignStart="@+id/editText3"
    android:layout_below="@+id/editText3"
    android:layout_marginTop="18dp"
    android:ems="10"
    android:hint="@string/date"
    android:inputType="date" />
<EditText
    android:id="@+id/editText5"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignLeft="@+id/editText4"
    android:layout_alignStart="@+id/editText4"
    android:layout_below="@+id/editText4"
    android:layout_marginTop="18dp"
    android:ems="10"
    android:hint="@string/contact_number"
    android:inputType="phone" />
<Button
```

```
    android:id="@+id/button"
    style="@android:style/Widget.Button"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentLeft="true"
    android:layout_alignParentStart="true"
    android:layout_below="@+id/editText5"
    android:layout_marginTop="62dp"
    android:text="@string/submit"
    android:textSize="16sp"
    android:textStyle="normal|bold" />
</RelativeLayout>
```

Step 3: Now open app -> java -> package -> MainActivity.java and add the below code. In this we just fetch the text from the edittext, further with the button click event a toast will show the text fetched before.

```
package com.example.edittextexample;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
   Button submit;
   EditText name, password, email, contact, date;

   @Override
   protected void onCreate(Bundle savedInstanceState) {
```

```
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        name = (EditText) findViewById(R.id.editText1);
        password = (EditText) findViewById(R.id.editText2);
        email = (EditText) findViewById(R.id.editText3);
        date = (EditText) findViewById(R.id.editText4);
        contact = (EditText) findViewById(R.id.editText5);
        submit = (Button) findViewById(R.id.button);
        submit.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (name.getText().toString().isEmpty() || password.getText().toString().isEmpty() ||
email.getText().toString().isEmpty() || date.getText().toString().isEmpty()
                        || contact.getText().toString().isEmpty()) {
                    Toast.makeText(getApplicationContext(), "Enter the Data", Toast.LENGTH_SHORT).show();
                } else {
                    Toast.makeText(getApplicationContext(), "Name -  " + name.getText().toString() + " \n" + "Password -  " +
password.getText().toString()
                            + " \n" + "E-Mail -  " + email.getText().toString() + " \n" + "Date -  " + date.getText().toString()
                            + " \n" + "Contact -  " + contact.getText().toString(), Toast.LENGTH_SHORT).show();
                }
            }
        });
    }
}
```

Output:

Now start the AVD in Emulator and run the App. You will see screen asking you to fill the data in required fields like name, password(numeric), email, date, contact number. Enter data and click on button. You will see the data entered will be displayed as Toast on screen.

Example II – EditText in Android Studio

Below is the example of edit text in which we get the value from a edit text on button click event and then display it in a Toast. Below is the final output and code.

Step 1: Create a new project in Android Studio and name it EditTextExample.

Select File -> New -> New Project and Fill the forms and click "Finish" button.

Step 2: Now Open res -> layout -> xml (or) activity_main.xml and add following code. Here we will design one EditText for filling name and one Button which will be used to display the name entered by the user.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context=".MainActivity">
   <EditText
      android:id="@+id/simpleEditText"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:background="#F2F2F2"
      android:hint="Enter Your Name Here"
      android:padding="15dp"
      android:textColorHint="#000"
      android:textStyle="bold|italic"
      android:layout_marginTop="100dp" />
   <Button
      android:id="@+id/displayText"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerInParent="true"
      android:background="#000"
      android:padding="10dp"
```

```
    android:text="Display Text"
    android:textColor="#0f0"
    android:textStyle="bold" />
</RelativeLayout>
```

Step 3: Now open app -> java -> package -> MainActivity.java and add the below code. The explanation is included in the code itself as comment.

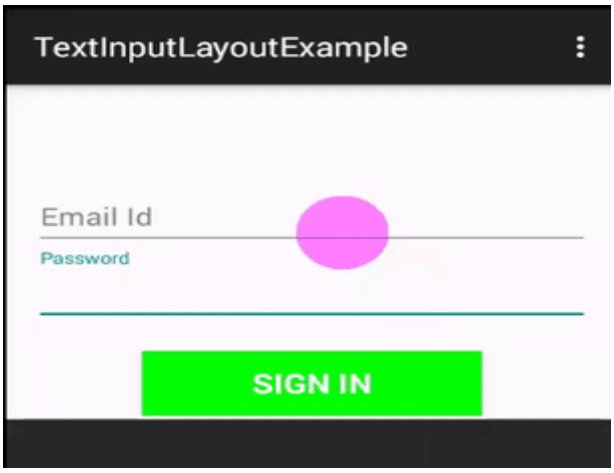
```
package example.abhiandriod.edittextexample;
import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    final EditText simpleEditText = (EditText) findViewById(R.id.simpleEditText);//get the id for edit text
    Button displayText = (Button) findViewById(R.id.displayText);//get the id for button
    displayText.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View view) {
        if (simpleEditText.getText().toString() != null)//check whether the entered text is not null
        {
```

```
            Toast.makeText(getApplicationContext(), simpleEditText.getText().toString(),
Toast.LENGTH_LONG).show();//display the text that you entered in edit text

        }

    }

});

}

}
```

Output: Now start the AVD in Emulator and run the App. You will see screen asking you to fill your name. Enter your name and click on button. You will see the name entered will be displayed as Toast on screen.

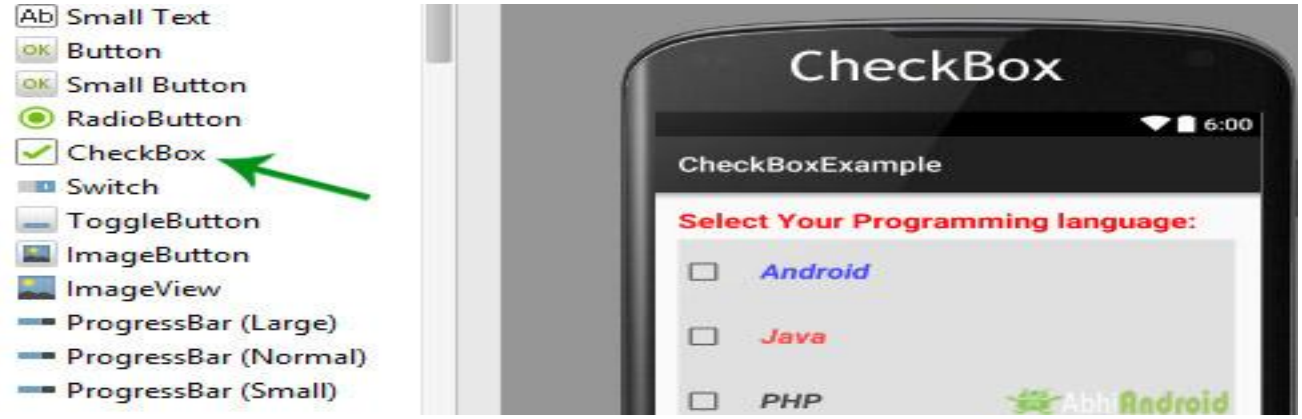
TextInputLayout / Floating Labels In EditText:

TextInputLayout is a new element introduced in Material Design Support library to display the floating labelin EditText. Read our advance Floating Labels tutorial to learn how to use it in your App.



Choosing Options with Checkbox:

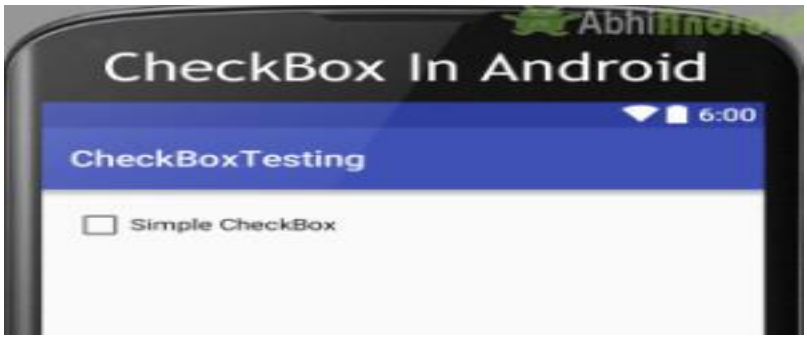
In Android, CheckBox is a type of two state button either unchecked or checked in Android. Or you can say it is a type of on/off switch that can be toggled by the users. You should use checkbox when presenting a group of selectable options to users that are not mutually exclusive. CompoundButton is the parent class of CheckBoxclass.



In android there is a lot of usage of check box. For example, to take survey in Android app we can list few options and allow user to choose using CheckBox. The user will simply checked these checkboxes rather than type their own option in EditText. Another very common use of CheckBox is as remember me option in Login form.

CheckBox code in XML:

```
<CheckBox

android:id="@+id/simpleCheckBox"

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:text="Simple CheckBox"/>
```



Important Note: You can check the current state of a check box programmatically by using isChecked() method. This method returns a Boolean value either true or false, if a check box is checked then it returns true otherwise it returns false. Below is an example code in which we checked the current state of a check box.

```
//initiate a check box

CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);

//check current state of a check box (true or false)

Boolean checkBoxState = simpleCheckBox.isChecked();
```

Attributes of CheckBox:

Now let's we discuss important attributes that helps us to configure a check box in XML file (layout).

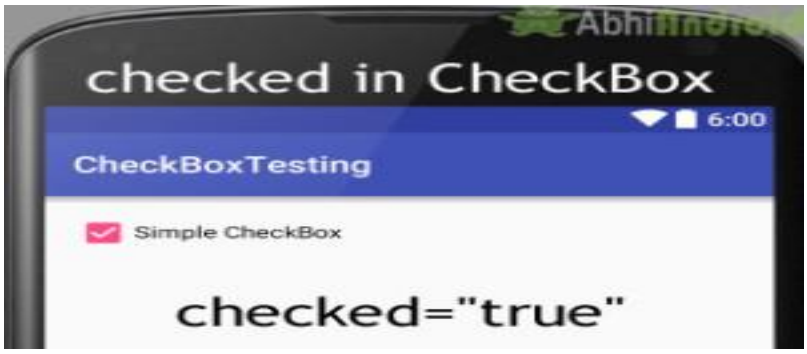
1.id: id is an attribute used to uniquely identify a check box. Below we set the id of a image button.

```
<CheckBox

android:id="@+id/simpleCheckBox"

android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:text="Simple CheckBox"/>
```

2. checked: checked is an attribute of check box used to set the current state of a check box. The value should be true or false where true shows the checked state and false shows unchecked state of a check box. The default value of checked attribute is false. We can also set the current state programmatically.

Below is an example code in which we set true value for checked attribute that sets the current state to checked.

```
<CheckBox

   android:id="@+id/simpleCheckBox"

   android:layout_width="wrap_content"

   android:layout_height="wrap_content"

   android:text="Simple CheckBox"

   android:checked="true"/> <!--set the current state of the check box-->
```



Setting Current State Of CheckBox In Java Class:

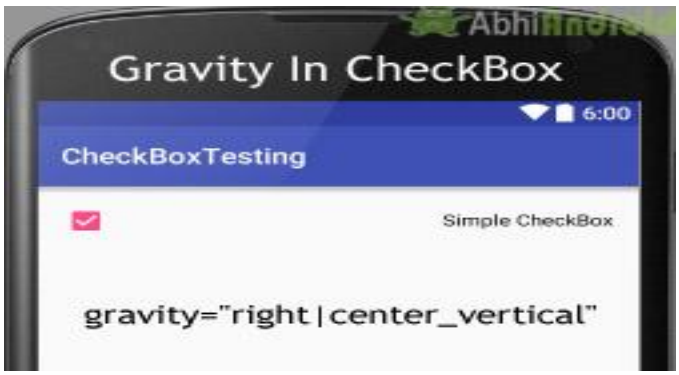
Below we set the current state of CheckBox in java class.

```
/*Add in Oncreate() funtion after setContentView()*/

// initiate a check box

CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);

// set the current state of a check box

simpleCheckBox.setChecked(true);
```

3. gravity: The gravity attribute is an optional attribute which is used to control the alignment of the text in CheckBox like left, right, center, top, bottom, center_vertical, center_horizontal etc.

Below we set the right and center_vertical gravity for the text of a check box.

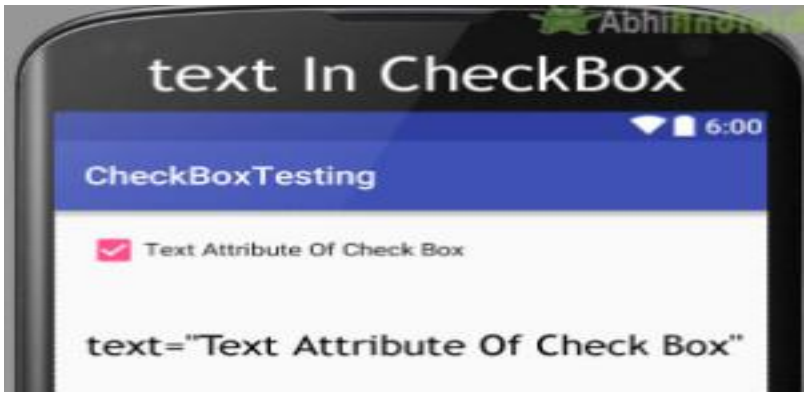
```
<CheckBox          android:id="@+id/simpleCheckBox"

   android:layout_width="fill_parent"

   android:layout_height="wrap_content"

   android:text="Simple CheckBox"

   android:checked="true"

   android:gravity="right|center_vertical"/>  <!-- gravity of the check box-->
```



4. text: text attribute is used to set the text in a check box. We can set the text in xml as well as in the javaclass.

Below is the example code with explanation included in which we set the text “Text Attribute Of Check Box” for a check box.

```
<CheckBox

   android:id="@+id/simpleCheckBox"

   android:layout_width="wrap_content"

   android:layout_height="wrap_content"

   android:checked="true"

   android:text="Text Attribute Of Check Box"/> <!--displayed text of the check box-->
```



Setting text in CheckBox In Java class:

Below is the example code in which we set the text of a check box programmatically means in java class.

```
/*Add in Oncreate() funtion after setContentView()*/

// initiate check box

CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);

// displayed text of the check box

simpleCheckBox.setText("Text Attribute Of Check Box");
```

5. textColor: textColor attribute is used to set the text color of a check box. Color value is in form of “#argb”, “#rgb”, “#rrggbb”, or “#aarrggbb”.

Below we set the red color for the displayed text of a check box.

```
<CheckBox

   android:id="@+id/simpleCheckBox"
```

```
android:layout_width="wrap_content"

android:layout_height="wrap_content"

android:text="Text is Red Color"

android:textColor="#f00"

android:checked="true"/> <!-- red color for the text of check box-->
```

Setting textColor in CheckBox In Java class:

Below we set the text color of a check box programmatically.

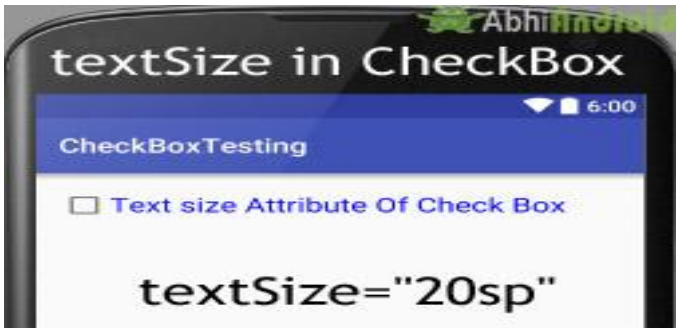
```
/*Add in Oncreate() funtion after setContentView()*/

//initiate the checkbox

CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);

//red color for displayed text

simpleCheckBox.setTextColor(Color.RED);
```

6. textSize: textSize attribute is used to set the size of text of a check box. We can set the text size in sp(scale independent pixel) or dp(density pixel).

Below is the example code in which we set the 20sp size for the text of a check box.

```
<CheckBox

   android:id="@+id/simpleCheckBox"

   android:layout_width="wrap_content"

   android:layout_height="wrap_content"

   android:text="Text size Attribute Of Check Box"

   android:textColor="#00f"

   android:checked="false"
```

```
android:textSize="20sp"/><!--set Text Size of text in CheckBox-->
```



Setting Text Size in CheckBox In Java class:

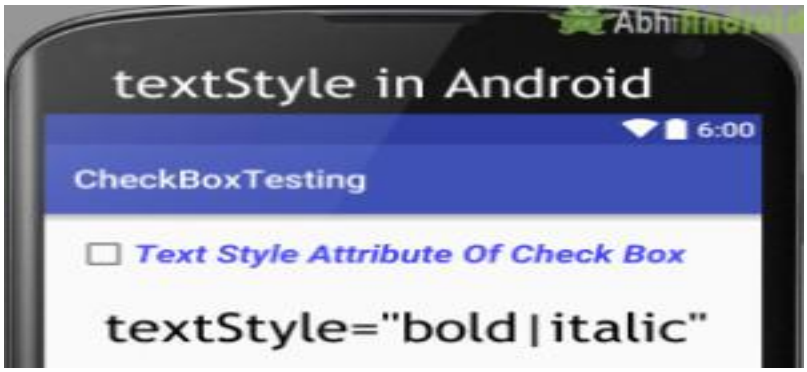
Below we set the text size of a check box in java class.

```
/*Add in Oncreate() funtion after setContentView()*/

CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);

//set 20sp displayed text size

simpleCheckBox.setTextSize(20);
```

7. textStyle: textStyle attribute is used to set the text style of the text of a check box. The possible text styles are bold, italic and normal. If we need to use two or more styles for a text view then "|" operator is used for that.

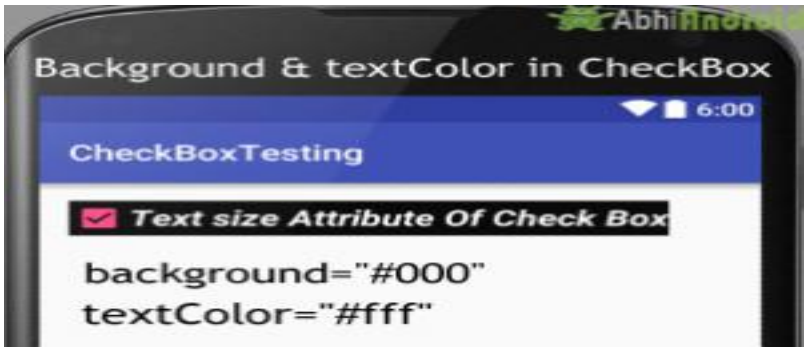
Below we set the bold and italic text styles for text of a check box.

```
<Check Box          android:id="@+id/simpleCheckBox"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:text="Text Style Attribute Of Check Box"
   android:textColor="#44f"
   android:textSize="20sp"
   android:checked="false"
   android:textStyle="bold|italic"/>
```



8. background: background attribute is used to set the background of a check box. We can set a color or a drawable in the background of a check box.
Below we set the black color for the background, white color for the displayed text of a check box.

```
<Check Box

   android:id="@+id/simpleCheckBox"

   android:layout_width="wrap_content"

   android:layout_height="wrap_content"

   android:text="Text size Attribute Of Check Box"

   android:textColor="#fff"

   android:textSize="20sp"

   android:textStyle="bold|italic"

   android:checked="true"

   android:background="#000" /><!-- black background for the background of check box-->
```



Setting Background in CheckBox In Java class:
Below is the example code in which we set the background color of a check box programmatically means in java class.

```
/*Add in Oncreate() funtion after setContentView()*/

CheckBox simpleCheckBox = (CheckBox) findViewById(R.id.simpleCheckBox);

// set background in CheckBox

simpleCheckBox.setBackgroundColor(Color.BLACK);
```

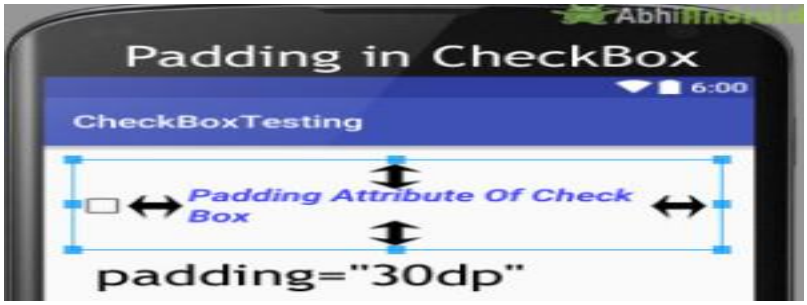
9. padding: padding attribute is used to set the padding from left, right, top or bottom.

- **paddingRight :**set the padding from the right side of the check box**.**
- **paddingLeft :**set the padding from the left side of the check box**.**
- **paddingTop :**set the padding from the top side of the check box**.**

- **paddingBottom :**set the padding from the bottom side of the check box**.**
- **Padding :**set the padding from the all side's of the check box**.**

Below is the example code of padding where we set the 30dp padding from all the side's of the check box.

```
<Check Box          android:id="@+id/simpleCheckBox"

   android:layout_width="wrap_content"

   android:layout_height="wrap_content"

   android:text="Padding Attribute Of Check Box"

   android:textColor="#44f"

   android:textSize="20sp"

   android:textStyle="bold|italic"

   android:checked="false"

   android:padding="30dp"/> <!--30dp padding from all side's-->
```



Example of CheckBox In Android Studio:

Below is the example of CheckBox in Android, in which we display five check boxes using background and other attributes we discusses earlier in this post. Every check box represents a different subject name and whenever you click on a check box then text of that checked check box is displayed in a toast. Below is the final output, code and step by step explanation of the example:

Step 1: Create a new project and name it CheckBoxExample

In this step we create a new project in android studio by filling all the necessary details of the app like app name, package name, api versions etc.

Select File -> New -> New Project and Fill the forms and click "Finish" button.

Step 2: Now Open res -> layout -> **activity_main.xml (or) main.xml** and add the following code:

In this step we open an xml file and add the code for displaying five one TextView and check boxes.

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context=".MainActivity">
   <Text View
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="Select Your Programming language: "
      android:textColor="#f00"
      android:textSize="20sp"
      android:textStyle="bold" />
   <Linear Layout
      android:id="@+id/linearLayout"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:layout_marginTop="30dp"
```

```
android:background="#e0e0e0"
android:orientation="vertical">
<Check Box
   android:id="@+id/androidCheckBox"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_centerHorizontal="true"
   android:checked="false"
   android:padding="20dp"
   android:text="@string/android"
   android:textColor="#44f"
   android:textSize="20sp"
   android:textStyle="bold|italic" />
<Check Box
   android:id="@+id/javaCheckBox"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:layout_centerHorizontal="true"
   android:checked="false"
   android:padding="20dp"
   android:text="@string/java"
   android:textColor="#f44"
   android:textSize="20sp"
   android:textStyle="bold|italic" />
<Check Box
   android:id="@+id/phpCheckBox"
   android:layout_width="wrap_content"
```

```
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:checked="false"
    android:padding="20dp"
    android:text="@string/php"
    android:textColor="#444"
    android:textSize="20sp"
    android:textStyle="bold|italic" />
<Check Box
    android:id="@+id/pythonCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:checked="false"
    android:padding="20dp"
    android:text="@string/python"
    android:textColor="#888"
    android:textSize="20sp"
    android:textStyle="bold|italic" />
<Check Box
    android:id="@+id/unityCheckBox"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_centerHorizontal="true"
    android:checked="false"
    android:padding="20dp"
    android:text="@string/unity"
```

```
        android:textColor="#101010"
        android:textSize="20sp"
        android:textStyle="bold|italic" />
    </LinearLayout>
</RelativeLayout>
```

Step 3: Now Open app -> java-> package -> **MainActivity.java**

In this step we add the code to initiate the check boxes we created. And then we perform click event on button and display the text for selected check boxes using a toast.

```
package example.abhiandriod.checkboxexample;
import android.graphics.Color;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity implements View.OnClickListener {
    CheckBox android, java, python, php, unity3D;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        // initiate views
        android = (CheckBox) findViewById(R.id.androidCheckBox);
        android.setOnClickListener(this);
        java = (CheckBox) findViewById(R.id.javaCheckBox);
```

```
    java.setOnClickListener(this);
    python = (CheckBox) findViewById(R.id.pythonCheckBox);
    python.setOnClickListener(this);
    php = (CheckBox) findViewById(R.id.phpCheckBox);
    php.setOnClickListener(this);
    unity3D = (CheckBox) findViewById(R.id.unityCheckBox);
    unity3D.setOnClickListener(this);
}
@Override
public void onClick(View view) {
    switch (view.getId()) {
        case R.id.androidCheckBox:
            if (android.isChecked())
                Toast.makeText(getApplicationContext(), "Android", Toast.LENGTH_LONG).show();
            break;
        case R.id.javaCheckBox:
            if (java.isChecked())
                Toast.makeText(getApplicationContext(), "Java", Toast.LENGTH_LONG).show();
            break;
        case R.id.phpCheckBox:
            if (php.isChecked())
                Toast.makeText(getApplicationContext(), "PHP", Toast.LENGTH_LONG).show();
            break;
        case R.id.pythonCheckBox:
            if (python.isChecked())
                Toast.makeText(getApplicationContext(), "Python", Toast.LENGTH_LONG).show();
            break;
```

```
        case R.id.unityCheckBox:
            if (unity3D.isChecked())
               Toast.makeText(getApplicationContext(), "Unity 3D", Toast.LENGTH_LONG).show();
            break;
      }
   }
}
```

Step 5: Open res ->values -> **strings.xml**

In this step we show string file which is used to store string data of an app.

```
<resources>
   <string name="app_name">CheckBoxExample</string>
   <string name="hello_world">Hello world!</string>
   <string name="action_settings">Settings</string>
   <string name="android">Android</string>
   <string name="java">Java</string>
   <string name="php">PHP</string>
   <string name="python" >Python</string>
   <string name="unity">Unity 3D</string>
</resources>
```

Output: Now start the AVD in Emulator and run the App. You will see 5 checkbox option asking you to choose your programming language. Select and the text of that particular CheckBox will appear on Screen.

Choosing Mutually Exclusive Items Using Radio Buttons:

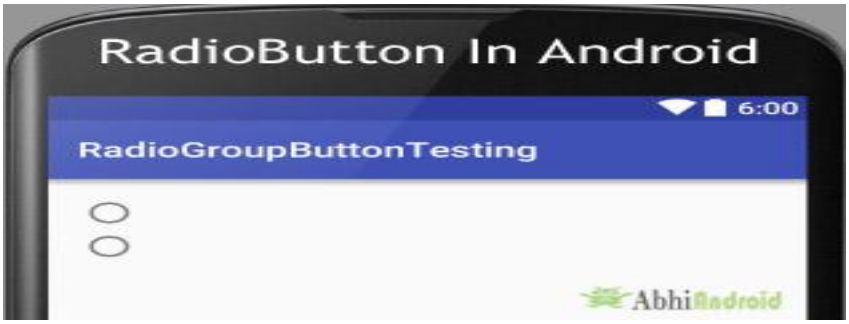
In Android, RadioButton are mainly used together in a RadioGroup. In RadioGroup checking the one radio button out of several radio button added in it will automatically unchecked all the others. It means at one time we can checked only one radio button from a group of radio buttons which belong to same radio group. The most common use of radio button is in Quiz App.

RadioButon is a two state button that can be checked or unchecked. If a radio button is unchecked then a user can check it by simply clicking on it. Once a RadiaButton is checked by user it can't be unchecked by simply pressing on the same button. It will automatically unchecked when you press any other RadioButton within same RadioGroup.

Important Note: RadioGroup is a widget used in Android for the grouping of radio buttons and provide the feature of selecting only one radio button from the set. When a user try to select any other radio button within same radio group the previously selected radio button will be automatically unchecked.

Radio Group And Radio Button code in XML:

```
<Radio Group
   android:layout_width="wrap_content"
   android:layout_height="wrap_content">
      <Radio Button
      android:id="@+id/simpleRadioButton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
      <Radio Button
      android:id="@+id/simpleRadioButton1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"/>
</Radio Group>
```



Checking Current State Of Radio Button:

You can check the current state of a radio button programmatically by using isChecked() method. This method returns a Boolean value either true or false. if it is checked then returns true otherwise returns false. Below is an example code with explanation in which we checked the current state of a radio button.

/*Add in Oncreate() funtion after setContentView()*/

RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton); // initiate a radio button

Boolean RadioButtonState = simpleRadioButton.isChecked(); // check current state of a radio button (true or false).

Table Of Contents :

Attributes of RadioButton In Android:

Now let's we discuss important attributes that helps us to create a beautiful radio button in xml file (layout).

1. id: id is an attribute used to uniquely identify a radio button.

```
<Radio Button
   android:id="@+id/simpleRadioButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"/>
```

2. checked: checked attribute in radio button is used to set the current state of a radio button. We can set it either true or false where true shows the checked state and false shows unchecked state of a radio button. As usual default value of checked attribute is false. We can also set the current state in JAVA.

Below we set true value for checked attribute which sets the current state to checked of a Button

```
<Radio Button
   android:id="@+id/simpleRadioButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="true"/> <!-- set the current state of the radio button-->
```

Setting checked State Of RadioButton In Java Class:

Below code set the current state of RadioButton to checked programmatically.

```
/*Add in Oncreate() funtion after setContentView()*/

// initiate a radio button

RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton);

// set the current state of a radio button

simpleRadioButton.setChecked(true);
```

3. text: text attribute is used to set the text in a radio button. We can set the text both ways either in XML or in JAVA class.

Below is the example code with explanation included in which we set the text “I am a radiobutton” of a radio button.

```
<RadioButton

   android:id="@+id/simpleRadioButton"

   android:layout_width="wrap_content"

   android:layout_height="wrap_content"

   android:checked="true"

   android:layout_centerHorizontal="true"

   android:text="I am a radiobutton" /> <!-- displayed text of radio button-->
```

Setting text of RadioButton In Java class:

Below we set the text of a radio button programmatically:

```
/*Add in Oncreate() funtion after setContentView()*/

RadioButton simpleRadioButton=(RadioButton) findViewById(R.id.simpleRadioButton);

simpleRadioButton.setText("I am a radiobutton"); // displayed text of radio button
```

4. gravity: The gravity attribute is an optional attribute which is used to control the alignment of text like left, right, center, top, bottom, center_vertical, center_horizontal etc.

Below is the example code with explanation included in which we set the center gravity for the text of a radio button.

```
<Radio Button
   android:id="@+id/simpleRadioButton"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
   android:checked="true"
   android:text="I am a Button"
   android:gravity="center"/> <!-- center gravity of the text-->
```

5. textColor: textColor attribute is used to set the text color of a radio button. Color value is in the form of “#argb”, “#rgb”, “#rrggbb”, or “#aarrggbb”.

Below we set the red color for the displayed text of a radio button.

```
<Radio Button
   android:id="@+id/simpleRadioButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="true"
   android:layout_centerHorizontal="true"
```

android:text="Male"

android:textColor="#f00" /><!--red color for displayed text-->

Setting text Color of Radio Button text In Java class:

Below we set the text color of a radio button programmatically.

/*Add in Oncreate() funtion after setContentView()*/

RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton);// initiate radio button

simpleRadioButton.setTextColor(Color.RED); //red color for displayed text of radio button

6. text Size: text Size attribute is used to set the size of the text of a radio button. We can set the text size in sp(scale independent pixel) or dp(density pixel).

Below we set the 25sp size for the text of a radio button.

```
<RadioButton
   android:id="@+id/simpleRadioButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="true"
   android:layout_centerHorizontal="true"
   android:text="AbhiAndroid"
   android:textColor="#f00"
   android:textSize="25sp"/> <!--setting  text size-->
```

Setting textSize Of RadioButton Text In Java class:

Below we set the text size of a radio button programmatically:

/*Add in Oncreate() funtion after setContentView()*/

RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton); // initiate radio button

simpleRadioButton.setTextSize(25); // set 25sp displayed text size of radio button

7. textStyle: textStyle attribute is used to set the text style of the text of a radio button. The possible text styles are bold, italic and normal. If we need to use two or more styles for a text view then "|" operator is used for that.

Below is the example code with explanation included in which we set the bold and italic text styles for text of a radio button.

```
<Radio Button
   android:id="@+id/simpleRadioButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="true"
   android:textSize="25sp"
   android:layout_centerHorizontal="true"
   android:text="Male"
   android:textColor="#f00"
   android:textStyle="bold|italic"/> <!-- bold and italic text style-->
```

8. background: background attribute is used to set the background of a radio button. We can set a color or a drawable in the background of a radio button.

Below we set the black color for the background and red color for the displayed text of a radio button.

```
<RadioButton

   android:id="@+id/simpleRadioButton"

   android:layout_width="wrap_content"

   android:layout_height="wrap_content"

   android:checked="true"

   android:textSize="25sp"

   android:textStyle="bold|italic"
```

```
android:padding="20dp"
android:layout_centerHorizontal="true"
android:text="Male"
android:textColor="#f00"
android:background="#000"/> <!-- black background for radio button-->
```

Setting Background Of RadioButton In Java class:

Below we set the background color of a radio button programmatically.

```
/*Add in Oncreate() funtion after setContentView()*/
RadioButton simpleRadioButton = (RadioButton) findViewById(R.id.simpleRadioButton);
simpleRadioButton.setBackgroundColor(Color.BLACK);
```

9. padding: padding attribute is used to set the padding from left, right, top or bottom.

paddingRight: set the padding from the right side of the radio button.

paddingLeft : set the padding from the left side of the radio button.

paddingTop : set the padding from the top side of the radio button.

paddingBottom: set the padding from the bottom side of the radio button.

Padding: set the padding from the all side's of the radio button.

Below we set padding attribute of 20dp padding from all the side's of the radio button.

```
<Radio Button
   android:id="@+id/simpleRadioButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="true"
   android:textSize="25sp"
   android:textStyle="bold|italic"
```

```
android:layout_centerHorizontal="true"

android:text="AbhiAndroid"

android:textColor="#f00"

android:padding="40dp"/> <!--40dp padding from all the sides of radio button-->
```

10. drawableBottom, drawableTop, drawableLeft And drawableRight: These attribute draw the drawable to the below of the text of RadioButton.

Below we set the icon to the right of the text of a RadioButton.

```
<RadioButton
   android:id="@+id/simpleRadioButton"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"
   android:checked="true"
   android:textSize="25sp"
   android:padding="20dp"
   android:layout_centerHorizontal="true"
   android:text="AbhiAndroid"
   android:textColor="#f00"
   android:drawableRight="@drawable/ic_launcher" /> <!-- drawable icon at the right of radio button-->
```

Example Of RadioButton And RadioGroup in Android Studio:

Below is the example of Radiobutton in Android where we display five radio buttons with background and other attributes. The radio buttons are used to select your favorite WWE superstar with one “submit” button. Below is the final output, download code and step by step explanation of tutorial:

Step 1: Create a new project and name it Radio Button Example
Select File -> New -> New Project and Fill the forms and click "Finish" button.
Step 2: Open res -> layout -> activity_main.xml (or) main.xml and add following code:
In this step we open an xml file and add the code for displaying 5 Radio Button and one normal button.

```
<Linear Layout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:orientation="vertical"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context=".MainActivity">
   <Linear Layout
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
      android:background="#e0e0e0"
      android:orientation="vertical">

      <Text View
         android:layout_width="wrap_content"
         android:layout_height="wrap_content"
         android:text="Select your favourite wwe SuperStar :: "
         android:textColor="#000"
         android:textSize="20sp"
         android:textStyle="bold" />
      <Radio Group
         android:layout_width="wrap_content"
         android:layout_height="wrap_content">
         <Radio Button
            android:id="@+id/johnCena"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginLeft="20dp"
```

```
        android:layout_marginTop="10dp"
        android:checked="true"
        android:text="@string/johnCena"
        android:textColor="#154"
        android:textSize="20sp"
        android:textStyle="bold" />
    <Radio Button
        android:id="@+id/randyOrton"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="20dp"
        android:layout_marginTop="10dp"
        android:checked="false"
        android:text="@string/randyOrton"
        android:textColor="#154"
        android:textSize="20sp"
        android:textStyle="bold" />
    <Radio Button
        android:id="@+id/romanReigns"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="20dp"
        android:layout_marginTop="10dp"
        android:checked="false"
        android:text="@string/romanReigns"
        android:textColor="#154"
        android:textSize="20sp"
        android:textStyle="bold" />
    <Radio Button
        android:id="@+id/goldBerg"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="20dp"
        android:layout_marginTop="10dp"
        android:checked="false"
        android:text="@string/goldBerg"
        android:textColor="#154"
        android:textSize="20sp"
        android:textStyle="bold" />
    <Radio Button
        android:id="@+id/sheamus"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginLeft="20dp"
        android:layout_marginTop="10dp"
        android:checked="false"
        android:text="@string/sheamus"
        android:textColor="#154"
        android:textSize="20sp"
        android:textStyle="bold" />
</RadioGroup>
```

```
    <Button
      android:id="@+id/submitButton"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_gravity="center"
      android:layout_margin="20dp"
      android:background="#0f0"
      android:padding="10dp"
      android:text="Submit"
      android:textColor="#fff"
      android:textSize="20sp"
      android:textStyle="bold" />
  </Linear Layout>
</LinearLayout>
```

Step 3: Open src -> package -> MainActivity.java

In this step we open MainActivity and add the code to initiate the RadioButton and normal button. We also perform click event on button and display the selected superstar's name by using a Toast.

```
package example.gb.radiobuttonexample;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.RadioButton;
import android.widget.Toast;
public class MainActivity extends AppCompatActivity {
  RadioButton johnCena, randyOrton, goldBerg, romanReigns, sheamus;
  String selectedSuperStar;
  Button submit;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    johnCena = (RadioButton) findViewById(R.id.johnCena);
    randyOrton = (RadioButton) findViewById(R.id.randyOrton);
    goldBerg = (RadioButton) findViewById(R.id.goldBerg);
    romanReigns = (RadioButton) findViewById(R.id.romanReigns);
    sheamus = (RadioButton) findViewById(R.id.sheamus);
    submit = (Button) findViewById(R.id.submitButton);
    submit.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        if (randyOrton.isChecked()) {
          selectedSuperStar = randyOrton.getText().toString();
        } else if (sheamus.isChecked()) {
          selectedSuperStar = sheamus.getText().toString();
        } else if (johnCena.isChecked()) {
          selectedSuperStar = johnCena.getText().toString();
        } else if (romanReigns.isChecked()) {
          selectedSuperStar = romanReigns.getText().toString();
        } else if (goldBerg.isChecked()) {
          selectedSuperStar = goldBerg.getText().toString();
```

```
            }
            Toast.makeText(getApplicationContext(), selectedSuperStar, Toast.LENGTH_LONG).show(); // print the value of selected super star
        }
    });
  }
}
```

Step 4: Open res -> values -> strings.xml

In this step we open String file which is used to store string data of the app.

```
<resources>
   <string name="app_name">RadioButtonExample</string>
   <string name="hello_world">Hello world!</string>
   <string name="action_settings">Settings</string>
   <string name="randyOrton">Randy Orton</string>
   <string name="johnCena">John Cena</string>
   <string name="romanReigns">Roman Reigns</string>
   <string name="goldBerg">Gold Berg</string>
   <string name="sheamus">Sheamus</string>
</resources>
```

Run The App:

Now run the App in Emulator and you will see 5 RadioButton in RadioGroup listing WWE superstar name. Now choose your favorite one and click on Submit Button. The name will be displayed on Screen.

CHAPTER 3

Laying Out Controls in Containers

A container is a view used to contain other views. Android offers a collection of view classes that act as containers for views. These container classes are called layouts, and as the name suggests, they decide the organization, size, and position of their children views.

Let's start the chapter with an introduction to different layouts used in Android applications.

Introduction to Layouts

Layouts are basically containers for other items known as `Views`, which are displayed on the screen. Layouts help manage and arrange views as well. Layouts are defined in the form of XML files that cannot be changed by our code during runtime.

Table 3.1 shows the layout managers provided by the Android SDK.

TABLE 3.1 Android Layout Managers

Layout Manager	Description
LinearLayout	Organizes its children either horizontally or vertically
RelativeLayout	Organizes its children relative to one another or to the parent
AbsoluteLayout	Each child control is given a specific location within the bounds of the container

IN THIS CHAPTER

Layout Manager	Description
FrameLayout	Displays a single view; that is, the next view replaces the previous view and hence is used to dynamically change the children in the layout
TableLayout	Organizes its children in tabular form
GridLayout	Organizes its children in grid format

The containers or layouts listed in Table 3.1 are also known as `ViewGroups` as one or more `Views` are grouped and arranged in a desired manner through them. Besides the `ViewGroups` shown here Android supports one more `ViewGroup` known as ScrollView, which is discussed in Chapter 4, "Utilizing Resources and Media."

LinearLayout

The LinearLayout is the most basic layout, and it arranges its elements sequentially, either horizontally or vertically. To arrange controls within a linear layout, the following attributes are used:

- **`android:orientation`**—Used for arranging the controls in the container in horizontal or vertical order
- **`android:layout_width`**—Used for defining the width of a control
- **`android:layout_height`**—Used for defining the height of a control
- **`android:padding`**—Used for increasing the whitespace between the boundaries of the control and its actual content
- **`android:layout_weight`**—Used for shrinking or expanding the size of the control to consume the extra space relative to the other controls in the container
- **`android:gravity`**—Used for aligning content within a control
- **`android:layout_gravity`**—Used for aligning the control within the container

Applying the `orientation` Attribute

The `orientation` attribute is used to arrange its children either in horizontal or vertical order. The valid values for this attribute are `horizontal` and `vertical`. If the value of the `android:orientation` attribute is set to `vertical`, the children in the linear layout are arranged in a column layout, one below the other. Similarly, if the value of the `android:orientation` attribute is set to `horizontal`, the controls in the linear layout are arranged in a row format, side by side. The orientation can be modified at runtime through the `setOrientation()` method. That is, by supplying the values `HORIZONTAL` or `VERTICAL` to the `setOrientation()` method, we can arrange the children of the LinearLayout in row or column format, respectively.

Applying the `height` and `width` Attributes

The default height and width of a control are decided on the basis of the text or content that is displayed through it. To specify a certain height and width to the control, we use the `android:layout_width` and `android:layout_height` attributes. We can specify the values for the `height` and `width` attributes in the following three ways:

- By supplying specific dimension values for the control in terms of `px` (pixels), `dip/dp` (device independent pixels), `sp` (scaled pixels), `pts` (points), `in` (inches), and `mm` (millimeters). For example, the `android:layout_width="20px"` attribute sets the width of the control to 20 pixels.
- By providing the value as `wrap_content`. When assigned to the control's height or width, this attribute resizes the control to expand to fit its contents. For example, when this value is applied to the width of the `TextView`, it expands so that its complete text is visible.
- By providing the value as `match_parent`. When assigned to the control's height or width, this attribute forces the size of the control to expand to fill up all the available space of the enclosing container.

NOTE

For layout elements, the value `wrap_content` resizes the layout to fit the controls added as its children. The value `match_parent` makes the layout expand to take up all the space in the parent layout.

Applying the `padding` Attribute

The `padding` attribute is used to increase the whitespace between the boundaries of the control and its actual content. Through the `android:padding` attribute, we can set the same amount of padding or spacing on all four sides of the control. Similarly, by using the `android:paddingLeft`, `android:paddingRight`, `android:paddingTop`, and `android:paddingBottom` attributes, we can specify the individual spacing on the left, right, top, and bottom of the control, respectively.

The following example sets the spacing on all four sides of the control to 5 pixels:

```
android:padding="5dip"
```

Similarly, the following example sets the spacing on the left side of the control to 5 pixels:

```
android:paddingLeft="5dip"
```

NOTE

To set the padding at runtime, we can call the `setPadding()` method.

Let's see how the controls are laid out in the LinearLayout layout using an example. Create a new Android Project called `LinearLayoutApp`. The original default content of the layout file `activity_linear_layout_app.xml` appears as shown in Listing 3.1.

LISTING 3.1 Default Code in the Layout File `activity_linear_layout_app.xml`

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true"
        android:text="@string/hello_world"
        tools:context=".LinearLayoutAppActivity" />
</RelativeLayout>
```

Let's apply the LinearLayout and add three `Button` controls to the layout. Modify the `activity_linear_layout_app.xml` to appear as shown in Listing 3.2.

LISTING 3.2 The `activity_linear_layout_app.xml` File on Adding Three `Button` Controls

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical" >
    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/Banana"
        android:text="Banana"
        android:layout_width="match_parent"
        android:layout_height="wrap_content" />
</LinearLayout>
```

The `orientation` of LinearLayout is set to `vertical`, declaring that we want to arrange its child elements vertically, one below the other. The height and width of the layout are set to expand to fill up all the available space of the enclosing container, that is, the device screen. Three `Button` controls are added to the layout, which appear one below the other. The IDs and text assigned to the three `Button` controls are `Apple`, `Mango`, and `Banana`, respectively. The `height` of the three controls is set to `wrap_content`, which is enough to accommodate the text. Finally, the `width` of the three controls is set to `match_parent`, so that the width of the three controls expands to fill up the available space of the LinearLayout container. We see the output shown in Figure 3.1.

FIGURE 3.1 Three `Button` controls arranged vertically in LinearLayout

To see the controls appear horizontally, set the `orientation` attribute of the LinearLayout to `horizontal`. We also need to set the `layout_width` attribute of the three controls to `wrap_content`; otherwise, we will be able to see only the first `Button` control, the one with the `Apple` ID. If the `layout_width` attribute of any control is set to `match_parent`, it takes up all the available space of the container, hiding the rest of the controls behind it. By setting the values of the `layout_width` attributes to `wrap_content`, we make sure that the width of the control expands just to fit its content and does not take up all the available space. Let's modify the `activity_linear_layout_app.xml` to appear as shown in Listing 3.3.

LISTING 3.3 The `activity_linear_layout_app.xml` File on Setting Horizontal Orientation to the `Button` Controls

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="horizontal" >
    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
```

```
    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/Banana"
        android:text="Banana"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

The controls are arranged horizontally, as shown in Figure 3.2.

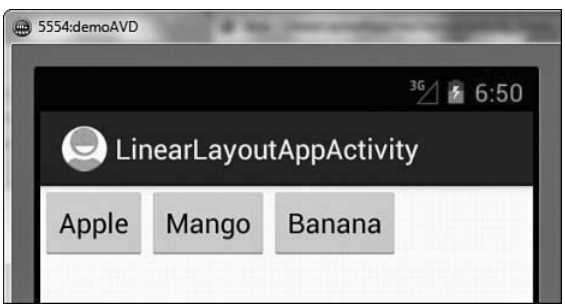


FIGURE 3.2 Three `Button` controls arranged horizontally in LinearLayout

Applying the `weight` Attribute

The `weight` attribute affects the size of the control. That is, we use `weight` to assign the capability to expand or shrink and consume extra space relative to the other controls in the container. The values of the `weight` attribute range from `0.0` to `1.0`, where `1.0` is the highest value. Let's suppose a container has two controls and one of them is assigned the `weight` of `1`. In that case, the control assigned the `weight` of `1` consumes all the empty space in the container, whereas the other control remains at its current size. If we assign a `weight` of `0.0` to both the controls, nothing happens and the controls maintain their original size. If both the attributes are assigned the same value above `0.0`, both the controls consume the extra space equally. Hence, `weight` lets us apply a size expansion ratio to the controls. To make the middle `Button` control, `Mango`, take up all the available space of the container, let's assign a `weight` attribute to the three controls. Modify the `activity_linear_layout_app.xml` file to appear as shown in Listing 3.4.

LISTING 3.4 The `activity_linear_layout_app.xml` File on Applying the `weight` Attribute to the `Button` Controls

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="0.0" />
    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="1.0" />
    <Button
        android:id="@+id/Banana"
        android:text="Banana"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_weight="0.0" />
</LinearLayout>
```

By setting the `layout_weight` attributes of `Apple`, `Mango`, and `Banana` to `0.0`, `1.0`, and `0.0`, respectively, we allow the `Mango` button control to take up all the available space of the container, as shown in Figure 3.3 (left). If we set the value of `layout_weight` of the `Banana` button control to `1.0` and that of `Mango` back to `0.0`, then all the available space of the container is consumed by the `Banana` button control, as shown in Figure 3.3 (middle). Similarly if we set the `layout_weight` of all controls to `1.0`, the entire container space will be equally consumed by the three controls, as shown in Figure 3.3 (right).

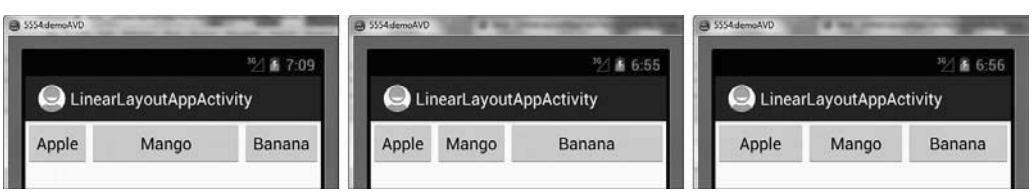


FIGURE 3.3 (left) The `weight` attribute of the Mango `Button` control set to 1.0, (middle) the `weight` attribute of the Banana `Button` control set to 1.0, and (right) all three `Button` controls set to the same `weight` attribute

Similarly if we set the weight of `Apple`, `Mango`, and `Banana` to `0.0`, `1.0`, and `0.5`, respectively, we get the output shown in Figure 3.4.

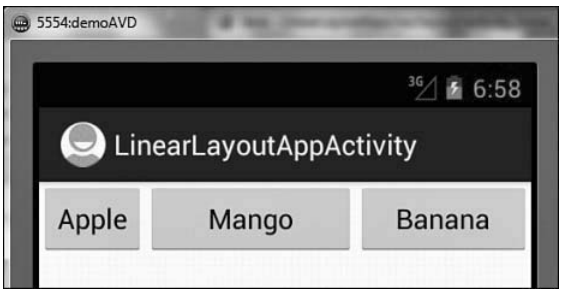


FIGURE 3.4 The `weight` attribute of the Apple, Mango, and Banana `Button` controls set to 0.0, 1.0, and 0.5

We can see that the text of the three controls is center-aligned. To align the content of a control, we use the `Gravity` attribute.

Applying the `Gravity` Attribute

The `Gravity` attribute is for aligning the content within a control. For example, to align the text of a control to the center, we set the value of its `android:gravity` attribute to `center`. The valid options for `android:gravity` include `left`, `center`, `right`, `top`, `bottom`, `center_horizontal`, `center_vertical`, `fill_horizontal`, and `fill_vertical`. The task performed by few of the said options is as follows:

- **`center_vertical`**—Places the object in the vertical center of its container, without changing its size
- **`fill_vertical`**—Grows the vertical size of the object, if needed, so it completely fills its container
- **`center_horizontal`**—Places the object in the horizontal center of its container, without changing its size
- **`fill_horizontal`**—Grows the horizontal size of the object, if needed, so it completely fills its container
- **`center`**—Places the object in the center of its container in both the vertical and horizontal axis, without changing its size

We can make the text of a control appear at the center by using the `android:gravity` attribute, as shown in this example:

```
android:gravity="center"
```

We can also combine two or more values of any attribute using the `|` operator. The following example centrally aligns the text horizontally and vertically within a control:

```
android:gravity="center_horizontal|center_vertical"
```

Figure 3.5 shows the `android:gravity` attribute set to left and right for the `Button` controls `Mango` and `Banana`.

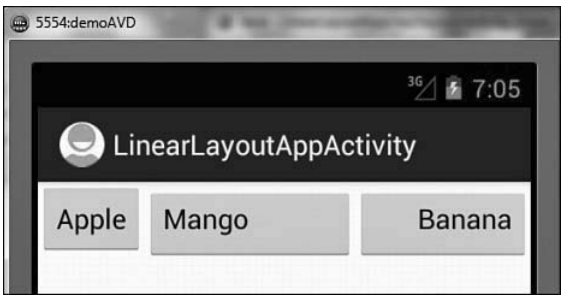


FIGURE 3.5 The text in the Mango and Banana `Button` controls aligned to the left and right, respectively, through the `android:gravity` attribute

Besides the `android:gravity` attribute, Android provides one more similar attribute, `android:layout_gravity`. Let's explore the difference between the two.

Using the `android:layout_gravity` Attribute

Where `android:gravity` is a setting used by the `View`, the `android:layout_gravity` is used by the container. That is, this attribute is used to align the control within the container. For example, to align the text within a `Button` control, we use the `android:gravity` attribute; to align the `Button` control itself in the LinearLayout (the container), we use the `android:layout_gravity` attribute. Let's add the `android:layout_gravity` attribute to align the `Button` controls themselves. To see the impact of using the `android:layout_gravity` attribute to align the `Button` controls in the LinearLayout, let's first arrange them vertically. So, let's modify `activity_linear_layout_app.xml` to make the `Button` controls appear vertically, one below the other as shown in Listing 3.5.

LISTING 3.5 The `activity_linear_layout_app.xml` File on Arranging the `Button` Controls Vertically

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <Button
        android:id="@+id/Banana"
        android:text="Banana"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

The preceding code arranges the `Button` controls vertically, as shown in Figure 3.6 (left). To align the `Button` controls `Mango` and `Banana` to the center and to the right of the LinearLayout container, add the following statements to the respective tags in the `activity_linear_layout_app.xml` layout file:

```
android:layout_gravity="center"
```

and

```
android:layout_gravity="right"
```

The two `Button` controls, `Mango` and `Banana`, are aligned at the center and to the right in the container, as shown in Figure 3.6 (middle).

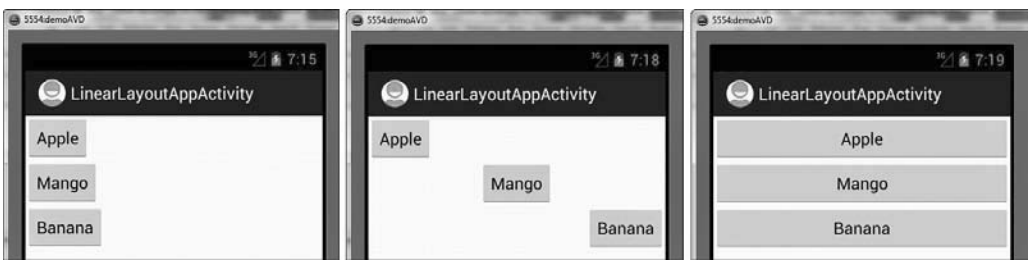


FIGURE 3.6 (left) The three `Button` controls vertically aligned with the `width` attribute set to `wrap_content`, (middle) the Mango and Banana `Button` controls aligned to the center and right of container, and (right) the width of the three `Button` controls expanded to take up all the available space

At the moment, the `layout_width` attribute of the three controls is set to `wrap_content`. The width of the three controls is just enough to accommodate their content. If we now set the value of the `android:layout_width` attribute for all three controls to `match_parent`, we find that all three `Button` controls expand in width to take up all the available space of the container, as shown in Figure 3.6 (right). Now we can apply the `android:gravity` attribute to align the text within the controls. Let's add the following three attributes to the `Button` controls `Apple`, `Mango`, and `Banana`:

```
android:gravity="left"
android:gravity="center"
```

and

```
android:gravity="right"
```

These lines of code align the content of the three `Button` controls to the `left`, to the `center`, and to the `right` within the control, as shown in Figure 3.7 (left). Because the three `Button` controls are arranged vertically in the layout (the orientation of the LinearLayout is set to vertical), the application of the `weight` attribute makes the controls

expand vertically instead of horizontally as we saw earlier. To see the effect, let's add the following statement to the tags of all three `Button` controls:

```
android:layout_weight="0.0"
```

As expected, there will be no change in the height of any control, as the `weight` value assigned is `0.0`. Setting an equal value above `0.0` for all three controls results in equal division of empty space among them. For example, assigning the `android:layout_weight="1.0"` to all three controls results in expanding their height, as shown in Figure 3.7 (middle).

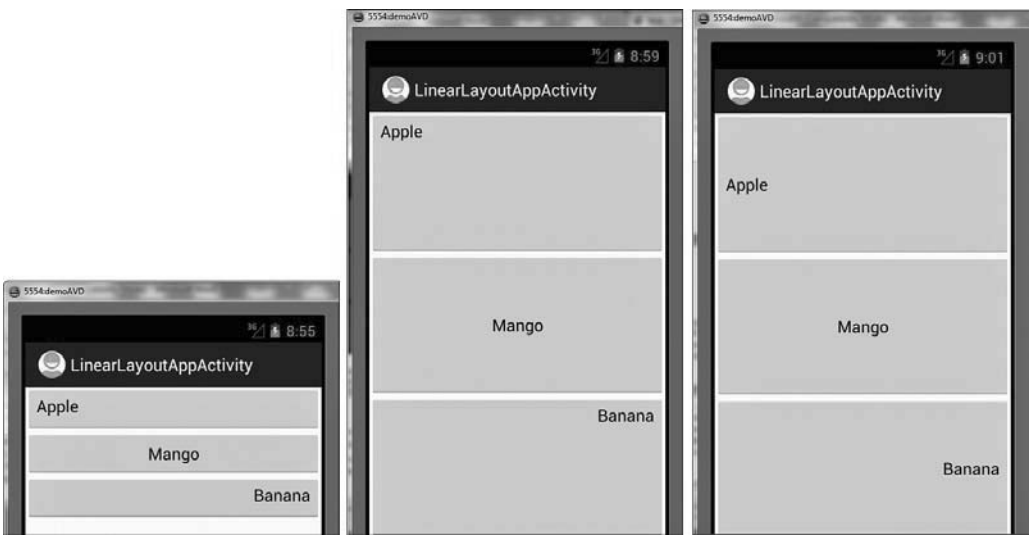


FIGURE 3.7 (left) The three `Button` controls with their text aligned to the left, center, and right, (middle) the vertical available space of the container apportioned equally among the three `Button` controls, and (right) the text of the three `Button` controls vertically aligned to the center

In the middle image of Figure 3.7, we see that the text in the `Apple` and `Banana` controls is not at the vertical center, so let's modify their `android:gravity` value, as shown here:

`android:gravity="center_vertical"` for the `Apple` control

`android:gravity="center_vertical|right"` for the `Banana` control

The `center_vertical` value aligns the content vertically to the center of the control, and the `right` value aligns the content to the right of the control. We can combine the values of the attribute using the `|` operator. After applying the values as shown in the preceding two code lines, we get the output shown in Figure 3.7 (right).

RelativeLayout

In RelativeLayout, each child element is laid out in relation to other child elements; that is, the location of a child element is specified in terms of the desired distance from the existing children. To understand the concept of relative layout practically, let's create a

new Android project called `RelativeLayoutApp`. Modify its layout file `activity_relative_layout_app.xml` to appear as shown in Listing 3.6.

LISTING 3.6 The `activity_relative_layout_app.xml` File on Arranging the `Button` Controls in the RelativeLayout Container

```
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <Button
        android:id="@+id/Apple"
        android:text="Apple"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="15dip"
        android:layout_marginLeft="20dip" />
    <Button
        android:id="@+id/Mango"
        android:text="Mango"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:padding="28dip"
        android:layout_toRightOf="@id/Apple"
        android:layout_marginLeft="15dip"
        android:layout_marginRight="10dip"
        android:layout_alignParentTop="true" />
    <Button
        android:id="@+id/Banana"
        android:text="Banana"
        android:layout_width="200dip"
        android:layout_height="50dip"
        android:layout_marginTop="15dip"
        android:layout_below="@id/Apple"
        android:layout_alignParentLeft="true" />
    <Button
        android:id="@+id/Grapes"
        android:text="Grapes"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:minWidth="100dp"
        android:layout_alignParentRight="true"
        android:layout_below="@id/Banana" />
    <Button
        android:id="@+id/Kiwi"
        android:text="Kiwi"
```

```
        android:layout_width="100dip"
        android:layout_height="wrap_content"
        android:layout_below="@id/Banana"
        android:paddingTop="15dip"
        android:paddingLeft="25dip"
        android:paddingRight="25dip" />
</RelativeLayout>
```

Before we understand how the controls in the previous code block are placed, let's have a quick look at different attributes used to set the positions of the layout controls.

Layout Control Attributes

The attributes used to set the location of the control relative to a container are

- **android:layout_alignParentTop**—The top of the control is set to align with the top of the container.
- **android:layout_alignParentBottom**—The bottom of the control is set to align with the bottom of the container.
- **android:layout_alignParentLeft**—The left side of the control is set to align with the left side of the container.
- **android:layout_alignParentRight**—The right side of the control is set to align with the right side of the container.
- **android:layout_centerHorizontal**—The control is placed horizontally at the center of the container.
- **android:layout_centerVertical**—The control is placed vertically at the center of the container.
- **android:layout_centerInParent**—The control is placed horizontally and vertically at the center of the container.

The attributes to control the position of a control in relation to other controls are

- **android:layout_above**—The control is placed above the referenced control.
- **android:layout_below**—The control is placed below the referenced control.
- **android:layout_toLeftOf**—The control is placed to the left of the referenced control.
- **android:layout_toRightOf**—The control is placed to the right of the referenced control.

The attributes that control the alignment of a control in relation to other controls are

- android:layout_alignTop— The top of the control is set to align with the top of the referenced control.

- **android:layout_alignBottom**—The bottom of the control is set to align with the bottom of the referenced control.
- **android:layout_alignLeft**—The left side of the control is set to align with the left side of the referenced control.
- **android:layout_alignRight**—The right side of the control is set to align with the right side of the referenced control.
- **android:layout_alignBaseline**—The baseline of the two controls will be aligned.

For spacing, Android defines two attributes: `android:layout_margin` and `android:padding`. The `android:layout_margin` attribute defines spacing for the container, while `android:padding` defines the spacing for the view. Let's begin with padding.

- **android:padding**—Defines the spacing of the content on all four sides of the control. To define padding for each side individually, use `android:paddingLeft`, `android:paddingRight`, `android:paddingTop`, and `android:paddingBottom`.
- **android:paddingTop**—Defines the spacing between the content and the top of the control.
- **android:paddingBottom**—Defines the spacing between the content and the bottom of the control.
- **android:paddingLeft**—Defines the spacing between the content and the left side of the control.
- **android:paddingRight**—Defines the spacing between the content and the right side of the control.

Here are the attributes that define the spacing between the control and the container:

- **android:layout_margin**—Defines the spacing of the control in relation to the controls or the container on all four sides. To define spacing for each side individually, we use the `android:layout_marginLeft`, `android:layout_marginRight`, `android:layout_marginTop`, and `android:layout_marginBottom` options.
- **android:layout_marginTop**—Defines the spacing between the top of the control and the related control or container.
- **android:layout_marginBottom**—Defines the spacing between the bottom of the control and the related control or container.
- **android:layout_marginRight**—Defines the spacing between the right side of the control and the related control or container.
- **android:layout_marginLeft**—Defines the spacing between the left side of the control and the related control or container.

The layout file `activity_relative_layout_app.xml` arranges the controls as follows:

The `Apple` button control is set to appear at a distance of `15dip` from the top and `20dip` from the left side of the `RelativeLayout` container. The width of the `Mango` button control is set to consume the available horizontal space. The text `Mango` appears at a distance of `28dip` from all sides of the control. The `Mango` control is set to appear to the right of the `Apple` control. The control is set to appear at a distance of `15dip` from the control on the left and `10dip` from the right side of the relative layout container. Also, the top of the `Button` control is set to align with the top of the container.

The `Banana` button control is assigned the `width` and `height` of `200dip` and `50dip`, respectively. The control is set to appear `15dip` below the `Apple` control. The left side of the control is set to align with the left side of the container.

The `Grapes` button control is set to appear below the `Banana` button control, and its width is set to expand just enough to accommodate its content. The height of the control is set to take up all available vertical space. The text `Grapes` is automatically aligned vertically; that is, it appears at the center of the vertical height when the `height` attribute is set to `match_parent`. The minimum width of the control is set to `100dip`. The right side of the control is set to align with the right side of the container.

The `Kiwi` Button control is set to appear below the `Banana` control. Its width is set to `100dip`, and the height is set to just accommodate its content. The text `Kiwi` is set to appear at the distance of `15dip`, `25dip`, and `25dip` from the top, left, and right boundary of the control.

We don't need to make any changes to the `RelativeLayoutAppActivity.java` file. Its original content is as shown in Listing 3.7.

LISTING 3.7 The Default Code in the Activity File `RelativeLayoutAppActivity.java`

```
package com.androidunleashed.relativelayoutapp;

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

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

When the application is run, we see the output shown in Figure 3.8.

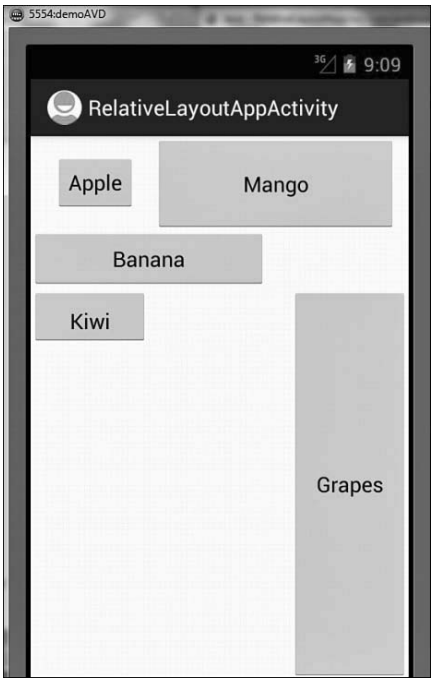


FIGURE 3.8 The five `Button` controls' layout relative to each other

We can make the text `Grapes` appear centrally at the top row by adding the following line:

```
android:gravity="center_horizontal"
```

So, its tag appears as follows:

```
<Button
    android:id="@+id/Grapes"
    android:text="Grapes"
    android:layout_width="wrap_content"
    android:layout_height="match_parent"
    android:minWidth="100dp"
    android:layout_alignParentRight="true"
    android:layout_below="@id/Banana"
    android:gravity="center_horizontal" />
```

The output is modified to appear as shown in Figure 3.9.

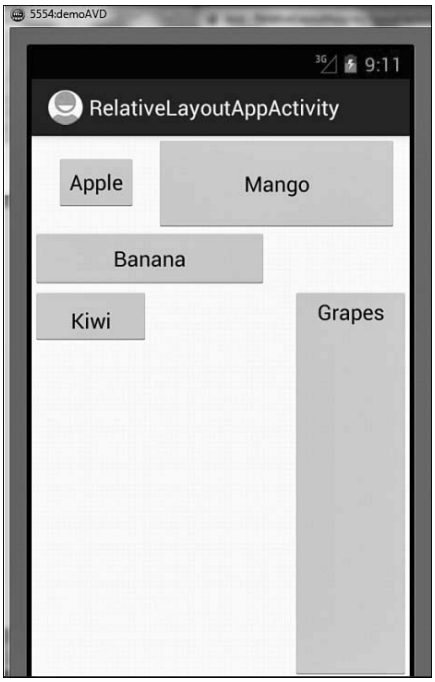


FIGURE 3.9 The Grapes `Button` control aligned horizontally at the center

Let's explore the concept of laying out controls in the RelativeLayout container by writing an application. The application that we are going to create is a simple `Login Form` application that asks the user to enter a `User ID` and `Password`. The `TextView`, `EditText`, and `Button` controls in the application are laid out in a RelativeLayout container (see Figure 3.10—left). If either the `User ID` or `Password` is left blank, the message `The User ID or password is left blank. Please Try Again` is displayed. If the correct `User ID` and `Password`, in this case, `guest`, are entered, then a welcome message is displayed. Otherwise, the message `The User ID or password is incorrect. Please Try Again` is displayed.

So, let's create the application. Launch the Eclipse IDE and create a new Android application called `LoginForm`. Arrange four `TextView` controls, two `EditText` controls, and a `Button` control in RelativeLayout, as shown in the layout file `activity_login_form.xml` displayed in Listing 3.8.

LISTING 3.8 The `activity_login_form.xml` on Laying Out the `TextView`, `EditText`, and `Button` Controls in the RelativeLayout Container

```
<RelativeLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >
    <TextView
        android:id="@+id/sign_msg"
```

```
        android:text = "Sign In"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:typeface="serif"
        android:textSize="25dip"
        android:textStyle="bold"
        android:padding="10dip"
        android:layout_centerHorizontal="true"/>
    <TextView
        android:id="@+id/user_msg"
        android:text = "User ID:"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_margin="10dip"
        android:layout_below="@+id/sign_msg" />
    <EditText
        android:id="@+id/user_ID"
        android:layout_height="wrap_content"
        android:layout_width="250dip"
        android:layout_below="@+id/sign_msg"
        android:layout_toRightOf="@+id/user_msg"
        android:singleLine="true" />
    <TextView
        android:id="@+id/password_msg"
        android:text = "Password:"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/user_msg"
        android:layout_margin="10dip"
        android:paddingTop="10dip"/>
    <EditText
        android:id="@+id/password"
        android:layout_height="wrap_content"
        android:layout_width="250dp"
        android:singleLine="true"
        android:layout_below="@+id/user_ID"
        android:layout_toRightOf="@+id/password_msg"
        android:password="true" />
    <Button
        android:id="@+id/login_button"
        android:text="Sign In"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_marginTop="10dip"
        android:layout_below="@+id/password_msg"/>
```

```
    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:id="@+id/response"
        android:layout_below="@+id/login_button"/>
</RelativeLayout>
```

The controls in the application are arranged in the RelativeLayout, as explained here:

- Through the `TextView` control `sign_msg`, the text `Sign In` is displayed horizontally centered at the top. It is displayed in bold serif font, `25 dip` in size. The text is padded with a space of `10dip` on all four sides of its container.
- Another `TextView` control, `user_msg`, displays the text `User ID` below the `TextView sign_msg`. The `TextView` is placed `10dip` from all four sides.
- An `EditText` control `user_ID` is displayed below `sign_msg` and to the right of `user_msg`. The width assigned to the `TextView` control is `250 dip` and is set to `single-line` mode, so if the user types beyond the given width, the text scrolls to accommodate extra text but does not run over to the second line.
- A `TextView password_msg` control displaying the text `Password:` is displayed below the `TextView user_msg`. The `TextView` control is placed at a spacing of `10dip` from all four sides, and the text `Password:` is displayed at `10dip` from the control's top boundary.
- An `EditText` control `password` is displayed below the `EditText user_ID` and to the right of the `TextView password_msg`. The `width` assigned to the `TextView` control is `250` dip and is set to `single-line` mode. In addition, the typed characters are converted into dots for security.
- A `Button` control `login_button` with the caption `Sign In` is displayed below the `TextView password_msg`. The button is horizontally centered and is set to appear at `10dip` distance from the `EditText` control `password`.
- A `TextView` control `response` is placed below the `Button login_button`. It is used to display messages to the user when the `Sign In` button is pressed after entering `User ID` and `Password`.

To authenticate the user, we need to access the `User ID` and `Password` that is entered and match these values against the valid `User ID` and `Password`. In addition, we want to validate the `EditText` controls to confirm that none of them is blank. We also want to welcome the user if he or she is authorized. To do all this, we write the code in the activity file `LoginFormActivity.java` as shown in Listing 3.9.

LISTING 3.9 Code Written in the Java Activity File `LoginFormActivity.java`

```
package com.androidunleashed.loginform;

import android.app.Activity;
import android.os.Bundle;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.view.View;
import android.widget.TextView;

public class LoginFormActivity extends Activity implements OnClickListener  {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_login_form);
        Button b = (Button)this.findViewById(R.id.login_button);
        b.setOnClickListener(this);
    }

    public void onClick(View v) {
        EditText userid = (EditText) findViewById(R.id.user_ID);
        EditText password = (EditText) findViewById(R.id.password);
        TextView resp = (TextView)this.findViewById(R.id.response);
        String usr = userid.getText().toString();
        String pswd = password.getText().toString();
        if(usr.trim().length() == 0 || pswd.trim().length() == 0){
            String str = "The User ID or password is left blank \nPlease Try Again";
            resp.setText(str);
        }
        else{
            if(usr.equals("guest") && pswd.equals("guest")) resp.setText("Welcome " +
            usr+ " ! ");
           else resp.setText("The User ID or password is incorrect \nPlease Try Again");
        }
    }
}
```

The `Button` control is accessed from the layout file and is mapped to the `Button` object `b`. This activity implements the `OnClickListener` interface. Hence, the class implements the callback method `onClick()`, which is invoked when a click event occurs on the `Button` control.

In the `onClick()` method, the `user_ID` and `password EditText` controls are accessed from the layout file and mapped to the `EditText` objects `userid` and `password`. Also, the `TextView` control `response` is accessed from the layout file and is mapped to the `TextView`

object `resp`. The `User ID` and `password` entered by the user in the two `EditText` controls are accessed through the objects `userid` and `password` and assigned to the two Strings `usr` and `pswd`, respectively. The data in the `usr` and `pswd` strings is checked for authentication. If the user has left any of the `EditText` controls blank, the message `The User ID or password is left blank. Please Try Again` is displayed, as shown in Figure 3.10 (left). If the `User ID` and `password` are correct, then a welcome message is displayed (see Figure 3.10—right). Otherwise, the message `The User ID or password is incorrect. Please Try Again` is displayed, as shown in Figure 3.10 (middle).

FIGURE 3.10 (left) The Login Form displays an error if fields are left blank, (middle) the Password Incorrect message displays if the user ID or password is incorrect, and (right) the Welcome message displays when the correct user ID and password are entered.

AbsoluteLayout

Each child in an AbsoluteLayout is given a specific location within the bounds of the container. Such fixed locations make AbsoluteLayout incompatible with devices of different screen size and resolution. The controls in AbsoluteLayout are laid out by specifying their exact *X* and *Y* positions. The coordinate 0,0 is the origin and is located at the top-left corner of the screen.

Let's write an application to see how controls are positioned in AbsoluteLayout. Create a new Android Project called `AbsoluteLayoutApp`. Modify its layout file, `activity_absolute_layout_app.xml`, as shown in Listing 3.10.

LISTING 3.10 The Layout File `activity_absolute_layout_app.xml` on Arranging Controls in the AbsoluteLayout Container

```
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="New Product Form"
        android:textSize="20sp"
```

```
        android.textStyle="bold"
        android:layout_x="90dip"
        android:layout_y="2dip"/>
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Product Code:"
        android:layout_x="5dip"
        android:layout_y="40dip" />
    <EditText
        android:id="@+id/product_code"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minWidth="100dip"
        android:layout_x="110dip"
        android:layout_y="30dip" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Product Name:"
        android:layout_x="5dip"
        android:layout_y="90dip"/>
    <EditText
        android:id="@+id/product_name"
        android:layout_width="200dip"
        android:layout_height="wrap_content"
        android:minWidth="200dip"
        android:layout_x="110dip"
        android:layout_y="80dip"
        android:scrollHorizontally="true" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Product Price:"
        android:layout_x="5dip"
        android:layout_y="140dip" />
    <EditText
        android:id="@+id/product_price"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:minWidth="100dip"
        android:layout_x="110dip"
        android:layout_y="130dip" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:id="@+id/click_btn"
        android:text="Add New Product"
        android:layout_x="80dip"
        android:layout_y="190dip" />
</AbsoluteLayout>
```

The controls in `activity_absolute_layout_app.xml` are as follows:

- The `New Product Form TextView` is set to appear `90dip` from the left and `2dip` from the top side of the container. The size of the text is set to `20sp`, and its style is set to `bold`.
- The `Product Code TextView` is set to appear `5dip` from the left and `40dip` from the top side of the container.
- The `product_code EditText` control is set to appear `110dip` from the left and `30dip` from the top side of the container. The minimum width of the control is set to `100dp`.
- The `ProductName TextView` control is set to appear `5dip` from the left and `90dip` from the top side of the container.
- The `product_name EditText` control is set to appear `110dip` from the left and `80dip` from the top side of the container. The minimum width of the control is set to `200dip`, and its text is set to scroll horizontally when the user types beyond its width.
- The `Product Price TextView` is set to appear `5dip` from the left and `140dip` from the top side of the container.
- The `product_price EditText` control is set to appear `110dip` from the left and `130dip` from the top side of the container. The minimum width of the control is set to `100dip`.
- The `click_btn Button`, `Add New Product`, is set to appear `80dip` from the left and `190dip` from the top side of the container.

If we don't specify the x, y coordinates of a control in AbsoluteLayout, it is placed in the origin point, that is, at location 0,0. If the value of the x and y coordinates is too large, the control does not appear on the screen. The values of the x and y coordinates are specified in any units, such as `sp`, `in`, `mm`, and `pt`.

After specifying the locations of controls in the layout file `activity_absolute_layout_app.xml`, we can run the application. There is no need to make any changes in the file `AbsoluteLayoutAppActivity.java`. When the application is run, we get the output shown in Figure 3.11.

FIGURE 3.11 Different controls laid out in AbsoluteLayout

The AbsoluteLayout class is not used often, as it is not compatible with Android phones of different screen sizes and resolutions.

The next layout we are going to discuss is FrameLayout. Because we will learn to display images in FrameLayout, let's first take a look at the `ImageView` control that is often used to display images in Android applications.

Using `ImageView`

An `ImageView` control is used to display images in Android applications. An image can be displayed by assigning it to the `ImageView` control and including the `android:src` attribute in the XML definition of the control. Images can also be dynamically assigned to the `ImageView` control through Java code.

A sample ImageView tag when used in the layout file is shown here:

```
<ImageView
    android:id="@+id/first_image"
    android:src = "@drawable/bintupic"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:scaleType="fitXY"
    android:adjustViewBounds="true"
    android:maxHeight="100dip"
    android:maxWidth="250dip"
    android:minHeight="100dip"
    android:minWidth="250dip"
    android:resizeMode="horizontal|vertical" />
```

Almost all attributes that we see in this XML definition should be familiar, with the exception of the following ones:

- **android:src**—Used to assign the image from drawable resources. We discuss drawable resources in detail in Chapter 4. For now, assume that the image in the `res/drawable` folder is set to display through the `ImageView` control via this attribute.

  Example:

  ```
  android:src = "@drawable/bintupic"
  ```

  You do not need to specify the image file extension. JPG and GIF files are supported, but the preferred image format is PNG.

- **android:scaleType**—Used to scale an image to fit its container. The valid values for this attribute include `fitXY`, `center`, `centerInside`, and `fitCenter`. The value `fitXY` independently scales the image around the X and Y axes without maintaining the aspect ratio to match the size of container. The value `center` centers the image in the container without scaling it. The value `centerInside` scales the image uniformly, maintaining the aspect ratio so that the width and height of the image fit the size of its container. The value `fitCenter` scales the image while maintaining the aspect ratio, so that one of its X or Y axes fits the container.

- **android:adjustViewBounds**—If set to `true`, the attribute adjusts the bounds of the `ImageView` control to maintain the aspect ratio of the image displayed through it.

- **android:resizeMode**—The `resizeMode` attribute is used to make a control resizable so we can resize it horizontally, vertically, or around both axes. We need to click and hold the control to display its resize handles. The resize handles can be dragged in the desired direction to resize the control. The available values for the `resizeMode` attribute include `horizontal`, `vertical`, and `none`. The `horizontal` value resizes the control around the horizontal axis, the `vertical` value resizes around the vertical axis, the `both` value resizes around both the horizontal and vertical axes, and the value `none` prevents resizing.

FrameLayout

FrameLayout is used to display a single `View`. The `View` added to a FrameLayout is placed at the top-left edge of the layout. Any other `View` added to the FrameLayout overlaps the previous `View`; that is, each `View` stacks on top of the previous one. Let's create an application to see how controls can be laid out using FrameLayout.

In the application we are going to create, we will place two `ImageView` controls in the FrameLayout container. As expected, only one `ImageView` will be visible, as one `ImageView` will overlap the other `ImageView`, assuming both `ImageView` controls are of the same size. We will also display a button on the `ImageView`, which, when selected, displays the hidden `ImageView` underneath.

Let's start with the application. Create a new Android project called `FrameLayoutApp`. To display images in Android applications, the image is first copied into the `res/drawable` folder and from there, it is referred to in the layout and other XML files. We look at the procedure for displaying images, as well as the concept of drawable resources, in detail in Chapter 4. For the time being, it is enough to know that to enable the image(s) to be referred to in the layout files placed in the `res/drawable` folder, the image needs to exist in the `res/drawable` folder. There are four types of drawable folders: `drawable-xhdpi`, `drawable-hdpi`, `/res/drawable-mdpi`, and `/res/drawable-ldpi`. We have to place images of different resolutions and sizes in these folders. The graphics with the resolutions `320 dpi`, `240dpi`, `160 dpi`, and `120dpi` (`96 x 96 px`, `72 x 72 px`, `48 x 48 px`, and `36 x 36 px`), are stored in the `res/drawable-xhdpi`, `res/drawable-hdpi`, `res/drawable-mdpi`, and `res/drawable-ldpi` folders, respectively. The application picks up the appropriate graphic from the correct folder. So, if we copy two images called `bintupic.png` and `bintupic2.png` of the preceding size and resolution and paste them into the four `res/drawable` folders, the `Package Explorer` resembles Figure 3.12.

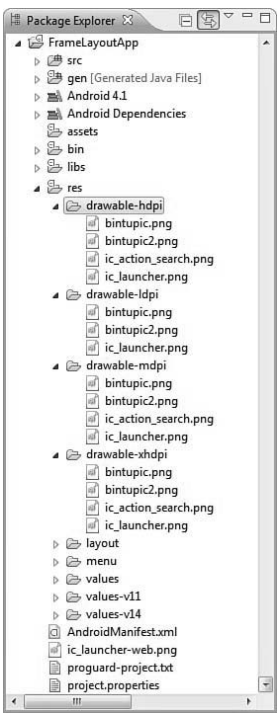


FIGURE 3.12 The `Package Explorer` window showing the two images, `bintupic.png` and `bintupic2.png`, dropped into the `res/drawable` folders

To display two `ImageView`s and a `TextView` in the application, let's write the code in the layout file `activity_frame_layout_app.xml` as shown in Listing 3.11.

LISTING 3.11 The Layout File `activity_frame_layout_app.xml` on Arranging the `ImageView` and `TextView` Controls in the FrameLayout Container

```
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="match_parent"
    android:layout_height="match_parent">
    <ImageView
        android:id="@+id/first_image"
        android:src = "@drawable/bintupic"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="fitXY" />
    <ImageView
        android:id="@+id/second_image"
        android:src = "@drawable/bintupic2"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:scaleType="fitXY" />
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Click the image to switch"
        android:layout_gravity="center_horizontal|bottom"
        android:padding="5dip"
        android:textColor="#ffffff"
        android:textStyle="bold"
        android:background="#333333"
        android:layout_marginBottom="10dip" />
</FrameLayout>
```

The `first_image` and `second_image ImageView` controls are set to display the images `bintupic.png` and `bintupic2.png`, respectively. To make the two images stretch to cover the entire screen, the `scaleType` attribute in the `ImageView` tag is set to `fitXY`. A `TextView`, `Click the image to switch`, is set to display at the horizontally centered position and at a distance of `10dip` from the bottom of the container. The spacing between the text and the boundary of the `TextView` control is set to `5dip`. The background of the text is set to a dark color, the foreground color is set to white, and its style is set to bold. When a user selects the current image on the screen, the image should switch to show the hidden image. For this to occur, we need to write code in the activity file as shown in Listing 3.12.

LISTING 3.12 Code Written in the Java Activity File `FrameLayoutAppActivity.java`

```
package com.androidunleashed.framelayoutapp;

import android.app.Activity;
import android.os.Bundle;
import android.widget.ImageView;
import android.view.View.OnClickListener;
import android.view.View;

public class FrameLayoutAppActivity extends Activity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_frame_layout_app);
        final ImageView first_image = (ImageView)this.findViewById(R.id.first_image);
        final ImageView second_image = (ImageView)this.findViewById(R.id.second_image);
        first_image.setOnClickListener(new OnClickListener(){
            public void onClick(View view) {
                second_image.setVisibility(View.VISIBLE);
                view.setVisibility(View.GONE);
            }
        });
        second_image.setOnClickListener(new OnClickListener(){
            public void onClick(View view) {
                first_image.setVisibility(View.VISIBLE);
                view.setVisibility(View.GONE);
            }
        });
    }
}
```

The two `first_image` and `second_image ImageView` controls are located through the `findViewById` method of the Activity class and assigned to the two `ImageView` objects, `first_image` and `second_image`, respectively. We register the click event by calling the `setOnClickListener()` method with an `OnClickListener`. An anonymous listener is created on the fly to handle click events for the `ImageView`. When the `ImageView` is clicked, the `onClick()` method of the listener is called. In the `onClick()` method, we switch the images; that is, we make the current `ImageView` invisible and the hidden `ImageView` visible. When the application runs, we see the output shown in Figure 3.13 (left). The application shows an image, and the other image is hidden behind it because in FrameLayout one View overlaps the other. When the user clicks the image, the images are switched, as shown in Figure 3.13 (right).

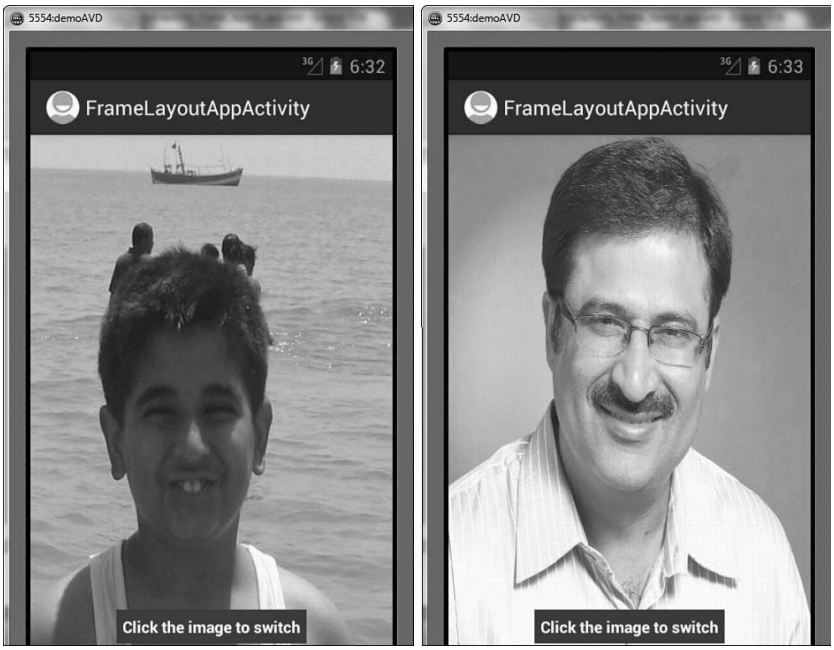


FIGURE 3.13 (left) An image and a `TextView` laid out in FrameLayout, and (right) the images switch when clicked

TableLayout

The TableLayout is used for arranging the enclosed controls into rows and columns. Each new row in the TableLayout is defined through a `TableRow` object. A row can have zero or more controls, where each control is called a `cell`. The number of columns in a TableLayout is determined by the maximum number of cells in any row. The width of a column is equal to the widest cell in that column. All elements are aligned in a column; that is, the width of all the controls increases if the width of any control in the column is increased.

NOTE

We can nest another TableLayout within a table cell, as well.

Operations Applicable to TableLayout

We can perform several operations on TableLayout columns, including stretching, shrinking, collapsing, and spanning columns.

Stretching Columns

The default width of a column is set equal to the width of the widest column, but we can stretch the column(s) to take up available free space using the `android:stretchColumns`

attribute in the TableLayout. The value assigned to this attribute can be a single column number or a comma-delimited list of column numbers. The specified columns are stretched to take up any available space on the row.

Examples:

- **`android:stretchColumns="1"`**—The second column (because the column numbers are zero-based) is stretched to take up any available space in the row.
- **`android:stretchColumns="0,1"`**—Both the first and second columns are stretched to take up the available space in the row.
- **`android:stretchColumns="*"`**—All columns are stretched to take up the available space.

Shrinking Columns

We can shrink or reduce the width of the column(s) using the `android:shrinkColumns` attribute in the TableLayout. We can specify either a single column or a comma-delimited list of column numbers for this attribute. The content in the specified columns word-wraps to reduce their width.

NOTE

By default, the controls are not word-wrapped.

Examples:

- **`android:shrinkColumns="0"`**—The first column's width shrinks or reduces by word-wrapping its content.
- **`android:shrinkColumns="*"`**—The content of all columns is word-wrapped to shrink their widths.

Collapsing Columns

We can make the column(s) collapse or become invisible through the `android:collapseColumns` attribute in the TableLayout. We can specify one or more comma-delimited columns for this attribute. These columns are part of the table information but are invisible. We can also make column(s) visible and invisible through coding by passing the `Boolean` values `false` and `true`, respectively, to the `setColumnCollapsed()` method in the TableLayout. For example:

- **`android:collapseColumns="0"`**—The first column appears collapsed; that is, it is part of the table but is invisible. It can be made visible through coding by using the `setColumnCollapsed()` method.

Android Image Gallery Using Viewpager

Screen slides are transitions between one entire screen to another and are common with UIs like setup wizards or slideshows. ViewPager can animate screen slides automatically.

ViewPager is a layout manager that allows the user to flip left and right through pages of data. We supply an implementation of a **PagerAdapter** to generate the pages that the view shows.

Android *image slider* slides one entire screen to another screen. Image slider is created by **ViewPager** which is provided by support library. To implement image slider, you need to inherit ViewPager class which extends PagerAdapter.

File: activity_main.xml

In activity_main.xml file, we have wrapped ViewPager inside RelativeLayout.

```
<RelativeLayout
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="fill_parent"
   android:layout_height="fill_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="com.example.test.imageslider.MainActivity">
    <android.support.v4.view.ViewPager
      android:id="@+id/viewPage"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent" />
 </RelativeLayout>
```

File: MainActivity.java

```
package com.example.test.imageslider;
import android.support.v4.view.ViewPager;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
 public class MainActivity extends AppCompatActivity {
    @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      ViewPager mViewPager = (ViewPager) findViewById(R.id.viewPage);
      ImageAdapter adapterView = new ImageAdapter(this);
      mViewPager.setAdapter(adapterView);
   }
}
```

ImageAdapter class

Now create ImageAdapter class which extends PagerAdapter for android image slider.
Place some images in drawable folder which are to be slid.

File: ImageAdapter.java

```
package com.example.test.imageslider;
import android.content.Context;
import android.support.v4.view.PagerAdapter;
import android.support.v4.view.ViewPager;
import android.view.View;
import android.view.ViewGroup;
import android.widget.ImageView;
public class ImageAdapter extends PagerAdapter{
   Context mContext;
    ImageAdapter(Context context) {
      this.mContext = context;
   }     public boolean isViewFromObject(View view, Object object) {
      return view == ((ImageView) object);     }
    private int[] sliderImageId = new int[]{R.drawable.image1, R.drawable.image2, R.drawable.image3,R
.drawable.image4, R.drawable.image5,   };
   public Object instantiateItem(ViewGroup container, int position) {
      ImageView imageView = new ImageView(mContext);
      imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
      imageView.setImageResource(sliderImageId[position]);
      ((ViewPager) container).addView(imageView, 0);
      return imageView;
   }    public void destroyItem(ViewGroup container, int position, Object object) {
      ((ViewPager) container).removeView((ImageView) object);
   }
   public int getCount() {
      return sliderImageId.length;
   }
}
```

We need to override following methods of PagerAdapter class.

1. **isViewFromObject(View, Object):** This method checks the view whether it is associated with key and returned by instantiateItem().
2. **instantiateItem(ViewGroup, int):** This method creates the page position passed as an argument.
3. **destroyItem(ViewGroup, int, Object):** It removes the page from its current position from container. In this example we simply removed object using removeView().
4. **getCount():** It returns the number of available views in ViewPager.

Dalvik Debug Monitor Service (DDMS)

Android ships with a debugging tool called the Dalvik Debug Monitor Server (DDMS), which provides port-forwarding services, screen capture on the device, thread and heap information on the device, logcat, process, and radio state information, incoming call and SMS spoofing, location data spoofing, and more. It is not an exhaustive exploration of all the features and capabilities.

DDMS features:

Viewing heap usage for a process

DDMS allows you to view how much heap memory a process is using. This information is useful in tracking heap usage at a certain point of time during the execution of your application.

Tracking memory allocation of objects

DDMS provides a feature to track objects that are being allocated to memory and to see which classes and threads are allocating the objects. This allows you to track, in real time, where objects are being allocated when you perform certain actions in your application. This information is valuable for assessing memory usage that can affect application performance.

Working with an emulator or device's file system

DDMS provides a File Explorer tab that allows you to view, copy, and delete files on the device. This feature is useful in examining files that are created by your application or if you want to transfer files to and from the device.

Starting method profiling

Method profiling is a means to track certain metrics about a method, such as number of calls, execution time, and time spent executing the method. If you want more granular control over where profiling data is collected, use the startMethodTracing() and stopMethodTracing() methods.

Using the Network Traffic tool

In Android 4.0, the DDMS (Dalvik Debug Monitor Server) includes a Detailed Network Usage tab that makes it possible to track when your application is making network requests. Using this tool, you can monitor how and when your app transfers data and optimize the underlying code appropriately.

Using LogCat

LogCat is integrated into DDMS, and outputs the messages that you print out using the Log class along with other system messages such as stack traces when exceptions are thrown

Running DDMS

DDMS is integrated into Eclipse and is also shipped in the `tools/` directory of the SDK. DDMS works with both the emulator and a connected device. If both are connected and running simultaneously, DDMS defaults to the emulator.

From Eclipse: Click Window > Open Perspective > Other... > DDMS.
From the command line: Type `ddms` (or `./ddms` on Mac/Linux) from the `tools/` directory.
From Android studio click on Tools>Android>Android device Monitor.

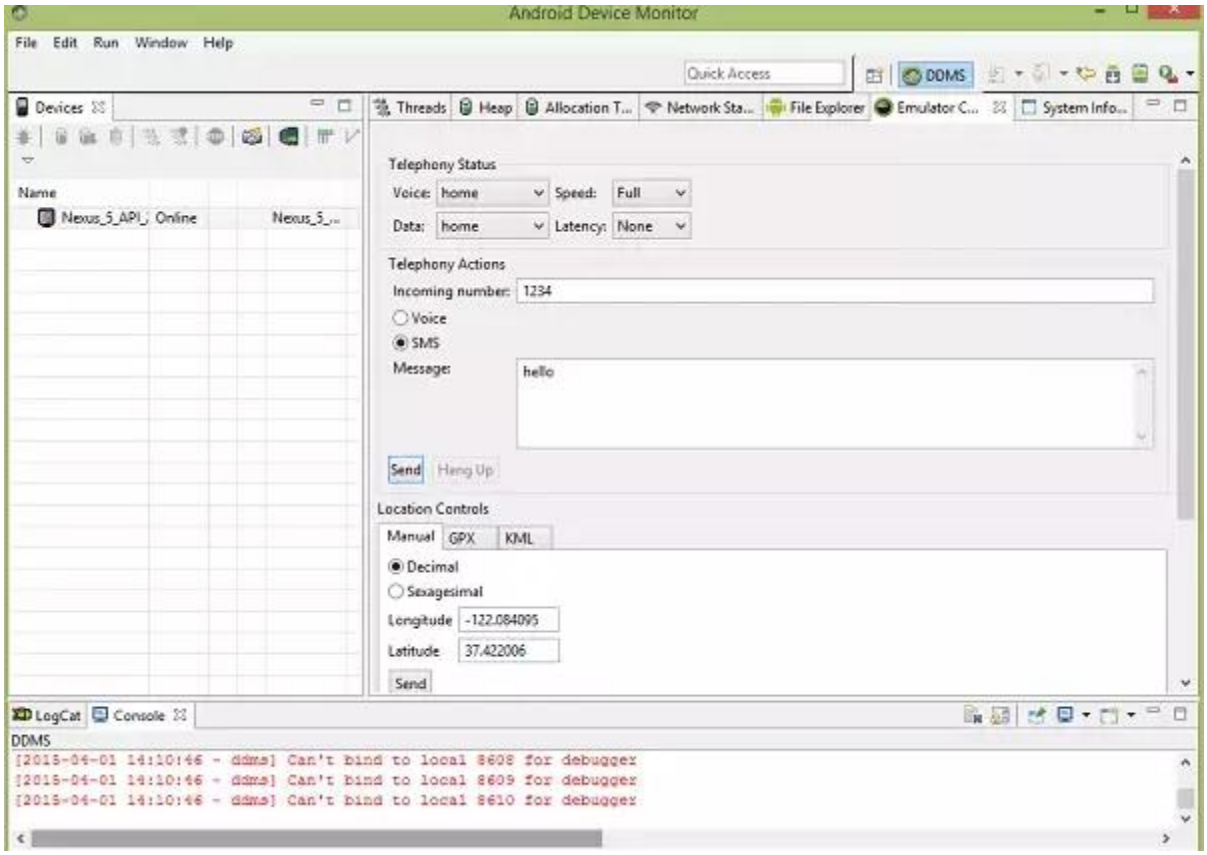
How it works

- In android, each application runs in its own process and each process run in the virtual machine. Each VM exposes a unique port, that a debugger can attach to.

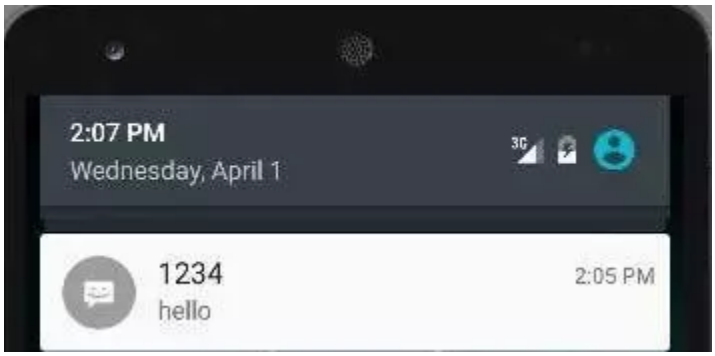
- When DDMS starts, it connects to adb. When a device is connected, a VM monitoring service is created between adb and DDMS, which notifies DDMS when a VM on the device is started or terminated.

Making SMS

Making sms to emulator.we need to call telnet client and server as shown below.



Now click on send button, and you will see an sms notification in the emulator window. It is shown below –



Dalvik Debug Monitor Service (DDMS)

Android ships with a debugging tool called the Dalvik Debug Monitor Server (DDMS), which provides port-forwarding services, screen capture on the device, thread and heap information on the device, logcat, process, and radio state information, incoming call and SMS spoofing, location data spoofing, and more. This page provides a modest discussion of DDMS features; it is not an exhaustive exploration of all the features and capabilities.

Viewing heap usage for a process

DDMS allows you to view how much heap memory a process is using. This information is useful in tracking heap usage at a certain point of time during the execution of your application.

Tracking memory allocation of objects

DDMS provides a feature to track objects that are being allocated to memory and to see which classes and threads are allocating the objects. This allows you to track, in real time, where objects are being allocated when you perform certain actions in your application. This information is valuable for assessing memory usage that can affect application performance.

Working with an emulator or device's file system

DDMS provides a File Explorer tab that allows you to view, copy, and delete files on the device. This feature is useful in examining files that are created by your application or if you want to transfer files to and from the device.

Starting method profiling

Method profiling is a means to track certain metrics about a method, such as number of calls, execution time, and time spent executing the method. If you want more granular control over where profiling data is collected, use the `startMethodTracing()` and `stopMethodTracing()` methods.

Using the Network Traffic tool

In Android 4.0, the DDMS (Dalvik Debug Monitor Server) includes a Detailed Network Usage tab that makes it possible to track when your application is making network requests. Using this tool, you can monitor how and when your app transfers data and optimize the underlying code appropriately.

Using LogCat

LogCat is integrated into DDMS, and outputs the messages that you print out using the `Log` class along with other system messages such as stack traces when exceptions are thrown

Running DDMS

DDMS is integrated into Eclipse and is also shipped in the `tools/` directory of the SDK. DDMS works with both the emulator and a connected device. If both are connected and running simultaneously, DDMS defaults to the emulator.

- From Eclipse: Click **Window > Open Perspective > Other... > DDMS**.
- From the command line: Type `ddms` (or `./ddms` on Mac/Linux) from the `tools/` directory.

- From Android studio click on **Tools>Android>Android device Monitor**.

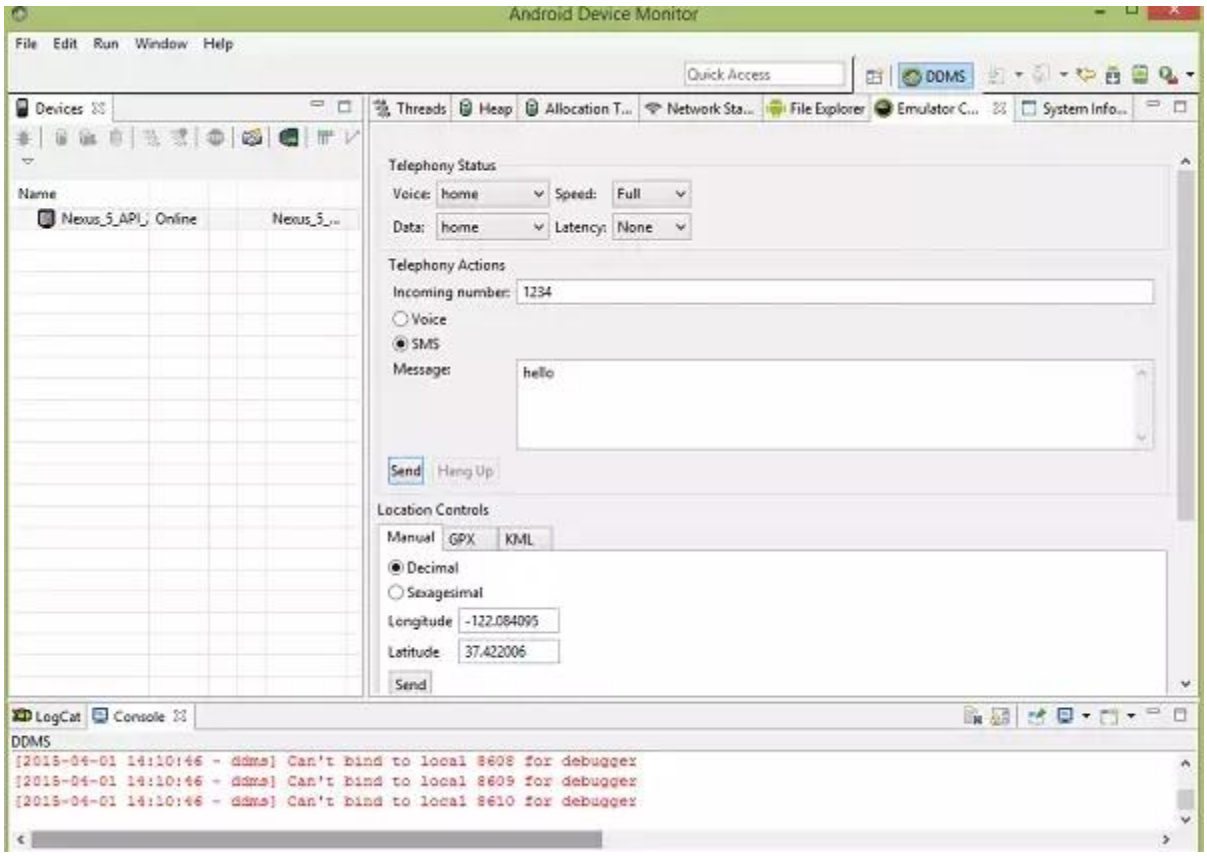
How it works

- In android, each application runs in its own process and each process run in the virtual machine. Each VM exposes a unique port, that a debugger can attach to.

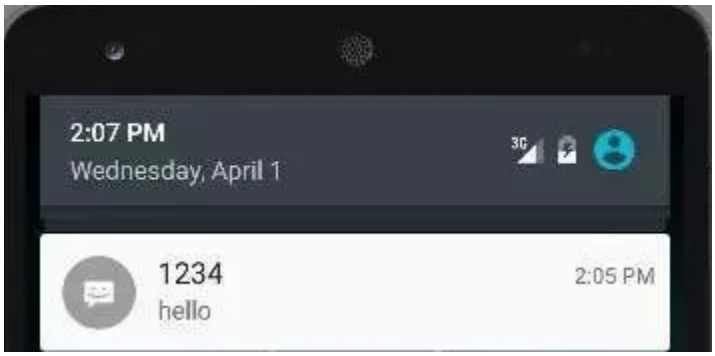
- When DDMS starts, it connects to adb. When a device is connected, a VM monitoring service is created between adb and DDMS, which notifies DDMS when a VM on the device is started or terminated.

Making SMS

Making sms to emulator.we need to call telnet client and server as shown below.



Now click on send button, and you will see an sms notification in the emulator window. It is shown below –


2:07 PM
Wednesday, April 1
3G
1234
hello
2:05 PM

Android Studio provides a debugger that allows you to do the following and more:

a. Select a device to debug your app on.
b. Set breakpoints in your Java, Kotlin, and C/C++ code.
c. Examine variables and evaluate expressions at runtime.

You can start a debugging session as follows:

1. Set some breakpoints in the app code.

2. In the toolbar, click **Debug**  to display the **Select Deployment Target** window.

   - If no devices appear in the **Select Deployment Target** window after you click **Debug**, then you need to either connect or click **Create new virtual device** to use the Android Emulator.
   - If, instead of the **Select Deployment Target** window, you see a dialog asking if you want to "switch from Run to Debug," that means your app is already running on the device and it will restart in order to begin debugging. If you'd rather keep the same instance of the app running, click **Cancel Debug** and instead attach the debugger to a running app.

3. **Select a deployment target and click OK.**

   Android Studio builds an APK, signs it with a debug key, installs it on your selected device, and runs it. If you add, Android Studio also runs the LLDB debugger in the **Debug** window to debug your native code.

4. If the **Debug** window is not open, select **View > Tool Windows > Debug** (or click **Debug**  in the tool window bar), and then click the **Debugger** tab, as shown in figure.

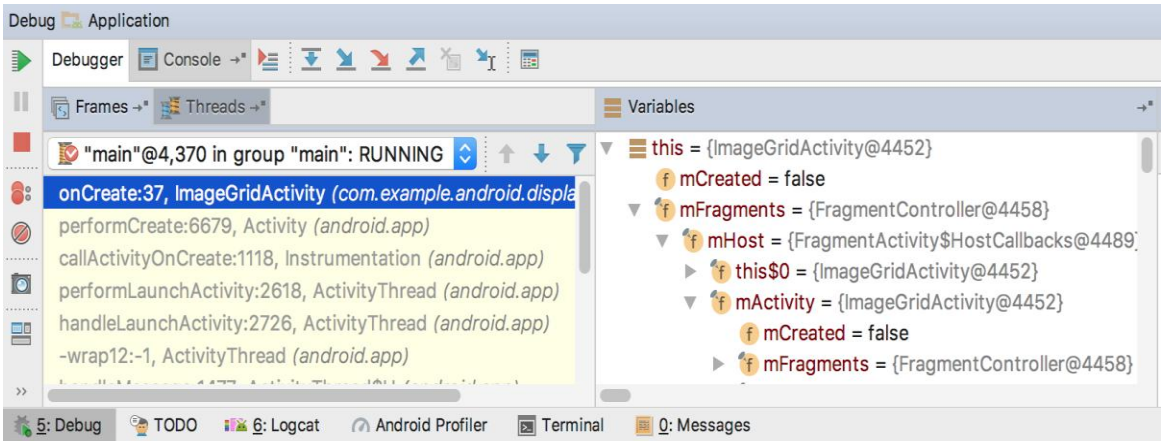


Figure 1. The Debugger window, showing the current thread and the object tree for a variable

OR

The Android Studio Debugger

Similar in name, the Android Studio version is also called the debugger. It allows us to monitor our application in real time, and can provide additional insight regarding a connected device or emulator. Let's walk through using the debugger in Android Studio, to familiarize ourselves with this process and make future bugs easier to track down and address.

RestaurantsActivity.java

```
...
public class RestaurantsActivity extends AppCompatActivity {
  @Bind(R.id.locationTextView) TextView mLocationTextView;
  @Bind(R.id.listView) ListView mListView;

  private String[] restaurants = new String[] {"Sweet Hereafter", "Cricket", "Hawthorne Fish House", "Viking Soul Food",
      "Red Square", "Horse Brass", "Dick's Kitchen", "Taco Bell", "Me Kha Noodle Bar",
      "La Bonita Taqueria", "Smokehouse Tavern", "Pembiche", "Kay's Bar", "Gnarly Grey", "Slappy Cakes", "Mi Mero Mole" };
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_restaurants);
    ButterKnife.bind(this);

    ArrayAdapter adapter = new ArrayAdapter(this, android.R.layout.simple_list_item_1, restaurants);
    mListView.setAdapter(adapter);
    mListView.setOnItemClickListener(new AdapterView.OnItemClickListener() {
      @Override
      public void onItemClick(AdapterView<?> adapterView, View view, int i, long l) {
        String restaurant = ((TextView)view).getText().toString();
        Toast.makeText(RestaurantsActivity.this, restaurant, Toast.LENGTH_LONG).show();
      }
    });
    Intent intent = getIntent();
    String location = intent.getStringExtra("location");
    mLocationTextView.setText("Here are all the restaurants near: " + location);
  }
}
```

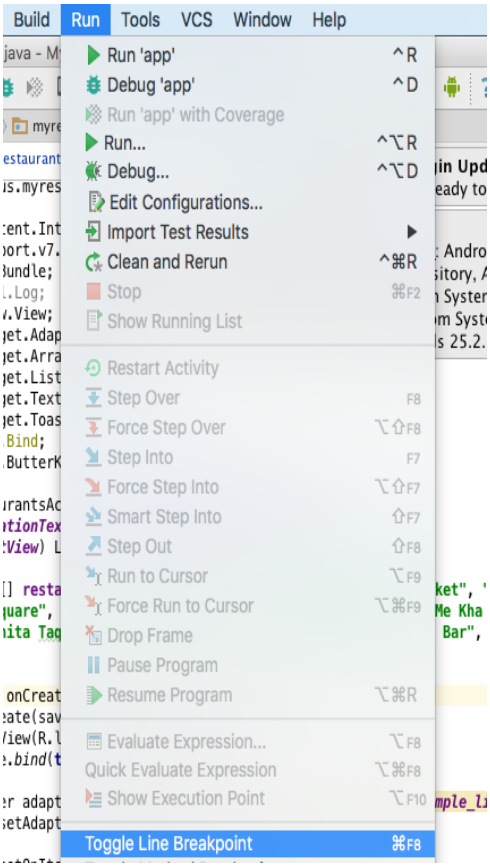
Breakpoints

Next, in order to pause the code at a given location, we need to add something called a breakpoint.
A **breakpoint** is a language-agnostic term meaning the location at which code is intentionally paused.

Placing Breakpoints

We can insert a breakpoint (or even multiple breakpoints) where we would like our code to pause. There are several ways to add a breakpoint.

1. Click the line where you would like to add a breakpoint. Then, select *Run > Toggle Line Breakpoint*.

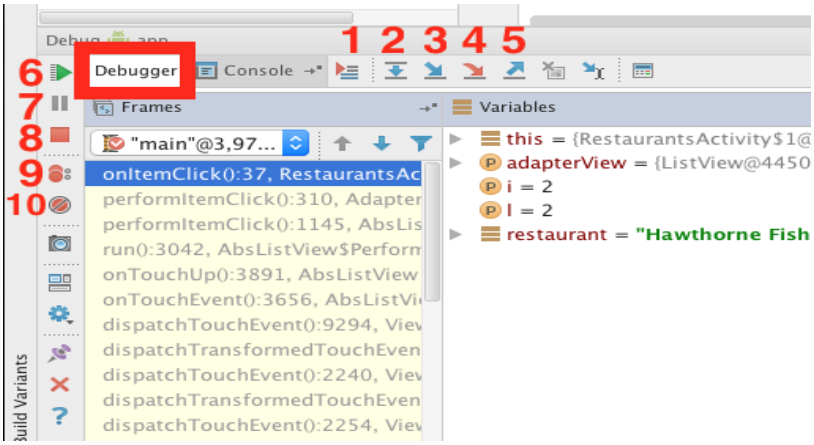


A red circle will appear directly to the left of this line of code. The area this red circle resides in is known as the **gutter**:

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_restaurants);
    ButterKnife.bind(this);

    ArrayAdapter adapter = new ArrayAdapter(this, android.R.layout.simple_list_item_1, resta
    mListView.setAdapter(adapter);

    mListView.setOnItemClickListener((adapterView, view, i, l) → {
            String restaurant = ((TextView)view).getText().toString();
            Toast.makeText(RestaurantsActivity.this, restaurant, Toast.LENGTH_LONG).show();
    });
```

To remove a breakpoint in the same fashion, we can simply select the same line, and then **select *Run > Toggle Line Breakpoint* again to toggle it off.**

The Debugger pane will open in the lower half of the Android Studio window. The debugger contains multiple buttons for interacting with code. Here are the 10 you'll use the most:



1. Show Execution Point

This will place the cursor back to the spot you're currently debugging. (ie: if you insert a breakpoint somewhere, look around in a few other files, you can hit this to return to your original breakpoint).

2. Step Over

This advances to the next line of code *without* entering a method.

3. Step Into

This will advance to the first line of code inside a method call.

4. Force Step Into

This will forcibly advance to the first line of code inside a method call, if the option above does not work for any reason.

5. Step Out

This advances to the next line of code *outside* of the current method.

6. Resume Program

This will continue running the app normally.

7. Pause Program

This will be greyed-out at first, because the program is *already* paused. If you opt to resume the program, you may pause it again with this option.

8. Stop App

This halts the running applicaiton in the emulator or device entirely.

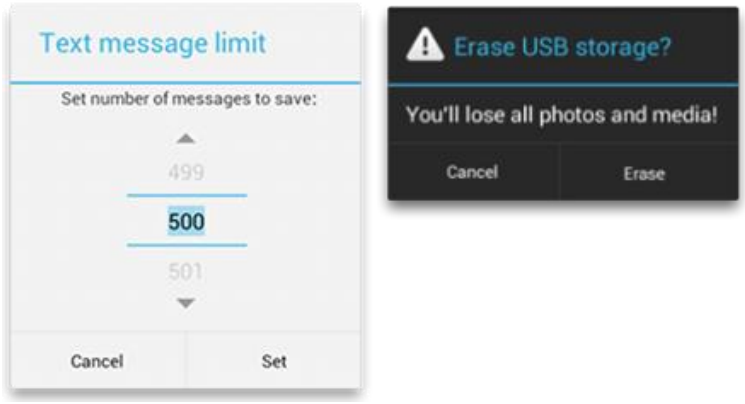
9. View Breakpoints

This will open a window that will summarize exactly which breakpoints have been inserted into what areas of your application. In addition, it will allow you to customize settings for each individual breakpoints. For instance, you can select *Remove once hit* to automatically remove the breakpoint after it pauses your code. *Log message to console* to include a message in the logcat when this breakpoint is hit, or add conditions, log messages, and filters.

Dialogs

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.

Dialog Design



The `Dialog` class is the base class for dialogs, but you should avoid instantiating `Dialog` directly. Instead, use one of the following subclasses:

a. **`AlertDialog`:**
   A dialog that can show a title, up to three buttons, a list of selectable items, or a custom layout.

b. **`DatePickerDialog` or `TimePickerDialog`:**
   A dialog with a pre-defined UI that allows the user to select a date or time.

These classes define the style and structure for your dialog, but you should use a `DialogFragment` as a container for your dialog. The `DialogFragment` class provides all the controls you need to create your dialog and manage its appearance, instead of calling methods on the `Dialog` object.

Creating a Dialog Fragment

You can accomplish a wide variety of dialog designs—including custom layouts and those described in the Dialogs design guide—by extending `DialogFragment` and creating a `AlertDialog` in the `onCreateDialog()` callback method.

```
public class FireMissilesDialogFragment extends
DialogFragment {
    @Override
    public Dialog onCreateDialog(Bundle
savedInstanceState) {
        // Use the Builder class for convenient dialog
construction
        AlertDialog.Builder builder = new
AlertDialog.Builder(getActivity());
builder.setMessage(R.string.dialog_fire_missiles)
               .setPositiveButton(R.string.fire, new
DialogInterface.OnClickListener() {
      public void onClick(DialogInterface dialog, int id)
{                    // FIRE ZE MISSILES!
                }
            })
            .setNegativeButton(R.string.cancel, new
DialogInterface.OnClickListener() {
                public void onClick(DialogInterface
dialog, int id) {
                    // User cancelled the dialog
                }
            });
        // Create the AlertDialog object and return it
        return builder.create();
    }
}
```

Now, when you create an instance of this class and call `show()` on that object, the dialog appears as shown in figure 1.

The next section describes more about using the `AlertDialog.Builder` APIs to create the dialog.

Depending on how complex your dialog is, you can implement a variety of other callback methods in the `DialogFragment`, including all the basic fragment lifecycle methods.

Building an Alert Dialog

The `AlertDialog` class allows you to build a variety of dialog designs and is often the only dialog class you'll need. As shown in figure 2, there are three regions of an alert dialog:

Title
This is optional and should be used only when the content area is occupied by a detailed message, a list, or custom layout. If you need to state a simple message or question (such as the dialog in figure 1), you don't need a title.

Content area
This can display a message, a list, or other custom layout.

Action buttons
There should be no more than three action buttons in a dialog.

To build an `AlertDialog`:

```
// 1. Instantiate an AlertDialog.Builder with its constructor
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());

// 2. Chain together various setter methods to set the dialog characteristics
builder.setMessage(R.string.dialog_message)
       .setTitle(R.string.dialog_title);

// 3. Get the AlertDialog from create()
AlertDialog dialog = builder.create();
```

Adding buttons

To add action buttons like those in figure 2, call the `setPositiveButton()` and `setNegativeButton()` methods:

```
AlertDialog.Builder builder = new AlertDialog.Builder(getActivity());
// Add the buttons
builder.setPositiveButton(R.string.ok, new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
               // User clicked OK button
           } });
builder.setNegativeButton(R.string.cancel, new DialogInterface.OnClickListener() {
           public void onClick(DialogInterface dialog, int id) {
               // User cancelled the dialog
           }
       });
// Set other dialog properties
...
// Create the AlertDialog
AlertDialog dialog = builder.create();
```

The `set...Button()` methods require a title for the button (supplied by a string resource) and a`DialogInterface.OnClickListener` that defines the action to take when the user presses the button.

There are three different action buttons you can add:

1. **Positive:**
   You should use this to accept and continue with the action (the "OK" action).
2. **Negative:**
   You should use this to cancel the action.
3. **Neutral:**
   You should use this when the user may not want to proceed with the action, but doesn't necessarily want to cancel. It appears between the positive and negative buttons. For example, the action might be "Remind me later."

Apart from this, you can use other functions provided by the builder class to customize the alert dialog. These are listed below

Sr.No	Method type & description
1	**setIcon(Drawable icon)** This method set the icon of the alert dialog box.
2	**setCancelable(boolean cancel able)** This method sets the property that the dialog can be cancelled or not
3	**setMessage(CharSequence message)** This method sets the message to be displayed in the alert dialog
4	**setMultiChoiceItems(CharSequence[] items, boolean[] checkedItems, DialogInterface.OnMultiChoiceClickListener listener)** This method sets list of items to be displayed in the dialog as the content. The selected option will be notified by the listener
6	**setTitle(CharSequence title)** This method set the title to be appear in the dialog

Fragments

A **Fragment** is a piece of an activity which enable more modular activity design. It will not be wrong if we say, a fragment is a kind of **sub-activity**.

Following are important points about fragment −

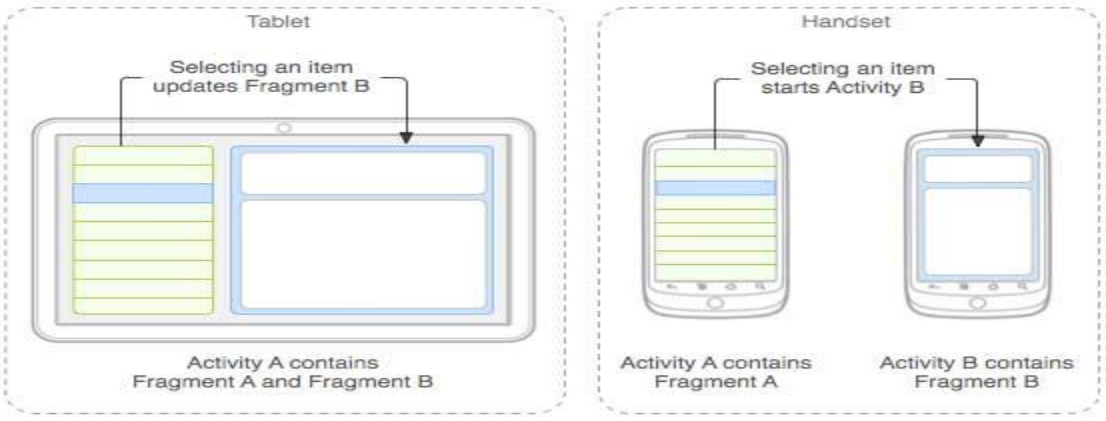
- A fragment has its own layout and its own behaviour with its own life cycle callbacks.
- You can add or remove fragments in an activity while the activity is running.
- You can combine multiple fragments in a single activity to build a multi-pane UI.
- A fragment can be used in multiple activities.
- Fragment life cycle is closely related to the life cycle of its host activity which means when the activity is paused, all the fragments available in the activity will also be stopped.
- A fragment can implement a behaviour that has no user interface component.
- Fragments were added to the Android API in Honeycomb version of Android which API version 11.

You create fragments by extending **Fragment** class and You can insert a fragment into your activity layout by declaring the fragment in the activity's layout file, as a **<fragment>** element.

We had a limitation because we can show only a single activity on the screen at one given point in time. So we were not able to divide device screen and control different parts separately.

But with the introduction of fragment we got more flexibility and removed the limitation of having a single activity on the screen at a time. Now we can have a single activity but each activity can comprise of multiple fragments which will have their own layout.

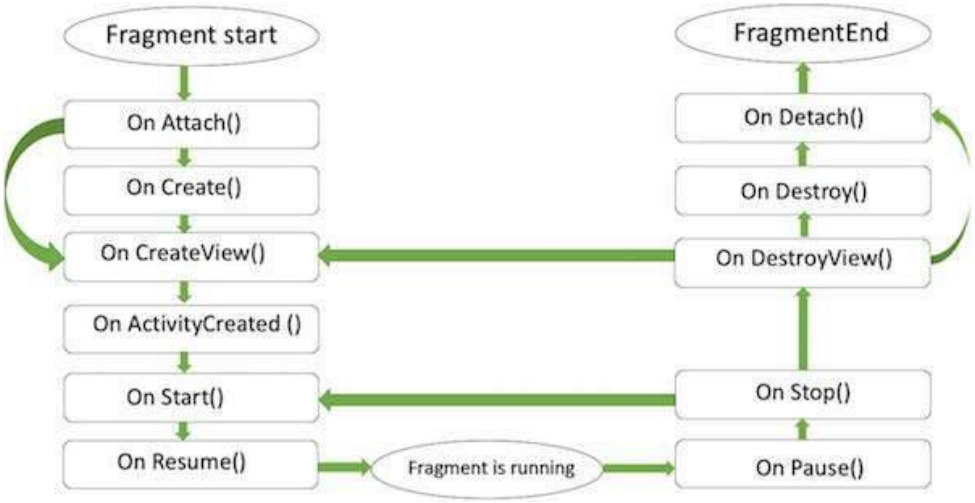
Following is a typical example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.



The application can embed two fragments in Activity A, when running on a tablet-sized device. However, on a handset-sized screen, there's not enough room for both fragments, so Activity A includes only the fragment for the list of articles, and when the user selects an article, it starts Activity B, which includes the second fragment to read the article.

Fragment Life Cycle

Android fragments have their own life cycle very similar to an android activity. This section briefs different stages of its life cycle.



FRAGMENT LIFECYCLE

Here is the list of methods which you can to override in your fragment class −

No	Method	Description
1)	onAttach(Activity)	it is called only once when it is attached with activity.
2)	onCreate(Bundle)	It is used to initialize the fragment.
3)	onCreateView(LayoutInflater, ViewGroup, Bundle)	creates and returns view hierarchy.
4)	onActivityCreated(Bundle)	It is invoked after the completion of onCreate() method.
5)	onViewStateRestored(Bundle)	It provides information to the fragment that all the saved state of fragment view hierarchy has been restored.
6)	onStart()	makes the fragment visible.
7)	onResume()	makes the fragment interactive.

8)	onPause()	is called when fragment is no longer interactive.
9)	onStop()	is called when fragment is no longer visible.
10)	onDestroyView()	allows the fragment to clean up resources.
11)	onDestroy()	allows the fragment to do final clean up of fragment state.
12)	onDetach()	It is called immediately prior to the fragment no longer being associated with its activity.

Types of Fragments

Basically fragments are divided as three stages as shown below.

1. **Single frame fragments** − Single frame fragments are using for hand hold devices like mobiles, here we can show only one fragment as a view.
2. **List fragments** − fragments having special list view is called as list fragment.
3. **Fragments transaction** − Using with fragment transaction. we can move one fragment to another fragment. When a scene changes, a Transition has two main responsibilities −
   - Capture the state of each view in both the start and end scenes.
   - Create an Animator based on the differences that will animate the views from one scene to the other.

Android Fragment Example:

Let's have a look at the simple example of android fragment.

File: activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
   android:layout_width="fill_parent"
   android:layout_height="fill_parent">
    <fragment
      android:id="@+id/fragment1"
      android:name="example.javatpoint.com.fragmentexample.Fragment1"
      android:layout_width="0px"
      android:layout_height="match_parent" >
      <fragment
      android:id="@+id/fragment2"
      android:name="example.javatpoint.com.fragmentexample.Fragment2"
      android:layout_width="0px"
      android:layout_height="match_parent"/>
  </LinearLayout>
```

File: fragment_fragment1.xml

```
<FrameLayout
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#F5F5DC"
    tools:context="example.javatpoint.com.fragmentexample.Fragment1">
     <!--
 TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"

        android:text="@string/hello_blank_fragment" />
  </FrameLayout>
```

File: fragment_fragment2.xml

```
<FrameLayout 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:background="#F0FFFF"
    tools:context="example.javatpoint.com.fragmentexample.Fragment2">
     <!--
 TODO: Update blank fragment layout -->
    <TextView
        android:layout_width="match_parent"
        android:layout_height="match_parent"

        android:text="@string/hello_blank_fragment" />
  </FrameLayout>
```

MainActivity class

File: MainActivity.java

```
package example.javatpoint.com.fragmentexample;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
 public class MainActivity extends AppCompatActivity {
        @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

    }
}
```

File: Fragment1.java

```
package example.javatpoint.com.fragmentexample;
import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
 public class Fragment1 extends Fragment
{
     @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                             Bundle savedInstanceState) {
        // Inflate the layout for this fragment
        return inflater.inflate(R.layout.fragment_fragment1, container, false);
    }
 }
```

File: Fragment2.java

```
package example.javatpoint.com.fragmentexample;
 import android.os.Bundle;
import android.support.v4.app.Fragment;
import android.view.LayoutInflater;
import android.view.View;
import android.view.ViewGroup;
  public class Fragment2 extends Fragment
 {      @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
    }

    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container,
                Bundle savedInstanceState) {
       // Inflate the layout for this fragment
 return inflater.inflate(R.layout.fragment_fragment2, container, false);
    }

}
```

Selecting the Date and Time in One Application:

Android Date Time picker are used a lot in android apps. These components are used to select date and time in a customized user interface. We will use DatePickerDialog and TimePickerDialog classes with Calendar class.

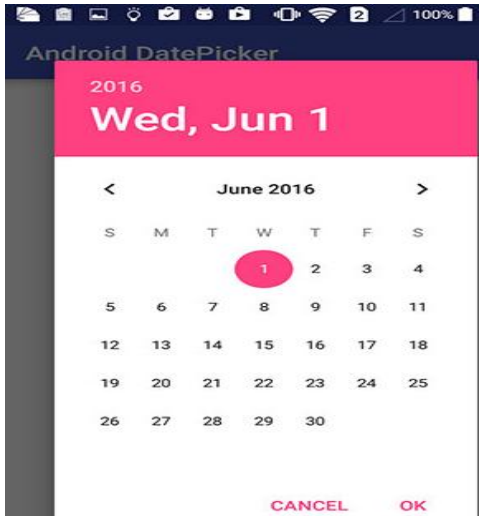
DatePickerDialog and TimePickerDialog

- Though a Date Picker and Time Picker can be used as independent widgets but they occupy more space on the screen.
- Hence using them inside a Dialog is a better choice. Fortunately android provides use with its own DatePickerDialog and TimePickerDialog classes.

DatePickerDialog and **TimePickerDialog** classes have onDateSetListener() and onTimeSetListener()callback methods respectively. These callback methods are invoked when the user is done with filling the date and time respectively.

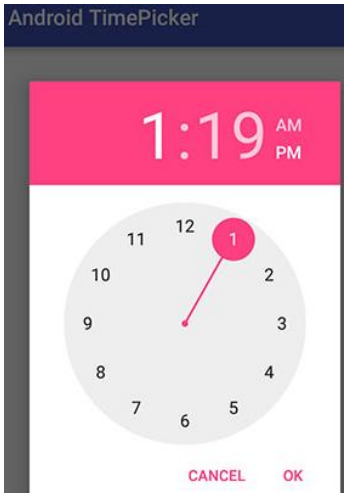
The **DatePickerDialog class consists of a 5 argument constructor** with the parameters listed below.

1. **Context**: It requires the application context
2. **CallBack Function**: onDateSet() is invoked when the user sets the date with the following parameters:
   - int year : It will be store the current selected year from the dialog
   - int monthOfYear : It will be store the current selected month from the dialog
   - int dayOfMonth : It will be store the current selected day from the dialog
3. **int mYear** : It shows the the current year that's visible when the dialog pops up
4. **int mMonth** : It shows the the current month that's visible when the dialog pops up
5. **int mDay** : It shows the the current day that's visible when the dialog pops up



The **TimePickerDialog class consists of a 5 argument constructor** with the parameters listed below.

1. **Context**: It requires the application context
2. **CallBack Function**: onTimeSet() is invoked when the user sets the time with the following parameters:
   - int hourOfDay : It will be store the current selected hour of the day from the dialog
   - int minute : It will be store the current selected minute from the dialog
3. **int mHours** : It shows the the current Hour that's visible when the dialog pops up
4. **int mMinute** : It shows the the current minute that's visible when the dialog pops up
5. **boolean false** : If its set to false it will show the time in 24 hour format else not



Strings.Xml

We are going to update our project strings.xml file located in the values folder inside the res folder. Open the file and add the code below to it.

```
<string name="app_name">Android
TimePicker</string>
<resources>
    <string name="app_name">Android
TimePicker</string>
```

```
    <string name="select_time">Select from time
picker</string>
    <string name="not_text"> </string>
</resources>
```

Colors.Xml

Open the colors.xml file in the same location as the strings.xml file and add the code below to the file.

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#3F51B5</color>
    <color
name="colorPrimaryDark">#303F9F</color>
```

```
    <color name="colorAccent">#FF4081</color>
    <color name="colorBlack">#000000</color>
    <color name="colorDivider">#B6B6B6</color>
</resources>
```

Activity_main.xml

In the main layout of the app, we are going to add a Button and TextView controls. When the button is clicked, it will open a TimePicker dialog that will enable the user to select a time. The selected time will be displayed on the TextView control.

Open the layout file and add the code below to the file.

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context="com.inducesmile.androidtimepicker.MainActivity">
   <Button
      android:id="@+id/select_time"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginTop="40dp"
      android:text="@string/select_time"
      android:padding="16dp"
      android:layout_alignParentTop="true"
      android:layout_centerHorizontal="true"/>
   <TextView
      android:id="@+id/selected_time"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_centerHorizontal="true"
      android:layout_below="@+id/select_time"
      android:textStyle="bold"
      android:layout_marginTop="80dp"
      android:text="@string/not_text" />
</RelativeLayout>
```

Mainactivity.Java

- In the MainActivity, we are going to obtain the reference to the View controls. The next thing is to create an inner Fragment class that will extends the DialogFragment and the class will implement TimePickerDialog.OnTimeSetListener.

- In the button click event method, the object of the TimePickerFragment class will be created and the show() method of the object is called. It takes the FragmentManager and a string unique identifier as parameters.

- Open the MainActivity.java file and add the code below to the class.

```
import android.app.Dialog;
import android.app.DialogFragment;
import android.app.TimePickerDialog;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.text.format.DateFormat;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import java.util.Calendar;
public class MainActivity extends AppCompatActivity {
  protected static TextView displayCurrentTime;
  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
    displayCurrentTime = (TextView)findViewById(R.id.selected_time);
    Button displayTimeButton = (Button)findViewById(R.id.select_time);
    assert displayTimeButton != null;
    displayTimeButton.setOnClickListener(new View.OnClickListener() {
      @Override
      public void onClick(View v) {
        TimePicker mTimePicker = new TimePicker();
        mTimePicker.show(getFragmentManager(), "Select time");
      }
    });
  }
  public static class TimePicker extends DialogFragment implements
TimePickerDialog.OnTimeSetListener {
    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {
      final Calendar c = Calendar.getInstance();
      int hour = c.get(Calendar.HOUR_OF_DAY);
      int minute = c.get(Calendar.MINUTE);
      return new TimePickerDialog(getActivity(), this, hour, minute,
DateFormat.is24HourFormat(getActivity()));
    }
    @Override
    public void onTimeSet(android.widget.TimePicker view, int hourOfDay, int minute) {
      displayCurrentTime.setText("Selected Time: " + String.valueOf(hourOfDay) + " : " +
String.valueOf(minute));
    }
  }
}
```

Datepicker

Activity_Main.Xml
Open the activity_main layout file and add the code below

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"

android:paddingBottom="@dimen/activity_vertical_
margin"

android:paddingLeft="@dimen/activity_horizontal_m
argin"

android:paddingRight="@dimen/activity_horizontal_
margin"

android:paddingTop="@dimen/activity_vertical_marg
in"

tools:context="com.inducesmile.androiddatepicker.M
ainActivity">
```

```
  <Button
     android:id="@+id/select_time"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_marginTop="40dp"
     android:text="@string/select_time"
     android:padding="16dp"
     android:layout_alignParentTop="true"
     android:layout_centerHorizontal="true"/>
  <TextView
     android:id="@+id/selected_time"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:layout_centerHorizontal="true"
     android:layout_below="@+id/select_time"
     android:textStyle="bold"
     android:layout_marginTop="80dp"
     android:text="@string/no_text" />
</RelativeLayout>
```

Date Time Picker Dialog : `activity_main.xml`

```
<RelativeLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   <EditText
      android:layout_width="200dp"
      android:layout_height="wrap_content"
      android:id="@+id/in_date"
      android:layout_marginTop="82dp"
      android:layout_alignParentTop="true"
      android:layout_alignParentLeft="true"
      android:layout_alignParentStart="true" />
   <Button
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="SELECT DATE"
      android:id="@+id/btn_date"
      android:layout_alignBottom="@+id/in_date"
      android:layout_toRightOf="@+id/in_date"
      android:layout_toEndOf="@+id/in_date" />
```

```
    <EditText
        android:layout_width="200dp"
        android:layout_height="wrap_content"
        android:id="@+id/in_time"
        android:layout_below="@+id/in_date"
        android:layout_alignParentLeft="true"
        android:layout_alignParentStart="true" />
     <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="SELECT TIME"
        android:id="@+id/btn_time"
        android:layout_below="@+id/btn_date"
        android:layout_alignLeft="@+id/btn_date"
        android:layout_alignStart="@+id/btn_date" />
</RelativeLayout>
```

The `MainActivity.java` class is given below:

```
package com.journaldev.datetimepickerdialog;

import android.app.DatePickerDialog;
import android.app.TimePickerDialog;
import
android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.DatePicker;
import android.widget.EditText;
import android.widget.TimePicker;
import java.util.Calendar;

public class MainActivity extends
AppCompatActivity implements
      View.OnClickListener {

   Button btnDatePicker, btnTimePicker;
   EditText txtDate, txtTime;
   private int mYear, mMonth, mDay, mHour,
mMinute;
   @Override
   protected void onCreate(Bundle
savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
btnDatePicker=(Button)findViewById(R.id.btn_da
te);
btnTimePicker=(Button)findViewById(R.id.btn_ti
me);
 txtDate=(EditText)findViewById(R.id.in_date);
txtTime=(EditText)findViewById(R.id.in_time);
      btnDatePicker.setOnClickListener(this);
      btnTimePicker.setOnClickListener(this);
   }
   @Override
   public void onClick(View v) {
      if (v == btnDatePicker) {
         // Get Current Date
         final Calendar c = Calendar.getInstance();
         mYear = c.get(Calendar.YEAR);
         mMonth = c.get(Calendar.MONTH);
 mDay = c.get(Calendar.DAY_OF_MONTH);
         DatePickerDialog datePickerDialog = new
DatePickerDialog(this,
               new
DatePickerDialog.OnDateSetListener() {
                  @Override
                  public void onDateSet(DatePicker
view, int year,
                                        int monthOfYear, int
dayOfMonth) {
                     txtDate.setText(dayOfMonth + "-" +
(monthOfYear + 1) + "-" + year);
                  }
               }, mYear, mMonth, mDay);
         datePickerDialog.show();
      }
      if (v == btnTimePicker) {

         // Get Current Time
         final Calendar c = Calendar.getInstance();
         mHour = c.get(Calendar.HOUR_OF_DAY);
         mMinute = c.get(Calendar.MINUTE);

         // Launch Time Picker Dialog
         TimePickerDialog timePickerDialog = new
TimePickerDialog(this,
               new
TimePickerDialog.OnTimeSetListener() {

                  @Override
                  public void onTimeSet(TimePicker
view, int hourOfDay,
                                        int minute) {

                     txtTime.setText(hourOfDay + ":" +
minute);
                  }
               }, mHour, mMinute, false);
         timePickerDialog.show();
      }
   }
}
```

Using ListView:

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

android.widget.ListView is widely used in android mobile app. It displays all the list items in the form of a vertical list.

There are two ways to create a ListView.

1. Use **android.widget.ListView** widget directly.
2. Make your activity class extends **android.app.ListActivity**. Then you can call it's setAdapter() method to set item data, and it's onListItemClick() method is used to response list item click action.

After you get the ListView object, you need to add item data in it through Adapter. There are following Adapters that you can use.

1. ArrayAdapter.
2. SimpleAdapter.
3. BaseAdapter.

You can even use other adapters, but these three adapters are most used.

ListView Properties and Methods:

1. **android:choiceMode** : The value can be singleChoice (can only select one item), multipleChoice (Can select multiple item) or none.
2. **android:divider** : It is a color or drawable object id, used to set divider color.
3. **android:dividerHeight** : Set the divider height.
4. **android:entries** : The value is a array resource id defined in android studio. The array data will be shown in the ListView.
5. **android:headerDividersEnabled** : Boolean value, true means draw divider after header view.
6. **android:footerDividersEnabled** : Boolean value, true means draw divider before footer view.
7. **setOnItemClickListener()**: Set OnItemClickListener object which will response to ListView item click event.

```
<ListView
   android:id="@+id/listViewExample"
   android:layout_width="match_parent"
   android:layout_height="wrap_content"
   android:divider="@android:color/holo_green_light"
   android:dividerHeight="5dp"
   android:clickable="true"
   android:choiceMode="singleChoice" />
```

Inside a ListView, we can show list of Text items by using TextView, or pictures using ImageView, or any other view or a combination of views.

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

Array Adapter

You can use this adapter when your data source is an array. By default, ArrayAdapter creates a view for each array item by calling toString() on each item and placing the contents in a **TextView**. Consider you have an array of strings you want to display in a ListView, initialize a new **ArrayAdapter** using a constructor to specify the layout for each string and the string array –

```
ArrayAdapter adapter = new ArrayAdapter<String>(this,R.layout.ListView,StringArray);
```

Here are arguments for this constructor –

- First argument **this** is the application context. Most of the case, keep it **this**.
- Second argument will be layout defined in XML file and having **TextView** for each string in the array.
- Final argument is an array of strings which will be populated in the text view.

Once you have array adapter created, then simply call **setAdapter()** on your **ListView** object as follows –

```
ListView listView = (ListView) findViewById(R.id.listview);
listView.setAdapter(adapter);
```

src/com.example.ListDisplay/ListDisplay.java. This file can include each of the fundamental life cycle methods.

```
package com.example.ListDisplay;
import android.os.Bundle;
import android.app.Activity;
import android.view.Menu;
import android.widget.ArrayAdapter;
import android.widget.ListView;
public class ListDisplay extends Activity {
  // Array of strings...
  String[] mobileArray =
{"Android","IPhone","WindowsMobile","Blackberry",
    "WebOS","Ubuntu","Windows7","Max OS X"};
    @Override
  protected void onCreate(Bundle
savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
        ArrayAdapter adapter = new
ArrayAdapter<String>(this,R.layout.activity_listview,
mobileArray);
        ListView listView = (ListView)
findViewById(R.id.mobile_list);
    listView.setAdapter(adapter);
  }
}
```

Following will be the content of **res/layout/activity_main.xml** file –

```
<LinearLayout
  android:layout_width="match_parent"
  android:layout_height="match_parent"
  android:orientation="vertical"
  tools:context=".ListActivity" >
  <ListView
    android:id="@+id/mobile_list"
    android:layout_width="match_parent"
    android:layout_height="wrap_content" >
  </ListView>
</LinearLayout>
```

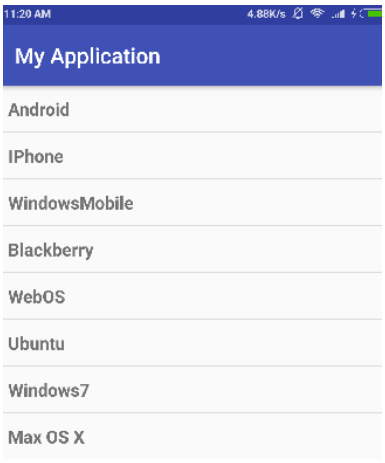
Following will be the content of **res/values/strings.xml** to define two new constants –

```
<resources>
  <string name="app_name">ListDisplay</string>
  <string name="action_settings">Settings</string>
</resources>
```

Following will be the content of **res/layout/activity_listview.xml** file –

```
<TextView
  android:id="@+id/label"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:padding="10dip"
  android:textSize="16dip"
  android:textStyle="bold" >
</TextView>
```

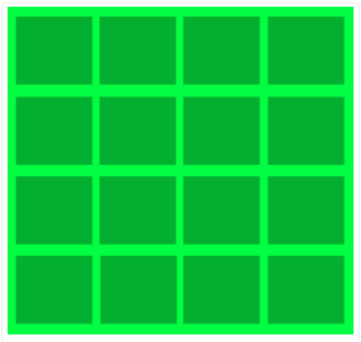
OUTPUT:



Using the GridView Control

Grid view is a layout in android which lets you arrange components in a two-dimensional scrollable grid. Components in a GirdView are not necessarily static, it can be stored in a ListAdapter(Adapter is like a bridge between data and UI components). Following figure shows an example of GridView in android.

```
<GridView
android:id="@+id/gridView1"
android:numColumns="auto_fit"
android:gravity="center"
android:columnWidth="80dp"
android:stretchMode="columnWidth"
android:layout_width="fill_parent"
android:layout_height="fill_parent"
android:numColumns="3">
</GridView>
```



- Android **GridView** shows items in two-dimensional scrolling grid (rows & columns) and the grid items are not necessarily predetermined but they automatically inserted to the layout using a **ListAdapter.**
- An adapter actually bridges between UI components and the data source that fill data into UI Component. Adapter can be used to supply the data to like spinner, list view, grid view etc.
- The **ListView** and **GridView** are subclasses of **AdapterView** and they can be populated by binding them to an **Adapter**, which retrieves data from an external source and creates a View that represents each data entry.

Grid view is very common in applications like calculator, calendar and many others.

Attributes of Grid View

Following are some XML attributes of Android grid view layout.

- **android: gravity,** represents gravity in each cell like center, bottom, top, left etc.
- **android: columnWidth,** used to specify width of column for each cell.
- **android: horizontalSpacing,** specify horizontal space between columns of grid.
- **android: verticalSpacing,** used to specify vertical space between rows of grid.
- **android: numColumns,** specify number of columns to show.

Methods of Grid View

Grid view has many methods to use; some of them are explained here

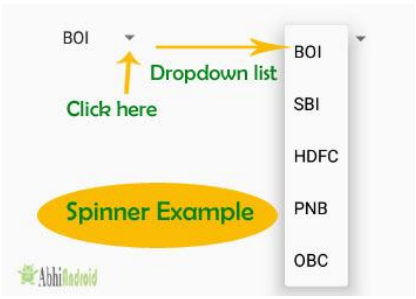
- **getAccessibilityClassName():** return the class name of object.
- **getAdapter():** return adapter associated with it.
- **getColumnWidth():** return width of a column on grid.
- **getGravity():** return gravity of grid component that is how they are aligned horizontally.
- **getHorizontalSpacing():** return horizontal spacing between grid component.
- **getNumColumns():** get number of columns in the grid.
- **setAdapter(ListAdapter adapter):** used to set the data behind grid view.
- **setColumnWidth(int columnWidth):** used to set column width.
- **setGravity(int gravity):** used to set gravity of grid's component.
- **setHorizontalSpacing(int horizontalSpacing):** used to set horizontal spacing to place item in grid.
- **setVerticalSpacing(int verticalSpacing):** used to set vertical spacing to place item on grid.

Following are the code for MainActivity.java

```
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.GridView;
import android.widget.TextView;
import android.widget.Toast;
public class MainActivity extends Activity {
GridView gridView;
static final String[] numbers = new String[] {
"A", "B", "C", "D", "E",
"F", "G", "H", "I", "J",
"K", "L", "M", "N", "O",
"P", "Q", "R", "S", "T",
"U", "V", "W", "X", "Y", "Z", "\n","\n","\n","\n",
"a", "b","c","d","e",
"f","g","h","i","j",
"k","l","m","n","o",
"p","q","r","s","t",
"u","v","w","x","y",
"z"};
@Override
public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);

gridView = (GridView) findViewById(R.id.gridView1);
ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,
android.R.layout.simple_list_item_1, numbers);
gridView.setAdapter(adapter);
gridView.setOnItemClickListener(new AdapterView.OnItemClickListener()
{
public void onItemClick(AdapterView<?> parent, View v,
int position, long id) {
Toast.makeText(getApplicationContext(),
((TextView) v).getText(), Toast.LENGTH_SHORT).show();
} }); } }
```

Output

Using the Spinner control



In Android, Spinner provides a quick way to select one value from a set of values. Android spinners are nothing but the drop down-list seen in other programming languages.

In a default state, a spinner shows its currently selected value. It provides a easy way to select a value from a list of values.

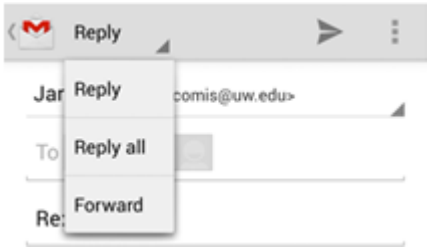
In Simple Words we can say that a spinner is like a combo box of AWT or swing where we can select a particular item from a list of items. Spinner is a sub class of AsbSpinner class.

Important Note: Spinner is associated with Adapter view so to fill the data in spinner we need to use one of the Adapter class.

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

Spinner allows you to select an item from a drop down menu

For example. When you are using Gmail application you would get drop down menu as shown below, you need to select an item from a drop down menu.



File : res/values/strings.xml

```
<resources>
<string
name="app_name">MyAndroidApp</string>
<string name="country_prompt">Choose a
country</string>
<string-array name="country_arrays">
<item>Malaysia</item>
     <item>United States</item>
       <item>New Zealand</item>
      <item>India</item>
    </string-array>
</resources>
```

Open "**res/layout/main.xml**" file, add two spinner components and a button.

1. In "spinner1", the "`android:entries`" represents the selection items in spinner.
2. In "spinner2", the selection items will be defined in code later.

File : res/layout/main.xml

```
<LinearLayout
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
<Spinner
  android:id="@+id/spinner1"
  android:layout_width="match_parent"
android:layout_height="wrap_content"
android:entries="@array/country_arrays"
android:prompt="@string/country_prompt" />
```

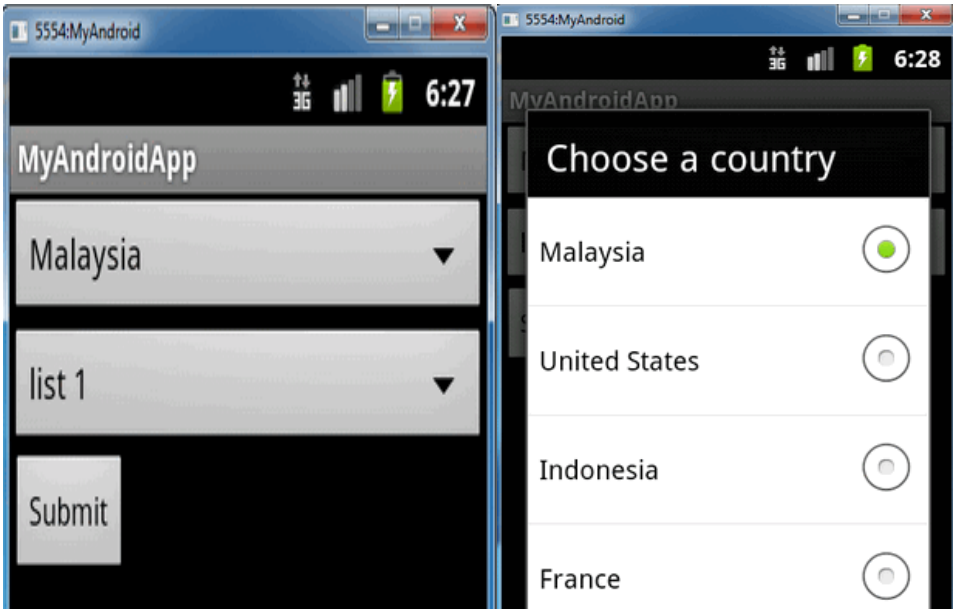
```
<Spinner
  android:id="@+id/spinner2"
android:layout_width="match_parent"
android:layout_height="wrap_content" />
<Button
       android:id="@+id/btnSubmit"
        android:layout_width="wrap_content"
       android:layout_height="wrap_content"
android:text="Submit" />
</LinearLayout>
```

MyAndroidAppActivity.java

```
package com.mkyong.android;
import java.util.ArrayList;
import java.util.List;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.Spinner;
import android.widget.Toast;
public class MyAndroidAppActivity extends Activity {
  private Spinner spinner1, spinner2;
  private Button btnSubmit;
  @Override
  public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        addItemsOnSpinner2();
        addListenerOnButton();
        addListenerOnSpinnerItemSelection();
  }
  // add items into spinner dynamically
  public void addItemsOnSpinner2() {

        spinner2 = (Spinner) findViewById(R.id.spinner2);
        List<String> list = new ArrayList<String>();
        list.add("list 1");
        list.add("list 2");
        list.add("list 3");
        ArrayAdapter<String> dataAdapter = new ArrayAdapter<String>(this,
                android.R.layout.simple_spinner_item, list);
        dataAdapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown
_item);
```

```
spinner2.setAdapter(dataAdapter);
  }
  public void addListenerOnSpinnerItemSelection() {
        spinner1 = (Spinner) findViewById(R.id.spinner1);
        spinner1.setOnItemSelectedListener(new CustomOnItemSelectedListener());
  }
  // get the selected dropdown list value
  public void addListenerOnButton() {

        spinner1 = (Spinner) findViewById(R.id.spinner1);
        spinner2 = (Spinner) findViewById(R.id.spinner2);
        btnSubmit = (Button) findViewById(R.id.btnSubmit);
        btnSubmit.setOnClickListener(new OnClickListener() {
          @Override
          public void onClick(View v) {
            Toast.makeText(MyAndroidAppActivity.this,
                 "OnClickListener : " +
                "\nSpinner 1 : "+ String.valueOf(spinner1.getSelectedItem()) +
                "\nSpinner 2 : "+ String.valueOf(spinner2.getSelectedItem()),
                 Toast.LENGTH_SHORT).show();
          }
        });
  }
}
```

Output



Accessing Databases with the ADB:

Android Debug Bridge (adb)

Android Debug Bridge (adb) is a versatile command-line tool that lets you communicate with a device.

The adb command facilitates a variety of device actions, such as installing and debugging apps, and it provides access to a Unix shell that you can use to run a variety of commands on a device. It is a client-server program that includes three components:

- **A client**, which sends commands. The client runs on your development machine. You can invoke a client from a command-line terminal by issuing an adb command.
- **A daemon (adbd)**, which runs commands on a device. The daemon runs as a background process on each device.
- **A server**, which manages communication between the client and the daemon. The server runs as a background process on your development machine.

Make sure you have the android **adb** tool on your PATH.
Run this command in the terminal: `adb version` If you have adb on your path, it should show something similar to this:
`Android Debug Bridge version 1.0.31`

Accessing the SQLITE database

1. Open a new Terminal window.
2. Enter adb shell
3. Navigate to the folder where your apps databases are:
   cd /data/data/<your app package name>/databases/
4. Type ls to see which database files are present.
5. Open a database file with the sqlite3 tool: sqlite3 database-name.db
6. You can now browse around the database, using the different supported commands.
   To see all commands, type .help
   To list all tables in this database, type .tables To list all rows in a table, type SELECT * FROM tablename;
7. When you are done browsing the database, type .exit to return from the sqlite3 tool.

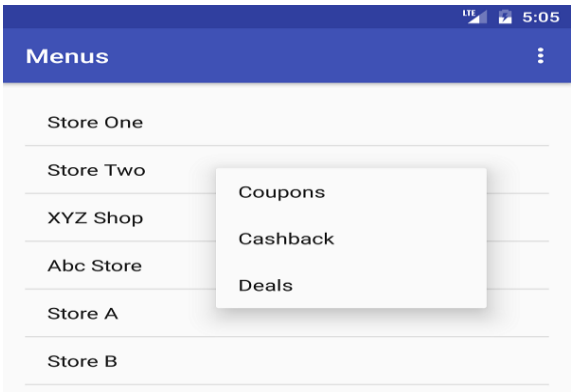
Applying a Context Menu to a List View

- Android context menu appears when user press long click on the element. It is also known as floating menu.
- It affects the selected content while doing action on it. It doesn't support item shortcuts and icons.
- Context menu is one type of menu that can be implemented in android applications to show list of actions to users. Other types of menus which can be provided in android apps are options menu and popup menu.
- Context menus are used to display actions which are related to a specific UI element. Context menus allow users to perform certain actions on selected view.

Context menu can be provided in **two ways, as fixed top bar and as floating list of menu items**.

Floating context menu

To display floating context menu, you need to call activity's registerForContextMenu method. This method registers the view passed to it for context menu.



Drag one listview from the pallete, now the xml file will look like this:

File: activity_main.xml

```
<android.support.constraint.ConstraintLayout
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   tools:context="example.javatpoint.com.contextmenu.MainActivity">
       <ListView
      android:layout_width="368dp"
      android:layout_height="495dp"
      android:id="@+id/listView"
      android:layout_marginEnd="8dp"
      android:layout_marginStart="8dp"
      android:layout_marginTop="8dp"
      app:layout_constraintHorizontal_bias="0.0"
      app:layout_constraintTop_toTopOf="parent" />
</android.support.constraint.ConstraintLayout>
```

Create a separate menu_main.xml file in menu directory for menu items.

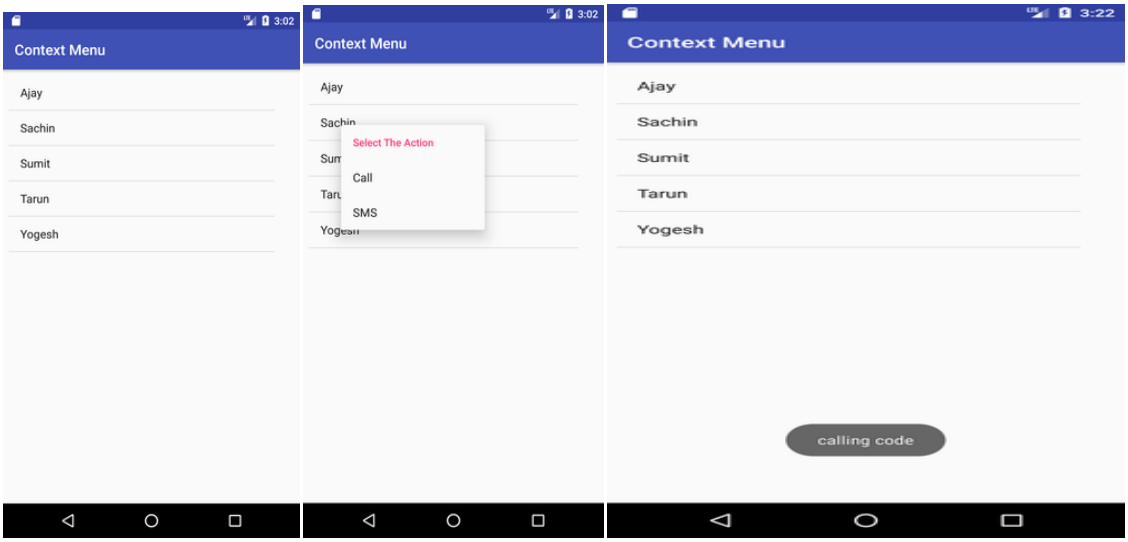
```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/call"
      android:title="Call" />
   <item android:id="@+id/sms"
      android:title="SMS" />
</menu>
```

Let's write the code to display the context menu on press of the listview.

File: MainActivity.java

```
package example.javatpoint.com.contextmenu;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.ContextMenu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

public class MainActivity extends AppCompatActivity

{
   ListView listView;
   String contacts[]={"Ajay","Sachin","Sumit","Tarun","Yogesh"};
   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);
      listView=(ListView)findViewById(R.id.listView);
ArrayAdapter<String> adapter=new ArrayAdapter<String>(this,android.R.layout.simple_list_ite
m_1,contacts);
      listView.setAdapter(adapter);
      // Register the ListView  for Context menu
      registerForContextMenu(listView);
```

```
}
@Override
public void onCreateContextMenu(ContextMenu menu, View v, ContextMenu.ContextMenuInfo
menuInfo)
   {
      super.onCreateContextMenu(menu, v, menuInfo);
      MenuInflater inflater = getMenuInflater();
      inflater.inflate(R.menu.menu_main, menu);
      menu.setHeaderTitle("Select The Action");
   }
   @Override
   public boolean onContextItemSelected(MenuItem item){
      if(item.getItemId()==R.id.call)
{
     Toast.makeText(getApplicationContext(),"calling code",Toast.LENGTH_LONG).show();
      }
      else if(item.getItemId()==R.id.sms)
{
     Toast.makeText(getApplicationContext(),"sending sms code",Toast.LENGTH_LONG).show();

      }else{
         return false;
      }
      return true;
   }

}
```



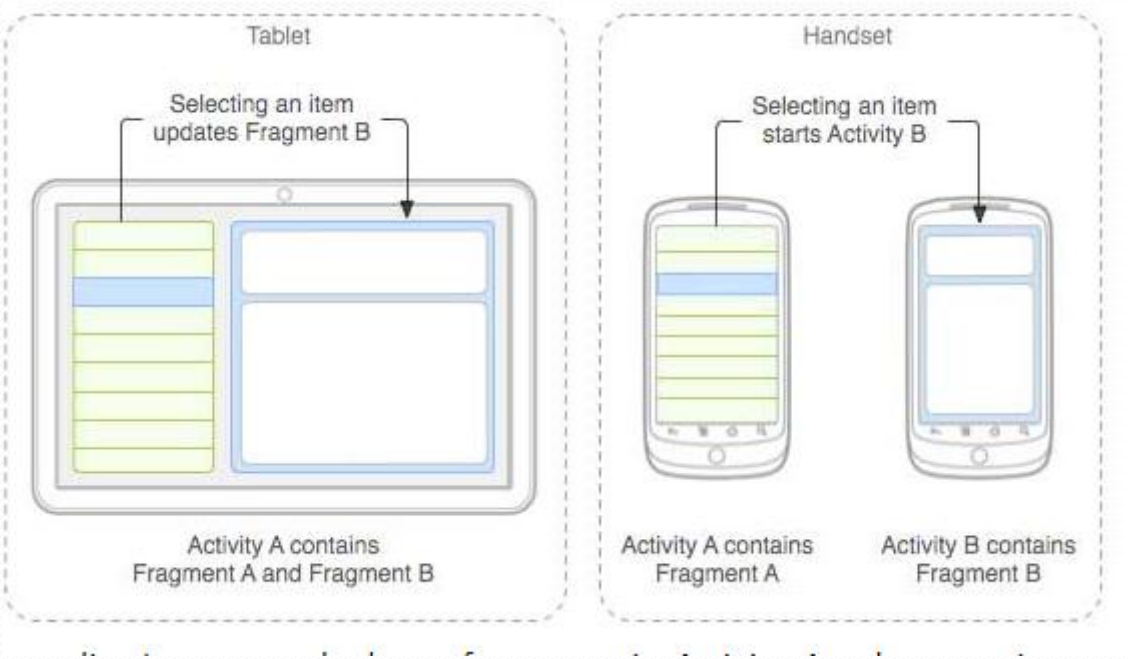
FRAGMENTS

- A **Fragment** is a piece of an activity which enable more modular activity design. It will not be wrong if we say, a fragment is a kind of **sub-activity**.

Following are important points about fragment –

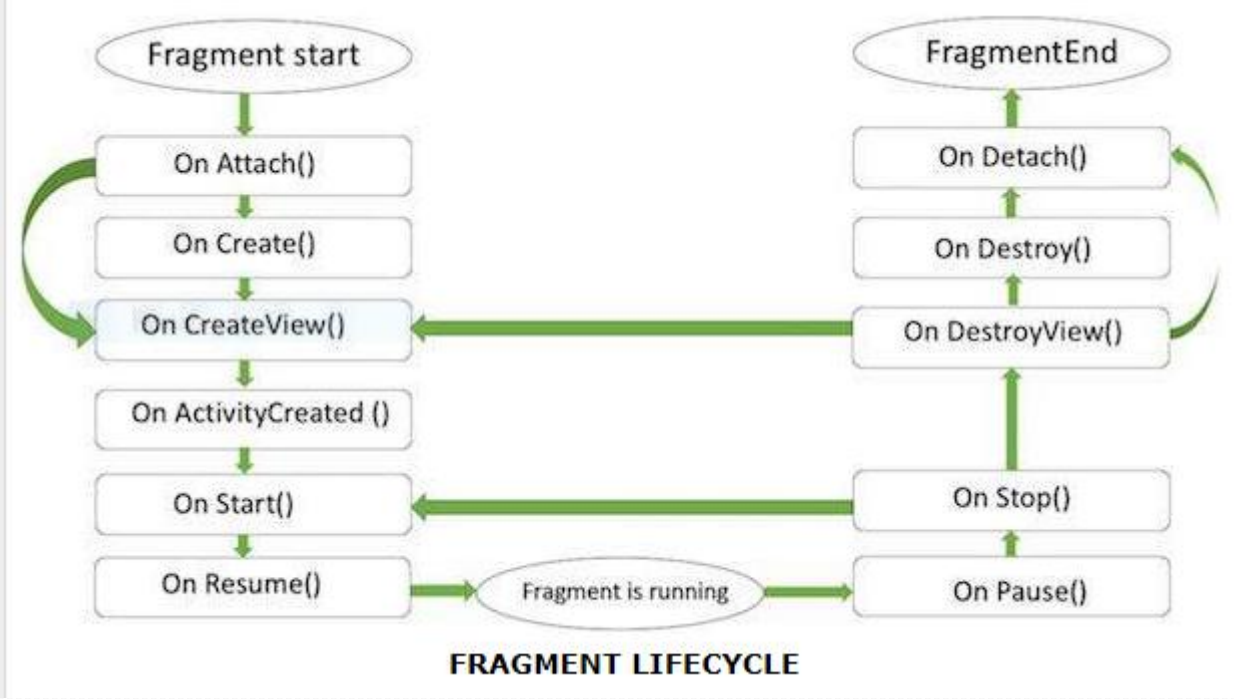
- A fragment has its own layout and its own behaviour with its own life cycle callbacks.
- You can add or remove fragments in an activity while the activity is running.
- You can combine multiple fragments in a single activity to build a multi-pane UI.
- A fragment can be used in multiple activities.
- Fragment life cycle is closely related to the life cycle of its host activity which means when the activity is paused, all the fragments available in the activity will also be stopped.
- A fragment can implement a behaviour that has no user interface component.
- Fragments were added to the Android API in Honeycomb version of Android which API version 11.

- Following is a typical example of how two UI modules defined by fragments can be combined into one activity for a tablet design, but separated for a handset design.



Fragment Life Cycle

- Android fragments have their own life cycle very similar to an android activity. This section briefs different stages of its life cycle.



FRAGMENT LIFECYCLE

How to use Fragments?

- First of all decide how many fragments you want to use in an activity. For example let's we want to use two fragments to handle landscape and portrait modes of the device.
- Next based on number of fragments, create classes which will extend the *Fragment* class. The Fragment class has above mentioned callback functions. You can override any of the functions based on your requirements.
- Corresponding to each fragment, you will need to create layout files in XML file. These files will have layout for the defined fragments.
- Finally modify activity file to define the actual logic of replacing fragments based on your requirement.

Types of Fragments

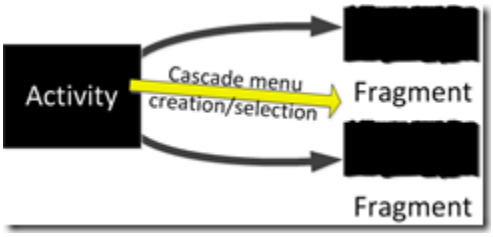
- Single frame fragments – Single frame fragments are using for hand hold devices like mobiles, here we can show only one fragment as a view.
- List fragments – fragments having special list view is called as list fragment
- Fragments transaction – Using with fragment transaction. we can move one fragment to another fragment.

Android Menu Creation

Android menus provide an excellent means to solicit user direction and feedback. From a developer's perspective, what is ideal about menus is that even though there are many different types of menus (see Types of Menus below), menu resources and the Android Menu API that defines menus, sub menus and menu items is the same across menu types. Making it easier to reuse menus in different parts of the application (or even across applications).

However, some of the API with regard to establishment of menus and selection of menu items can be a bit confusing. The creation and selection handling procedures are not the same for the menu types. Different menus get associated with different types of visual components in Android: sometime associated to activities and sometimes associated to view components.

Also, adding to some of the complexity is that activities and fragments can both contribute to the creation and selection of menus. I like to tell my Android students that the creation and selection of menu/menu items is a kind of cascading process. The Activity may start the creation of the menu, but associated Fragments (in order of their association to the Activity) can contribute to the creation of the menu. Likewise, an Activity's callback method is triggered when a menu item is selected, but the associated Fragments can also be allowed to provide a callback method that allows the Fragment to react after the Activity's callback yet before the menu selection is considered fully "handled." So in this way, the creation of menus and reaction to menu item selection cascades down from Activity through to associated Fragments.



In the next two blog posts, I hope do untangle some of the complexity with regard to menu creation and menu item selection. Specifically, in this post, I hope to provide clarity to the menu creation process. In the next post, I'll tackle menu selection.

Types of Menus

First, before looking at menu and menu item creation (and selection in the next post), it is important to understand there are several types of menus in Android. The use and display of menus is also very much

dependent on which version of Android you are using. While menu and menu items can be created programmatically, it is usually easier (and more reusable) to craft menus and menu items from a menu resource file. All the menu types shown below were created from this same menu resource file.

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/option1"
        android:icon="@android:drawable/ic_menu_save"
        android:showAsAction="ifRoom"
        android:title="save"/>
    <item
        android:id="@+id/option2"
        android:icon="@android:drawable/ic_menu_delete"
        android:showAsAction="ifRoom"
        android:title="delete"/>
    <item
        android:id="@+id/option3"
        android:icon="@android:drawable/ic_menu_edit"
        android:showAsAction="ifRoom"
        android:title="edit"/>
    <item
        android:id="@+id/option4"
        android:icon="@android:drawable/ic_menu_search"
        android:showAsAction="ifRoom"
        android:title="search"/>
```

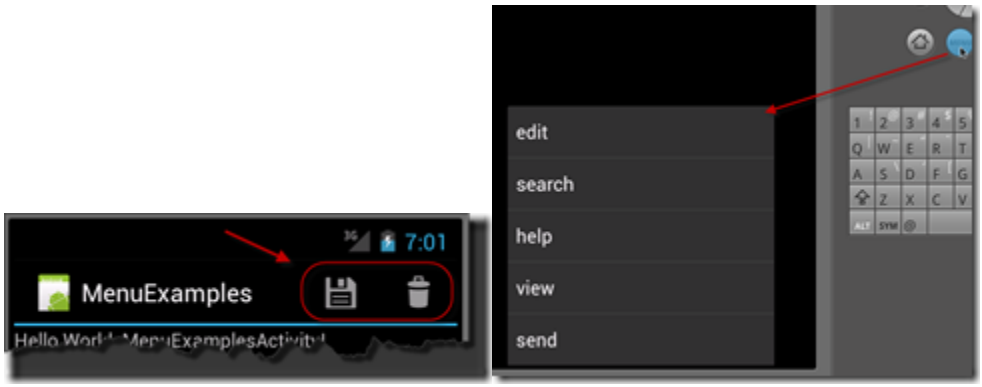
```
    <item
        android:id="@+id/option5"
        android:icon="@android:drawable/ic_menu_help"
        android:showAsAction="ifRoom"
        android:title="help"/>
    <item
        android:id="@+id/option6"
        android:icon="@android:drawable/ic_menu_view"
        android:showAsAction="ifRoom"
        android:title="view"/>
    <item
        android:id="@+id/option6"
        android:icon="@android:drawable/ic_menu_send"
        android:showAsAction="ifRoom"
        android:title="send"/>
</menu>
```

Options Menu

Since the beginning of Android, there have been options menus and context menus. Options menus are one per Activity. The options menu associated with an Activity gets displayed when the user hits the Menu button on the Android device when the associated Activity is the active (i.e. showing) Activity. Up to six menu items can be displayed in an options menu. When there are more than six items, the a More option (like that shown below) is displayed as the sixth item and if selected by the user it causes Android to display a list of other options. As you can see below, menu item icons and title are displayed in an options menu.

Action Bar

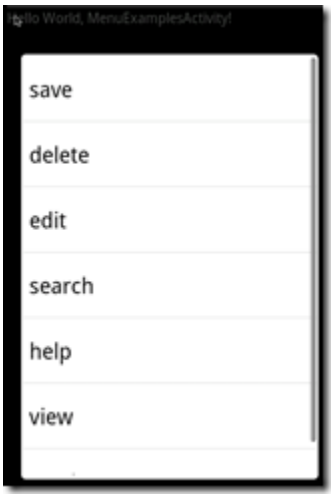
As of Android 3.0, the presence of a Menu button is not guaranteed on devices. That makes the options menu obsolete and it is replaced by the action bar. Fortunately, for developers, the older options menu simply becomes the action bar menu. The same menu resources used in the picture above create the action bar below without any code changes – save a change in the manifest file to indicate an Android 3.0 environment or better is required of the runtime environment.



When there is not room to display all the menu options in the action bar (as shown above), you must go to the "overflow menu" (shown to the right in the pictures above). To get to the overflow menu, press the Menu button on the Android virtual device. On actual devices, a soft options menu button appears as three vertical dots. You can read an earlier blog post **here** to learn about the changes to device buttons and how to develop for the action bar menus.

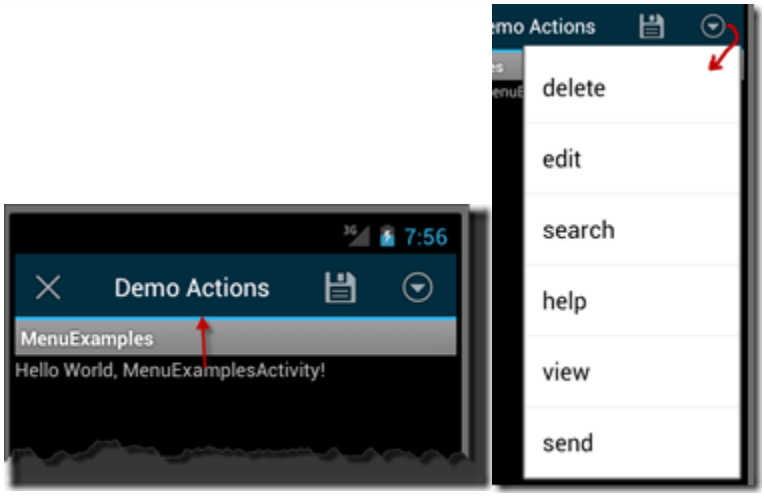
Context Menu

Context menus are attached to View components (widgets) in an Activity. They are displayed when a user long presses on the associated View component. Much of the Android documentation refers to these menus as "floating" menus as they are to float or hover over the associated View when long pressed. The same menu/menu item resources used to create the options menus or action bar above can also be used to display a context menu. Below, a context menu created from the same menu resources is displayed on the long press of an associated TextView widget. Note that the menu items in context menu have text but no icons – another characteristic that separates context menus from options menus or action bars.



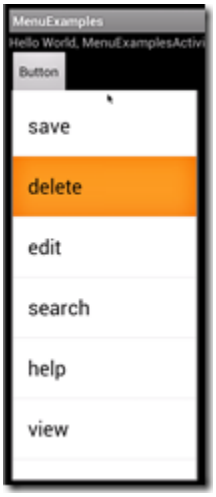
Context Action Bar

Like options menus, context menus have been relegated to the "archive" features list of Android. As of Android 3.0, Android documentation suggests using the context action bar in place of context menus. [**NOTE**: a context action bar is not an action bar. Sound similar and are displayed close to the same area of the screen, but they are separate entities.] A contextual action bar is displayed at the top of the screen in a horizontal bar. Like context menus, the context action bar is associated to a View component and is displayed when the user long presses on the associated View component.. However, it can also be displayed on other View events such as button clicks, check boxes being checked, etc. Developers create menu and menu item resources for the context action mode just as if creating a context menu (or options menu or action bar). A context action bar is shown below (labeled "Demo Actions") using the same menu resource above and activated again with a long press on the TextView widget. When there is not enough room to display all the context actions (as is the case here), then a drop down selector indicates the availability of an overflow menu (shown to the right in the pictures below).



Popup Menus

The last of the menu types is the popup menu. This type of menu was made available in Android 3.0. Popup menu is a modal menu that is anchored on a View – like a Button and triggered by action with that View (such as the Button being pressed). The popup menu gets displayed above or below the View (depending on where there is room on the screen. Below, a popup menu created from the same menu/menu item resources above is displayed below a button when pressed.



You can learn more about the details of the various menus from the Android developer's site **here**.

Menu Creation

Ok, now that you have an appreciation of the different types of menus in Android, how do these menus get created. Well, the answer is "it depends" based on the type of menu.

Creating Option Menus and Action Bars

The options menu and action bar are associated to an Activity. Therefore, simply provide an onCreateOptionsMenu( ) method in your Activity class to create the options menu or action bar. In the example below, it is assumed that the resource menu XML provide earlier in this post is saved in a file /res/menu/my_first_menu.xml. That resource file is "inflated" into a options menu (or action bar) with the use of the Android provided MenuInflater.

```
public class MenuExamplesActivity extends Activity {

    // ... other Activity methods/variables

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

The onCreateOptionsMenu( ) gets called by Android when the options menu is first requested by the user (before Android 3.0) or when the Activity is created (Android 3.0 and newer). The boolean returned by this method indicates whether Android should display the options menu (or action bar). If you detect something in this method (say you determine the access level of the user does not allow for them to use some of the menu options) you can stop the display of the menu by returning false.

While not used as often, you can also use the Menu API (found **here**) to programmatically add items to the menu passed to this method instead of inflating a menu from the XML resource file.

To get an action bar versus an options menu, just make sure your manifest file indicates the minimum SDK in use is Android 3.0 (API level 11).

```
<uses-sdk android:minSdkVersion="11" />
```

Fragments Contribute to Action Bars

Fragments represent a portion of the visual display screen provided by an Activity. Like action bars, fragments were added to the Android API with Android 3.0. Learn more about fragments **here**.

Fragments can also add menu items to the action bar (or otherwise modify the action bar menu items). Without getting too deeply into the API around fragments, suppose you have an Activity that is made up of two fragments (per the layout resource file defined below).

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <fragment
        android:id="@+id/frag1"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:name="com.intertech.FirstFragment" />
    <fragment
        android:id="@+id/frag2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:name="com.intertech.SecondFragment" />
</LinearLayout>
```

When creating the fragments, if the fragment's onCreate( ) method includes a call to setHasOptionsMenu( ) with a parameter value of true, then the fragment's own onCreateOptionsMenu( ) method gets called after owning Activity's onCreateOptonsMenu( ) method to add (or delete or update) menu items from the Activity's action bar.

```
public class FirstFragment extends Fragment {

    // ... other methods/variables

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setHasOptionsMenu(true);
    }
}
```

Here is an example of a fragment's onCreateOptionsMenu( ) method. Notice that this method differs from the Activity version of the method a little in that it is passed a menu inflater. So you do not have to create the menu inflator, simply use it to inflate any additional menu items to the existing menu.

```
public void onCreateOptionsMenu(Menu menu, MenuInflater inflater){
    inflater.inflate(R.menu.my_additional_frag_menu_options, menu);
}
```

If the /res/menu/my_additional_frag_menu_options.xml contained a crop menu item (as shown below), then this menu items would be added to the menu items of the action bar (as shown in the overflow menu picture).

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

```
        android:id="@+id/option6"
        android:icon="@android:drawable/ic_menu_crop"
        android:showAsAction="ifRoom"
        android:title="crop"/>
</menu>
```

If there are multiple fragments associated to the Activity (and each uses setHasOptionsMenu( ) to include additional action bar menu items) then the menu items from the various fragments are added in order of the fragments associated to the Activity.

Creating Context Menus

Context menus are associated with a view component. So inside of the Activity (typically in the creation of the Activity and establishment of its contents), you must register a context menu to a View. The code below attaches a context menu to the long press of a TextView component. Importantly, note the last two lines of the onCreate( ) method of the Activity.

```
public class MenuExamplesActivity extends Activity {

    // ... other methods/variables
```

```
    @Override
    public void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      TextView view = (TextView) findViewById(R.id.myText);
      registerForContextMenu(view);
    }
}
```

Once a context menu has been registered for a View, the Activity must also include a method to create the view when the long press occurs on a View component in the Activity. Provide an onCreateContextMenu( ) to react to a long press and create/display the context menu for any associated View of the Activity. An example is provided below. Notice this method gets passed the View component that was long pressed so that if multiple View components in the Activity are to have context menus, the appropriate context menu can be displayed. In the example below, the assumption is that only one View is registered for a context menu and therefore no such check is necessary.

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
        ContextMenuInfo menuInfo) {
  super.onCreateContextMenu(menu, v, menuInfo);
  MenuInflater inflater = getMenuInflater();
  inflater.inflate(R.menu.my_first_menu, menu);
}
```

Creating a Context Action Bar

The creation of a context action bar in context action mode can perhaps be the most complex process of all Android menu options. It requires the implementation of an ActionMode callback and a listener that triggers

the callback. Typically, this is done with some anonymous inner classes inside of the onCreate( ) method of the Activity.

Below, the example code associates an ActionMode.Callback with the long click listener of the same TextView component as used in the context menu example.

```
public class MenuExamplesActivity extends Activity {

    ActionMode.Callback myCallback;
    ActionMode myActionMode;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView view = (TextView) findViewById(R.id.myText);
        myCallback = new ActionMode.Callback() {
            @Override
            public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
                return false;
            }

            @Override
            public void onDestroyActionMode(ActionMode mode) {
                myActionMode = null;
            }
```

```
        @Override
        public boolean onCreateActionMode(ActionMode mode, Menu menu) {
            mode.setTitle("Demo Actions");
            getMenuInflater().inflate(R.menu.my_first_menu, menu);
            return true;
        }

        @Override
        public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
            return true;
        };
    };

    OnLongClickListener listener = new OnLongClickListener() {
        @Override
        public boolean onLongClick(View v) {
            if (myActionMode != null)
                return false;
            else
                myActionMode = startActionMode(myCallback);
            return true;
        }
    };
```

```
        view.setOnLongClickListener(listener);
    }
}
```

However, note that the ActionMode.Callback could just as easily be associated to any other type of event listener. Therefore the context action bar is one that allows for some dynamic uses and causes for display. Note that the very important creation method associated with the context action bar is the onCreateActionMode( ) method of the callback. This method, like other creation methods you have seen in this blog (notably the options menu and action bar) returns true if the context action bar and menu options are to be displayed or false if the bar should not be displayed.

Creating Popup Menus

Like context menus and often like context action bars, popup menus are associated to a View component. Unlike some of the other menu types, however, popup menus are often created on the fly when the event occurs. For example, if you want to bring up a popup menu when a Button gets pressed, simple define a onClick method or listener that reacts to the button being clicked. Below, a button in an Activity's layout resource is setup to trigger the callPopUp( ) method of the Activity.

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

    <!-- more layout -->

    <Button
        android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
```

```
        android:text="Button" android:onClick="callPopUp"/>

</LinearLayout>
```

Here is an example implementation of the callPopUp( ) method that creates and displays a popup menu. Again, note the use of the MenuInflater to create the menu. Importantly, it is the show( ) method on the popup menu that is used to physically display the popup menu once created.

```
public class MenuExamplesActivity extends Activity {

    // ... other methods/variables

    public void callPopUp(View view) {
        PopupMenu popup = new PopupMenu(this, view);
        MenuInflater inflater = popup.getMenuInflater();
        inflater.inflate(R.menu.my_first_menu, popup.getMenu());
        popup.show();
    }
}
```

Android Menu Selection

Menu resources and the Android Menu API that defines menus, sub menus and menu items is the same across menu types. This makes it easier to reuse menus in different parts of the application (or even across applications). However, while there are many commonalities in how menu selection is handled, the API can sometimes be a bit confusing.

As a matter of review, recall that there are many types of menus in Android.

- Options menus (considered deprecated as of Android 3.0)
- Action bar menus (option menu's replacement as of Android 3.0)
- Context menus (should not be used for Android 3.0 platforms and beyond)
- Context action bar (the context menu replacement)
- Popup menus (new as of Android 3.0).

Now, let's look at how to react to a menu item being selected in each type of menu. Let's also explore the situation where by the menu item selection may be handled by an associated fragment (recall the fragments may, in some menu types, help to create all or part of a menu).

Menu Item Selection

Option Menu and Action Bar Menu Item Selection

Option menus and action bars are associated with an Activity. Therefore, it is in an Activity you must provide a onOptionsItemSelected( ) method.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.option1:
        Log.i("OptionsMenu", "option 1 selected from activity");
        return true;
    //  ... other item selection handling
    default:
```

```
        return super.onOptionsItemSelected(item);
    }
}
```

Notice that this method is passed the menu item selected. In this method, it is up to you to determine which menu item was selected and to take appropriate action. The onOptionsItemSelected( ) method must return a boolean. This boolean is an indication to Android that the selection of the menu item has been dealt with and no further reaction is needed. Why would the method not handle menu item selection you might ask? When fragments are used to help render the display, then handling of the action bar menu item selection can be passed down to the fragment instances for handling as discussed in the next section.

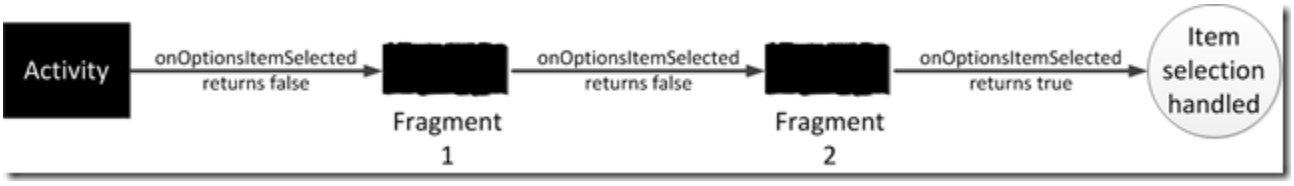
Fragments and Cascading Menu Item Selection

Fragments represent a portion of the visual display screen provided by an Activity. Like action bars, fragments were added to the Android API with Android 3.0. Learn more about fragments **here**. When an Activity's onOptionsItemSelected( ) method returns false, it is an indication that the menu item selection was not handled. This causes Android to pass the item selection event down to any fragment instances that are associated to the Activity. Therefore, fragments too can have an onOptionsItemSelected( ) method to react to menu items begin selected. Here is an example.

```
public class FirstFragment extends Fragment {
    //  ... other methods/variables


    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case R.id.option1:
            Log.i("OptionsMenu", "option 1 selected from frag 1");
            return true;
        //  ... other menu item selection handling
        default:
```

```
            return false;
        }
    }
}
```

You might note that the onOptionsItemSelected( ) method of a fragment also must return a boolean. When there are multiple fragments associated with an Activity, each fragment's onOptionsItemSelected( ) method is called in the order the fragments are associated to the Activity until one method handles the item selection and returns true. In fact, if the Activity and fragment methods all return false, then the handling of the menu item selection is actually passed to the Activity super class. The super class does nothing with the event by default. This theme of handling methods either at the Activity or at the fragment level will be seen in the other menu item selection handling operations below.



Context Menu Bar Item Selection

Handling context menu item selection is very similar to options menu (or action bar) menu item selection. You must provide an onContextItemSelected( ) method in the Activity. Note that even though a context menu is attached to a View component, menu item selection is handled in the Activity. Therefore, the Activity's method must react to all context menu item selections.

```
@Override
public boolean onContextItemSelected(MenuItem item) {
    switch (item.getItemId()) {
    case R.id.option1:
        Log.i("Context Menu", "item 1 selected from context menu");
        return true;
    //  ... handle other context menu item selection
```

```
    default:
        return super.onContextItemSelected(item);
    }
}
```

Notice too that the onContextItemSelected( ) method, like the onOptionsItemSelected( ) method returns a boolean. Again, if this method returns true, it indicates that the Activity handled the item selection. If it returns false, it allows any associated fragment to implement the same method and handle the menu item selection.

Context Action Bar Menu Item Selection

Recall from my last post that in order to provide a context action bar, you must implement an ActionMode Callback and a listener that triggers the callback. The ActionMode Callback is typically done as an anonymous inner class. The implementation of a ActionMode Callback requires that the onActionItemClicked( ) method be implemented. An example is provided below. You can tell that its construction is very similar to the onOptionsItemSelected( ) and onContextItemSelected( ) methods.

```
@Override
public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
    switch (item.getItemId()) {
    case R.id.option1:
        Log.i("content action bar", "item 1 selected");
        mode.finish();
        return true;
    //  ... handle other item selections
    return false;
}
```

Note that the onActionItemClicked( ) method is also passed the ActionMode object. This allows you to optionally dismiss the context action bar (or take other actions with the action bar) after an item has been

selected. Note the call to mode.finish( ) in the code above. In fact, this is the call that dismisses the context action bar.

Popup Menu Item Selection

Handling a popup menu item selection means attaching a OnMenuitemClickListener to the PopupMenu. Usually, the OnMenuItemClickListener is created via anonymous inner class. The OnMenuItemClickListener needs an onMenuItemClick( ) method implementation that provides reaction to any popup menu option being selected. An example is shown below (note lines 7-19).

```
 1: public void callPopUp(View view) {
 2:
 3:     PopupMenu popup = new PopupMenu(this, view);
 4:     MenuInflater inflater = popup.getMenuInflater();
 5:     inflater.inflate(R.menu.my_first_menu, popup.getMenu());
 6:
 7:     popup.setOnMenuItemClickListener(new OnMenuItemClickListener() {
 8:         @Override
 9:         public boolean onMenuItemClick(MenuItem item) {
10:             switch (item.getItemId()) {
11:             case R.id.option1:
12:                 Log.i("OptionsMenu", "option 1 selected from activity");
13:                 return true;
14:             //  ... handle other menu item selections
15:             default:
16:                 return false;
17:             }
18:         }
```

```
19:     });
20:
21:     popup.show();
22: }
```

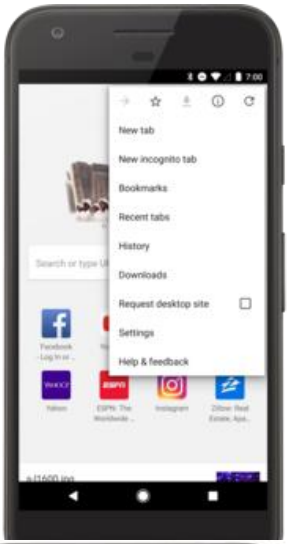
Wrap Up

If you would like to learn more about Android – from Activities, to background Services, to how to use many of the devices capabilities (like telephony and location services), please consider Intertech's Android training.

Menus and Their Types, Creating Menus Through XML, Creating Menus through Coding

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

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



1. **Options menu and app bar**

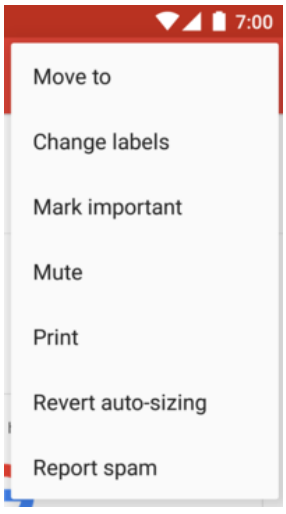
   The options menu is the primary collection of menu items for an activity. It's where you should place actions that have a global impact on the app, such as "Search," "Compose email," and "Settings."

2. **Context menu and contextual action mode**

   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.

   The contextual action mode displays action items that affect the selected content in a bar at the top of the screen and allows the user to select multiple items.

3. **Popup menu**



A popup menu displays a list of items in a vertical list that's anchored to the view that invoked the menu. It's good for providing an overflow of actions that relate to specific content or to provide options for a second part of a command.

Actions in a popup menu should **not** directly affect the corresponding content—that's what contextual actions are for. Rather, the popup menu is for extended actions that relate to regions of content in your activity.

Defining a Menu in XML

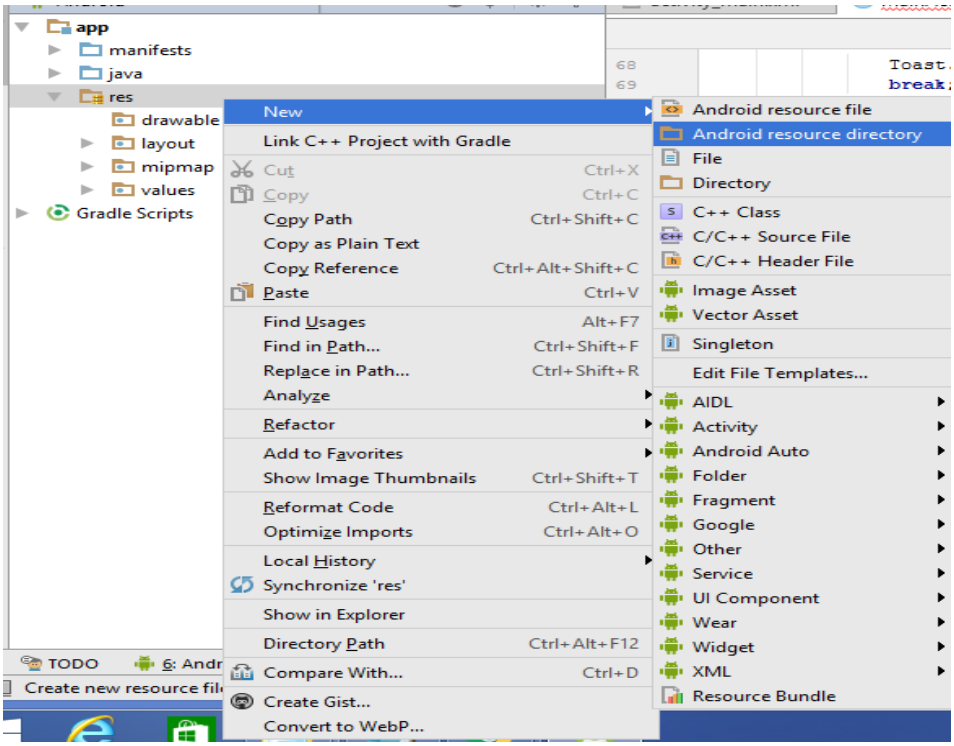
For all menu types, Android provides a standard XML format to define menu items. Instead of building a menu in your activity's code, you should define a menu and all its items in an XML menu resource. You can then inflate the menu resource (load it as a Menu object) in your activity or fragment.

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

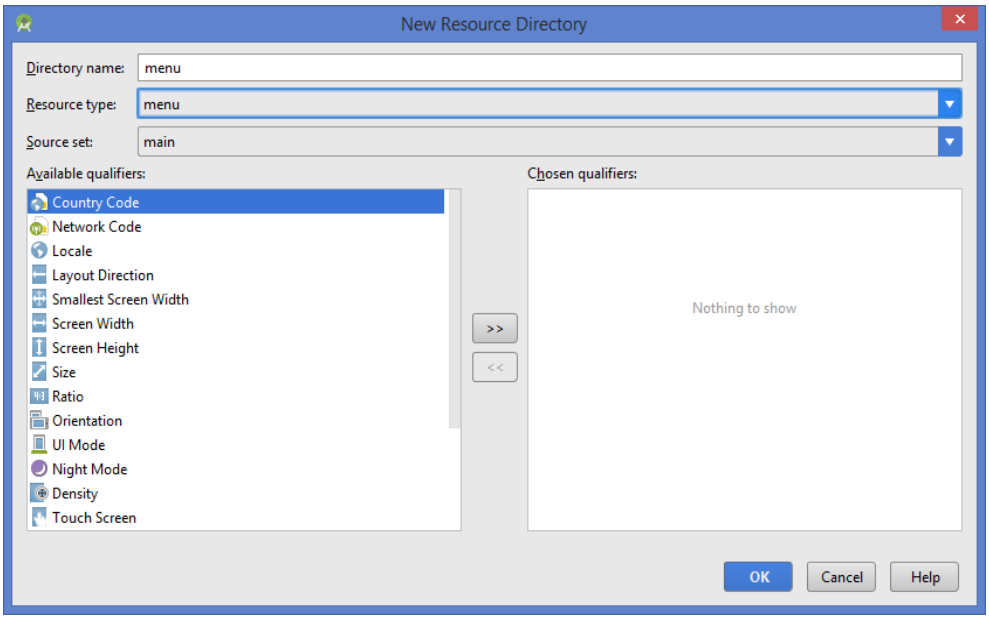
1. It's easier to visualize the menu structure in XML.
2. It separates the content for the menu from your application's behavioral code.
3. It allows you to create alternative menu configurations for different platform versions, screen sizes, and other configurations by leveraging the app resources framework.

How to create a menu_file.xml file in menu directory?

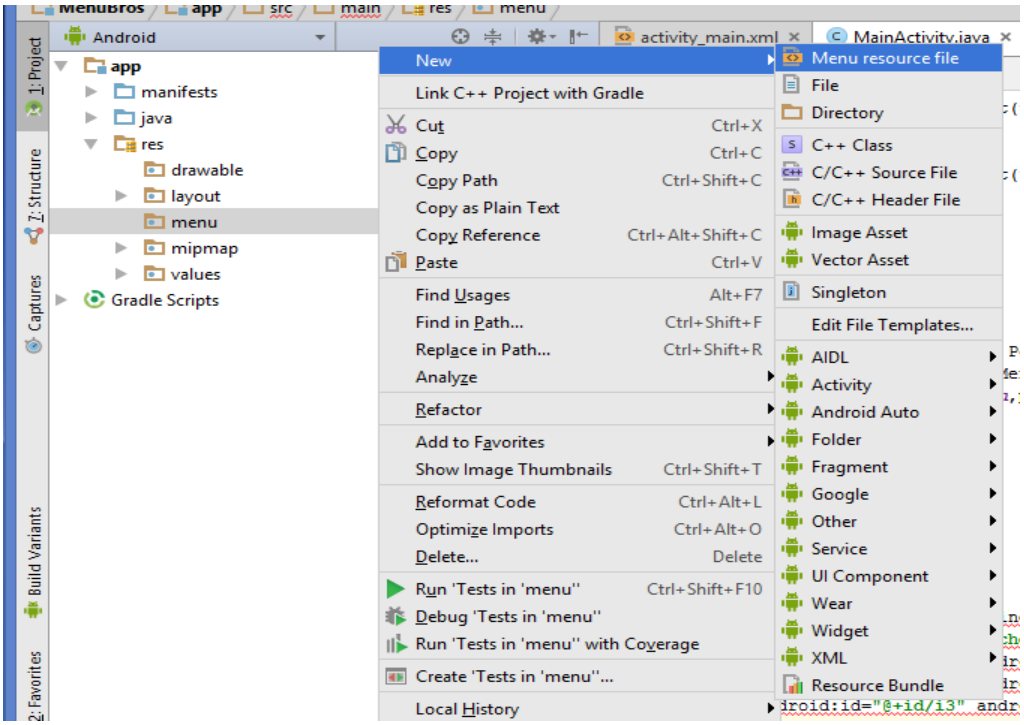
To define the menu_file.xml file, first create a menu directory under res folder. This is done by right clicking on **res --> new --> Android resource directory**.



Then a **new window** will appear. **Type menu in the directory name and choose menu in the Resource type**. Then, click on **OK**.



A new menu directory would be made under res directory. Add menu_file.xml file in menu directory by right clicking on **menu --> New --> Menu resource file**.



To define the menu, create an XML file inside your projects 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. For more information, see the section about Creating Menu Groups.

Here's an example menu named game_menu.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/new_game"
        android:icon="@drawable/ic_new_game"
        android:title="@string/new_game"
        android:showAsAction="ifRoom"/>
   <item android:id="@+id/help"
        android:icon="@drawable/ic_help"
        android:title="@string/help" />
</menu>
```

The <**item**> element supports several attributes you can use to define an item's appearance and behavior. The items in the above menu include the following attributes:

android:id

A resource ID that's unique to the item, which allows the application to recognize the item when the user selects it.

android:icon

A reference to a drawable to use as the item's icon.

android:title

A reference to a string to use as the item's title.

android:showAsAction

Specifies when and how this item should appear as an action item in the app bar.

You can add a submenu to an item in any menu by adding a <menu> element as the child of an <item>.

Submenus are useful when your application has a lot of functions that can be organized into topics, like items in a PC application's menu bar (File, Edit, View, etc.).

For example:

```
<?xml version="1.0" encoding="utf-8"?>
<menu
xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/file"
        android:title="@string/file" >
      <!-- "file" submenu -->
      <menu>
         <item android:id="@+id/create_new"
              android:title="@string/create_new" />
         <item android:id="@+id/open"
              android:title="@string/open" />
      </menu>
   </item>
</menu>
```

Steps for creating option menu

1. create a menu xml
2. Register the menu in Activity
3. code implementation for click on menu items

1. Create xml for menu

In menu folder- create new xml file for menu.
menu.main.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
   <item android:id="@+id/search"
   android:title="Search" />

   <item android:id="@+id/menu_profile"
   android:title="Profile" />

   <item android:id="@+id/setting"
   android:title="Setting" />

   <item android:id="@+id/account"
   android:title="Account" />

</menu>
```

2. Register the menu in Activity

For registering option menu we have to use **public boolean onCreateOptionsMenu(Menu menu)** method.

```
@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;
}
```

3. Code implementation for click on menu items

When you want to action perform on click menu item you have to use **public boolean onOptionsItemSelected(MenuItem item)** method.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {

   int id = item.getItemId();

   switch(id){
      case R.id.search:
         // implement your code
         Toast.makeText(getApplicationContext(),"Search clicked",Toast.LENGTH_LONG).show();

         break;
      case R.id.profile:
         // implement your code
         Toast.makeText(getApplicationContext(),"Profile clicked",Toast.LENGTH_LONG).show();

         break;
      case R.id.setting:
         // implement your code
         Toast.makeText(getApplicationContext(),"Setting clicked",Toast.LENGTH_LONG).show();

         break;
      case R.id.account:
         // implement your code
         Toast.makeText(getApplicationContext(),"Account clicked",Toast.LENGTH_LONG).show();

         break;
      }

   return super.onOptionsItemSelected(item);
}
```

Steps for creating option menu

1. Register the View to which context menu should be associated by calling **registerForContextMenu ()** and pass it the instance of view. like registerForContextMenu(**btnContext**);

2. Implement the onCreateContextMenu() method in your Activity or Fragment. When the registered view get the context event (long press event) , the system calls ***onCreateContextMenu()*** method where you can create context menu item.Like:

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                        ContextMenu.ContextMenuInfo menuInfo) {
   super.onCreateContextMenu(menu, v, menuInfo);
   MenuInflater inflater = getMenuInflater();
   inflater.inflate(R.menu.context_menu, menu);
}
```

3. Implement onContextItemSelected()

Here you can code for click menu item.The getItemId() method queries the ID for the selected menu item, which you should assign to each menu item in XML using the android:id attribute.

```
@Override
public boolean onContextItemSelected(MenuItem item)

{
   AdapterView.AdapterContextMenuInfo info = (AdapterView.AdapterContextMenuInfo) item.getMenuInfo();
   switch (item.getItemId())
{
      case R.id.rename:
        // implement your code
       Toast.makeText(getApplicationContext(),"Setting clicked",Toast.LENGTH_LONG).show();
          return true;
      case R.id.change_background:
          // implement your code
       Toast.makeText(getApplicationContext(),"change background",Toast.LENGTH_LONG).show();
          return true;
      case R.id.setting:
          // implement your code
      Toast.makeText(getApplicationContext(),"Setting clicked",Toast.LENGTH_LONG).show();
          return true;
      case R.id.account:
          // implement your code
        Toast.makeText(getApplicationContext(),"Account clicked",Toast.LENGTH_LONG).show();
          return true;
      default:
          return super.onContextItemSelected(item);
   }
}
```

Steps to create Popup menu

1. Instantiate a **popup** menu with its constructor, which takes the current application **context** and the view to which the menu should be anchored.

2.Use MenuInflater to inflate your menu into the Menu object returned by PopMenu.getMenu().On API leve 14 and above, you can use PopupMenu.inflate() instead.

```
    popupmenu.setOnClickListener (new View.OnClickListener ()
{
       @Override
public void onClick(View v) {
        showPopupMenu(v);
        }
       });
```

3. Call PopupMenu.show () like this

```
private void showPopupMenu(View v)
{
   PopupMenu popupMenu = new PopupMenu(MainActivity.this, v);
   popupMenu.getMenuInflater ().inflate (R.menu.popup_menu, popupMenu.getMenu ());

   popupMenu.setOnMenuItemClickListener (new PopupMenu.OnMenuItemClickListener ()
{
```

```
@Override
 public boolean onMenuItemClick(MenuItem item)
{
        switch (item.getItemId()) {
           case R.id.delete:

              //implement your code
              Toast.makeText(MainActivity.this,"delete clicked",Toast.LENGTH_LONG).show();
              break;
           case R.id.action:
              //implement your code
              Toast.makeText(MainActivity.this,"action clicked",Toast.LENGTH_LONG).show();
             break;

           case R.id.save:
              //implement your code
              Toast.makeText(MainActivity.this,"save clicked",Toast.LENGTH_LONG).show();
              break;
           default:
              break;
        };
        return true;
     }
   });

   popupMenu.show();
}
```

Receiving SMS Messages

We need to have a receiver class that will extend BroadcastReceiver class.

To receive SMS we to declare

```
<uses-permission android:name="android.permission.RECEIVE_SMS"> in manifest file
```

Create a new Class SMSReceiver
In manifest we need to register the SMSReceiver like this

```
<receiver android:name=".SmsReceiver">
        <intent-filter>
          <action android:name=
            "android.provider.Telephony.SMS_RECEIVED" />
        </intent-filter>
 </receiver>
```

Your manifest file should look like this

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="net.learn2develop.SMSMessaging"
    android:versionCode="1"
    android:versionName="1.0.0">
  <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name=".SMS"
            android:label="@string/app_name">
        <intent-filter>
          <action android:name="android.intent.action.MAIN" />
          <category android:name="android.intent.category.LAUNCHER" />
        </intent-filter>
      </activity>
       <receiver android:name=".SmsReceiver">
        <intent-filter>
          <action android:name=
            "android.provider.Telephony.SMS_RECEIVED" />
        </intent-filter>
      </receiver>
    </application>
   <!-- Permission to Receive SMS  -->
   <uses-permission android:name="android.permission.RECEIVE_SMS">
   </uses-permission>
</manifest>
```

SMSReceiver class

```
public class SmsReceiver extends BroadcastReceiver
{
   @Override
   public void onReceive(Context context, Intent intent)
   {
      //---get the SMS message passed in---
      Bundle bundle = intent.getExtras();
      SmsMessage[] msgs = null;
      String messageReceived = "";
      if (bundle != null)
      {
         //---retrieve the SMS message received---
         Object[] pdus = (Object[]) bundle.get("pdus");
         msgs = new SmsMessage[pdus.length];
         for (int i=0; i<msgs.length; i++)

         {
            msgs[i] = SmsMessage.createFromPdu((byte[])pdus[i]);
            messageReceived += msgs[i].getMessageBody().toString();
            messageReceived += "\n";
         }
          //---display the new SMS message---
         Toast.makeText(context, messageReceived, Toast.LENGTH_SHORT).show();

         // Get the Sender Phone Number

         String senderPhoneNumber=msgs[0].getOriginatingAddress ();

      }
   }
}
```

To get the Sender's Number use

```
msgs[0].getOriginatingAddress ()
```

Sending Email

Email is messages distributed by electronic means from one system user to one or more recipients via a network.

- Before starting Email Activity, You must know Email functionality with intent, Intent is carrying data from one component to another component with-in the application or outside the application.
- To send an email from your application, you don't have to implement an email client from the beginning, but you can use an existing one like the default Email app provided from Android, Gmail, Outlook, K-9 Mail etc.
- For this purpose, we need to write an Activity that launches an email client, using an implicit Intent with the right action and data. In this example, we are going to send an email from our app by using an Intent object that launches existing email clients.

We can easily send email in android via intent. You need to write few lines of code only as given below different parts of our Intent object required to send an email.

1. **Intent Object - Action to send Email**

You will use **ACTION_SEND** action to launch an email client installed on your Android device. Following is simple syntax to create an intent with ACTION_SEND action.

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
```

2. **Intent Object - Data/Type to send Email**

To send an email you need to specify **mailto:** as URI using setData() method and data type will be to **text/plain** using setType() method as follows −

```
emailIntent.setData(Uri.parse("mailto:"));
emailIntent.setType("text/plain");
```

3. **Intent Object - Extra to send Email**

Android has built-in support to add TO, SUBJECT, CC, TEXT etc. fields which can be attached to the intent before sending the intent to a target email client.

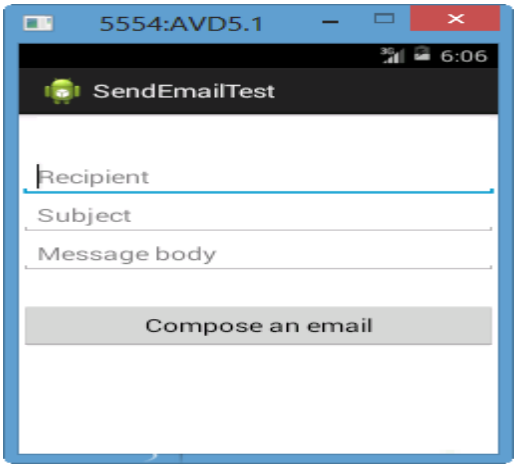
You can use following extra fields in your email −

Sr.No.	Extra Data & Description
1	EXTRA_BCC: A String[] holding e-mail addresses that should be blind carbon copied.
2	EXTRA_CC : A String[] holding e-mail addresses that should be carbon copied.
3	EXTRA_EMAIL: A String[] holding e-mail addresses that should be delivered to.
4	EXTRA_HTML_TEXT: A constant String that is associated with the Intent, used with ACTION_SEND to supply an alternative to EXTRA_TEXT as HTML formatted text.
5	EXTRA_SUBJECT: A constant string holding the desired subject line of a message.
6	EXTRA_TEXT :A constant CharSequence that is associated with the Intent, used with ACTION_SEND to supply the literal data to be sent.
7	EXTRA_TITLE: A CharSequence dialog title to provide to the user when used with a ACTION_CHOOSER.

Here is an example showing you how to assign extra data to your intent −

```
emailIntent.putExtra(Intent.EXTRA_EMAIL  , new String[]{"Recipient"});
emailIntent.putExtra(Intent.EXTRA_SUBJECT, "subject");
emailIntent.putExtra(Intent.EXTRA_TEXT   , "Message Body");
```

The out-put of above code is as below shown an image



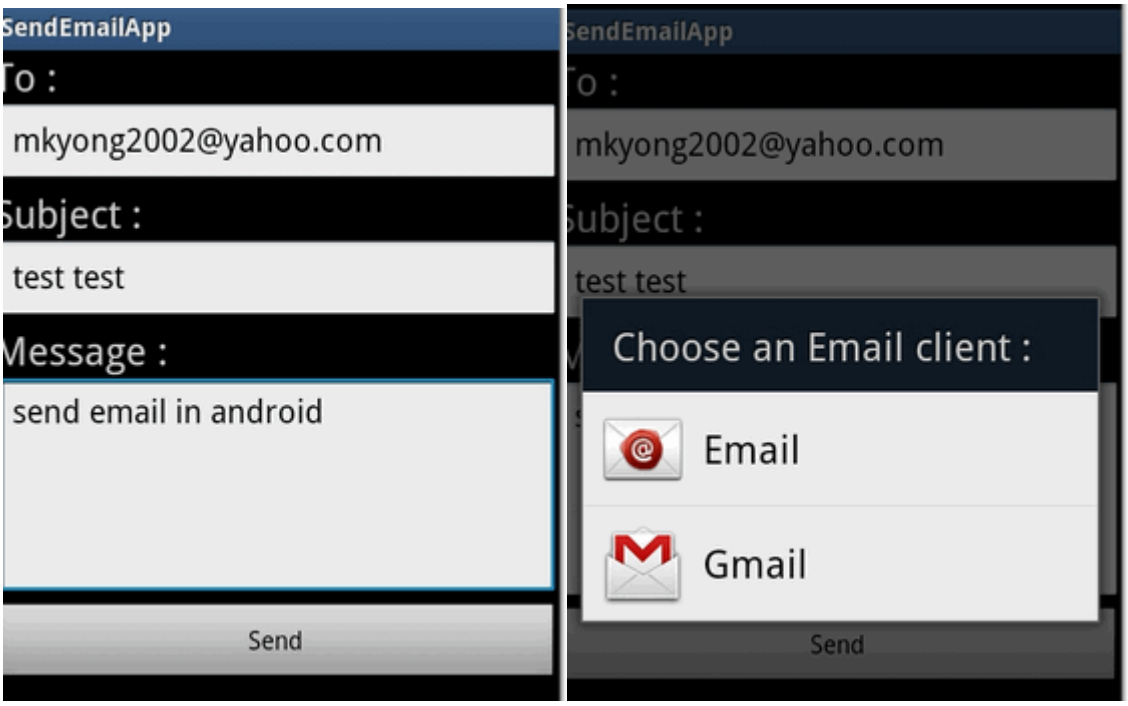
File : res/layout/main.xml

```
<LinearLayout
    android:id="@+id/linearLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >
    <TextView
        android:id="@+id/textViewPhoneNo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="To : "
        android:textAppearance="?android:attr/textAppearanceLarge" />
    <EditText
        android:id="@+id/editTextTo"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="textEmailAddress" >
        <requestFocus />
    </EditText>
    <TextView
        android:id="@+id/textViewSubject"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Subject : "
        android:textAppearance="?android:attr/textAppearanceLarge" />
    <EditText
        android:id="@+id/editTextSubject"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
    </EditText>
    <TextView
        android:id="@+id/textViewMessage"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Message : "
        android:textAppearance="?android:attr/textAppearanceLarge" />
    <EditText
        android:id="@+id/editTextMessage"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="top"
        android:inputType="textMultiLine"
        android:lines="5" />
    <Button
        android:id="@+id/buttonSend"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Send" />
</LinearLayout>
```

Full activity class to send an Email. Read the `onClick()` method, it should be self-explanatory.

```
package com.mkyong.android;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

public class SendEmailActivity extends Activity {

	Button buttonSend;
	EditText textTo;
	EditText textSubject;
	EditText textMessage;

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.main);
		buttonSend = (Button) findViewById(R.id.buttonSend);
		textTo = (EditText) findViewById(R.id.editTextTo);
		textSubject = (EditText) findViewById(R.id.editTextSubject);
		textMessage = (EditText) findViewById(R.id.editTextMessage);

		buttonSend.setOnClickListener(new OnClickListener() {
		public void onClick(View v) {

				String to = textTo.getText().toString();
				String subject = textSubject.getText().toString();
				String message = textMessage.getText().toString();

				Intent email = new Intent(Intent.ACTION_SEND);
				email.putExtra(Intent.EXTRA_EMAIL, new String[]{ to});
				//email.putExtra(Intent.EXTRA_CC, new String[]{ to});
				//email.putExtra(Intent.EXTRA_BCC, new String[]{to});
				email.putExtra(Intent.EXTRA_SUBJECT, subject);
				email.putExtra(Intent.EXTRA_TEXT, message);

				//need this to prompts email client only
				email.setType("message/rfc822");
startActivity(Intent.createChooser(email, "Choose an Email client :"));
		}		});}}
```

Output


SendEmailApp
To :
mkyong2002@yahoo.com
Subject :
test test
Message :
send email in android
Send
SendEmailApp
To :
mkyong2002@yahoo.com
Subject :
test test
Choose an Email client :
Email
Gmail
Send

Sending SMS

In Android, you can use `SmsManager` API or device's `Built-in SMS` application to send a SMS message.

SmsManager API

```
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage("phoneNo", null, "sms message", null, null);
```

Built-in SMS application

```
Intent sendIntent = new Intent(Intent.ACTION_VIEW);
sendIntent.putExtra("sms_body", "default content");
sendIntent.setType("vnd.android-dir/mms-sms");
startActivity(sendIntent);
```

Of course, both need **SEND_SMS permission**.

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

Apart from the above method, there are few other important functions available in SmsManager class. These methods are listed below −

Sr.No.	Method & Description
1	ArrayList<String> divideMessage(String text) This method divides a message text into several fragments, none bigger than the maximum SMS message size.
2	static SmsManager getDefault() This method is used to get the default instance of the SmsManager
3	void sendDataMessage(String destinationAddress, String scAddress, short destinationPort, byte[] data, PendingIntent sentIntent, PendingIntent deliveryIntent) This method is used to send a data based SMS to a specific application port.
4	void sendMultipartTextMessage(String destinationAddress, String scAddress, ArrayList<String> parts, ArrayList<PendingIntent> sentIntents, ArrayList<PendingIntent> deliveryIntents) Send a multi-part text based SMS.
5	void sendTextMessage(String destinationAddress, String scAddress, String text, PendingIntent sentIntent, PendingIntent deliveryIntent) Send a text based SMS.

Android layout file to textboxes (phone no, sms message) and button to send the SMS message.

File : res/layout/main.xml

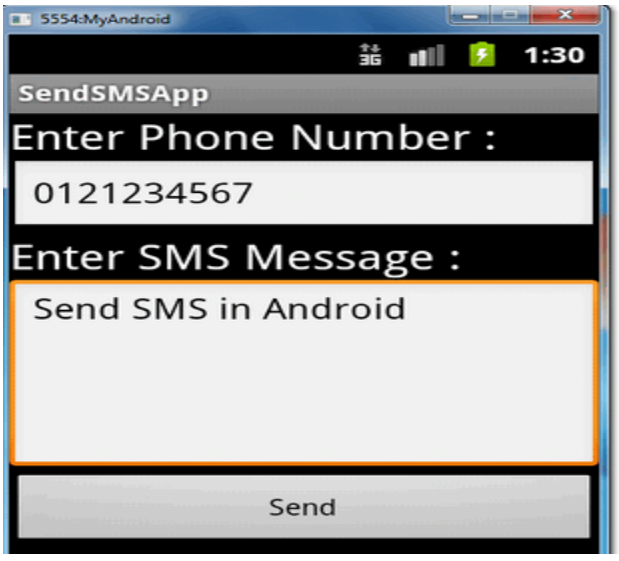
```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/linearLayout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical" >

    <TextView
        android:id="@+id/textViewPhoneNo"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Enter Phone Number : "
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <EditText
        android:id="@+id/editTextPhoneNo"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:phoneNumber="true" >
    </EditText>

    <TextView
        android:id="@+id/textViewSMS"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Enter SMS Message : "
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <EditText
        android:id="@+id/editTextSMS"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:inputType="textMultiLine"
        android:lines="5"
        android:gravity="top" />

    <Button
        android:id="@+id/buttonSend"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Send" />

</LinearLayout>
```

***File : SendSMSActivity.java* – Activity to send SMS via `SmsManager`.**

```
package com.mkyong.android;
import android.app.Activity;
import android.os.Bundle;
import android.telephony.SmsManager;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class SendSMSActivity extends Activity {
        Button buttonSend;
        EditText textPhoneNo;
        EditText textSMS;
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
                buttonSend = (Button) findViewById(R.id.buttonSend);
                textPhoneNo = (EditText) findViewById(R.id.editTextPhoneNo);
                textSMS = (EditText) findViewById(R.id.editTextSMS);
                buttonSend.setOnClickListener(new OnClickListener() {
                @Override
        public void onClick(View v)
{
String phoneNo = textPhoneNo.getText().toString();
String sms = textSMS.getText().toString();
Try
{
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage(phoneNo, null, sms, null, null);
Toast.makeText(getApplicationContext(), "SMS Sent!",Toast.LENGTH_LONG).show();
                        }
catch (Exception e)
{
Toast.makeText(getApplicationContext(),"SMS faild, please try again
later!",Toast.LENGTH_LONG).show();
        e.printStackTrace();
                        }
                    }
                });
        }
}
```

File : AndroidManifest.xml , need **SEND_SMS** permission.

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

    <uses-sdk android:minSdkVersion="10" />

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

    <application
        android:debuggable="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name" >
        <activity
            android:label="@string/app_name"
            android:name=".SendSMSActivity" >
            <intent-filter >
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```



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.

- 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

android.database.sqlite

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

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

Sr.No	Method & Description
1	**openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler)**
2	**openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags)**
3	**openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory)**
4	**openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory)**

Database - Insertion

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


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

Database - Fetching

```
Cursor resultSet = mydatbase.rawQuery("Select * from employee",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.

Sr.No	Method & Description
1	getColumnCount() This method return the total number of columns of the table.
2	getColumnIndex(String columnName) This method returns the index number of a column by specifying the name of the column
3	getColumnName(int columnIndex) This method returns the name of the column by specifying the index of the column
4	getColumnNames() This method returns the array of all the column names of the table.
5	getCount() This method returns the total number of rows in the cursor
6	getPosition() This method returns the current position of the cursor in the table
7	isClosed() This method returns true if the cursor is closed and return false otherwise

SQLiteOpenHelper

It is a class found inside android.database.sqlite package. It is a helper class that helps in creating the database, handling the operations and also the version management.

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) {}
}
```

onCreate() and **onUpgrade()** methods of SQLiteOpenHelper class.

- There are two constructors of SQLiteOpenHelper class.

Constructor	Description
SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)	creates an object for creating, opening and managing the database.
SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version, DatabaseErrorHandler errorHandler)	creates an object for creating, opening and managing the database. It specifies the error handler.

Methods of SQLiteOpenHelper class

Method	Description
public abstract void onCreate(SQLiteDatabase db)	called only once when database is created for the first time.
public abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)	called when database needs to be upgraded.
public synchronized void close()	closes the database object.
public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion)	called when database needs to be downgraded.

Menus and Their Types

- Menus are a common user interface component in many types of applications. To provide a familiar and consistent user experience,

1.Options menu and app bar

- 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."

2.Context menu and contextual action mode

- 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.

- **Popup menu**
  - A popup menu displays a list of items in a vertical list that's anchored to the view that invoked the menu.

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/new_game"
          android:icon="@drawable/ic_new_game"
          android:title="@string/new_game"
          android:showAsAction="ifRoom"/>
    <item android:id="@+id/help"
          android:icon="@drawable/ic_help"
          android:title="@string/help" />

</menu>
```

```xml
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/file"
          android:title="@string/file" >
        <!-- "file" submenu -->
        <menu>
            <item android:id="@+id/create_new"
                  android:title="@string/create_new" />
            <item android:id="@+id/open"
                  android:title="@string/open" />
        </menu>
    </item>
</menu>
```

showAsAction=”never”

- **always** - will always show in the action bar
- **never** - will never show, and therefore will be available through the overflow menu
- **ifRoom** - only if there is sufficient space in the action bar, then it would be shown. Keep in mind that per the documentation, there is a limit to how many icons you can have on the action bar.
- **withText**-will include the item’s title in the action bar
- **collapseActionView** - if this item has an action view associated with it, it will become collapsible(from API 14 and above)

To define the menu, create an XML file inside your projects 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. For more information, see the section about Creating Menu Groups.

Create xml for menu

```
<?xml version="1.0" encoding="UTF-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item android:id="@+id/search"
    android:title="Search" />

    <item android:id="@+id/menu_profile"
    android:title="Profile" />

    <item android:id="@+id/setting"
    android:title="Setting" />

    <item android:id="@+id/account"
    android:title="Account" />

</menu>
```

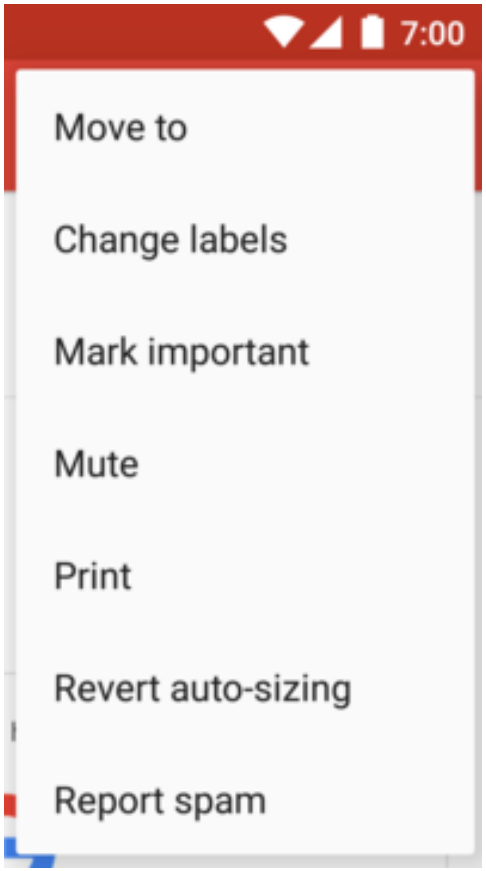
2. Register the menu in Activity

```
@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) {

   int id = item.getItemId();

   switch(id){
      case R.id.search:
         // implement your code
         Toast.makeText(getApplicationContext(),"Search clicked",Toast.LENGTH_LONG).show();

         break;
      case R.id.profile:
         // implement your code
         Toast.makeText(getApplicationContext(),"Profile clicked",Toast.LENGTH_LONG).show();

         break;
      case R.id.setting:
         // implement your code
         Toast.makeText(getApplicationContext(),"Setting clicked",Toast.LENGTH_LONG).show();

         break;
      case R.id.account:
         // implement your code
         Toast.makeText(getApplicationContext(),"Account clicked",Toast.LENGTH_LONG).show();

         break;
      }

   return super.onOptionsItemSelected(item);
}
```


New tab
New incognito tab
Bookmarks
Recent tabs
History
Downloads
Request desktop site
Settings
Help & feedback


SHAKESPEARES
Image result for Shakespeare
Download image
Open image in new tab
Search Google for this image
Share image
William Shakespeare - Wikipedia


7:00
Move to
Change labels
Mark important
Mute
Print
Revert auto-sizing
Report spam

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	openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags, DatabaseErrorHandler errorHandler) This method only opens the existing database with the appropriate flag mode. The common flags mode could be OPEN_READWRITE OPEN_READONLY
2	openDatabase(String path, SQLiteDatabase.CursorFactory factory, int flags) It is similar to the above method as it also opens the existing database but it does not define any handler to handle the errors of databases
3	openOrCreateDatabase(String path, SQLiteDatabase.CursorFactory factory) It not only opens but create the database if it not exists. This method is equivalent to openDatabase method.
4	openOrCreateDatabase(File file, SQLiteDatabase.CursorFactory factory) This method is similar to above method but it takes the File object as a path rather then a string. It is equivalent to file.getPath()

Database - Insertion

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

```
mydatabase.execSQL("CREATE TABLE IF NOT EXISTS TutorialsPoint(Username VARCHAR,Password VARCHAR);");
mydatabase.execSQL("INSERT INTO TutorialsPoint VALUES('admin','admin');");
```

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

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

Database - Fetching

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

```
Cursor resultSet = mydatbase.rawQuery("Select * from TutorialsPoint",null);
resultSet.moveToFirst();
String username = resultSet.getString(0);
String password = resultSet.getString(1);
```

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

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

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

What is SQLiteOpenHelper?

It is a class found inside android.database.sqlite package. It is a helper class that helps in creating the database, handling the operations and also the version management.

To use sqliteopenhelper, we will create a class, and then we will extend SQLiteOpenHelper inside the class that we created.

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) {}
}
```

SQLiteOpenHelper class

The android.database.sqlite.SQLiteOpenHelper class is used for database creation and version management.

For performing any database operation, you have to provide the implementation of **onCreate()** and **onUpgrade()** methods of SQLiteOpenHelper class.

Constructors of SQLiteOpenHelper class

There are two constructors of SQLiteOpenHelper class.

Constructor	Description
SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version)	creates an object for creating, opening and managing the database.
SQLiteOpenHelper(Context context, String name, SQLiteDatabase.CursorFactory factory, int version, DatabaseErrorHandler errorHandler)	creates an object for creating, opening and managing the database. It specifies the error handler.

Methods of SQLiteOpenHelper class

There are many methods in SQLiteOpenHelper class. Some of them are as follows:

Method	Description
public abstract void onCreate(SQLiteDatabase db)	called only once when database is created for the first time.
public abstract void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)	called when database needs to be upgraded.
public synchronized void close ()	closes the database object.
public void onDowngrade(SQLiteDatabase db, int oldVersion, int newVersion)	called when database needs to be downgraded.

Using the Notification System

Android allows to put notification into the title bar of your application. The user can expand the notification bar and by selecting the notification the user can trigger another activity.

A notification is a message you can display to the user outside of your application's normal UI. When you tell the system to issue a notification, it first appears as an icon in the notification area.

To see the details of the notification, the user opens the notification drawer. Both the notification area and the notification drawer are system-controlled areas that the user can view at any time.

Elements of Notification

1. Content title
2. Large icon
3. Content text
4. Content info
5. Small icon
6. Time that the notification was issued.

Creating a Notification

We need to specify the UI information and actions for a notification in a NotificationCompat.Builder object.

To create the notification itself, you call NotificationCompat.Builder.build(), which returns a Notification object containing your specifications.

To issue the notification, you pass the Notification object to the system by calling **NotificationManager.notify().**

A Notification object must contain the following:

- A small icon, set by setSmallIcon()
- A title, set by setContentTitle()
- Detail text, set by setContentText()

The following code snippet illustrates a simple notification that specifies an activity to open when the user clicks the notification.
Notice that the code creates a TaskStackBuilder object and uses it to create the PendingIntent for the action

```
NotificationCompat.Builder mBuilder =
        new NotificationCompat.Builder(this)
        .setSmallIcon(R.drawable.notification_icon)
        .setContentTitle("My notification")
        .setContentText("Hello World!");

// Creates an explicit intent for an Activity in your app
Intent resultIntent = new Intent(this, ResultActivity.class);

// The stack builder object will contain an artificial back stack for the
// started Activity.
// This ensures that navigating backward from the Activity leads out of
// your application to the Home screen.

TaskStackBuilder stackBuilder = TaskStackBuilder.create(this);

// Adds the back stack for the Intent (but not the Intent itself)

stackBuilder.addParentStack(ResultActivity.class);

// Adds the Intent that starts the Activity to the top of the stack

stackBuilder.addNextIntent(resultIntent);
PendingIntent resultPendingIntent =
        stackBuilder.getPendingIntent(
            0,
            PendingIntent.FLAG_UPDATE_CURRENT
        );
mBuilder.setContentIntent(resultPendingIntent);
NotificationManager mNotificationManager =
    (NotificationManager) getSystemService(Context.NOTIFICATION_SERVICE);

// mId allows you to update the notification later on.

mNotificationManager.notify(mId, mBuilder.build());
```

Removing or Cancelling Notifications

Notifications remain visible until one of the following happens:

- The user dismisses the notification either individually or by using "Clear All" (if the notification can be cleared).
- The user clicks the notification, and you called setAutoCancel() when you created the notification.
- You call cancel() for a specific notification ID. This method also deletes ongoing notifications.
- You call cancelAll(), which removes all of the notifications you previously issued.

Working with Telephony Manager

- Provides access to information about the telephony services on the device. Applications can use the methods in this class to determine telephony services and states, as well as to access some types of subscriber information.

- Applications can also register a listener to receive notification of telephony state changes.

- The **android.telephony.TelephonyManager** class provides information about the telephony services such as subscriber id, sim serial number, phone network type etc. Moreover, you can determine the phone state etc.

You do not instantiate this class directly; instead, you retrieve a reference to an instance through **Context.getSystemService(Context.TELEPHONY_SERVICE).**

Permission Required:

To work with Telephony Manager and to read the phone details we need

<uses-permission android:name="android.permission.READ_PHONE_STATE"/> permission. So add this permission in your manifest file.

Accessing the Telephony Manager:

Have an object of TelephonyMnager

```
TelephonyManager tm=(TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
```

Get IMEI Number of Phone

```
String IMEINumber=tm.getDeviceId();
```

Get Subscriber ID

```
String subscriberID=tm.getDeviceId();
```

Get SIM Serial Number

```
String SIMSerialNumber=tm.getSimSerialNumber();
```

Get Network Country ISO Code

```
String networkCountryISO=tm.getNetworkCountryIso();
```

Get SIM Country ISO Code

```
String SIMCountryISO=tm.getSimCountryIso();
```

Get the device software version

```
String softwareVersion=tm.getDeviceSoftwareVersion()
```

Get the Voice mail number

```
String voiceMailNumber=tm.getVoiceMailNumber();
```

Get the Phone Type CDMA/GSM/NONE

/ /Get the type of network you are connected with

```
int phoneType=tm.getPhoneType();
switch (phoneType)
{
      case (TelephonyManager.PHONE_TYPE_CDMA):
            // your code
               break;
      case (TelephonyManager.PHONE_TYPE_GSM)
            // your code
               break;
      case (TelephonyManager.PHONE_TYPE_NONE):
            // your code
               break;
 }
```

Find whether the Phone is in Roaming, returns true if in roaming

```
boolean isRoaming=tm.isNetworkRoaming();
 if(isRoaming)
       phoneDetails+="\nIs In Roaming : "+"YES";
 else
       phoneDetails+="\nIs In Roaming : "+"NO";
```

Get the SIM state/Details

```
int SIMState=tm.getSimState();
switch(SIMState)
{
      case TelephonyManager.SIM_STATE_ABSENT :
        // your code
        break;
      case TelephonyManager.SIM_STATE_NETWORK_LOCKED :
        // your code
        break;
      case TelephonyManager.SIM_STATE_PIN_REQUIRED :
        // your code
        break;
      case TelephonyManager.SIM_STATE_PUK_REQUIRED :
        // your code
        break;
      case TelephonyManager.SIM_STATE_READY :
        // your code
        break;
      case TelephonyManager.SIM_STATE_UNKNOWN :
        // your code
        break;
        }
```

Drag one textview from the pallete, now the xml file will look like this.

File: activity_main.xml

```
<RelativeLayout xmlns:androclass="http://schemas.android.com/apk/res/android"
   xmlns:tools="http://schemas.android.com/tools"
   android:layout_width="match_parent"
   android:layout_height="match_parent"
   android:paddingBottom="@dimen/activity_vertical_margin"
   android:paddingLeft="@dimen/activity_horizontal_margin"
   android:paddingRight="@dimen/activity_horizontal_margin"
   android:paddingTop="@dimen/activity_vertical_margin"
   tools:context=".MainActivity" >
   <TextView
      android:id="@+id/textView1"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_alignParentLeft="true"
      android:layout_alignParentTop="true"
      android:layout_marginLeft="38dp"
      android:layout_marginTop="30dp"
      android:text="Phone Details:" />
 </RelativeLayout>
```

Now, write the code to display the information about the telephony services.

File: MainActivity.java

```
package com.javatpoint.telephonymanager;
import android.os.Bundle;
import android.app.Activity;
import android.content.Context;
import android.telephony.TelephonyManager;
import android.view.Menu;
import android.widget.TextView;
 public class MainActivity extends Activity {
   TextView textView1;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.activity_main);
```

```
 textView1=(TextView)findViewById(R.id.textView1);
        //Get the instance of TelephonyManager
       TelephonyManager  tm=(TelephonyManager)getSystemService(Context.TELEPHONY_SERVICE);
        //Calling the methods of TelephonyManager the returns the information
       String IMEINumber=tm.getDeviceId();
       String subscriberID=tm.getDeviceId();
       String SIMSerialNumber=tm.getSimSerialNumber();
       String softwareVersion=tm.getDeviceSoftwareVersion();
       String voiceMailNumber=tm.getVoiceMailNumber();
             //Get the phone type
       String strphoneType="";
        int phoneType=tm.getPhoneType();
        switch (phoneType)
       {
             case (TelephonyManager.PHONE_TYPE_CDMA):
                    strphoneType="CDMA";
                       break;
             case (TelephonyManager.PHONE_TYPE_GSM):
                    strphoneType="GSM";
                       break;
             case (TelephonyManager.PHONE_TYPE_NONE):
                     strphoneType="NONE";
                        break;
        }
          //getting information if phone is in roaming
       boolean isRoaming=tm.isNetworkRoaming();
       String info="Phone Details:\n";
       info+="\n IMEI Number:"+IMEINumber;
       info+="\n SubscriberID:"+subscriberID;
       info+="\n Sim Serial Number:"+SIMSerialNumber;
       info+="\n Software Version:"+softwareVersion;
       info+="\n Voice Mail Number:"+voiceMailNumber;
       info+="\n Phone Network Type:"+strphoneType;
       info+="\n In Roaming? :"+isRoaming;
 textView1.setText(info);//displaying the information in the textView
    }
}
```

You need to provide **READ_PHONE_STATE** permission in the AndroidManifest.xml file.

File: AndroidManifest.xml

```
<manifest xmlns:androclass="http://schemas.android.com/apk/res/android"
   package="com.javatpoint.telephonymanager"
   android:versionCode="1"
   android:versionName="1.0" >
    <uses-sdk
      android:minSdkVersion="8"
      android:targetSdkVersion="17" />
   <uses-permission android:name="android.permission.READ_PHONE_STATE"/>
   <application
      android:allowBackup="true"
      android:icon="@drawable/ic_launcher"
      android:label="@string/app_name"
      android:theme="@style/AppTheme" >
      <activity
         android:name="com.javatpoint.telephonymanager.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>   </application>   </manifest>
```